RA

stichting
mathematisch
centrum

MC

REKENAFDELING RA MR 132/72 APRIL

L. GEURTS, L. MEERTENS, G. NOGAREDE,
M. REM and R.P. VAN DE RIET

# THE MC ELAN ASSEMBLER

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat 49, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

Table of contents

## 0. Introduction

The original ELAN assembler, written in ELAN, had, at the time the first thoughts about the MC ELAN assembler came up, a numer of inconveniences , in particular its documentation and its reaction upon errors were far from ideal. Furthermore, it was felt desirable to have an output parity checked punched tape and an alphabetic listing of all identifiers used and their occurrences in the text. The assembler should accept any text and be as much error-recoverable as possible. Therefore, a new assembler was planned to be written in ALGOL 60. The result is described in this report. The macro feature designed and implemented by M. Rem, has been incorporated also, but is described in [4].

The first chapter contains a general outline of the assembly process; the second chapter deals with the treatment of the syntactic structure; the third chapter describes the reading equipment, in particular, with respect to error recovery; the fourth charpter gives the necessary information about the name list technique; in the fifth chapter the output part of the assembler is considered, in particular, with respect to the parity checked code for which a special loader has been written in ELAN; the output to be printed is finally discussed in chapter 6.
In two appendices, A and B, the ALGOL text of the assembler, together with a listing of all the occurring identifiers (A), and the ELAN text of the loader are reproduced (B). The latter one as being output by the assembler, together with a listing of the identifiers used. Moreover, a test program for the loader is reproduced.
The programs in the appendices have been amply supplied with comments; they form the nucleus of this report. The chapters are meant to facilitate the reading of the programs only.
References to sections in appendix A are by means of the prefix "A".

Several people have been cooperating in the construction of the MC ELAN assembler.
The general outline and a first working program were made by R.P. van de Riet.

Error recovery, use of secondary storage and a neat lay-out of the printed output have been programmed by L. Meertens and L. Geurts.
The name-list technique and the alphabetic listing have been treated by G. Nogarede.
The final version, in which a number of redundances and inconsistencies have been removed, has been prepared by M. Rem, who also wrote chapters 4 and 5.
Finally, the loader and a test program have been programmed by R.P. van de Riet.

The initial plans are due to F.E.J. Kruseman Aretz, B.J. Mailloux, K.K. Koksma, P.J. van der Laarschot and E.G.M. Broerse.

Considerable help has been given by J. van Vaalen, G. ten Velden, J. Wolleswinkel and C. Zuidema.

## 1. The general outline

The assembly proces consists of the following phases to be executed in the indicated ordering:

### (i) Initialization

During this phase the tables are filled; i.e. the arrays:

op op reg, d26 op op reg,
reg op op, d26 reg op op,
fctn op, d26 fctn op

which contain all the information necessary to produce the code for all different instructions. Not only the operation code, but also the allowed variants, allowed addressing modes and the possible minus modification are stored in these words.
The tables are filled, not by reading the bits directly from input tape, but by reading a special form of an ELAN program, the "initial program", in which the bits are coded.
This program consists mainly of lines of the following kind:

```
" 1,UYN, A + x     , PZE;       " 000 000 xx xx xx xxxxx xxxxx xxxxx
"40, YN, MULAS(-x), PZE; :DYN;" 010 001 xx xx xx xxxxx xxxxx xxxxx
```

with the meaning:

a) In the EL manual for ELAN programmers [2], the meaning of these instructions can be found under 1 and 40, respectively.
b) The U, Y or N variants can all be used with the instruction A + x. Only the Y and N variants can be used in the instruction MULAS(-x).
c) All possible operands are allowed in A + x.
   All possible operands, except the ":DYN operand", are allowed in MULAS(-x).
d) The minus-modification is allowed in MULAS.
e) All variants, P, Z or E are allowed in both A + x and MULAS(x).
f) The bit patterns are indicated: the bits denoted by "0" or "1" are the definitive bits, the bits denoted by "x" are calculated by the assembler.

It is the syntax-reading equipment, which determines the array element,belonging to the instruction A + x or to MULAS(-x), which has to be filled during the initalization.

(ii) The first scan

During this phase the program is scanned for the first time and the identifiers in the declaration parts are picked up and their values are determined from their defining places as labels in the marginal part of the program. Also the locations where the instruction counter is changed are read carefully.
Most of the instructions (except the SKIP instruction of course) are skipped until the statement separator.

(iii) The second scan

During this phase the program is scanned for a second time and the code is produced by means of the array element designated by the syntax-reading equipment (RE ELAN instruction).

The assembler consists functionally of the three parts described above; it cannot be split materially in the three indicated parts, however. There is one syntax-reading apparatus which, as far as the reading is concerned is repeated during the two scans.
Advantages over separately programmed scans are:

a) The assembler recognizes the text during the two scans in the same way.
b) Pieces of assembler program which belong logically together, which refer to the same syntactical structure but which differ in the two scans, are written closely together. This is an important factor for the readability and hence for the correctness of the assembler.
c) That the same syntax-reading apparatus is used during the initialization as well as during the second scan to find the array element to be filled or to be inspected is the guarantee for the correct code to be produced, provided the initial ELAN program contains the correct information.

The form in which this information is given, however, is optimally close to the form this information is given in the reference manual. (Although, in practice, it turned out after months of use that still two bits were interchanged.)

The form in which the program is written is "top to bottom". The program starts with the procedure "RE ELAN block" and it ends with the declaration and initialization of the variables used.
The variables and procedures have identifiers with a meaning suggestive enough to compensate for the lack of more comment. A general rule is that identifiers of the form "RE <synt structure>" are the identifiers of procedures which read the syntactic structure indicated.
The frequently occurring procedure call: "RE" reads one syntactic unit and delivers the type of the syntactic unit in the variable "synt unit".
Some types are: "begin symbol", "identifier", "special identifier", "number" and "colon symbol".
In general, syntactic units are directly composed of the lowest syntactic structure: the symbols (or characters). Sometimes they consist of only one symbol (as "colon symbol"), sometimes of more (as "begin symbol" or "identifier").
As soon as a "number" or an "identifier" has been read by "RE", all relevant information is available as e.g. its value or whether it is a new identifier, or the type of the identifier, or the place in the name-list: "contents of" where more information can be found or should be stored.

The reading process reads always one symbol further than the final symbol of the syntactic unit read.
This is a strategy which is sometimes necessary (numbers, identifiers) and sometimes not necessary (begin symbol, colon symbol). A rigorous strategy, in reading one symbol ahead always, leads, however, to the most simple algorithm. This has some consequence on the print process, which prints the last but one symbol read; moreover, new lines are treated carefully, since new lines are interpreted as statement sepa-

rators.

For historical reasons the assembler does not distinguish between small and capital letters; neither are the symbols "(" and "[" and the symbols ")" and "]" distinguished.
In the printer output of the ELAN text, however, the original text appears.

## 2. The syntactic structure of ELAN

The syntax of ELAN will now be reproduced from the ALGOL 60 program, as given in Appendix A. The numbers refer to sections in this appendix.
The peculiar symbol ".," stands for a simple semicolon ";", which could not be used in the comment part of the ALGOL 60 program.
The syntactic structure <nlcr> means a "new-line-carriage-return" symbol.
The meaning of LCA|...|RUSA|LVIFA|...|IFSC|LVIFON|...|AFOFF is the full list of all shift instructions and flipflop instructions.
The meaning of PLUSA|...|SUBCD is the full list of all other functional instructions, which behave normally.

<ELAN program>::= <ELAN block> | <location>: <ELAN block>

<ELAN block>::= <block begin> <compound tail>
  <block begin>::= 'BEGIN' <ELAN declaration>
  <statement separator>::= <nlcr> |; | <comment> <nlcr> | <comment> ;
  <comment>::= "<one line of symbols not containing <nlcr> or ;>
  <compound tail>::= <ELAN statement> <statement separator>
        <compound tail> | 'END'
<ELAN declaration>::= <ord decl and MT decl> <statement separator> |
        <MT declaration> <statement separator>
  <ord decl and MT decl>::= <list of initializations> |
        <list of initializations> <comma> <MT declaration>
  <MT declaration>::= 'MT' <list of identifiers>
  <list of identifiers>::= <identifier> |
        <identifier> <comma> <list of identifiers>
  <comma>::= , | <comma> <statement separator>
  <list of initializations>::= <intialization> |
        <list of initializations> <comma> <intialization>
  <initialization>::= <identifier> | <identifier> = <operand>

<ELAN statement>::= <marginal part> <ELAN instruction>
<marginal part>::= <empty> | <label sequence> | <location and label>
  <label sequence>::= <label> | <label sequence> <label>
  <label>::= <identifier>:

```
<location and labels>::= <location>: | <location>: <label sequence>
<location>::= <known location>
<known location>::= M[<expression>] | <identifier>[<expression>]


<ELAN instruction>::= <empty> | <STAT address operand> | <unsigned real> |
          <adding operator> <unsigned number> | <BI or IP instruction> |
          <SKIP instruction> | <ELAN block> <letgit string option> |
          <UYN part> <funtional instruction> <PZE part> |
          <UYN part> <arithmetic instruction> <PZE part>
<BI or IP instruction>::= 'BI' | 'IP'
<SKIP instruction>::= 'SKIP' <expression>
<letgit string option>::= <letter> <letgit string option> |
        <digit> <letgit string option> | <empty>
<UYN part>::= <empty> | U, | Y, | N,
<PZE part>::= <empty> |,P  |,Z  |,E
<functional instruction>::= <special identifier> | <shift instruction> |
        <special identifier>(<right operand>)
<shift instruction>::= <shift identifier>(<shift expr>)
<shift identifier>::= LCA | ... | RUSA | LVIFA | ... | IFSC |
        LVIFON | ... | AFOFF
<shift expr>::= <expression> | B | B + <unsigned expression>
<arithmetic instruction>::= <register> <operator> <operand> |
        <left operand> <operator> <register>
<register>::= A | S | B | F | G
<operator>::= = | =- | <adding operator> | <multiplying operator>
<adding operator>::= + | - | '+' | '-'
<multiplying operator>::= x | / | 'x' | 'x'-
<unsigned number>::= <unsigned integer> | <unsigned real>
<unsigned integer>::= <unsigned decimal> | '<unsigned octal>'
```

```
<left operand>::= <STAT operand> | <STATB operand> | <DYN operand>
<operand>::= <left operand> | <address operand>
  <address operand>::= <STAT address operand> |
          <DYN address operand>
  <STAT operand>::= M | M[<expression>] | <identifier> | T |
          <identifier> <expression>
  <STATB operand>::= M[B<Bexpr>] | <identifier> [B<Bexpr>]
  <Bexpr>::= <empty> | <adding operator> <unsigned expression>
  <DYN operand>::= <DYN M symbol> |
          <DYN M symbol>[<q expr>]
  <DYN M symbol>::= MG | MA | MS | MC | MT | MD | M <p expr>
  <q expr>::= <expression>
  <p expr>::= <digit> | <digit> <digit>
  <STAT address operand>::= : <STAT operand> | <unsigned integer> |
          (<expression>) | : <register>
  <DYN address operand>::= : <DYN operand>

<expression>::= <unsigned expression> |
          <adding operator> <unsigned expression>
  <unsigned expression>::= <term> |
          <unsigned expression> <adding operator> <term>
  <term>::= <primary> | <term> <multiplying operator> <primary>
  <primary>::= <STAT address operand>
```

## 3. Reading equipment and Error recovery

All the high-level syntax reading procedures use the procedure "RE" only (in a few places "RE through barrier" is used, this will be discussed later on), and the result is a new syntactic unit.
From a syntactic unit to the lowest-level heptad on a punched tape, the following hierarchy of procedures is used (see section A8):

"RE" calls "READ synt unit" and does some administration concerning the number of begins and ends; moreover it refuses to call "READ synt unit" if "stat sep barrier" ≡ true and the old syntactic unit was a statement separator or an end symbol. The reasons will be discussed in section 3.1.

"READ synt unit" reads the syntactic unit using "NS". In principle, the first symbol of the syntactic unit has been read already; this is not the case if a new line occurred, in which case ("NS deferred" ≡ true) the first symbol is read explicitly.
An important aid to this procedure is the procedure "STORE letgits with" for storing and searching a name in the name-list (see chapter 4).

"NS" reads one symbol, by means of "RE NS".
As the assembler uses a look-ahead-technique, by means of "LOOK AHEAD sub text bus", the actual reading pointer "reading ptr" may be ahead of the reading pointer "ptr of text" which points to the first not yet syntactically treated symbol. It is the duty of "NS" to print the old symbol read.

"RE NS" either reads a symbol with "RESYM1", and stores it by means of the array "ELAN line" and the procedure "store into buffer" on secondary storage, or takes it from secondary storage by means of the procedure "fetch from drum".

"RESYM1" either gets the next symbol from the string consisting of the initial ELAN program during the initialization, or it uses "RESYM".

"RESYM" builds the symbol from a heptad written by "REHEP", the standard MC procedure.

## 3.1. The handling of syntactical errors

In a prior version of the assembler, the handling of errors involved nothing beyond their mere detection. In an experimental assembler this may be overlooked, but in an assembler claiming practical usability, this is, of course, unacceptable. Take, e.g., the following piece of text:

```
        A = MC-1], P
    Y, DO(TSP)
    N, DO(TSN)
```

This would be correct ELAN, but for the omission of one square bracket "[". The syntactical structure, in case of an error, is determined by taking the alternative which happens to be the last. Thus, instead of taking the obvious interpretation, consisting of three statements, one of which is erroneous, this results in an interpretation:

```
A + MC
1
, P;
, DO(
); N, DO
(TSN);
```

which makes little sense, and - which is worse - has a far more than local influence by disarranging the address counter. It is clearly desirable that there be a synchronization between statements as recognized by the assembler, and the pieces of source text delimited by statement separators. This synchronization has been achieved as follows:

a) one type of discrepancy arises when the assembler has recognized a complete ELAN statement, and the next syntactical unit is no statement separator. In this case, the procedure "RE ELAN statement" will skip the remaining text until a statement separator is encountered. For reasons of security, an end symbol will serve too to end this quest, although, according to the present syntax, it may not be used to terminate a statement. At the end of the declarations of a block a similar synchronization takes place, which is, however, achieved in a different way, explained later on.

b) another source of difficulties is the possibility that the assembler has not yet recognized a complete statement when the syntactical unit at hand is already a statement separator or an end symbol. As the points where this situation may arise lay sprinkled through the assembler-program, some special device is needed to prevent getting out of step: consider the syntactical units to be queued up before a frontier. At each time, the syntactical unit at hand is the one at the head of the queue. The "acceptance" of a syntactical unit by the assembler is the transgression of the frontier by one syntactical unit. At the frontier there is a barrier. When this barrier is up, each syntactical unit is allowed to pass, but when it is down, the passage is barred to statement separators and end symbols. Now the essence of the solution lies in the following: normally, this barrier is down, so no statement separators and end symbols will be accepted; only on some very specific occasions, where the assembler is aware of the significance of the syntactical unit at hand being a statement separator or end symbol, the barrier will be raised for one symbol, thus allowing acceptance. This device guarantees, with absolute security, that no statement separator or end symbol will ever be accepted without due attention. As to the implementation, the state of the barrier is reflected by the global Boolean variable "stat sep barrier". Its inhibitive effect is seen in the very beginning of the procedure "RE", where the old syntactical unit will be retained (i.e., not accepted) when it is a statement separator or end symbol but the barrier is down.
In order to facilitate the use of the device, the procedure "RE through barrier" has been introduced, which will accept one syntactical unit unconditionally, and which is the one and only instance which ever raises the barrier, to lower it again as soon as possible. The following list of occasions where "RE through barrier" is called, is exhaustive:

(i) in the procedure "START block", when searching for the first begin symbol of the source text, in order to skip meaningless new line characters. This use provides an initialization for "stat sep barrier" at the same time.

(ij) in the procedure "RE ELAN block", to accept (the) statement separator(s) between the declarations and the first statement, or between statements, or between the last statement of a block and its end symbol.

(iij) in the procedure "RE ELAN block" to accept the symbol terminating a block.

(iv) in the procedure "RE poss decl id" when called with parameter true and when the syntactical unit at hand is a statement separator, in order to skip it. This case occurs when, analyzing the declarations, a symbol "," is encountered (see the syntax of <comma>).

Besides this major synchronizing apparatus, a number of other aids to synchronization on a lower level have been added. On various occasions, after some syntactical construction has been recognized, it is known from the syntax what the following syntactical unit should be (as was the case with <ELAN statement> <statement separator>). When this unit happens to be not present, sometimes a hunt will be started, taking care, however, not to get stuck on a statement separator or end symbol. This is done, by means of the procedure "REQUIRE", to find the matching "]" or ")" to an otherwise unmatched "[" or "(" and to find the terminating "," to a declaration. The latter case serves at the same time to synchronize the declarations with the statement separators, as the hunt for the ",", when it is not present, will be terminated by a statement separator (or end symbol). It may be questioned to what extent this synchronizing device is useful, as experience suggests that in about half of the cases the assembler will not find the missing unit (except in the case of the declarations). A not insignificant advantage, however, is the prevention of a stream of meaningless error messages.

In a large number of other cases where the correct syntactical unit is known but not present, it will, as it were, be inserted into the sequence of syntactical units. This is accomplished by the procedure "CHECK" and governed by the global Boolean variable "CHECK fault": the assembler will take the course as though the correct syntactical unit were present, and the actual syntactical unit will not be accepted, but retained till the next occasion. A discussion on the merits of this construction is rather

precarious, but experience has not brought to light any undersirable effects.

## 3.2. The error list

The error message, evoked by the procedures "ERROR", "CHECK" and "REQUIRE" of section A13, consists of:

"error nr:" a number and the representation of the syntactic unit treated, and in the first scan the line number.

Sometimes the error message is followed by: "synt unit should be" followed by the representation of the syntactic unit required.

An exhaustive list of error numbers follows:

210 block does not start with 'BEGIN';

300 regular declaration does not start with an identifier or 'MT';

301 regular declaration of an identifier that already occurred in this block;

302 in the list of regular declarations a comma is not followed by an identifier;

305 non-initialized identifier in the list of regular declarations does not occur as a label anywhere;

310 regular declaration not followed by a comma or a statement separator;

312 in MT-declaration 'MT' is not followed by a comma or an identifier;

313 MT-declaration of an identifier that already occurred in this block;

315 MT-declared identifier does not occur as a label anywhere;

320 MT-declaration not followed by a comma or a statement separator;

400 label identifier not declared;

410 label identifier has been initialized or has already occurred as a label;

415 location identifier not declared;

420 location identifier not of type STAT;

430 location expression not followed by ];

500 statement not followed by a statement separator or 'END';
510 plus or minus at the beginning of an instruction not followed by digit, point or lower-ten;
520 U, Y or N not followed by a comma;
530 after an instruction, comma is not followed by P,Z or E;
540 register at the beginning of an arithmetic instruction not followed by an operator;
545 right operand of an arithmetic instruction (type :STAT) is negative;
550 left operand of an arithmetic instruction is of type :STAT or :DYN;
560 left operand of an arithmetic instruction not followed by an operator;
570 no F, G, A, S or B register where required;
580 expression of shift instruction is negative or greater than 31
(for instructions LVIFA, ..., IFSC : greater than 1,
for instructions LVIFON, ..., AFOFF: greater than 39);
585 operand of functional instruction (type :STAT) is negative;
590 operand of functional instruction not followed by );
595 shift identifier not followed by (;
600 colon at the beginning of an operand not followed by STAT or DYN operand;
610 real number in operand or expression;
620 no ) in expression that begins with (;
630 no ] after q-expression of DYN or :DYN operand;
635 p-expression of DYN or :DYN operand is negative or greater than 63;
640 q-expression of DYN or :DYN operand smaller than -256 or greater than 255;
650 Unknown identifier in operand or expression;
655 non-STAT identifier followed by [B], [B+ or [B-;
670 [ in STAT, STATB or :STAT operand not followed by the accessory ];
680 implicit q-expression of MT-declared operand smaller than -256 or greater than 255;
690 operand begins with inadmissible character;

700 operand in expression is not of type :STAT;

800 administration space of the assembler exhausted;
810 apostrophe not followed by B, E, I, M or S;
811 'B not followed by E or I;
815 last apostrophe missing in 'BEGIN', 'END', 'SKIP', 'MT', 'BI' or 'IP';
820 last apostrophe missing in '+' or '×';
830 8 or 9 in octal number;
840 last apostrophe missing in octal number;
860 apostrophe followed by inadmissible character;
870 . or $_{10}$ or $_{10}^{+}$ or $_{10}^{-}$ not followed by a digit;
871 input tape has more than 7 tracks;
872 unknown punching;
873 more than 200 characters on one line between [ and the first non-layout character after ] (note: from [ on the text has not been printed yet);
874 extra 'END's added;
875 program too long;
876 [ not followed by the accessory ];
880 administration space of the assembler exhausted;
881 administration space for the calls relatively exhausted;
882 adminsitration space for the calls absolutely exhausted;

920 initialization error: no number where required;
930 " " : no comma where required;
940 " " : no 'END' where required;
950 " " : no statement separator where required;
960 " " : no quote where required;
970/980 " " : more than 27 bits specified in one word;
985 " " : operand not of type STAT, :STAT, STATB, DYN or :DYN;
990 " " : instruction is not an op op reg instruction, reg op op instruction or function instruction;

1000 operand of an instruction not allowed (or: operator in this combination of operands not allowed);

1010 U, Y or N where not allowed;

1020 P, Z or E where not allowed;

1030 negative operator or operand in instruction where not allowed;

1040 operand negative or greater than 32767
(if STATB then smaller than -16384 or greater than 16383);

1045 integer expression greater than integer capacity;

1060 octal too great for printing;

1065 octal has too much digits for printing;

2000 name-list full;

2010 instruction counter points out of the memory
(see number on the preceding line);

2020 inadmissible syntactic unit has been interpreted as plus
(this error message is always preceded by error 540 or 560).

# 4. Name list Organization

A few remarks have to be made before we can enter into the details of the name list organization.

## 4.1. Preliminary remarks

### 4.1.1. Block structure

ELAN has - like ALGOL 60 - a block structure. That implies that one name can be used for different identifiers, if they are declared in different blocks.
The assembler gives every block a block number, and in the array "block" we have at each moment a survey of the valid block numbers.
Note that names not declared are assumed to be declared in the outermost block; a suitable warning is given in the form of error message 400.

### 4.1.2. Extra ends

If the input is not sufficient, that means if there are more 'begins' than 'ends', then the assembler will generate as many ends as are required, at the same time giving the error message 874. This makes sure that the assembler will come into its second scan, in which it produces the listing of the ELAN program.

### 4.1.3. Identifiers

We distinguish two kinds of identifiers, the special identifiers, and the identifiers declared by the program, which we will simply call "identifiers".
Identifiers can be defined (= can be given a value) in two ways:
1. By giving the identifier an explicit value in the declaration.
2. By using the identifier as a label in the margin.
The ELAN programmer is not allowed to redeclare the special identifiers.

## 4.2. The data structure chosen

The information about the identifiers and the special identifiers is stored as a binary tree in the array "contents of". Besides this array we

use an array "LINE" to register the lines of occurence of every identifier. Every name in the name list has two pointers, one to a name that preceeds it in alphabetical order, and one to a name that succeeds it. As there may be several identifiers with the same name, we use an administration section for each identifier. The administration sections that belong to one name form a linked list with the name administration as list head. The order in this list is from high to lower block numbers. We can thus get the appropriate administration section by going through the list until we find a section with a valid block number.

name administration:

1) number of memory words used for the storage of the letters and digits of the name
2) the name
3) pointer to a preceding name
4) pointer to a succeeding name
5) pointer to the first administration section

administration section:

6) block number
7) type
8) value
9) line of declaration
10) pointer to the place in the array "LINE" where the last occurrence of this identifier is recorded
11) pointer to the next administration section

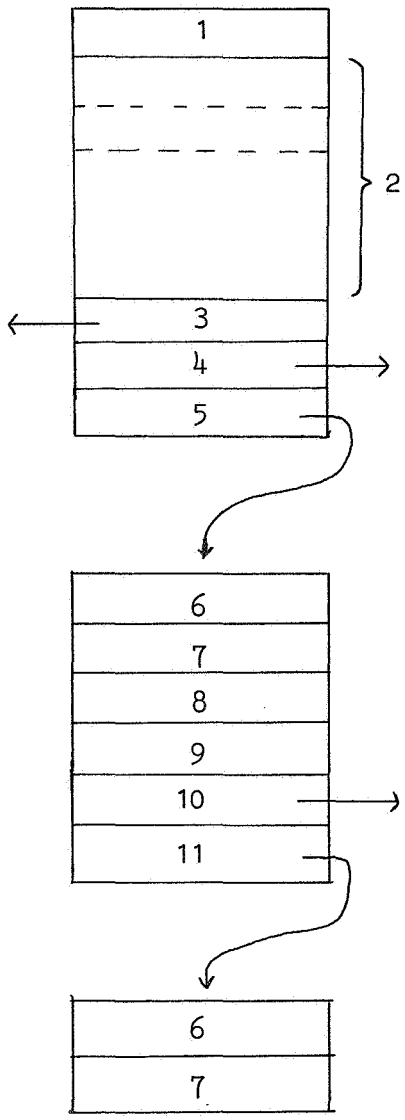
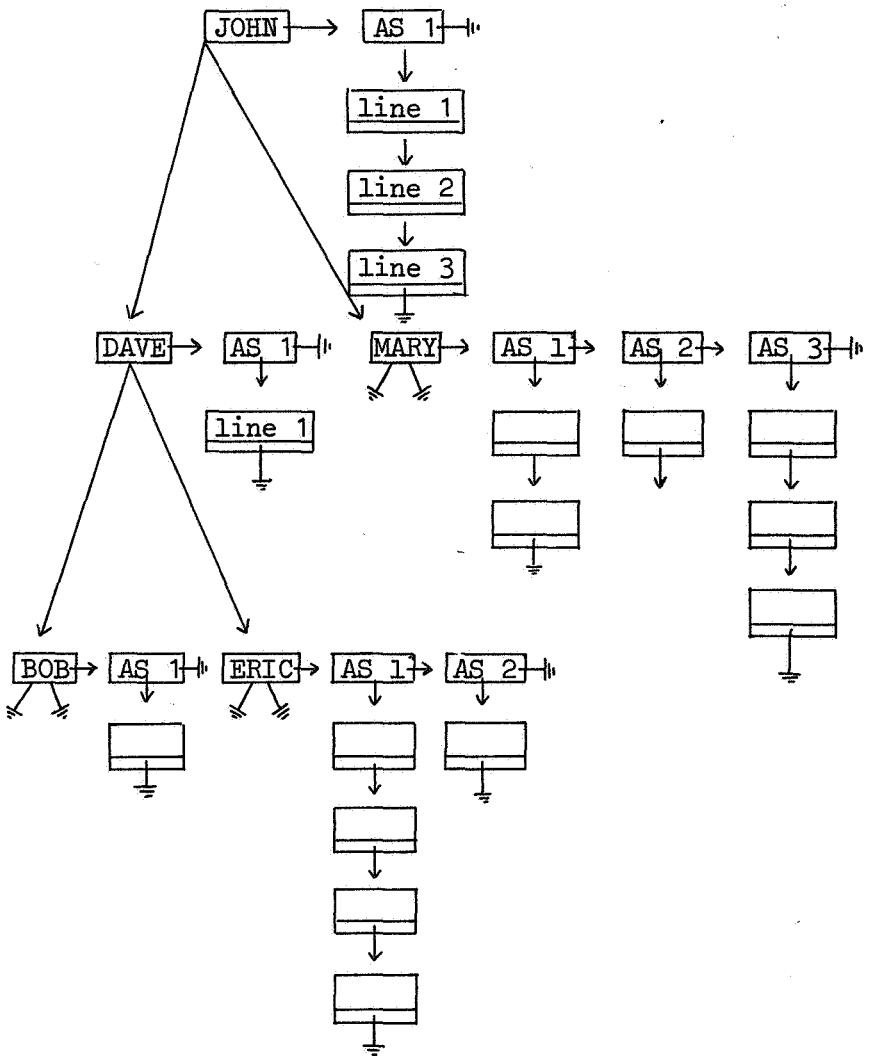



fig. 1. The administration in "contents of".

In this way we get a binary tree where not only every node is a list head of the linked list of administration sections, but where also every administration section is used as a list head for the linked list of the occurrences of this identifier, as recorded in the array "LINE".




AS i = i-th administration section
line i= a line number of the line where the identifier occurred
⏚ = empty pointer.

fig. 2. Example of storage in "contents of" and "LINE".

It is, in order to get an acceptable sorting speed, important that the tree does not grow too fast in one direction. Therefore, we start with the insertion of the 132 special identifiers. By presenting them in a special sequence we accomplish that the tree has an optimal flat shape to begin with.

During the first scan the ELAN text is read, the administration in "contents of" is updated and the text is transported to the drum. During the second scan the text is fetched back from the drum, and the space on the drum that comes free may be used to store the administration of "LINE".

A discussion of the procedures "STORE in contents of" and "store in LINE", which together with "st" and "ST", create the administration sections and fill them, may be found in [1].

## 5. The code produced

In section A10 the information about the instructions is read from an initial string and, after a suitable transformation, stored in the tables. The entries in the tables are: the type of the operand (STAT operand, STAT address operand, STATB operand, DYN operand, DYN address operand), the operator (+, ×, /, =, '+', '×'), the register (A, B, S, G, F) and the function (PLUSA, ..., MEMPROT).
At the very beginning all array elements are set equal to -1. Only the information of the input string can change the array elements. Hence, instructions which are forbidden correspond to -1. Inspection of the bits of all the instructions shows that bits $d_4$ $d_3$ $d_2$ $d_1$ are either filled with an "x" or with an "0". They may be used, therefore, to hold the information about variants and a possible minus:

| $d_4$ $d_3$ | U allowed | Y allowed | N allowed |
|---|---|---|---|
| 0 0 | no | no | no |
| 0 1 | no | yes | yes |
| 1 0 | yes | yes | yes |

| $d_2$ | |
|---|---|
| 0 | neither P, nor Z nor E allowed |
| 1 | P, Z or E allowed |

| $d_1$ | |
|---|---|
| 0 | minus not allowed |
| 1 | minus allowed |

The code producing procedures of section A11 consist mainly of "PRODUCE INSTR CODE", which, by means of the values of the variables "INF WORD" and "d26 INF WORD" assigned in the syntax reading equipment (A5) to the correct array elements of the tables, produces the code through the procedure "PR binary number". The latter procedure prints and punches the code.

The punched tape consists of a number n, n > 0, of blocks. Each block consists of a number m, m > 0, of words.
Before, between and after the blocks the tape may consist of blank heptads. Either, the words are IP words and then the tape can be read directly by the X8, or the words are BI words and then the binary loader, reproduced in appendix B should be in the X8 to read the words.
Each IP word consists of 27 bits: $d_{26}\ d_{25} \cdots d_1\ d_0$. They are preceded by one bit 1, and then divided into 4 parts:

$$p_1 = 1\ \ d_{26}\ d_{25}\ d_{23} \cdot d_{22}\ d_{21}\ d_{20},$$

$$\cdot \quad \cdot \quad \cdot$$

$$p_4 = d_6\ d_5\ d_4\ d_3 \cdot d_2\ d_1\ d_0$$

in which order they occur as heptads on punched tape (a 1 corresponds with a hole).
Each BI word consists of 27 bits: $d_{26}\ d_{25} \cdots d_1\ d_0$. They are preceded by three bits: P $C_1$ $C_2$ and then divided into 5 parts:

$$p_1 = 0\ P\ \ C_1\ \ C_2\ \cdot d_{26}\ d_{25}\ d_{24}$$

$$p_2 = 0\ d_{23}\ d_{22}\ d_{21} \cdot d_{20}\ d_{19}\ d_{18}$$

$$\cdot \quad \cdot \quad \cdot$$

$$p_5 = 0\ \ d_5\ \ d_4\ \ d_3\ \cdot\ d_2\ \ d_1\ \ d_0$$

where P is such that the number of ones, (the number of holes) is odd; for the first word of a block $C_1 = C_2 = 1$; for the other words $C_1 = C_2 = 0$. In principle, the p's occur in this order on the punched tape; they may, however, be separated by any piece of tape beginning with erase (1111.111) and ending with erase. In this piece of tape an end-of-tape or a begin-of-tape characterization, according to the MICRO- and the MCALL-system, may occur.

The 27 bits of the first word of a block can be divided into

$$T = d_{26}\ d_{25} \cdots d_{19}\ d_{18}$$

and $$A = d_{17}\ d_{16} \cdots d_1\ d_0.$$

If T = A = 0, then the block is the last block of the tape. Otherwise,

the block consists of another T words $w_1$, ..., $w_T$, which have to be stored in memory locations M[A+1], ..., M[A+T]; if T = 0 and A $\neq$ 0, T is considered to be 512.

For binary tapes, a loader has been written which is self-replaceable and self-destructive.
Initially, the loader is placed in the higher addresses, it checks its checksum and it stores -0's everywhere except in its own area and in the places M[i], $0 \leq i \leq 512$.
As soon as a word from the input binary tape has to be written on a place of the loader itself, the loader moves downwards until it finds a traject consisting of -0's which is long enough. It, furthermore, fills the old place with -0's.
Two little programs situated in the "holy places", M[i], $0 \leq i < 24$, perform the actions: "clear" for filling a traject with -0's and forcing an interrupt from the IP reader in behalf of the programmer of the binary tape who may have filled M[24], and "move" for removing the loader to a new place.
The program itself is amply supplied with comment.
A test program for the loader and for the assembler has been written and is also reproduced in appendix B.
The declarations of the latter program have been designed mainly to test the assembler.

Errors during loading of binary tapes

During loading of binary tapes the following errors are detected, upon which the X8 comes into the dynamic stop situation. The A register determines which error occurred.

A=0: sumcheck of loader not OK;
A=1: tapereader nok;
A=2: $d_6$ is punched but heptad is not erase;
A=3: number of holes in binary word even;
A=4: the first word of a block does not begin with $C_1 = C_2 = 1$;

A=5: a word of a block, not the first one, does not begin with $C_1 = C_2 = 0$;

A=6: one of the following addresses a will erroneously be filled: $0 \leq a \leq 23$, $57 \leq a \leq 62$;

A=7: there is no space left for the loader.

## 6. Printer output

Besides the paper tape output there are three kinds of output that come over the high-speed printer:

1) Error messages, they can occur during the first scan, as well as during the second.
2) The ELAN-text, together with the assembled code.
3) The name list with lines of occurrence of every identifier.

sub 2:
Every line is built up in the array "line buffer". The first 8 positions of the line contain the line number, the next 11 positions the store address, and the positions 20 up to 32 the assembled code. The rest of the line contains the relevant ELAN line. The whole line is printed at once. If there are more than 144 positions in one line, we continue with a new line. Four or more nlcr's have as effect a new page.
Blanks preceeding an nlcr are skipped. (This is especially useful when the

ELAN text is punched on cards.)

sub 3:
For every identifier the following is successively printed:
1) the name (each name only once)
2) type and value:

| type: | meaning: | notation: |
|---|---|---|
| 1 | STAT operand | M['......'] |
| 2 | STATB operand | M[B+'......'] |
| 3 | DYN operand | Mp[q] |
| 4 | STAT ad operand | '......' |
| 5 | :DYN operand | :Mp[q] |
| 6 | unknown type | ??? |
| 10 | declared | ? |
| 11 | MT declared | 'MT'? |
| 12 | MT declared and defined | ('MT')M['......'] |

where '......' denotes an octal and p or q a decimal number.

Note that types 6, 10 and 11 indicate erroneous programs.

3) block number.

4) line of declaration.

5) lines of occurrence, where a plus denotes the line of definition.

References

[1] R.P. van de Riet, G. Nogarede, A note on automatic storage of arbitrary trees in ALGOL 60.
Report NR 16/71 Mathematical Centre, May 1971 Amsterdam.

[2] Programmering EL X8, NV Philips-Electrologica, Den Haag.

[3] F.E.J. Kruseman Aretz, J.B. Mailloux, K.K. Koksma, E.G.M. Broerse.
Mathematisch Centrum, Internal note.

[4] M. Rem, De MC-ELAN Macroprocessor, Mathematisch Centrum,
rapport NR 21/71.

Appendix A

The ALGOL 60 text of the assembler is now reproduced followed by an alphabetic listing of all identifiers used and the line numbers of their defining and applied occurences in the text.
For this listing a program of L. Meertens has been used.

```
begin comment MC ELAN1 assembler and definition, 010471. R 2172s.

  1. ELAN program.

  <ELAN program>::= <ELAN block> | <location>: <ELAN block>

  2. ELAN block.

  <ELAN block>::= <block begin> <compound tail>
  <block begin>::= 'begin' <ELAN declaration>
  <statement separator>::= <nlcr> | .; | <comment> <nlcr> |
                           <comment> .;
  <comment>::= "<one line of symbols not containing <nlcr> or .;>
  <compound tail>::= 'end' |
          <ELAN statement> <statement separator> <compound tail>

  Algorithm:;

  procedure RE ELAN block;
  begin block number:= max block number:= max block number + 1;
    cntr of begins:= cntr of begins + 1;
    block[cntr of begins]:= line buffer[-1]:= block number;
    RE block begin;
  st sep: if synt unit = statement separator then
    begin RE through barrier; goto st sep end;
    if synt unit ≠ end symbol then
    begin RE ELAN statement; goto st sep end;
    if block number > 2 then
      begin if line buffer[-1] > 2 then PRINT LINE;
        line buffer[-1]:= block number
      end;
    cntr of begins:= cntr of begins - 1;
    block number:= block[cntr of begins];
    RE through barrier
  end RE ELAN block;

  procedure RE block begin;
  begin CHECK(begin symbol,210);
    RE poss decl id (false);
    RE ELAN declaration
  end RE block begin;

  procedure RE poss decl id (stat sep allowed);
  value stat sep allowed; Boolean stat sep allowed;
  begin comment The variable "declaration" is used to let the
    "search for identifier" process of section 9 know that a
    declaration is treated.
    Due to the definition of <comma> several statement separators
    may be read, for which "RE through barrier" has to be called.
    This situation is indicated by the value of "stat sep allowed".
    The purpose of the procedure is to read the first synt unit
```

```
52        a declaration starts with;
53        declaration:= true; RE;
54      st sep:
55        if synt unit = statement separator ∧ stat sep allowed then
56        begin RE through barrier; goto st sep end;
57        declaration:= false
58      end RE poss decl id;
59
60      comment:
61
62      3.  ELAN declaration.
63
64      <ELAN declaration>::=
65         <ord decl and MT decl> <statement separator> |
66         <MT declaration> <statement separator> | <statement separator>
67      <ord decl and MT decl>::= <list of initializations> |
68         <list of initializations> <comma> <MT declaration>
69      <MT declaration>::= 'MT' <list of identifiers>
70      <list of identifiers>::= <identifier> |
71         <identifier> <comma> <list of identifiers>
72      <comma>::= , | <comma> <statement separator>
73      <list of initializations>::= <initialization>|
74         <list of initializations> <comma> <initialization>
75      <initialization>::= <identifier>|<identifier> = <operand>
76
77      Algorithm:;
78
79      procedure RE ELAN declaration;
80      begin integer a,t;
81        ERROR(synt unit ≠ identifier ∧
82        synt unit ≠ MT declaration symbol ∧
83        synt unit ≠ statement separator, 300);
84        if synt unit = identifier then
85        begin L1: a:= place of identifier; CHECK(identifier,302);
86          if first scan then ERROR(¬ new identifier,301); RE;
87          if synt unit = equals symbol then
88          begin RE; t:= RE register or operand;
89            if first scan then
90            begin contents of[a]:= t;
91              contents of[a + 1]:= real to int(value of operand)
92            end else
93            if place enough then
94            begin comment The linenumber administration is updated,
95              see section 4 and section 9;
96              contents of[a + 2]:= -contents of[a + 2]
97            end
98          end else
99          begin if first scan then
100           begin contents of[a]:= declared; contents of[a+1]:= 0
101           end It is necessary that, during the first scan,
102             the declared variable obtains a value somewhere in a
```

```
103           marginal part. The value of "contents of[a]" then changes
104           into "STAT operand". During the second scan it is tested
105           whether this has actually occurred.
106         else ERROR(type of identifier ≠ STAT operand,305)
107       end;
108       if synt unit ≠ statement separator then
109                           REQUIRE (comma symbol, 310);
110       if synt unit = comma symbol then
111       begin RE poss decl id (true);
112         if synt unit = MT declaration symbol then goto MT; goto L1
113       end; CHECK fault:= false; RE
114     end;
115   MT: if synt unit = MT declaration symbol then
116     begin RE poss decl id (false);
117   L2: CHECK (identifier,312); if first scan then
118       begin ERROR(¬ new identifier,313);
119         contents of[place of identifier]:= MT declared;
120         contents of[place of identifier + 1]:= 0
121       end It is necessary that, during the first scan,
122         the MT declared variable obtains a value somewhere
123         in a marginal part. The value of "contents of[a]" then
124         changes into "MT declared and defined". During the second
125         scan it is tested whether this has actually occurred.
126       else ERROR(type of identifier ≠ MT declared and defined,315);
127       RE;
128       if synt unit ≠ statement separator then
129                           REQUIRE(comma symbol, 320);
130       if synt unit = comma symbol then
131       begin RE poss decl id (true); goto L2 end;
132       CHECK fault:= false; RE
133     end
134   end RE ELAN declaration;
135
136   comment:
137
138   4. Marginal part.
139
140   <marginal part>::=
141             <empty> | <label sequence> | <location and labels>
142   <label sequence>::= <label> | <label sequence> <label>
143   <label>::= <identifier>:
144   <location and labels>::=
145             <location>: | <location>: <label sequence>
146   <location>::= <known location>
147   <known location>::= M[<expression>] | <identifier>[expression>]
148
149   Algorithm:;
150
151   procedure RE marginal part;
152   if ¬ RE label sequence then RE location and labels;
```

```
153
154     Boolean procedure RE label sequence;
155     begin RE label sequence:= false;
156     again:if synt unit = identifier ∧ next symbol = colon symbol then
157       begin RE label sequence:= true;
158         if first scan then
159         begin ERROR(new identifier,400);
160           if type of identifier = declared then
161           contents of[place of identifier]:= STAT operand else
162                   if type of identifier = MT declared then
163           contents of[place of identifier]:= MT declared and defined
164           else ERROR(type of identifier ≠ declared ∧
165                      type of identifier ≠ MT declared, 410);
166           contents of[place of identifier + 1]:= address counter
167         end else if place enough then
168         begin comment The linenumber of defining occurence of
169           the label is placed negatively in LINE. See section 9;
170           integer a;
171           a:= contents of[place of identifier + 3];
172           a:= a - a : max of buffer × max of buffer;
173           LINE[a]:= -LINE[a]
174         end;
175         RE; RE; comment PR ad cntr prints the address counter;
176         if synt unit = statement separator then PR ad cntr;
177         goto again
178       end
179     end RE label sequence;
180
181     procedure RE location and labels;
182     if(synt unit = M symbol ∨ synt unit = identifier) ∧
183        next symbol = sub symbol then
184     begin LOOK AHEAD sub text bus;
185       if next symbol = colon symbol then
186       begin if second scan then PUNCH; address counter:= 0;
187         if synt unit = identifier then
188         begin if new identifier then ERROR(true,415) else
189           begin ERROR(type of identifier ≠ STAT operand,420);
190             address counter:= value of identifier
191           end
192         end;
193         RE; RE; Add to ad cntr(RE expression);
194         REQUIRE(bus symbol, 430);
195         RE; RE; RE label sequence
196       end
197     end RE location and labels;
198
199     comment:
200
201     5.  ELAN statement, ELAN instruction.
202
203     <ELAN statement>::= <marginal part> <ELAN instruction>
```

```
204     <ELAN instruction>::=
205           <empty> | <STAT address operand> | <unsigned real> |
206           <adding operator> <unsigned number> |
207           <BI or IP instruction> | <SKIP instruction> |
208           <ELAN block> <letgit string option> |
209           <UYN part> <functional instruction> <PZE part> |
210           <UYN part> <arithmetic instruction> <PZE part>
211     <BI or IP instruction>::= 'BI' | 'IP'
212     <SKIP instruction>::= 'SKIP' <expression>
213     <letgit string option>::= <letter> <letgit string option> |
214                               <digit> <letgit string option> |
215                               <empty>
216     <UYN part>::= <empty> | U, | Y, | N,
217     <PZE part>::= <empty> |, P |, Z |, E
218     <functional instruction>::=
219                 <special identifier> | <shift instruction> |
220                 <special identifier> ( <right operand> )
221     <shift instruction>::= <shift identifier>(<shift expr>)
222     <shift identifier>::= LCA | ... | RUSA | LVIFA | ... | IFSC |
223                           LVIFON | ... | AFOFF
224     <shift expr>::= <expression> | B | B + <unsigned expression>
225     The value v of an expression in the shift expr
226     of a shift instruction should satisfy either
227     0 ≤ v ≤ 31 or 0 ≤ v ≤ 1 or 0 ≤ v ≤ 39,
228     the upper bound depending on the shift identifier.
229     <special identifier>::= plusa | ... | subcd
230     <arithmetic instruction>::= <register> <operator> <operand> |
231                                 <left operand> <operator> <register>
232     <register>::= A | S | B | F | G
233     <operator>::= = | =- | <adding operator> | <multiplying operator>
234     <adding operator>::= + | - | '+' | '+'-
235     <multiplying operator>::= × | / | 'x' | 'x'-
236     <unsigned number>::= <unsigned integer> | <unsigned real>
237     <unsigned integer>::= <unsigned decimal> | '<unsigned octal>'
238
239     Algorithm:;
240
241     procedure RE ELAN statement;
242     begin RE marginal part; RE ELAN instruction;
243       if synt unit ≠ statement separator ∧
244          synt unit ≠ end symbol then
245       begin ERROR (true, 500); SKIP until statement separator end
246     end RE ELAN statement;
247
248     procedure RE ELAN instruction;
249     if synt unit = number ∨ is adding operator(synt unit) then
250     begin Boolean neg;
251       neg:= synt unit = minus symbol;
252       if synt unit ≠ number then
253       begin RE; CHECK (number, 510) end;
254       if neg then
```

```
255       begin if type of number = real type then
256         value of real number:= - value of real number else
257         value of number:= - value of number
258       end;
259       PRODUCE NUMBER CODE; Add to ad cntr(1); RE
260     end else
261     if synt unit = BI symbol then
262     begin IP:= false; RE end else
263     if synt unit = IP symbol then
264     begin IP:= true; RE end else
265     if synt unit = SKIP symbol then
266     begin PUNCH;
267           RE; Add to ad cntr(RE expression)
268     end else
269     if synt unit = begin symbol then RE ELAN block else
270     if ¬(synt unit = statement separator ∨
271           synt unit = end symbol) then
272     begin if ¬ initialization ∧ first scan then
273       begin SKIP until statement separator;
274             Add to ad cntr (1)
275       end else
276       if synt unit = open symbol ∨ synt unit = colon symbol then
277       begin PRODUCE EXPR CODE( RE expression );
278         Add to ad cntr(1)
279       end else
280       begin if is UYN symbol(synt unit) then
281         begin UYN:= synt unit; RE;
282           CHECK( comma symbol, 520); RE
283         end else
284         UYN:= -1;
285         if synt unit = function identifier then
286         RE functional instruction else RE arithmetic instruction;
287         if synt unit = comma symbol then
288         begin RE; ERROR(¬ is PZE symbol(synt unit),530);
289           PZE:= synt unit; RE
290         end else PZE:= -1;
291         if initialization then DEFINE INF WORDS else
292                 if second scan then PRODUCE INSTR CODE;
293         Add to ad cntr(1)
294       end
295     end RE ELAN instruction;
296
297     procedure RE arithmetic instruction;
298     begin if is register symbol(synt unit) then
299       begin register:= index of register; RE;
300         ERROR(¬ is operator symbol(synt unit),540);
301         if is adding operator(synt unit) then
302         begin minus for right operand:= synt unit = minus symbol;
303           operator:= index of operator(plus symbol); RE
304         end else
305         begin operator:= index of operator(synt unit); RE;
```

```
306           RE possible minus symbol
307         end;
308         right operand:= RE register or operand;
309         if right operand = STAT ad op ∧ value of operand < 0 then
310         begin ERROR(true, 545); value of operand:=-value of operand;
311           minus for right operand:= ¬ minus for right operand
312         end;
313         INF WORD:= reg op op[register,operator,right operand];
314         d26INF WORD:= d26reg op op[register,operator,right operand];
315         type of instruction:= reg op op instruction
316       end else
317       begin left operand:= RE operand;
318         if left operand = STAT ad op ∨
319         left operand = DYN ad op then left operand:=ERROR(true,550);
320         ERROR(¬ is operator symbol(synt unit),560);
321         if is adding operator(synt unit) then
322         begin minus for right operand:= synt unit = minus symbol;
323           operator:= index of operator(plus symbol); RE
324         end else
325         begin operator:= index of operator(synt unit); RE;
326           RE possible minus symbol
327         end;
328         ERROR(¬ is register symbol(synt unit),570);
329         register:= index of register; RE;
330         INF WORD:= op op reg[left operand,operator,register];
331         d26INF WORD:= d26op op reg[left operand,operator,register];
332         type of instruction:= op op reg instruction
333       end
334     end RE arithmetic instruction;
335
336     procedure RE functional instruction;
337     begin integer upper bound;
338       fctn:= type of function identifier; RE;
339       if fctn < t8 then upper bound:=t15 - 1 else
340       if fctn < t9 then begin fctn:=fctn-t8; upper bound:=31 end else
341       if fctn < t10 then begin fctn:=fctn-t9; upper bound:=1 end else
342                          begin fctn:=fctn - t10; upper bound:=39 end;
343       if synt unit = open symbol then
344       begin RE; if upper bound < t15 - 1 then
345         begin if synt unit = B symbol then
346           begin right operand:= STAT B operand; RE;
347             if synt unit = close symbol then
348             begin value of operand:=0;
349               minus for right operand:= false ;
350               goto END
351             end
352           end else
353           begin right operand:= STAT operand end;
354           value of operand:= RE expression;
355           minus for right operand:= false ;
```

```
356           ERROR(second scan ∧
357             ¬(0<value of operand ∧ value of operand ≤ upper bound),
358               580)
359         end else
360         begin RE possible minus symbol;
361           right operand:= RE register or operand;
362           if right operand = STAT ad op ∧ value of operand < 0 then
363           begin ERROR( true , 585);
364             value of operand:= - value of operand;
365             minus for right operand:= ¬ minus for right operand
366           end
367         end;
368       END: REQUIRE (close symbol,590); RE
369       end else
370       begin ERROR (upper bound < t15 - 1, 595);
371         right operand:= 0; value of operand:= 0;
372         minus for right operand:= false
373       end;
374       INF WORD:= fctn op[fctn,right operand];
375       d26INF WORD:= d26fctn op[fctn,right operand];
376       type of instruction:= fctn instruction
377     end RE functional instruction;
378
379     procedure RE possible minus symbol;
380     if synt unit = minus symbol then
381     begin RE; minus for right operand:= true end else
382               minus for right operand:= false;
383
384     integer procedure RE register or operand;
385     if is register symbol(synt unit) then
386     begin RE register or operand:= STAT operand;
387       value of operand:= value of register; RE
388     end else RE register or operand:= RE operand;
389
390     comment:
391
392     6.  Operand.
393
394     <left operand>::=<STAT operand> | <STATB operand> | <DYN operand>
395     <address operand>::= <STAT address operand> |
396                          <DYN address operand>
397     <operand>::= <left operand> | <address operand>
398     <STAT operand>::= M | M[<expression>] | <identifier> | T |
399                       <identifier>[<expression>]
400     <STATB operand>::= M[B<Bexpr>] | <identifier> [B<Bexpr>]
401     <Bexpr>::= <empty> | <adding operator> <unsigned expression>
402     <DYN operand>::= <DYN M symbol> |
403                      <DYN M symbol>[<q expr>]
404     <DYN M symbol>::= MG | MA | MS | MC | MT | MD | M <p expr>
405     <q expr>::= <expression>
406     <p expr>::= <digit> | <digit> <digit>
```

```
407     <STAT address operand>::= : <STAT operand> | <unsigned integer> |
408                                 (<expression>) | : <register>
409     <DYN address operand>::= : <DYN operand>
410
411     Algorithm:;
412
413     integer procedure RE operand;
414     if synt unit = colon symbol then
415     begin integer op;
416       RE; op:= RE register or operand;
417       RE operand:= if op = STAT operand then STAT ad op else
418                    if op = DYN operand then DYN ad op else
419                   ERROR( true , 600)
420     end else
421     if synt unit = number then
422     begin ERROR(type of number ≠ integral type,610);
423       RE operand:= STAT ad op; value of operand:= value of number; RE
424     end else
425     if synt unit = open symbol then
426     begin RE; RE operand:= STAT ad op;
427       value of operand:= RE expression;
428       REQUIRE (close symbol, 620); RE
429     end else
430     if synt unit = DYN M symbol then
431     begin integer p,q;
432       RE operand:= DYN operand;
433       p:= type of DYN M symbol;
434       p:= if p = - G symbol then 58 else
435           if p = - A symbol then 59 else
436           if p = - S symbol then 60 else
437           if p = - C symbol then 61 else
438           if p = - T symbol then 62 else
439           if p = - D symbol then 63 else p; RE;
440       if synt unit = sub symbol then
441       begin RE; q:= RE expression;
442         REQUIRE (bus symbol,630); RE
443       end else q:= 0;
444       ERROR(0 > p ∨ p > 63,635); ERROR(-256 > q ∨ q > 255,640);
445       value of operand:= p × 512 + 256 + q
446     end else
447     if synt unit = M symbol ∨ synt unit = identifier then
448     begin integer type;
449       if synt unit = identifier then
450       begin ERROR(new identifier,650);
451         type:= type of identifier;
452         ERROR(type = declared ∨ type = MT declared,655);
453         value of operand:= if type = MT declared and defined then
454         value of identifier - (address counter + 1)
455         else value of identifier
456       end else
457       begin type:= STAT operand; value of operand:= 0 end;
```

```
458       RE; if synt unit = sub symbol then
459       begin RE; if synt unit = B symbol then
460         begin RE; ERROR(type ≠ STAT operand,660);
461           type:= STATB operand;
462           value of operand:= value of operand + 16384;
463           if synt unit = bus symbol then goto END
464         end;
465         value of operand:= value of operand + RE expression;
466     END: REQUIRE (bus symbol,670); RE
467       end;
468       if type = MT declared and defined then
469       begin ERROR(-256>value of operand ∨ value of operand >255,680);
470         value of operand:= 62 × 512 + 256 + value of operand;
471         RE operand:= DYN operand
472       end else RE operand:= if type = declared then STAT operand else
473       if type = MT declared then DYN operand else type
474     end else
475     begin RE operand:= STAT operand;
476       value of operand:= if synt unit = T symbol then 62 else
477                          ERROR(true,690); RE
478     end RE operand;
479
480     comment:
481
482     7.  Expression.
483
484     <expression>::= <unsigned expression> |
485                     <adding operator> <unsigned expression>
486     <unsigned expression>::= <term> |
487               <unsigned expression> <adding operator> <term>
488     <term>::= <primary> | <term> <multiplying operator> <primary>
489     <primary>::= <STAT address operand>
490
491     Algorithm:;
492
493     integer procedure logic sum (a, b); real a,b;
494     logic sum:= logic operation (a, b, 1);
495
496     integer procedure logic prod (a, b); real a,b;
497     logic prod:= logic operation (a, b, 2);
498
499     integer procedure logic operation (a, b, opcode);
500     value a, b, opcode; real a, b; integer opcode;
501     begin integer ia, ib, ia2, ib2, res, k, tk;
502       Boolean nega, negb;
503       Boolean procedure op (a, b); value a, b; Boolean a, b;
504       op:= if opcode = 1 then ¬a = b else
505            if opcode = 2 then a ∧ b else false ;
506
507       ia:= real to int(a); ib:= real to int(b);
508       nega:= 1 / ia < 0; negb:= 1 / ib < 0;
```

```
509       res:= if op(nega, negb) then t26 else 0;
510       if nega then ia:= ia - t26; if negb then ib:= ib - t26;
511       for k:= 0 step 1 until 25 do
512       begin tk:= if k = 0 then t0 else tk × t1;
513         ia2:= ia : t1; ib2:= ib : t1;
514         if op(ia ≠ ia2 × t1, ib ≠ ib2 × t1) then res:= res + tk;
515         ia:= ia2; ib:= ib2
516       end;
517       logic operation:= res
518     end logic operation;
519
520     real procedure RE expression;
521     begin
522       real procedure elevator(floor); value floor; integer floor;
523       if floor = 0 then elevator:= primary else
524       begin real el;
525         integer s;
526         el:= elevator(floor - 1);
527       again: s:= synt unit;
528         if floor = 2 ∧
529         (s = plus symbol ∨ s = minus symbol ∨ s = logic plus symbol)∨
530         floor = 1 ∧
531         (s = times symbol ∨ s = over symbol ∨ s = logic times symbol)
532         then
533         begin RE;
534           el:= if s =  plus symbol then el + elevator(floor - 1)
535           else if s = minus symbol then el - elevator(floor - 1)
536           else if s = logic plus symbol then
537                                 logic sum( el, elevator( floor - 1 ))
538           else if s = times symbol then el × elevator(floor - 1)
539           else if s =  over symbol then el : elevator(floor - 1)
540           else if s = logic times symbol then
541                                 logic prod( el, elevator( floor - 1 ))
542           else 1; goto again
543         end;
544         elevator:= el
545       end elevator;
546
547       real procedure primary;
548       if synt unit = plus symbol then
549       begin RE; primary:= primary end else
550       if synt unit = minus symbol then
551       begin RE; primary:= - primary end else
552       begin real t;
553         t:= RE operand; ERROR( t ≠ STAT ad op, 700);
554         primary:= value of operand
555       end primary;
556
557       RE expression:= elevator(2);
558     END:
559     end RE expression;
```

```
560
561     comment:
562
563     8.  Reading equipment.
564
565     Algorithm:;
566
567     integer tape symbol, pr tape symbol, case code, lower case,
568       lower case code, upper case, upper case code, dummy code,
569       error code, endsymcount;
570     integer array symcode[0:255];
571     boolean stat sep barrier, NS deferred, from string;
572
573     procedure RE;
574     if ¬ initialization ∧ stat sep barrier ∧
575       (synt unit = statement separator ∨ synt unit = end symbol) then
576     retain old synt unit: else
577     if CHECK fault then CHECK fault:= false else
578     begin if not behind last end then synt unit := READ synt unit;
579       if synt unit = begin symbol then
580       nr of begins:= nr of begins + 1 else
581       if synt unit = end symbol then
582       begin nr of begins:= nr of begins - 1;
583         if nr of begins > 0 then
584         begin L:
585           if is letter(symbol) ∨ is digit(symbol) then
586           begin NS; goto L end
587         end else not behind last end:= false
588       end
589     end RE;
590
591     integer procedure READ synt unit;
592     if NS deferred then
593     begin NS; NS deferred:= false;
594       READ synt unit:= READ synt unit
595     end else
596     if is letter(symbol) then
597     begin integer p,q,n,A,m,q1,q2,q3,i;
598       Boolean end;
599       p:= ptr of inf list; n:= A:= 0;
600     again: n:= n + 1; A:= A × t6 + symbol + 1; NS;
601       end:= ¬(is letter(symbol) ∨ is digit(symbol));
602       if n = n : 4 × 4 ∨ end then
603       begin p:= p + 1; ERROR(p > max of inf list - 10,800);
604         contents of[p]:= q:= A; A:= 0
605       end;
606       if ¬ end then goto again;
607       m:= p - ptr of inf list; contents of[ptr of inf list]:= m;
608       if n = 1 then
609       begin q1:= q - 1;
610         for i:= A symbol, B symbol, E symbol, F symbol, G symbol,
```

```
611                 M symbol, N symbol, P symbol, S symbol, T symbol,
612                 U symbol, Y symbol, Z symbol do
613         if i = q1 then begin READ synt unit:= i; goto END end
614       end else
615       if n = 2 then
616       begin q1:= q : t6; q2:= q - q1 × t6 - 1; q1:= q1 - 1;
617         if q1 = M symbol then
618         begin for i:= A symbol, C symbol, D symbol,
619                       G symbol, S symbol, T symbol do
620           if i = q2 then
621           begin READ synt unit:= DYN M symbol;
622             type of DYN M symbol:= -i; goto END
623           end;
624           if is digit(q2) then
625           begin READ synt unit:= DYN M symbol;
626             type of DYN M symbol:= q2; goto END
627           end
628         end else
629         if initialization then
630         begin if q1 = Y symbol ∧ q2 = N symbol then
631           begin READ synt unit:= YN symbol; goto END end
632         end
633       end else
634       if n = 3 then
635       begin q1:= q : t12; q2:= q - q1 × t12; A:= q2 : t6;
636         q3:= q2 - A × t6; q2:= A - 1; q1:= q1 - 1; q3:= q3 - 1;
637         if q1 = M symbol ∧ is digit(q2) ∧ is digit(q3) then
638         begin READ synt unit:= DYN M symbol;
639           type of DYN M symbol:= q2 × 10 + q3; goto END
640         end else
641         if initialization then
642         begin if q1 = U symbol ∧ q2 = Y symbol ∧ q3 = N symbol then
643           begin READ synt unit:= UYN symbol; goto END end else
644           if q1 = P symbol ∧ q2 = Z symbol ∧ q3 = E symbol then
645           begin READ synt unit:= PZE symbol; goto END end
646         end
647       end;
648       contents of[p]:=q × t6 ↑ (m × 4 - n);
649       READ synt unit:=STORE letgits with(block number);
650       comment The procedure "STORE letgits with" stores the name
651       read with all its information.;
652     END:
653     end is letter else if symbol = apostrophe symbol then
654     begin NS; if is letter(symbol) then
655       begin integer synt unit;
656         if symbol = letter b then
657         begin NS;
658           if symbol = letter e then synt unit:= begin symbol else
659           if symbol = letter i then synt unit:= BI symbol else
660           synt unit:= ERROR( true , 811 )
661         end else
```

```
662         if symbol = letter e then synt unit:= end symbol else
663         if symbol = letter i then synt unit:= IP symbol else
664         if symbol = letter S then synt unit:= SKIP symbol else
665         if symbol = letter M then synt unit:=MT declaration symbol
666         else synt unit:= ERROR( true , 810 );
667         READ synt unit:= synt unit;
668       LA: NS; if is letter (symbol) then goto LA;
669         if symbol ≠ apostrophe symbol then ERROR (true, 815) else
670         if synt unit ≠ end symbol ∨ nr of begins > 1 then NS
671       end else
672       if symbol = plus symbol ∨ symbol = times symbol then
673       begin READ synt unit:=
674         if symbol = plus symbol then logic plus symbol else
675         logic times symbol; NS;
676         if symbol ≠ apostrophe symbol then ERROR (true, 820) else NS
677       end else
678       if is digit(symbol) then
679       begin integer i;
680         Boolean neg;
681         READ synt unit:= number;
682         type of number:= integral type; neg:= symbol ≥ 4;
683         value of number:= 0; i:= 0;
684       again: i:= i + 1; if symbol > 7 then
685         begin ERROR(true,830); symbol:= 0 end;
686         value of number:= value of number × 8 + symbol;
687         ERROR(i = 10,835); NS; if is digit(symbol) then goto again;
688         if neg∧i=9 thenvalue of number:=value of number-real t26+t26;
689         if symbol ≠ apostrophe symbol then ERROR (true, 840) else NS
690       end else
691       if symbol = apostrophe symbol then
692       begin symbol:= quote symbol;
693         READ synt unit:= READ synt unit
694       end else
695       ERROR(true,860)
696     end apostrophe symbol else
697
698     if is digit(symbol) ∨ symbol = point symbol ∨
699                           symbol = lower ten symbol then
700     begin real r,length,i,i1;
701       real procedure integer;
702       begin i:= symbol; ERROR(¬ is digit(symbol),870); length:= 10;
703       again: NS; if is digit(symbol) then
704         begin i:=i×10 + symbol; length:=length × 10; goto again end;
705         integer:= i
706       end integer;
707
708       READ synt unit:= number; type of number:= integral type;
709       if symbol=lower ten symbol then begin r:=1; goto LOWER TEN end;
710       if is digit(symbol) then i1:= integer else i1:= 0;
711       if symbol = point symbol then
712       begin type of number:= real type; NS;
```

```
713         r:= i1 + integer/length
714       end else r:= i1;
715     LOWER TEN: if symbol = lower ten symbol then
716       begin boolean minus;
717         NS; minus:= symbol = minus symbol;
718         if is adding operator(symbol) then NS;
719         r:= r × 10 ↑ (if minus then -integer else integer);
720         type of number:= real type
721       end;
722       if type of number = integral type then value of number:=i1 else
723       value of real number:= r
724     end number else
725
726     if symbol = nlcr symbol ∨ symbol = semicolon symbol then
727     begin if initialization then
728       begin L2: NS; if symbol = nlcr symbol then goto L2 end else
729       NS deferred:= true;
730       READ synt unit:= statement separator
731     end else
732
733     if symbol = quote symbol then
734     begin L3: NS;
735       if ¬(symbol=nlcr symbol ∨ symbol=semicolon symbol)then goto L3;
736       READ synt unit:= READ synt unit
737     end quote else
738
739     begin READ synt unit:= symbol; NS
740     end READ synt unit;
741
742     procedure RE through barrier;
743     begin stat sep barrier:= false;
744       RE; stat sep barrier:= true
745     end RE through barrier;
746
747     procedure LOOK AHEAD sub text bus;
748     begin RE NS; again: if next symbol = sub symbol then
749       begin LOOK AHEAD sub text bus; goto again end else
750       if next symbol = bus symbol then RE NS else
751       if next symbol = apostrophe symbol then
752       begin RE NS;
753         if next symbol=apostrophe symbol ∨ next symbol=letter ethen
754         ERROR (true, 876) else goto again
755       end else
756       if is stat sep (next symbol) then
757       ERROR (true, 876) else
758       begin RE NS; goto again end
759     end LOOK AHEAD sub text bus;
760
761     procedure SKIP until statement separator;
762     begin L: if symbol = apostrophe symbol then
763       begin RE;
```

```
764         if synt unit≠end symbol ∧ synt unit≠statement separator then
765         goto L
766       end else
767       if ¬ is stat sep (symbol) then
768       begin NS; goto L end else RE
769     end SKIP until statement separator;
770
771     procedure NS;
772     begin
773     L: if ¬ initialization then PR ELAN SYM (pr tape symbol);
774       ptr of text:= ptr of text + 1;
775       if ptr of text ≤ reading ptr then
776       pr tape symbol:= ELAN line[ptr of text] else
777       begin RE NS; pr tape symbol:= tape symbol end;
778       if is layout (pr tape symbol) then goto L;
779       symbol:= if pr tape symbol = pr sub symbol then sub symbol else
780                if pr tape symbol = pr bus symbol then bus symbol else
781                if 37 ≤ pr tape symbol ∧ pr tape symbol ≤ 62 then
782                pr tape symbol - 27 else pr tape symbol
783     end NS;
784
785     procedure RE NS;
786     begin L: reading ptr:= reading ptr + 1;
787       if first scan then
788       begin tape symbol:= RESYM1;
789         if ¬ initialization then stow into buffer (tape symbol)
790       end else
791       tape symbol:= fetch from buffer;
792       if readingptr=ptr of text then reading ptr:=ptr of text:=0 else
793       begin if reading ptr > max of ELAN line then ERROR(true,873)
794         else ELAN line[reading ptr]:= tape symbol;
795         if is layout (tape symbol) then goto L
796       end;
797       next symbol:=if tape symbol=pr sub symbol then sub symbol else
798                    if tape symbol=pr bus symbol then bus symbol else
799                    if 37 ≤ tape symbol ∧ tape symbol ≤ 62 then
800                    tape symbol - 27 else tape symbol
801     end RE NS;
802
803     integer procedure RESYM1;
804     begin integer s;
805       s:= if from string then Init stringsym else RESYM;
806       if s = 255 then
807       begin from string:= false; s:= nlcr symbol end;
808       RESYM1:= s
809     end RESYM1;
810
811     integer procedure RESYM;
812     begin integer hep, code;
813     L: if rehep available then hep:= REHEP else
814       begin code:= endstringsym; goto endresym end;
```

```
815       if hep ≥ t7 then
816       begin ERROR (true, 871); hep:= hep - hep : t7 × t7 end;
817       if hep = lower case then
818       begin case code:= lower case code; goto L end;
819       if hep = upper case then
820       begin case code:= upper case code; goto L end;
821       code:= symcode[case code + hep];
822       if code = dummy code then goto L;
823       if code = error code then begin ERROR (true, 872); goto L end;
824     endresym: RESYM:= code
825     end RESYM;
826
827     integer procedure endstringsym;
828     begin integer sym;
829       sym:= STRINGSYMBOL(endsymcount,↓
830       'end'
831       ↑);
832       if sym= nlcr symbol ∧ endsymcount= 0 then ERROR(true, 874);
833       endsymcount:= if sym= nlcr symbol ∧ endsymcount> 1 then 0 else
834         endsymcount+1;
835       endstringsym:= sym
836     end endstringsym;
837
838     boolean procedure rehep available;
839     rehep available:= ¬ first scan = second scan;
840
841     comment:
842
843     9.  Name list equipment and secondary storage usage.
844
845     The procedure STORE letgits with (block number) takes care that
846     each identifier of the ELAN source text is entered into the
847     name list. It uses the procedures "STORE in contents of" , "ST",
848     "store in LINE" and "st" for storing variable amounts
849     of information into the arrays "contents of" and "LINE".
850     The strategy is:
851     All incoming information is stored linearly in the array
852     "contents of" , and pointers connect the names
853     alphabetically together. In the lower part of this array,
854     the special identifiers are stored.
855     The administration section for each identifier i is as follows:
856     (a) number of memory places of i.
857     (b) the letters and/or digits of i are stored in
858         groups of four letters/digits into one memory place.
859     (c) the pointer to the next name, which preceeds alphabeticly i.
860     (d) the pointer to the next name, which follows alphabeticly i.
861     (e) the pointer to the administration section of i.
862     (f) block number.
863     (g) type.
864     (h) value.
865     (i) line of declaration.
```

```
866     (j) pointer to the memory place in the integer array "LINE" for
867         storing the calls of i.
868     (k) pointer to the next administration section of an identifier
869         with the same name, but with another block number.
870
871     Algorithm:;
872
873     integer array contents of [1:22x1024], LINE[1:4096];
874     integer ptr of inf list, pointer of ptr of inf list,drum pointer,
875       place of identifier,max of inf list,end fctn part,line pointer;
876     boolean place enough;
877
878     integer.procedure STORE letgits with(block number);
879     value block number; integer block number;
880     begin integer point in tree, num in tree, old letgits, new num,
881       num, result of comparison, pointer to next identifier,
882       first administration cell, block number in tree,
883       pointer to connect with, new letgits, i;
884
885       procedure OLD IDENTIFIER;
886       begin new identifier:= false;
887         place of identifier:= first administration cell + 1;
888         type of identifier:=contents of[first administration cell+1];
889         if type of identifier= unknown type then
890         begin new identifier:=true; type of identifier:=declared end;
891         value of identifier:=
892                       contents of [first administration cell + 2];
893         if ¬declaration ∧ block number ≠ 1 ∧
894                           block number ≠ 2 ∧ second scan then
895         begin
896           pointer to connect with:=
897                        contents of[first administration cell + 4];
898           contents of[first administration cell + 4]:=
899           store in LINE(ST(ST(0,
900                           line number),
901                           pointer to connect with))
902         end;
903         goto OUT
904       end OLD IDENTIFIER;
905
906       procedure NEW IDENTIFIER;
907       if ptr of inf list < end fctnpart then
908       begin STORE in contents of (st(0,
909                                      block number));
910         goto OUT
911       end else
912       begin new identifier:= true ;
913         place of identifier:= ptr of inf list + 1 ;
914         type of identifier:= declared;
915         value of identifier:= 0;
916         if block number ≠ 0 then
```

```
917         STORE in contents of (st(st(st(st(st(0,
918                                 block number),
919                                 unknown type),
920                                 value of identifier),
921                                 line number),
922                                 0),
923                                 pointer to connect with))
924         else
925         STORE in contents of (st(st(st(st(st(0,
926                                 block number),
927                                 unknown type),
928                                 value of identifier),
929                                 0),
930                                 store in LINE(ST(ST(0,
931                                                 line number),
932                                                 0))),
933                                 0));
934         goto OUT
935       end NEW IDENTIFIER;
936
937       point in tree:= 1; new num:= contents of [ptr of inf list];
938       if new num > max of num then max of num:= new num;
939     COMPARE:
940       num in tree:= contents of [point in tree];
941       num:= if num in tree < new num then num in tree else new num;
942       for i:= 1 step 1 until num do
943       begin old letgits:= contents of [point in tree + i];
944         new letgits:= contents of [ptr of inf list + i];
945         if new letgits < old letgits then
946         begin result of comparison:=1;
947           goto after comparison
948         end else
949         if new letgits > old letgits then
950         begin result of comparison:= 2; goto after comparison end
951       end;
952       result of comparison:= if new num < num in tree then 1 else
953                              if new num > num in tree then 2 else 3;
954     after comparison:
955       pointer to next identifier:=
956       contents of[point in tree+num in tree+result of comparison];
957       if result of comparison ≠ 3 then
958       begin if pointer to next identifier= 0 then
959         begin contents of[
960           point in tree+num in tree+result of comparison]:=
961                                 point in tree:= ptr of inf list;
962           pointer of ptr of inf list:=
963                       ptr of inf list:= ptr of inf list + new num + 1;
964           STORE in contents of (st(st(st(0,
965                                   0),
966                                   0),
967                                   STORE in contents of (0)));
```

```
968           if ¬ declaration then block number:= 0 else
969           pointer to connect with:= 0;
970           NEW IDENTIFIER
971         end;
972         point in tree:= pointer to next identifier; goto COMPARE
973       end else
974       begin first administration cell:= pointer to next identifier;
975         block number in tree:=contents of[first administration cell];
976         if point in tree < end fctn part then
977         begin type of function identifier:= block number in tree;
978           goto OUT
979         end else
980         if declaration ∧ first scan then
981         begin if block number in tree = block number then
982           OLD IDENTIFIER else
983           begin
984            contents of[point in tree+num in tree+3]:=ptr of inf list;
985             pointer to connect with:= first administration cell;
986             NEW IDENTIFIER
987           end
988         end else
989         begin next: for i:= cntr of begins step -1 until 0 do
990           if block[i]= block number in tree then OLD IDENTIFIER;
991           if contents of [first administration cell + 5]= 0 then
992           begin
993            contents of[first administration cell+5]:=ptr of inf list;
994             block number:= 0; NEW IDENTIFIER
995           end;
996           first administration cell:=
997                              contents of[first administration cell + 5];
998           block number in tree:=
999                              contents of[first administration cell];
1000          goto next
1001        end
1002      end;
1003
1004  OUT: STORE letgits with:= if point in tree < end fctnpart then
1005                              function identifier else identifier
1006  end STORE letgits with;
1007
1008  integer procedure STORE in contents of(information);
1009  integer information;
1010  begin integer inf;
1011    STORE in contents of:= ptr of inf list:=
1012                             pointer of ptr of inf list;
1013    inf:= information;
1014    ptr of inf list:= pointer of ptr of inf list
1015  end STORE in contents of;
1016
1017  integer procedure st(x,y); integer x,y;
1018  begin integer aux, auxiliary pointer;
```

```
1019        pointer of ptr of inf list:= pointer of ptr of inf list + 1;
1020        ERROR(pointer of ptr of inf list > max of inf list, 880);
1021        aux:= x; auxiliary pointer:= ptr of inf list;
1022        contents of [ptr of inf list]:= y;
1023        ptr of inf list:= auxiliary pointer + 1;
1024        st:= 0
1025      end st;
1026
1027      integer procedure store in LINE(inf); value inf; integer inf;
1028      store in LINE:= if place enough then line pointer-1+drum pointer
1029                      else 0;
1030
1031      integer procedure ST(x,y); value x; integer x,y;
1032      begin line pointer:= line pointer + 1;
1033        if line pointer > max of buffer then
1034        begin if drum pointer ≤ drumptr - max of buffer ∧
1035                 drum pointer ≤ max of drum - 2 × max of buffer then
1036          begin TO DRUM(LINE,drum pointer);
1037            drum pointer:= drum pointer + max of buffer;
1038            line pointer:= 1
1039          end else
1040          begin ERROR(drum pointer > drumptr - max of buffer, 881);
1041            ERROR(drum pointer ≥ max of drum - 2 × max of buffer, 882);
1042            place enough:= false
1043          end
1044        end;
1045        if place enough then LINE[line pointer]:= y;
1046        ST:= 0
1047      end ST;
1048
1049      integer count3, t8i, triple, buffer ptr, max of buffer,
1050        drum ptr, min of drum, max of drum;
1051      integer array buffer[1 : 4096];
1052
1053      procedure stow into buffer(s); value s; integer s;
1054      begin if s = nlcr symbol then
1055        begin skip space: if count3 = 0 then
1056          begin if buffer ptr = 0 then
1057            begin if drum ptr ≤ min of drum then
1058              begin t8i:= 1; triple:= 0; goto end skip space end else
1059              begin drum ptr:= drum ptr - max of buffer;
1060                FROM DRUM(buffer,drum ptr); buffer ptr:= max of buffer
1061              end
1062            end;
1063            triple:= buffer[buffer ptr];
1064            buffer ptr:= buffer ptr - 1; count3:= 3; t8i:= t8 × t8 × t8
1065          end;
1066          t8i:= t8i/t8; s:= triple : t8i; if s = space symbol then
1067          begin triple:= triple - s × t8i;
1068            count3:= count3 - 1; goto skip space
1069          end;
```

```
1070        t8i:= t8i × t8;
1071      end skip space:
1072        s:= nlcr symbol
1073      end;
1074      if count3 < 3 then
1075      begin triple:= s × t8i + triple; t8i:= t8i × t8;
1076        count3:= count3 + 1
1077      end else
1078      begin buffer ptr:= buffer ptr + 1;
1079        if buffer ptr > max of buffer then
1080        BUFFER TO DRUM; buffer[buffer ptr]:= triple;
1081        triple:= s; t8i:= t8; count3:= 1
1082      end
1083    end stow into buffer;
1084
1085    procedure BUFFER TO DRUM;
1086    begin ERROR (drum ptr + max of buffer - 1 > max of drum, 875);
1087      TO DRUM (buffer, drum ptr);
1088      drum ptr:= drum ptr + max of buffer; buffer ptr:= 1
1089    end BUFFER TO DRUM;
1090
1091    integer procedure fetch from buffer;
1092    begin integer trip;
1093      if count3 > 3 then
1094      begin buffer ptr:= buffer ptr + 1;
1095        if buffer ptr > max of buffer then BUFFER FROM DRUM;
1096        triple:= buffer[buffer ptr]; count3:= 0
1097      end;
1098      trip:=triple : t8; fetch from buffer:=abs(triple - trip × t8);
1099      triple:= trip; count3:= count3 + 1
1100    end fetch from buffer;
1101
1102    procedure BUFFER FROM DRUM;
1103    begin FROM DRUM (buffer, drum ptr);
1104      drum ptr:= drum ptr + max of buffer; buffer ptr:= 1
1105    end BUFFER FROM DRUM;
1106
1107    comment:
1108
1109    10.  Initialization of information words.
1110
1111    Algorithm:;
1112
1113    integer STAT operand, STATB operand, DYN operand, STAT ad op,
1114      DYN ad op, unknown type, type of instruction, INF WORD,
1115      reg op op instruction, op op reg instruction, fctn instruction;
1116    boolean d26 INF WORD;
1117    integer array reg op op[1:5,1:6,1:5],
1118                  op op reg[1:3,1:6,1:5], fctn op[1:140,0:5];
1119    boolean array d26reg op op[1:5,1:6,1:5],
1120                  d26op op reg[1:3,1:6,1:5], d26fctn op[1:140,0:5];
```

```
1121
1122    procedure INITIALIZE inf words;
1123    begin integer i,j,k;
1124      for i:= 1,2,3 do for j:= 1,2,3,4,5,6 do for k:= 1,2,3,4,5 do
1125      op op reg[i,j,k]:= 1;
1126      for i:=1,2,3,4,5 do for j:=1,2,3,4,5,6 do for k:=1,2,3,4,5 do
1127      reg op op[i,j,k]:= 1;
1128      for i:= 1 step 1 until number of functions do
1129      for j:= 0,1,2,3,4,5 do fctn op[i,j]:= 1;
1130      STAT operand:= 1; STATB operand:= 2; DYN operand:= 3;
1131      STAT ad op:= 4; DYNad op:= 5; unknown type:= 6;
1132      reg op op instruction:= 1; op op reg instruction:= 2;
1133      fctn instruction:= 3;
1134      block number:= max block number:= max block number + 1;
1135      cntr of begins:=cntr of begins + 1;
1136      block[cntr of begins]:=block number;
1137      RE block begin;
1138    again: CHECK(number,920); RE; CHECK(comma symbol,930);
1139      RE; RE ELAN instruction;
1140      if synt unit = number then goto again;
1141      CHECK(end symbol,940); cntr of begins:=cntr of begins - 1
1142    end INITIALIZE inf words;
1143
1144    procedure DEFINE INF WORDS;
1145    begin boolean not,d26;
1146      integer i,W;
1147      boolean array type is permitted[1:5];
1148      for i:= 1,2,3,4,5 do type is permitted[i]:= true;
1149      CHECK(statement separator,950);
1150      if symbol ≠ quote symbol then
1151      begin again: RE; i:= RE operand; type is permitted[i]:= false;
1152        if synt unit = comma symbol then goto again
1153      end;
1154      ERROR(symbol ≠ quote symbol,960);
1155      W:= i:= 0; not:= true;
1156    L:NS; if symbol = nlcr symbol then goto END;
1157      if not then
1158      begin not:= false; d26:= symbol = 1 end else
1159      begin i:= i + 1; ERROR(i > 26,970);
1160        if symbol > 1 then symbol:= 0; W:= W × 2 + symbol
1161      end;
1162      goto L;
1163    END: ERROR(i < 26,980);
1164      begin
1165        integer procedure WORD;
1166        WORD:= W +
1167          (if i = STAT operand  then 0 else
1168           if i = STAT ad op    then 1 else
1169           if i = STATB operand then 2 else
1170           if i = DYN operand   then 3 else
1171           if i = DYN ad op     then 0 else ERROR(true,985)) × t19 +
```

```
1172          (if minus for right operand then t1 - t21 else 0) +
1173          (if PZE > 0 then t2 else 0) +
1174          (if UYN = UYN symbol then t1 else
1175           if UYN =  YN symbol then t0 else 0) × t3;
1176
1177        procedure ASSIGN(B, w1, dw1, w2, dw2);
1178        boolean B, dw1, dw2; integer w1, w2;
1179        if B then
1180        begin for i:= STAT operand, STAT ad op,
1181                      STATB operand, DYN operand do
1182          begin if type is permitted[i] then
1183            begin w1:= WORD; dw1:= d26 end
1184          end
1185        end else
1186        begin i:= STAT operand; w2:= WORD; dw2:= d26 end ASSIGN;
1187
1188        if type of instruction = op op reg instruction then
1189        begin type is permitted[STAT ad op]:= false;
1190          ASSIGN(left operand = STAT operand,
1191                 op op reg[i,operator,register],
1192               d26op op reg[i,operator,register],
1193                  op op reg[left operand,operator,register],
1194               d26op op reg[left operand,operator,register])
1195        end else
1196        if type of instruction = reg op op instruction then
1197        ASSIGN(right operand = STAT operand,
1198                reg op op[register,operator,i],
1199             d26reg op op[register,operator,i],
1200                reg op op[register,operator,right operand],
1201             d26reg op op[register,operator,right operand]) else
1202        if type of instruction = fctn instruction then
1203        ASSIGN(right operand = STAT operand,
1204                fctn op[fctn,i],
1205             d26fctn op[fctn,i],
1206                fctn op[fctn,right operand],
1207             d26fctn op[fctn,right operand]) else
1208        ERROR(true,990); RE; RE
1209      end
1210    end DEFINE INF WORDS;
1211
1212    integer stringsymcount;
1213
1214    integer procedure Init stringsym;
1215    begin Init stringsym:= STRINGSYMBOL(stringsymcount,
1216  ⊧       ,
1217  G,A,S,C,T,D,B,M,[,],×,/,+,−,=,F,E,N,P,Y,U,Z,
1218  .,10,:,',",;,(,),b,e,i,S,M,
1219
1220  LVIFSC)INT,RCAS(GOTO,LCS(NORAS,RUAS(DIVA,IFOFF/JUMP,LUS(MINS,PLUSA,
1221  REPE,SUBC,AFON/DO,IFA)IFS)IVOFF,LCA(LUA(LVIFA)MINA,MULS,OVOFF,
1222  PLUSS,RCSA(REPZ,RUSA(TENAS,AFOFF/CLP,DIVAS,DOS,GOTOR,IFAC)IFON/
```

```
1223  IFSC)ITVON,IVON,JUMPR,LCAS(LCSA(LUAS(LUSA(LVIFAC)MEMPROT,MINB,
1224  MULAS,NORA,NORS,OVON,PLUSB,RCA(RCS(REP,REPP,RUA(RUS(SUB,SUBCD,
1225  TENS,LVIFON/LVIFOFF/LVIFS)REP3Z,REP1Z,REP5Z,REP0Z,REP2Z,REP4Z,
1226  REP6Z,REP0E,REP1E,REP2E,REP3E,REP4E,REP5E,REP6E,REP7E,REP0,REP1,
1227  REP2,REP3,REP4,REP5,REP6,REP7,REP0P,REP1P,REP2P,REP3P,REP4P,REP5P,
1228  REP6P,REP7P,REP7Z,SUB15,SUB11,SUB5,SUB1,SUB13,SUB3,SUB7,SUB0,
1229  SUB10,SUB12,SUB14,SUB2,SUB4,SUB6,SUB8,SUB9,TRAA,TRAS,TRAB,TRSA,
1230  TRSS,TRSB,TRBA,TRBS,TRBB,TRANA,TRANS,TRANB,TRSNA,TRSNS,TRSNB,
1231  TRBNA,TRBNS,TRBNB;
1232
1233  'begin' x = M[1000],
1234          STAT = M[2000],
1235          STATB = M[B],
1236          DYN = MC;
1237
1238  1,      UYN, A+x,       PZE;      "000 000 xx xx xx  xxxxx xxxxx xxxxx
1239  1.1,    UYN, A+:DYN,    PZE;      "000 100 01 xx xx  xxxxx xxxxx xxxxx
1240  2,      UYN, A-x,       PZE;      "000 001 xx xx xx  xxxxx xxxxx xxxxx
1241  2.1,    UYN, A-:DYN,    PZE;      "000 101 01 xx xx  xxxxx xxxxx xxxxx
1242  3,      UYN, A=x,       PZE;      "000 010 xx xx xx  xxxxx xxxxx xxxxx
1243  3.1,    UYN, A=:DYN,    PZE;      "000 110 01 xx xx  xxxxx xxxxx xxxxx
1244  4,      UYN, A=-x,      PZE;      "000 011 xx xx xx  xxxxx xxxxx xxxxx
1245  4.1,    UYN, A=-:DYN,   PZE;      "000 111 01 xx xx  xxxxx xxxxx xxxxx
1246  5,      UYN, x+A,       PZE;      "000 100 xx xx xx  xxxxx xxxxx xxxxx
1247  6,      UYN, x-A,       PZE;      "000 101 xx xx xx  xxxxx xxxxx xxxxx
1248  7,      YN,  x=A,       PZE;      "000 110 xx xx xx  xxxxx xxxxx xxxxx
1249  8,      YN,  x=-A,      PZE;      "000 111 xx xx xx  xxxxx xxxxx xxxxx
1250  9,   PLUSA(x), PZE; :STAT,:DYN;"000 110 xx xx 01  xxxxx xxxxx xxxxx
1251  10,  MINA(x),  PZE; :STAT,:DYN;"000 111 xx xx 01  xxxxx xxxxx xxxxx
1252
1253  11,     UYN, S+x,       PZE;      "001 000 xx xx xx  xxxxx xxxxx xxxxx
1254  11.1,   UYN, S+:DYN,    PZE;      "001 100 01 xx xx  xxxxx xxxxx xxxxx
1255  12,     UYN, S-x,       PZE;      "001 001 xx xx xx  xxxxx xxxxx xxxxx
1256  12.1,   UYN, S-:DYN,    PZE;      "001 101 01 xx xx  xxxxx xxxxx xxxxx
1257  13,     UYN, S=x,       PZE;      "001 010 xx xx xx  xxxxx xxxxx xxxxx
1258  13.1,   UYN, S=:DYN,    PZE;      "001 110 01 xx xx  xxxxx xxxxx xxxxx
1259  14,     UYN, S=-x,      PZE;      "001 011 xx xx xx  xxxxx xxxxx xxxxx
1260  14.1,   UYN, S=-:DYN,   PZE;      "001 111 01 xx xx  xxxxx xxxxx xxxxx
1261  15,     UYN, x+S,       PZE;      "001 100 xx xx xx  xxxxx xxxxx xxxxx
1262  16,     UYN, x-S,       PZE;      "001 101 xx xx xx  xxxxx xxxxx xxxxx
1263  17,     YN,  x=S,       PZE;      "001 110 xx xx xx  xxxxx xxxxx xxxxx
1264  18,     YN,  x=-S,      PZE;      "001 111 xx xx xx  xxxxx xxxxx xxxxx
1265  19,  PLUSS(x), PZE; :STAT,:DYN;"001 110 xx xx 01  xxxxx xxxxx xxxxx
1266  20,  MINS(x),  PZE; :STAT,:DYN;"001 111 xx xx 01  xxxxx xxxxx xxxxx
1267
1268  21,     UYN, B+x,       PZE;      "100 000 xx xx xx  xxxxx xxxxx xxxxx
1269  21.1,   UYN, B+:DYN,    PZE;      "100 100 01 xx xx  xxxxx xxxxx xxxxx
1270  22,     UYN, B-x,       PZE;      "100 001 xx xx xx  xxxxx xxxxx xxxxx
1271  22.1,   UYN, B-:DYN,    PZE;      "100 101 01 xx xx  xxxxx xxxxx xxxxx
1272  23,     UYN, B=x,       PZE;      "100 010 xx xx xx  xxxxx xxxxx xxxxx
1273  23.1,   UYN, B=:DYN,    PZE;      "100 110 01 xx xx  xxxxx xxxxx xxxxx
```

```
1274   24,     UYN, B=-x,       PZE;       "100 011 xx xx xx  xxxxx xxxxx xxxxx
1275   24.1,   UYN, B=-:DYN,    PZE;       "100 111 01 xx xx  xxxxx xxxxx xxxxx
1276   25,     UYN, x+B,        PZE;       "100 100 xx xx xx  xxxxx xxxxx xxxxx
1277   26,     UYN, x-B,        PZE;       "100 101 xx xx xx  xxxxx xxxxx xxxxx
1278   27,     YN,  x=B,        PZE;       "100 110 xx xx xx  xxxxx xxxxx xxxxx
1279   28,     YN,  x=-B,       PZE;       "100 111 xx xx xx  xxxxx xxxxx xxxxx
1280   29,  PLUSB(x), PZE; :STAT,:DYN;"100 110 xx xx 01  xxxxx xxxxx xxxxx
1281   30,  MINB(x),  PZE; :STAT,:DYN;"100 111 xx xx 01  xxxxx xxxxx xxxxx
1282
1283   31,     UYN, A'+'x,      PZE;:DYN;"010 100 xx xx xx  xxxxx xxxxx xxxxx
1284   32,     UYN, A'+'-x,     PZE;:DYN;"010 101 xx xx xx  xxxxx xxxxx xxxxx
1285   33,     UYN, A'×'x,      PZE;:DYN;"010 110 xx xx xx  xxxxx xxxxx xxxxx
1286   34,     UYN, A'×'-x,     PZE;:DYN;"010 111 xx xx xx  xxxxx xxxxx xxxxx
1287
1288   35,     UYN, S'+'x,      PZE;:DYN;"011 100 xx xx xx  xxxxx xxxxx xxxxx
1289   36,     UYN, S'+'-x,     PZE;:DYN;"011 101 xx xx xx  xxxxx xxxxx xxxxx
1290   37,     UYN, S'×'x,      PZE;:DYN;"011 110 xx xx xx  xxxxx xxxxx xxxxx
1291   38,     UYN, S'×'-x,     PZE;:DYN;"011 111 xx xx xx  xxxxx xxxxx xxxxx
1292
1293   39,     YN,  MULAS(x),  PZE;:DYN;"010 000 xx xx xx  xxxxx xxxxx xxxxx
1294   40,     YN,  MULAS(-x), PZE;:DYN;"010 001 xx xx xx  xxxxx xxxxx xxxxx
1295   41,     YN,  MULS(x),   PZE;:DYN;"010 010 xx xx xx  xxxxx xxxxx xxxxx
1296   42,     YN,  MULS(-x),  PZE;:DYN;"010 011 xx xx xx  xxxxx xxxxx xxxxx
1297
1298   43,     YN,  DIVAS(x),  PZE;:DYN;"011 000 xx xx xx  xxxxx xxxxx xxxxx
1299   44,     YN,  DIVAS(-x), PZE;:DYN;"011 001 xx xx xx  xxxxx xxxxx xxxxx
1300   45,     YN,  DIVA(x),   PZE;:DYN;"011 010 xx xx xx  xxxxx xxxxx xxxxx
1301   46,     YN,  DIVA(-x),  PZE;:DYN;"011 011 xx xx xx  xxxxx xxxxx xxxxx
1302
1303   47,     YN,  F=x,        PZE;       "110 010 xx xx xx  xxxxx xxxxx xxxxx
1304   47.1,   YN,  F=:DYN,     PZE;       "111 010 01 xx xx  xxxxx xxxxx xxxxx
1305   48,     YN,  F=-x,       PZE;       "110 011 xx xx xx  xxxxx xxxxx xxxxx
1306   48.1,   YN,  F=-:DYN,    PZE;       "111 011 01 xx xx  xxxxx xxxxx xxxxx
1307   49,     YN,  F+x,        PZE;:DYN;"110 000 xx xx xx  xxxxx xxxxx xxxxx
1308   50,     YN,  F-x,        PZE;:DYN;"110 001 xx xx xx  xxxxx xxxxx xxxxx
1309   51,     YN,  F×x,        PZE;:DYN;"111 000 xx xx xx  xxxxx xxxxx xxxxx
1310   52,     YN,  F/x,        PZE;:DYN;"111 001 xx xx xx  xxxxx xxxxx xxxxx
1311   53,     YN,  x=F,        PZE;       "111 010 xx xx xx  xxxxx xxxxx xxxxx
1312   54,     YN,  x=-F,       PZE;       "111 011 xx xx xx  xxxxx xxxxx xxxxx
1313
1314   55,          G=x,        PZE;       "110 010 xx xx 01  xxxxx xxxxx xxxxx
1315   047,    YN,  G=:STAT,    PZE;       "110 010 01 xx xx  xxxxx xxxxx xxxxx
1316   047.1,  YN,  G=:DYN,     PZE;       "111 010 01 xx xx  xxxxx xxxxx xxxxx
1317   56,          G=-x,       PZE;       "110 011 xx xx 01  xxxxx xxxxx xxxxx
1318   048,    YN,  G=-:STAT,   PZE;       "110 011 01 xx xx  xxxxx xxxxx xxxxx
1319   048.1,  YN,  G=-:DYN,    PZE;       "111 011 01 xx xx  xxxxx xxxxx xxxxx
1320   57,          G+x,        PZE;:DYN;"110 000 xx xx 01  xxxxx xxxxx xxxxx
1321   049,    YN,  G+:STAT,    PZE;       "110 000 01 xx xx  xxxxx xxxxx xxxxx
1322   58,          G-x,        PZE;:DYN;"110 001 xx xx 01  xxxxx xxxxx xxxxx
1323   050,    YN,  G-:STAT,    PZE;       "110 001 01 xx xx  xxxxx xxxxx xxxxx
1324   59,          G×x,        PZE;:DYN;"111 000 xx xx 01  xxxxx xxxxx xxxxx
```

```
1325   051,   YN,  GX:STAT,  PZE;      "111 000 01 xx xx  xxxxx xxxxx xxxxx
1326   60,          G/x,      PZE;:DYN;"111 001 xx xx 01  xxxxx xxxxx xxxxx
1327   052,   YN,  G/:STAT,  PZE;      "111 001 01 xx xx  xxxxx xxxxx xxxxx
1328   61,          x=G,      PZE;      "111 010 xx xx 01  xxxxx xxxxx xxxxx
1329   62,          x=-G,     PZE;      "111 011 xx xx 01  xxxxx xxxxx xxxxx
1330
1331   63,    UYN, GOTO(x);              "101 010 xx 00 xx  xxxxx xxxxx xxxxx
1332   63.1,  UYN, GOTO(:DYN);           "101 011 11 00 xx  xxxxx xxxxx xxxxx
1333   64,    UYN, JUMP(x);  :DYN;       "101 000 xx 00 xx  xxxxx xxxxx xxxxx
1334   65,    UYN, JUMP(-x); :DYN;       "101 001 xx 00 xx  xxxxx xxxxx xxxxx
1335   66,    UYN, GOTOR(x);             "101 010 xx 01 xx  xxxxx xxxxx xxxxx
1336   66.1,  UYN, GOTOR(:DYN);          "101 011 11 01 xx  xxxxx xxxxx xxxxx
1337   67,    UYN, JUMPR(x); :DYN;       "101 000 xx 01 xx  xxxxx xxxxx xxxxx
1338   68,    UYN, JUMPR(-x);:DYN;       "101 001 xx 01 xx  xxxxx xxxxx xxxxx
1339
1340   69,    UYN, REP(:STAT);           "101 101 11 00 xx  xxxxx xxxxx xxxxx
1341   70,    UYN, REPP(:STAT);          "101 101 11 01 xx  xxxxx xxxxx xxxxx
1342   71,    UYN, REPE(:STAT);          "101 101 11 11 xx  xxxxx xxxxx xxxxx
1343   72,    UYN, REPZ(:STAT);          "101 101 11 10 xx  xxxxx xxxxx xxxxx
1344
1345   73,    UYN, SUB(:STAT);           "101 110 00 00 xx  xxxxx xxxxx xxxxx
1346   74,    UYN, SUBC(x);              "101 110 xx 01 xx  xxxxx xxxxx xxxxx
1347   74.1,  UYN, SUBC(:DYN);           "101 111 01 01 xx  xxxxx xxxxx xxxxx
1348
1349   76,    UYN, DO(x);  :STAT,:DYN;"101 111 xx 01 xx  xxxxx xxxxx xxxxx
1350   77,    UYN, DOS(x); :STAT,:DYN;"101 111 xx 11 xx  xxxxx xxxxx xxxxx
1351
1352   78,    YN,  LCA(1),   PZE;      "110 110 x0 xx xx  00000 00000 xxxxx
1353   79,    YN,  LCS(1),   PZE;      "111 110 x0 xx xx  00000 00000 xxxxx
1354   80,    YN,  LCAS(1),  PZE;      "110 110 x0 xx xx  00000 00010 xxxxx
1355   81,    YN,  LCSA(1),  PZE;      "111 110 x0 xx xx  00000 00010 xxxxx
1356   82,    YN,  RCA(1),   PZE;      "110 111 x0 xx xx  00000 00000 xxxxx
1357   83,    YN,  RCS(1),   PZE;      "111 111 x0 xx xx  00000 00000 xxxxx
1358   84,    YN,  RCAS(1),  PZE;      "110 111 x0 xx xx  00000 00010 xxxxx
1359   85,    YN,  RCSA(1),  PZE;      "111 111 x0 xx xx  00000 00010 xxxxx
1360
1361   86,    YN,  LUA(1),   PZE;      "110 110 x0 xx xx  00000 00001 xxxxx
1362   87,    YN,  LUS(1),   PZE;      "111 110 x0 xx xx  00000 00001 xxxxx
1363   88,    YN,  LUAS(1),  PZE;      "110 110 x0 xx xx  00000 00011 xxxxx
1364   89,    YN,  LUSA(1),  PZE;      "111 110 x0 xx xx  00000 00011 xxxxx
1365   90,    YN,  RUA(1),   PZE;      "110 111 x0 xx xx  00000 00001 xxxxx
1366   91,    YN,  RUS(1),   PZE;      "111 111 x0 xx xx  00000 00001 xxxxx
1367   92,    YN,  RUAS(1),  PZE;      "110 111 x0 xx xx  00000 00011 xxxxx
1368   93,    YN,  RUSA(1),  PZE;      "111 111 x0 xx xx  00000 00011 xxxxx
1369
1370   94,    UYN, NORA,     PZE;      "110 110 00 xx xx  00000 00101 00000
1371   95,    UYN, NORS,     PZE;      "111 110 00 xx xx  00000 00101 00000
1372   96,    YN,  NORAS,    PZE;      "110 110 00 xx xx  00000 00111 00000
1373
1374   97,    YN,  TENS,     PZE;      "111 110 00 xx xx  00001 00000 00001
1375   98,    YN,  TENAS,    PZE;      "111 110 00 xx xx  00001 00000 00000
```

```
1376
1377   99,     YN,  CLP;              "110 110 00 00 xx  11001 10000 00000
1378   100,    UYN, INT;              "110 110 00 00 xx  11010 00000 00000
1379
1380   101.0, UYN, REP0(:STAT);       "101 100 00 00 xx  xxxxx xxxxx xxxxx
1381   101.1, UYN, REP1(:STAT);       "101 100 01 00 xx  xxxxx xxxxx xxxxx
1382   101.2, UYN, REP2(:STAT);       "101 100 10 00 xx  xxxxx xxxxx xxxxx
1383   101.3, UYN, REP3(:STAT);       "101 100 11 00 xx  xxxxx xxxxx xxxxx
1384   101.4, UYN, REP4(:STAT);       "101 101 00 00 xx  xxxxx xxxxx xxxxx
1385   101.5, UYN, REP5(:STAT);       "101 101 01 00 xx  xxxxx xxxxx xxxxx
1386   101.6, UYN, REP6(:STAT);       "101 101 10 00 xx  xxxxx xxxxx xxxxx
1387   101.7, UYN, REP7(:STAT);       "101 101 11 00 xx  xxxxx xxxxx xxxxx
1388
1389   102.0, UYN, REP0P(:STAT);      "101 100 00 01 xx  xxxxx xxxxx xxxxx
1390   102.1, UYN, REP1P(:STAT);      "101 100 01 01 xx  xxxxx xxxxx xxxxx
1391   102.2, UYN, REP2P(:STAT);      "101 100 10 01 xx  xxxxx xxxxx xxxxx
1392   102.3, UYN, REP3P(:STAT);      "101 100 11 01 xx  xxxxx xxxxx xxxxx
1393   102.4, UYN, REP4P(:STAT);      "101 101 00 01 xx  xxxxx xxxxx xxxxx
1394   102.5, UYN, REP5P(:STAT);      "101 101 01 01 xx  xxxxx xxxxx xxxxx
1395   102.6, UYN, REP6P(:STAT);      "101 101 10 01 xx  xxxxx xxxxx xxxxx
1396   102.7, UYN, REP7P(:STAT);      "101 101 11 01 xx  xxxxx xxxxx xxxxx
1397
1398   103.0, UYN, REP0Z(:STAT);      "101 100 00 10 xx  xxxxx xxxxx xxxxx
1399   103.1, UYN, REP1Z(:STAT);      "101 100 01 10 xx  xxxxx xxxxx xxxxx
1400   103.2, UYN, REP2Z(:STAT);      "101 100 10 10 xx  xxxxx xxxxx xxxxx
1401   103.3, UYN, REP3Z(:STAT);      "101 100 11 10 xx  xxxxx xxxxx xxxxx
1402   103.4, UYN, REP4Z(:STAT);      "101 101 00 10 xx  xxxxx xxxxx xxxxx
1403   103.5, UYN, REP5Z(:STAT);      "101 101 01 10 xx  xxxxx xxxxx xxxxx
1404   103.6, UYN, REP6Z(:STAT);      "101 101 10 10 xx  xxxxx xxxxx xxxxx
1405   103.7, UYN, REP7Z(:STAT);      "101 101 11 10 xx  xxxxx xxxxx xxxxx
1406
1407   104.0, UYN, REP0E(:STAT);      "101 100 00 11 xx  xxxxx xxxxx xxxxx
1408   104.1, UYN, REP1E(:STAT);      "101 100 01 11 xx  xxxxx xxxxx xxxxx
1409   104.2, UYN, REP2E(:STAT);      "101 100 10 11 xx  xxxxx xxxxx xxxxx
1410   104.3, UYN, REP3E(:STAT);      "101 100 11 11 xx  xxxxx xxxxx xxxxx
1411   104.4, UYN, REP4E(:STAT);      "101 101 00 11 xx  xxxxx xxxxx xxxxx
1412   104.5, UYN, REP5E(:STAT);      "101 101 01 11 xx  xxxxx xxxxx xxxxx
1413   104.6, UYN, REP6E(:STAT);      "101 101 10 11 xx  xxxxx xxxxx xxxxx
1414   104.7, UYN, REP7E(:STAT);      "101 101 11 11 xx  xxxxx xxxxx xxxxx
1415
1416   105. 0,UYN, SUB0(:STAT);       "101 110 00 00 xx  xxxxx xxxxx xxxxx
1417   105. 1,UYN, SUB1(:STAT);       "101 110 01 00 xx  xxxxx xxxxx xxxxx
1418   105. 2,UYN, SUB2(:STAT);       "101 110 10 00 xx  xxxxx xxxxx xxxxx
1419   105. 3,UYN, SUB3(:STAT);       "101 110 11 00 xx  xxxxx xxxxx xxxxx
1420   105. 4,UYN, SUB4(:STAT);       "101 111 00 00 xx  xxxxx xxxxx xxxxx
1421   105. 5,UYN, SUB5(:STAT);       "101 111 01 00 xx  xxxxx xxxxx xxxxx
1422   105. 6,UYN, SUB6(:STAT);       "101 111 10 00 xx  xxxxx xxxxx xxxxx
1423   105. 7,UYN, SUB7(:STAT);       "101 111 11 00 xx  xxxxx xxxxx xxxxx
1424   105. 8,UYN, SUB8(:STAT);       "101 110 00 10 xx  xxxxx xxxxx xxxxx
1425   105. 9,UYN, SUB9(:STAT);       "101 110 01 10 xx  xxxxx xxxxx xxxxx
1426   105.10,UYN, SUB10(:STAT);      "101 110 10 10 xx  xxxxx xxxxx xxxxx
```

```
1427   105.11,UYN, SUB11(:STAT);       "101 110 11 10 xx  xxxxx xxxxx xxxxx
1428   105.12,UYN, SUB12(:STAT);       "101 111 00 10 xx  xxxxx xxxxx xxxxx
1429   105.13,UYN, SUB13(:STAT);       "101 111 01 10 xx  xxxxx xxxxx xxxxx
1430   105.14,UYN, SUB14(:STAT);       "101 111 10 10 xx  xxxxx xxxxx xxxxx
1431   105.15,UYN, SUB15(:STAT);       "101 111 11 10 xx  xxxxx xxxxx xxxxx
1432
1433   106,   UYN, SUBCD(x);           "101 110 xx 11 xx  xxxxx xxxxx xxxxx
1434   106.1, UYN, SUBCD(:DYN);        "101 111 01 11 xx  xxxxx xxxxx xxxxx
1435
1436   107,   UYN, LVIFA(1), PZE;      "110 110 x0 xx xx  11111 00000 0000x
1437   108,   UYN, LVIFAC(1),PZE;      "110 110 x0 xx xx  11111 00010 0000x
1438   109,   UYN, IFA(1),   PZE;      "110 110 x0 xx xx  11110 10000 0000x
1439   110,   UYN, IFAC(1),  PZE;      "110 110 x0 xx xx  11110 10010 0000x
1440   111,   UYN, LVIFS(1), PZE;      "111 110 x0 xx xx  11111 00000 0000x
1441   112,   UYN, LVIFSC(1),PZE;      "111 110 x0 xx xx  11111 00010 0000x
1442   113,   UYN, IFS(1),   PZE;      "111 110 x0 xx xx  11110 10000 0000x
1443   114,   UYN, IFSC(1),  PZE;      "111 110 x0 xx xx  11110 10010 0000x
1444
1445   115,   YN,  LVIFON(1);          "111 110 x0 00 xx  11101 0000x xxxxx
1446   116,   YN,  LVIFOFF(1);         "110 110 x0 00 xx  11101 0000x xxxxx
1447   117,   YN,  IFON(1);            "111 110 x0 00 xx  11100 1000x xxxxx
1448   118,   YN,  IFOFF(1);           "110 110 x0 00 xx  11100 1000x xxxxx
1449   119,   YN,  AFON(1);            "111 110 x0 00 xx  11100 0000x xxxxx
1450   120,   YN,  AFOFF(1);           "110 110 x0 00 xx  11100 0000x xxxxx
1451   121,   YN,  IVON;               "111 110 00 00 xx  11000 00000 00000
1452   122,   YN,  IVOFF;              "110 110 00 00 xx  11000 00000 00000
1453   123,   YN,  OVON;               "111 110 00 00 xx  11000 10000 00000
1454   124,   YN,  OVOFF;              "110 110 00 00 xx  11000 10000 00000
1455   125,   YN,  ITVON;              "111 110 00 00 xx  11001 00000 00000
1456   126,   YN,  MEMPROT;            "111 110 11 00 xx  11110 11000 00000
1457
1458   127,   UYN, TRAA,     PZE;      "110 110 00 xx xx  00000 01000 00000
1459   128,   UYN, TRAS,     PZE;      "110 110 00 xx xx  00000 01001 00000
1460   129,   UYN, TRAB,     PZE;      "110 110 00 xx xx  00000 01010 00000
1461   130,   UYN, TRSA,     PZE;      "110 110 00 xx xx  00000 01000 00001
1462   131,   UYN, TRSS,     PZE;      "110 110 00 xx xx  00000 01001 00001
1463   132,   UYN, TRSB,     PZE;      "110 110 00 xx xx  00000 01010 00001
1464   133,   UYN, TRBA,     PZE;      "110 110 00 xx xx  00000 01000 00010
1465   134,   UYN, TRBS,     PZE;      "110 110 00 xx xx  00000 01001 00010
1466   135,   UYN, TRBB,     PZE;      "110 110 00 xx xx  00000 01010 00010
1467
1468   136,   UYN, TRANA,    PZE;      "110 111 00 xx xx  00000 01000 00000
1469   137,   UYN, TRANS,    PZE;      "110 111 00 xx xx  00000 01001 00000
1470   138,   UYN, TRANB,    PZE;      "110 111 00 xx xx  00000 01010 00000
1471   139,   UYN, TRSNA,    PZE;      "110 111 00 xx xx  00000 01000 00001
1472   140,   UYN, TRSNS,    PZE;      "110 111 00 xx xx  00000 01001 00001
1473   141,   UYN, TRSNB,    PZE;      "110 111 00 xx xx  00000 01010 00001
1474   142,   UYN, TRBNA,    PZE;      "110 111 00 xx xx  00000 01000 00010
1475   143,   UYN, TRBNS,    PZE;      "110 111 00 xx xx  00000 01001 00010
1476   144,   UYN, TRBNB,    PZE;      "110 111 00 xx xx  00000 01010 00010
1477   'end'
```

```
1478
1479  'begin'          count0 = M[ 0], count1 = M[ 1], count2 = M[ 2],
1480    count3 = M[ 3], count4 = M[ 4], count5 = M[ 5], count6 = M[ 6],
1481    count7 = M[ 7], count  = count7,
1482    link0  = M[ 8], link1  = M[ 9], link2  = M[10], link3  = M[11],
1483    link4  = M[12], link5  = M[13], link6  = M[14], link7  = M[15],
1484    link8  = M[16], link9  = M[17], link10 = M[18], link11 = M[19],
1485    link12 = M[20], link13 = M[21], link14 = M[22], link15 = M[23],
1486    link   = link0,
1487    D = M[63];});
1488
1489      stringsymcount:= stringsymcount + 1
1490    end Init stringsym;
1491
1492    comment:
1493
1494    11.  Code produce equipment.
1495
1496    Algorithm:;
1497
1498    procedure PRODUCE INSTR CODE;
1499    begin integer cond,cond for UYN,cond for PZE,cond for minus,
1500      function part,X8 word;
1501      cond:= INF WORD - INF WORD : t5 × t5;
1502      cond:= cond : t1; function part:= INF WORD - cond × t1;
1503      cond for UYN:= cond : 4;
1504      cond:= cond - cond for UYN × 4;
1505      cond for PZE:= cond : t1;
1506      cond for minus:= cond - cond for PZE × t1;
1507      ERROR(INF WORD = 1,1000);
1508      if UYN > 0 then
1509      ERROR(cond for UYN=0 ∨ cond for UYN=1 ∧ UYN=U symbol,1010);
1510      if PZE > 0 then ERROR(cond for PZE = 0,1020);
1511      if minus for right operand then ERROR(cond for minus = 0,1030);
1512      if ¬ (0 < value of operand ∧ value of operand < t15) then
1513      begin ERROR (true, 1040); value of operand:= 0 end;
1514      X8 word:= abs(
1515                function part +
1516                (if minus for right operand then t21 else 0) +
1517                (if UYN = U symbol then 1 else
1518                 if UYN = Y symbol then 2 else
1519                 if UYN = N symbol then 3 else 0) × t15 +
1520                (if PZE = P symbol then 1 else
1521                 if PZE = Z symbol then 2 else
1522                 if PZE = E symbol then 3 else 0) × t17 +
1523                 value of operand);
1524      PR binary number(d26 INF WORD,X8 word)
1525    end PRODUCE INSTR CODE;
1526
1527    procedure PRODUCE NUMBER CODE;
1528    if type of number=integral type ∨ type of number=octal type then
```

```
1529    PR integral number(real to int(value of number)) else
1530    PR real number(value of real number);
1531
1532    procedure PRODUCE EXPR CODE(expr);
1533    PR integral number(real to int(expr));
1534
1535    integer procedure real to int (r); value r; real r;
1536    begin integer sgn, i;
1537      real q;
1538      sgn:= sign (1 / r); r:= abs (r); q:= entier (r / real t26);
1539      ERROR (q ≠ 0, 1045); i:= abs (r - q × real t26);
1540      real to int:=(if entier(q/t1)×t1=q then i else i+t26)×sgn
1541    end real to int;
1542
1543    procedure PR binary number(d26,w);
1544    value d26,w; boolean d26; integer w;
1545    if second scan then
1546    PR integral number (if d26 then t26 + w else w);
1547
1548    procedure PR integral number(i); value i; integer i;
1549    if second scan then
1550    begin cntr:= cntr + 1; words[cntr]:= i;
1551      if cntr = 1 then address old:= address counter;
1552      if cntr = 511 then PUNCH;
1553     PRINT octal(9,i,false)
1554    end PR integral number;
1555
1556    procedure PR real number(r); value r; real r;
1557    begin real array R[1:1];
1558      integer array I[1:2];
1559      R[1]:= r; TO DRUM(R,153598); FROM DRUM(I,153598);
1560      PR integral number(I[1]); Add to ad cntr(1); PRINT LINE;
1561      PR integral number(I[2])
1562    end PR real number;
1563
1564    procedure PRINT octal(n,x,punch);
1565    value n,x; integer n,x; boolean punch;
1566    begin boolean neg;
1567      integer p8, digit, i;
1568      neg:= 1 / x < 0; if neg then ERROR (n < 9, 1060); x:= abs (x);
1569      p8:= 8 ↑ (n - 1); if x ≥ p8 × 8 then
1570      begin ERROR (true, 1065); x:= x - x : (p8 × 8) × (p8 × 8) end;
1571      if first scan = second scan then
1572      begin PRSYM(apostrophesymbol);
1573        if punch then PUSYM(apostrophesymbol)
1574      end;
1575      if n= 6 then i:= 2 else if n = 9 then i:= 13;
1576      for p8:= p8, p8 : 8 while p8 ≠ 0 do
1577      begin digit:= x : p8; x:= x - digit × p8;
1578        digit:= if neg then 7 - digit else digit;
1579        if first scan = second scan then
```

```
1580          begin PRSYM(digit); if punch then PUSYM(digit) end else
1581          begin line buffer[i]:= digit; i:= i+1 end;
1582        end;
1583        if first scan = second scan then
1584        begin PRSYM(apostrophesymbol);
1585          if punch then PUSYM(apostrophesymbol)
1586        end;
1587      end PRINT octal;
1588
1589      integer cntr, address old;
1590      boolean IP;
1591      integer array words[1:511];
1592
1593      procedure PUNCH;
1594      if IP then punch IP else punch BI;
1595
1596      procedure single IP (word); value word; integer word;
1597      begin integer q, hep;
1598        if 1 / word > 0 then q:= t6 else
1599        begin q:= t6 + t5; word:= word - t26 end;
1600        hep:= word : t21; word:= word - hep × t21; PUHEP (q + hep);
1601        hep:= word : t14; word:= word - hep × t14; PUHEP (    hep);
1602        hep:= word :  t7; PUHEP (hep); PUHEP (word - hep × t7)
1603      end single IP;
1604
1605      procedure punch IP;
1606      if ¬initialization ∧ cntr≠0 then
1607      begin integer i, cntr1;
1608        for i:= 1 step 1 until 10 do PUHEP(0);
1609        cntr1:= if address old= 0 then cntr+1 else cntr;
1610        if cntr ≥ t8 then
1611          single IP((cntr1 - t8) × t18 + address old + t26 - 1) else
1612          single IP(cntr1 × t18 + address old - 1);
1613        for i:=1 step 1 until cntr do single IP(words[i]);
1614        cntr:=0;
1615      end punch IP;
1616
1617      procedure single BI(word, c1, c2);
1618      value word, c1, c2; integer word; boolean c1, c2;
1619      begin integer i, p, q, odd;
1620        integer array hep[1:5];
1621        p:= q:= word; odd:= if q≠ 0 then sign(q) else sign(1/q);
1622        if odd= -1 then
1623        begin hep[5]:= 4; p:= q:= q-t26 end else hep[5]:= 0;
1624        for i:= 1 step 1 until 26 do
1625        begin odd:= EVEN(q)×odd; q:= q: 2 end;
1626        q:= p;
1627        for i:= 1 step 1 until 4 do
1628        begin q:= q: 64; hep[i]:= p-q× 64; p:= q end;
1629        hep[5]:= hep[5]+p+(if odd = 1 then 32 else 0)+
1630          (if c1 then 16 else 0)+(if c2 then 8 else 0);
```

```
1631        for i:= 5 step -1 until 1 do PUHEP(hep[i])
1632      end single BI;
1633
1634      procedure punch BI;
1635      if ¬ initialization ∧ cntr ≠ 0 then
1636      begin integer i, cntr1;
1637        for i:= 1 step 1 until 10 do PUHEP(0);
1638        cntr1:= if address old= 0 then cntr+1 else cntr;
1639        if cntr> t8 then
1640          single BI((cntr1-t8)×t18+address old+t26-1,true,true) else
1641          single BI(cntr1 × t18 + address old - 1, true, true);
1642        for i:= 1 step 1 until cntr do single BI(words[i],false,false);
1643        cntr:= 0;
1644      end punch BI;
1645
1646      procedure Add to ad cntr(i); value i; integer i;
1647      begin if address counter<0 ∨ address counter>sixty four K then
1648        begin fixt(12,0,address counter);
1649          address counter:=0; ERROR( true ,2010)
1650        end;
1651        if i≠1 ∧ i≠0 then PUNCH; PR ad cntr;
1652        address counter:= address counter + i
1653      end Add to ad cntr;
1654
1655      comment:
1656
1657      12.  Some Boolean procedures.
1658
1659      Algorithm:;
1660
1661      boolean procedure is adding operator(s); value s; integer s;
1662      is adding operator:= s = plus symbol ∨ s = minus symbol;
1663
1664      boolean procedure is PZE symbol(s); value s; integer s;
1665      is PZE symbol:= s = P symbol ∨ s = Z symbol ∨
1666                      s = E symbol ∨ s = PZE symbol;
1667
1668      boolean procedure is UYN symbol(s); value s; integer s;
1669      is UYN symbol:= s = U symbol ∨ s = Y symbol ∨ s = N symbol ∨
1670                      s = YN symbol ∨ s = UYN symbol;
1671
1672      boolean procedure is register symbol(s); value s; integer s;
1673      begin
1674        procedure A(r,i,v); value r,i,v; integer r,i,v;
1675        if s = r then
1676        begin is register symbol:= true; index of register:= i;
1677          value of register:= v; goto end
1678        end A;
1679
1680        is register symbol:= false;
1681        A(F symbol,1,57); A(G symbol,2,58); A(A symbol,3,59);
```

```
1682      A(S symbol,4,60); A(B symbol,5,61);
1683    end:
1684    end;
1685
1686    boolean procedure is operator symbol(s); value s; integer s;
1687    is operator symbol:= s = plus symbol   ∨ s = minus symbol ∨
1688                         s = times symbol  ∨ s = over symbol  ∨
1689                         s = equals symbol ∨ s = logic plus symbol ∨
1690                         s = logic times symbol;
1691
1692    boolean procedure is letter(s); value s; integer s;
1693    is letter:= 10 ≤ s ∧ s ≤ 35;
1694
1695    boolean procedure is digit(s); value s; integer s;
1696    is digit:= 0 ≤ s ∧ s ≤ 9;
1697
1698    boolean procedure is layout(s); value s; integer s;
1699    is layout:= s = space symbol ∨ s = tab symbol;
1700
1701    boolean procedure is stat sep(s); value s; integer s;
1702    is stat sep:= s = nlcr symbol ∨ s = semicolon symbol ∨
1703                                     s = quote symbol;
1704
1705    integer procedure index of operator(s); value s; integer s;
1706    index of operator:= if s = plus   symbol then 1 else
1707                        if s = times  symbol then 2 else
1708                        if s = over   symbol then 3 else
1709                        if s = equals symbol then 4 else
1710                        if s = logic plus  symbol then 5 else
1711                        if s = logic times symbol then 6 else
1712                        ERROR( true ,2020);
1713
1714    comment:
1715
1716    13.  Treatment of Errors.
1717
1718    Algorithm:;
1719
1720    boolean CHECK fault;
1721
1722    integer procedure ERROR(B,n); value B,n; Boolean B; integer n;
1723    if B then
1724    begin PRINT LINE;
1725      PRINTTEXT(‡error nr :‡); ABSFIXT(4,0,n);
1726      PR synt unit ( synt unit ); ERROR:=1;
1727      if first scan then
1728      begin SPACE(50 - printpos); ABSFIXT(6,0,line number) end;
1729      PRSYM(nlcr symbol)
1730    end ERROR;
1731
1732    integer procedure CHECK(s,n); value s,n; integer s,n;
```

```
1733    if synt unit = s then
1734    begin CHECK:= 1; CHECK fault:= false end else
1735    begin CHECK:= ERROR (true, n); PRSYM(nlcr symbol);
1736      PRINTTEXT (‹synt unit should be: ›); PR synt unit (s);
1737      PRSYM(nlcr symbol);
1738      CHECK fault:= true
1739    end CHECK;
1740
1741    integer procedure REQUIRE (s, n); value s, n; integer s, n;
1742    begin REQUIRE:= CHECK (s, n); if CHECK fault then
1743    L: begin RE;
1744         if synt unit = s then else
1745         if synt unit = statement separator ∨
1746            synt unit = end symbol then
1747         CHECK fault:= true else goto L
1748       end
1749    end REQUIRE;
1750
1751    procedure PR synt unit (s); value s; integer s;
1752    if s ≤ 127 then PRSYM (s) else
1753    begin
1754      procedure a(p,r); value p; integer p; string r;
1755      if s = p then
1756      begin PRINTTEXT (r); goto exit end a;
1757
1758      a(begin symbol,‹'begin'›); a(end symbol,‹'end'›);
1759      a(SKIP symbol,‹'skip'›); a(DYN M symbol,‹<dyn m symbol>›);
1760      a(logic plus symbol,‹'+'›); a(logic times symbol,‹'×'›);
1761      a(statement separator,‹<statement separator>›);
1762      a(MT declaration symbol,‹'mt'›); a(identifier,‹<identifier>›);
1763      a(number,‹<number>›);
1764      a(function identifier,‹<function identifier>›);
1765      PRINT (s); exit:
1766    end PR synt unit;
1767
1768    comment:
1769
1770    14.  Declaration and initialization of symbols.
1771
1772    Algorithm:;
1773
1774    integer space symbol,tab symbol,nlcr symbol,comma symbol,
1775      G symbol,A symbol,S symbol,C symbol,T symbol,D symbol,B symbol,
1776      M symbol,pr sub symbol,pr bus symbol,times symbol,over symbol,
1777      plus symbol,minus symbol,equals symbol,F symbol,E symbol,
1778      N symbol,P symbol,Y symbol,U symbol,Z symbol,point symbol,
1779      lower ten symbol,colon symbol,apostrophe symbol,quote symbol,
1780      semicolon symbol,open symbol,sub symbol,close symbol,
1781      bus symbol,letter b,letter e,letter i,letter S,letter M,
1782      begin symbol,end symbol,BI symbol,IP symbol,SKIP symbol,
1783      DYN M symbol,type of DYN M symbol,UYN symbol,YN symbol,
```

```
1784      PZE symbol,logic plus symbol,logic times symbol,
1785      statement separator,MT declaration symbol,identifier,number,
1786      function identifier,synt unit,underline,bar;
1787
1788    procedure INITIALIZE RESYM;
1789    begin integer nlcr, tab, not, question mark, symcount, hep;
1790      integer procedure stringsym;
1791      begin integer sym;
1792      L: sym:= STRINGSYMBOL (symcount, ┤?_|
1793          ┐12?4??78??┐??_? ??3?56??9 ?????0??t?vw??z┐
1794          ??????<s?u??xy??₁₀?? ?—??l?no??r??????jk?m?
1795          ?pq??,?????ab?d??gh??.????+??c?ef??i┐?┐???
1796
1797          ┐\×?=??](???┐??|? ??/?;[??) ?????∧??T?VW??Z┐
1798          ??????>S?U??XY??'?? ?┐??L?NO??R??????JK?M?
1799          ?PQ?????????AB?D??GH??:????"??C?EF??I┐?┐???┤);
1800        symcount:= symcount + 1;
1801        if sym = nlcr ∨ sym = tab then goto L;
1802        if sym = underline ∨ sym = bar then symcount:= symcount + 1;
1803        stringsym:= if sym = not then dummy code else
1804        if sym = question mark then error code else sym
1805      end stringsym;
1806
1807      not:= question mark:= underline:= bar:= nlcr:= tab:= -1;
1808      dummy code:= -1; error code:= -2; symcount:= 0;
1809      not:=stringsym; question mark:=stringsym; underline:=stringsym;
1810      stringsym; bar:= stringsym; nlcr:= stringsym; tab:= stringsym;
1811      lower case:= 122; upper case:= 124;
1812      lower case code:= 0; upper case code:= 128;
1813      for case code:= lower case code, upper case code do
1814      for hep:= 0 step 1 until 127 do
1815      symcode[case code + hep]:= stringsym;
1816      symcode[lower case code+26]:=symcode[upper case code+26]:=nlcr;
1817      symcode[lower case code+62]:=symcode[upper case code+62]:=tab;
1818      symcode[upper case code+64]:=not;
1819      symcode[upper case code+91]:=question mark;
1820      case code:= lower case code
1821    end INITIALIZE RESYM;
1822
1823    procedure INITIALIZE symbols;
1824    begin integer i;
1825      procedure R(s); integer s;
1826      begin L: s:= RESYM1; if s = space symbol ∨ s = tab symbol ∨
1827        s = nlcr symbol ∨ s = comma symbol then goto L
1828        else if 37 ≤ s ∧ s ≤ 62 then s:= s - 27
1829      end R;
1830
1831      procedure P(s); integer s;
1832      s:= i:= i + 1;
1833
```

```
1834        INITIALIZE RESYM;
1835        space symbol:=RESYM1; tab symbol:=RESYM1; comma symbol:=RESYM1;
1836        nlcr symbol:= RESYM1;
1837        R(G symbol);R(A symbol); R(S symbol); R(C symbol); R(T symbol);
1838        R(D symbol);R(B symbol); R(M symbol); R(pr sub symbol);
1839        R(pr bus symbol);R(times symbol);R(over symbol);R(plus symbol);
1840        R(minus symbol); R(equals symbol); R(F symbol); R(E symbol);
1841        R(N symbol); R(P symbol); R(Y symbol); R(U symbol);R(Z symbol);
1842        R(point symbol);R(lower ten symbol); R(colon symbol);
1843        R(apostrophe symbol); R(quote symbol); R(semicolon symbol);
1844        R(open symbol); sub symbol:= open symbol; R(close symbol);
1845        bus symbol:= close symbol; R(letter b); R(letter e);
1846        R(letter i); R(letter S); R(letter M);
1847        i:= 127;
1848        P(begin symbol);P(end symbol); P(SKIP symbol); P(DYN M symbol);
1849        P(UYN symbol);P(YN symbol);P(PZE symbol);P(logic plus symbol);
1850        P(logic times symbol); P(statement separator);
1851        P(MT declaration symbol);P(identifier); P(number);
1852        P(function identifier);P(BI symbol); P(IP symbol)
1853      end INITIALIZE symbols;
1854
1855      comment:
1856
1857      15.  Other auxiliary equipment.
1858
1859      Algorithm:;
1860
1861      integer array block[0:50], ELAN line[1:200], line buffer[-1:200];
1862      boolean first scan,second scan,initialization,declaration,
1863        minus for right operand,new identifier,
1864        not behind last end, punch list;
1865      integer declared,cntr of begins,block number,nr of begins,
1866        sixty four K,MT declared and defined,MT declared,
1867        type of number,real type,integral type,octal type,UYN,PZE,
1868        index of register,value of register,right operand,
1869        left operand,operator,register,fctn,type of identifier,
1870        value of identifier,type of function identifier,
1871        number of functions,t0,t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,
1872        t13,t14,t15,t16,t17,t18,t19,t20,t21,t22,t23,t24,t25,t26,
1873        symbol,next symbol,reading ptr,ptr of text,max of ELAN line,
1874        line number,max of num,max block number,line counter,
1875        fill pointer;
1876      real value of operand,value of number,value of real number,
1877        address counter,real t26;
1878
1879      procedure INITIALIZE other variables;
1880      begin real ti;
1881        real procedure d;
1882        begin d:= ti; ti:= ti × 2 end d;
1883
1884        declared:= 10; MT declared:= 11; MT declared and defined:= 12;
```

```
1885      real type:= 1; integral type:= octal type:= 2;
1886      ti :=1; t0 :=d; t1 :=d; t2 :=d; t3 :=d; t4 :=d; t5 :=d; t6 :=d;
1887      t7 :=d; t8 :=d; t9 :=d; t10:=d; t11:=d; t12:=d; t13:=d; t14:=d;
1888      t15:=d; t16:=d; t17:=d; t18:=d; t19:=d; t20:=d; t21:=d; t22:=d;
1889      t23:=d; t24:=d; t25:=d;
1890      real t26:= d; t26:= 1 - real t26;
1891      sixty four K:= t16; max of ELAN line:= 200;
1892      max of buffer:= min of drum:= t12;
1893      max of drum:= 37.5 × max of buffer - 1;
1894      max of inf list:= 22 × t10;
1895      pointer of ptr of inf list:=ptr of inf list:=1;max of num:=0;
1896      end fctn part:= 10 000; contents of[ptr of inf list]:= 0;
1897      line pointer:= drum pointer:= 0;
1898      place enough:= IP:= true;
1899
1900      line buffer[ 1]:= line buffer[ 8]:= line buffer[12]:=
1901      line buffer[22]:= apostrophesymbol;
1902      line buffer[ 9]:= colon symbol;
1903      line buffer[10]:= line buffer[11]:= line buffer[23]:=
1904      line buffer[24]:= space symbol;
1905
1906      begin integer p,m,n,A,i,s;
1907        boolean end;
1908      L0: i:= 0; declaration:=true;
1909      L1: s:= RESYM1; if 37 ≤ s ∧ s ≤ 62 then s:= s - 27;
1910        if ¬ is letter(s) then goto L1;
1911        n:= A:= 0; p:= ptr of inf list; i:= i + 1;
1912      L2: n:= n + 1; A:= A × t6 + s + 1; s:= RESYM1;
1913        if 37 ≤ s ∧ s ≤ 62 then s:= s - 27;
1914        end:= ¬(is letter(s) ∨ is digit(s));
1915        if n = n ÷ 4 × 4 ∨ end then
1916        begin p:= p + 1; ERROR(p > max of inf list,2000);
1917          contents of[p]:= A; A:= 0
1918        end;
1919        if ¬ end then goto L2;
1920        m:= p - ptr of inf list; contents of[ptr of inf list]:= m;
1921        contents of[p]:= contents of[p] × t6 ↑ (m × 4 - n);
1922        A:=if s = open  symbol then t8 else
1923           if s = close symbol then t9 else
1924           if s = over  symbol then t10 else 0;
1925        if ptr of inf list = 1 then
1926        begin
1927          pointer of ptr of inf list:=
1928                     ptr of inf list:= ptr of inf list + p;
1929          STORE in contents of(st(st(st(0,
1930                                  0),
1931                                  0),
1932                                  STORE in contents of(st(0,
1933                                                       i + A))
1934                                  ))
1935        end else
```

```
1936        STORE letgits with(i + A);
1937        if s ≠ semicolon symbol then goto L1;
1938        number of functions:= i; end fctn part:=ptr of inf list - 1;
1939        declaration:= false
1940      end
1941    end INITIALIZE other variables;
1942
1943    comment:
1944
1945    16.  Printing equipment.
1946
1947    Algorithm:;
1948
1949    procedure PR ELAN SYM(s); value s; integer s;
1950    if linenumber < 0 then
1951    begin if s=nlcr symbol then line number:=line number+1 end else
1952    if s = nlcr symbol then line counter:= line counter + 1 else
1953    begin if line counter ≥ 4 then
1954      begin PRINT LINE;
1955            line buffer[0]:= linenumber:= linenumber + line counter;
1956            if second scan ∧ LINE NUMBER ≠ 1 then NEW PAGE;
1957            fill pointer:= 24; line counter:= 0
1958      end else
1959      if line counter > 0 then
1960      begin for line counter:=line counter-1 while line counter≥0 do
1961      begin PRINT LINE;
1962            line buffer[0]:= linenumber:= linenumber + 1
1963      end;
1964        fill pointer:= 24; line counter:= 0
1965      end;
1966      if second scan then
1967      begin fill pointer:= fill pointer + 1;
1968        line buffer[fill pointer]:=s;
1969        if fill pointer = 200 then
1970        begin PRINT LINE; fill pointer:= 24 end
1971      end;
1972      if s = semicolon symbol then PRINT LINE
1973    end PR ELAN SYM;
1974
1975    procedure PR ad cntr;
1976    if second scan then PRINT octal(6,address counter,false);
1977
1978    procedure PRINT LINE;
1979    if second scan then
1980    begin integer i;
1981      if line buffer[0] > 0 then
1982      begin ABSFIXT(6,0,line buffer[0]); line buffer[0]:=-1
1983      end else space(8);
1984      if line buffer[2] ≥0 then
1985      begin for i:= 1 step 1 until 11 do PRSYM(line buffer[i]);
1986        line buffer[2]:=-1
```

```
1987        end else
1988        if line buffer[-1]>2 then
1989        begin ABSFIXT(6,0,line buffer[-1]-2);
1990          line buffer[-1]:=-1; SPACE(3)
1991        end else SPACE(11);
1992        if line buffer[13] > 0 then
1993        begin for i:=12 step 1 until 24 do PRSYM(line buffer[i]);
1994          line buffer[13]:=-1
1995        end else SPACE(13);
1996        for i:=25 step 1 until fill pointer do
1997        begin PRSYM(line buffer[i]); if line buffer[i]≠tab symbol then
1998          line buffer[i]:=space symbol
1999        end;
2000        PRSYM(nlcr symbol)
2001      end PRINT LINE;
2002
2003      procedure PR list(entry); integer entry;
2004      begin comment This procedure prints out the interesting part of
2005        the namelist. If you want it punched, please punch after the
2006        last 'end' the symbols "nl".;
2007        integer i,q1,q2,q3,q4,marge;
2008
2009        procedure p sym(q); value q; integer q;
2010        if q ≠ -1 then P(q);
2011
2012        procedure P(q); value q; integer q;
2013        begin PRSYM(q); if punch list then PUSYM(q) end P;
2014
2015        procedure S(q); value q; integer q;
2016        begin integer end space;
2017          end space:= printpos + q;
2018        L: if((printpos + 1) : 8 + 1) × 8 ≤ end space then
2019          begin TAB; if punch list then PUSYM(tab symbol);
2020            goto L
2021          end else
2022          begin if punch list then PUSPACE(end space - printpos);
2023                SPACE(end space - printpos);
2024          end
2025        end S;
2026
2027        procedure A(n,m,x); value n,m,x; integer n,m; real x;
2028        begin ABSFIXT(n,m,x); if punch list then ABSFIXP(n,m,x) end A;
2029
2030        procedure F(n,m,x); value n,m,x; integer n,m; real x;
2031        begin FIXT(n,m,-x); if punch list then FIXP(n,m,x) end F;
2032
2033        procedure pr list(entry1); value entry1; integer entry1;
2034        if entry1 ≠ 0 then
2035        begin integer num;
2036          boolean write;
2037          procedure PRINT name;
```

```
2038        for i:= 1 step 1 until num do
2039        begin q4:= contents of[entry1 + i];
2040          q3:= q4 : t6; q2:= q3 : t6; q1:= q2 : t6;
2041          q4:= q4 -q3 × t6 - 1; q3:= q3 - q2 × t6 - 1;
2042          q2:= q2 -q1 × t6 - 1; q1:= q1 - 1;
2043          p sym(q1); p sym(q2); p sym(q3); p sym(q4)
2044        end PRINT name;
2045
2046        num:= contents of[entry1]; write:= false;
2047        pr list(contents of[entry1 + num + 1]);
2048        if entry1 > end fctn part then
2049        begin
2050          procedure pr inf(entry2); value entry2; integer entry2;
2051          if entry2 ≠ 0 then
2052          begin integer block, type, value, line, cc;
2053            procedure PRINT value;
2054            begin integer i,pointer;
2055              integer array ar[1:18];
2056              procedure store in ar(information);
2057              value information; integer information; ;
2058
2059              integer procedure st(x,y); value x; integer x,y;
2060          begin pointer:=pointer+1;ar[pointer]:=y;st:=0 end st;
2061
2062              pointer:= 0;
2063              if type = 1 then store in ar(st(st(0,
2064                                               M symbol),
2065                                               pr sub symbol)) else
2066              if type = 2 then store in ar(st(st(st(st(0,
2067                                               M symbol),
2068                                               pr sub symbol),
2069                                               B symbol),
2070                                               plus symbol)) else
2071              if type = 3 then store in ar(st(0,
2072                                               M symbol)) else
2073              if type = 5 then store in ar(st(st(0,
2074                                               colon symbol),
2075                                               M symbol)) else
2076              if type =12 then store in ar(st(st(st(st(st(st(st(0,
2077                                               sub symbol),
2078                                               apostrophe symbol),
2079                                               M symbol),
2080                                               T symbol),
2081                                               apostrophe symbol),
2082                                               bus symbol),
2083                                               M symbol),
2084                                               pr sub symbol));
2085
2086              if type = 1 ∨ type = 2 ∨ type = 4 ∨ type = 12 then
2087              begin if value > t19 - 1 ∨ 1/value< 0 then
2088            begin S(6-pointer);
```

```
2089            for i:= 1 step 1 until pointer do P(ar[i]);
2090                PRINT octal(9,value,punchlist);
2091                P(if type = 4 then space symbol else pr bus symbol)
2092              end else
2093          begin S(9-pointer);
2094            for i:= 1 step 1 until pointer do P(ar[i]);
2095                PRINT octal(6,value,punchlist);
2096                P(if type = 4 then space symbol else pr bus symbol)
2097              end
2098            end else
2099            if type= 3 ∨ type= 5 then
2100            begin integer p,q,qq;
2101              p:= value : 512; q:= value -512×p-256;
2102              if p≥ 58 ∧ p< 63 then
2103              store in ar (st(0,
2104              if p=58 then G symbol else if p=59 then A symbol else
2105              if p=60 then S symbol else if p=61 then C symbol else
2106              if p= 62 then T symbol else D symbol)) else
2107              if p≥ 0 ∧ p< 10 then store in ar(st(0,
2108                                                  p)) else
2109              if p≥ 10 ∧ p< 100 then
2110              begin qq:= p: 10; store in ar(st(st(0,
2111                                                 qq),
2112                                                 p-qq×10))
2113              end;
2114              store in ar(st(0,
2115                          pr sub symbol));
2116              if q< 0 then
2117              begin store in ar(st(0,
2118                                   minus symbol));
2119                              q:= -q
2120              end;
2121              if q< 10 then store in ar(st(st(0,
2122                                             q),
2123                                             pr bus symbol)) else
2124              if q< 100 then
2125              begin qq:= q: 10; store in ar(st(st(st(0,
2126                                                    qq),
2127                                                    q-qq×10),
2128                                                    pr bus symbol))
2129              end else
2130              if q< 1000 then
2131              begin qq:= q: 100; store in ar(st(st(st(st(0,
2132                                                        qq),
2133                                                        (q-qq×100): 10),
2134                                                        q-q: 10×10),
2135                                                        pr bus symbol))
2136              end;
2137          S(18-pointer);
2138          for i:= 1 step 1 until pointer do P(ar[i])
2139            end else
```

```
2140            if type= 6 then
2141            begin S(8); for i:= 1,2,3 do P(122); S(7) end else
2142            if type= 10 then
2143            begin S(9); P(122); S(8) end else
2144            if type= 11 then
2145            begin S(6); PRINTTEXT(‡'mt' ?‡);
2146              if punchlist then PUTEXT(‡'mt' ?‡); S(6)
2147            end else
2148            A(16,0,value)
2149          end PRINT value;
2150
2151          block:= contents of[entry2];
2152          type:= contents of[entry 2 + 1];
2153          value:= contents of[entry 2 + 2];
2154          line:= contents of[entry 2 + 3];
2155          cc:= 0;
2156          pr inf(contents of[entry 2 + 5]);
2157          begin
2158        procedure pr call(entry3);value entry3;integerentry3;
2159            if entry3 ≠ 0 then
2160            begin integer line;
2161              if entry3 - drum pointer > 0 ∧
2162                 entry3 - drum pointer ≤ max of buffer then else
2163          begin drum pointer:=
2164            (entry3-1) : max of buffer × max of buffer;
2165               FROM DRUM(LINE,drum pointer)
2166            end;
2167          entry3:=entry3-(entry3-1):max of buffer×max of buffer;
2168            if entry3 > 0 ∧ entry3 ≤ 4096 then
2169          begin line:= LINE[entry3];
2170            pr call(LINE[entry3 + 1])
2171          end else
2172            line:= 0;
2173            cc:= cc + 1; if cc=11 then
2174          begin P(nlcr symbol);
2175            if LINE NUMBER = 1 then PRINT name;
2176            if printpos ≤ marge + 32 then
2177            S(marge + 32 - printpos) else
2178               begin P(nlcr symbol); S(marge + 32) end;
2179               cc:= 1
2180            end;
2181            if line > 0 then A(6,0,line) else F(6,0,line)
2182          end pr call;
2183
2184          if block ≠ 1 ∧ block ≠ 2 then
2185          begin if ¬ write ∨ LINE NUMBER = 1 then
2186            begin write:= true; PRINT name end;
2187            if printpos ≤ marge then S(marge - printpos) else
2188            begin P(nlcr symbol); S(marge) end;
2189            PRINT value;
2190            if block ≠ 0 then A(4,0,block - 2) else F(4,0,0);
```

```
2191                if line > 0 then A(6,0,line) else F(6,0,line);
2192                pr call(contents of[entry2 + 4]);
2193                P(nlcr symbol)
2194              end
2195            end
2196          end pr inf;
2197          pr inf(contents of[entry1 + num + 3])
2198        end;
2199        write:= false;
2200        pr list(contents of[entry1 + num + 2])
2201      end pr list;
2202
2203      marge:= max of num × 4; if marge > 20 then marge:= 20;
2204      if punch list then
2205      begin for i:= 1 step 1 until 5 do RUNOUT;
2206        ABSFIXP(2,0,marge); PUNLCR
2207      end;
2208      pr list(entry)
2209    end PR list;
2210
2211    comment:
2212
2213    17. The main program.
2214
2215    Algorithm:;
2216
2217    begin
2218      procedure START block;
2219      begin not behind last end:= true; declaration:= false;
2220        pr tape symbol:= space symbol; reading ptr:= ptr of text:= 0;
2221        NS deferred:= true; CHECK fault:= false;
2222        address counter:= 0; line counter:= cntr:= 0;
2223      L: RE through barrier; if synt unit ≠ begin symbol then goto L
2224      end START block;
2225
2226      PRINTTEXT (⊧mc elan1 assembler d.d. 01 04 71⊧);
2227      stringsymcount:= endsymcount:= 0; from string:= true;
2228      INITIALIZE symbols; INITIALIZE other variables;
2229      initialization:= first scan:= true; second scan:= false;
2230      cntr of begins:= 0; linenumber:= - 11;
2231      block[cntr of begins]:= max block number:= nr of begins:= 0;
2232      START block;
2233      INITIALIZE inf words; initialization:= false;
2234      count3:= triple:= 0; t8i:= t0; buffer ptr:=0;
2235      drum ptr:= min of drum; nr of begins:= -1;
2236      START block; RE ELAN block; PR ELAN SYM (pr tape symbol);
2237      for count3:= count3, count3, count3 do stow into buffer (0);
2238      BUFFER TO DRUM;
2239      first scan:= false; second scan:= IP:= true;
2240      cntr of begins:= max block number:= 1; nr of begins:= -1;
2241      linenumber:= -11; fill pointer:=24; line buffer[0]:= 1;
```

```
2242        line buffer[2]:= line buffer[-1]:= line buffer[13]:= -1;
2243        count3:= 3; buffer ptr:= max of buffer; drum ptr:= min of drum;
2244        NEW PAGE; RUNOUT; RUNOUT;
2245        START block; RE ELAN block; PR ELAN SYM (pr tape symbol);
2246        PR ELAN SYM(nlcr symbol); PR ELAN SYM(space symbol);
2247        PUNCH; if IP then single IP (0) else single BI(0,true,true);
2248        TO DRUM(LINE,drum pointer); NEW PAGE;
2249        second scan:= false;
2250        begin integer hep1, hep2;
2251          hep1:= pr tape symbol; hep2:= REHEP; punch list:= false;
2252        L:if hep2-hep1=2 ∨ hep2-hep1=96 then punch list:=true else
2253          if rehep available then
2254          begin hep1:= hep2; hep2:= REHEP; goto L end
2255        end;
2256        PR list(1); RUNOUT; RUNOUT
2257      end
2258    end
2259
```

```
2250 hep1
       2250 2251 2x2252 2254

2250 hep2
       2250 2251 2x2252 2x2254

 597 i
        597  610  2x613  618  620  622

 679 i
        679  683  2x684  687  688

 700 i
        700  702  2x704  705

 883 i
        883  942  943  944  989  990

1123 i
       1123 1124 1125 1126 1127 1128 1129

1146 i
       1146 2x1148 2x1151 1155 3x1159 1163 1167 1168 1169 1170
       1171 1180 1182 1186 1191 1192 1198 1199 1204 1205

1536 i
       1536 1539 2x1540

1548 i
       3x1548 1550 1553

1567 i
       1567 2x1575 3x1581

1607 i
       1607 1608 2x1613

1619 i
       1619 1624 1627 1628 2x1631

1636 i
       1636 1637 2x1642

1646 i
       3x1646 2x1651 1652

1674 i
       3x1674 1676

1824 i
       1824 2x1832 1847
```

1672 isregistersymbol
298 328 385 1672 1676 1680

1701 isstatsep
756 767 1701 1702

1668 isUYNsymbol
280 1668 1669

1123 j
1123 1124 1125 1126 1127 2x1129

501 k
501 511 512

1123 k
1123 1124 1125 1126 1127

584 L
584 586

762 L
762 765 768

773 L
773 778

786 L
786 795

813 L
813 818 820 822 823

1156 L
1156 1162

1743 L
1743 1747

1792 L
1792 1801

1826 L
1826 1827

2018 L
2018 2020

2223 L
2x2223

2009 q
3x2009 2x2010

2012 q
3x2012 2x2013

2015 q
3x2015 2017

2100 q
2100 2101 2116 2x2119 2121 2122 2124 2125 2127 2130
2131 2133 2x2134

597 q1
597 609 613 4x616 617 630 2x635 2x636 637 642
644

2007 q1
2007 2040 3x2042 2043

597 q2
597 616 620 624 626 630 2x635 2x636 637 639
642 644

2007 q2
2007 2x2040 2041 2x2042 2043

597 q3
597 3x636 637 639 642 644

2007 q3
2007 2x2040 3x2041 2043

2007 q4
2007 2039 2040 2x2041 2043

2100 qq
2100 2110 2111 2112 2125 2126 2127 2131 2132 2133

1789 questionmark
1789 1804 1807 1809 1819

1779 quotesymbol
692 733 1150 1154 1703 1779 1843

700 r
700 709 713 714 2x719 723

1535 r
3x1535 4x1538 1539

1871 t4
1871 1886

1871 t5
2x1501 1599 1871 1886

1871 t6
600 2x616 635 636 648 1598 1599 1871 1886 1912
1921 3x2040 2x2041 2042

1871 t7
815 2x816 2x1602 1871 1887

1871 t8
339 340 3x1064 1066 1070 1075 1081 2x1098 1610 1611
1639 1640 1871 1887 1922

1049 t8i
1049 1058 1064 3x1066 1067 2x1070 3x1075 1081 2234

1871 t9
340 341 1871 1887 1923

1871 t10
341 342 1871 1887 1894 1924

1871 t11
1871 1887

1871 t12
2x635 1871 1887 1892

1872 t13
1872 1887

1872 t14
2x1601 1872 1887

1872 t15
339 344 370 1512 1519 1872 1888

1872 t16
1872 1888 1891

1872 t17
1522 1872 1888

1872 t18
1611 1612 1640 1641 1872 1888

1872 t19
1171 1872 1888 2087

```
2036 write
        2036 2046 2185 2186 2199

1017 x
        2x1017 1021

1031 x
        3x1031

1564 x
        1564 2x1565 3x1568 1569 3x1570 3x1577

2027 x
        3x2027 2x2028

2030 x
        3x2030 2x2031

2059 x
        3x2059

1500 X8word
        1500 1514 1524

1017 y
        2x1017 1022

1031 y
        2x1031 1045

2059 y
        2x2059 2060

1783 YNsymbol
         631 1175 1670 1783 1849

1778 Ysymbol
         612  630  642 1518 1669 1778 1841

1778 Zsymbol
         612  644 1521 1665 1778 1841

ranges =   210
nlex =   447
nidd =  4087
nlink =  4071
```

Appendix B

In this appendix we reproduce the ELAN text of the binary loader, such as it is produced by the assembler including adresses and machine instructions. This text is followed by:

a) the ELAN source text of a test program which tests the binary loader,

b) the printer output as produced by the ELAN assembler.

The test program has been chosen such, that it not only tests the loader, but that it is also an illustrative example for the assembler, in particular with respect to declarations and different types of operands.

```
"Test for ELAN assembler and binary loader rpr 060571/1, 2014n
'begin' zz = M,yy = M[:zz],xx = zz,ww = t,vv = :xx[:xx + :zz],
        uu = M[b],tt = M[b - (:yy + vv)],ss = zz[b + (:yy + vv)],
        aa,bb = ('200 000' - '2 000'), end of memory


                  'bi'
M['123']:         1; 2; 3; 4; 5; 6;
M['330']:         -16
M[56]:            1
M[63]:            1
M[513]:           aa: 1; 2; 3; 4; 5
end of memory:    ('200 000' - 1)
        'begin' zz = MG,yy = MA,xx = MS,ww = MC,vv = MT,
                uu = MD,tt = M23[45],ss = M58[10 - 55]
        M[bb - '100']:  1
        M[bb - '400']:  2
        M[bb - '700']:  3
        M[bb - '1400']: 4
        M[bb - '1700']: 5
        "Note that the begin-address of the loader is bb and that its
        "length is '425' instructions.
        'end'
        'begin' uu = :aa,tt = 22,ss = (:aa + :a -:aa)
        M[bb + '350']: -1; -2; -3; -4; -5
                "The loader is now shifted over '55'
                " places, and the machine is inspected to check that
                " the end-address of the loader is indeed
                "       (bb + '150' - 1) = '176 347'.
                "The technique to force the loader to stop is by means
                " of an end-of-tape situation.
                'begin' uu = :mg,tt = :ma,ss = :mc
                M[bb + '150']: -1; -2; -3; -4; -5
                "The loader is now shifted to the first free traject,
                " its end-address is now (bb - '700' - 1) = '175 077'
                " once more this has to be inspected.
                'end'
        M[24]:    GOTO (25)
                        G = M[56]; G + M[63]; G + M['123']; G + M['124']
                        G + M['127']; G + M['130']; G + M['330']
                        B = end of memory
                        G + MC[-1]
                    U, B - 512, P
                    Y, JUMP(-3)
                        G - end of memory, Z
                    Y, A = -1
                    N, A = -2    "a forbidden situation.
                    U, A = MA    "the X8 stops.
'end'   'end'
```

```
                                "LOADER OF THE BINARY PAPERTAPES, PRODUCED BY THE MC - ELAN
                                " ASSEMBLER, RPR 100370 - R 2014N
      1                         'BEGIN' VERY FIRST BEGIN, VERY FIRST END, CLEAR, MOVE, INITIALIZATION,
                                        LAST ADDRESS = ('200 000' - 1), SUMCHECK, RESET NR = 38,
                                        RESET AR = M[64 + RESETNR * 4],
                                   'MT' BEGINOFREADINGPROGRAM, ENDOFREADINGPROGRAM, LENGTHOFREADINGPROGRAM,
                                        ENDOFMEMORY, IP READER, BSTACK, BRUSH, DYST, REHEP,
                                        INIT REHEP, INIT REHEP AFTER MOVING, REHEP INF,
                                        REBIWORD, REBITAPE, SETWORD, ENDACT,
                                        IPAR, SHADOW IPAR, SHADOW RESET AR
'000001': '000176000'   M[1]:                   ;INITIALIZATION
'000002':               CLEAR:          "THIS ROUTINE SETS RANGE M[B] - M[B + COUNT0 - 1]
                                        "TO THE CONTENTS OF G, IT IS ACTIVATED IN ENDACT
'000002': '726175400'                           MC = G
'000003': '540400002'                           REPO P(:CLEAR)
'000004': '760060000'                           IV ON                   ")NOW THE INTERRUPT OF THE
'000005': '512000001'                           JUMP (-1)               ")READER OCCURS
'000006':               MOVE:           "THIS ROUTINE MOVES RANGE M[A] - M[A + COUNT0 - 1] TO
                                        "M[B] - M[B + COUNT0 - 1] AND SETS THE OLD RANGE TO THE
                                        "CONTENTS OF G, IT IS CALLED IN SET WORD
'000006': '126073400'                           S = MA
'000007': '166075400'                           MC = S
'000010': '726173400'                           MA = G
'000011': '002000001'                           A + 1
'000012': '540400006'                           REPO P (:MOVE)
'000013': '520000027'                           GOTO (LINK 15)
'000014': '777777723'   SUMCHECK:               '777 777 723'
'000015': '413711611'                           '413 711 611'
'000030': '520000001'   M[24]:                  GOTO (M[1])
'000000':               M[LAST ADDRESS - 1023]:

'176000':               INITIALIZATION: VERY FIRST BEGIN;
      2                 'BEGIN'         NODECLARATION = 0
                                "A SUMCHECK IS DONE ON THE LOADER ITSELF
'176000': '622000000'                           F = 0
'176001': '426076752'                           B = ENDOFREADINGPROGRAM
'176002': '402000001'                           B + 1
'176003': '606175377'                           G + MC[-1]
'176004': '417176746'                       U, B = BEGINOFREADINGPROGRAM, Z
'176005': '512300003'                       N, JUMP (-3)
'176006': '611000014'                          F = SUMCHECK, Z
'176007': '622300000'                       N, A = 0
'176010': '526376744'                       N, GOTO (DYST)
                                "THE MEMORY IS CLEARED:
'176011': '632000000'                          G = -0
'176012': '422001000'                          B = 512
'176013': '726175400'                          MC = G
'176014': '457176736'                       U, BEGINOFREADINGPROGRAM - B, Z
'176015': '512300003'                       N, JUMP (-3)
'176016': '426076736'                          B = ENDOFREADINGPROGRAM
'176017': '402000001'                          B + 1
'176020': '726175400'                          MC = G
'176021': '457176736'                       U, ENDOFMEMORY - B, Z
'176022': '512300003'                       N, JUMP (-3)
                                "THE NUMBER OF THE IP - READER IS DETERMINED.
                                "STARTING THE COMPUTER, ALL THE I(NTERRUPT) F(LIPFLOP)S ARE CLEARED
                                "DURING THE ACTIVATION OF THE IP - READER THE IF FOR THIS APPARATUS IS SET
'176023': '660075000'                          IFA (0)                  "TAKE THE FIRST IF - WORD IN A
'176024': '760075001'                          IFS (1)                  "TAKE THE SECOND IF - WORD IN S
```

```
    61 '176025':  '660000340'                    NCRAS
    62 '176026':  '412500007'                  U, B = 7, P
    63 '176027':  '430300075'                  N, B = -B
    64 '176030':  '402300047'                  N, B + 39
    65 '176031':  '412200010'                  Y, B = 8
    66 '176032':  '466076723'                     IPREADER = B
    67 '176033':  '664071000'                     IF OFF (B)
    68 '176034':  '620100075'                     G = B
    69 '176035':  '602004000'                     G + '000 004 000'
    70                        "THE STACK IS INITIALIZED:
    71 '176036':  '462076722'                     B = :BSTACK
    72 '176037':  '572476755'                     SUBC (:BRUSH)
    73                        "THE INITIALIZATION IS STARTED:
    74 '176040':  '572476444'                     SUBC(:INIT REHEP)
    75                        "THE TAPE IS READ:
    76 '176041':  '536076554'                     GOTO(:REBITAPE)
    77       2                'END' INITIALIZATION
    78
    79 '176042':              REHEP:  "THIS SUBROUTINE, CALLED BY SUBC(:REHEP), DELIVERS IN S
    80                                "THE NEXT HEPTAD OF THE TAPE, IN AN 'END-OF-TAPE' SITUATION
    81                                "NEW STARTING COMMANDS ARE GIVEN TO THE READER TILL THE NEXT
    82                                "TAPE HAS BEEN INSERTED.
    83                                "AS THE FIRST HEPTAD, WHICH IS READ, IS UNDEFINED, DUE TO THE
    84                                "BUFFERING SYSTEM OF THE READER, THIS ONE IS SKIPPED.
    85                                "THE NEXT NON - LOCALS ARE USED: IPREADER, DYST
    86       3                        'BEGIN'  BUFLENGTH = 10,
    87                                   'MT' HEPPOINTER, STARTLINK, CODEWORD, ENDWORD, FIRSTONE, SAVEB,
    88                                        D18, STARTCOMMAND, OK, HEP
    89 '176042':  '466076466'                     SAVEB = B
    90 '176043':  '022000012'                     A = BUFLENGTH
    91 '176044':  '056576442'                  U, HEPPOINTER - A, P        "HEPPOINTER > BUFLENGTH?
    92 '176045':  '536376430'                  N, GOTO (:HEP)
    93                        "THE BUFFER IS EMPTY, TO START THE READER, A NEW COMMAND HAS TO BE GIVEN:
    94 '176046':  '626176716' STARTCOMMAND:       G = IPAR
    95 '176047':  '432000000'                     B = -0
    96 '176050':  '466072401'                     MG[1] = B                   "IFT:= -1
    97 '176051':  '426076460'                     B = D18
    98 '176052':  '466072402'                     MG[2] = B                   "AFT:= 1
    99 '176053':  '426076702'                     B = IPREADER
   100 '176054':  '764070001'                     AFON(B)                     "START IPREADER
   101 '176055':  '026572401'                  U, A = MG[1], P               "WAIT FOR COMMUNICATION
   102 '176056':  '512300002'                  N, JUMP(-2)                   "FROM THE READER
   103                       "NOK SITUATIONS ARE HANDLED:
   104 '176057':  '026476453'                     A = ENDWORD, P
   105 '176060':  '536276411'                  Y, GOTO (:OK)
   106 '176061':  '671000062'                     RUA(18), Z                  "NBK?
   107 '176062':  '022200001'                  Y, A = 1
   108 '176063':  '526276671'                  Y, GOTO (DYST)                "PHONE TECHNICIAN
   109                       "NOW WE HANDLE THE END-OF-TAPE SITUATION.
   110                       "A NEW STARTING COMMAND IS GIVEN, AFTER SETTING CNT AR0 = 0.
   111                       "FIRSTONE IS USED TO MARK, WHETHER THE FIRST HEPTAD IN THE
   112                       "BUFFER IS TO BE CONSIDERED.
   113 '176064':  '022000002'                     A = 2
   114 '176065':  '066076442'                     FIRSTONE = A
   115 '176066':  '626176676'                     G = IPAR
   116 '176067':  '162076444'                     S = :STARTLINK
   117 '176070':  '166072400'                     MG = S
   118 '176071':  '536076354'                     GOTO(:STARTCOMMAND)
   119 '176072':             OK:      "A BUFFER HAS BEEN READ:
   120 '176072':  '026076435'                     A = FIRSTONE
```

```
121 '176073':  '066076433'                      HEPPOINTER = A
122 '176074':  '022000001'                      A = 1
123 '176075':  '066076432'                      FIRSTONE = A
124 '176076':                HEP:    "A NEW HEPTAD CAN BE SELECTED:
125 '176076':  '162076435'                      S = :STARTLINK
126 '176077':  '106076427'                      S + HEPPOINTER
127 '176100':  '126074401'                      S = MS[1]
128 '176101':  '022000001'                      A = 1
129 '176102':  '046076424'                      HEPPOINTER + A
130 '176103':  '426076425'                      B = SAVEB
131 '176104':  '526475377'                      GOTOR (MC[-1])
132                          "INITIALIZATION OF REHEP
133 '176105':  '022000013'  INITREHEP:          A = (BUFLENGTH + 1)
134 '176106':  '066076420'                      HEPPOINTER = A
135 '176107':  '022000002'                      A = 2
136 '176110':  '066076417'                      FIRSTONE = A
137 '176111':  '126076644'                      S = IPREADER
138 '176112':  '760000042'                      LUS(2)
139 '176113':  '102000100'                      S + 64
140 '176114':  '166076650'                      IPAR = S
141 '176115':  '022000012'  INITREHEPAFTERMOVING:  A = BUFLENGTH
142 '176116':  '660000062'                      LUA (18)                "CODEWORD:= CNTPART*D18
143 '176117':  '042076415'                      A + :STARTLINK[1]       "                      + ADR
144 '176120':  '066076414'                      CODEWORD = A
145 '176121':  '026176643'                      G = IPAR
146 '176122':  '162076411'                      S = :STARTLINK
147 '176123':  '166072400'                      MC = S                  "AR0:= :STARTLINK
148 '176124':  '166074400'                      MS = S
149 '176125':  '166074377'                      MS[-1] = S
150 '176126':  '526475377'                      GOTOR (MC[-1])
151                          "SPACE FOR VARIABLES AND THE BUFFER:
152 '176127':  '000000013'  HEPPOINTER:        (BUFLENGTH + 1)
153 '176130':  '000000002'  FIRSTONE:          2
154 '176131':  '000000000'  SAVEB:             0
155 '176132':  '001000000'  D18:               '1 000 000'
156 '176133':  '000000000'  ENDWORD:           0
157 '176134':  '000000000'  STARTLINK:         0
158 '176135':  '000000000'  CODEWORD:          0
159 '176136':  '000000000'                      0
160 '176137':  '000000000'                      0
161 '176140':  '000000000'                      0
162 '176141':  '000000000'                      0
163 '176142':  '000000000'                      0
164 '176143':  '000000000'                      0
165 '176144':  '000000000'                      0
166 '176145':  '000000000'                      0
167 '176146':  '000000000'                      0
168 '176147':  '000000000'                      0
169       3                 'END' REHEP
170
171 '176150':                REHEPINF:  "THIS SUBROUTINE READS A HEPTAD FROM A TAPE, THE LENGTH OF
172                                      "WHICH CAN BE CONSIDERED TO BE INFINITE, DUE TO HANDLING OF
173                                      "END-OF-TAPE SITUATIONS INTERNALLY. A CALL, BY SUBC (:REHEPINF)
174                                      "DELIVERS ONE HEPTAD IN S.
175       4                 'BEGIN' 'MT' START
176 '176150':  '572476271'  START:               SUBC(:REHEP)
177 '176151':  '363100100'                    U, S '*' 64, Z            "7-TH HOLE NOT PUNCHED?
178 '176152':  '526675377'                    Y, GOTOR(MC[-1])
179 '176153':  '113000177'                       S = 127, Z             "IS HEPTAD ERASE?
180 '176154':  '022300002'                    N, A = 2
```

```
181 '176155':  '526376577'                    N, GOTO(DYST)
182 '176156':  '572476963'                       SUBC (:REHEP)          "SKIP
183 '176157':  '113000177'                       S = 127, Z             "     BI = TAPE TILL
184 '176160':  '512300003'                    N, JUMP (-3)              "                  ERASE
185 '176161':  '536076366'                       GOTO (:START)
186       4                'END' REHEPINF
187
188 '176162':                REBIWORD:       "THIS SUBROUTINE, CALLED BY SUBC (:REBIWORD), READS A BI = WORD
189                                  "IN A AND S. A CONTAINS C1 AND C2, S CONTAINS D26 - D0.
190                                  "IN CASE OF A PARITY ERROR THE MACHINE WILL STOP DYNAMICALLY,
191                                  "THE FIRST HEPTAD IS EXPECTED BY REBIWORD IN S
192       5                'BEGIN' 'MT' SAVEA, SAVES, CNT, REPEAT, D26
193 '176162':  '020000004'                        A = 4
194 '176163':  '066076426'                       CNT = A                "CNT:= 4
195 '176164':  '660000146'  REPEAT;              LUAS(6)                "AS:= AS * 2**6
196 '176165':  '066076425'                       SAVEA = A
197 '176166':  '166076425'                       SAVES = S
198 '176167':  '572476360'                       SUBC (:REHEPINF)
199 '176170':  '026076422'                       A = SAVEA
200 '176171':  '106076422'                       S + SAVES              "AS:= AS + HEPTAD
201 '176172':  '526176417'                       G = CNT
202 '176173':  '513000001'                       G - 1, Z
203 '176174':  '726176415'                       CNT = G
204 '176175':  '536376366'                    N, GOTO (:REPEAT)         "IF CNT > 0 THEN GOTO REPEAT
205                          "THE PARITY HAS TO BE CHECKED AND THE SIGNBIT OF S SET.
206                          "THIS BIT IS NOT INVOLVED IN THE SHIFTPROCESS,
207 '176176':  '670400001'                        RCA (1), P             "D26 = 0? A:= A 'DIV' 2
208 '176177':  '346376415'                    N, S '+' D26
209 '176200':  '276376414'                    N, A '+'-D26              "SET BIT26:= 0
210 '176201':  '240000074'                       A '+' S                "PAR(A)'+'PAR(S) = PAR(A'+'S)
211 '176202':  '066075400'                       MC = A
212 '176203':  '026075377'                       A = MC[-1]             "LP:= PAR(A '+' S)
213 '176204':  '240000074'                       A '+' S                "A:= A '+' S '+' S = A
214 '176205':  '600063000'                       CLP                    "C:= LP
215 '176206':  '020300003'                    N, A = 3                  "EVEN
216 '176207':  '526376545'                    N, GOTO(DYST)
217 '176210':  '272000004'                       A '*' -4               "DELETE PARITY BIT
218 '176211':  '526475377'                       GOTOR (MC[-1])
219 '176212':  '000000000'  CNT:                 0
220 '176213':  '000000000'  SAVEA;               0
221 '176214':  '000000000'  SAVES;               0
222 '176215':  '400000000'  D26;                 '400 000 000'
223       5                'END' REBIWORD
224
225 '176216':                REBITAPE: "THIS SUBROUTINE, CALLED BY GOTO (:REBITAPE), READS THE BI = BLOCKS
226                                  "OF THE BI = TAPE. TO SET A MEMORY WORD, SET WORD IS CALLED, IN THE
227                                  "CASE OF AN END MARKER ENDACT.
228       6                'BEGIN' 'MT' CNT, ADR, D18M1, READBIWORD, READBIBLOCK
229                          "WE START BY READING THE FIRST WORD, THE DIRECTIVE, OF THE BLOCK.
230 '176216':  '572476331'  READBIBLOCK:          SUBC (:REHEPINF)       "S:= REHEP
231 '176217':  '021100074'                    U, A = S, Z               "S BLANK
232 '176220':  '512200003'                    Y, JUMP (-3)
233 '176221':  '572476340'                       SUBC(:REBIWORD)        "AS:= BI= WORD
234 '176222':  '013100003'                    U, A = 3, Z
235 '176223':  '020300004'                    N, A = 4
236 '176224':  '526376530'                    N, GOTO (DYST)
237 '176225':  '020000074'                       A = S                  "ADR
238 '176226':  '266076430'                       A '*' D18M1            "   BECOMES
239 '176227':  '066076426'                       ADR = A                "      ADDRESS PART OF S
240 '176230':  '770000022'                       RCS (18)               "CNT
```

```
241 '176231':  '363000777'                    S '*' '777', Z          "   BECOMES
242 '176232':  '122201000'                 Y, S = 512                 "      COUNTPART
243 '176233':  '166076421'                    CNT = S                 "               OF S
244                         "NOW WE HANDLE THE CASE OF AN END-MARKER
245 '176234':  '113101000'                 U, S = 512, Z
246 '176235':  '003200000'                 Y, A = 0, Z
247 '176236':  '572676534'                 Y, SUBC (:END ACT)
248                         "CNT BI- WORDS SHOULD BE READ AND STORED SEQUENTIALLY IN THE MEMORY
249                         "STARTING FROM ADR + 1.
250 '176237':  '572476310' READBIWCRD:        SUBC(:REHEPINF)         "S:= HEPTAD
251 '176240':  '572476321'                    SUBC (:REBIWORD)        "AS:= BI-WORD
252 '176241':  '003000000'                    A = 0, Z                "C1 'NE' 0 ∨ C2 'NE' 0?
253 '176242':  '022300005'                 N, A = 5
254 '176243':  '526376511'                 N, GOTO (DYST)
255 '176244':  '022000001'                    A = 1
256 '176245':  '046076410'                    ADR + A
257 '176246':  '020076407'                    A = ADR
258 '176247':  '572476410'                    SUBC (:SETWORD)
259                         "THE READING PROGRAM MAY HAVE BEEN MOVED NOW, THE VALUES OF CNT AND ADR
260                         "HOWEVER ARE UNCHANGED
261 '176250':  '020076404'                    A = CNT
262 '176251':  '013000001'                    A = 1, Z                "CNT:= CNT - 1
263 '176252':  '066076402'                    CNT = A                 "IF CNT>0 THEN GOTO REPEAT
264 '176253':  '536376363'                 N, GOTO (:READBIWORD)
265                         "READ THE NEXT BI BLOCK:
266 '176254':  '530076341'                    GOTO (:READBIBLOCK)
267 '176255':  '000000000' CNT:               0
268 '176256':  '000000000' ADR:               0
269 '176257':  '000777777' D18M1:             '777 777'
270       6                 'END' REBITAPE
271
272 '176260':               SETWORD: "THIS SUBROUTINE SETS MA:= S. IF EXECUTION OF THE INSTRUCTION
273                                  "SHOULD OVERWRITE THE READINGPROGRAM, THEN BEFORE EXECUTION
274                                  "A) IT IS CHECKED, WHETHER THERE EXISTS A COHERENT RANGE IN
275                                  "   THE MEMORY FOR THE READINGPROGRAM, AND
276                                  "B) THE READINGPROGRAM IS MOVED.
277                                  "ASSUMED TO BE GLOBAL ARE: BEGINOFREADINGPROGRAM,
278                                  "ENDOFREADINGPROGRAM, LENGTHOFREADINGPROGRAM.
279       7                       'BEGIN' 'MT' SAVEA, SAVES, SAVEB, REPEAT, RETURNADDRESS, SET, DISPL.
280 '176260':  '066076467'                    SAVEA = A
281 '176261':  '056676471'                 U, BEGINOFREADINGPROGRAM - A, P
282 '176262':  '016776471'                 N, A = ENDOFREADINGPROGRAM, P
283 '176263':  '536376420'                 N, GOTO (:DISPL)
284                         "NOW WE TEST WHETHER THE MEMORY CELL ASKED FOR BY THE PROGRAM TO BE LOADED
285                         "IS ONE OF THE 'HOLY' ADDRESSES
286 '176264':  '020076463'                    A = SAVEA
287 '176265':  '012500070'                 U, A = :M[56], P
288 '176266':  '013500076'                 U, A = :M[62], E
289 '176267':  '012600027'                 Y, A = :LINK15, P
290 '176270':  '022300006'                 N, A = 6
291 '176271':  '526376463'                 N, GOTO (DYST)
292 '176272':  '020076455'                    A = SAVEA
293                         "SECONDLY WE TEST WHETHER THE MEMORY CELL IS ONE OF THE
294                         "APPARATUS REGISTERS OF THE IP READER OR THE RESET, IN WHICH
295                         "CASE A SHADOW ADMINISTRATION IS PERFORMED.
296 '176273':  '626176471'                    G = IPAR
297 '176274':  '052572403'                 U, A = :MG[3], P
298 '176275':  '053572377'                 U, A = :MG[-1], E
299 '176276':  '010300072'                 N, A = G
300 '176277':  '042376466'                 N, A = :SHADOW IPAR
```

```
301 '176300':  '013100330'                    U, A = :RESET AR, Z
302 '176301':  '062276470'                    Y, A = : SHADOW RESET AR
303 '176302':  '166073400'  SET:                 MA = S
304 '176303':  '526475377'                       GOTOR (MC[-1])
305 '176304':               DISPL: "A, S AND B SHOULD BE SAVED AS THE READINGPROGRAM IS DISPLACED.
306 '176304':  '466076444'                       SAVEB = B
307 '176305':  '166076444'                       SAVES = S
308                        "FIRST IT IS CHECKED, WHETHER THERE EXISTS A RANGE, IN WHICH THE
309                        "READINGPROGRAM FITS, I.E. A RANGE M[I], I = START ADDRESS,...,
310                        "END ADDRESS, M[I] = 0, ENDADDRESS - STARTADDRESS = ENDOFREADINGPROGRAM
311                        "- BEGINOFREADINGPROGRAM, SETADDRESS 'GE' END ADDRESS + 1.
312 '176306':  '026076450'                       A = LENGTHOFREADINGPROGRAM ")A IS THE NUMBER OF WORDS,
313 '176307':  '006076443'                       A + BEGINOFREADINGPROGRAM  ")THAT IS NEEDED FOR THE NEW
314 '176310':  '016076437'                       A = SAVEA                  ")RANGE, THE FIRST TIME THIS IS
315                                                                         ")LESS THAN
316                                                                         ")LENGTHOFREADINGPROGRAM
317 '176311':  '426076441'                       B = BEGINOFREADINGPROGRAM
318 '176312':  '620100073'  REPEAT:              G = A
319 '176313':  '602000777'                       G + 511
320 '176314':  '410500072'                    U, B = G, P                   "ANY SPACE LEFT?
321 '176315':  '022300007'                    N, A = 7
322 '176316':  '526376436'                    N, GOTO (DYST)
323 '176317':  '120000075'                       S = B                      ")S CONTAINS THE LAST ADDRESS
324 '176320':  '110000073'                       S = A                      ")TO BE TESTED
325 '176321':  '026076435'                       A = LENGTHOFREADINGPROGRAM
326                                                                         ")THE NEXT TIME WE HAVE TO TEST
327                                                                         ")OVER THE WHOLE RANGE
328 '176322':  '627175377'                       G = MC[-1], Z              "M[B-1] = 0? B:= B - 1
329 '176323':  '536376366'                    N, GOTO (:RPEPEAT)
330 '176324':  '630500072'                       G = - G, P
331 '176325':  '536376364'                    N, GOTO (:REPEAT)
332 '176326':  '411100074'                    U, B = S, Z
333 '176327':  '512300006'                    N, JUMP (-6)                  "THE WHOLE RANGE TESTED?
334                        "A NEW RANGE IS FOUND WHICH MATCHES THE REQUIREMENTS, THE BASE ADDRESS OF
335                        "IT IS CONTAINED BY B. THE NEXT STEP IS TO MOVE THE READINGPROGRAM TO
336                        "M[B] - M[B + LENGTHOFREADINGPROGRAM - 1].
337                        "THIS IS DONE BY CALLING MOVE, STORED IN THE 'HOLY CORE'. MOVE EXPECTS
338                        "IN A THE OLD BEGIN ADDRESS, IN B THE NEW BEGIN ADDRESS, IN G THE VALUE,
339                        "WHICH SHOULD BE STORED IN THE OLD RANGE, IN COUNT0 THE LENGTH OF THE
340                        "READINGPROGRAM AND IN LINK15 THE RETURN ADDRESS.
341 '176330':  '060000000'                       COUNT0 = A
342 '176331':  '026076421'                       A = BEGINOFREADINGPROGRAM
343 '176332':  '110000073'                       S = A                      ")S = SHIFTLENGTH
344 '176333':  '146076415'                       SAVEB + S                  ")S AS WELL AS THE STACK TO
345 '176334':  '146076424'                       BSTACK + S                 "WHICH IT POINTS, THE DEPTH OF
346                                                                         ")WHICH NOW EQUALS 1, HAVE TO BE
347                                                                         ")MODIFIED.
348 '176335':  '146076416'                       ENDOFREADINGPROGRAM + S
349 '176336':  '146076414'                       BEGINOFREADINGPROGRAM + S
350 '176337':  '142076403'                       S + :RETURNADDRESS
351 '176340':  '160000027'                       LINK15 = S
352 '176341':  '632000000'                       G = -0                     "THE OLD RANGE IS SET TO -0
353 '176342':  '522000006'                       GOTO (:MOVE)
354 '176343':  '426076405'  RETURNADDRESS:       B = SAVEB
355 '176344':  '572476150'                       SUBC (:INITREHEPAFTERMOVING)
356 '176345':  '026076402'                       A = SAVEA
357 '176346':  '126076403'                       S = SAVES
358 '176347':  '536076332'                       GOTO (:SET)
359 '176350':  '000000000'  SAVEA:               0
360 '176351':  '000000000'  SAVEB:               0
```

```
   361 '176352':  '0000000000'  SAVES:                  0
   362       7                  'END' SETWORD
   363
   364                          "SOME SPACE FOR CONSTANTS AND VARIABLES IS RESERVED:
   365 '176353':  '0001760000'  BEGINOFREADINGPROGRAM:  ;INITIALIZATION
   366 '176354':  '0001764240'  ENDOFREADINGPROGRAM:    ;VERYFIRSTEND
   367 '176355':  '0007777777'  DYST:                   '777 777'
   368 '176356':  '0000000000'  IPREADER:               0
   369 '176357':  '0000000425'  LENGTHOFREADINGPROGpAM: (:VERYFIRSTEND - ;VERYFIRSTBEGIN + 1)
   370 '176360':  '0002000000'  ENDOFMEMORY:            (LASTADDRESS + 1)
   371 '176361':  '0000000000'  BSTACK:                 0                        ")FOR THE STACK 4 WORDS
   372 '176362':  '0000000000'                          0                        ")ARE NEEDED ONLY
   373 '176363':  '0000000000'                          0
   374 '176364':  '0000000000'                          0
   375 '176365':  '0000000000'  IPAR:                   0
   376 '176366':  '0000000000'  SHADOW IPAR:            0
   377 '176367':  '0000000000'                          0
   378 '176370':  '0000000000'                          0
   379 '176371':  '0000000000'                          0
   380 '176372':  '0000000000'  SHADOW RESET AR:        0
   381
   382 '176373':                ENDACT: "THIS ROUTINE HANDLES THE END ACTIONS:
   383                          "FIRSTLY THE APPARATUS REGISTERS OF THE IP READER AND THE RESET
   384                          "ARE FILLED.
   385 '176373':  '0620763720'                          A = : SHADOW IPAR
   386 '176374':  '4280763730'                          B = IPAR
   387 '176375':  '1620754040'                          S = : MC[+4]
   388 '176376':  '6261734000'                          G = MA
   389 '176377':  '7261754000'                          MC = G
   390 '176400':  '0020000001'                          A + 1
   391 '176401':  '4111000740'                       U, B = S, Z
   392 '176402':  '5123000005'                       N, JUMP(-5)
   393 '176403':  '0260763660'                          A = SHADOW RESET AR
   394 '176404':  '0600000330'                          RESET AR[0] = A
   395                          "SECONDLY, THE PROGRAM IS
   396                          "CLEARED, BY THE ROUTINE CLEAR, WHICH IS STORED IN THE 'HOLY' CORE.
   397                          "THIS ROUTINE EXPECTS IN B THE BEGINADDRESS, IN G THE VALUE, WITH WHICH
   398                          "THE READERPROGRAM HAS TO BE OVERWRITTEN AND IN COUNT0 THE NUMBER OF
   399                          "WORDS TO BE CLEARED. EVENTUALLY AN INTERRUPT OF THE IP - READER IS FORCED.
   400 '176405':  '0600600000'                          IV OFF                   "THE MACHINE BECOMES DEAF
   401 '176406':  '4260763470'                          B = IPREADER
   402 '176407':  '7640710000'                          IF ON (B)                "IF THE CPU WERE LISTENING,THERE
   403                                                                           "WOULD COME AN INTERRUPT NOW
   404 '176410':  '4260763420'                          B = BEGINOFREADINGPROGRAM
   405 '176411':  '4260763450'                          A = LENGTHOFREADINGPROGRAM
   406 '176412':  '0600000000'                          COUNT0 = A
   407 '176413':  '6320000000'                          G = -0
   408 '176414':  '5220000002'                          GOTO (:CLEAR)
   409 '176415':  '7201003330'  BRUSH:                  RESET AR[0] = G
   410 '176416':  '7600700460'                          AFON (RESETNR)
   411 '176417':  '7600750000'                          IFS (0)
   412 '176420':  '7604000002'                          LCS (2), P
   413 '176421':  '5122000003'                       Y, JUMP (-3)
   414 '176422':  '6600710460'                          IFOFF (RESETNR)
   415 '176423':  '5264753770'                          GOTOR(MC[-1])
   416 '176424':  '0000000001'  VERYFIRSTEND:           1
   417       1                  'END'
```

```
ADR                     ('MT')M['176256']   6   228   239   256   257   +268
BEGINOFREADINGPROGRAM
                        ('MT')M['176353']   1     7    40    49   281   313   317   342   349   +365   404
BRUSH                   ('MT')M['176415']   1     8    72  +409
BSTACK                  ('MT')M['176361']   1     8    71   345  +371
BUFLENGTH                     '000012'      3   +86    90   133   141   152
CLEAR                       M['000002']     1     4   +13    16   408
CNT                     ('MT')M['176212']   5   192   194   201   203   +219
                        ('MT')M['176255']   6   228   243   261   263   +267
CODEWORD                ('MT')M['176135']   3    87   144  +158
D16                     ('MT')M['176132']   3    88    97  +155
D1841                   ('MT')M['176257']   6   228   238  +269
D26                     ('MT')M['176215']   5   192   208   209  +222
DISPL                   ('MT')M['176304']   7   279   283  +305
DYST                    ('MT')M['176355']   1     8    44   108   181   216   236   254   291   322   +367
ENDACT                  ('MT')M['176373']   1    10   247  +382
ENDOFMEMORY             ('MT')M['176360']   1     8    54  +370
ENDOFREADINGPROGRAM     ('MT')M['176354']   1     7    37    51   282   348   +366
ENDWORD                 ('MT')M['176133']   3    87   104  +156
FIRSTONE                ('MT')M['176130']   3    87   114   120   123   136   +153
HEP                     ('MT')M['176076']   3    88    92  +124
HEPPOINTER              ('MT')M['176127']   3    87    91   121   126   129   134   +152
INITIALIZATION              M['176000']     1     4    12   +33   365
INITREHEP               ('MT')M['176105']   1     9    74  +133
INITREHEPAFTERMOVING    ('MT')M['176115']   1     9  +141   355
IPAR                    ('MT')M['176365']   1    11    94   115   140   145   296   +375   386
IPREADER                ('MT')M['176356']   1     8    66    99   137  +368   401
LASTADDRESS                   '177777'      1   +5    31   370
LENGTHOFREADINGPROGRAM
                        ('MT')M['176357']   1     7   312   325  +369   405
MOVE                        M['000006']     1     4   +19    26   353
NODECLARATION                 '000000'      2   +34
OK                      ('MT')M['176072']   3    88   105  +119
READBIBLOCK             ('MT')M['176216']   6   228  +230   266
READBIWORD              ('MT')M['176237']   6   228  +250   264
REBITAPE                ('MT')M['176216']   1    10    76  +225
REBIWORD                ('MT')M['176162']   1    10  +160   233   251
REHEP                   ('MT')M['176042']   1     8   +79   176   182
REHEPINF                ('MT')M['176150']   1     9  +171   198   230   250
REPEAT                  ('MT')M['176164']   5   192  +195   204
                        ('MT')M['176312']   7   279  +318   329   331
RESETAR                     M['000330']     1   +6   301   394   409
RESETNR                       '000046'      1   +5     6   410   414
RETURNADDRESS           ('MT')M['176343']   7   279   350  +354
SAVEA                   ('MT')M['176213']   5   192   196   199  +220
                        ('MT')M['176350']   7   279   280   286   292   314   356   +359
SAVEB                   ('MT')M['176131']   3    87    89   130  +154
                        ('MT')M['176351']   7   279   306   344   354  +360
SAVES                   ('MT')M['176214']   5   192   197   200  +221
                        ('MT')M['176352']   7   279   307   357  +361
SET                     ('MT')M['176302']   7   279  +303   358
SETWORD                 ('MT')M['176260']   1    10   256  +272
SHADOWIPAR              ('MT')M['176366']   1    11   300  +376   385
SHADOWRESETAR           ('MT')M['176372']   1    11   302  +380   393
START                   ('MT')M['176150']   4   175  +176   185
STARTCOMMAND            ('MT')M['176046']   3    88   +94   118
STARTLINK               ('MT')M['176134']   3    87   116   125   143   146   +157
SUMCHECK                    M['000014']     1     5   +28    42
VERYFIRSTBEGIN              M['176000']     1     4   +33   369
VERYFIRSTEND                M['176424']     1     4   366   369   +416
```

```
     1                    "TEST FOR BEAM ASSEMBLER AND BINARY LOADER RPR 060571/1, 2014N
     2       1            'BEGIN' ZZ = M,YY = M[:ZZ],XX = ZZ,WW = T,VV = :XX[:XX + :ZZ],
     3                            UU = M[B],TT = M[B - (:YY + VV)],SS = ZZ[B + (:YY + VV)],
     4                            AA,BB = ('200 000' - '2 000'), END OF MEMORY
     5
     6
     7                                      'B';
     6 '001123':  '000000001'  M['123']:    1;
       '001124':  '000000002'                  2;
       '001125':  '000000003'                     3;
       '001126':  '000000004'                        4;
       '001127':  '000000005'                           5;
       '001130':  '000000006'                              6;

     9 '000330':  '777777757'  M['330']:    -16
    10 '000070':  '000000001'  M[56]:       1
    11 '000077':  '000000001'  M[63]:       1
    12 '001001':  '000000001'  M[513]:      AA: 1;
       '001002':  '000000002'                  2;
       '001003':  '000000003'                     3;
       '001004':  '000000004'                        4;
       '001005':  '000000005'                           5
    13 '001006':  '000177777'  END OF MEMORY;  ('200 000' - 1)
    14       2                        'BEGIN' ZZ = M9,YY = MA,XX = MB,WW = Me,VV = MF,
    15                                       UU = MB,TT = M23[45],SS = M58[10 - 55]
    16 '175700':  '000000001'          M[BB - '100']:  1
    17 '175400':  '000000002'          M[BB - '400']:  2
    18 '175100':  '000000003'          M[BB - '700']:  3
    19 '174400':  '000000004'          M[BB - '1400']: 4
    20 '174100':  '000000005'          M[BB - '1700']: 5
    21                                 "NOTE THAT THE BEGIN-ADDRESS OF THE LOADER IS BB AND THAT ITS
    22                                 "LENGTH IS '425' INSTRUCTIONS.
    23       2                         'END'
    24       3                         'BEGIN' UU = :AA,TT = 22,SS = (:AA + :A -:AA)
    25 '176350':  '777777776'          M[BB + '350']: -1;
       '176351':  '777777775'                             -2;
       '176352':  '777777774'                                -3;
       '176353':  '777777773'                                   -4;
       '176354':  '777777772'                                      -5
    26                                 "THE LOADER IS NOW SHIFTED OVER '55'
    27                                 " PLACES, AND THE MACHINE IS INSPECTED TO CHECK THAT
    28                                 " THE END-ADDRESS OF THE LOADER IS INDEED
    29                                 "       (BB + '150' - 1) = '176 347'.
    30                                 "THE TECHNIQUE TO FORCE THE LOADER TO STOP IS BY MEANS
    31                                 " OF AN END-OF-TAPE SITUATION.
    32       4                         'BEGIN' UU = :MG,TT = :MA,SS = :MC
    33 '176150':  '777777776'          M[BB + '150']: -1;
       '176151':  '777777775'                             -2;
       '176152':  '777777774'                                -3;
       '176153':  '777777773'                                   -4;
       '176154':  '777777772'                                      -5
    34                                 "THE LOADER IS NOW SHIFTED TO THE FIRST FREE TRAJECT,
    35                                 " ITS END-ADDRESS IS NOW (BB - '700' - 1) = '175 077'
    36                                 " ONCE MORE THIS HAS TO BE INSPECTED.
    37       4                         'END'
    38 '000030':  '522000031'  M[24]:    0000 (25)
    39 '000031':  '600100070'                 G = M[56];
       '000032':  '600100077'                    G + M[63];
       '000033':  '600100123'                       G + M['123'];
       '000034':  '600100124'                          G + M['124']
```

```
   40 '000035':  '600100127'                    G + M['127'];
      '000036':  '600100130'                              G + M['130'];
      '000037':  '600100330'                                          G + M['330']
   41 '000040':  '420001006'                    B = END OF MEMORY
   42 '000041':  '606175377'                    G + M[-1]
   43 '000042':  '412501000'                 U, B = 512, P
   44 '000043':  '512200003'                 V, JUMP(-3)
   45 '000044':  '611101006'                    G = END OF MEMORY, Z
   46 '000045':  '032200001'                 V, A = -1
   47 '000046':  '032300002'                 N, A = -2     "A FORBIDDEN SITUATION.
   48 '000047':  '026173400'                 U, A = MA     "THE V8 STOPS.
   49       3                 'END'   'END
            1
```

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| AA | M['001001'] | 1 | 4 | +12 | 24 | 24 | 24 | | | |
| BB | '176000' | 1 | +4 | 16 | 17 | 18 | 19 | 20 | 25 | 33 |
| ENDOFMEMORY | M['001006'] | 1 | 4 | +13 | 41 | 45 | | | | |
| SS | M[B+'040000'] | 1 | +3 | | | | | | | |
| | MG[-45] | 2 | +15 | | | | | | | |
| | '000073' | 3 | +24 | | | | | | | |
| | :MC[0] | 4 | +32 | | | | | | | |
| TT | M[B+'040000'] | 1 | +3 | | | | | | | |
| | M23[45] | 2 | +15 | | | | | | | |
| | '000026' | 3 | +24 | | | | | | | |
| | :MA[0] | 4 | +32 | | | | | | | |
| UU | M[B+'040000'] | 1 | +3 | | | | | | | |
| | MD[0] | 2 | +15 | | | | | | | |
| | '001001' | 3 | +24 | | | | | | | |
| | :MG[0] | 4 | +32 | | | | | | | |
| VV | '000000' | 1 | +2 | 3 | 3 | | | | | |
| | MT[0] | 2 | +14 | | | | | | | |
| WW | M['000076'] | 1 | +2 | | | | | | | |
| | MC[0] | 2 | +14 | | | | | | | |
| XX | M['000000'] | 1 | +2 | 2 | 2 | | | | | |
| | MS[0] | 2 | +14 | | | | | | | |
| YY | M['000000'] | 1 | +2 | 3 | 3 | | | | | |
| | MA[0] | 2 | +14 | | | | | | | |
| ZZ | M['000000'] | 1 | +2 | 2 | 2 | 2 | 3 | | | |
| | MG[0] | 2 | +14 | | | | | | | |