

RA

stichting
mathematisch
centrum



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

BIBLIOTHEEK MATHEMATISCH CENTRUM
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	1
1. The general outline	3
2. The syntactic structure of ELAN	7
3. Reading equipment and Error recovery	10
3.1. The handling of syntactical errors	11
3.2. The error list	14
4. Name list organization	18
4.1. Preliminary remarks	18
4.2. The data structure chosen	18
5. The code produced	22
6. Printer output	26
Appendix A	29
Appendix B	111

0. Introduction

The original ELAN assembler, written in ELAN, had, at the time the first thoughts about the MC ELAN assembler came up, a number 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 chapter 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 process 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 xxxxxx xxxxxx xxxxxx

"40, YN, MULAS(-x), PZE; :DYN;" 010 001 xx xx xx xxxxxx xxxxxx xxxxxx

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 initialization.

(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 ".₂" stands for a simple semicolon ";", which could not be used in the comment part of the ALGOL 60 program.

The syntactic structure <nldr> 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>::= <nldr> | ; | <comment> <nldr> | <comment> ;
  <comment>::= "<one line of symbols not containing <nldr> 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>
  <intialization>::= <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> <functional 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 undesirable 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 'x';
 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.

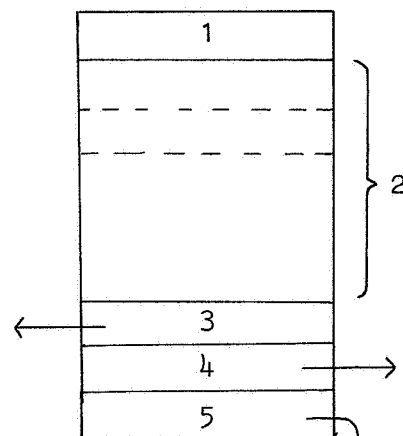
4.2. The data structure chosen

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 occurrence of every identifier. Every name in the name list has two pointers, one to a name that precedes 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

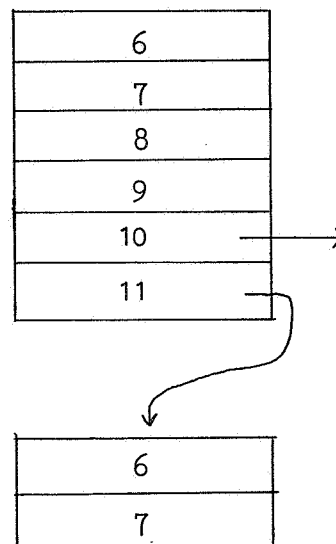
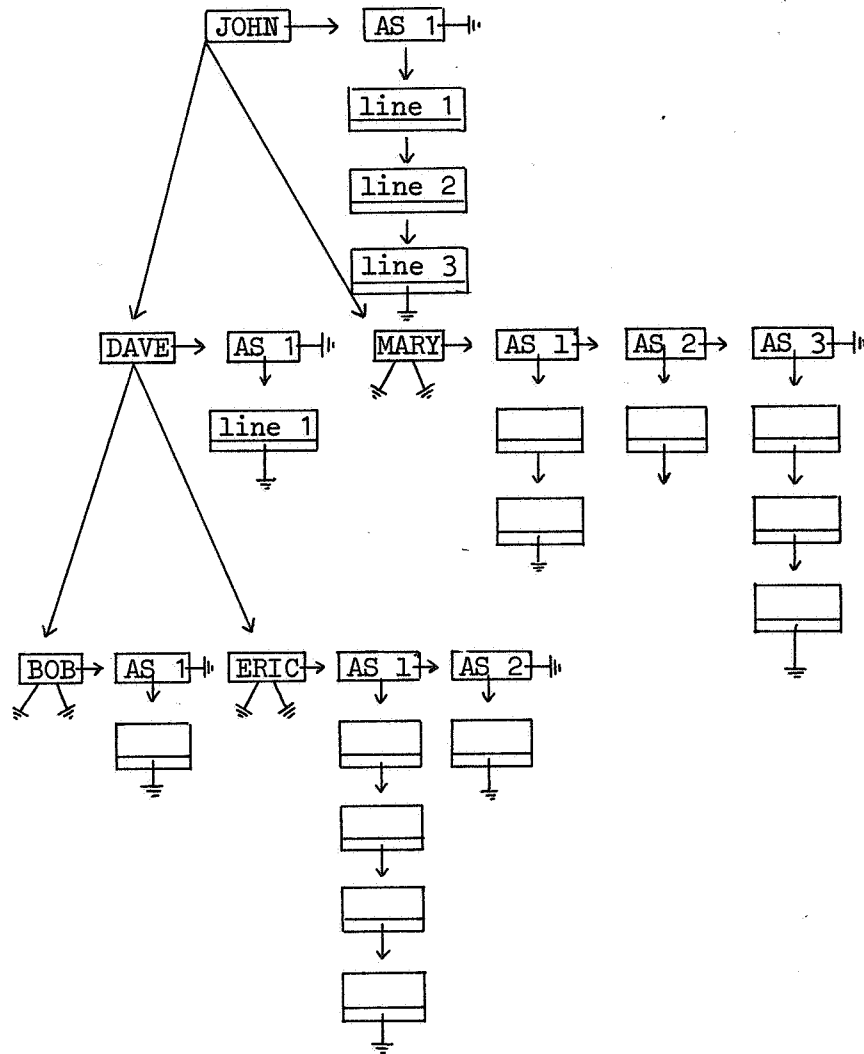


fig. 1. The administration in "contents of".

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: $d_{26} d_{25} \dots d_1 d_0$. They are preceded by one bit 1, and then divided into 4 parts:

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

. . .

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

in which order they occur as heptads on punched tape (a 1 corresponds with a hole).

Each BI word consists of 27 bits: $d_{26} d_{25} \dots d_1 d_0$. They are preceded by three bits: $P C_1 C_2$ and then divided into 5 parts:

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

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

. . .

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

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

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

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

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

the block consists of another T words $w_1, \dots, w_T$, which have to be stored in memory locations $M[A+1], \dots, M[A+T]$; 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] Programming 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 occurrences in the text.

For this listing a program of L. Meertens has been used.

```

1  begin comment MC ELAN1 assembler and definition, 010471. R 2172s.
2
3  1. ELAN program.
4
5  <ELAN program> ::= <ELAN block> | <location>: <ELAN block>
6
7  2. ELAN block.
8
9  <ELAN block> ::= <block begin> <compound tail>
10 <block begin> ::= 'begin' <ELAN declaration>
11 <statement separator> ::= <nocr> | .2 | <comment> <nocr> |
12                               <comment> .2
13 <comment> ::= "<one line of symbols not containing <nocr> or .2>"
14 <compound tail> ::= 'end' |
15                     <ELAN statement> <statement separator> <compound tail>
16
17 Algorithm;
18
19 procedure RE ELAN block;
20 begin block number := max block number := max block number + 1;
21   cnt of begins := cnt of begins + 1;
22   block[cnt of begins] := line buffer[-1] := block number;
23   RE block begin;
24 st sep: if synt unit = statement separator then
25   begin RE through barrier; goto st sep end;
26   if synt unit ≠ end symbol then
27   begin RE ELAN statement; goto st sep end;
28   if block number > 2 then
29   begin if line buffer[-1] > 2 then PRINT LINE;
30     line buffer[-1] := block number
31   end;
32   cnt of begins := cnt of begins - 1;
33   block number := block[cnt of begins];
34   RE through barrier
35 end RE ELAN block;
36
37 procedure RE block begin;
38 begin CHECK(begin symbol, 210);
39   RE poss decl id (false);
40   RE ELAN declaration
41 end RE block begin;
42
43 procedure RE poss decl id (stat sep allowed);
44 value stat sep allowed; Boolean stat sep allowed;
45 begin comment The variable "declaration" is used to let the
46   "search for identifier" process of section 9 know that a
47   declaration is treated.
48   Due to the definition of <comma> several statement separators
49   may be read, for which "RE through barrier" has to be called.
50   This situation is indicated by the value of "stat sep allowed".
51   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(1 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 linenummer 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  $\neq$  STAT operand,305)
107         end;
108         if synt unit  $\neq$  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(7 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  $\neq$  MT declared and defined,315);
127         RE;
128         if synt unit  $\neq$  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 7 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  $\wedge$  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  $\neq$  declared  $\wedge$ 
165             type of identifier  $\neq$  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  $\times$  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  $\vee$  synt unit = identifier)  $\wedge$ 
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  $\neq$  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  $0 \leq v \leq 31$  or  $0 \leq v \leq 1$  or  $0 \leq v \leq 39$ ,
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' | 'x'-
236 <unsigned number> ::= <unsigned integer> | <unsigned real>
237 <unsigned integer> ::= <unsigned decimal> | '<unsigned octal>'
238

```

Algorithm::

```

240
241 procedure RE ELAN statement;
242 begin RE marginal part; RE ELAN instruction;
243   if synt unit  $\neq$  statement separator  $\wedge$ 
244     synt unit  $\neq$  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  $\vee$  is adding operator(synt unit) then
250   begin Boolean neg;
251     neg := synt unit = minus symbol;
252     if synt unit  $\neq$  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 1(synt unit = statement separator V
271       synt unit = end symbol) then
272   begin if 1 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 V 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(1 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(1 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:= 1 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 V
319    left operand = DYN ad op then left operand:=ERROR(true,550);
320    ERROR(1 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(1 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          7(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:= 7 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  $\neq$  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  $\vee$  p > 63, 635); ERROR(-256 > q  $\vee$  q > 255, 640);
445   value of operand := p  $\times$  512 + 256 + q
446 end else
447 if synt unit = M symbol  $\vee$  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  $\vee$  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  $\neg$  initialization  $\wedge$  stat sep barrier  $\wedge$ 
575 (synt unit = statement separator  $\vee$  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)  $\vee$  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  $\times$  t6 + symbol + 1; NS;
601 end :=  $\neg$ (is letter(symbol)  $\vee$  is digit(symbol));
602 if n = n : 4  $\times$  4  $\vee$  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  $\neg$  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(7 is digit(symbol),870); length:= 10;
703     again: NS; if is digit(symbol) then
704         begin i:=iX10 + 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 then
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 1 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 1 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 1 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(endresymcount, ↵
830   'end'
831   ↵);
832   if sym = nlcr symbol ∧ endresymcount = 0 then ERROR(true, 874);
833   endresymcount := if sym = nlcr symbol ∧ endresymcount > 1 then 0 else
834   endresymcount + 1;
835   endstringsym := sym
836   end endstringsym;
837
838   boolean procedure rehep available;
839   rehep available := 1 first scan = second scan;
840
841   comment:

```

9. Name list equipment and secondary storage usage.

The procedure STORE letgits with (block number) takes care that each identifier of the ELAN source text is entered into the name list. It uses the procedures "STORE in contents of", "ST", "store in LINE" and "st" for storing variable amounts of information into the arrays "contents of" and "LINE".

The strategy is:

All incoming information is stored linearly in the array "contents of", and pointers connect the names alphabetically together. In the lower part of this array, the special identifiers are stored.

The administration section for each identifier i is as follows:

- (a) number of memory places of i.
- (b) the letters and/or digits of i are stored in groups of four letters/digits into one memory place.
- (c) the pointer to the next name, which preceeds alphabetically i.
- (d) the pointer to the next name, which follows alphabetically i.
- (e) the pointer to the administration section of i.
- (f) block number.
- (g) type.
- (h) value.
- (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  $\neg$ declaration  $\wedge$  block number  $\neq$  1  $\wedge$ 
894 block number  $\neq$  2  $\wedge$  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  $\neq$  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 7 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 x 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 x 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 x t8 x t8
1065 end;
1066 t8i:= t8i/t8; s:= triple : t8i; if s = space symbol then
1067 begin triple:= triple - s x 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  $\neq$  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  $\neq$  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  $\times$  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))  $\times$  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      .,.,:,'",,;(,),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, REPOZ, REP2Z, REP4Z,
1226 REP6Z, REPOE, REP1E, REP2E, REP3E, REP4E, REP5E, REP6E, REP7E, REPO, REP1,
1227 REP2, REP3, REP4, REP5, REP6, REP7, REPOP, 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 xxxxxx xxxxxx xxxxxx
1239 1.1,    UYN, A+:DYN,   PZE;      "000 100 01 xx xx xxxxxx xxxxxx xxxxxx
1240 2,      UYN, A-x,      PZE;      "000 001 xx xx xx xxxxxx xxxxxx xxxxxx
1241 2.1,    UYN, A=:DYN,   PZE;      "000 101 01 xx xx xxxxxx xxxxxx xxxxxx
1242 3,      UYN, A=x,      PZE;      "000 010 xx xx xx xxxxxx xxxxxx xxxxxx
1243 3.1,    UYN, A=:DYN,   PZE;      "000 110 01 xx xx xxxxxx xxxxxx xxxxxx
1244 4,      UYN, A=x,      PZE;      "000 011 xx xx xx xxxxxx xxxxxx xxxxxx
1245 4.1,    UYN, A=:DYN,   PZE;      "000 111 01 xx xx xxxxxx xxxxxx xxxxxx
1246 5,      UYN, x+A,      PZE;      "000 100 xx xx xx xxxxxx xxxxxx xxxxxx
1247 6,      UYN, x-A,      PZE;      "000 101 xx xx xx xxxxxx xxxxxx xxxxxx
1248 7,      YN,  x=A,      PZE;      "000 110 xx xx xx xxxxxx xxxxxx xxxxxx
1249 8,      YN,  x=A,      PZE;      "000 111 xx xx xx xxxxxx xxxxxx xxxxxx
1250 9,      PLUSA(x), PZE; :STAT, :DYN; "000 110 xx xx 01 xxxxxx xxxxxx xxxxxx
1251 10,     MINA(x),  PZE; :STAT, :DYN; "000 111 xx xx 01 xxxxxx xxxxxx xxxxxx
1252
1253 11,     UYN, S+x,      PZE;      "001 000 xx xx xx xxxxxx xxxxxx xxxxxx
1254 11.1,   UYN, S+:DYN,   PZE;      "001 100 01 xx xx xxxxxx xxxxxx xxxxxx
1255 12,     UYN, S-x,      PZE;      "001 001 xx xx xx xxxxxx xxxxxx xxxxxx
1256 12.1,   UYN, S=:DYN,   PZE;      "001 101 01 xx xx xxxxxx xxxxxx xxxxxx
1257 13,     UYN, S=x,      PZE;      "001 010 xx xx xx xxxxxx xxxxxx xxxxxx
1258 13.1,   UYN, S=:DYN,   PZE;      "001 110 01 xx xx xxxxxx xxxxxx xxxxxx
1259 14,     UYN, S=x,      PZE;      "001 011 xx xx xx xxxxxx xxxxxx xxxxxx
1260 14.1,   UYN, S=:DYN,   PZE;      "001 111 01 xx xx xxxxxx xxxxxx xxxxxx
1261 15,     UYN, x+S,      PZE;      "001 100 xx xx xx xxxxxx xxxxxx xxxxxx
1262 16,     UYN, x-S,      PZE;      "001 101 xx xx xx xxxxxx xxxxxx xxxxxx
1263 17,     YN,  x=S,      PZE;      "001 110 xx xx xx xxxxxx xxxxxx xxxxxx
1264 18,     YN,  x=S,      PZE;      "001 111 xx xx xx xxxxxx xxxxxx xxxxxx
1265 19,     PLUS(x), PZE; :STAT, :DYN; "001 110 xx xx 01 xxxxxx xxxxxx xxxxxx
1266 20,     MINS(x),  PZE; :STAT, :DYN; "001 111 xx xx 01 xxxxxx xxxxxx xxxxxx
1267
1268 21,     UYN, B+x,      PZE;      "100 000 xx xx xx xxxxxx xxxxxx xxxxxx
1269 21.1,   UYN, B+:DYN,   PZE;      "100 100 01 xx xx xxxxxx xxxxxx xxxxxx
1270 22,     UYN, B-x,      PZE;      "100 001 xx xx xx xxxxxx xxxxxx xxxxxx
1271 22.1,   UYN, B=:DYN,   PZE;      "100 101 01 xx xx xxxxxx xxxxxx xxxxxx
1272 23,     UYN, B=x,      PZE;      "100 010 xx xx xx xxxxxx xxxxxx xxxxxx
1273 23.1,   UYN, B=:DYN,   PZE;      "100 110 01 xx xx xxxxxx xxxxxx xxxxxx

```

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

1325	051,	YN,	GX:STAT,	PZE;	"111 000 01 xx xx	xxxxxx	xxxxxx	xxxxxx
1326	60,		G/x,	PZE;:DYN;	"111 001 xx xx 01	xxxxxx	xxxxxx	xxxxxx
1327	052,	YN,	G/:STAT,	PZE;	"111 001 01 xx xx	xxxxxx	xxxxxx	xxxxxx
1328	61,		x=G,	PZE;	"111 010 xx xx 01	xxxxxx	xxxxxx	xxxxxx
1329	62,		x=-G,	PZE;	"111 011 xx xx 01	xxxxxx	xxxxxx	xxxxxx
1330								
1331	63,	UYN,	GOTO(x);		"101 010 xx 00 xx	xxxxxx	xxxxxx	xxxxxx
1332	63.1,	UYN,	GOTO(:DYN);		"101 011 11 00 xx	xxxxxx	xxxxxx	xxxxxx
1333	64,	UYN,	JUMP(x); :DYN;		"101 000 xx 00 xx	xxxxxx	xxxxxx	xxxxxx
1334	65,	UYN,	JUMP(-x); :DYN;		"101 001 xx 00 xx	xxxxxx	xxxxxx	xxxxxx
1335	66,	UYN,	GOTOR(x);		"101 010 xx 01 xx	xxxxxx	xxxxxx	xxxxxx
1336	66.1,	UYN,	GOTOR(:DYN);		"101 011 11 01 xx	xxxxxx	xxxxxx	xxxxxx
1337	67,	UYN,	JUMPR(x); :DYN;		"101 000 xx 01 xx	xxxxxx	xxxxxx	xxxxxx
1338	68,	UYN,	JUMPR(-x); :DYN;		"101 001 xx 01 xx	xxxxxx	xxxxxx	xxxxxx
1339								
1340	69,	UYN,	REP(:STAT);		"101 101 11 00 xx	xxxxxx	xxxxxx	xxxxxx
1341	70,	UYN,	REPP(:STAT);		"101 101 11 01 xx	xxxxxx	xxxxxx	xxxxxx
1342	71,	UYN,	REPE(:STAT);		"101 101 11 11 xx	xxxxxx	xxxxxx	xxxxxx
1343	72,	UYN,	REPZ(:STAT);		"101 101 11 10 xx	xxxxxx	xxxxxx	xxxxxx
1344								
1345	73,	UYN,	SUB(:STAT);		"101 110 00 00 xx	xxxxxx	xxxxxx	xxxxxx
1346	74,	UYN,	SUBC(x);		"101 110 xx 01 xx	xxxxxx	xxxxxx	xxxxxx
1347	74.1,	UYN,	SUBC(:DYN);		"101 111 01 01 xx	xxxxxx	xxxxxx	xxxxxx
1348								
1349	76,	UYN,	DO(x); :STAT,:DYN;		"101 111 xx 01 xx	xxxxxx	xxxxxx	xxxxxx
1350	77,	UYN,	DOS(x); :STAT,:DYN;		"101 111 xx 11 xx	xxxxxx	xxxxxx	xxxxxx
1351								
1352	78,	YN,	LCA(1),	PZE;	"110 110 x0 xx xx	00000	00000	xxxxxx
1353	79,	YN,	LCS(1),	PZE;	"111 110 x0 xx xx	00000	00000	xxxxxx
1354	80,	YN,	LCAS(1),	PZE;	"110 110 x0 xx xx	00000	00010	xxxxxx
1355	81,	YN,	LCSA(1),	PZE;	"111 110 x0 xx xx	00000	00010	xxxxxx
1356	82,	YN,	RCA(1),	PZE;	"110 111 x0 xx xx	00000	00000	xxxxxx
1357	83,	YN,	RCS(1),	PZE;	"111 111 x0 xx xx	00000	00000	xxxxxx
1358	84,	YN,	RCAS(1),	PZE;	"110 111 x0 xx xx	00000	00010	xxxxxx
1359	85,	YN,	RCSA(1),	PZE;	"111 111 x0 xx xx	00000	00010	xxxxxx
1360								
1361	86,	YN,	LUA(1),	PZE;	"110 110 x0 xx xx	00000	00001	xxxxxx
1362	87,	YN,	LUS(1),	PZE;	"111 110 x0 xx xx	00000	00001	xxxxxx
1363	88,	YN,	LUAS(1),	PZE;	"110 110 x0 xx xx	00000	00011	xxxxxx
1364	89,	YN,	LUSA(1),	PZE;	"111 110 x0 xx xx	00000	00011	xxxxxx
1365	90,	YN,	RUA(1),	PZE;	"110 111 x0 xx xx	00000	00001	xxxxxx
1366	91,	YN,	RUS(1),	PZE;	"111 111 x0 xx xx	00000	00001	xxxxxx
1367	92,	YN,	RUAS(1),	PZE;	"110 111 x0 xx xx	00000	00011	xxxxxx
1368	93,	YN,	RUSA(1),	PZE;	"111 111 x0 xx xx	00000	00011	xxxxxx
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 xxxxxx xxxxxx xxxxxx
1381	101.1,	UYN,	REP1(:STAT);	"101	100	01	00	xx xxxxxx xxxxxx xxxxxx
1382	101.2,	UYN,	REP2(:STAT);	"101	100	10	00	xx xxxxxx xxxxxx xxxxxx
1383	101.3,	UYN,	REP3(:STAT);	"101	100	11	00	xx xxxxxx xxxxxx xxxxxx
1384	101.4,	UYN,	REP4(:STAT);	"101	101	00	00	xx xxxxxx xxxxxx xxxxxx
1385	101.5,	UYN,	REP5(:STAT);	"101	101	01	00	xx xxxxxx xxxxxx xxxxxx
1386	101.6,	UYN,	REP6(:STAT);	"101	101	10	00	xx xxxxxx xxxxxx xxxxxx
1387	101.7,	UYN,	REP7(:STAT);	"101	101	11	00	xx xxxxxx xxxxxx xxxxxx
1388								
1389	102.0,	UYN,	REPOP(:STAT);	"101	100	00	01	xx xxxxxx xxxxxx xxxxxx
1390	102.1,	UYN,	REP1P(:STAT);	"101	100	01	01	xx xxxxxx xxxxxx xxxxxx
1391	102.2,	UYN,	REP2P(:STAT);	"101	100	10	01	xx xxxxxx xxxxxx xxxxxx
1392	102.3,	UYN,	REP3P(:STAT);	"101	100	11	01	xx xxxxxx xxxxxx xxxxxx
1393	102.4,	UYN,	REP4P(:STAT);	"101	101	00	01	xx xxxxxx xxxxxx xxxxxx
1394	102.5,	UYN,	REP5P(:STAT);	"101	101	01	01	xx xxxxxx xxxxxx xxxxxx
1395	102.6,	UYN,	REP6P(:STAT);	"101	101	10	01	xx xxxxxx xxxxxx xxxxxx
1396	102.7,	UYN,	REP7P(:STAT);	"101	101	11	01	xx xxxxxx xxxxxx xxxxxx
1397								
1398	103.0,	UYN,	REPOZ(:STAT);	"101	100	00	10	xx xxxxxx xxxxxx xxxxxx
1399	103.1,	UYN,	REP1Z(:STAT);	"101	100	01	10	xx xxxxxx xxxxxx xxxxxx
1400	103.2,	UYN,	REP2Z(:STAT);	"101	100	10	10	xx xxxxxx xxxxxx xxxxxx
1401	103.3,	UYN,	REP3Z(:STAT);	"101	100	11	10	xx xxxxxx xxxxxx xxxxxx
1402	103.4,	UYN,	REP4Z(:STAT);	"101	101	00	10	xx xxxxxx xxxxxx xxxxxx
1403	103.5,	UYN,	REP5Z(:STAT);	"101	101	01	10	xx xxxxxx xxxxxx xxxxxx
1404	103.6,	UYN,	REP6Z(:STAT);	"101	101	10	10	xx xxxxxx xxxxxx xxxxxx
1405	103.7,	UYN,	REP7Z(:STAT);	"101	101	11	10	xx xxxxxx xxxxxx xxxxxx
1406								
1407	104.0,	UYN,	REPOE(:STAT);	"101	100	00	11	xx xxxxxx xxxxxx xxxxxx
1408	104.1,	UYN,	REP1E(:STAT);	"101	100	01	11	xx xxxxxx xxxxxx xxxxxx
1409	104.2,	UYN,	REP2E(:STAT);	"101	100	10	11	xx xxxxxx xxxxxx xxxxxx
1410	104.3,	UYN,	REP3E(:STAT);	"101	100	11	11	xx xxxxxx xxxxxx xxxxxx
1411	104.4,	UYN,	REP4E(:STAT);	"101	101	00	11	xx xxxxxx xxxxxx xxxxxx
1412	104.5,	UYN,	REP5E(:STAT);	"101	101	01	11	xx xxxxxx xxxxxx xxxxxx
1413	104.6,	UYN,	REP6E(:STAT);	"101	101	10	11	xx xxxxxx xxxxxx xxxxxx
1414	104.7,	UYN,	REP7E(:STAT);	"101	101	11	11	xx xxxxxx xxxxxx xxxxxx
1415								
1416	105. 0,	UYN,	SUB0(:STAT);	"101	110	00	00	xx xxxxxx xxxxxx xxxxxx
1417	105. 1,	UYN,	SUB1(:STAT);	"101	110	01	00	xx xxxxxx xxxxxx xxxxxx
1418	105. 2,	UYN,	SUB2(:STAT);	"101	110	10	00	xx xxxxxx xxxxxx xxxxxx
1419	105. 3,	UYN,	SUB3(:STAT);	"101	110	11	00	xx xxxxxx xxxxxx xxxxxx
1420	105. 4,	UYN,	SUB4(:STAT);	"101	111	00	00	xx xxxxxx xxxxxx xxxxxx
1421	105. 5,	UYN,	SUB5(:STAT);	"101	111	01	00	xx xxxxxx xxxxxx xxxxxx
1422	105. 6,	UYN,	SUB6(:STAT);	"101	111	10	00	xx xxxxxx xxxxxx xxxxxx
1423	105. 7,	UYN,	SUB7(:STAT);	"101	111	11	00	xx xxxxxx xxxxxx xxxxxx
1424	105. 8,	UYN,	SUB8(:STAT);	"101	110	00	10	xx xxxxxx xxxxxx xxxxxx
1425	105. 9,	UYN,	SUB9(:STAT);	"101	110	01	10	xx xxxxxx xxxxxx xxxxxx
1426	105.10,	UYN,	SUB10(:STAT);	"101	110	10	10	xx xxxxxx xxxxxx xxxxxx

1427	105.11, UYN,	SUB11(:STAT);	"101 110 11 10 xx	xxxxxx xxxxxx xxxxxx
1428	105.12, UYN,	SUB12(:STAT);	"101 111 00 10 xx	xxxxxx xxxxxx xxxxxx
1429	105.13, UYN,	SUB13(:STAT);	"101 111 01 10 xx	xxxxxx xxxxxx xxxxxx
1430	105.14, UYN,	SUB14(:STAT);	"101 111 10 10 xx	xxxxxx xxxxxx xxxxxx
1431	105.15, UYN,	SUB15(:STAT);	"101 111 11 10 xx	xxxxxx xxxxxx xxxxxx
1432				
1433	106,	UYN, SUBCD(x);	"101 110 xx 11 xx	xxxxxx xxxxxx xxxxxx
1434	106.1,	UYN, SUBCD(:DYN);	"101 111 01 11 xx	xxxxxx xxxxxx xxxxxx
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 xxxxxx
1446	116,	YN, LVIFOFF(1);	"110 110 x0 00 xx	11101 0000x xxxxxx
1447	117,	YN, IFON(1);	"111 110 x0 00 xx	11100 1000x xxxxxx
1448	118,	YN, IFOFF(1);	"110 110 x0 00 xx	11100 1000x xxxxxx
1449	119,	YN, AFON(1);	"111 110 x0 00 xx	11100 0000x xxxxxx
1450	120,	YN, AFOFF(1);	"110 110 x0 00 xx	11100 0000x xxxxxx
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 1 (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  $\neq$  0, 1045); i:= abs (r - q  $\times$  real t26);
1540   real to int:=(if entier(q/t1) $\times$ t1=q then i else i+t26) $\times$ 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  $\wedge$  (n - 1); if x > p8  $\times$  8 then
1570   begin ERROR (true, 1065); x:= x - x : (p8  $\times$  8)  $\times$  (p8  $\times$  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  $\neq$  0 do
1577   begin digit:= x : p8; x:= x - digit  $\times$  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 1 initialization  $\wedge$  cntr  $\neq$  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)xt18+address old+t26-1,true,true) else
1641     single BI(cntr1  $\times$  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  $\vee$  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 $\neq$ 1  $\wedge$  i $\neq$ 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  $\vee$  s = minus symbol;
1663
1664 boolean procedure is PZE symbol(s); value s; integer s;
1665 is PZE symbol:= s = P symbol  $\vee$  s = Z symbol  $\vee$ 
1666   s = E symbol  $\vee$  s = PZE symbol;
1667
1668 boolean procedure is UYN symbol(s); value s; integer s;
1669 is UYN symbol:= s = U symbol  $\vee$  s = Y symbol  $\vee$  s = N symbol  $\vee$ 
1670   s = YN symbol  $\vee$  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 V
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, 'x');
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         112?4??78??7?? ? ??3?56??9 ?????0??t?vw??z1
1794         ?????<s?u?xy??10?? ?-??l?no??r??????jk?m?
1795         ?pq??,????ab?d??gh??.????+??c?ef??i1?7??7
1796
1797         1VX?=? ?](??7??|? ??/?;[??) ?????^??T?VW??Z1
1798         ?????>S?U??XY??'?? ?7??L?NO??R??????JK?M?
1799         ?PQ? ??????AB?D??GH??.????"??C?EF??I1?7??7⌋);
1800     symcount := symcount + 1;
1801     if sym = nlcr V sym = tab then goto L;
1802     if sym = underline V 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 V s = tab symbol V
1827     s = nlcr symbol V 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      t1 :=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 7 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:= 7(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 7 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(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(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 V type= 5 then
2100      begin integer p,q,qq;
2101      p:= value : 512; q:= value -512xp-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-qqx10))
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-qqx10),
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-qqx100): 10),
2134      q-q: 10x10),
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 PUTTEXT('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 1 write V 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; linewidth:= -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     linewidth:= -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 V 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

```

80 a 80 85 90 91 2x96 2x100
 170 a 170 171 3x172 2x173
 493 a 2x493 494
 496 a 2x496 497
 499 a 499 2x500 507
 503 a 3x503 504 505
 1754 a 1754 2x1758 2x1759 2x1760 1761 2x1762 1763 1764
 597 A 597 599 2x600 2x604 635 2x636
 1674 A 1674 3x1681 2x1682
 1906 A 1906 1911 2x1912 2x1917 1922 1933 1936
 2027 A 2027 2148 2181 2190 2191
 - abs 1098 1514 1538 1539 1568
 - ABSFIXP 2028 2206
 - ABSFIXT 1725 1728 1982 1989 2028
 1877 addresscounter
 166 186 190 454 1551 2x1647 1648 1649 2x1652 1877
 1976 2222
 1589 addressold
 1551 1589 1609 1611 1612 1638 1640 1641
 1646 Addtoadcentr
 193 259 267 274 278 293 1560 1646

76

954 aftercomparison
947 950 954

156 again
156 177

527 again
527 542

600 again
600 606

684 again
684 687

703 again
703 704

748 again
748 749 754 758

1138 again
1138 1140

1151 again
1151 1152

1779 apostrophesymbol
653 669 676 689 691 751 753 762 1572 1573
1584 1585 1779 1843 1901 2078 2081

2055 ar
2055 2060 2089 2094 2138

1177 ASSIGN
1177 1190 1197 1203

1775 Asymbol
435 610 618 1681 1775 1837 2104

1018 aux
1018 1021

1018 auxiliariypointer
1018 1021 1023

493 b
2x493 494

496 b
2x496 497

499 b
499 2x500 507

503 b
3x503 504 505

1177 B
1177 1178 1179

1722 B
3x1722 1723

1786 bar
1786 1802 1807 1810

1782 beginsymbol
38 269 579 658 1758 1782 1848 2223

1782 BIsymbol
261 659 1782 1852

1861 block
22 33 990 1136 1861 2231

2052 block
2052 2151 2x2184 2x2190

1865 blocknumber
20 22 28 30 33 649 1134 1136 1865

878 blocknumber
878 2x879 893 894 909 916 918 926 968 981
994

882 blocknumberintree
882 975 977 981 990 998

1775 Bsymbol
345 459 610 1682 1775 1838 2069

1051 buffer
1051 1060 1063 1080 1087 1096 1103

1102 BUFFERFROMDRUM
1095 1102

1049 bufferptr
1049 1056 1060 1063 2x1064 2x1078 1079 1080 1088 2x1094
1095 1096 1104 2234 2243

78

1085 BUFFERTODRUM

1080 1085 2238

1781 bussymbol

194 442 463 466 750 780 798 1781 1845 2082

1617 c1

1617 2x1618 1630

1617 c2

1617 2x1618 1630

567 casecode

567 818 820 821 1813 1815 1820

2052 cc

2052 2155 3x2173 2179

1732 CHECK

38 85 117 253 282 2x1138 1141 1149 1732 1734
1735 1742

1720 CHECKfault

113 132 2x577 1720 1734 1738 1742 1747 2221

1780 closesymbol

347 368 428 1780 1844 1845 1923

1589 cntr

3x1550 1551 1552 1589 1606 2x1609 1610 1613 1614 1635
2x1638 1639 1642 1643 2222

1607 cntr1

1607 1609 1611 1612

1636 cntr1

1636 1638 1640 1641

1865 cntrofbegin

2x21 22 2x32 33 989 2x1135 1136 2x1141 1865 2230
2231 2240

812 code

812 814 821 822 823 824

1779 colonsymbol

156 185 276 414 1779 1842 1902 2074

1774 commasymbol

109 110 129 130 282 287 1138 1152 1774 1827
1835

939 COMPARE
939 972

1499 cond
1499 1501 3x1502 1503 2x1504 1505 1506

1499 condforminus
1499 1506 1511

1499 condforPZE
1499 1505 1506 1510

1499 condforUYN
1499 1503 1504 2x1509

873 contentsof
90 91 2x96 2x100 119 120 161 163 166 171
604 607 648 873 888 892 897 898 937 940
943 944 956 959 975 984 991 993 997 999
1022 1896 1917 1920 2x1921 2039 2046 2047 2151 2152
2153 2154 2156 2192 2197 2200

1049 count3
1049 1055 1064 2x1068 1074 2x1076 1081 1093 1096 2x1099
2234 4x2237 2243

1775 Csymbol
437 618 1775 1837 2105

1881 d
1881 1882 7x1886 8x1887 8x1888 3x1889 1890

1145 d26
1145 1158 1183 1186

1543 d26
1543 2x1544 1546

1120 d26fctnop
375 1120 1205 1207

1116 d26INFWORD
314 331 375 1116 1524

1120 d26opopreg
331 1120 1192 1194

1119 d26regopop
314 1119 1199 1201

1862 declaration
53 57 893 968 980 1862 1908 1939 2219

1865 declared
100 160 164 452 472 890 914 1865 1884

1144 DEFINEINFWORDS
291 1144

1567 digit
1567 2x1577 3x1578 2x1580 1581

874 drumpointer
874 1028 1034 1035 1036 2x1037 1040 1041 1897 2161
2162 2163 2165 2248

1050 drumptr
1034 1040 1050 1057 2x1059 1060 1086 1087 2x1088 1103
2x1104 2235 2243

1775 Dsymbol
439 618 1775 1838 2106

568 dummycode
568 822 1803 1808

1177 dw1
1177 1178 1183

1177 dw2
1177 1178 1186

1114 DYNadop
319 418 1114 1131 1171

1783 DYNMsymbol
430 621 625 638 1759 1783 1848

1113 DYNoperand
418 432 471 473 1113 1130 1170 1181

524 el
524 526 2x534 535 537 538 539 541 544

1861 ELANline
776 794 1861

522 elevator
522 523 526 534 535 537 538 539 541 544
557

598 end
598 601 602 606

1683 end
 1677 1683

 1907 end
 1907 1914 1915 1919

 368 END
 350 368

 466 END
 463 466

 558 END
 558

 652 END
 613 622 626 631 639 643 645 652

 1163 END
 1156 1163

 875 endfctnpart
 875 907 976 1004 1896 1938 2048

 824 endresym
 814 824

 1071 endskip space
 1058 1071

 2016 endspace
 2016 2017 2018 2022 2023

 827 endstringsym
 814 827 835

 1782 endsymbol
 26 244 271 575 581 662 670 764 1141 1746
 1758 1782 1848

 569 endsymcount
 569 829 832 2x833 834 2227

 - entier
 1538 1540

 2003 entry
 2x2003 2208

 2033 entry1
 3x2033 2034 2039 2046 2047 2048 2197 2200

2050 entry2

3x2050 2051 2151 2152 2153 2154 2156 2192

2158 entry3

3x2158 2159 2161 2162 2164 3x2167 2x2168 2169 2170

1777 equalssymbol

87 1689 1709 1777 1840

1722 ERROR

81	86	106	118	126	159	164	188	189	245
288	300	310	319	320	328	356	363	370	419
422	2x444	450	452	460	469	477	553	603	660
666	669	676	685	687	689	695	702	754	757
793	816	823	832	1020	1040	1041	1086	1154	1159
1163	1171	1208	1507	1509	1510	1511	1513	1539	1568
1570	1649	1712	1722	1726	1735	1916			

569 errorcode

569 823 1804 1808

1777 Esymbol

610 644 1522 1666 1777 1840

- EVEN

1625

1765 exit

1756 1765

1532 expr

1532 1533

2030 F

2030 2181 2190 2191

1869 fctn

338	339	3x340	3x341	2x342	374	375	1204	1205	1206
1207	1869								

1115 fctninstruction

376 1115 1133 1202

1118 fctnop

374 1118 1129 1204 1206

1091 fetchfrombuffer

791 1091 1098

1875 fillpointer

1875 1957 1964 2x1967 1968 1969 1970 1996 2241

882 firstadministrationcell

882 887 888 892 897 898 974 975 985 991
993 996 997 999

1862 firstscan

86 89 99 117 158 272 787 839 980 1571
1579 1583 1727 1862 2229 2239

- FIXP

2031

- fixt

1648

- FIXT

2031

522 floor

3x522 523 526 528 530 534 535 537 538 539
541

- FROMDRUM

1060 1103 1559 2165

571 fromstring

571 805 807 2227

1777 Fsymbol

610 1681 1777 1840

1786 functionidentifier

285 1005 1764 1786 1852

1500 functionpart

1500 1502 1515

1775 Gsymbol

434 610 619 1681 1775 1837 2104

812 hep

812 813 815 3x816 817 819 821

1597 hep

1597 3x1600 3x1601 3x1602

1620 hep

1620 2x1623 1628 2x1629 1631

1789 hep

1789 1814 1815

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

1906 i
 1906 1908 2x1911 1933 1936 1938
 1980 i
 1980 2x1985 2x1993 1996 2x1997 1998
 2007 i
 2007 2038 2039 2205
 2054 i
 2054 2x2089 2x2094 2x2138 2141
 1558 I
 1558 1559 1560 1561
 700 i1
 700 2x710 713 714 722
 501 ia
 501 507 508 2x510 513 514 515
 501 ia2
 501 513 514 515
 501 ib
 501 507 508 2x510 513 514 515
 501 ib2
 501 513 514 515
 1785 identifier
 81 84 85 117 156 182 187 447 449 1005
 1762 1785 1851
 1705 indexofoperator
 303 305 323 325 1705 1706
 1868 indexofregister
 299 329 1676 1868
 1010 inf
 1010 1013
 1027 inf
 3x1027
 1008 information
 1008 1009 1013
 2056 information
 2056 2x2057

1114 INFWORD
 313 330 374 1114 2x1501 1502 1507

 1862 initialization
 272 291 574 629 641 727 773 789 1606 1635
 1862 2229 2233

 1122 INITIALIZEinfwords
 1122 2233

 1879 INITIALIZEothervariables
 1879 2228

 1788 INITIALIZERESYM
 1788 1834

 1823 INITIALIZESymbols
 1823 2228

 1214 Initstringsym
 805 1214 1215

 701 integer
 701 705 710 713 2x719

 1867 integraltype
 422 682 708 722 1528 1867 1885

 1590 IP
 262 264 1590 1594 1898 2239 2247

 1782 IPsymbol
 263 663 1782 1852

 1661 isaddingoperator
 249 301 321 718 1661 1662

 1695 isdigit
 585 601 624 2x637 678 687 698 702 703 710
 1695 1696 1914

 1698 islayout
 778 795 1698 1699

 1692 isletter
 585 596 601 654 668 1692 1693 1910 1914

 1686 isoperatorsymbol
 300 320 1686 1687

 1664 isPZESymbol
 288 1664 1665

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

2252 L
2252 2254

1908 L0
1908

85 L1
85 112

1909 L1
1909 1910 1937

117 L2
117 131

728 L2
2x728

1912 L2
1912 1919

734 L3
734 735

668 LA
2x668

1869 leftoperand
317 318 2x319 330 331 1190 1193 1194 1869

700 length
700 702 2x704 713

1781 letterb
656 1781 1845

1781 lettere
658 662 753 1781 1845

1781 letteri
659 663 1781 1846

1781 letterM
665 1781 1846

1781 letterS
664 1781 1846

2052 line
2052 2154 3x2191

2160 line
 2160 2169 2172 3x2181

873 LINE
 2x173 873 1036 1045 2165 2169 2170 2248

1861 linebuffer
 22 29 30 1581 1861 3x1900 1901 1902 3x1903 1904
 1955 1962 1968 1981 2x1982 1984 1985 1986 1988 1989
 1990 1992 1993 1994 2x1997 1998 2241 3x2242

1874 linecounter
 1874 2x1952 1953 1955 1957 1959 3x1960 1964 2222

1874 linenumber
 900 921 931 1728 1874 1950 2x1951 2x1955 2x1962 2230
 2241

- LINENUMBER
 1956 2175 2185

875 linepointer
 875 1028 2x1032 1033 1038 1045 1897

499 logicoperation
 494 497 499 517

1784 logicplussymbol
 529 536 674 1689 1710 1760 1784 1849

496 logicprod
 496 497 541

493 logicsum
 493 494 537

1784 logictimesymbol
 531 540 675 1690 1711 1760 1784 1850

747 LOOKAHEADsubtextbus
 184 747 749

567 lowercase
 567 817 1811

568 lowercasecode
 568 818 1812 1813 1816 1817 1820

715 LOWERTEN
 709 715

1779 lowertensymbol
 699 709 715 1779 1842

597 m
 597 2x607 648

1906 m
 1906 2x1920 1921

2027 m
 3x2027 2x2028

2030 m
 3x2030 2x2031

2007 marge
 2007 2176 2177 2178 2x2187 2188 3x2203 2206

1874 maxblocknumber
 2x20 2x1134 1874 2231 2240

1049 maxofbuffer
 2x172 1033 1034 1035 1037 1040 1041 1049 1059 1060
 1079 1086 1088 1095 1104 1892 1893 2162 2x2164 2x2167
 2243

1050 maxofdrum
 1035 1041 1050 1086 1893

1873 maxofELANline
 793 1873 1891

875 maxofinflist
 603 875 1020 1894 1916

1874 maxofnum
 2x938 1874 1895 2203

1050 minofdrum
 1050 1057 1892 2235 2243

716 minus
 716 717 719

1863 minusforrightoperand
 302 2x311 322 349 355 2x365 372 381 382 1172
 1511 1516 1863

1777 minussymbol
 251 302 322 380 529 535 550 717 1662 1687
 1777 1840 2118

1776 Msymbol
 182 447 611 617 637 1776 1838 2064 2067 2072
 2075 2079 2083

115 MI
 112 115

1785 MIdeclarationsymbol
 82 112 115 665 1762 1785 1851

1866 MIdeclared
 119 162 165 452 473 1866 1884

1866 MIdeclaredanddefined
 126 163 453 468 1866 1884

597 n
 597 599 2x600 2x602 608 615 634 648

1564 n
 1564 2x1565 1568 1569 2x1575

1722 n
 3x1722 1725

1732 n
 3x1732 1735

1741 n
 3x1741 1742

1906 n
 1906 1911 2x1912 2x1915 1921

2027 n
 3x2027 2x2028

2030 n
 3x2030 2x2031

250 neg
 250 251 254

680 neg
 680 682 688

1566 neg
 1566 2x1568 1578

502 nega
 502 508 509 510

502 negb

502 508 509 510

1863 newidentifier

86 118 159 188 450 886 890 912 1863

906 NEWIDENTIFIER

906 970 986 994

883 newletgits

883 944 945 949

880 newnum

880 937 2x938 2x941 952 953 963

- NEWPAGE

1956 2244 2248

989 next

989 1000

1873 nextsymbol

156 183 185 748 750 751 2x753 756 797 1873

1789 nler

1789 1801 1807 1810 1816

1774 nlcrsymbol

726 728 735 807 832 833 1054 1072 1156 1702
1729 1735 1737 1774 1827 1836 1951 1952 2000 2174
2178 2188 2193 2246

1145 not

1145 1155 1157 1158

1789 not

1789 1803 1807 1809 1818

1864 notbehindlastend

578 587 1864 2219

1865 nrofbegins

2x580 2x582 583 670 1865 2231 2235 2240

771 NS

586 593 600 654 657 668 670 675 676 687
689 703 712 717 718 728 734 739 768 771
1156

571 NSdeferred

571 592 593 729 2221

1778 Nsymbol
 611 630 642 1519 1669 1778 1841

 881 num
 881 941 942

 2035 num
 2035 2038 2046 2047 2197 2200

 1785 number
 249 252 253 421 681 708 1138 1140 1763 1785
 1851

 1871 numberoffunctions
 1128 1871 1938

 880 numintree
 880 940 2x941 952 953 956 960 984

 1867 octaltype
 1528 1867 1885

 1619 odd
 1619 1621 1622 2x1625 1629

 885 OLDIDENTIFIER
 885 982 990

 880 oldletgits
 880 943 945 949

 415 op
 415 416 417 418

 503 op
 503 504 509 514

 499 opcode
 499 2x500 504 505

 1780 opensymbol
 276 343 425 1780 2x1844 1922

 1869 operator
 303 305 313 314 323 325 330 331 1191 1192
 1193 1194 1198 1199 1200 1201 1869

 1118 opopreg
 330 1118 1125 1191 1193

 1115 opopreginstruction
 332 1115 1132 1188

1004 OUT

903 910 934 978 1004

1776 oversymbol

531 539 1688 1708 1776 1839 1924

431 p

431 433 2x434 435 436 437 438 2x439 2x444 445

597 p

597 599 3x603 604 607 648

1619 p

1619 1621 1623 1626 2x1628 1629

1754 p

3x1754 1755

1906 p

1906 1911 3x1916 1917 1920 2x1921 1928

2100 p

2100 2x2101 2x2102 2x2104 2x2105 2106 2x2107 2108 2x2109 2110
2112

1831 P

1831 4x1848 4x1849 2x1850 3x1851 3x1852

2012 P

2010 2012 2089 2091 2094 2096 2138 2141 2143 2174
2178 2188 2193

1567 p8

1567 2x1569 2x1570 4x1576 2x1577

876 placeenough

93 167 876 1028 1042 1045 1898

875 placeofidentifier

85 119 120 161 163 166 171 875 887 913

1777 plussymbol

303 323 529 534 548 672 674 1662 1687 1706
1777 1839 2070

2054 pointer

2054 3x2060 2062 2088 2089 2093 2094 2137 2138

874 pointerofptrofinflist

874 962 1012 1014 2x1019 1020 1895 1927

883 pointertoconnectwith
 883 896 901 923 969 985

 881 pointertonextidentifier
 881 955 958 972 974

 880 pointintree
 880 937 940 943 956 960 961 972 976 984
 1004

 1778 pointsymbol
 698 711 1778 1842

 1975 PRadcntr
 176 1651 1975

 1543 PRbinarynumber
 1524 1543

 1776 prbussymbol
 780 798 1776 1839 2091 2096 2123 2128 2135

 2158 prcall
 2158 2170 2192

 1949 PRELANSYM
 773 1949 2236 2245 2x2246

 547 primary
 523 547 2x549 2x551 554

 2050 printf
 2050 2156 2197

 - PRINT
 1765

 1548 PRintegralnumber
 1529 1533 1546 1548 1560 1561

 1978 PRINTLINE
 29 1560 1724 1954 1961 1970 1972 1978

 2037 PRINTname
 2037 2175 2186

 1564 PRINToctal
 1553 1564 1976 2090 2095

 - printpos
 1728 2017 2018 2022 2023 2176 2177 2x2187

- PRINTTEXT

1725 1736 1756 2145 2226

2053 PRINTvalue

2053 2189

2033 prlist

2033 2047 2200 2208

2003 PRlist

2003 2256

1532 PRODUCEEXPRCODE

277 1532

1498 PRODUCEINSTRCODE

292 1498

1527 PRODUCENUMBERCODE

259 1527

1556 PRrealnumber

1530 1556

1776 prsubsymbol

779 797 1776 1838 2065 2068 2084 2115

- PRSYM

1572 1580 1584 1729 1735 1737 1752 1985 1993 1997

2000 2013

1751 PRsyntunit

1726 1736 1751

567 prtapesymbol

567 773 776 777 778 779 780 2x781 2x782 2220

2236 2245 2251

2009 psym

2009 4x2043

1778 Psymbol

611 644 1520 1665 1778 1841

874 ptrofinflist

599 2x607 874 907 913 937 944 961 2x963 984

993 1011 1014 1021 1022 1023 1895 1896 1911 2x1920

1925 2x1928 1938

1873 ptroftext

2x774 775 776 2x792 1873 2220

- PUHEP
 1600 1601 2x1602 1608 1631 1637

1564 punch
 1564 1565 1573 1580 1585

1593 PUNCH
 186 266 1552 1593 1651 2247

1634 punchBI
 1594 1634

1605 punchIP
 1594 1605

1864 punchlist
 1864 2013 2019 2022 2028 2031 2090 2095 2146 2204
 2251 2252

- PUNLCR
 2206

- PUSPACE
 2022

- PUSYM
 1573 1580 1585 2013 2019

- PUTEXT
 2146

1867 PZE
 289 290 1173 1510 1520 1521 1522 1867

1784 PZEsymbol
 645 1666 1784 1849

431 q
 431 441 443 2x444 445

597 q
 597 604 609 2x616 2x635 648

1537 q
 1537 1538 2x1539 2x1540

1597 q
 1597 1598 1599 1600

1619 q
 1619 4x1621 2x1623 3x1625 1626 4x1628

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

1556 r
3x1556 1559

1674 r
3x1674 1675

1754 r
2x1754 1756

1557 R
1557 2x1559

1825 R
1825 5x1837 4x1838 4x1839 4x1840 5x1841 3x1842 3x1843 2x1844 2x1845
3x1846

573 RE
53 86 88 113 127 132 2x175 2x193 2x195 253
259 262 264 267 281 282 288 289 299 303
305 323 325 329 338 344 346 368 381 387
416 423 426 428 439 441 442 458 459 460
466 477 533 549 551 573 744 763 768 1138
1139 1151 2x1208 1743

1873 readingptr
775 2x786 2x792 793 794 1873 2220

591 READsyntunit
578 591 2x594 613 621 625 631 638 643 645
649 667 673 681 2x693 708 730 2x736 739

1877 realt26
688 1538 1539 1877 2x1890

1535 realtoint
91 2x507 1529 1533 1535 1540

1867 realtype
255 712 720 1867 1885

297 REarithmeticinstruction
286 297

37 REblockbegin
23 37 1137

19 REELANblock
19 269 2236 2245

79 REELANdeclaration
40 79

248 REELANinstruction
 242 248 1139

 241 REELANstatement
 27 241

 520 REexpression
 193 267 277 354 427 441 465 520 557

 336 REfunctionalinstruction
 286 336

 1869 register
 299 313 314 329 330 331 1191 1192 1193 1194
 1198 1199 1200 1201 1869

 1117 regopop
 313 1117 1127 1198 1200

 1115 regopopinstruction
 315 1115 1132 1196

 - REHEP
 813 2251 2254

 838 rehepavailable
 813 838 839 2253

 154 RElabelsequence
 152 154 155 157 195

 181 RElocationandlabels
 152 181

 151 REMarginalpart
 151 242

 785 RENS
 748 750 752 758 777 785

 413 REoperand
 317 388 413 417 423 426 432 471 472 475
 553 1151

 43 REpossdeclid
 39 43 111 116 131

 379 REpossibleminusymbol
 306 326 360 379

1741 REQUIRE
 109 129 194 368 428 442 466 1741 1742

 384 RRegisteroroperand
 88 308 361 384 386 388 416

 501 res
 501 509 2x514 517

 881 resultofcomparison
 881 946 950 952 956 957 960

 811 RESYM
 805 811 824

 803 RESYM1
 788 803 808 1826 3x1835 1836 1909 1912

 576 retainoldsyntunit
 576

 742 REthroughbarrier
 25 34 56 742 2223

 1868 rightoperand
 308 309 313 314 346 353 361 362 371 374
 375 1197 1200 1201 1203 1206 1207 1868

 - RUNOUT
 2205 2x2244 2x2256

 525 s
 525 527 3x529 3x531 534 535 536 538 539 540

 804 s
 804 805 806 807 808

 1053 s
 3x1053 1054 2x1066 1067 1072 1075 1081

 1661 s
 3x1661 2x1662

 1664 s
 3x1664 2x1665 2x1666

 1668 s
 3x1668 3x1669 2x1670

 1672 s
 3x1672 1675

1686 s
 3x1686 2x1687 2x1688 2x1689 1690
 1692 s
 3x1692 2x1693
 1695 s
 3x1695 2x1696
 1698 s
 3x1698 2x1699
 1701 s
 3x1701 2x1702 1703
 1705 s
 3x1705 1706 1707 1708 1709 1710 1711
 1732 s
 3x1732 1733 1736
 1741 s
 3x1741 1742 1744
 1751 s
 3x1751 2x1752 1755 1765
 1825 s
 2x1825 3x1826 2x1827 4x1828
 1831 s
 2x1831 1832
 1906 s
 1906 5x1909 1910 2x1912 4x1913 2x1914 1922 1923 1924 1937
 1949 s
 3x1949 1951 1952 1968 1972
 2015 s
 2015 2088 2093 2137 2x2141 2x2143 2145 2146 2177 2178
 2187 2188
 1862 secondscan
 186 292 356 839 894 1545 1549 1571 1579 1583
 1862 1956 1966 1976 1979 2229 2239 2249
 1780 semicolonsymbol
 726 735 1702 1780 1843 1937 1972
 1536 sgn
 1536 1538 1540

- sign
 1538 2x1621

 1617 singleBI
 1617 1640 1641 1642 2247

 1596 singleIP
 1596 1611 1612 1613 2247

 1866 sixtyfourK
 1647 1866 1891

 1055 skipspace
 1055 1068

 1782 SKIPsymbol
 265 664 1759 1782 1848

 761 SKIPuntilstatementseparator
 245 273 761

 - space
 1983

 - SPACE
 1728 1990 1991 1995 2023

 1774 spacesymbol
 1066 1699 1774 1826 1835 1904 1998 2091 2096 2220
 2246

 1775 Ssymbol
 436 611 619 1682 1775 1837 2105

 1017 st
 908 6x917 6x925 3x964 1017 1024 3x1929 1932

 2059 st
 2059 2060 2x2063 4x2066 2071 2x2073 8x2076 2103 2107 2x2110
 2114 2117 2x2121 3x2125 4x2131

 1031 ST
 2x899 2x930 1031 1046

 2218 STARTblock
 2218 2232 2236 2245

 1113 STATadop
 309 318 362 417 423 426 553 1113 1131 1168
 1180 1189

1113 STATBoperand
 346 461 1113 1130 1169 1181

1785 statementseparator
 24 55 83 108 128 176 243 270 575 730
 764 1149 1745 1761 1785 1850

1113 STAToperand
 106 161 189 353 386 417 457 460 472 475
 1113 1130 1167 1180 1186 1190 1197 1203

43 statsepallowed
 43 2x44 55

571 statsepbarrier
 571 574 743 744

2056 storeinar
 2056 2063 2066 2071 2073 2076 2103 2107 2110 2114
 2117 2121 2125 2131

1008 STOREincontentsof
 908 917 925 964 967 1008 1011 1929 1932

1027 storeinLINE
 899 930 1027 1028

878 STOREletgitwith
 649 878 1004 1936

1053 stowintobuffer
 789 1053 2237

1790 stringsym
 1790 1803 3x1809 4x1810 1815

- STRINGSYMBOL
 829 1215 1792

1212 stringsymcount
 1212 1215 2x1489 2227

24 stsep
 24 25 27

54 stsep
 54 56

1780 subsymbol
 183 440 458 748 779 797 1780 1844 2077

828 sym
828 829 832 833 835

1791 sym
1791 1792 2x1801 2x1802 1803 2x1804

1873 symbol
2x585 596 600 2x601 653 654 656 658 659 662
663 664 665 668 669 2x672 674 676 678 682
684 685 686 687 689 691 692 2x698 699 2x702
703 704 709 710 711 715 717 718 2x726 728
733 2x735 739 762 767 779 1150 1154 1156 1158
3x1160 1873

570 symcode
570 821 1815 2x1816 2x1817 1818 1819

1789 symcount
1789 1792 2x1800 2x1802 1808

1786 syntunit
24 26 55 81 82 83 84 87 108 110
112 115 128 130 156 176 2x182 187 243 244
2x249 251 252 261 263 265 269 270 271 2x276
280 281 285 287 288 289 298 300 301 302
305 320 321 322 325 328 343 345 347 380
385 414 421 425 430 440 2x447 449 458 459
463 476 527 548 550 2x575 578 579 581 2x764
1140 1152 1726 1733 1744 1745 1746 1786 2223

655 syntunit
655 658 659 660 662 663 664 665 666 667
670

80 t
80 88 90

552 t
552 2x553

1871 t0
512 1175 1871 1886 2234

1871 t1
512 2x513 2x514 1172 1174 2x1502 1505 1506 2x1540 1871
1886

1871 t2
1173 1871 1886

1871 t3
1175 1871 1886

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

1872 t20

1872 1888

1872 t21

1172 1516 2x1600 1872 1888

1872 t22

1872 1888

1872 t23

1872 1889

1872 t24

1872 1889

1872 t25

1872 1889

1872 t26

509 2x510 688 1540 1546 1599 1611 1623 1640 1872
1890

1789 tab

1789 1801 1807 1810 1817

- TAB

2019

1774 tabsymbol

1699 1774 1826 1835 1997 2019

567 tapesymbol

567 777 788 789 791 794 795 797 798 2x799
2x800

1880 ti

1880 3x1882 1886

1776 timesymbol

531 538 672 1688 1707 1776 1839

501 tk

501 2x512 514

- TODRUM

1036 1087 1559 2248

1092 trip

1092 2x1098 1099

1049 triple

1049 1058 1063 1066 2x1067 2x1075 1080 1081 1096 2x1098
1099 2234

1775 Tsymbol

438 476 611 619 1775 1837 2080 2106

448 type

448 451 2x452 453 457 460 461 468 472 2x473

2052 type

2052 2063 2066 2071 2073 2076 4x2086 2091 2096 2x2099
2140 2142 2144 2152

1147 typeispermitted

1147 1148 1151 1182 1189

1783 typeofDYNMsymbol

433 622 626 639 1783

1870 typeoffunctionidentifier

338 977 1870

1869 typeofidentifier

106 126 160 162 164 165 189 451 888 889
890 914 1869

1114 typeofinstruction

315 332 376 1114 1188 1196 1202

1867 typeofnumber

255 422 682 708 712 720 722 2x1528 1867

1786 underline

1786 1802 1807 1809

1114 unknowntype

889 919 927 1114 1131

337 upperbound

337 339 340 341 342 344 357 370

568 uppercase

568 819 1811

568 uppercasecode

568 820 1812 1813 1816 1817 1818 1819

1778 Usymbol

612 642 1509 1517 1669 1778 1841

1867 UYN
 281 284 1174 1175 1508 1509 1517 1518 1519 1867

1783 UYNsymbol
 643 1174 1670 1783 1849

1674 v
 3x1674 1677

2052 value
 2052 2x2087 2090 2095 2x2101 2148 2153

1870 valueofidentifier
 190 454 455 891 915 920 928 1870

1876 valueofnumber
 2x257 423 683 2x686 2x688 722 1529 1876

1876 valueofoperand
 91 309 2x310 348 354 2x357 362 2x364 371 387
 423 427 445 453 457 2x462 2x465 2x469 2x470 476
 554 2x1512 1513 1523 1876

1876 valueofrealnumber
 2x256 723 1530 1876

1868 valueofregister
 387 1677 1868

1543 w
 1543 2x1544 2x1546

1146 W
 1146 1155 2x1160 1166

1177 w1
 1177 1178 1183

1177 w2
 1177 1178 1186

1596 word
 3x1596 1598 2x1599 3x1600 3x1601 2x1602

1617 word
 1617 2x1618 1621

1165 WORD
 1165 1166 1183 1186

1591 words
 1550 1591 1613 1642

110

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 addresses 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'


```

1  "LOADER OF THE BINARY PAPERTAPES, PRODUCED BY THE MC - ELAN
2  "ASSEMBLER, RPR 103370 - R 2014N
3  'BEGIN' VERY FIRST BEGIN, VERY FIRST END, CLEAR, MOVE, INITIALIZATION,
4  LAST ADDRESS = (1200 000 - 1), SUMCHECK, RESET NR = 38,
5  RESET AR = (164 + RESETNR * 4),
6  'MT, BEGINOFREADINGPROGRAM, ENDOFREADINGPROGRAM, LENGTHOFREADINGPROGRAM,
7  ENDOFMEMORY, IP READER, BSTACK, BRUSH, DYST, REHEP,
8  INIT REHEP, INIT REHEP AFTER MOVING, REHEP INF,
9  REBIWORD, REBITAPE, SETWORD, ENDACT,
10 IPAR, SHADOW IPAR, SHADOW RESET AR
11
12 '001001': '000176000'
13 '001002':
14
15 '001003': '726175400'
16 '001004': '540400002'
17 '001005': '760060000'
18 '001006': '512000001'
19
20
21
22 '001007': '126073400'
23 '001008': '166075400'
24 '001009': '726173400'
25 '001010': '002000001'
26 '001011': '540400006'
27 '001012': '520000027'
28 '001013': '777777723'
29 '001014': '413711611'
30 '001015': '520000001'
31 '001016':
32
33 '176000':
34
35
36 '176001': '622000000'
37 '176002': '426076752'
38 '176003': '402000001'
39 '176004': '606175377'
40 '176005': '417176746'
41 '176006': '512300003'
42 '176007': '611000014'
43 '176008': '622300000'
44 '176009': '526376744'
45
46 '176010': '632000000'
47 '176011': '422001000'
48 '176012': '726175400'
49 '176013': '557176736'
50 '176014': '512300003'
51 '176015': '426076736'
52 '176016': '402000001'
53 '176017': '726175400'
54 '176018': '457176736'
55 '176019': '512300003'
56
57
58
59 '176020': '660075000'
60 '176021': '760075001'

```

"THIS ROUTINE SETS RANGE M[B] - M[B + COUNT0 - 1]
 "TO THE CONTENTS OF G. IT IS ACTIVATED IN ENDACT
 MC = G
 REPO P (1: CLEAR)
 IV ON
 JUMP (-1)
 "NOW THE INTERRUPT OF THE
 "READER OCCURS
 "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
 S = MA
 MC = S
 MA = G
 A = 1
 REPO P (1: MOVE)
 GOTO (LINK 15)
 777 777 723
 413 711 611
 GOTO (M[1])
 M[24]:
 N[1] LAST ADDRESS - 1023]:
 INITIALIZATION: VERY FIRST BEGIN
 'BEGIN' NODECLARATION = 0
 "A SUMCHECK IS DONE ON THE LOADER ITSELF
 F = 0
 B = ENDOFREADINGPROGRAM
 B + 1
 G = MC(-1)
 U, B = BEGINOFREADINGPROGRAM, Z
 N, JUMP (-3)
 F = SUMCHECK, Z
 N, A = 0
 N, GOTO (DYST)
 "THE MEMORY IS CLEARED:
 G = -0
 B = 512
 MC = G
 U, BEGINOFREADINGPROGRAM - B, Z
 N, JUMP (-3)
 B = ENDOFREADINGPROGRAM
 B + 1
 MC = G
 U, ENDOFMEMORY - B, Z
 N, JUMP (-3)
 "THE NUMBER OF THE IP - READER IS DETERMINED.
 "STARTING THE COMPUTER, ALL THE (INTERUPT) FLIPFLOPS ARE CLEARED
 "DURING THE ACTIVATION OF THE IP - READER THE IF FOR THIS APPARATUS IS SET
 IFA (0)
 IFS (1)
 "TAKE THE FIRST IF - WORD IN A
 "TAKE THE SECOND IF - WORD IN S

```

61 176023: 060000340'
62 176026: 412500007'
63 176027: 430300075'
64 176030: 402300047'
65 176031: 412200010'
66 176032: 466076723'
67 176033: 664071060'
68 176034: 620100075'
69 176035: 602004000'
70
71 176036: 462076722'
72 176037: 572476755'
73
74 176040: 572476444'
75
76 176041: 536076554'
77 2
78
79 176042:
80
81
82
83
84
85
86
87
88
89 176043: 466076466'
90 176045: 022000012'
91 176044: 056576462'
92 176045: 536376430'
93
94 176046: 026176716'
95 176047: 432000000'
96 176050: 466072401'
97 176051: 426076460'
98 176052: 466072402'
99 176053: 426076702'
100 176054: 754070000'
101 176055: 026572401'
102 176056: 512300002'
103
104 176057: 026476453'
105 176060: 536276411'
106 176061: 671000062'
107 176062: 022200001'
108 176063: 526276671'
109
110
111
112
113 176064: 022000002'
114 176065: 066076442'
115 176066: 626176676'
116 176067: 162076444'
117 176070: 166072400'
118 176071: 536076354'
119 176072:
120 176072: 026076435'

      NORAS
      U, B = 7, P
      N, B = -B
      N, B + 39
      Y, B = R
      IPREADER = B
      IF OFF (B)
      G = B
      G + '000 004 000'

      B = BSTACK
      SUBC (:BRUSH)
      "THE STACK IS INITIALIZED:
      "THE INITIALIZATION IS STARTED:
      SUBC (:INIT REHEP)
      "THE TAPE IS READ:
      GOTO (:REBITAPE)

      "END: INITIALIZATION

      REHEP: "THIS SUBROUTINE, CALLED BY SUBC (:REHEP), DELIVERS IN S
      "THE NEXT HEPTAD OF THE TAPE, IN AN 'END-OF-TAPE' SITUATION
      "NEW STARTING COMMANDS ARE GIVEN TO THE READER TILL THE NEXT
      "TAPE HAS BEEN INSERTED.
      "AS THE FIRST HEPTAD, WHICH IS READ, IS UNDEFINED, DUE TO THE
      "BUFFERING SYSTEM OF THE READER, THIS ONE IS SKIPPED.
      "THE NEXT NON - LOCALS ARE USED: IPREADER, DYST
      'BEGIN' BUFLNGTH = 10,
      'MT' HEPPONTER, STARTLINK, CODEWORD, ENDWORD, FIRSTONE, SAVES,
      D18, STARTCOMMAND, OK, HEP
      SAVES = B
      A = BUFLNGTH
      U, HEPPONTER = A, P      "HEPPONTER > BUFLNGTH?
      N, GOTO (:HEP)
      "THE BUFFER IS EMPTY, TO START THE READER, A NEW COMMAND HAS TO BE GIVEN:
      STARTCOMMAND:
      G = IPAR
      B = -0
      NG[1] = B
      B = D18
      NG[2] = 5
      B = IPREADER
      AFON(B)
      U, A = NG[1], P
      N, JUMP (-2)

      "NOK SITUATIONS ARE HANDLED:
      A = ENDWORD, P
      Y, GOTO (:OK)
      RUA(18), Z
      "NBK?
      Y, A = 1
      Y, GOTO (DYST)
      "PHONE TECHNICIAN

      "NOW WE HANDLE THE END-OF-TAPE SITUATION.
      "A NEW STARTING COMMAND IS GIVEN, AFTER SETTING CNT ARO = 0.
      "FIRSTONE IS USED TO MARK, WHETHER THE FIRST HEPTAD IN THE
      "BUFFER IS TO BE CONSIDERED.
      A = 2
      FIRSTONE = A
      G = IPAR
      S = :STARTLINK
      MG = S
      GOTO (:STARTCOMMAND)
      OK: "A BUFFER HAS BEEN READ:
      A = FIRSTONE

```

```

121 '176073' '060076433'
122 '176074' '022000001'
123 '176075' '060076432'
124 '176076'
125 '176076' '62076435'
126 '176077' '106076427'
127 '176100' '126074401'
128 '176101' '022000001'
129 '176102' '046076424'
130 '176103' '426076425'
131 '176104' '526475377'
132
133 '176105' '022000013'
134 '176106' '066076420'
135 '176107' '022000002'
136 '176110' '066076417'
137 '176111' '126076644'
138 '176112' '760000042'
139 '176113' '102000100'
140 '176114' '166076450'
141 '176115' '022000012'
142 '176116' '060000062'
143 '176117' '042076415'
144 '176120' '066076414'
145 '176121' '026176643'
146 '176122' '162076411'
147 '176123' '166072400'
148 '176124' '166074400'
149 '176125' '166074377'
150 '176126' '526475377'
151
152 '176127' '000000013'
153 '176130' '000000002'
154 '176131' '000000000'
155 '176132' '001000000'
156 '176133' '000000000'
157 '176134' '000000000'
158 '176135' '000000000'
159 '176136' '000000000'
160 '176137' '000000000'
161 '176140' '000000000'
162 '176141' '000000000'
163 '176142' '000000000'
164 '176143' '000000000'
165 '176144' '000000000'
166 '176145' '000000000'
167 '176146' '000000000'
168 '176147' '000000000'
169
170
171 '176150'
172
173
174
175
176 '176150' '572476271'
177 '176151' '365100100'
178 '176152' '526675377'
179 '176153' '113000177'
180 '176154' '022300002'

HEP: "A NEW HEPTAD CAN BE SELECTED:
      S = :STARTLINK
      S + HEPPONTER
      S = MS[1]
      A = 1
      HEPPONTER + A
      B = SAVEB
      GOTOR (MC[-1])

"INITIALIZATION OF REHEP
INITREHEP:
      A = (BULENGTH + 1)
      HEPPONTER = A
      A = 2
      FIRSTONE = A
      S = IPREADER
      LUS(2)
      S + 64
      IPAR = S
      A = BULENGTH
      LUA (16)
      A + :STARTLINK[1]
      CODEWORD = A
      G = IPAR
      S = :STARTLINK
      MS = S
      MS = S
      MS[1] = S
      GOTOR (MC[-1])

"SPACE FOR VARIABLES AND THE BUFFER:
HEPPONTER: 2
FIRSTONE: 0
SAVEB: 0
D18: 1 000 000
ENDWORD: 0
STARTLINK: 0
CODEWORD: 0
"END' REHEP

REHEPINF: "THIS SUBROUTINE READS A HEPTAD FROM A TAPE, THE LENGTH OF
           "WHICH CAN BE CONSIDERED TO BE INFINITE, DUE TO HANDLING OF
           "END-OF-TAPE SITUATIONS INTERNALLY. A CALL, BY SUBC (:REHEPINF)
           "DELIVERS ONE HEPTAD IN S.
           'BEGIN' 'MT' START
           START:
           SUBC(:REHEP)
           U, S '1 64, Z
           Y, GOTOR(MC[-1])
           S = 127, Z
           N, A = 2

"CODEWORD:= CNTPART*D18 + ADR
"
"ARO:= :STARTLINK

```

```

181 '176155': '526376577'
182 '176156': '572476263'
183 '176157': '113000177'
184 '176160': '512300003'
185 '176161': '536076366'
186
187
188 '176162':
189
190
191
192
193 '176162': '0220000004'
194 '176163': '060076426'
195 '176164': '640000146'
196 '176165': '060076425'
197 '176166': '130076425'
198 '176167': '572476360'
199 '176170': '026076422'
200 '176171': '100076422'
201 '176172': '526176417'
202 '176173': '613000001'
203 '176174': '726176413'
204 '176175': '536376366'
205
206
207 '176176': '670400000'
208 '176177': '348376415'
209 '176200': '278376414'
210 '176201': '240000074'
211 '176202': '060075400'
212 '176203': '026075377'
213 '176204': '240000074'
214 '176205': '600063000'
215 '176206': '023000003'
216 '176207': '526376545'
217 '176210': '272000004'
218 '176211': '526475377'
219 '176212': '000000000'
220 '176213': '000000000'
221 '176214': '000000000'
222 '176215': '400000000'
223
224
225 '176216':
226
227
228
229
230 '176216': '572476331'
231 '176217': '021100074'
232 '176220': '512200003'
233 '176221': '572476330'
234 '176222': '013100003'
235 '176223': '022300004'
236 '176224': '526376530'
237 '176225': '020000074'
238 '176226': '266076430'
239 '176227': '066076426'
240 '176230': '770000022'

N, GOTO(DYST)
SUBC (:REHEP)
S = 127, Z
N, JUMP (-3)
GOTO (:START)

'END' REHEPINF

REBIWORD: "THIS SUBROUTINE, CALLED BY SURC (:REBIWORD), READS A BI = WORD
          "IN A AND S, A CONTAINS C1 AND C2, S CONTAINS D26 - D0.
          "IN CASE OF A PARITY ERROR THE MACHINE WILL STOP DYNAMICALLY.
          "THE FIRST HEPTAD IS EXPECTED BY REBIWORD IN S
          'BEGIN' 'MT' SAVEA, SAVES, CNT, REPEAT, D26
          A = 4
          CNT = 4
          "AS: = AS * 2**6
          REPEAT:
          LUAS(6)
          SAVEA = A
          SAVES = 5
          SUBC (:REHEPINF)
          A = SAVEA
          S + SAVES
          G = CNT
          G = 1, Z
          CNT = G
          "IF CNT > 0 THEN GOTO REPEAT
          N, GOTO (:REPEAT)
          "THE PARITY HAS TO BE CHECKED AND THE SIGNBIT OF S SET.
          "THIS BIT IS NOT INVOLVED IN THE SHIFTPROCESS.
          D26 = 0? A1 = A 'DIV' 2
          "SET BIT26: = 0
          "PAR(A) + 'PAR(S) = PAR(A) + 'S
          "LP: = PAR(A) + 'S
          "A: = A + 'S + 'S = A
          "C: = LP
          "EVEN
          "DELETE PARITY BIT
          RCA (1), P
          N, S + 'D26
          N, A + 'D26
          A + 'S
          MC = A
          A = MC[-1]
          A + 'S
          CLP
          N, A = 3
          N, GOTO(DYST)
          A + ' -4
          GOTOR (MC[-1])
          C
          SAVEA:
          SAVES:
          D26:
          '400 000 000'
          'END' REBIWORD

REBIPE: "THIS SUBROUTINE, CALLED BY GOTO (:REBIPE), READS THE BI = BLOCKS
        "OF THE BI = TAPE. TO SET A MEMORY WORD, SET WORD IS CALLED, IN THE
        "CASE OF AN END MARKER ENDCT.
        'BEGIN' 'MT' CNT, ADR, D18M1, READBIWORD, READBIBLOCK
        "WE START BY READING THE FIRST WORD, THE DIRECTIVE, OF THE BLOCK.
        READBIBLOCK:
        SUBC (:REHEPINF)
        U, A = S, Z
        Y, JUMP (-3)
        SUBC (:REBIWORD)
        U, A = 3, Z
        N, A = 4
        N, GOTO (DYST)
        A = S
        A + ' D18M1
        ADR = A
        RCS (18)
        "BECOMES
        " ADDRESS PART OF S
        "CNT

```

```

241 176231: 036000777
242 176232: 122201000
243 176233: 166076421
244
245 176234: 113101000
246 176235: 003200000
247 176236: 572676534
248
249
250 176237: 572476310
251 176240: 572476321
252 176241: 003000000
253 176242: 023300025
254 176243: 526376511
255 176244: 022000001
256 176245: 046076410
257 176246: 026076407
258 176247: 572476410
259
260
261 176250: 026076404
262 176251: 013000001
263 176252: 086076402
264 176253: 536376363
265
266 176254: 536076341
267 176255: 000000000
268 176256: 000000000
269 176257: 000777777
270
271
272 176260:
273
274
275
276
277
278
279
280 176260: 066076467
281 176261: 056576471
282 176262: 016776471
283 176263: 536376423
284
285
286 176264: 026076463
287 176265: 012500070
288 176266: 013500076
289 176267: 012600027
290 176270: 020300066
291 176271: 526376463
292 176272: 026076455
293
294
295
296 176273: 626176471
297 176274: 052572403
298 176275: 053572377
299 176276: 010300072
300 176277: 042376466

```

S '1777', Z " BECOMES
 Y, S = 512 " " COUNTPART OF S
 CNT = S
 "NOW WE HANDLE THE CASE OF AN END-MARKER
 U, S = 512, Z
 Y, A + 0, Z
 Y, SUBC (:END ACT)
 "CNT BI- WORDS SHOULD BE READ AND STORED SEQUENTIALLY IN THE MEMORY
 "STARTING FROM ADR + 1.
 READBIWCRD:
 SUBC (:REHEPINF) "S:= HEPTAD
 SUBC (:REBIWORD) "AS:= BI-WORD
 A + 0, Z "C1 'NE' 0 ~ C2 'NE' 0?
 N, A = 5
 N, GOTO (DYST)
 A = 1
 ADR + A
 A = ADR
 SUBC (:SETWORD)
 "THE READING PROGRAM MAY HAVE BEEN MOVED NOW, THE VALUES OF CNT AND ADR
 "HOWEVER ARE UNCHANGED
 A = CNT
 A = 1, Z
 CNT = A
 "CNT:= CNT - 1
 "IF CNT>0 THEN GOTO REPEAT
 N, GOTO (:READBIWORD)
 "READ THE NEXT BI BLOCK:
 GOTO (:READBIBLOCK)
 0
 0
 1777 777
 D18M1:
 "END' REBITAPE
 SETWORD: "THIS SUBROUTINE SETS MA:= S, IF EXECUTION OF THE INSTRUCTION
 "SHOULD OVERWRITE THE READING PROGRAM, THEN BEFORE EXECUTION
 "A) IT IS CHECKED, WHETHER THERE EXISTS A COHERENT RANGE IN
 " THE MEMORY FOR THE READING PROGRAM, AND
 "B) THE READING PROGRAM IS MOVED.
 "ASSUMED TO BE GLOBAL ARE: BEGINOFREADINGPROGRAM,
 "ENDOFREADINGPROGRAM, LENGTHOFREADINGPROGRAM,
 "BEGIN' 'MT' SAVEA, SAVES, SAVES, REPEAT, RETURNADDRESS, SET, DISPL.
 SAVEA = A
 U, BEGINOFREADINGPROGRAM = A, P
 N, A = ENDOFREADINGPROGRAM, P
 N, GOTO (:DISPL)
 "NOW WE TEST WHETHER THE MEMORY CELL ASKED FOR BY THE PROGRAM TO BE LOADED
 "IS ONE OF THE 'HOLY' ADDRESSES
 A = SAVEA
 U, A = :M[56], P
 U, A = :M[82], E
 Y, A = :LINK15, P
 N, A = 6
 N, GOTO (DYST)
 A = SAVEA
 "SECONDLY WE TEST WHETHER THE MEMORY CELL IS ONE OF THE
 "APPARATUS REGISTERS OF THE IP READER OR THE RESET, IN WHICH
 "CASE A SHADOW ADMINISTRATION IS PERFORMED.
 G = IPAR
 U, A = :M[3], P
 U, A = :M[1], E
 N, A = 6
 N, A = :SHADOW IPAR

```

301 '176300': '013100330'
302 '176301': '062276470'
303 '176302': '160073403'
304 '176303': '526475377'
305 '176304': '460076444'
306 '176304': '166076444'
307 '176305':
308
309
310
311
312 '176306': '026076450'
313 '176307': '006076443'
314 '176310': '016076437'
315
316
317 '176311': '426076441'
318 '176312': '620100073'
319 '176313': '602000777'
320 '176314': '410500072'
321 '176315': '023300007'
322 '176316': '526376436'
323 '176317': '120000075'
324 '176320': '110000073'
325 '176321': '026076435'
326
327
328 '176322': '627175377'
329 '176323': '536376366'
330 '176324': '630500072'
331 '176325': '536376364'
332 '176326': '441100074'
333 '176327': '512300006'
334
335
336
337
338
339
340
341 '176330': '060000000'
342 '176331': '028076421'
343 '176332': '110000073'
344 '176333': '146076415'
345 '176334': '146076424'
346
347
348 '176335': '146076416'
349 '176336': '146076414'
350 '176337': '142076403'
351 '176340': '160000027'
352 '176341': '632000000'
353 '176342': '522000006'
354 '176343': '426076405'
355 '176344': '572476150'
356 '176345': '026076402'
357 '176346': '126076403'
358 '176347': '536076332'
359 '176350': '000000000'
360 '176351': '000000000'

U, A = !RESET AR, Z
Y, A = ! SHADOW RESET AR
MA = S
GOTO (MC[-1])
DISPL: "A, S AND B SHOULD BE SAVED AS THE READING PROGRAM IS DISPLACED.
SAVES = B
SAVES = S

"FIRST IT IS CHECKED, WHETHER THERE EXISTS A RANGE, IN WHICH THE
"READING PROGRAM FITS, I.E. A RANGE M[1], I = START ADDRESS,....
"END ADDRESS, M[1] = 0, ENDADDRESS = STARTADDRESS = END OF READING PROGRAM
"= BEGIN OF READING PROGRAM, SET ADDRESS 'GE', END ADDRESS + 1.
A = LENGTH OF READING PROGRAM "A IS THE NUMBER OF WORDS,
A + BEGIN OF READING PROGRAM "THAT IS NEEDED FOR THE NEW
A = SAVEA
"RANGE, THE FIRST TIME THIS IS
"LESS THAN
"LENGTH OF READING PROGRAM

B = BEGIN OF READING PROGRAM
G = A
G + 511
"ANY SPACE LEFT?
N, B = G, P
N, A = 7
N, GOTO (DYST)
S = B
S = A
A = LENGTH OF READING PROGRAM
"THE NEXT TIME WE HAVE TO TEST
"OVER THE WHOLE RANGE
"M[B-1] = 0? B:= B - 1

"THE WHOLE RANGE TESTED?
"IT IS CONTAINED BY B. THE NEXT STEP IS TO MOVE THE READING PROGRAM TO
"MB] - MB] + LENGTH OF READING PROGRAM - 1].
"THIS IS DONE BY CALLING MOVE, STORED IN THE 'HOLY CORE'. MOVE EXPECTS
"IN A THE OLD BEGIN ADDRESS, IN B THE NEW BEGIN ADDRESS, IN G THE VALUE,
"WHICH SHOULD BE STORED IN THE OLD RANGE, IN COUNT0 THE LENGTH OF THE
"READING PROGRAM AND IN LINK15 THE RETURN ADDRESS.
COUNT0 = A
A = BEGIN OF READING PROGRAM
S = A
SAVES + S
BSTACK + S

"S = SHIFTLLENGTH
"S AS WELL AS THE STACK TO
"WHICH IT POINTS, THE DEPTH OF
"WHICH NOW EQUALS 1, HAVE TO BE
"MODIFIED.

END OF READING PROGRAM + S
BEGIN OF READING PROGRAM + S
S + !RETURN ADDRESS
LINK15 = S
G = -0
GOTO (!MOVE)
B = SAVEB
SUBC (!INITREHEAFTERMOVING)
A = SAVEA
S = SAVES
GOTO (!SET)
SAVEA:
SAVEB:
0

```

```

361 '176352': '000000000'
362 'END' SETWORD 0
363
364
365 '176353': '000176000'
366 '176354': '00176424'
367 '176355': '00077777'
368 '176356': '00000000'
369 '176357': '000000425'
370 '176360': '00020000'
371 '176361': '000000000'
372 '176362': '000000000'
373 '176363': '000000000'
374 '176364': '000000000'
375 '176365': '000000000'
376 '176366': '000000000'
377 '176367': '000000000'
378 '176370': '000000000'
379 '176371': '000000000'
380 '176372': '000000000'
381
382 '176373':
383
384
385 '176373': '060076372'
386 '176374': '420076373'
387 '176375': '160075404'
388 '176376': '620173400'
389 '176377': '726175400'
390 '176400': '000000001'
391 '176401': '411000074'
392 '176402': '512000005'
393 '176403': '026076366'
394 '176404': '0600000330'
395
396
397
398
399
400 '176405': '060060000'
401 '176406': '420076347'
402 '176407': '760071000'
403
404 '176410': '420076342'
405 '176411': '020076345'
406 '176412': '060000000'
407 '176413': '630000000'
408 '176414': '520000002'
409 '176415': '720100330'
410 '176416': '760070046'
411 '176417': '760075000'
412 '176420': '760400002'
413 '176421': '512000003'
414 '176422': '660071046'
415 '176423': '526075377'
416 '176424': '0000000001'
417

```

SAVES: 0
 'END' SETWORD
 "SOME SPACE FOR CONSTANTS AND VARIABLES IS RESERVED:
 BEGINOFREADINGPROGRAM: ;INITIALIZATION
 ENDOFREADINGPROGRAM: IVERYFIRSTEND
 DYST: '777 777'
 IPREADER: 0
 LENGTHOFREADINGPROGRAM: (:VERYFIRSTEND - :VERYFIRSTBEGIN + 1)
 ENDOFMEMORY: (LASTADDRESS + 1)
 BSTACK: 0
 "FOR THE STACK 4 WORDS
 "ARE NEEDED ONLY
 IPAR: 0
 SHADOW IPAR: 0
 SHADOW IPAR: 0
 SHADOW RESET AR: 0

ENDOCT: "THIS ROUTINE HANDLES THE END ACTIONS:
 "FIRSTLY THE APPARATUS REGISTERS OF THE IP READER AND THE RESET
 "ARE FILLED.
 A = : SHADOW IPAR
 B = IPAR
 S = : MC[+4]
 G = MA
 MC = G
 A + 1
 U, B = S, Z
 N, JUMP(-5)
 A = SHADOW RESET AR
 RESET AR[0] = A

"SECONDLY, THE PROGRAM IS
 "CLEARED, BY THE ROUTINE CLEAR, WHICH IS STORED IN THE 'HOLY' CORE.
 "THIS ROUTINE EXPECTS IN B THE BEGINADDRESS, IN G THE VALUE, WITH WHICH
 "THE READERPROGRAM HAS TO BE OVERWRITTEN AND IN COUNT0 THE NUMBER OF
 "WORDS TO BE CLEARED. EVENTUALLY AN INTERRUPT OF THE IP - READER IS FORCED.
 "THE MACHINE BECOMES DEAF
 IV OFF
 R = IPREADER
 IF CN (B)
 "IF THE CPU WERE LISTENING, THERE
 "WOULD COME AN INTERRUPT NOW
 B = BEGINOFREADINGPROGRAM
 A = LENGTHOFREADINGPROGRAM
 COUNT0 = A
 G = -0
 GOTO (:CLEAR)
 RESET AR[0] = G
 AFON (RESETNR)
 IFS (0)
 LCS (2), P
 Y, JUMP (-3)
 IFOFF (RESETNR)
 GOTOR(MC[-1])
 1
 VERYFIRSTEND:
 'END'


```

"TEST FOR BINARY 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 = (YY + VV),SS = ZZ(B + (:VV + VV)),
AA,BB = ('200 000' - '2 000'), END OF MEMORY

```

```

7 '001123': '000000001' M['123']: 'B'
8 '001124': '000000002' 1:
9 '001125': '000000003' 2:
10 '001126': '000000004' 3:
11 '001127': '000000005' 4:
12 '001130': '000000006' 5:
13 '001330': '777777775' M['330']: -16
14 '001070': '000000001' 1
15 '001077': '000000001' M[56]: 1
16 '001001': '000000001' M[63]: 1
17 '001002': '000000002' AA: 1
18 '001003': '000000003' 2:
19 '001004': '000000004' 3:
20 '001005': '000000005' 4:
21 '001006': '000177777' 5
22 '175700': '000000001' 'BEGIN' ZZ = M,YY = M,XX = M,WW = M,VV = M,
23 '175400': '000000002' UU = M,B,TT = M23[45],SS = M58[10 - 55]
24 '175100': '000000003' M[35] - '100': 1
25 '174400': '000000004' M[38] - '400': 2
26 '174100': '000000005' M[42] - '700': 3
27 '173500': '777777775' M[45] - '1400': 4
28 '173351': '777777775' M[48] - '1700': 5
29 '173352': '777777774' 'NOTE THAT THE BEGIN-ADDRESS OF THE LOADER IS BB AND THAT ITS
30 '173353': '777777773' "LENGTH IS '425' INSTRUCTIONS.
31 '173354': '777777772' 'END'
32 '176150': '777777776' 'BEGIN' UU = :AA,TT = 22,SS = (:AA + :A - :AA)
33 '176151': '777777775' M[BB + '350']: -1;
34 '176152': '777777774' -2; -3;
35 '176153': '777777773' -4; -5
36 '176154': '777777772' "THE LOADER IS NOW SHIFTED OVER '55'
37 " PLACES" AND THE MACHINE IS INSPECTED TO CHECK THAT
38 " THE END-ADDRESS OF THE LOADER IS INDEED
39 " (BB + '150' - 1) = '176 347'.
40 "THE TECHNIQUE TO FORCE THE LOADER TO STOP IS BY MEANS
41 " OF AN END-OF-TAPE SITUATION.
42 'BEGIN' UU = :MG,TT = :MA,SS = :MC
43 M[BB + '150']: -1;
44 -2;
45 -3;
46 -4;
47 -5
48 "THE LOADER IS NOW SHIFTED TO THE FIRST FREE TRAJECT,
49 " ITS END-ADDRESS IS NOW (BB - '700' - 1) = '175 077'
50 " ONCE MORE THIS HAS TO BE INSPECTED.
51 'END'
52 M[24]: 6000 (25) G = M[56]; G + M[63];
53 '001030': '5220000031' 4
54 '001031': '620100070' G + M[123]; G + M[124];
55 '001032': '200100077'
56 '001033': '600100123'
57 '001034': '600100124'

```

```

40 '001035': '600100127'
   '001036': '600100130'
   '001037': '600100330'
41 '001040': '420001006'
42 '001041': '606175377'
43 '001042': '412501000'
44 '001043': '512200003'
45 '001044': '611101006'
46 '001045': '032200001'
47 '001046': '032300002'
48 '001047': '026173400'
49
   'END'
   'END'

```

```

   'END'

```

```

   3
   1

```

```

   G = M['127'];   G = M['130'];   G = M['330']

   B = END OF MEMORY
   G = M[1-1]
   U, B = 512, P
   V, JUMP(-3)
   G = END OF MEMORY, Z
   Y, A = -1
   W, A = -2   "A FORBIDDEN SITUATION.
   U, A = M*   "THE #8 STOPS.

```

AA	M['00'001']	4							
BB	M['17000']	4							
ENDOFMEMORY	M['00'000']	4							
SS	M['B+'04000']	3							
	PG1-45]	15							
	'000073'	24							
TT	:MC[0]	32							
	M['B+'04000']	3							
	#23[45]	15							
	'000026'	24							
UU	:MA[0]	32							
	M['B+'04000']	3							
	MD[0]	15							
	'001001'	24							
	:MG[0]	32							
VV	'000000'	2							
	MT[0]	14							
WW	M['000076']	2							
	MC[0]	14							
XX	M['000000']	2							
	MS[0]	14							
YY	M['000000']	2							
	MA[0]	14							
ZZ	M['000000']	2							
	MG[0]	14							