F.E.J. KRUSEMAN ARETZ
P.J.W. TEN HAGEN
H.L. OUDSHOORN

AN ALGOL 60 COMPILER
IN ALGOL 60

TEXT OF THE MC-COMpiler FOR THE EL-X8

MATHEMATISCH CENTRUM
AMSTERDAM 1973
# CONTENTS

## PREFACE

0. INTRODUCTION
   0.0 Historical notes
   0.1 Parsing and macro-generation
   0.2 Left - to - right scans
   0.3 Modules
   0.4 The macro processor
   0.5 Error handling
   0.6 Evaluation
   0.7 Hints to the reader
   0.8 References

1. THE ALGOL 60 PROGRAM

2. FIXED DATA
   2.0 Description
   2.1 Data

3. EXAMPLES
   3.0 Description
   3.1 Example 1
   3.2 Example 2

4. TABLES
   4.1 Optimized macros
   4.2 Macro descriptions
   4.3 Instruction table
   4.4 Actual Parameter Descriptors

5. STRUCTURE OF THE NAME LIST
   5.1 Name cells
   5.2 Block cells

6. CROSS - REFERENCE TABLE
PREFACE

The ALGOL 60 program, presented in this report, was written essentially in the years 1964-1965 at the Mathematical Centre in Amsterdam by F.E.J. Kruseman Aretz (now at Philips Research Laboratories, Eindhoven).

It is able to translate programs, written in the source language ALGOL 60, into a target language, being bit-patterns in the memory of an EL-X8 computer. This ALGOL version of the ALGOL compiler has been used as a blueprint for a version in ELAN, the symbolic assembly language for the EL-X8 and has been in operation from November 1965.

This report is rather late, due to many other obligations on the part of one of the authors. However, although the EL-X8 will be obsolete soon, publication of this report still seems of some value, because most of it is rather machine-independent, and as such it may serve as an illustration of compiler techniques.

For the preparation of this report, P.J.W. ten Hagen updated the ALGOL 60 version of the compiler and brought it in close correspondence to the ELAN version. Later on some further improvements were made, especially in the algorithm generating the line number bookkeeping instructions. For the purpose of demonstration, a special output procedure has been added.

FKA
0. INTRODUCTION

0.0 HISTORICAL NOTES

The ALGOL 60 program presented in this report, constitutes a complete ALGOL 60 translator. It is able to read in a text, written in ALGOL 60 and punched in the so-called MC flexewriter code, and to produce from it a machine program for an EL-X8 computer.

The MC ALGOL 60 system for the EL-X8 has been in development for a couple of years, and it had to fulfill a number of (almost contradictory) requirements:
1) it should be adapted to the smallest X8 configuration possible (i.e. with a core memory of 16K words of 27 bits, no backing store),
2) it should have high efficiency at compile-time and produce reasonably efficient object programs,
3) it should have a modular structure, so that it could be easily adapted to future requirements,
4) it should check against program errors wherever possible,
5) the compiler should be of educational value, so that it could be used as an introduction to compiler-writing,

The development of the system has been the work of a team, in which, among others, cooperated: F.J.M. Barning (standard functions, EL-X8 simulator), J.A.Th.M. van Berckel (I/O routines, operating system "PICO"), F.E.J. Kruseman Aretz (run-time routines, compiler) and J.J.B.M. Nederkoorn (run-time routines, documentation). The key work, namely the design of the compiler output and the development of the run-time routines, was done by Nederkoorn and Kruseman Aretz in the years 1963-1964. It was partly described by Nederkoorn in a preliminary report [1], of which only a few copies were made and distributed. The compiler was written in ALGOL 60 first. So it could be tested out, in parts, using the ALGOL 60 implementations (I and II) for the EL-X1 computer at the NC. In this version full attention was paid to clarity. Afterwards the compiler was hand-coded into ELAN, the symbolic assembly language for the EL-X8, and debugged using the X8 simulator on the EL-X1 as developed by Barning.

Using the ALGOL text as a blueprint, the hand-coding took a few months
only, and relatively few coding errors occurred. The result was documented by Kruseman Aretz and Mailloux [2], together with the run-time routines and a simple operating system.

In the years from 1966 onwards two other documents were produced, one [3] defining the compiler output in great detail, the other [4] being written as lecture notes to a course given at the University of Amsterdam during the academic year 1969-1970. Both documents are in Dutch. Moreover, a very special aspect was described in [5].

It is hoped that the ALGOL text of the compiler is largely self-explanatory, even without any knowledge of either the structure of the EL-X8 instruction set or of the object program produced by the compiler. The following sections will therefore only sketch very briefly some general characteristics of the compiler and present some advice on how to study the program.

0.1 PARSING AND MACRO-GENERATION

The ALGOL 60-compiler given in this tract applies a top-to-bottom analysis which is closely related to the syntax of the language as presented in [6]. It is divided into a large number of often rather small procedures, each with a specific task. By way of example we discuss in the following a procedure that analyses and compiles a factor. According to [6] a factor either is a primary, or is constructed from (1) a factor, (2) the operator "+" and (3) a primary. Each factor preceding a "+" can be composite itself. Thus \(ab+e\) is a factor with the following structure:

\[
\begin{array}{c}
\text{primary} \\
\text{factor}
\end{array} +
\begin{array}{c}
\text{primary} \\
\text{factor}
\end{array} +
\begin{array}{c}
\text{primary} \\
\text{factor}
\end{array}
\]

The object code for a dyadic arithmetic operation will, apart from optimization, contain the following components:

1) code for the evaluation of the first operand, delivering the result in a register dedicated to arithmetic results,
2) code that puts the result on the top of the stack,
3) code for the evaluation of the second operand, delivering the result in the register mentioned above,
4) code for the operation, to be carried out on the number on the top of the stack and on the result in the register.
The result of the operation is delivered in the same register.

We can indicate the code for step 2 by a symbolic instruction STACK and the code for step 4, in the case of exponentation, by TTP (short for "to the power"). Probably, such a symbolic instruction has to be replaced by a number of machine instructions in the actual object program. But the introduction of symbolic instructions (in the next we will often refer to them as macros) is extremely useful for the separation between the machine-independent part of the compiler (the syntax analyser and the macro generator) and its machine-dependent part (the macro processor). They can be seen as instructions for some hypothetical machine that can be implemented on various equipment.

If in the expression \(a^{b+c}\) the identifiers \(a\), \(b\) and \(c\) correspond to simple variables of type real and we introduce a macro TRV (short for "take real variable") for the action of loading the value of such a variable into the arithmetic register, the translation of the given expression in terms of macros reads:

<table>
<thead>
<tr>
<th>TRV(a)</th>
<th>load the value of (a) into the register</th>
</tr>
</thead>
<tbody>
<tr>
<td>STACK</td>
<td>save the register in the stack</td>
</tr>
<tr>
<td>TRV(b)</td>
<td>load the value of (b) into the register</td>
</tr>
<tr>
<td>TTP</td>
<td>compute (top of stack)()register, i.e. (a^b)</td>
</tr>
<tr>
<td>STACK</td>
<td>save the register (i.e. (a^b)) in the stack</td>
</tr>
<tr>
<td>TRV(c)</td>
<td>load the value of (c) into the register</td>
</tr>
<tr>
<td>TTP</td>
<td>compute (top of stack)()register, i.e. ((a^b)^c)</td>
</tr>
</tbody>
</table>

The identifiers \(a\), \(b\) and \(c\) occur here as macro parameters, i.e. as parameters to the macro TRV. In the final machine instructions, to be produced out of this macro, they will be replaced by addresses, given to them by the compiler.
After these introductory remarks we present a procedure Factor as an example of a compiler procedure. This procedure has to analyse and compile a factor from an arithmetic expression. It assumes that the first basic symbol of that factor is available in the compiler variable Last symbol (coded in some internal representation), that subsequent basic symbols can be read into that variable by means of calls of a procedure next symbol, and that it leaves in that variable the first basic symbol not belonging to the factor. For the analysis and compilation of primaries a procedure Primary is available with analogous specification. The text of Factor reads:

```
procedure Factor;
begin integer ls;
    Primary;
    for ls := last symbol while ls = ttp do
        begin Macro(STACK);
        next symbol; Primary;
        Macro(TTP)
    end
end Factor;
```

In this procedure the identifier ttp denotes a variable initialised with the internal representation of the basic symbol "#". In the translation for basic given above the macro TRV is produced by (three successive calls of) Primary, the macros STACK and TTP by Factor itself.

We see that the procedure Factor serves two purposes: source text reading (and analysis) and macro generation. It uses a procedure Primary to deal with its substructures. In its turn, however, a factor is a building block for a term. This is reflected in the compiler procedure Term that calls Factor for the treatment of factors. Terms are components of simple arithmetic expressions and those in turn components of arithmetic expressions or of relations, and again these structures can be found back in the corresponding compiler procedures.
The procedure Factor given on page 46 differs from the one presented above in two respects:
1) it is divided into two separate procedures called Factor and Next primary.
2) iteration has been replaced by recursion.

The existence of a separate procedure Next primary is useful in those situations in which the first primary in an arithmetic expression has been read before it is decided to compile an arithmetic expression, either because this could not be decided in advance or for reasons of optimization. We come back to this point in the next section. The replacement of iteration by recursion has been carried out at many places throughout the compiler for theoretical reasons mainly.

0.2 LEFT-TO-RIGHT SCANS

In the previous section we stated that the ALGOL 60 compiler given in this tract consists of a large number of procedures. To most syntactic variables (like <statement>, <expression> or <primary>) correspond procedures, which, in general, read, analyse and compile a consecutive piece of source text, often calling other procedures for dealing with substructures. In this way, the complete source text of a program is read, analysed and compiled by one call of the procedure Program (see page 77). In this process, the source text is scanned from left to right by successive calls of next symbol, delivering the next basic symbol in the variable last symbol. Each basic symbol can, in this way, be inspected several times and from different procedures, until the next basic symbol is read. Then the previous one is lost forever, and all decisions to be based on it must have been taken; the compiler now uses back-tracking and the syntactic analysis is completely a direct process (we will not go into what properties of the ALGOL 60 syntax allow for such a compiler).

In general, the process of analysis is one chain of decisions of how to continue and what to do next; those decisions are based on the basic symbol most recently read and on information about the identifier most recently read. Information about identifiers can be derived from their declarations or, in the case of formal parameters, from specifications and from the ways in which they are used elsewhere in the procedure they belong to.
As an example we look to the procedure Stat (see page 68). This procedure for the analysis of a statement reads essentially:

```
procedure Stat;
begin
  integer n;
  if letter last symbol
    then begin n := Identifier;
      if Designational(n)
        then begin Label declaration(n); Stat end
      else begin if Subexpvar(n) v last symbol = colonqual
                 then Assignstat(n)
                 else Procstat(n)
        end
    end
  else ...
end Stat;
```

Decisions are taken here both on basic symbols (letter last symbol or last symbol = colonqual) or on properties of an identifier (Designational(n)), being true for label identifiers and for switch identifiers, or Subexpvar(n), being true for array identifiers and for switch identifiers).

The information about identifiers, constituting a non-context-free element in the ALGOL 60 grammar, is collected in two separate scans. The compiler, therefore, is a three-pass compiler. In all three passes the same source text is scanned from left to right, and no "intermediate language" is used. In the first scan the source text is read from paper tape; in the next two scans the basic symbols are taken from a text array, in which they are stored during the first scan. The tasks of the three scans are:

prescan0 (pages 17/24):

1) construction of a name list (or symbol table), containing all identifiers declared in the program, all formal identifiers and all labels, with all relevant data. The block structure of the program is reflected in this list;
2) construction of a constant list, containing all unsigned numbers occurring in the program except small unsigned integers;
3) construction of a text array, containing all basic symbols of the source text except those occurring in comments (in this text array are also stored all "new-line-carriage-return" symbols for the purpose of keeping track of line numbers during the next two scans).

All three lists are stored in one and the same array called space in the following way:

```
constant list  text array  name list
  |         |         |
  0         |         |
  prog base | text base |
          end of memory
          nl base
```

prescan1 (pages 25/43):
1) collection of information about the use of formal parameters within their procedure body, e.g., occurrence as left part in an assignment statement, about whether they are used as array identifier or as procedure identifier, and if so the number of subscripts or parameters, possibly type information. This information is used in the third scan for checking consistent use of the identifier throughout the procedure body and for checking the correspondence between actual and formal parameters;
2) collection of information about the use of labels (for label identifiers that are not used otherwise than immediately following a goto in the same block as the one in which they occur as label, no "pseudo label variables" are introduced at run time);
3) address assignment to all variables and pseudo variables;
4) space reservation for all variables and pseudo variables that are global to all procedures in the program, and for all own variables and own arrays. This space is reserved in the array space immediately following the constant list; moreover, pseudo label variables placed in this "static space" are given an initial value;
5) addition to the name list of all identifiers of the program for which no declaration (or occurrence as formal parameter) is found.
translate (pages 45/90):

1) syntax checking;
2) address assignment to procedure-, switch- and label identifiers (with their "program addresses");
3) generation of the object program.

The object program is placed in the array space immediately following the static space. Thus, the object program is generated in core, ready for execution: the compiler is a "load-and-go" compiler.

Historically, the third scan was developed first; afterwards the two prescans were written in such a way that they collect the information required by the third scan. This can be seen in many places; some advantages and drawbacks will be discussed later on.

With the existence of a name list in mind the meaning of the statement \( n := \text{Identifier} \) in the body of the procedure \( \text{Stat} \) given above will be clear: an identifier is read; then it is looked up in the name list; as function value of \( \text{Identifier} \) a pointer in this list is delivered. This pointer, then, can be used by Boolean procedures like \( \text{Designational} \) or \( \text{Subearvar} \), or can be handed over to other procedures (like \( \text{Assignatrat} \)).

We conclude this section, dealing with scans, by some remarks about a number of constructions in which the first few symbols are not necessarily sufficient to recognize the syntactic category. As an example let us consider:

\[
a := b[i, j+1] ...
\]

The basic symbol following "\( [\)" determines the meaning of the subscripted variable: a second left part or (part of) the right hand side expression. In such cases the object codes for both possibilities are chosen to have a common start and to differ only from the point where the discrimination is made. In the example given above the object codes read in terms of macros (with certain assumptions on the nature of the identifiers \( b, i \) and \( j \)):
The macro names TIV, TSIC, TAK, STAA and TSR are abbreviations for "take integer variable", "take small integer constant", "take array key", "stack A" and "take subscripted variable", respectively. These macros are generated, directly or indirectly, by a call of Subscripted variable occurring in the procedure Realassign on page 58 (or possibly in Intassign, if \( a \) is of type integer). If the symbol after "J" is not a colonequal, the subscripted variable is a first primary of an arithmetic expression. In that case, the first factor is completed by a call of Next primary (page 46), the first term by a call of Next factor (ibidem) and the arithmetic expression by a call of Next term (ibidem). These three procedures are called through a call of Rest of arithexp (page 52) from Realassign.

Other cases in which the syntactic category cannot be recognized immediately are:

\[ for \: i := \text{arithmetic expression} \ldots \]

in which the nature of the for list element is only known after the complete analysis and compilation of the arithmetic expression, or:

\[ \text{if} \: (...) \ldots \]

where the expression between parentheses is either a Boolean expression
or an arithmetic expression. In some exceptional cases the object codes for two possible interpretations are generated both, e.g., in the case of an unsigned integer as actual parameter, if, at the given position, an interpretation as label is not excluded.

0.3 MODULES

An important factor that contributes to the readability of especially the third scan is its modularity. By this we mean that for a study of the procedures that carry out the syntactic analysis and that generate the macros, no knowledge at all is necessary about, among others, the structure of the name list or the codes that are used in them for the description of kind and type of an identifier. All questions about properties of identifiers are put by means of procedure calls. Consequently, it is possible to choose any system whatsoever for the structure of the name list and for the codes used without any change in the procedures for text analysis and macro generation (the bulk of the third scan). In practice this fact was used during the development of the third scan for testing purposes at a time that no name list was available: all name list procedures then produced standard answers [7].

(The structure of the name list is relevant only for an analysis of the name-list procedures of the second or third scan themselves or for a study of the first scan, in which the construction of the name list - one of its major tasks - is integrated with syntax analysis to some extent. The information required for such a study is briefly presented in chapter 5).

In the sequel we describe the logical modules of the compiler. Some of them can be found in the code of more than one scan. Sometimes some of the procedures belonging to a logical module have been made global to all three scans by putting them in front of the procedures prescan0, prescan1 and translate.

1) syntax analyzer + macro generator

present in all three scans; to it belong: in prescan0 the procedures Program (page 17) up to and including Begin statement (page 23), in prescan1 the procedures Arithexp (page 25) up to an including Label declaration (page 35) and Scan code (page 42), in translate the procedures
Arithexp (page 45) upto and including Label declaration (page 77), Skip parameter list (page 88), Translate code (page 88) and Arithconstant (page 89) upto and including Relatmacro (page 89). Also the global procedures skip type declaration (page 13) upto and including skip rest of statement (page 13) belong to this module.

2) macro processor
to be found in the third scan, running from the procedure Substitute (page 77) upto and including B reaction (page 82). Its main task is to generate one or more machine instructions to each macro and to put these in the array space. The macro processor is explained more fully in the next section.

3) name-list procedures
present in all three scans; to it belong the global procedures read identifier (page 11) upto and including skip identifier (page 13), in prescan the procedures Process identifier and Identifier (page 23) and large portions of the procedures Block (pages 17, 18 and 19), Declaration list (pages 20 and 21), Label declaration (page 22) and Int Lab decleration (page 23), in prescan the procedures Add type (page 39) upto and including Identifier (page 42), and in translate the procedures Code bits (page 82) upto and including Identifier (page 87).

4) next symbol procedures
this module consists of the procedures next symbol (page 4) upto and including operator last symbol (page 9), all global. The isolation of the basic symbol from a sequence of flexowriter punchings is done hierarchically: next tape symbol converts to internal representation taking into account the most recent shift, insymbol isolates basic symbols, often by combining several symbols into one (e.g., "":" and "=" into ":="), next basic symbol deals with "new-line-carriage-return" symbols, and next symbol mainly skips comments.

5) unsigned-number procedures
unsigned number (page 10) reads and converts unsigned numbers; it is not realistic but intended only to illustrate some of the problems of number analysis and conversion. unsigned number delivers a mantissa,
being integer, in value of constant, a decimal exponent in decimal exponent and two Boolean values in real number and small. Besides unsigned number belong to this module: the global procedure unsigned integer (page 11), Store numerical constant (page 23) in prescan0 and unsigned number (page 89) in translate.

Note: procedure identifiers of global procedures are written with small letters only (except the procedure ERRORMESSAGE); the first letter of a procedure local to one of the three scans is always capital.

0.4 THE MACRO PROCESSOR

As has been stated before, the main task of the macro processor is to generate one or more machine instructions to each macro and to put these in the array space. This is actually done by the procedure Produce (page 79).

In addition to this task, the macro processor performs some local optimizations, carried out by the procedure Macro2 (page 78). By these optimizations the code for expressions like \( i + 2 \), \( x > y \) or \( -k \) is simplified considerably. Instead of:

\[
\begin{align*}
&\text{TIV}(i) \\
&\text{STACK} \\
&\text{TSIC}(2) \\
&\text{ADD}
\end{align*}
\]

for \( i + 2 \) (assuming \( i \) to be a simple variable of type integer),

\[
\begin{align*}
&\text{TIV}(i) \\
&\text{ADDSIC}(2)
\end{align*}
\]

is obtained.

This optimization is performed on the following rules:

1) each macro triple consisting of the macro STACK, a Simple arithmetic take macro (one of the macros TRV, TIV, TRC, TIC or TSIC) and an Optimizable operator (one of the macros ADD, SUB, MUL, DIV, EQU, UQU,
LES, MST, MOR or LST) is replaced by one macro according to table 4.1; 
2) each macro pair consisting of a Simple arithmetic take macro and the 
macro NEG is replaced by one macro according to the same table. 
(The properties of each individual macro can be deduced from the value 
of its macro identifier; these values are given in section 2.1 and decoded 
in table 4.2).

For application of these rules the procedure Macro2 can be in one out of 
four states, recorded in the global variable state:

\[ state = 0 \] indicates the normal state,
\[ state = 1 \] indicates that the macro STACK is in stock,
\[ state = 2 \] indicates that the macro STACK and a Simple arithmetic take 
macro are in stock (the last being specified in the global vari- 
ables stack0 and stack1),
\[ state = 3 \] indicates that a Simple arithmetic take macro is in stock (spe- 
cified as above).

The fact that the procedure Macro2 can have one or two macros in stock 
has some consequences in the case that the compiler needs the address at 
which the code for the next macro will be placed in the array space. This 
occurs e.g. frequently during the syntax analysis + macro generation for 
for-list elements. In these cases the macro processor has to be brought 
in its normal state first. This is done through the generation of the 
macro EMPTY (in the procedure Onedrcounter, page 78) to which no instruc-
tions correspond at all.

The simple optimization rules given above are rather effective: on an 
average they eliminate 53 per cent. of the macro STACK and 62 per cent. 
of the macro NEG.

Another task of the macro processor is to keep track of the changes of 
the stack pointer that will occur at run time, for the purpose of minimi-
zing the number of checks on stack-overflow at run time (the EL-XB origin-
ally did not have memory protection; therefore certain macros had to 
check for overflow). An explanation of this aspect of the macro processor 
is too lengthy to be given here; it is concentrated in the procedure
Process stack pointer (page 80) and the resulting number is added to the object program as sum of maxima (see page 50).

To the macro processor belong also the procedures Substitute and Subst2 (page 77). These have to do with forward references. As an example we discuss the expression \texttt{if b then 2 else 3}, for which the following macros are generated:

\begin{verbatim}
    TBV(b)     (take Boolean value)
    COJU(future1)    (conditional jump)
    TSIC(2)          (take small integer constant)
    JU(future2)      (jump)
\end{verbatim}

\texttt{future1:} \hspace{1cm} \texttt{TSIC(3)}
\texttt{future2:}

The addresses \texttt{future1} and \texttt{future2} are unknown to the compiler at the moment that the macros COJU and JU, respectively, are generated. Therefore, they will be set zero initially, and in the local variables \texttt{future1} and \texttt{future2} of the procedure \texttt{Arithexp} (page 45) the locations in the array \texttt{space} will be recorded at which the jump instructions are placed (the assignment to these variables occurs through the last statement of the procedure \texttt{Macro2}). As soon as the true destinations for the jumps are known they can be substituted at those locations.

If several macros refer forward to the same, yet unknown location, they are linked together in a chain in the array \texttt{space}, ending at the macro that made the first reference to that location and therefore has a preliminary address part equal to zero. The same mechanism is used for references to procedures, switches and labels that are declared later. In that case the key address of the chain is recorded in the name list (cf. the procedure \texttt{Mark position in name list}, page 86).

0.5 ERROR HANDLING

An important task of any compiler is to check against syntactic and semantic errors in the source text. In general, it is relatively easy to find out whether a text is, according to certain rules, a correct program or not. It is far more difficult to interpret an incorrect text and to try to
indicate as many errors as possible. In fact, strictly spoken this is an ill-defined task, and to any strategy counter-examples with a very bad error-message performance are easily found.

The strategy adopted by the compiler in this tract is a rather simple one. It consists of two rules mainly:

1) if, in a given compiler procedure, the occurrence of a certain basic symbol is expected but inspection of last symbol shows that it is not there, an error message is produced and in general the task of the given compiler procedure will be considered to be completed. Perhaps the trouble is passed over then to the calling procedure that, in turn, may produce an error message and pass over the problem to its calling procedure;

2) there is only one level in the compiler that is allowed to skip a piece of source text until (a) certain basic symbol(s) is (are) met: the procedure Compound tail, present both in prescanl (page 31) and translate (page 69). It requires that after the translation of the first statement of (the remainder of) a compound tail a semicolon or an end is found. Otherwise it produces (in translate) an error message and skips to the next semicolon or end (during this skip, the syntax analysis is resumed temporarily at the occurrence of a compound statement or block for the purpose of matching corresponding begin and end symbols; for analogous reasons a string (like ';' is skipped as one integral unit).

As was indicated above, the performance of such a strategy is not always good. For the source text:

```plaintext
begin integer i;
  for i:= 1 step 1 until 10 do
    print(if 2 ≤ i < 4 then 0 else i)
end
```

the following error messages are produced, all with the same basic symbol ("<") in last symbol:
<table>
<thead>
<tr>
<th>error number</th>
<th>meaning</th>
<th>produced by</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>331</td>
<td>&quot;then&quot; missing in expression of yet unknown type</td>
<td>Exp</td>
<td>55</td>
</tr>
<tr>
<td>334</td>
<td>expression starts with unadmissable symbol</td>
<td>Simplexp</td>
<td>56</td>
</tr>
<tr>
<td>332</td>
<td>&quot;else&quot; missing</td>
<td>Exp</td>
<td>55</td>
</tr>
<tr>
<td>361</td>
<td>&quot;)&quot; missing</td>
<td>Parameter list</td>
<td>63</td>
</tr>
<tr>
<td>367</td>
<td>&quot;;&quot; or &quot;end&quot; missing</td>
<td>Compound tail</td>
<td>69</td>
</tr>
</tbody>
</table>

These error messages do not seem very applicable! (note that the production of the messages 334 and 332 is caused by the fact that the procedure Exp does not strictly apply the first rule given above). However, in daily practice most error messages are to the point and in exceptional cases only it is hard to trace the true error(s) in the source text. Most important, moreover, is the fact that the compiler keeps on the rails, even in case of strong "misunderstanding" of a text.

A number of errors is detected in all three scans of the compiler. In general, these errors are reported in the third scan only (cf. the difference between the procedures Compound tail in prescan1 and translate).

This is done for two reasons: the third scan has been developed first, and it has to perform a more detailed analysis of the source text than the preceding scans. The fact that error messages are postponed to the last scan whenever possible has one important disadvantage: some texts never reach this scan (e.g. through an "end of file" in the first scan). A better approach would have been to report all errors related to delimiters in the first scan and all errors related to types is the third scan.

The check against errors is rather extended. It includes an effective check on the correspondence of actual and formal parameters, except for function designators and procedure statements with formal procedure identifier. A complete check on this correspondence can be carried out at run time only, but for the sake of run-time efficiency this has been omitted (this has caused troubles in, on an average, one out of 30 000 programs).
0.6 EVALUATION

The compiler presented in this tract has been in operation (in its machine-code version) for about 8 years now and it has had considerable influence on the use of ALGOL 60 in the Netherlands. Its success is due to several external factors, like the presence of an EL-XS at several important institutions and the absence of an equivalent FORTRAN compiler for that machine. But certainly its success also resulted from a number of properties of the ALGOL system itself, like:

1) it is simple to use, even by the inexpert programmer;
2) it accepts rather complete ALGOL 60; the exceptions are well-defined;
3) it has good and rather complete diagnostics;
4) it is almost free of errors; exotic constructions and complex combinations do work properly;
5) the compiler is fast (although its name-list technique is rather primitive): on an average some 12 per cent. of the processing time of a program is compile time;
6) the compiler is rather short: it does not exceed 5000 instructions;
7) the object program is, given the absence of any advanced optimizing, rather efficient: on an average some 25 per cent. of the processing time of a program is used for arithmetic operations and for the evaluation of standard functions;
8) it has been easy to adapt the system to new situations, like extension of core memory, new operating systems or addition of a library of pre-compiled ALGOL procedures.

Most of these properties originate from the transparent structure of the compiler. It proved to be possible to run hundreds of ALGOL 60 programs a day, often produced by inexpert programmers, with only one or two people for user assistance.

There are also some weak points:

1) exceptionally, lacking correspondence of actual and formal parameters leads to troubles at run time;
2) exceptionally again, incorrect programs derail the compiler, causing addressing errors. This point is caused by the fact that, for some incorrect texts, the three scans of the compiler make a different interpretation of the block structure of that text. Partly this originates
from the structure of the first scan, which does not analyse statements except for the occurrence of labels, for clauses, compound statements or blocks (prescan0 has been called sometimes "the art of skipping"). This fact could be improved by writing a more extended first scan. But that would not remedy the problem completely. For, another reason for incongruency of block-structure interpretation is the insystematic way in which, in the scans, decisions are taken: now on the contents of last symbol, then again on properties of some identifier. However, the properties of identifiers are not equally available in all three scans, and therefore they are not suited to base a parsing algorithm on;

3) some 23 per cent. of the processing time of a program is related to the access to array elements. The performance of the indexing system could have been improved by a number of measures like special index routines for one- and two-dimensional arrays or, in the case of an assignment to a subscripted variable, indexing before the right hand side expression is evaluated;

4) ambitious plans to implement own dynamic arrays and advanced string operations (hence the extension of the expression types with a type "string") never were completed, although the compiler is prepared for it completely.

It has been shown afterwards that the first three points can be fully circumvented with no essential change in the methods used for the ALGOL 60 system described here.

We conclude this section by a few remarks on the question how machine-(in) dependent this compiler is. The most important property of the EL-X8 in this respect is its arithmetic. The EL-X8 uses the Grau-representation [8] for floating-point numbers. Consequently, integers form a true subset of the reals and mixed-mode arithmetic expressions are no problem at all: no types are distinguished for arithmetic expressions. Secondly, all arithmetic operations are performed in the floating-point register and not in the top of a stack. In the other case no explicit stack operations would have been necessary. Another property of the EL-X8 is its addressing system. This, however, can hardly be found in the syntax analyzer; it has, of course, influenced the macro processor.
0.7 HINTS TO THE READER

For a study of the ALGOL text of the compiler the following order is proposed:
1) the compilation of arithmetic expressions (Arithexp upto and including Subscript list);
2) the macro processor (forget about Process stack pointer and Process parameter);
3) the compilation of Boolean expressions (Boolexp upto and including Arboolrest);
4) the compilation of string expressions (Stringexp sq.), designational expressions (Desigexp sq.) and expressions of unknown type (Exp sq.);
5) the compilation of statements (Statement sq.);
6) the compilation of blocks (Block);
7) the compilation of programs (Program);
8) the syntax-analysis part of prescan1;
9) the name-list procedures of translate and of prescan1;
10) the compilation of actual parameters (Parameter list);
11) prescan0.

Try always to formulate the task of yet unaalysed procedures!
The page on which each compiler procedure can be found is easily found in the cross-reference table (page 157).
REFERENCES

[1] NEDERKOORN, J.J.B.M., Prolegomena to X8 ALGOL,
    Amsterdam, Mathematical Centre, report R 989 (1964).

    MC ALGOL 60 system for the EL X8,
    Amsterdam, Mathematical Centre, report MR 8k (1966).

    X8 - ALGOL 60 - vertaler van het MC,
    Amsterdam, Mathematical Centre, report MR 121 (1971).

    Programmeren voor rekenautomaten; de MC ALGOL 60 vertaler
    voor de EL X8,
    Amsterdam, Mathematical Centre, MC Syllabus 13 (1971).

    numbers during the execution phase of ALGOL 60 programs, in
    MC - 25 Informatica Symposium,
    Amsterdam, Mathematical Centre Tracts 37 (1971).

    Copenhagen, Regnecentralen (1962).

[7] KRUSEMAN ARETZ, F.E.J., ALGOL 60 Translation for Everybody,
    Elektronische Datenverarbeitung 9 (1964), 233-244.

[8] GRAU, A.A., On a Floating-Point Number Representation For Use
    with Algorithmic Languages,
1. THE ALGOL 60 PROGRAM

begin
comment ALGOL 60 — version of the ALGOL 60 — translator
for the EL — X8, F.E.J. Kruseman Arets;

comment basic symbols;
Integer plus, minus, m1, div, idi, ttp, equ, uqu, les, mst, mor, lst,
non, qvl, imp, or, and, goto, for, step, until, while, do,
comm, period, ten, colon, semicolon, colonequal, space sbl,
if, then, else, comment, open, close, sub, bus, quote, unquote,
begin, end, own, ren, integ, boole, stri, array, proceed, switch,
label, value, true, false, new line, underlining, bar;

comment other global integers;
Integer case, lower case, stock, stock1, last symbol, line counter,
last identifier, last identifier1,
quote counter, run number, shift,
type, char, character, value character, arr decla macro,
value of constant, decimal exponent, decimal count,
word count, nlp, last nlp, n, integer label,
block cell pointer, next block cell pointer,
dimension, forcount, instrct counter, dp0,
function letter, function digit, c variant,
nl base, prog base, text base, text pointer,
end of text, end of memory, start, end of list,
d0, d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12, d13, d14,
d15, d16, d17, d18, d19, d20, d21, d22, d23, d24, d25,
re, in, bo, st, ar, nondes, des, un, arbo, intlab;

comment macro identifiers;
Integer STACK, NE3, ADD, SUB, MIL, DIV, IDI, TTP,
EQ, UQU, LESS, MST, MOR, LST, START, NON, QVL, IMP, OR, AND,
STIA, TSI, TSB, TSST, TPSU, TSL, TPSL, TST,
STIR, STSI, STSS, STSE, STSST, STPSU,
ENTRIES, TFP, SAS, DECS, FAD, TASH, TASI, TASHB, TASS, TASIU,
EXITST, FADCV, TSCV, TSCVU, TSCV, EXIT, TESTT, TESTT2,
CV, CTV, CVN, CSVN, CVL, CEN, CLIP, TAV, TIAV,
RAD, LAD, BAD, STAB, CRAB, GIDE, CRAB, GSTE, CRSTAB,
ICS, EXTIP, EXITPC, REJST, JUA, EMPTY,
ABS, SIGN, ENTER, SQRT, EXP, LN, END;

comment macro2 identifiers;
Integer TRW, TIV, TPC, TIC, TSC, TBV, TBC, TSTV, TLY, TAK, TSWE,
STI, STI, SSTIT, STB, SSTIT, DOS, DOS2, DOS3,
JU, JU1, LU, LUI, CUA, YCNU, SUBJ, ICUB, ICUE, DECB, DO,
TRR, ENTER, DPT, INCUR, TDL, ENTERPB, NIL, LAST,
LAD, TDA, TNA, TAA, SWF, EXTI, EXITPC, EXITC, EXITSV,
CODE, SLNC, RINC, LNC;

comment global Booleans;
Boolean letter last symbol, digit last symbol, arr declarator last symbol,
type declarator last symbol, in array declaration, in formal list,
text in memory, own type, int labels, real number, small,
erroreous, derraneous, wanted;
comment global arrays:

integer array internal representation[0 : 127], word delimiter[0 : 23],
macro list[0 : 511], table[5 : 59],
instruct list[0 : 203], mask[0 : 9];

comment start of initialization;
plus:= read; minus:= read; mul:= read; div:= read;
add:= read; ttp:= read; equ:= read; uqu:= read;
les:= read; mat:= read; xor:= read; lst:= read;
non:= read; qvl:= read; imp:= read; ori:= read;
and:= read; goto:= read; for:= read; step:= read;
until:= read; while:= read; do:= read; comma:= read;
period:= read; ten:= read; colon:= read; semicolon:= read;
colonequal:= read; space =:= read; if:= read;
else:= read; comment:= read; open:= read; close:= read;
sub:= read; bus:= read; quote:= read; unquote:= read;
beg:= read; end:= read; own:= read; res:= read;
integ:= read; boole:= read; stri:= read; array:= read;
proced:= read; switch:= read; label:= read; value:= read;
true:= read; false:= read; new line:= read;
underlining:= read; bar:= read; lower case:= read;

STACK := read; NEG := read; ADD := read; SUB := read;
MUL := read; DIV := read; IDI := read; TTP := read;
EQU := read; UQU := read; LES := read; MST := read;
MOD := read; LST := read; STAB := read; MCM := read;
QVL := read; EXP := read; OR := read; AND := read;
STA := read; TST := read; TSI := read; TSB := read;
TSST := read; TPSU := read; TSL := read; TPSL := read;
TCST := read; STSR := read; STSI := read; SINSI := read;
STSS := read; STSSST := read; STPSU := read; ENTRIES := read;
TPO := read; SAS := read; PECS := read; PAD := read;
TAS := read; TASI := read; TSH := read; TASST := read;
TISP := read; ETRIPS := read; PADC := read; TSCV := read;
TSCV := read; TSCVU := read; EXT := read; TSSI := read;
TEST1 := read; CV := read; CIV := read; CSV := read;
CST := read; CLV := read; CEN := read; CLPS := read;
TAV := read; TLA := read; RAD := read; IAD := read;
BAD := read; STAD := read; ORAD := read; OAD := read;
ORD := read; CSTAD := read; IOS := read; EXITP := read;
EXITP := read; ERUST := read; JUL := read; EMPTY := read;
ABS := read; SIGN := read; ENTER := read; SQX := read;
EXP := read; LN := read; END := read;
begin integer array space[0 : end of memory];

procedure ERRORMESSAGE (n); integer n;
begin Integer 1;
  erroneous := true;
  if n = 122 \ V n = 123 \ V n = 126 \ V n = 127 \ V n = 129
  then erroneous := true;
  if n > run number
  then Begin NLCR; PRINTTEXT {ker}; print (n); 
                   print (line counter); print (last symbol);
           for i := 0 step 1 until word count do 
             print (space[nl base - last nlp - 1])
  end
end ERRORMESSAGE;
integer procedure next symbol;
begin
integer symbol;
next0: symbol:= if stock1 > 0 then stock1 else next basic symbol;
      stock1:= -1;
      if (last symbol = semicolon V last symbol = begin) \symbol = comment
      then begin skip0: symbol:= next basic symbol;
           if symbol != semicolon then goto skip0;
           goto next0
      end;
      if last symbol = end
      then begin
          skip1: if symbol != end \ symbol != semicolon \ symbol != else
          then begin symbol:= next basic symbol; goto skip1
      end
      else
          if symbol = 125
          then begin stock1:= next basic symbol;
                if stock1 > 9 \ stock1 < 64
                then begin skip2: stock1:= next basic symbol;
                    if stock1 > 9 \ stock1 < 64
                    then goto skip2;
                    if stock1 = colon
                    then stock1:= next basic symbol
                    else ERROMESSAGE (100);
                    if stock1 = open then stock1:= - stock1
                    else ERROMESSAGE (101);
                symbol:= comma
          end
          else symbol:= close
      end;
digit last symbol := symbol < 10 \ symbol = period \ symbol = ten;
letter last symbol:= symbol < 64 \ 1 digit last symbol;
next symbol:= last symbol:= symbol;
outsymbol (run number, symbol);
test pointers
end next symbol;

integer procedure next basic symbol;
begin
integer symbol;
next0: insymbol (run number, symbol);
if symbol = new line
  then begin line counter:= line counter + 1;
     if quote counter = 0
     then begin outsymbol (run number, symbol);
        goto next0
  end
  next basic symbol:= symbol
end next basic symbol;
procedure insymbol (source, destination); integer source, destination;
begin Integer symbol, i;
if (source = 200 v source = 300) w text in memory
then
begin destination:= bit string(d8 x shift, shift,
space[2text base + text pointer]);
if shift < 257
then shift:= d8 x shift
else begin shift:= 1; text pointer:= text pointer + 1 end
end
else
begin symbol:= if stock > 0 then stock else next tape symbol;
stock:= -1;
if symbol > bus
then
begin if symbol = 123 then symbol:= space sb1;
if quote counter > 0
then
begin if symbol = bar
then
begin next0: stock:= next tape symbol;
if stock = bar then goto next0;
if stock = les
then quote counter:= quote counter + 1
else
if stock = mor
then
begin if quote counter = 1
then begin symbol:= unquote;
stock:= -symbol
end
else quote counter:=
quote counter - 1
end
end
else if symbol = 124
then symbol:= colon
else if symbol = 125 then symbol:= close
end
else
if symbol > new line
then
begin if symbol = bar
then
begin next1: symbol:= next tape symbol;
if symbol = bar then goto next1;
symbol:= if symbol = and then ttp else
if symbol = equ then ugu else
if symbol = les then quote else
if symbol = mor then unquote
else 160
end
else
else
if symbol = underlining
  then
  begin
    symbol := the underlined symbol;
    if symbol > 63
  then
    symbol :=
    if symbol = 124 then id1 else
    if symbol = less then met else
    if symbol = more then 1st else
    if symbol = non then imp else
    if symbol = equ then qvl
      else 161
  end
else
  begin
    stock := next tape symbol;
    if stock = underlining
  then
    begin
      symbol := the underlined symbol +
      d7 \times symbol;
      for i := 0 step 1 until 23 do
      begin
        if word delimiter[i] \div d7 = symbol
        then
          begin
            symbol := word delimiter[i];
            symbol := symbol -
            symbol \div d7 \times d7;
            goto next2
          end;
        end;
      symbol := 162;
      next2: stock := next tape symbol;
    if stock = underlining
    then
      begin
        the underlined symbol;
        goto next2
      end;
    end
  end
else
  if symbol = 124
  then begin
    stock := next tape symbol;
  if stock = equ
  then begin
    symbol := colon equal;
    stock := = symbol
  end
  else symbol := colon
  end;
else
  insymbol (runnumber, symbol)
end;
destination := symbol
end insymbol;
integer procedure the underlined symbol;
begin
  integer symbol;
  symbol := next tape symbol;
  the underlined symbol := if symbol = underlining then the underlined symbol else symbol
end
the underlined symbol;

integer procedure next tape symbol;
begin
  integer symbol, head;
  symbol := internal representation[RESEP];
  if symbol > 0 then begin
    head := symbol \* d9;
    next tape symbol := abs (if case = lower case then symbol \* d9 \* head else head)
  end
  else begin
    if symbol < -2 then case := - symbol else
    if symbol = 0 then ERRORMESSAGE (102) else
    if symbol = -1 then ERRORMESSAGE (103);
    next tape symbol := next tape symbol
  end
end
next tape symbol;

procedure outsymbol (destination, source); integer destination, source;
begin
  if destination = 100 \* text in memory
  then begin
    space [text base + text pointer] :=
    space [text base + text pointer] \* shift \* source;
    if shift < 257 then shift := d9 \* shift
    else begin
      shift := 1; text pointer := text pointer + 1;
      space [text base + text pointer] := 0
    end
  end
end
outsymbol;

Boolean procedure arithoperator last symbol;
begin
  arithoperator last symbol := last symbol = plus \lor
  last symbol = minus \lor
  last symbol = mul \lor
  last symbol = div \lor
  last symbol = idt \lor
  last symbol = ttp
end
arithoperator last symbol;
Boolean procedure relatoperator last symbol;
begin  relatoperator last symbol:= last symbol = les \lor
    last symbol = met \lor
    last symbol = equ \lor
    last symbol = lac \lor
    last symbol = mor \lor
    last symbol = ugu
end  relatoperator last symbol;

Boolean procedure booloperator last symbol;
begin  booloperator last symbol:= last symbol = qvl \lor
    last symbol = imp \lor
    last symbol = or \lor
    last symbol = and
end  booloperator last symbol;

Boolean procedure declarator last symbol;
begin own type:= last symbol = own; if own type then next symbol;
type:= if last symbol = rea then 0 else
      \text{If } last symbol = integ \text{ then } 1 \text{ else}
      \text{If } last symbol = boole \text{ then } 2 \text{ else}
      \text{If } last symbol = stri \text{ then } 3 \text{ else } 1000;
      \text{if type < 4 then next symbol}
      \text{else begin if own type then ERRORMESSAGE (104);}
      \text{If } last symbol = array \text{ then type:= 0}
end;
arr declarator last symbol:= last symbol = array;
if arr declarator last symbol \land run number = 300
then arr decla macro:= if own type
      \text{Then (if type = 0 then ORAD else}
      \text{If type = 1 then OIAD else}
      \text{If type = 2 then ORAD else OSTD)\text{else (if type = 0 then RAD else}
      \text{If type = 1 then IAD else}
      \text{If type = 2 then RAD else STAD);}
chara:= if arr declarator last symbol
  \text{Then 8}
else if last symbol = switch
  \text{then 14}
else if last symbol = proced
  \text{then (if type < 4 then 16 else 24)}
else type;
type declarator last symbol:= chara < 4;
if own type \land chara > 8 then ERRORMESSAGE (105);
\text{If type < 4 \land last symbol = switch then ERRORMESSAGE (106);}
\text{If chara < 25 \land run number = 100}
then character:= (((if type declarator last symbol
  \text{then type}
  \text{else if type < 4}
  \text{then type + chara}
  \text{else chara}) +
  (if own type then 32 else 0)) \times d19;
declarator last symbol:= chara < 25
end declarator last symbol;
Boolean procedure specifier last symbol;
begin type:= if last symbol = rea then 0 else
if last symbol = integ then 1 else
if last symbol = boole then 2 else
if last symbol = stri then 3 else
if last symbol = array then 5 else 1000;
if type < 4 then next symbol;
chara:= if last symbol = label then 6 else
if last symbol = switch then 14 else 1000;
if type + chara < 1000 then ERROUMESSAGE (107);
chara:= if last symbol = array then 8 else
if last symbol = proced then (if type < 4 then 16
else 24)
else chara;
if chara < 25 then next symbol;
if chara + type < 2000 \& run number = 100
then begin value character:= (if chara > 8 then type else
if chara = 6 then 6 else
if type = 5 then 8
else type + chara) + 64;
character:= ((if type > 5
then chara
else (if type > 1 then type else 1) +
( if chara < 1000 then chara
else 0))) + 96) x d19
end;
specifier last symbol:= chara + type < 2000
end specifier last symbol;

Boolean procedure operator last symbol;
begin operator last symbol:= arithoperator last symbol \&
relatoperator last symbol \&
booloperator last symbol
end operator last symbol;
procedure unsigned number;
begin  integer sign of exponent;
  if last symbol < 10
    then begin value of constant := unsigned integer (0);
            real number := digit last symbol
    end
  else begin value of constant := if last symbol = ten then 1
                        else 0;
        real number := true
    end;
decimal exponent := 0;
if real number
  then begin
    next0: if last symbol < 10
          then begin decimal exponent := decimal exponent + 1;
               next symbol; goto next0
          end;
    if last symbol = period
      then begin next symbol;
          value of constant :=
             unsigned integer (value of constant);
          decimal exponent := decimal exponent -
             decimal count;
        next1: if last symbol < 10
            then begin next symbol; goto next1 end
        end;
    if last symbol = ten
      then begin next symbol; sign of exponent := 1;
          if last symbol = plus
            then next symbol
          else if last symbol = minus
            then begin next symbol;
                sign of exponent := -1
            end;
          decimal exponent := decimal exponent +
             sign of exponent x
             unsigned integer (0);
          if last symbol < 10
            then begin ERROMESSAGE (109);
               next2: if next symbol < 9
                     then goto next2
        end
        end;
small := value of constant < d15 ∧ 1 real number
end unsigned number;
integer procedure unsigned integer (start); integer start;
begin integer word;
  word:= start; decimal count:= 0;
  if last symbol > 9 then ERRORMESSAGE (109);
next0: if last symbol < 10
  then begin if word < 6710886 \ (word = 6710886 \ last symbol < 4)
    then begin word:= 10 \ word + last symbol;
      decimal count:= decimal count + 1;
      next symbol; goto next0
    end
  end;
  unsigned integer:= word
end unsigned integer;

procedure read identifier;
begin integer word, count;
  word:= count:= word count:= 0;
  if letter last symbol
  then begin
next0: if last symbol < 64
    then if count = 4
      then begin word count:= word count + 1;
        word:= count:= 0
      end;
      word:= space[nl base - nlp - word count]:= d6 \ word - last symbol - 1;
      count:= count + 1; next symbol; goto next0
    end
  else begin last identifier:= space[nl base - nlp];
    last identifier:= if word count = 0
      then 0
      else space[nl base - nlp - 1]
  end
end read identifier;

integer procedure next pointer (n); integer n;
begin integer word, pointer;
  pointer:= n;
next0: word:= space[nl base - pointer];
  if word < 0 then begin pointer:= pointer + 1; goto next0 end;
  if word \ d25 then begin pointer:= word \ d13 \ d13;
      goto next0
  end;
  next pointer:= pointer
end next pointer;
integer procedure look up;
begin
  integer count, pointer;
  pointer := block cell pointer +
  (if in formal list V in array declaration
   then 5 else 4);
next0:  pointer := next pointer (pointer);
for count := 0 step 1 until word count do
  begin
    if space[nl base - pointer - count] ≠
      space[nl base - last nlp - count]
      then goto next1;
  end;
  pointer := pointer + word count + 1;
  if space[nl base - pointer] < 0
  then begin
    next1:  pointer := pointer + 1;
    goto if space[nl base - pointer] < 0 then next1
      else next0;
  end;
  look up := pointer
end look up;

Boolean procedure in name list;
begin
  integer head;
  if real number V 1 int labels
  then in name list := false
  else begin
    head := value of constant : d18;
    space[nl base - nlp] := d12 - head;
    space[nl base - nlp - 1] :=
      (head - 1) x d18 - value of constant;
    word count := 1;
    space[nl base - nlp - 2] := 6 x d19;
    last nlp := nlp;
    integer label := look up;
    in name list := integer label < nlp
  end
end in name list;

integer procedure next identifier (n); integer n;
begin
  integer pointer;
  pointer := next pointer (n) + 1;
next0: if space[nl base - pointer] < 0
  then begin
    pointer := pointer + 1; goto next0 end;
  next identifier := pointer
end next identifier;
procedure skip identifier;
begin
if last symbol < $<$ then begin next symbol; skip identifier end
end skip identifier;

procedure skip type declaration;
begin
if letter last symbol then skip identifier;
if last symbol = comma then begin next symbol; skip type declaration end
end skip type declaration;

procedure skip value list;
begin
if last symbol = value then begin next symbol; skip type declaration;
if last symbol = semicolon then next symbol
end skip value list;

procedure skip specification list;
begin
if specifier last symbol then begin skip type declaration;
if last symbol = semicolon then next symbol; skip specification list
end skip specification list;

procedure skip string;
begin
quote counter:= 1;
next0: if next symbol # unquote then goto next0;
quote counter:= 0
end skip string;

procedure skip rest of statement (pr); procedure pr;
begin
if last symbol = do then begin next symbol; pr end
else
if last symbol = goto \ last symbol = for \ last symbol = begin then pr;
if last symbol = quote then skip string;
if last symbol # semicolon \ last symbol # end then begin next symbol;
end skip rest of statement (pr)
end skip rest of statement;
integer procedure bit string (kn, n, code Word); integer kn, n, code word;
begin integer k;
  k = code word \* kn; bit string := (code word - k \* kn) \* n
end bit string;

integer procedure display level;
begin display level :=
  bit string (d6, d0, space[nl base - block cell pointer - 1])
end display level;

integer procedure top of display;
begin top of display :=
  bit string (d13, d6, space[nl base - block cell pointer - 1])
end top of display;

integer procedure local space;
begin local space := space[nl base - block cell pointer - 1] \* d13
end local space;

integer procedure proc level;
begin proc level :=
  bit string (d6, d0, space[nl base - block cell pointer - 2])
end proc level;

Boolean procedure use of counter stack;
begin use of counter stack :=
  bit string (d7, d6, space[nl base - block cell pointer - 2]) = 1
end use of counter stack;

integer procedure status;
begin status := space[nl base - block cell pointer - 2] \* d13
end status;

Boolean procedure in code (n); integer n;
begin in code := bit string (d25, d24, space[nl base - n - 1]) = 1
end in code;

integer procedure type bits (n); integer n;
begin type bits := bit string (d22, d19, space[nl base - n])
end type bits;
Boolean procedure local label (n); integer n;
begin
  local label:=
  nonformal label (n) \wedge
  bit string(d6, do),
  space[nl base - corresponding block cell pointer (n) - 1]) =
  display level
end local label;

Boolean procedure nonformal label (n); integer n;
begin
  nonformal label:= space[nl base - n] :: d19 = 6
end nonformal label;

integer procedure corresponding block cell pointer (n); integer n;
begin
  integer p;
  p:= block cell pointer;
next0: if n < p \lor (n > space[nl base - p - 2] :: d13 \wedge p > 0)
  then begin p:= space[nl base - p] :: d13 goto next0 end;
  corresponding block cell pointer:= p
end corresponding block cell pointer;

procedure entrance block;
begin
  block cell pointer:= next block cell pointer;
  next block cell pointer:=
  bit string (d13, do, space[nl base - block cell pointer])
end entrance block;

procedure exit block;
begin
  block cell pointer:= space[nl base - block cell pointer] :: d13
end exit block;

procedure init;
begin
  stock:= stock1:= last symbol:= word count:= -1;
  shift:= 1;
  line counter:= quote counter:= forcount:= 0;
  in formal list:= in array declaration:= false;
  case:= lower case; text pointer:= 0
end init;
procedure test pointers;
begin
  Integer fprog, fnl, i, shift;
  if text in memory then
    begin
      fprog := text base +
        (if runumber = 300 then text pointer else 0) -
        instruct counter;
      fnl := nl base - nlp -
        (text base +
        (if runumber = 100 then text pointer
          else end of text));
      if fprog + fnl < 40
        then begin text in memory := false; test pointers end
      else if fprog < 20
        then begin
          shift := (fnl - fprog) div 2;
          for i := text base + text pointer
do
            step - 1 until text base do
            space[i + shift] := space[i];
          text base := text base + shift
        end
      else if fnl < 20
        then begin
          shift := (fprog - fnl) div 2;
          for i := text base step 1
            until text base + text pointer do
            space[i] := space[i + shift];
          text base := text base - shift
        end
end
end test pointers;
procedure prescanC;
begin
  Integer old block cell pointer, displ level, proc level,
  global count, local count, label count, local for count,
  max for count, internal block depth, string occurrence,
  subcount, array pointer;
end

procedure Program;
begin
  Integer n;
  character:= 6 x d19;
  if letter last symbol
    then begin read identifier;
      if last symbol = colon
        then begin n:= Process identifier;
          Label declaration (n)
        end
      else ERROMESSAGE (111);
    Program
  end
  else
    if digit last symbol
      then begin unsigned number;
        if last symbol = colon then Int lab declaration
          else ERROMESSAGE (112);
      Program
    end
  else
    if last symbol = begin
      then Begin statement
      else begin ERROMESSAGE (113); next symbol; Program end
    end Program;

integer procedure Block (proc identifier); integer proc identifier;
begin
  integer dump1, dump2, dump3, dump4, dump5, dump6, dump7, dump2,
        n, formal count;
  dump1:= block cell pointer; dump2:= local for count;
  dump3:= max for count; dump4:= local count;
  dump5:= label count; dump6:= internal block depth;
  dump7:= string occurrence; dump8:= proc level;
  local for count:= max for count:= local count:= label count:=
  internal block depth:= string occurrence:= 0;
  block cell pointer:= nlp + 1;
  space[nl base - old block cell pointer]:= space[nl base - old block cell pointer] + block cell pointer;
  old block cell pointer:= block cell pointer;
  space[nl base - block cell pointer]:= dump1 x d13;
  space[nl base - block cell pointer - 1]:= displ level:= displ level + 1;
  space[nl base - block cell pointer - 3]:= 0;
  nlp:= nlp + 6;
if proc identifier > 0
  then
    begin
      proc level:= displ level; formal count:= 0;
      space[nl base - block cell pointer - 4]:= - d25 - nlp;
      if last symbol = open
        then begin
            character:= 127 × d19;
            next0: next symbol; Identifier;
            space[nl base - nlp]:= 0; nlp:= nlp + 1;
            formal count:= formal count + 1;
            if last symbol = comma then goto next0;
            if last symbol = close then next symbol
            else ERROMESSAGE (114)
        end;
      if last symbol = semicolon then next symbol
        else ERROMESSAGE (115);
      space[nl base - proc identifier - 1]:= d22 + formal count + 1;
      if last symbol = value
        then
          begin
            next1: next symbol; n:= Identifier;
            if n > last nlp then ERROMESSAGE (116)
              else space[nl base - n]:= 95 × d19;
            nlp:= last nlp;
            if last symbol = comma then goto next1;
            if last symbol = semicolon then next symbol
              else ERROMESSAGE (117)
          end;
          next2: if specifier last symbol
            then
              begin
                next3: n:= Identifier;
                if n > last nlp
                  then ERROMESSAGE (118)
                    else if space[nl base - n] = 127 × d19
                      then space[nl base - n]:= character
                      else if space[nl base - n] ≠ 95 × d19
                        then ERROMESSAGE (119)
                        else if value character > 75
                          then ERROMESSAGE (120)
                          else
                            begin
                              space[nl base - n]:= value character × d19;
                              if type = 3
                                then string occurrence:= d6
                            end;
                nlp:= last nlp;
                if last symbol = comma
                  then begin
                      next symbol; goto next3 end;
                  if last symbol = semicolon then next symbol
                    else ERROMESSAGE (121);
              end;
          goto next2
space[nl base - nlp] := - d25 - 4 - dump1; nlp := nlp + 1;
space[nl base - block cell pointer - 4] := - d25 - nlp;
if last symbol = quote
  then begin
    space[nl base - proc identifier - 1] :=
    space[nl base - proc identifier - 1] + d25;
    next4 := next symbol;
    if last symbol ≠ unquote then goto next4;
  end
else
  if last symbol = begin
    then begin
      next symbol;
      if declarator last symbol then Declaration list;
      Compound tail; next symbol
    end
  end else Statement
else
begin
  space[nl base - nlp] := - d25 - 4 - dump1; nlp := nlp + 1;
  space[nl base - block cell pointer - 4] := - d25 - nlp;
  Declaration list; Compound tail
end;

space[nl base - block cell pointer - 2] :=
d13 × nlp + string occurrence + proc level;
for i := 0 step 1 until max for count - 1 do
  space[nl base - nlp - i] := d19;
space[nl base - block cell pointer - 1] :=
space[nl base - block cell pointer - 1] +
d6 × (internal block depth + 1);
if proc level > 1
then
  space[nl base - block cell pointer - 1] :=
  space[nl base - block cell pointer - 1] +
d13 × (max for count + local count) +
  else
    global count := global count + max for count +
    local count + label count;
  nlp := nlp + max for count;
  space[nl base - nlp] := - d25 - 5 - block cell pointer;
  nlp := nlp + 1;
  space[nl base - block cell pointer + 1] := - d25 - nlp;
  disp1 level := space[nl base - dump - 1];
  Block := internal block depth + 1;
  block cell pointer := dump1; local for count := dump2;
  max for count := dump3; local count := dump4;
  label count := dump5;
  internal block depth := dump6;
  string occurrence := dump7; proc level := dump8
end Block;

procedure Compound tail;
begin
  Statement; if last symbol = semicolon
  then begin
    next symbol; Compound tail end
end Compound tail;
procedure Declaration list;
begin  Integer n;  count;
next0: if type declarator last symbol
        then begin count:= 0;
            next1: count:= count + 1;
            n:= Identifier;
            if n < last nl then ERRORMESSAGE (122);
            if last symbol = comma
            then begin next symbol; goto next1; end;
            if type = 0 ∨ type = 3 then count:= 2 * count;
            if own type then global count:= global count + count
            else local count:= local count + count;
        if type = 3 then string occurrence:= d6
        end
    else
    if arr declarator last symbol
        then begin count:= array pointer:= 0;
            next2: count:= count + 1;
            next symbol;  n:= Identifier;
            if n < last nl then ERRORMESSAGE (123);
            space(nl base = nl); array pointer:= nl;
            array pointer:= nl;
            if last symbol = comma then goto next2;
            dimension:= 0;
            if last symbol = sub
            then begin sub-count:= 1;
                next symbol;
                if letter last symbol
                then skip identifier
                else if digit last symbol
                then begin unsigned number;
                    Store numerical constant
                end;
            if last symbol = quote then skip string;
            if last symbol = colon
            then begin dimension:= dimension + 1;
                goto next3
            end;
            if last symbol = sub
            then begin subcount:= subcount + 1;
                goto next3
            end;
            if last symbol = bus then goto next3;
            if subcount > 1
            then begin subcount:= subcount - 1;
                goto next3
            end;
            next symbol;
        if dimension = 0 then ERRORMESSAGE (124)
        else dimension:= dimension + 1
        end
    else ERRORMESSAGE (125);
next4: n:= space[nl base - array pointer];
space[nl base - array pointer]:= dimension;
array pointer:= n;
if n ≠ 0 then goto next5;
if own type
then global count:=
global count + (3 × dimension + 3) × count
else local count:= local count + count;
if last symbol = comm
then begin count:= 0; goto next2 end;
if type = 3 then string occurrences:= d6
end
else
if last symbol = switch
then begin next symbol; n:= Identifier;
if n < last nlp then ERROMESSAGE (126);
space[nl base - nlp]:= 0; nlp:= nlp + 1;
next5: next symbol;
if letter last symbol
then skip identifier
else if digit last symbol
then begin unsigned number;
    Store numerical constant
end;
if last symbol = quote then skip string;
if last symbol ≠ semicolon then goto next5
end
else begin next symbol; n:= Identifier;
if n < last nlp then ERROMESSAGE (127);
    nlp:= nlp + 1;
if type < 4
then begin space[nl base - nlp]:= type × d19;
    nlp:= nlp + 1
end;
Block (n)
end;
if last symbol = semicolon then next symbol
else ERROMESSAGE (128);
if declarator last symbol then goto next0
end Declaration list;
procedure Statement;
begin
  integer n, lfc;
  ifc := Local for count;
next0: character := 6 x d19;
next1: if letter last symbol
    then begin read identifier;
      if last symbol = colon
        then begin n := Process identifier;
          Label declaration (n);
          goto next1
        end
    end
  else
    if digit last symbol
      then begin unsigned number;
        if last symbol = colon
          then begin Int lab declaration; goto next1 end
        else Store numerical constant
      end
  else
    if last symbol = for
      then begin local for count := local for count + 1;
        if local for count > max for count
          then max for count := local for count
      end
  else
    if last symbol = begin
      then begin Begin statement; next symbol; goto next1 end
    else
      if last symbol = quote then skip string;
      if last symbol + semicolon + last symbol = end
        then begin next symbol; goto next1 end;
      local for count := lfc
end Statement;

procedure Label declaration (n); integer n;
begin
  if n < last nlp then ERRORMESSAGE (129); 
  if label count = 0
    then space [nl base - block cell pointer - 3] := d13 x (nlp - 1);
    label count := label count + 2;
    space [nl base - nlp] := d16; nlp := nlp + 1;
end Label declaration;

procedure Int lab declaration;
begin
  if real number
    then begin ERRORMESSAGE (130); next symbol end
  else begin Int labels := true;
    in name list; nlp := nlp + 3;
    Label declaration (integer label)
end Int lab declaration;
procedure Begin statement;
begin integer n;
    next symbol;
    if declarator last symbol
    then begin n:= Block (0);
            if n > internal block depth
            then internal block depth:= n
    end
    else Compound tail
end Begin statement;

procedure Store numerical constant;
begin if | small
    then begin space[prog base + instruct counter]:= value of constant;
            space[prog base + instruct counter + 1]:= decimal exponent;
            instruct counter:= instruct counter + 2
    end
end Store numerical constant;

integer procedure Process identifier;
begin last nlp:= nlp; nlp:= nlp + word count + 2;
    space[nl base - nlp + 1]:= character;
    Process identifier:= look up
end Process identifier;

integer procedure Identifier;
begin read identifier;
    Identifier:= Process identifier
end Identifier;
main program of prescan0:
  runumber := 100; init;
  local for count := max for count := local count := label count :=
  global count := internal block depth := string occurrence :=
  disp level := prc level := 0;
  old block cell pointer := block cell pointer := nlp;
  int labels := false;
  space[text base] :=
  space[nl base - block cell pointer] :=
  space[nl base - block cell pointer - 1] :=
  space[nl base - block cell pointer - 3] := 0;
  nlp := block cell pointer + 6;
  space[nl base - block cell pointer - 4] := - d25 - nlp;
  next symbol;
  Program;
  space[nl base - block cell pointer - 1] :=
    (global count + max for count + label count) × d13 +
    (internal block depth + 1) × (d13 + d6);
  space[nl base - block cell pointer - 2] := nlp × d13;
  for n := 0 step 1 until max for count - 1 do
    space[nl base - nlp - n] := d19;
    nlp := nlp + max for count;
  space[nl base - block cell pointer - 5] := - d25 - nlp;
end prescan0;
procedure prescan;
begin

procedure Arithexp;
begin
  if last symbol = if then Ifclause (arithexp)
  else Simple arithexp
end Arithexp;

procedure Simple arithexp;
begûn
  Integer n;
  if last symbol = plus \lor last symbol = minus
  then
    next0: next symbol;
    if last symbol = open
    then begin next symbol; Arithexp;
      if last symbol = close then next symbol
    end
    else
      if digit last symbol then unsigned number
      else
        if letter last symbol
        then begin n:= Identifier; Arithmetic (n);
          Subscripted variable(n); Function designator(n)
        end
        else
          if last symbol = if then Arithexp;
          if arithoperator last symbol then goto next0
end Simple arithexp;

procedure Subscripted variable (n); integer n;
begin
  if last symbol = sub then begin Subvar (n);
    dimension:= Subcrlst;
    List length (n)
  end
end Subscripted variable;

integer procedure Subcrlst;
begin
  next symbol; Arithexp;
  if last symbol = comma then Subcrlst:= Subcrlst + 1
  else begin if last symbol = bus
            then next symbol;
            Subcrlst:= 1
  end
end Subcrlst;

procedure Boolexp;
begin
  if last symbol = if then Ifclause (Boolexp)
  else Simple boolean
end Boolexp;
procedure Simple boolean;
begin  Integer n, type;
   if last symbol = non then next symbol;
   if last symbol = open then begin
      begin
         next symbol; Exp (type);
         if last symbol = close then next symbol
      end
   else
      if letter last symbol then begin
         n:= Identifier;
         Subscripted variable (n);
         Function designator (n);
         if arithoperator last symbol V
         relator operator last symbol
         then Arithmetic (n)
         else Boolean (n)
      end
   else
      if digit last symbol V last symbol = plus V last symbol = minus
      then Simple arithmetic
      else
         if last symbol = true V last symbol = false then next symbol;
      Rest of exp (type)
end Simple boolean;

procedure Stringexp;
begin  if last symbol = if then Ifclause (Stringexp)
   else Simple stringexp
end Stringexp;

procedure Simple stringexp;
begin  Integer n;
   if last symbol = open
   then begin
      next symbol; Stringexp;
      if last symbol = close then next symbol
   end
   else
      if letter last symbol
      then begin
         n:= Identifier; String (n);
         Subscripted variable (n);
         Function designator (n)
      end
   else
      if last symbol = quote
      then begin
         quote counter:= 1;
         noto; next symbol;
         if last symbol = unquote
         then begin quote counter:= 0;
         next symbol
         end
         else goto nexto
      end
end Simple stringexp;
procedure Desigexp;
begin  if last symbol = if then Ifclause (Desigexp)
        else Simple desigexp
end Desigexp;

procedure Simple desigexp;
begin  integer n;
    if last symbol = open
        then begin next symbol; Desigexp;
        if last symbol = close then next symbol
    end
    else
    if letter last symbol
        then begin n := Identifier; Designational (n);
        Subscripted variable (n)
    end
    else
    if digit last symbol
        then begin unsigned number;
        if in name list
        then Designational (integer label)
        end
end Simple desigexp;

procedure Exp (type); integer type;
begin  if last symbol = if
    then begin next symbol; Boolexp;
    if last symbol = else
    then begin next symbol; Type exp (type)
    end
    else Simpleexp (type)
end Exp;

procedure Type exp (type); integer type;
begin  if type = ar \type = re \type = in
    then Arithexp
    else if type = bo
    then Boolexp
    else if type = st
    then Stringexp
    else if type = des
    then Designexp
    else Exp (type)
end Type exp;
procedure Simplex (type); integer type;
begin
integer n;
type := un;
if last symbol = open
then begin next symbol; Exp (type);
if last symbol = close then next symbol
end	only
if letter last symbol
then begin n := Identifier; Subscripted variable (n);
Function designator (n);
if arithoperator last symbol \or
relatoperator last symbol
then Arithmetic (n)
else if booloperator last symbol
then Boolean (n)
else begin if nonformal label (n)
then Designational (n);
type := type bits (n)
end
end	only
if digit last symbol
then begin unsigned number;
if in name list
then Designational (integer label)
else type := ar
end	only
if last symbol = plus \or last symbol = minus
then begin Simple arithexp; type := ar end
else
if last symbol = non \or last symbol = true \or last symbol = false
then begin Simple boolean; type := bo end
else
if last symbol = quote
then begin Simple stringexp; type := st; goto end;
Rest of exp (type);
end;
end Simplex;

procedure Rest of exp (type); integer type;
begin
if arithoperator last symbol
then begin next symbol; Simple arithexp;
type := ar
end;
if relatoperator last symbol
then begin next symbol; Simple arithexp;
type := bo
end;
if booloperator last symbol
then begin next symbol; Simple boolean;
type := bo
end
end Rest of exp;
procedure Assignstat (n); integer n;
begin Subscripted variable (n);
    if last symbol = colon:= equal then Right hand side (n)
end Assignstat;

procedure Right hand side (n); integer n;
begin Integer m, type, type n;
    Assigned to (n); type n := type bits (n);
    next symbol;
    if letter last symbol
    then begin m := Identifier; Subscripted variable (m);
        if last symbol = colon:= equal
        then
        begin Insert (type n, m);
            Right hand side (m); type:= type bits (m)
        end
    else
    begin Function designator (m);
        if arithoperator last symbol ∨
            relationaloperator last symbol
        then Arithmetic (m)
        else if booloperator last symbol
            then Boolean (m)
        else begin Arbopt (m);
            type:= if type n = re ∨ type n = in
                then ar
                else type n;
            Insert (type, m);
            type:= type bits (m);
            if type = re ∨ type = in
                then type:= ar
            end
        end
        Rest of expr (type)
    end
    else begin m := type n; Type expr (type n);
        if m ≠ nonede then type n := m;
        type:= if type n = re ∨ type n = in then ar
            else type n
    end
    Insert (type, n)
end Right hand side;
procedure Insert (type, n); integer type, n;
begin  if type = re
    then Real (n)
    else if type = in
        then Integer (n)
        else if type = bo
            then Boolean (n)
        else if type = ar then Arithmetic (n)
end Insert;

procedure Function designator (n); integer n;
begin  if last symbol = open then begin Function (n);
        dimension:= Parlist;
        List length (n)
    end
end Function designator;

integer procedure Parlist;
begin  next symbol; Actual parameter;
    if last symbol = comma
        then Parlist:= Parlist + 1
    else begin if last symbol = close then next symbol;
    Parlist:= 1
    end
end Parlist;

procedure Actual parameter;
begin  Integer type;
    Exp (type)
end Actual parameter;

procedure Procstat (n); integer n;
begin  Proc (n);
    dimension:= if last symbol = open then Parlist else 0;
    List length (n)
end Procstat;
procedure Statement;
begin  integer n;
  if letter last symbol
  then begin n := Identifier;
    if last symbol = colon
      then labelled statement (n)
    else begin if last symbol = sub ∨
      last symbol = colon equal
      then Assignstat (n)
    else Procstat (n)
  end
else
  if digit last symbol
  then begin unsigned number;
    if last symbol = colon
      then Intlabelled statement
  end
else
  if last symbol = goto then Gostat
else
  if last symbol = begin
  then begin next symbol;
    if declarator last symbol then Block
    else Compound tail;
    next symbol
  end
else
  if last symbol = if then Ifclause (Statement)
else
  if last symbol = for then Forstat
end Statement;

procedure Gostat;
begin  integer n;
  next symbol;
  if letter last symbol
  then begin n := Identifier;
    if ' local label (n)
      then begin Designational (n);
        Subscripted variable (n)
      end
else
  Designexp
end Gostat;

procedure Compound tail;
begin
  if last symbol ≠ colon ∧ last symbol ≠ end
  then skip rest of statement (Statement);
  if last symbol = semicolon
  then begin next symbol; Compound tail end
end Compound tail;
procedure Ifclause (pr); procedure pr;
begin next symbol; Boolean;
  if last symbol = then then next symbol; pr;
  if last symbol = else then begin next symbol; pr end
end Ifclause;

procedure Forstat;
begin Integer n;
  next symbol;
  if letter last symbol
  then begin n:= Identifier; Arithmetic (n);
    Subscripted variable (n);
    if last symbol = colon equal
    then
      next0; next symbol; Arithexpr;
      if last symbol = step
      then begin next symbol; Arithexpr;
        if last symbol = until
        then begin next symbol;
        Arithexpr
      end
      else
      if last symbol = while
      then begin next symbol; Boolean end;
      if last symbol = comma then goto next0;
      if last symbol = do then next symbol;
      forcount:= forcount + 1;
      Statement;
      forcount:= forcount - 1
    end
end Forstat;

procedure Switch declaration;
begin Integer n;
  next symbol;
  if letter last symbol
  then begin n:= Identifier;
    if last symbol = colon equal
    then begin dimension:= Switchlist;
      Switch length (n)
    end
  end
end Switch declaration;

integer procedure Switchlist;
begin next symbol; Desigexpr;
  if last symbol = comma then Switchlist:= Switchlist + 1
  else Switchlist:= 1
end Switchlist;
procedure Array declaration;
begin
  integer i, n, count;
  next symbol; n:= Identifier; count:= 1;
next0: if last symbol = comma then begin next symbol;
        if letter last symbol
        then skip identifier;
        count:= count + 1; goto next0
        end;
        if last symbol = sub
        then begin is array declaration:= true;
            dimension:= Bound pair list;
            is array declaration:= false
            end
        end else dimension:= 0;
Check dimension (n);
if own type then for i:= 1 step 1 until count do
  begin
    Address (n, instruct counter);
    instruct counter:= instruct counter + 3*dimension + 6;
    n:= next identifier (n)
  end;
if last symbol = comma then Array declaration
end Array declaration;

integer procedure Bound pair list;
begin
  next symbol; Arithexp;
  if last symbol = colon then begin next symbol; Arithexp end;
  if last symbol = comma
  then Bound pair list:= Bound pair list + 1
  else begin if last symbol = bus then next symbol;
          Bound pair list:= 1
  end
end Bound pair list;
procedure Procedure declaration;
    begin
        integer n, m;
        next symbol; m = Identifier; entrance block;
        if last symbol = open
            then begin in formal list:= true;
                next symbol; m = Identifier;
                if space[1] base - m] = 95 x d19
                    then begin ERRORMESSAGE (201);
                        space[1] base - m] := 127 x d19
                    end;
                if last symbol = comma then goto nextO;
                if last symbol = close then next symbol;
                in formal list:= false
            end;
        if last symbol = semicolon then next symbol;
        skip value list; skip specification list;
        if in code (n)
            then Scan code (n)
        else begin if space[1] base - n] ; d19 = 19 ∧
                then use of counter stack
                space[1] base - block cell pointer - 2] :=
                space[1] base - block cell pointer - 2] + 64;
                if last symbol = begin
                    then begin next symbol;
                        if declarator last symbol
                            then Declaration list;
                        Compound tail; next symbol
                    end
                else Statement;
                Addressing of block identifiers (n)
            end
        end Procedure declaration;

procedure Block;
    begin
        entrance block; Declaration list; Compound tail;
        Addressing of block identifiers (0)
    end Block;

procedure Declaration list;
    begin
        if typeddeclarator last symbol then skip type declaration
        else
            if arr declarator last symbol then Array declaration
            else
                if last symbol = switch then Switch declaration
                else
                    if last symbol = semicolon then next symbol;
                    if declarator last symbol then Declaration list
                end
            end Declaration list;
procedure Program;
begin
  Integer n;
  if letter last symbol
    then begin n:= Identifier;
        if last symbol = colon
          then Label declaration (n);
        Program
    end
  else
    if digit last symbol
      then begin unsigned number;
        if in name list
          then Label declaration (integer label);
        Program
      end
  else
    if last symbol = begin
      then begin next symbol;
        if declarator last symbol
          then Block
        else Compound tail
      end
    else begin next symbol; Program end
  Program;
end

procedure labelled statement (n); Integer n;
begin
  if nonformal label (n) then Label declaration (n);
Statement
end

procedure Intlabelled statement;
begin
  if in name list then Label declaration (integer label);
Statement
end

procedure Label declaration (n); Integer n;
begin
  if proc level = 0
    then begin Designational (n); Address (n, instruct counter);
        space[nl base - n - 1]:= space[nl base - n - 1] + instruct counter +
          d20 \times \text{forcount};
        space[prog base + instruct counter]:= 0;
        space[prog base + instruct counter + 1]:= d16 \times \text{display level} + d40;
        instruct counter:= instruct counter + 2
    end
  else
    space[nl base - n - 1]:= space[nl base - n - 1] +
          d20 \times \text{forcount};
next symbol
end Label declaration;
procedure Addressing of block identifiers (n); integer n;
begin
  integer counter, f, code, code1;
  if n = 0 then
    space[nl base - block cell pointer - 1] :=
    space[nl base - block cell pointer - 1] + d13;
  if proc level > 0 then
    begin
      counter := d9 x display level + d9;
      if n = 0 then
        counter := counter + 1 + d18
      else
        begin
          counter := counter + display level + top of display;
          f := block cell pointer + 5;
          next0: f := next identifier (f);
          if f > block cell pointer then
            begin
              Address (f, counter);
              code1 := space[nl base - f] \ d18;
              code := code1 \ 2;
              counter := counter +
              (if code = 64 \ code = 67 \ code = 70
                then 2
                else if code < 96
                  then 1
                  else if code1 = 2 \ code
                    then 2 else 4));
              goto next0;
            end;
          counter := counter + d18;
          code := space[nl base - n] \ d19;
          if code \ 24 then
            begin
              f := if wanted then 3 else
              (if code = 76 \ code = 19 then 2 else 1;
                Address (n + 2, counter);
                counter := counter + f;
                space[nl base - block cell pointer - 1] :=
                space[nl base - block cell pointer - 1] +
                d13 \ f
              )
            end;
end;
f := status;
next1: if space[nl base - f] > 0
  then begin Address (f, counter); counter := counter + 1;
    f := f + 1;
  goto next1
end;

f := block cell pointer + k;
next2: f := next identifier (f); code := space[nl base - f] : d19;
  if f > block cell pointer ∧ f < status ∧ code < 64
  then begin if code > 24
    then begin if code < 36
      then begin Address (f, instruct counter);
        instruct counter :=
        instruct counter +
        (if code = 32 ∨ code = 35
        then 2 else 1)
      end
    end
  else if code < 14
    then begin if code ≠ 6 ∨
      (code = 6 ∧
      bit string (d19, d18, space[nl base - f - 1]) = 0)
      then begin Address (f, counter);
        counter :=
        counter +
        (if code = 0 ∨ code = 3 ∨ code = 6
        then 2 else 1)
      end
    end
  goto next2
end;
if counter > d18 + d9 × (display level + 1)
then ERRORMESSAGE (202);
exit block
end
else Static addressing
end Addressing of block identifiers;
procedure Static addressing;
begin  Integer f, code;
    if status;  
next0:  if space[nl base - f] > 0
    then begin Address (f, instruct counter);
           instruct counter:= instruct counter + 1; f:= f + 1;
        goto next0
        end;
    f:= block cell pointer + 4;
next1:  f:= next identifier (f); code:= space[nl base - f] ; d19;
    if f > block cell pointer ∧ f < status
then begin if code > 24 ∧ code < 36 ∨ code < 14 ∧ code ≥ 6
    then begin Address (f, instruct counter);
       instruct counter:=
       instruct counter +
       (if code = 0 ∨ code = 3 ∨
       code = 32 ∨ code = 35 then 2
               else 1)
        end;
        goto next1
        end;
    exit Block
end Static addressing;
procedure Add type (n, t); integer n, t;
begin
    Integer code, new code, type;
    new code := code := space[nl base - n] * d19;
    if code > 95
    then begin if code = 127
    then new code := 95 + t
    else if code = 120 \ t < 6
    then new code := 112 + t
    else begin
        type := code := code := 8 * 8;
        if type = un \ (type = nondes \ t < 5) \ (type = ar \ t < 2)
        then new code := code := type + t
    end;
    space[nl base - n] :=
    space[nl base - n] := (code := new code) * d19
end
end Add type;

procedure Real (n); integer n; begin Add type (n, re) end Real;

procedure Integer (n); integer n; begin Add type (n, in) end Integer;

procedure Boolean (n); integer n; begin Add type (n, bo) end Boolean;

procedure String (n); integer n; begin Add type (n, st) end String;

procedure Arithmetic (n); integer n;
begin Add type (n, ar) end Arithmetic;

procedure Arbost (n); integer n;
begin Add type (n, nondes) end Arbost;
procedure Designational (n); integer n;
begin integer p;
   if nonformal label (n)
      then
         begin if bit string (d19, d18, space[nl base - n - 1]) = 1
              then
                 begin space[nl base - n - 1] :=
                    abs (space[nl base - n - 1] - d18);
                     p := corresponding block cell pointer (n);
                     if bit string (d16, d13, space[nl base - p - 2]) > 0
                        then begin space[nl base - p - 3] :=
                             space[nl base - p - 3] + 1;
                             space[nl base - p - 1] :=
                             space[nl base - p - 1] + d14
                        end
                     else Add type (n, des)
                 end Designational;
         end
   else Add type (n, des)
end Designational;

procedure Assigned to (n); integer n;
begin integer code;
   code := space[nl base - n] + d19;
   if code > 255
      then
         begin if code = 127 then code := 101;
              if code < 102 then space[nl base - n] := code x d19 + d18
         end
else Add type (n, nondes)
end Assigned to;

procedure Subscrvar (n); integer n;
begin integer code, new code;
   code := space[nl base - n] + d19;
   if code > 255
      then
         begin new code := if code = 127
                           then 111
                           else if code < 104
                                  then code + 8
                                  else code;
            space[nl base - n] := space[nl base - n] +
            (new code - code) x d19
         end
end Subscrvar;
procedure Proc (n); integer n;
begin
  integer code, new code;
  code := space[nl base - n] + d19;
  if code > 95
    then begin new code := if code = 127
      then 120
      else if code < 102
        then code + 16
        else codes;
    space[nl base - n] := space[nl base - n] +
      (new code - code) × d19
  end Proc;
end;

procedure Function (n); integer n;
begin
  Arobst (n); Proc (n) end Function;

procedure List length (n); integer n;
begin
  integer word;
  if space[nl base - n] + d19 > 95
    then begin word := space[nl base - n - 1];
      if bit string (d18, d9, word) = 0
        then space[nl base - n - 1] := word + dimension + 1
    end List length;

procedure Switch length (n); integer n;
begin
  space[nl base - n - 1] := dimension + 1 end Switch length;

procedure Address (n, m); integer n, m;
begin
  integer word;
  word := space[nl base - n] + d18;
  space[nl base - n] := word × d10 + n
end Address;

procedure Check dimension (n); integer n;
begin
  if space[nl base - n - 1] ≠ dimension + 1
    then begin ERRORMESSAGE (203);
      space[nl base - n - 1] := dimension + 1
    end
end Check dimension;
integer procedure Identifier;
  begin
    integer n;
    last nlp := nlp; read identifier; Identifier := n: look up;
    if n > nlp then Ask librarian;
    if n > nlp then begin ERRORMESSAGE (sof);
      nlp := nlp + word count + 3;
      space[nl base - nlp + 1] := 0
    end Identifier;
  end

procedure Scan code (n); integer n;
begin Block cell pointer := space[nl base - block cell pointer] ; df;  
  next0: next symbol; if last symbol = minus then next symbol;  
    if letter last symbol then Identifier else unsigned integer (0);
    if last symbol = comma then goto next0;  
    if last symbol = unquote then next symbol
  end Scan code;

procedure Ask librarian;
  begin comment if the current identifier occurs in the library  
    then this procedure will add a new name cell to the name list and increase nlp;  
  end Ask librarian;
main program of prescan:
  if 1 text in memory
      then begin NEWPAGE;
          PRINTTEXT ("input tape for prescan")
      end;
  runnumber:= 200; init;
  block cell pointer:= next block cell pointer:= 0;
  dp0:= instruct counter;
  instruct counter:= instruct counter + top of display;
  space[nl base - nlp]:= -1;
  next symbol; entrance block;
  Program; Static addressing;
  output
  end prescan;
procedure translate;
begin

integer last ln0, ln1, last ln0, macro, parameter, state,
stack0, stack1, b, ret level, max depth,
ret max depth, max depth isr, max display length,
max proc level, ecount, controlled variable, increment,
10, 11, 12, 13, 14, 15, number of switch elements,
switch identifier, switch list count, word,
address of constant, sum of maxima;

Boolean in switch declaration, in code body, if statement forbidden,
complicated, complex step element;

procedure Arithexp;
begin Integer future1, future2;
  if last symbol = if
    then begin future1 := future2 := 0;
      next symbol; Boolean; Macro2 (CJU, future1);
      if last symbol ≠ then then ERRORMESSAGE (300)
        else next symbol;
      Simple arithexp;
      if last symbol = else
        then begin Macro2 (JU, future2);
          Substitute (future1);
          next symbol; Arithexp;
          Substitute (future2)
        end
      else ERRORMESSAGE (301)
    end
  else Simple arithexp
end Arithexp;

procedure Simple arithexp;
begin if last symbol = minus then begin next symbol; Term;
    Macro (NEG)
    end
  else begin if last symbol = plus
    then next symbol;
    Term
    end;

  Next term
end Simple arithexp;
procedure Next term;
begin
  if last symbol = plus then begin
    Macro (STACK);
    next symbol; Term;
    Macro (ADD); Next term
  end
  else
  if last symbol = minus then begin
    Macro (STACK);
    next symbol; Term;
    Macro (SUB); Next term
  end
end Next term;

procedure Term; begin Factor; Next factor end Term;

procedure Next factor;
begin
  if last symbol = mul then begin
    Macro (STACK);
    next symbol; Factor;
    Macro (MUL); Next factor
  end
  else
  if last symbol = div then begin
    Macro (STACK);
    next symbol; Factor;
    Macro (DIV); Next factor
  end
  else
  if last symbol = idi then begin
    Macro (STACK);
    next symbol; Factor;
    Macro (IDI); Next factor
  end
end Next factor;

procedure Factor; begin Primary; Next primary end Factor;

procedure Next primary;
begin
  if last symbol = tt then begin
    Macro (STACK);
    next symbol; Primary;
    Macro (TT); Next primary
  end
end Next primary;
procedure Primary;
    begin
        integer n;
        if last symbol = open then begin
            next symbol; Arithexp;
            if last symbol = close then next symbol
                else ERRORMESSAGE (302)
        end
        else
            if digit last symbol then begin
                Unsigned number;
                Arithconstant
        end
        else if letter last symbol then begin
            n:= Identifier;
            Subscripted variable (n);
            Function designator (n);
            Arithname (n)
        end
        else ERRORMESSAGE (303);
        if last symbol = if ∨ last symbol = plus ∨
            last symbol = minus
            then Arithexp
        end
    end Primary;

procedure Arithname (n); integer n;
    begin
        if Nonarithmetic (n) then ERRORMESSAGE (304);
        complicated:= Formal (n) ∨ Function (n);
        if Simple (n)
            then begin
                if Formal (n) then Macro2 (DXS, n)
                if Integer (n) then Macro2 (TIW, n)
                else Macro2 (TRV, n)
            end
    end Arithname;

procedure Subscripted variable (n); integer n;
    begin
        if Subscrvvar (n) then begin
            Address description (n);
            if last symbol = colon-equal
                then begin
                    Macro (STACK);
                    Macro (SPAA)
                end
                else Evaluation of (n)
        end
    end Subscripted variable;
procedure Address description (n); integer n;
begin
  if last symbol = sub
  then begin
      next symbol; dimension:= Subscript list;
      Check dimension (n);
      if Formal (n) then Macro2 (DOS, n) else
        if Designational (n) then Macro2 (TSWE, n) else
          Macro2 (TAK, n)
  end
  else ERRORMESSAGE (305)
end Address description;

procedure Evaluation of (n); integer n;
begin
  if Designational (n)
    then begin
        if Formal (n) then Macro (TFL)
        else Macro (TSL)
    end
  else
    if Boolean (n) then Macro (TSB) else
    if String (n) then Macro (TSP) else
    if Formal (n) then Macro (TFSU) else
    if Integer (n) then Macro (TSI) else Macro (TSR)
end Evaluation of;

integer procedure Subscript list;
begin
  Arithexp;
  if last symbol = coma
  then begin
      Macro (STACK); next symbol;
      Subscript list:= Subscript list + 1
  end
  else begin
      if last symbol = bus
      then next symbol
      else ERRORMESSAGE (306);
      Subscript list:= 1
  end
end Subscript list;
procedure Boolean;
begin  Integer future1, future2;
  if last symbol = if then begin future1 := future2 := 0;
    next symbol; Boolean; Macro2 (COND, future1);
    if last symbol ≠ then then ERREMESSAGE (307)
      else next symbol;
      Simple boolean;
    if last symbol = else then begin Macro2 (JU, future2);
      Substitute (future1);
      next symbol; Boolean;
      Substitute (future2)
    end else ERREMESSAGE (308)
  end
else Simple boolean
end Boolean;

procedure Simple boolean;
begin  Implication; Next implication end Simple boolean;

procedure Next implication;
begin if last symbol = qv then begin macro (STAB);
    next symbol; Implication;
    macro (QV); Next implication
end Next implication;

procedure Implication; begin Boolean; Next boolean end Implication;

procedure Next boolean;
begin if last symbol = imp then begin macro (STAB);
    next symbol; Boolean;
    macro (IMP); Next boolean
end Next boolean;

procedure Boolean; begin Boolean; Next boolean end Boolean;

procedure Next boolean;
begin if last symbol = or then begin macro (STAB);
    next symbol; Boolean;
    macro (OR); Next boolean
end Next boolean;

procedure Boolean; begin Boolean; Next boolean end Boolean;
procedure Next boolsec;
begin  If last symbol = and then begin Macro (STAB);
              next symbol; Boolsec;
              Macro (AND); Next boolsec
end   Next boolsec;
end

procedure Boolsec;
begin  If last symbol = non then begin next symbol; Boolprim;
              Macro (NOT)
end   else Boolprim
end

procedure Boolprim;
begin  Integer type, n;
       If last symbol = open
              then begin next symbol; Arboolexp (type);
                              if last symbol = close then
                                              next symbol
                                              else ERROREM (309);
                                              if type = ar then Rest of relation else
                                              if type = arbo then begin if arthopera-"tor last symbol
                                                                  then Rest of relation
                                                                  else Relation
                                                                  end
              end
else  If letter last symbol then begin n:= Identifier;
              Subscripted variable (n);
              Boolprimrest (n)
end
else  If digit last symbol V last symbol = plus V last symbol = minus
              then begin Simple arithexp; Rest of relation end
else  If last symbol = true V last symbol = false
              then begin Macro2 (TBC, last symbol); next symbol end
else  ERROREM (310)
end Boolprim;
Boolean procedure Relation;
begin
  integer relmacro;
  if Relatoperator last symbol then begin
    relmacro:= Relatmacro; Macro (STACK);
    next symbol; Simple arithexp;
    Macro (relmacro); Relation:= true
  end
  else Relation:= false
end Relation;

decision Rest of relation;
begin
  Rest of arithexp;
  if 7 Relation then ERROMESSAGE (311)
end Rest of relation;

decision Booolprimrest (n) ; integer n;
begin
  Function designator (n);
  if Arithmetic (n) \ arithoperator last symbol
  \ relatoperator last symbol then begin
    Arithname (n); Rest of relation end
  else Boolean name (n)
end Boolean primrest;

decision Boolean name (n) ; integer n;
begin
  If Nonboolean (n) then ERROMESSAGE (312);
  if Simple (n) then begin if Formax (n) then
    Macro2 (DOS, n) else
  else
    Macro2 (TBV, n)
end Boolean name;

decision Arboolexp (type) ; integer type;
begin
  integer future1, future2;
  if last symbol = if then begin future1:= future2:= 0;
  next symbol; Boolean exp; Macro2 (CJU, future1);
  if last symbol \ then then ERROMESSAGE (313)
  else next symbol;
  Simple arboolexp (type);
  if last symbol = then
    begin
      Macro2 (SU, future2); Substitute (future1);
      next symbol;
      if type = bo then Boolean exp else
      if type = ar then Arithexp
    else Arboolexp (type);
    Substitute (future2)
  end
  else ERROMESSAGE (314)
end Arboolexp;

procedure Simple arboolexp (type); integer type;
begin Integer n;
   if last symbol = open
      then begin next symbol; Arboolexp (type);
         if last symbol = close then next symbol
            else ERRORMESSAGE (315);
            if type = bo V
               type = arbo A booloperator last symbol
               then begin Rest of boolexp; type:= bo end
            else if type = ar V
               arithoperator last symbol V
               relatoperator last symbol
               then Rest of arboolexp (type)
      end
   else if letter last symbol
      then begin n:= Identifier; Subscripted variable (n);
         Arboolrest (type, n)
      end
   else if digit last symbol V last symbol = plus V last symbol = minus
      then begin Simple arithexp; Rest of arboolexp (type) end
   else if last symbol = non V last symbol = :true V last symbol = false
      then begin Simple boolean; type:= bo end
   else begin ERRORMESSAGE (316); type:= arbo end
end Simple arboolexp;

procedure Rest of arithexp;
begin Next primary; Next factor; Next term Rest of arithexp;

procedure Rest of boolexp;
begin Next boolsec; Next boolfac; Next boolterm; Next implication
Rest of boolexp;

procedure Rest of arboolexp (type); integer type;
begin Rest of arithexp;
   if Relation
      then begin Rest of boolexp; type:= bo end
   else type:= ar
end Rest of arboolexp;
procedure Arboolrest (type, nj; integer type, nj;
begin
    Function designate (nj);
    if Boolean (nj) V bool operator last symbol
        then begin
            Boolean (nj); Rest of bool exp; type:= bo end
    else
        if Arithmetic (nj) V arith operator last symbol
            then begin
                Arithmetic (nj); Rest of ar bool exp (type) end
        else begin
            if String (nj) V Designational (nj)
                then ERRORESSAGE (317);
                  Macro2 (DOS, nj); type:= arbo end
end Arboolrest;

procedure Stringexp;
begin
    integer future1, future2;
    if last symbol = if
        then begin
            future1:= future2:= 0;
            next symbol; Bool exp; Macro2 (CJU, future1);
            if last symbol # then then ERRORESSAGE (319)
            else next symbol;
            Simple string exp;
            if last symbol = else
                then begin
                    Macro2 (JU, future2);
                    Substitute (future1);
                    next symbol; String exp;
                    Substitute (future2)
                end
            else ERRORESSAGE (319)
            end
        else Simple string exp
end Stringexp;

procedure Simple stringexp;
begin
    integer future, nj;
    if last symbol = open
        then begin
            next symbol; String exp;
            if last symbol = close then next symbol
            else ERRORESSAGE (320)
        end
    else
        if letter last symbol
            then begin
                ni:= Identifier; Subscripted variable (nj);
                Stringname (nj)
            end
    else
        if last symbol = quote
            then begin
                Macro (TCSR); future:= 0; Macro2 (JU, future);
                Constant string; Substitute (future)
            end
    else ERRORESSAGE (321)
end Simple string exp;
procedure Stringname (n); integer n;
begin
  if Nonstring (n) then ERFORMMESSAGE (322);
  Function designator (n);
    if Simple (n) then begin if Formal (n) then Macro2 (DOs, n)
      else Macro2 (TSTV, n)
    end;
end Stringname;

procedure Desigexp;
begin
  Integer future1, future2;
  if last symbol = if
    then begin future1 := future2 := 0;
      next symbol; Boolexp; Macro2 (COJU, future1);
      if last symbol # then then ERFORMMESSAGE (323)
    else next symbol;
      Simple designexp;
    if last symbol = else
      then begin Macro2 (JU, future2);
      Substitute (future1);
      next symbol; Designexp;
      Substitute (future2)
    end
    else ERFORMMESSAGE (324)
  end;
  else Simple designexp
end Desigexp;

procedure Simple designexp;
begin
  Integer n;
  if last symbol = open
    then begin next symbol; Desigexp;
      if last symbol = close then then symbol
    else ERFORMMESSAGE (325)
  end;
else
  if letter last symbol then begin n := Identifier;
    Subscripted variable (n);
    Designname (n)
  end;
else
  if digit last symbol then begin Unsigned number;
    if in name list
      then Macro2 (TLV, integer label)
      else ERFORMMESSAGE (326)
  end;
else ERFORMMESSAGE (327)
end Simple designexp;
procedure Designname (n); integer n;
begin
  if Nondesignational (n) then ERRORMESSAGE (328);
  if Simple (n) then begin
    if Formal (n) then Macro2 (DOS, n)
    else Macro2 (TLV, n)
  end
end Designname;

procedure Ardesexp (type); integer type;
begin
  Exp (type);
  if type = bo ∨ type = st then ERRORMESSAGE (329);
  if type = un then type := intlab else
  if type = nondes then type := ar
end Ardesexp;

procedure Nondesexp (type); integer type;
begin
  Exp (type);
  if type = des then ERRORMESSAGE (330);
  if type = un then type := nondes else
  if type = intlab then type := ar
end Nondesexp;

procedure Exp (type); integer type;
begin
  Integer future1, future2;
  if last symbol = if
  then begin
    future1 := future2 := 0;
    next symbol; Booleanx; Macro2 (CUJ, future1);
    if last symbol ≠ then then ERRORMESSAGE (331)
    else next symbol;
    Simplex (type);
    if last symbol = else
    then begin
      Macro2 (JU, future2);
      Substitute (future1); next symbol;
      if type = ar then Arithexp else
      if type = bo then Booleanx else
      if type = st then Stringexp else
      if type = des then Desigexp else
      if type = intlab then Ardesexp (type) else
      if type = nondes then Nondesexp (type) else
      Exp (type);
    end
  end
  else ERRORMESSAGE (332)
end Exp;
else Simplex (type)
procedure Simplex (type); integer type;
begin
  Integer n;
  if last symbol = open then begin
    next symbol; Exp (type);
    if last symbol = close then next symbol
    else ERRORMESSAGE (333);
    if type = bo ∨ (type = nondes ∨ type = un) ∧
      booloperator last symbol then begin Rest of boolexp; type:= bo end
    else
      if type ≠ st ∧ type ≠ des ∧ operator last symbol then
        Rest of arboolexp (type)
  end
  else
  if letter last symbol then begin
    n:= Identifier; Subscripted variable (n);
    Expset (type, n)
  end
  else
  if digit last symbol then begin
    Unsigned number; Arithconstant;
    if in name list ∧ (¬ operator last symbol)
      then begin Macro2 (TLY, integer label);
        type:= intlab
      end
    else
      Rest of arboolexp (type)
  end
  else
  if last symbol = plus ∨ last symbol = minus then
    Simple arboolexp (type)
  else
  if last symbol = non ∨ last symbol = true ∨ last symbol = false then
    Simple boolean; type:= bo.end
  else
  if last symbol = quote then begin Simple stringexp; type:= st end
  else
    begin ERRORMESSAGE (334); type:= un end
end Simplex;
procedure Exprest (type, n); integer type, n;
begin
  If Designational (n) then begin Designame (n); type:= des end
  else
    if String (n) then begin Stringame (n); type:= st end
    else begin Function designator (n);
      if Boolean (n) or boolexpr last symbol
      then begin Booleaname (n); Rest of boolexpr type:= bo end
      else if Arithmetic (n) or arithexpr last symbol
      then begin Arithame (n); Rest of arithexpr (type) end
      else begin if Simple (n) then Macro2 (DOS, n);
        type:= if Unknown (n) then un else nondes end
    end
end Exprest;

procedure Assignstat (n); integer n;
begin
  if Subscripted variable (n); then Distribute on type (n)
  else ERORMESSAGE (335)
end Assignstat;

integer procedure Distribute on type (n); integer n;
begin
  if Integer (n) then begin Intassign (n); Distribute on type:= in end
  else if Real (n) then Realassign (n); Distribute on type:= re end
  else if Boolean (n) then Boolassign (n); Distribute on type:= bo end
  else if String (n) then Stringassign (n); Distribute on type:= st end
  else Distribute on type:= if Arithmetic (n) then Arassign (n)
  else Unassign (n)
end Distribute on type;

procedure Prepare (n); integer n;
begin
  if Function (n) then begin if Formal (n) then ERORMESSAGE (336)
    else if Outside declaration (n) then ERORMESSAGE (337)
    else nr:= Local position (n) end
  else if Simple (n) or Formal (n) then Macro2 (DOS2, n);
next symbol
end Prepare;
Boolean procedure Intassign (n); integer n;
begin
    Integer m;
    Boolean rounded;
    if Noninteger (n) then ENORMMESSAGE (339);
    Prepare (n); rounded := False;
    if letter last symbol
    then begin
        m := Identifier; Subscripted variable (m);
        if last symbol = colonqual
        then rounded := Intassign (m)
        else begin Function designator (m);
        Arithname (m); Rest of arithexp
               end
            end
    else Arithexp;
    if Subvarvar (n)
    then begin
        if Formal (n) then Macro (STFSU)
        else begin
            if Rounded then Macro (STSI)
            else Macro (STSI)
        end
    else if Formal (n) then Macro2 (DOS3, n)
    else if rounded then Macro2 (STI, n)
    elseMacro2 (STI, n);
    Intassign := Formal (n) ▼ rounded
end Intassign;

procedure Reassign (n); integer n;
begin
    Integer m;
    if Nonreal (n) then ENORMMESSAGE (339);
    Prepare (n);
    if letter last symbol
    then begin
        m := Identifier; Subscripted variable (m);
        if last symbol = colonqual
        then Realassign (m)
        else begin Function designator (m);
        Arithname (m); Rest of arithexp
               end
            end
    else Arithexp;
    if Subvarvar (n)
    then begin
        if Formal (n) then Macro (STFSU)
        else Macro (STSR)
    end
    else if Formal (n) then Macro2 (DOS3, n)
    else Macro2 (STR, n)
end Reassign;
procedure Boolassign (n); integer n;
begin
  Integer m;
  if Nonboolean (n) then ERRORMESSAGE (340);
  Prepare (n);
  if letter last symbol
  then begin m:= Identifier; Subscripted variable (m);
    if last symbol = colon equal
    then Boolassign (m)
    else begin Boolprimrest (m); Rest of boolean end
  end
  else Boolexp;
  if Subscrvar (n) then Macro (STSB)
  else if Formal (n) then Macro2 (DOS3, n)
  else Macro2 (STSB, n)
end Boolassign;

procedure Stringassign (n); integer n;
begin
  Integer m;
  if Nonstring (n) then ERRORMESSAGE (341);
  Prepare (n);
  if letter last symbol
  then begin m:= Identifier; Subscripted variable (m);
    if last symbol = colon equal
    then Stringassign (m)
    else Stringname (m)
  end
  else Stringexp;
  if Subscrvar (n) then Macro (STST)
  else if Formal (n) then Macro2 (DOS3, n)
  else Macro2 (STST, n)
end Stringassign;

integer procedure Arassign (n); integer n;
begin
  Integer type, m;
  if Nonarithmetic (n) then ERRORMESSAGE (342);
  Prepare (n); type:= ar;
  if letter last symbol
  then begin m:= Identifier; Subscripted variable (m);
    if last symbol = colon equal
    then begin if Nonarithmetic (m)
      then ERRORMESSAGE (343);
      type:= Distribute on type (m)
    end
    else begin Function designator (m);
      Arithname (m); Rest of arithexp end
  end
  else Arithexp;
  if Subscrvar (n) then Macro (STSFU) else Macro2 (DOS3, n);
  Arassign:= type
end Arassign;
integer procedure Unassign (n); integer n;
begin integer type, m;
if Nontype (n) then ERRORMESAGE (344);
Prepare (n);
if letter last symbol
then begin m:= Identifier; Subscripted variable (m);
   if Nontype (m) then ERRORMESAGE (345);
   if last symbol = colonqual
   then type:= Distribute on type (m)
   else Exprest (type, m)
end
else Nondesexp (type);
if Subscrvar (n)
then begin if type = bo then Macro (STSB)
   else if type = st then Macro (STST)
   else Macro (STFSU)
end
else Macro2 (DS3, n);
Unassign:= type
end Unassign;

procedure Function designator (n); integer n;
begin if Proc (n)
   then begin if Nonfunction (n) then ERRORMESAGE (346);
          Procedure call (n)
   end
end Function designator;

procedure Procstat (n); integer n;
begin if Proc (n)
   then begin Procedure call (n);
       if 7 (In library (n) ∨ Function (n))
       then last lnc:= n;
       if Formal (n) ∨ (Function (n) ∧ String (n))
       then Macro (REJST)
   end
else ERRORMESAGE (347)
end Procstat;
procedure Procedure call (n); integer n;
begin
  integer number of parameters;
  if Operator like (n)
    then Process operator (n)
  else begin number of parameters := List length (n);
    if number of parameters ≠ 0
      then Parameter list (n, number of parameters)
      else if Formal (n)
        then Macro2 (DOS, n)
        else if In library(s) then Macro2 (ISUBJ, n)
        else Macro2 (SUBJ, n)
  end
end Procedure call;

integer procedure Ordinal number (n); integer n;
begin
  Ordinal number := if Formal (n) then 15
  else
    if Subscrvar (n)
      then (if Arithmetic (n)
        then (if Real (n) then 8 else 9)
        else if Boolean (n)
          then 10 else 11)
      else
        if Function (n)
          then (if Arithmetic (n)
            then (if Real (n) then 24 else 25)
            else if Boolean (n) then 26 else 27)
        else
          if Proc (n) then 30
          else
            if Arithmetic (n)
              then (if Real (n) then 0 else 1)
              else if Boolean (n)
                then 2
                else if String (n) then 3 else 14
  end Ordinal number;
procedure Parameter list (n, number of parameters);
    integer n, number of parameters;
    begin integer count, m, f, apd, type, future;
        boolean simple identifier;
        integer array descriptor list[1 : number of parameters];
        count := future := 0; f := n;
        if last symbol = open
            then begin
                next; count := count + 1; next symbol;
                actual parameter (apd, simple identifier, type, future);
                if count < number of parameters
                    then begin descriptor list[count] := apd;
                        if 'l' Formal (n)
                            then begin f := Next formal identifier (f);
                                if simple identifier
                                    then begin if Subscrvar (f)
                                        then begin if Nonsubscrvar (type)
                                            then ERRORMESSAGE (348);
                                                check type (f, type);
                                                check list length (f, type)
                                        end
                                    end
                                else if Proc (f)
                                    then begin if Nonproc (type)
                                        then ERRORMESSAGE (349);
                                                check list length (f, type);
                                                if Function (f)
                                                    then begin if Nofunction (type)
                                                        then ERRORMESSAGE (350);
                                                            check type (f, type)
                                                    end
                                                else if Simple (f)
                                                    then begin if Nonsimple (type)
                                                        then ERRORMESSAGE (351);
                                                            check type (f, type)
                                                    end
                                end
                        end
                    end
            end
        end
    end
else
begin if SubscrVar (r) ∨ Proc (f)
then ERRORMESSAGE (352);
if Assigned to (f) ∧ Nonassignable (apd)
then ERRORMESSAGE (353);
if Arithmetic (r) ∧
(type = bo ∨ type = st ∨ type = des)
then ERRORMESSAGE (354) else
if Boolean (f) ∧
type ≠ bo ∧ type ≠ nondes ∧ type ≠ un
then ERRORMESSAGE (355) else
if String (f) ∧
type ≠ st ∧ type ≠ nondes ∧ type ≠ un
then ERRORMESSAGE (356) else
if Designational (f) ∧
type ≠ des ∧ type ≠ un
then ERRORMESSAGE (357) else
if Ar bouts (f) ∧ type = des
then ERRORMESSAGE (358)
end
end
else ERRORMESSAGE (359);
if Last symbol = comma then goto next;
if Last symbol = close
then begin next symbol;
if count < number of parameters
then ERRORMESSAGE (360)
end
else ERRORMESSAGE (361)
else ERRORMESSAGE (362);
if Future ≠ 0 then Substitute (future);
if Formal (n) then Macro2 (XOS, n) else if In library (n)
then Macro2 (ISUB, n)
else Macro2 (SUBJ, n);
mi = 0;
next apd: if m < count ∧ m < number of parameters
then Begin mi = m + 1; apd = descriptor list [mi];
Macro2 (CODE, apd); goto next apd
end
end Parameter list;
procedure Actual parameter (apd, simple identifier, type, future);
integer apd, type, future; Boolean simple identifier;
begin integer n, begin address;
  begin address:= Order counter + (if future = 0 then 1 else 0);
    simple identifier:= false;
    if letter last symbol
  then
begin n:= Identifier;
  if last symbol = comma \ last symbol = close
  then
begin type:= n; simple identifier:= true;
    if Proc (n) \ 1 Formal (n)
  then
begin if future = 0 then Macro2 (JU, future);
      Macro2 (TPD);
      if In library (n) then Macro2 (IJU1, n)
noe Macro2 (JU1, n); 
  apd:= d20 \ Ordinal number (n) + begin address
end
else if Subscrvar (n) \ Designational (n) \ 
      \ 1 Formal (n)
  then begin if future = 0
      then Macro2 (JU, future);
      Macro2 (TSWS, n);
  apd:= 12 \ d20 + begin address
end
else apd:= d20 \ Ordinal number (n) +
                  Address (n) +
  (if Dynamic (n) then d18 else 0)
end
else
begin Start implicit subroutine (future);
  if Subscrvar (n) then Address description (n);
  if (last symbol = comma \ last symbol = close) \ 
      \ 1 Designational (n))
  then begin if Unknown (n) then Macro (SAS);
    Macro2 (EXTGS, \ 2 \ dimension);
  apd:= d20 \ (if Boolean (n) then 18 else
    if String (n) then 19 else
    if Formal (n) then 32 else
    if Real (n) then 16 else
    + Order Counter;
    type:= if Arithmetic (n) then ar else
    if Boolean (n) then bo else
    if String (n) then st else
    if Arbo1 (n) then nondes else un;
    Macro2 (SUB1, \ begin address);
    if Boolean (n) then Macro (TSBB) else
    if String (n) then Macro (TASS) else
    if Formal (n) then Macro (TASS) else
    if Integer (n) then Macro (TASI) else
    if Integer (n) then Macro (TASS) else
    else Macro (TASS);
    Macro (DECS); Macro2 (SUB1, \ begin address);
    Macro (FAD)
else
  begin
    if Subscrvar (n) then Evaluation of (n);
    Exprst (type, n); Macro (EXITIS);
    apd:= mask[type] + begin address
  end
end

else
  if digit last symbol
  then begin
    Unsigned number;
    if (last symbol = comma V last symbol = close) V
    (\ not in name list)
    then begin
      type:= ar; apd:= Number descriptor end
    else begin
      Start implicit subroutine (future);
      Arithconstant;
      if in name list: \ (\ not operator last symbol)
      then begin
        Macro2 (TLV, integer label);
        type:= intlab end
      else Rest of ar_boolexp (type);
      Macro (EXITIS);
      apd:= mask[type] + begin address
    end
  end
else
  if Last symbol = plus
  then begin
    next symbol;
    if digit last symbol
    then begin
      Unsigned number;
      if last symbol = comma V last symbol = close
      then begin
        type:= ar; apd:= Number descriptor end
      else begin
        Start implicit subroutine (future);
        Arithconstant;
        Rest of ar_boolexp (type);
        Macro (EXITIS);
        apd:= mask[type] + begin address
      end
    end
  else begin
    Start implicit subroutine (future);
    Arboolexp (type);
    Macro (EXITIS); apd:= mask[type] + begin address
  end
else
if last symbol = minus
then
begin next symbol;
if digit last symbol
then begin UnSigned number;
if (last symbol = comma \lor last symbol = close) \land
small
then
begin type:= ar;
apd:= 420 \times 13 + value of constant
end
else
begin Start implicit subroutine (future);
Arithconstant; Next primary; Next factor;
Macro (NEG); Rest of arboolexp (type);
Macro (EXITIS);
apd:= mask[type] + begin address
end
else begin Start implicit subroutine (future);
Term; Macro (NEG);
Rest of arboolexp (type);
Macro (EXITIS); apd:= mask[type] + begin address
end
end
else
if last symbol = true \lor last symbol = false
then
begin
if last symbol = comma \lor last symbol = close
then apd:= 420 \times 6 + (if n = true then 0 else 1)
else begin Start implicit subroutine (future);
Macro2 (TBC, n);
Rest of boolexp;
Macro (EXITIS);
apd:= mask[type] + begin address
end
else begin Start implicit subroutine (future); Exp (type);
Macro (EXITIS); apd:= mask[type] + begin address
end
end
Actual parameter;
procedure Start implicit subroutine (future); integer future;
begin  If future = 0 then Macro2 (JI, future);
       Macro (ENTRY);
end Start implicit subroutine;

integer procedure Number descriptor;
begin  Number descriptor:=
       if small then d20 * 7 + value of constant
       else d20 * (if real number then 4 else 5)
       + address of constant
end Number descriptor;

procedure Process operator (n); integer n;
begin  Integer count;
       count:= 0;
       if last symbol = open
       then begin
           next: next symbol; Arithexp; count:= count + 1;
           if last symbol = comma
           then begin Macro (STACK); goto next end;
           if last symbol = close
           then next symbol
           else ERNORMESSAGE (361)
       end;
       if count # List length (n) then ERNORMESSAGE (363);
       Macro (Operator macro (n))
end Process operator;

Boolean procedure Nonassignable (apd); integer apd;
begin  integer rank;
       rank:= apd : 420;
       Nonassignable:= (rank # 15) ∧ (rank = rank ± 16 × 16) > 3
end Nonassignable;

procedure Line;
begin  If lnc # last lnc then Line1 end Line;

procedure Line1;
begin  If wanted then begin last lnc:= lnc; Macro2 (LNC, lnc) end
end Line1;
procedure Statement;
begin  Ifstatement forbidden:= false; Stat end Statement;

procedure Unconditional statement;
begin  Ifstatement forbidden:= true; Stat end Unconditional statement;

procedure Stat;
begin  Integer n, save inc;
    If letter last symbol
        then begin save inc:= line counter;
        n:= Identifier;
            if Designtional (n)
                then begin label declaration (n); Stat end
            else begin inc:= save inc; Line;
                if Subscrvar (a) \ last symbol = colonequal
                    then Assignstat (n)
                else Procstat (n)
                end
        end
    else
    If digit last symbol
        then begin Unsigned number;
            if in name list
                then begin Label declaration (Integer label); Stat end
            else ERRMESSAGE (364)
        end
    else begin if last symbol = goto
        then begin Inc:= line counter; Line; Gotostat end
        else if last symbol = begin
            then begin Inc:= line counter; next symbol;
                if declarator last symbol
                    then begin Inc:= save inc; Line; Block end
                else Compound tail;
                next symbol
            end
        else if last symbol = if
            then begin if Ifstatement forbidden
                then ERRMESSAGE (365);
                Inc= line counter; Line; Ifstat
            end
        else if last symbol = for
            then begin Inc:= line counter; Line; Forstat;
                if last symbol = else
                    then ERRMESSAGE (366)
                end
        end
    end Stat;
procedure Gostat;
begin
  Integer n;
  next symbol;
  if letter last symbol
    then begin n:= Identifier ; Subscripted variable (n);
      if local label (n)
        then begin Test for count (n); Macro2 (JU, n) end
      else begin Designame (n); Macro (JUA) end
    end
  else begin Desigexp; Macro (JUA) end
end Gostat;

procedure Compound tail;
begin
  Statement;
  if last symbol ≠ semicolon ∧ last symbol ≠ end
    then begin ERROMESSAGE (393); skip rest of statement (Statement)
      end;
  if last symbol = semicolon
    then begin next symbol; Compound tail end
end Compound tail;

procedure Istat;
begin
  Integer future1, future2, save lnc, last lnc;
  future1:= future2:= 0; save lnc:= line counter;
  next symbol; Boolean Macro2 (CUJ, future1);
  if last symbol = then then next symbol else ERROMESSAGE (368);
  Unconditional statement;
  if last symbol = else
    then begin Macro2 (JU, future2); Substitute (future1);
      last lnc:= last lnc; last lnc:= save lnc;
      next symbol; Statement; Substitute (future2);
      if last lnc > last lnc then last lnc:= last lnc
    else begin Substitute (future1);
      if last lnc > save lnc then last lnc:= save lnc
    end
end Istat;

procedure Forstat;
begin
  Integer future, save lnc;
  save lnc:= line counter;
  10:= 0; next symbol; For 10;
  future:= 0; Macro2 (JU, future); if 10 ≠ 0 then Substitute (10);
  if last symbol = do then next symbol else ERROMESSAGE (369);
  Increase status (increment); for count:= for count + 1;
  Statement;
  Increase status (− increment); for count:= for count − 1;
  if last lnc < 0 ∨ lnc ≠ save lnc
    then begin lnc:= save lnc; Line1 end;
  Macro2 (JU, status); Substitute (future)
end Forstat;
procedure Store preparation;
begin
if Subscrvar (controlled variable) then Macro2 (SUBJ, -12)
else
if Formal (controlled variable)
then Macro2 (DOS2, controlled variable)
end Store preparation;

procedure Store macro;
begin
if Subscrvar (controlled variable)
then begin if Formal (controlled variable) then Macro (STFSU)
else
if Integer (controlled variable) then Macro (STSI)
else Macro (STSR);
end
else if Formal (controlled variable)
then Macro2 (DOS3, controlled variable)
else if Integer (controlled variable)
then Macro2 (STI, controlled variable)
else Macro2 (STR, controlled variable)
end Store macro;

procedure Take macro;
begin
if Subscrvar (controlled variable)
then Macro2 (SUBJ, -11)
else Arithname (controlled variable)
end Take macro;

procedure For list;
begin
if letter last symbol
then
begin
controlled variable:= Identifier;
if Nonarithmetic (controlled variable)
then ERRORMESSAGE (370);
if Subscrvar (controlled variable)
then
begin
13:= 0; Macro2 (JU, 13);
15:= Order counter;
Address description (controlled variable);
Macro2 (EXITSV, 1 - 2 x dimension);
11:= Order counter;
Macro2 (SUBJ, -14);
if Formal (controlled variable) then Macro (TSCVU)
else
if Integer (controlled variable) then Macro (TISCV)
else Macro (TIRCV);
12:= Order counter;
Macro2 (SUBJ, -14); Macro (FADCV);
Substitute (13)
end
else if Function (controlled variable)
then ERRORMESSAGE (371);
if last symbol # colonequal then ERRORMESSAGE (372);
list: l3:= Order counter;
   Macro2 (TSIC, 0); Macro2 (SSTI, status);
14:= Order counter;
Store preparation;
next symbol; Arithexp;
   if last symbol = comma V last symbol = do
      then begin Store macro; Macro2 (JU, 10);
               Substitute (13)
      end
   else
   if Last symbol = while
      then begin Store macro;
               next symbol; Boolean;
               Macro2 (YUMJU, 10); Subst2 (14, 13)
      end
   else
   if Last symbol = step
      then begin 15:= 0; Macro2 (JU, 15); 14:= Order counter;
               next symbol; complicated:= false; Arithexp;
               complex step element:= complicated V Order counter > 14 + 1;
               if complex step element then Macro (EXIT);
               Substitute (13);
               Store preparation; Take macro; Macro (STACK);
               if complex step element then Macro2 (SUBJ, + 14)
               else Macro2 (DO, 1 4);
               Macro (ADD);
               Substitute (15);
               Store macro;
               if Subscrvar (controlled variable) V
                  Formal (controlled variable)
               then Take macro;
               Macro (STACK);
               if last symbol = until
               then begin next symbol; Arithexp end
                  else ERROMESSAGE (373);
               Macro (TEST1);
               if complex step element then Macro2 (SUBJ, + 14)
                  else Macro2 (DO, 1 4);
               Macro (TEST2); Macro2 (YUMJU, 10)
      end
   else ERROMESSAGE (374);
   if Last symbol = comma then goto list
end
else ERROMESSAGE (375)
end FOR list;
procedure Switch declaration;
begin integer m;
next symbol;
if letter last symbol
then
begin switch identifier:= Identifier;
   number of switch elements:= List length (switch identifier);
   if last symbol = colonequal
   then
   begin integer array
      sword list[]; number of switch elements];
      switch list count:= 0; in switch declaration:= true;
      next: switch list count:= switch list count + 1;
      next symbol;
      if letter last symbol
      then
      begin m:= Identifier;
         if Nondesignational (m) then ERRORMESSAGE (376);
         Subscrvar (m)
         then
            begin sword:= $45613055 + Order counter;
               Subscripted variable (m); Macro (EXIT)
            end
         else
            sword:= (if Formal (m)
                  then $35685503
                      else $47189592 + (if Dynamic (m)
                                       then function digit
                                       else 0)) +
            Address (m)
      end
      else
      if digit last symbol
      then
      begin Unsigned number;
         if in name list
         then
            sword:= $4718592 +
               (if Dynamic (integer label)
                  then function digit
                  else 0) +
               Address (integer label)
               else ERRORMESSAGE (377)
      end
      else
      beginsword:= $45613055 + Order counter;
      Designexp; Macro (EXIT)
      end;
if switch list count > number of switch elements
    then ERROREM(376);
    sword list[sword list count]= sword;
    if last symbol = comma then goto next;
    if switch list count < number of switch elements
        then ERROREM(379);
        Mark position in name list (switch identifier);
        in switch declaration = false;
        Macro2 (CODE, number of switch elements);
        m:= 0;
    next sword:
        if m < switch list count ∧
            m < number of switch elements
            then begin m:= m + 1; sword:= sword list[m];
                Macro2 (CODE, sword); goto next sword
            end;
    end;
else ERROREM(380)
end;
else ERROREM(381)
end;

Switch declaration;

procedure Array declaration;
begin
    integer n, count;
    next symbol lnc:= line counter; Line;
    ni:= identifier; dimension:= list length (n); count:= 1;
    next:
        if last symbol = comma then begin next symbol; identifier;
            count:= count + 1; goto next
        end;
        if last symbol = sub then begin
            in array declaration:= true;
            Bound pair list;
            in array declaration:= false
        end;
    end;
    if last symbol = comma then Array declaration
end;

procedure Bound pair list;
begin
    next symbol; Arithexpr; Macro (STACK);
    if last symbol = colon then begin next symbol; Arithexpr;
        Macro (STACK)
    end;
else ERROREM(383);
    if last symbol = comma then Bound pair list
    else if last symbol = bus
        then next symbol
    else ERROREM(384)
end;

Bound pair list;
procedure Procedure declaration;
begin integer n, f, count, save inc;
ext symbol; f := n := Identifier;
Skip parameter list; skip value list; skip specification list;
if " in library (n) then Mark position in name list (n);
if " in code (n)
then Translate code
else begin if Function (n) then Set inside declaration (n, true);
entrance block;
Macro2 (IFTR, display level);
Macro2 (INCB, top of display);
for count := List length (n) step - 1 until 1 do
begin f := Next formal identifier (f);
if In value list (f)
then
begin if Subscrvar (f)
then Macro (CEN)
else
begin if Arithmetic (f)
then begin if Integer (f)
then Macro (CIV)
else Macro (CRV)
end
else if Boolean (f) then Macro (CBV)
else if String (f) then Macro (CSV)
elose Macro (CLV)
end
end
else if Assigned to (f) then Macro (CLPN)
elose Macro (CEN)
end;
Macro2 (TBL, display level);
Macro2 (ENTRPB, local space);
Label list; f := n;
for count := List length (n) step - 1 until 1 do
begin f := Next formal identifier (f);
if In value list (f) \ Subscrvar (f)
then begin Macro2 (TAA, f);
if Integer (f) then Macro (TIAV)
elose Macro (TAV)
end
end;
save inc := last inc; last inc := - line counter;
Save and restore inc (SLNC, n);
if last symbol = begin
    then begin next symbol; if declarator last symbol
        then Declaration list;
    Compound tail; next symbol
end;
else Statement;
inc := last inc := save inc;
if Function (n)
    then begin Set inside declaration (n, false);
        if Arithmetic (f) then Arithname (f) else
            if Boolean (f) then Boolname (f)
            else begin Stringname (f); Macro (LOS) end
        end;
Save and restore inc (RLNC, n);
if use of counter stack then Macro (EXITPC)
else Macro (EXITP);
exit block
end;
end Procedure declaration;

procedure Save and restore inc (macro, n); integer macro, n;
begin if wanted ∧ Function (n) then Macro2 (macro, Local position1 (n))
end; Save and restore inc;

procedure Block;
begin entrance block;
    Macro2 (TB1, display level); Macro2 (EWFB, local space);
    Label list; Declaration list; Compound tail;
    if use of counter stack then Macro2 (EXITC, display level)
    else Macro2 (EXITF, display level);
exit block
end; Block;
procedure Declaration list;
begin  Integer future, arr dec;
  Future:= arr dec:= 0;
next:  if type declarator last symbol then skip type declaration
  else
    if arr declarator last symbol
      then begin if future # 0
        then begin Substitute (future);
        future:= 0
        end;
        arr dec:= 1; Array declaration
      end
    else
      begin if future = 0 then Macro2 (JU, future);
        if last symbol = Switch then Switch declaration
        else Procedure declaration
      end;
    if last symbol = semicolon then next symbol
      else ERRORMESSAGE (385);
    if declarator last symbol then goto next;
    if future # 0 then Substitute (future);
    if arr dec # 0 then Macro2 (SWP, display level)
end Declaration list;

procedure Label list;
begin  Integer n, count;
  count:= Number of local labels;
  if count > 0
    then begin Macro2 (DECB, 2 * count);
        Macro2 (LAD, display level);
        n:= 0; for count1= count step = 1 until 1 do
          begin next; ni:= Next local label (n);
            if Super local (n) then goto next;
            if count = 1 then Macro2 (LAST, n)
            else Macro2 (NIL, n)
          end
    end
end Label list;
procedure Program;
  begin Integer n;
    if letter last symbol
      then begin n:= Identifier;
        if last symbol = colon
          then Label declaration (n);
  end
  else
    if digit last symbol
      then begin Unsigned number;
        if in name list \ last symbol = colon
          then Label declaration (integer label);
  end
  else
    if last symbol = begin
      then begin next symbol;
        if declarator last symbol then Block
        else Compound tail;
      end
    end
  else begin next symbol; Program end
end Program;

procedure Label declaration (n); integer n;
begin Last Inc:= - line counter;
  if Subscrvar (n) then begin ERRORMESSAGE (388);
    Subscripted variable (n)
  end
  else Mark position in name list (n);
  if last symbol = colon then next symbol else ERRORMESSAGE (389)
end Label declaration;

procedure Substitute (address); integer address;
begin Subst2 (Order counter, address) end Substitute;

procedure Subst2 (address1, address2);
value address1, address2; integer address1, address2;
begin Integer instruction, Instruct part, address part;
  address2:= abs (address2);
  instruction:= space[prog base + address2];
  Instruct part:= instruction : d15 × d15 -
    (if instruction < 0 then 32767 else 0);
  address part:= instruction - instruct part;
  space[prog base + address2]:= instruct part + address1;
  if address part = 0
    then begin if instruct part = end of list
      then space[prog base + address2]:= - space[prog base + address2]
      end
    else Subst2 (address1, address part)
end Subst2;
integer procedure Order counter;
begin Macro (ENXY); Order counter:= instruct counter
end Order counter;

procedure Macro (macro number); integer macro number;
begin Macro2 (macro number, parameter) end Macro;

procedure Macro2 (macro number, metaparameter);
integer macro number, metaparameter;
begin macro:= if macro number < 512 then macro list[macro number] else macro number;
    parameter:= metaparameter;
    if state = 0 then begin if macro = STACK then state:= 1
    else if Simple arithmetic take macro then Load (3)
    else Produce (macro, parameter);
    end;
    else if state = 1 then begin Load (2);
    if 7 Simple arithmetic take macro then begin Produce (STACK, parameter); Unload end
    end;
    else if state = 2 then begin if Optimizable operator then Optimize
    else begin Produce (STACK, parameter); state:= 3;
    Macro2 (macro, parameter)
    end;
    end;
    else if state = 3 then begin if macro = NEQ then Optimize
    else begin Unload; Macro2 (macro, parameter) end;
    if Forward jumping macro ∧ metaparameter ≤ 0 then Assign (metaparameter)
    end Macro2;
procedure Load (state i); integer state i;
begin
  stack0:= macro; stack1:= parameter; state:= state i end Load;

procedureUnload;
begin
  Produce (stack0, stack1); state:= C end Unload;

procedure Optimize;
begin
  stack0:= tabel[5 x Opt number (macro) + Opt number (stack0)];
  Unload
end Optimize;

procedure Assign (metaparameter); integer metaparameter;
begin
  metaparameter:= - (instruct counter - 1) end Assign;

procedure Produce (macro, parameter); integer macro, parameter;
begin
  integer number, par number, entry, count;
  if macro = EMPTY then
  else
    if macro = CODE
    then begin
      space[prog base + instruct counter]:= parameter;
      instruct counter:= instruct counter + 1;
      test pointers
    end
    else begin
      number:= Instruct number (macro);
      par number:= Par part (macro);
      entry:= Instruct part (macro) - 1;
      if par number > 0
      then Process parameter (macro, parameter);
      Process stack pointer (macro);
      for count:= 1 step 1 until number do
      PRODUCE (CODE, instruct list[entry + count] +
          (if count = par number
          then parameter else 0))
    end
  end
end Produce;
procedure Process stack pointer (macro); integer macro;
begin if 'l' in code body
then
begin integer reaction;
reaction:= B reaction (macro);
if reaction < 9
then begin b:= b + reaction - 1;
    if b > max depth then max depth:= b
    end
else
if reaction = 10 then b:= 0
else
if reaction = 11 then b:= b - 2 x (dimension - 1)
else
if reaction = 12
then begin ifecount = 0
then
begin ret level:= b;
    ret max depth:= max depth;
    b:= 0; max depth:= max depth isr
    end;
ecount:= ecoun t + 1
end
else
if reaction = 13
then begin if macro = EXITSV
then
begin if b > max depth isr
then max depth isr:= b;
    b:= b - 2 x (dimension - 1)
end;
if ecoun t = 1
then
begin if max depth > max depth isr
then max depth isr:= max depth;
    b:= ret level;
    max depth:= ret max depth
end;
if ecoun t > 0 then ecoun t:= ecoun t - 1
end
else
if reaction = 14
then begin b:= display level + top of display;
    if b > max display length
then max display length:= b;
    ret max depth:= max depth
end
else
if reaction = 15
then begin if b > max proc level
then max proc level:= b;
    b:= 0; max depth:= ret max depth
end
end Process stack pointer;
procedure Process parameter (macro, parameter);
  integer macro, parameter;
begin
  if Value like (macro)
  then
    begin
      if macro = TEC
      then parameter:= if parameter = true then 0 else 1
      else
        if macro = SWP then parameter:= d9 × parameter
        else
          if macro ≠ EXITSV then parameter:= abs (parameter)
    end
  else
    begin
      if macro = JJ ∨ macro = SUBJ ∨ macro = NIL ∨ macro = LAST
      then begin if parameter < 0
               then parameter:= - parameter
               else parameter:= Program address (parameter)
      end
    end
end Process parameter;

Boolean procedure Simple arithmetic take macro;
begin
  Simple arithmetic take macro:= bit string (d1, d0, macro) = 1
end Simple arithmetic take macro;

Boolean procedure Optimizable operator;
begin
  Optimizable operator:= bit string (d2, d1, macro) = 1
end Optimizable operator;

Boolean procedure Forward jumping macro;
begin
  Forward jumping macro:= bit string (d3, d2, macro) = 1
end Forward jumping macro;

Boolean procedure Value like (macro); integer macro;
begin
  Value like:= bit string (d4, d3, macro) = 1
end Value like;

integer procedure Opt number (macro); integer macro;
begin
  Opt number:= bit string (d8, d4, macro) end Opt number;

integer procedure Instruct number (macro); integer macro;
begin
  Instruct number:= bit string (d10, d8, macro) end Instruct number;
integer procedure Par part (macro); integer macro;
begin Par part:= bit string (d12, d10, macro) end Par part;

integer procedure Instruct part (macro); integer macro;
begin Instruct part:= bit string (d21, d12, macro) end Instruct part;

integer procedure B reaction (macro); integer macro;
begin B reaction:= macro ; d21 end B reaction;

integer procedure Code bits (n); integer n;
begin Code bits:= space[nl base = n] + d19 end Code bits;

integer procedure Character (n); integer n;
begin Character:= bit string (d24, d19, space[nl base = n])
end Character;

Boolean procedure Arithmetic (n); integer n;
begin Integer:= true;
  Tr:= type bits (n);
  Arithmetic:= Character (n) + 2^4 ∧ (1 < 2 ∨ 1 = 4)
end Arithmetic;

Boolean procedure Real (n); integer n;
begin Real:= Character (n) + 2^4 ∧ type bits (n) = 0 end Real;

Boolean procedure Integer (n); integer n;
begin Integer:= type bits (n) = 1 end Integer;

Boolean procedure Boolean (n); integer n;
begin Boolean:= type bits (n) = 2 end Boolean;

Boolean procedure String (n); integer n;
begin String:= type bits (n) = 3 end String;

Boolean procedure Designational (n); integer n;
begin Designational:= type bits (n) = 6 end Designational;

Boolean procedure Arbostr (n); integer n;
begin Arbostr:= Character (n) + 2^4 ∧ type bits (n) < 6 end Arbostr;

Boolean procedure Unknown (n); integer n;
begin Unknown:= type bits (n) = 7 end Unknown;
Boolean procedure Nonarithmetic (n); integer n;
  begin
    integer i;
    if= type bits (n);
    Nonarithmetic: Character (n) = 2^i \lor i = 2^i \lor i = 3 \lor i = 6
  end Nonarithmetic;

Boolean procedure Nonreal (n); integer n;
  begin
    Nonreal:= Nonarithmetic (n) \lor type bits (n) = 1 end Nonreal;

Boolean procedure Noninteger (n); integer n;
  begin
    Noninteger:= Nonarithmetic (n) \lor type bits (n) = 0
  end Noninteger;

Boolean procedure Nonboolean (n); integer n;
  begin
    integer i;
    if= type bits (n); Nonboolean:= i \neq 2 \land i \neq 5 \land i \neq 7
  end Nonboolean;

Boolean procedure Nonstring (n); integer n;
  begin
    integer i;
    if= type bits (n); Nonstring:= i \neq 3 \land i \neq 5 \land i \neq 7
  end Nonstring;

Boolean procedure Nondesignational (n); integer n;
  begin
    Nondesignational:= type bits (n) < 0 end Nondesignational;

Boolean procedure Nontype (n); integer n;
  begin
    Nontype:= type bits (n) = 0 \lor (Proc (n) \land Nonfunction (n))
  end Nontype;

Boolean procedure Simple (n); integer n;
  begin
    Simple:= 'Code bits (n) = 127 \lor Simple1 (n) end Simple;

Boolean procedure Simple1 (n); integer n;
begin
  Simple1:= Character (n) \land d3 \neq 0 end Simple1;

Boolean procedure Subscrvar (n); integer n;
begin
  Subscrvar:= Character (n) \land d3 = 1 end Subscrvar;

Boolean procedure Proc (n); integer n;
begin
  Proc:= Character (n) \land d3 > 1 \land Code bits (n) \neq 127 end Proc;

Boolean procedure Function (n); integer n;
begin
  Function:= Character (n) \land d3 = 2 end Function;
Boolean procedure Nonsimple (n); integer n;
begin Nonsimple:= \( \forall \text{Proc}(n) \)
then (\text{Formal}(n) \lor \text{Function}(n)) \land
\text{List length}(n) < 1
else false \)
end Nonsimple;

Boolean procedure Nonsubscvar (n); integer n;
begin Nonsubscvar:= Simple1(n) \lor \text{Proc}(n) end Nonsubscvar;

Boolean procedure Nonproc (n); integer n;
begin Nonproc:= \( \forall \text{Character}(n) 1 \leq d \leq 2 \lor \)
(Formal(n) \land Simple1(n) \land \forall \text{Assigned to}(n))
end Nonproc;

Boolean procedure Nonfunction (n); integer n;
begin Nonfunction:= \( \forall \text{Function}(n) \lor \text{Formal}(n) \)
end Nonfunction;

Boolean procedure Formal (n); integer n;
begin Formal:= \text{Code bits}(n) > 63 \land \text{Formal}(n)
end Formal;

Boolean procedure In value list (n); integer n;
begin In value list:= \text{Code bits}(n) > 63 \land \text{Formal}(n)
end In value list;

Boolean procedure Assigned to (n); integer n;
begin Assigned to:= \text{bit string}(d19, d18, \text{space}[\text{nl base} - n]) = 1
end Assigned to;

Boolean procedure Dynamic (n); integer n;
begin Dynamic:= \text{Code bits}(n) > 63 \lor \text{Assigned to}(n)
end Dynamic;

Boolean procedure In library (n); integer n;
begin In library:= \text{space}[\text{nl base} - n - 1] > 25 \land \text{In library}
end In library;

Boolean procedure Id1 (k, n); integer k, n;
begin Id1:= \text{bit string}(2 \times k, k, \text{space}[\text{nl base} - n - 1]) = 1 \land \text{Id1}
end Id1;

Boolean procedure Operator like (n); integer n;
begin Operator like:= \text{Id1}(d23, n) \land \text{Operator like};

Boolean procedure Outside declaration (n); integer n;
begin Outside declaration:= \text{Id1}(d22, n) \land \text{Outside declaration};
Boolean procedure Ass to function designator (n); integer n;
begin Ass to function designator:= Id1 (d21, n)
end Ass to function designator;

Boolean procedure Declared (n); integer n;
begin Declared:= Id1 (d19, n) end Declared;

Boolean procedure Super local (n); integer n;
begin Super local:= Id1 (d18, n) end Super local;

procedure Change (k, n); integer k, n;
begin Integer i, j;
  Tr:= space[nl base - n - 1]; j:= i = i : (2 x k) x (2 x k);
  space[nl base - n - 1]:= i + (if j < k then k else -k)
end Change;

integer procedure Local position (n); integer n;
begin if 'Ass to function designator (n) then Change (d21, n);
  Local position:= Local position! (n)
end Local position;

integer procedure Local position! (n); integer n;
begin Local position!:= n + 2 end Local position!;

procedure Set inside declaration (n, bool); integer n; Boolean bool;
begin Change (d22, n);
  if 'booll V Ass to function designator (n))
  then ERRORMESSAGE (390)
end Set inside declaration;
procedure Mark position in name list (n); integer n;
begin
  integer address;
  if Declared (n) then ERROREM (391)
  else begin address := Program address (n);
    if address = 0 then Substitute (address);
    Change (d19, n)
  end
end Mark position in name list;

integer procedure Program address (n); integer n;
begin
  integer word, head, m;
  m := if Code bits (n) = 6 then n + 1 else n;
  word := space[nl base - m];
  head := word + d18 x d18;
  if 'l Declared (n) then space[nl base - m] := head + Order counter;
  Program address := word - head
end Program address;

integer procedure Address (n); integer n;
begin
  integer word, tail, level;
  word := Code bits (n);
  if word > 13 and word < 25 then tail := Program address (n)
  else begin word := space[nl base - n];
    tail := word - word + d18 x d18;
    if Dynamic (n) then begin level := tail + 99;
      if level = proc level and
      7 in switch declaration then tail := tail + 99 x (63 - level)
    end
    Address := tail
end Address;

integer procedure List length (n); integer n;
begin
  List length := bit string (d95, d9, space[nl base - n - 1]) - 1
end List length;

procedure Test for count (n); integer n;
begin
  if space[nl base - n - 1] : d20 > for count then ERROREM (392)
end Test for count;
procedure Check dimension (n); integer n;
begin Integer i;
  if Code bits (n) = 1 then List length (n);
  if i > 0 ∧ i ≠ dimension then ERRORMESSAGE (393)
end Check dimension;

procedure Check list length (f, n); integer f, n;
begin Integer i, j;
  i:= List length (f);
  j:= if Code bits (n) = 1 then List length (n);
  if i > 0 ∧ j > 0 ∧ i ≠ j then ERRORMESSAGE (394)
end Check list length;

procedure Check type (f, n); integer f, n;
begin if (Designtational (f) ∧ Nondesigntational (n)) ∨
  (Arbost (f) ∧ Nontype (n)) ∨
  (Arithmetic (f) ∧ Nonarithmettic (n)) ∨
  (Boolean (f) ∧ Nonboolean (n)) ∨
  (String (f) ∧ Nonstring (n))
  then ERRORMESSAGE (395)
end Check type;

integer procedure Number of local labels;
begin Number of local labels:=
  bit string (d13, d40, space[nl base – block cell pointer – 3])
end Number of local labels;

integer procedure Next local label (n); integer n;
begin Next local label:=
  if n = 0 then space[nl base – block cell pointer – 3] + 3
  else next identifier (n)
end Next local label;

integer procedure Next formal identifier (n); integer n;
begin Next formal identifier:=
  next identifier (n + (if Formal (n) ∨ In library (n) ∨
    In value list (n) then 2
    else if Function (n) then 9 else 0))
end Next formal identifier;

procedure Increase status (increment); integer increment;
begin space[nl base – block cell pointer – 2]:= 
  space[nl base – block cell pointer – 2] + increment
end Increase status;

integer procedure Identifier;
begin read Identifier; Identifier:= look up Identifier;
procedure Skip parameter list;
begin  if last symbol = open
    then begin next symbol; skip type declaration;
    if last symbol = close then next symbol
    end;
  if last symbol = semicolon then next symbol
end Skip parameter list;

procedure Translate code;
begin  Integer macro, parameter;
  if last symbol = quote
    then begin in code body:= true;
    next; next symbol;
    if digit last symbol
    then
      begin macro:= unsigned integer (0);
      if macro < 512 then macro:= macro list[macro];
      if Par part (macro) > 0
      then
        begin
          if last symbol = comma
          then next symbol
          else ERRORMESSAGE (396);
          if letter last symbol
          then parameter:= Identifier
          else
            if digit last symbol
            then parameter:= unsigned integer (0)
            else
              if last symbol = minus
              then
                begin next symbol;
                if digit last symbol
                then parameter:=
                  unsigned integer (0)
                else ERRORMESSAGE (397)
                end
              else ERRORMESSAGE (399);
          Macro2 (macro, parameter)
        end
      else ERRORMESSAGE (400)
    end
  else ERRORMESSAGE (399);
  if last symbol = comma then goto next;
  if last symbol = unquote then next symbol
  else ERRORMESSAGE (401);
end Translate code;
procedure Unsigned number;
begin  Integer p;
     unsigned number;
      if 7 small
          then begin p:= 0;
       next: if p = dp0 then goto found;
            if space(prog base + p) # value of constant V
               space(prog base + p + 1) # decimal exponent
               then begin p:= p + 2; goto next end;
               found: address of constant:= p
      end
end Unsigned number;

procedure Arithconstant;
begin  if small then Macro2 (TSIC, value of constant)
       else
            if real number then Macro2 (TRC, address of constant)
            else Macro2 (TIC, address of constant)
end Arithconstant;

integer procedure Operator macro (n); integer n;
begin  Operator macro:= space[nl base = n - 2] end Operator macro;

procedure Constant string;
begin  Integer word, count;
       quote counter:= 1;
next0: word:= count:= 0;
next1: next symbol;
       if last symbol # unquote
          then begin word:= d8 x word + last symbol;
                   count:= count + 1;
               if count = 3
                   then begin Macro2 (CODE, word); goto next0 end;
               goto next1
          end;
next2: word:= d8 x word + 255; count:= count + 1;
       if count < 3 then goto next2;
Macro2 (CODE, word); quote counter:= 0; next symbol
end Constant string;

integer procedure Relatmacro;
begin  Relatmacro:= if last symbol = les then LES else
                  if last symbol = nst then MST else
                  if last symbol = nor then MIR else
                  if last symbol = Ist then LSR else
                  if last symbol = eqv then EQU else UQU
end Relatmacro;
main program of translate scan:
    if 't' text in memory
      then begin NEWPAGE;
        PRINTTEXT ('{input tape for translate scan}');
      end;
    start:= instruct counter; last nlp:= nlp;
    runnumber:= 300; init; increment:= d\3;
    state:= b:= max depth:= max depth isr:=
    max display length:= max proc level:= ecount:= 0;
    in switch declaration:= in code body:= false;
    next block cell pointer:= 0;
    entrance block; next symbol;
    Program;
    sum of maxima:= max depth + max depth isr +
    max display length + max proc level;
    Macro2 (CODE, sum of maxima);
    output
end translate;
procedure output;
begin
   Integer i, k, apostrophe, instruc number, par, address;
   procedure pucar (n); integer n;
   begin
      Integer i;
      for i:= 1 step 1 until n do PUNLR
   end pucar;
   procedure tabspace (n); integer n;
   begin
      Integer i, k;
      k:= n * 8;
      for i:= 1 step 1 until k do FUSYM (118);
      PUSPACE (n - k * 8)
   end tabspace;
   procedure absfixp (k); integer k;
   begin
      ABSFIXP (4, 0, k); pucar (2) end absfixp;
   procedure punch (bool); Boolean bool;
   begin
      if bool then PUTFEXT ('true')
         else PUTFEXT ('false');
   end punch;
   procedure punch octal (n); value n; integer n;
   begin
      Integer i, k;
      Boolean minussign;
      minussign:= n < 0;
      n:= abs (n);
      FUSYM (if minussign then minus else plus);
      FUSYM (apostrophe);
      for i:= d21, d18, d15, d12, d9, d6, d3, d0 do
         begin
            k:= n - i;
            n:= n - k * i;
            FUSYM (k)
            FUSYM (apostrophe)
         end
   end punch octal;
   apostrophe:= 120;
   PUNLR;
   if runumber = 100 then
      begin
         tabspace (22); PUTFEXT ('prescan'); pucar (2);
         PUTFEXT (' erroneous'); PUSPACE (14);
         punch (erroneous); PUTFEXT (' text length');
         PUSPACE (12);
         absfixp (if text in memory then text pointer + 1 else 0);
         PUTFEXT ('name'); pucar (3);
         for i:= 0 step 1 until nlp - 1 do
            begin
               tabspace (7); ABSFIXP (4, 0, i); PUSPACE (5);
               punch octal (space[nl base - 1]); PUNLR
            end;
      end
      STOPCODE;
   PUNLR; PUTFEXT ('top'); pucar (2);
   PUTFEXT ('start'); pucar (2);
   PUTFEXT ('program'); pucar (2);
for i:= prog base step 1 until instrct counter = 1 do
  begin
    tabspace (39); PUTTEXT (type scan); pucar (2);
    tabspace (39); punch (erroneous); tabspace (39);
    absfxip (if text in memory then text pointer + 1 else 0);
    pucar (2);
    for i:= 0 step 1 until nlp = 1 do
      begin
        tabspace (34); punch octal (space[nl base - 1]);
        PUNLCR
      end;
    STOPCODE; pucar (7);
  end
else
begin
  tabspace (54); PUTTEXT (translate); pucar (2);
  tabspace (55); punch (erroneous); tabspace (55);
  absfxip (if text in memory then text pointer + 1 else 0);
  pucar (2);
  for i:= 0 step 1 until nlp = 1 do
    begin
      tabspace (50); punch octal (space[nl base - 1]);
      PUSPACE (2); ABSFIXP (4, 0, 1); PUNLCR
    end;
  STOPCODE; PUNLCR;
  tabspace (55); absfxip (dp0);
  tabspace (55); absfxip (start); pucar (2);
  for i:= prog base step 1 until start = 1 do
    begin
     tabspace (48); FIXP (13, 0, space[i]);
      PUSPACE (2); ABSFIXP (4, 0, 1); PUNLCR
    end;
  PUNLCR;
end
for i:= start step 1 until instrct counter = 1 do
begin
  k:= space[i]; par:= k : 32768;
  address:= k - par * 32768;
  instruct number:= par / 10;
  par:= par - instruct number * 10;
  tabspace (48); ABSFIXP (4, 0, instruct number);
  ABSFIXP (1, 0, par); ABSFIXP (5, 0, address);
  PUSPACE (2); ABSFIXP (4, 0, 1); PUNLCR
end

main program:
   for n:= 0 step 1 until end of memory do space[n]:= 0;
   instruct counter:= prog base:= nlp:= 0;
   text base:= end of memory + 3;
   nl base:= end of memory;

   prescan0;
   if l.derroneous
      then begin prescan1;
            translate
            end;

endrun:
end
end
2. FIXED DATA

2.0 DESCRIPTION

This section contains a summary description of the data (given in section 2.1) to be offered to the compiler in front of the ALGOL 60 text that has to be translated. The major part of pages 2 and 3 of the compiler text is dedicated to reading and storing these data. We will briefly comment on the function, and possibly the coding, of categories of data items:

The values of the basic symbols are the internal values of the <delimiter>s (§6, 2.3.), the <logical value>s (§6, 2.2.2.) and the symbols "new line", "underlining", "bar" and "lower case".

The information stored into the macro identifiers is explained in section 4.2, while the variables function letter, function digit and c variant play a role in the procedure Process parameter (page 81). It is the task of the latter three to change the addressing mode of instructions into dynamic.

The i-th entry in the table internal representation gives the internal value of the flexowriter symbol corresponding to its 7-holes MC-flexowriter code punching, the binary pattern of which equals the number i. The coding is as follows:

0 means parity error (MC-flexowriter code has odd parity),
-1 stands for a punching without any corresponding character,
-2 means the corresponding symbol has to be skipped (blank, stopcode, backspace, erase),
-122 corresponds to a lower case punching, and
-124 to an upper case punching.

For all other values n we have:

\[ n = \text{(internal value of the upper shift symbol)} \times d_8 + \text{internal value of the lower shift symbol}, \]

\[ e.g., 9482 = 37 \times d_8 + 10, \] where 37 is the internal value of "A" and 10 is the internal value of "a".

The table word delimiter gives the correspondence between the internal values of a word delimiter and its first two symbols as follows:
\[ n = (\text{internal value of the first letter of the word delimiter}) \times d14 + \\
    (\text{internal value of the second letter of the word delimiter}) \times d7 + \\
    \text{internal value of the word delimiter,} \\
    \text{e.g., } 232160 = 14 \times d14 + 21 \times d7 + 96, \text{ where 14 is the internal} \\
    \text{value of "e", 21 that of "l", and 96 that of the word} \\
    \text{delimiter else.} \]

Hence no distinction is made between the basic symbols \textit{step} and \textit{strong}.

The values in the table \textit{macro list} only matter if code procedures are to be
accepted in full extent. In that case the \(i\)-th entry in the table should
contain the properties of the macro with macro number \(i\) (cf. table 4.2).

In \textit{table} the macrowords of the optimized macros are given analogous to the
list of macro identifiers. For their decoding see table 4.2.

The \textit{instruct list} should contain the bit-patterns for the machine instruc-
tions which constitute the macros generated by the compiler (those instruc-
tions are given in table 4.3). For easy interpretation of the output here
\textit{instruct list}(i) = i \times 10 \times d15.

For each of the ten possible types of an expression the table \textit{mask} gives a
code to be used in an actual parameter descriptor.

Finally, the variable \textit{end of memory} gives the upper bound of the array
\textit{space}, and \textit{wanted} determines whether objectcode for line number administra-
tion has to be generated (\textit{wanted} = 0) or not (\textit{wanted} \neq 0).
2.1 DATA

basic symbols:

64  65  66  67  68  69  70  71
72  73  74  75  76  77  78  79
80  81  82  83  84  85  86  87
88  89  90  91  92  93  94  95
96  97  98  99  100 101 102 103
104 105 106 107 108 109 110 111
112 113 114 115 116 117 118 119
120 121 122

macro identifiers:

12583168  840148  4213678  4207154  4215106  4195154
4219136  4223232  4227426  4227698  4235906  4284114
4344436  4258434  1055920  845728  6369792  6377984
6386176  6394368  10596608  23183872  23192064  23200256

23208448  23216384  8540416  8544512  8548608  21135616
21137712  21143268  21147904  21152000  21156096  25334496
8561376  8565472  8569568  8573664  8577760  8581856
8605952  8610048  8614144  27492608  8622336  8626432

8663028  8663624  8663720  14227328  8643072  12845312
10758256 10758352 12857600 12861696 12865792 17064192
8675680  8685776  21270784  21274880  21278976  21283072
21287168 21291264 21293560 21299446 8720640  8724736

8728832  8732928  8737024  8738160  8741376  8749832
8761600  8765696  8769792  8773888  8777984  8782080
8787217  8787145  8787257  8789369  8793488  8797592
8797520  8797516  8797520  8797516  8793508  8799604

8896096  90001472  87911552  9008904  9000048  9006056
9002676  8975736  9024768  9028864  9032972  9037080
9041152  9045248  9049344  9053440  9057536  9061632
9066728  9066736  9066728  9066736  9066728  9066736

9106952  9106696  8791304  87975616  914888  9118984
9119240  28001800  9098504  9202176  9209384  9217544

function letter:  32768

function digit:  65536

c variant:  96504
internal representation:

<table>
<thead>
<tr>
<th></th>
<th>20225</th>
<th>16898</th>
<th>0</th>
<th>17984</th>
<th>0</th>
<th>0</th>
<th>25863</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>25096</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>1</td>
<td>32698</td>
</tr>
<tr>
<td>31611</td>
<td>0</td>
<td>0</td>
<td>17155</td>
<td>0</td>
<td>23301</td>
<td>25606</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>32009</td>
<td>30583</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>20480</td>
<td>0</td>
<td>0</td>
<td>14365</td>
<td>0</td>
<td>14879</td>
<td>15136</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>19016</td>
<td>14108</td>
<td>0</td>
<td>14622</td>
<td>0</td>
<td>0</td>
<td>15393</td>
</tr>
<tr>
<td>15650</td>
<td>0</td>
<td>0</td>
<td>30809</td>
<td>0</td>
<td>-1</td>
<td>30326</td>
<td>0</td>
</tr>
<tr>
<td>19521</td>
<td>0</td>
<td>0</td>
<td>12309</td>
<td>0</td>
<td>12823</td>
<td>13080</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>13795</td>
<td>12052</td>
<td>0</td>
<td>12566</td>
<td>0</td>
<td>0</td>
<td>13337</td>
</tr>
<tr>
<td>13594</td>
<td>0</td>
<td>0</td>
<td>31319</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>9482</td>
<td>9739</td>
<td>0</td>
<td>10253</td>
<td>0</td>
<td>0</td>
<td>11024</td>
</tr>
<tr>
<td>11281</td>
<td>0</td>
<td>0</td>
<td>31832</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>31040</td>
<td>0</td>
<td>0</td>
<td>9996</td>
<td>0</td>
<td>10510</td>
<td>10767</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>11538</td>
<td>-122</td>
<td>0</td>
<td>-124</td>
<td>0</td>
<td>0</td>
<td>-2</td>
</tr>
</tbody>
</table>

cword delimiter:

| 296926 | 477407 | 232160 | 182120 | 238245 | 265297 | 248914 | 462574 |
| 194048 | 526549 | 216150 | 199777 | 397410 | 444267 | 297964 | 625773 |
| 163405 | 167407 | 413168 | 462961 | 345158 | 509859 | 478708 | 247157 |
macro list:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

tabel:

```
8799488 8803584 8799496 8799556 8805552 8807698 8811776
8815872 8811784 8815880 8819976 8831064 8828160
8881071 8828168 8833264 8836352 8840448 8836360
8840456 8844552 8848640 8852736 8848648 8852744
8856980 8860928 8869120 8869936 8869128 8877320
8861114 8869376 8861192 8869384 8877576 8886016
8890934 8896024 8898512 8910600 8922684 8930816
8926532 8930624 8940016 8947200 8955392 8946952
8955514 8910088 8885760 8898048 8885768 8898056
8910944
```
### instruct list:

<table>
<thead>
<tr>
<th>Number</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>327680</td>
</tr>
<tr>
<td>1966080</td>
<td>0293760</td>
</tr>
<tr>
<td>3932160</td>
<td>4659840</td>
</tr>
<tr>
<td>5898240</td>
<td>6225920</td>
</tr>
<tr>
<td>7864320</td>
<td>7819200</td>
</tr>
<tr>
<td>9830400</td>
<td>10158560</td>
</tr>
<tr>
<td>11796480</td>
<td>12128160</td>
</tr>
<tr>
<td>13762560</td>
<td>14090240</td>
</tr>
<tr>
<td>15728640</td>
<td>16056320</td>
</tr>
<tr>
<td>17694720</td>
<td>18022400</td>
</tr>
<tr>
<td>19660800</td>
<td>20316160</td>
</tr>
<tr>
<td>21626880</td>
<td>21994560</td>
</tr>
<tr>
<td>23592960</td>
<td>242016320</td>
</tr>
<tr>
<td>25559040</td>
<td>26214400</td>
</tr>
<tr>
<td>27525120</td>
<td>28105600</td>
</tr>
<tr>
<td>29491200</td>
<td>30145600</td>
</tr>
<tr>
<td>31457280</td>
<td>32112680</td>
</tr>
<tr>
<td>33423360</td>
<td>34078720</td>
</tr>
<tr>
<td>35389440</td>
<td>36044800</td>
</tr>
<tr>
<td>37355520</td>
<td>38010880</td>
</tr>
<tr>
<td>39321600</td>
<td>39976960</td>
</tr>
<tr>
<td>41287680</td>
<td>41943040</td>
</tr>
<tr>
<td>43253760</td>
<td>43909120</td>
</tr>
<tr>
<td>45219840</td>
<td>45875200</td>
</tr>
<tr>
<td>47185920</td>
<td>47841280</td>
</tr>
<tr>
<td>49152000</td>
<td>49807360</td>
</tr>
<tr>
<td>51118080</td>
<td>51773920</td>
</tr>
<tr>
<td>53084160</td>
<td>53739520</td>
</tr>
<tr>
<td>55050240</td>
<td>55706080</td>
</tr>
<tr>
<td>57016320</td>
<td>57671840</td>
</tr>
<tr>
<td>58982400</td>
<td>59637760</td>
</tr>
<tr>
<td>60948480</td>
<td>61603840</td>
</tr>
<tr>
<td>62914560</td>
<td>63569920</td>
</tr>
<tr>
<td>64880640</td>
<td>65535000</td>
</tr>
</tbody>
</table>

### mask:

20 21 22 23 20 31 26 31 29

### end of memory: 4096

### wanted:

0
3. EXAMPLES

3.0 DESCRIPTION

In the following two examples the translation of an array declaration, a for statement and a (recursive) procedure declaration is demonstrated. On each left page the state of affairs after each scan is given as provided by the procedure output. The contents of the name list are given in octal notation, the other numbers being decimal. On each right page, opposite to the name list some indication about its structure is given (see also chapter 5), and opposite to the object program the macros, generated by the compiler and its corresponding ELAN instructions are added, followed by some explanatory comments. For a more detailed explanation one is referred to [3], sections 4.8, 5.5 and 6.
3.1 EXAMPLE 1

```c
begin integer i;
    real array A[1:10];
    for i := 1 step 1 until 10 do A[i] := i + 3.14
end
```

<table>
<thead>
<tr>
<th>erroneous</th>
<th>prescan0</th>
<th>prescan1</th>
<th>translate</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

| text length | 14 | 14 | 14 |

nemelist

```
0 +'000000007' +'000000007' +'000000007' 0
1 +'000120200' +'000120200' +'000120200' 1
2 +'000500000' +'000500000' +'000500000' 2
3 +'000000000' +'000000000' +'000000000' 3
4 -'200000006' -'200000006' -'200000006' 4
5 -'2000000024' -'2000000024' -'2000000024' 5
6 -'2000000024' -'2000000024' -'2000000024' 6
7 +'000000000' +'000000000' +'000000000' 7
8 +'0000000101' +'0000000101' +'0000000101' 8
9 +'000440000' +'000440000' +'000440000' 9
10 +'000000000' +'000000000' +'000000000' 10
11 -'2000000015' -'2000000015' -'2000000015' 11
12 -'2000000004' -'2000000004' -'2000000004' 12
13 -'000000023' -'000000023' -'000000023' 13
14 +'002000000' +'002000000' +'002000000' 14
15 -'0000000046' -'0000000046' -'0000000046' 15
16 +'020000000' +'020000000' +'020000000' 16
17 +'000000002' +'000000002' +'000000002' 17
18 +'020000000' +'020000000' +'020000000' 18
19 -'200000014' -'200000014' -'200000014' 19
```

dp0

```
2 2
```

start 7

program

```
0 +314  +314  +314  0
1 -2   -2   -2   1
   +0   +0   +0   2
   +0   +0   +0   3
   +0   +0   +0   4
   +0   +0   +0   5
   +0   +0   +0   6
```
0  + [ block head
1  +
2  +
3  +
4  +
5  +
6  - jump over block cell
7  + [ block head
8  +
9  +
10 - pointer to name cells
11 - name cell for i
12 +
13 - name cell for A[1:10]
14 +
15 + descriptor of pseudo for variable
16 +
17 [ constant
18 list
19 [ static
20 space
21 [
A = :MC
F = :MD[1]
B + 1
SUB2 (:ENTERB)
S = 2
linecounter = S
F = 1
MC = F
F = 10
MC = F
F = 1
S = 1
A = 6
SUB3 (:RAD)
M1 = B
S = 3
linecounter = S
F = :M29
F = 1
GOTO (:M33)
M28: F = 1
M29: G = M[5]
MC = F
DO (M28)
F + MC[-2]
N, SUBC (:RND)
M[5] = G
N, F = F, E
DO (M28)
N, F = F, Z
Y, GOTO (:M44)
GOTO (:M52)
G = M[5]
A = M[6]
MC = F
MC = A
G = M[5]
F × M[0]
SUB2 (:STSR)
GOTO (M4)
GOTO (:ENDRUI)

real array
A[1 : 10];

for i := 1
till 10

sum of maxima
3.2 EXAMPLE 2

\begin{verbatim}
begin comment Ackermann-function;
  integer procedure A(m,n); value m,n; integer m,n;
  A := if m=0 then n+1 else if n=0 then A(m-1,1) else A(m-1,A(m,n-1));
  integer t;
  t := A(4,0)
end
\end{verbatim}

<table>
<thead>
<tr>
<th>erroneous</th>
<th>prescan0</th>
<th>prescan1</th>
<th>translate</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>

| text length | 27 | 27 | 27 |

<table>
<thead>
<tr>
<th>namelist</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>23</td>
</tr>
<tr>
<td>24</td>
</tr>
<tr>
<td>25</td>
</tr>
<tr>
<td>26</td>
</tr>
<tr>
<td>27</td>
</tr>
<tr>
<td>28</td>
</tr>
<tr>
<td>29</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>31</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>33</td>
</tr>
</tbody>
</table>
0 + [block head
1 +
2 +
3 +
4 +
5 +
6 - jump over block cell
7 + [block head
8 +
9 +
10 + [pointer to name cells
11 +
12 +
13 +
14 + name cell for A
15 +
16 - jump over block cell
17 + [block head
18 +
19 +
20 +
21 +
22 +
23 + [name cell for m
24 +
25 + [name cell for \( n \)]
26 +
27 + [jump outside
28 + [pointer to name cells of formal identifiers
29 + [name cell for \( i \)]
30 +
31 +
32 +
33 +]
```
dp0

start

program

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
<td>2</td>
</tr>
<tr>
<td>163</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>164</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>165</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>166</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>154</td>
<td>64</td>
<td>7</td>
</tr>
<tr>
<td>167</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>168</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>169</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>165</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>65</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>65</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>170</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>171</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>172</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>198</td>
<td>0</td>
<td>17</td>
</tr>
<tr>
<td>199</td>
<td>3</td>
<td>32517</td>
</tr>
<tr>
<td>202</td>
<td>0</td>
<td>19</td>
</tr>
<tr>
<td>203</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>97</td>
<td>3</td>
<td>32515</td>
</tr>
<tr>
<td>119</td>
<td>0</td>
<td>22</td>
</tr>
<tr>
<td>157</td>
<td>0</td>
<td>23</td>
</tr>
<tr>
<td>97</td>
<td>3</td>
<td>32516</td>
</tr>
<tr>
<td>105</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>154</td>
<td>0</td>
<td>26</td>
</tr>
<tr>
<td>97</td>
<td>3</td>
<td>32516</td>
</tr>
<tr>
<td>119</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>157</td>
<td>0</td>
<td>29</td>
</tr>
<tr>
<td>154</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>46</td>
<td>0</td>
<td>31</td>
</tr>
<tr>
<td>97</td>
<td>3</td>
<td>32515</td>
</tr>
<tr>
<td>108</td>
<td>0</td>
<td>33</td>
</tr>
<tr>
<td>56</td>
<td>0</td>
<td>34</td>
</tr>
<tr>
<td>159</td>
<td>0</td>
<td>35</td>
</tr>
<tr>
<td>64</td>
<td>31</td>
<td>36</td>
</tr>
<tr>
<td>22</td>
<td>4</td>
<td>37</td>
</tr>
<tr>
<td>154</td>
<td>0</td>
<td>38</td>
</tr>
</tbody>
</table>
```
3 TBL (1)  A = ME
   4 ENTER (1)  F = MD[1]
   5 B + 1  SUB2 (:ENTER)
   6 SUB2 (:ENTER)
   7 JU (64)  GOTO (:M64)
   8 DPT (2)  S = D
   9 A = -O, Z
   10 SUB (:DPT)
   11 B + 1
   12 INC (1)  SUB (:CIV)
   13 CIV  SUB (:CIV)
   14 TDL (2)  F = MA[3]
   15 ENTRP (:3)  B + 3
   16 SUB (:ENTRP)
   17 SINC (:D,5)  S = linecounter
   18 MD[7] = S
   19 S - 7
   20 linecounter = S
   21 TIV (:D,3)  G = MD[3]
   22 EQUISIC (0)  F - 0, Z
   23 N, GOTO (:M27)
   24 TIV (:D,4)  G = MD[4]
   25 ADDSIC (1)  F + 1
   26 GOTO (:M57)
   27 TIV (:D,4)  G = MD[4]
   28 EQUISIC (0)  F - 0, Z
   29 N, GOTO (:M39)
   30 CJOY (:39)  GOTO (:M35)
   31 ENTERS  ISR31: SUB (:ENTER)
   32 TIV (:D,3)  G = MD[3]
   33 SUBSIC (:1)  F - 1
   34 EXITIS  GOTO (:EXIT)
   35 SUBJ (8)  G55: SUB (:M58)
   36 EPR  (20*20+ISR31)
   37 GOTO (:M57)
   38 JU (57)  (7*20+1)
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>JU (54)</td>
<td>M39: GOTO (:M54)</td>
</tr>
<tr>
<td>40</td>
<td>ENTRIS</td>
<td>ISR40: SUB2 (:ENTRIS)</td>
</tr>
<tr>
<td>41</td>
<td>TV (ID,3)</td>
<td>G = MD[3]</td>
</tr>
<tr>
<td>42</td>
<td>SUBSC (1)</td>
<td>F - 1</td>
</tr>
<tr>
<td>43</td>
<td>EXITIS</td>
<td>GOTO (:EXITIS)</td>
</tr>
<tr>
<td>44</td>
<td>ENTRIS</td>
<td>ISR44: SUB2 (:ENTRIS)</td>
</tr>
<tr>
<td>45</td>
<td>TV (ID,4)</td>
<td>GOTO (:M50)</td>
</tr>
<tr>
<td>46</td>
<td>EXITIS</td>
<td>ISR46: SUB2 (:ENTRIS)</td>
</tr>
<tr>
<td>47</td>
<td>SUBSC (1)</td>
<td>G = MD[4]</td>
</tr>
<tr>
<td>48</td>
<td>EXITIS</td>
<td>F - 1</td>
</tr>
<tr>
<td>49</td>
<td>SUB (8)</td>
<td>GOTO (:EXITIS)</td>
</tr>
<tr>
<td>50</td>
<td>apd</td>
<td>M50: SUBC (:M8)</td>
</tr>
<tr>
<td>51</td>
<td>apd</td>
<td>(1X220+418+[I,3])</td>
</tr>
<tr>
<td>52</td>
<td>apd</td>
<td>(20X420+ISR46)</td>
</tr>
<tr>
<td>53</td>
<td>EXITIS</td>
<td>GOTO (:EXITIS)</td>
</tr>
<tr>
<td>54</td>
<td>SUB (8)</td>
<td>M54: SUBC (:M8)</td>
</tr>
<tr>
<td>55</td>
<td>apd</td>
<td>(20X420+ISR40)</td>
</tr>
<tr>
<td>56</td>
<td>apd</td>
<td>(20X420+ISR44)</td>
</tr>
<tr>
<td>57</td>
<td>STI (ID,5)</td>
<td>M57: S = F, Z</td>
</tr>
<tr>
<td>58</td>
<td>N, SUBC (:NND)</td>
<td>A</td>
</tr>
<tr>
<td>59</td>
<td>MD[5] = G</td>
<td>A (m - 1, m, n - 1)</td>
</tr>
<tr>
<td>60</td>
<td>G = MD[5]</td>
<td>assign</td>
</tr>
<tr>
<td>61</td>
<td>S = MD[7]</td>
<td>to</td>
</tr>
<tr>
<td>62</td>
<td>linecounter = S</td>
<td>A</td>
</tr>
<tr>
<td>63</td>
<td>EXITP</td>
<td>M04: S = 9</td>
</tr>
<tr>
<td>64</td>
<td>LNC (9)</td>
<td>linecounter = G</td>
</tr>
<tr>
<td>65</td>
<td>SUBC (:M8)</td>
<td>SUBC (:M8)</td>
</tr>
<tr>
<td>66</td>
<td>(7X420+4)</td>
<td>(7X420+0)</td>
</tr>
<tr>
<td>67</td>
<td>(7X420+0)</td>
<td>S = F, Z</td>
</tr>
<tr>
<td>68</td>
<td>SUB (8)</td>
<td>N, SUBC (:NND)</td>
</tr>
<tr>
<td>69</td>
<td>apd</td>
<td>M(2) = G</td>
</tr>
<tr>
<td>70</td>
<td>apd</td>
<td>B = MD[1]</td>
</tr>
<tr>
<td>71</td>
<td>STI (2)</td>
<td>GOTO (:ENDBRK)</td>
</tr>
<tr>
<td>72</td>
<td>EKITB</td>
<td>8</td>
</tr>
<tr>
<td>73</td>
<td>END</td>
<td>else</td>
</tr>
<tr>
<td>74</td>
<td>8</td>
<td>ISR for</td>
</tr>
<tr>
<td></td>
<td></td>
<td>m - 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ISR for</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n - 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(m, n - 1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>assign</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(4, 0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>assign</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sum of maxima</td>
</tr>
</tbody>
</table>
4. TABLES

4.1 OPTIMIZED MACROS

In the translation of arithmetical operations certain sequences of two or three macros are replaced by one macro, a so-called *optimized macro*, as described in section 0.3. In the table below the names of these optimized macros can be found. The numbers between parentheses are the opt numbers (see section 4.2). E.g., the sequence of macros TRC, NEG is replaced by the optimized macro TRIC and the sequence of macros STACK, TSIC, MUL by MULSIC.

<table>
<thead>
<tr>
<th></th>
<th>TIV</th>
<th>TIV</th>
<th>TRC</th>
<th>TIC</th>
<th>TSIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEG</td>
<td>TRIV</td>
<td>TNIV</td>
<td>TRIC</td>
<td>TNIC</td>
<td>TNSIC</td>
</tr>
<tr>
<td>ADD</td>
<td>ADDR</td>
<td>ADDIV</td>
<td>ADDRC</td>
<td>ADDIC</td>
<td>ADDSIC</td>
</tr>
<tr>
<td>SUB</td>
<td>SUBR</td>
<td>SUBIV</td>
<td>SUBRC</td>
<td>SUBIC</td>
<td>SUBSIC</td>
</tr>
<tr>
<td>MUL</td>
<td>MULR</td>
<td>MULIV</td>
<td>MULRC</td>
<td>MULIC</td>
<td>MULSIC</td>
</tr>
<tr>
<td>DIV</td>
<td>DIVR</td>
<td>DIVIV</td>
<td>DIVRC</td>
<td>DIVIC</td>
<td>DIVSIC</td>
</tr>
<tr>
<td>EQU</td>
<td>EQUV</td>
<td>EQUIV</td>
<td>EQURC</td>
<td>EQUIC</td>
<td>EQUIC</td>
</tr>
<tr>
<td>UQU</td>
<td>UQUV</td>
<td>UQUIV</td>
<td>UQURC</td>
<td>UQUIC</td>
<td>UQUIC</td>
</tr>
<tr>
<td>LES</td>
<td>LESR</td>
<td>LESIV</td>
<td>LESRC</td>
<td>LESIC</td>
<td>LESIC</td>
</tr>
<tr>
<td>MST</td>
<td>MSTR</td>
<td>MSTIV</td>
<td>MSTRC</td>
<td>MSTIC</td>
<td>MSTIC</td>
</tr>
<tr>
<td>MDR</td>
<td>MDRV</td>
<td>MDRIV</td>
<td>MDRRC</td>
<td>MDRIC</td>
<td>MDRSIC</td>
</tr>
<tr>
<td>LST</td>
<td>LSTV</td>
<td>LSTIV</td>
<td>LSTRC</td>
<td>LSTIC</td>
<td>LSTSIC</td>
</tr>
</tbody>
</table>
4.2 MACRO DESCRIPTIONS

This table lists information on all macro identifiers and optimized macros used in the ALGOL 60 version of the compiler and on some additional macros used in the ELAN version for standard I/O routines.

On each left page is given

in column 1: the macro name,

2: the macro number (if any) used in the ELAN version. There a
    macro is not identified by its name, but by its number, which
    serves as an entry into the macro list,

3: the order number of the constituent instructions, followed,
    if relevant, by a code for the parameter type as follows:

<table>
<thead>
<tr>
<th>code</th>
<th>parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>pointer into the constant list</td>
</tr>
<tr>
<td>n</td>
<td>pointer into the name list, leading to an address in the object program</td>
</tr>
<tr>
<td>p</td>
<td>if value like then an address in the object program else n</td>
</tr>
<tr>
<td>v</td>
<td>value (small integer)</td>
</tr>
<tr>
<td>np</td>
<td>if parameter &gt; 0 then n else</td>
</tr>
<tr>
<td>=</td>
<td>instruction,</td>
</tr>
</tbody>
</table>

4: the corresponding ELAN instructions,

5: the meaning of the macro name.

On each right page the macro name is repeated, followed by the decoding of the value of the macro identifier or optimized macro. The coding is as follows:

\[
\text{value} = B \text{reaction} + d21 + \text{instr part} + d12 + \text{par part} + d10 + \\
\text{instr nbr} + d8 + \text{opt nbr} + d4 + \text{value like} + d3 + \\
\text{forw jump} + d2 + \text{opt op} + d1 + \text{sat macro}.
\]

\text{B reaction} is used in Process stack pointer (page 80) to estimate the amount by which the stack will grow during execution (see section 0.4).

\text{Instruct part} gives the key to the first instruction of the macro in the
instruction table (4.3). $Par\ part$ gives the order number of the instruction to which the macro parameter has to be added. $Instr\ number$ gives the number of consecutive instructions from the instruction table which constitute the macro. So for a macro the instructions in instruct list $[\text{instr part}]$ up to and including that in instruct list $[\text{instr part} + \text{instr nbr} - 1]$ are generated. $Opt\ number$ gives for optimizable operators and simple arithmetic take macros (see section 0.4) the table entries into the table of optimized macros (4.1). $Value\ like$ splits the parameters into two classes, which are treated separately in $Process\ parameter$ (page 81). A value like parameter can never be a pointer into the name list. $Forward\ jumping$ denotes whether Macro2 (page 78) possibly has to assign the value of $instr\ counter$ to its $metaparameter$ (cf. 0.4). Opt $op$ and sat macro are the abbreviations of $Optimizable\ operator$ and $Simple\ arithmetic\ take\ macro$ respectively.
<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Macro Nr.</th>
<th>Macro Instruction</th>
<th>Elan Instructions</th>
<th>Meaning of Macro Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>76</td>
<td></td>
<td>$F = F, P$</td>
<td>absolute value</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>$N, F = -P$</td>
<td></td>
</tr>
<tr>
<td>ABSFIXP</td>
<td>134</td>
<td>SUBC (:ABSFIXP)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>2</td>
<td></td>
<td>$F + M[\text{reg}]$</td>
<td>add</td>
</tr>
<tr>
<td>ADDIC</td>
<td>-1</td>
<td></td>
<td>$G + M[\text{reg}]$</td>
<td>add integer constant</td>
</tr>
<tr>
<td>ADDIV</td>
<td>-1</td>
<td></td>
<td>$G + M[p]</td>
<td>add integer variable</td>
</tr>
<tr>
<td>ADDR</td>
<td>-1</td>
<td></td>
<td>$F + M[\text{reg}]$</td>
<td>add real constant</td>
</tr>
<tr>
<td>ADDRV</td>
<td>-1</td>
<td></td>
<td>$F + M[p]</td>
<td>add real variable</td>
</tr>
<tr>
<td>ADDOSIC</td>
<td>-1</td>
<td></td>
<td>$F + M[\text{reg}]$</td>
<td>add small integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>constant</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>19</td>
<td></td>
<td>$N, B = 1$</td>
<td>logical 'and'</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>$Y, S = M[\text{reg}], P$</td>
<td></td>
</tr>
<tr>
<td>ARCTAN</td>
<td>131</td>
<td>SUBC (:ARCTAN)</td>
<td></td>
<td>arctangent</td>
</tr>
<tr>
<td>BAD</td>
<td>64</td>
<td>SUBB (:BAD)</td>
<td></td>
<td>boolean array</td>
</tr>
<tr>
<td></td>
<td></td>
<td>declaration</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>call boolean by value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBV</td>
<td>55</td>
<td>SUBC (:CBV)</td>
<td></td>
<td>call expression by name</td>
</tr>
<tr>
<td>CEN</td>
<td>58</td>
<td>SUB (:CEN)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CIV</td>
<td>54</td>
<td>SUBC (:CIV)</td>
<td></td>
<td>call integer by value</td>
</tr>
<tr>
<td>CLPN</td>
<td>59</td>
<td>SUBI (:CLPN)</td>
<td></td>
<td>call left part by name</td>
</tr>
<tr>
<td>CLV</td>
<td>57</td>
<td>SUBC (:CLV)</td>
<td></td>
<td>call label by value</td>
</tr>
<tr>
<td>CODE</td>
<td>128</td>
<td>=par</td>
<td></td>
<td>store par in object</td>
</tr>
<tr>
<td></td>
<td></td>
<td>code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COJU</td>
<td>106</td>
<td></td>
<td>$N, \text{GOTO} (:M[\text{reg}])$</td>
<td>conditional jump</td>
</tr>
<tr>
<td>COS</td>
<td>130</td>
<td>SUBC (:COS)</td>
<td></td>
<td>cosine</td>
</tr>
<tr>
<td>CRV</td>
<td>53</td>
<td>SUBC (:CRV)</td>
<td></td>
<td>call real by value</td>
</tr>
<tr>
<td>CSTV</td>
<td>56</td>
<td>SUBI (:CSTV)</td>
<td></td>
<td>call string by value</td>
</tr>
<tr>
<td>DECB</td>
<td>110</td>
<td></td>
<td>$B = -M[\text{reg}]$</td>
<td>decrease B</td>
</tr>
<tr>
<td>DECS</td>
<td>38</td>
<td></td>
<td>$S = 2$</td>
<td>decrease S</td>
</tr>
<tr>
<td>macro name</td>
<td>B</td>
<td>instr. react.</td>
<td>par. part</td>
<td>instr. part</td>
</tr>
<tr>
<td>------------</td>
<td>---</td>
<td>--------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>ABS</td>
<td>4</td>
<td>86</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>ABSFIXP</td>
<td>0</td>
<td>187</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ADD</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ADDIC</td>
<td>4</td>
<td>104</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDIV</td>
<td>4</td>
<td>104</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDRC</td>
<td>4</td>
<td>103</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDRV</td>
<td>4</td>
<td>103</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ADDSIC</td>
<td>4</td>
<td>105</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>AND</td>
<td>3</td>
<td>25</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>ARCTAN</td>
<td>4</td>
<td>184</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>BAD</td>
<td>10</td>
<td>75</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>CBV</td>
<td>5</td>
<td>66</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>CEN</td>
<td>6</td>
<td>69</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>CV</td>
<td>5</td>
<td>65</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>CLPN</td>
<td>8</td>
<td>70</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>CLV</td>
<td>6</td>
<td>68</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>CDE</td>
<td>4</td>
<td>173</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CDJU</td>
<td>4</td>
<td>157</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>COS</td>
<td>4</td>
<td>183</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>CRV</td>
<td>6</td>
<td>64</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>CSTV</td>
<td>6</td>
<td>67</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>DECB</td>
<td>4</td>
<td>161</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DECS</td>
<td>4</td>
<td>49</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>macro name</td>
<td>macro instr. nr.</td>
<td>mas instructions</td>
<td>meaning of macro name</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-----------------</td>
<td>-----------------</td>
<td>----------------------</td>
<td></td>
</tr>
<tr>
<td>DIV</td>
<td>5 1</td>
<td>MC = F</td>
<td>divide</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>F = MC[⁻²]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>F = MC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVIC</td>
<td>- 1 c</td>
<td>G / M[par]</td>
<td>divide by integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVIV</td>
<td>- 1 n</td>
<td>G / Mp[q]</td>
<td>divide by integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVRC</td>
<td>- 1 c</td>
<td>F / M[par]</td>
<td>divide by real</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVRV</td>
<td>- 1 n</td>
<td>F / Mp[q]</td>
<td>divide by real</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVSIC</td>
<td>- 1 v</td>
<td>F / :M[par]</td>
<td>divide by small</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>integer constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DO</td>
<td>111 1 p</td>
<td>DO (M[par])</td>
<td>do an instruction</td>
<td></td>
</tr>
<tr>
<td>DOS</td>
<td>99 1 n</td>
<td>DOS (Mp[q])</td>
<td>dos apic 0</td>
<td></td>
</tr>
<tr>
<td>DOS2</td>
<td>100 1 n</td>
<td>DOS (Mp[q + 2])</td>
<td>dos apic 2</td>
<td></td>
</tr>
<tr>
<td>DOS3</td>
<td>101 1 n</td>
<td>DOS (Mp[q + 3])</td>
<td>dos apic 3</td>
<td></td>
</tr>
<tr>
<td>DPTR</td>
<td>114 1 v</td>
<td>S = D</td>
<td>display transport</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>A = -(M[par] - 2), Z</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>SUB (:DPTR)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMPTY</td>
<td>75</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>END</td>
<td>82 1</td>
<td>GOTO (:END RUN)</td>
<td>end of program</td>
<td></td>
</tr>
<tr>
<td>ENTER</td>
<td>78 1</td>
<td>SUBC (:ENTER)</td>
<td>enter</td>
<td></td>
</tr>
<tr>
<td>ENTRB</td>
<td>113 1 v</td>
<td>B + :M[par]</td>
<td>enter block</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>SUB2 (:ENTRB)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENTRIS</td>
<td>35 1</td>
<td>SUB2 (:ENTRIS)</td>
<td>enter implicit</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>subroutine</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENTRPB</td>
<td>117 1 v</td>
<td>B + :M[par]</td>
<td>enter procedure body</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>SUB2 (:ENTRBP)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQU</td>
<td>8 1</td>
<td>F = MC[-2], Z</td>
<td>equal</td>
<td></td>
</tr>
<tr>
<td>EQUIC</td>
<td>- 1 c</td>
<td>G = M[par], Z</td>
<td>equal to integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQUIV</td>
<td>- 1 n</td>
<td>G = Mp[q], Z</td>
<td>equal to integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EQURC</td>
<td>- 1 c</td>
<td>F = M[par], Z</td>
<td>equal to real constant</td>
<td></td>
</tr>
<tr>
<td>EQURV</td>
<td>- 1 n</td>
<td>F = Mp[q], Z</td>
<td>equal to real variable</td>
<td></td>
</tr>
<tr>
<td>EQUISIC</td>
<td>- 1 v</td>
<td>F = :M[par], Z</td>
<td>equal to small integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td>macro name</td>
<td>B</td>
<td>instr. react.</td>
<td>B</td>
<td>instr. part</td>
</tr>
<tr>
<td>------------</td>
<td>---</td>
<td>--------------</td>
<td>---</td>
<td>-------------</td>
</tr>
<tr>
<td>DIV</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>DIVIC</td>
<td>4</td>
<td>113</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DIVIV</td>
<td>4</td>
<td>113</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DIVRC</td>
<td>4</td>
<td>112</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DIVRV</td>
<td>4</td>
<td>112</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DIVSIC</td>
<td>4</td>
<td>114</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DO</td>
<td>4</td>
<td>162</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DOS</td>
<td>4</td>
<td>151</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DOS2</td>
<td>4</td>
<td>152</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DOS3</td>
<td>4</td>
<td>153</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DPTR</td>
<td>14</td>
<td>167</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>EMPTY</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>END</td>
<td>4</td>
<td>95</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ENTER</td>
<td>4</td>
<td>91</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ENTRB</td>
<td>4</td>
<td>165</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>ENTRIS</td>
<td>12</td>
<td>46</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ENTRFB</td>
<td>15</td>
<td>171</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>EQU</td>
<td>2</td>
<td>8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>EQUI</td>
<td>4</td>
<td>117</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EQUII</td>
<td>4</td>
<td>117</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EQUIC</td>
<td>4</td>
<td>115</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EQRV</td>
<td>4</td>
<td>115</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EQRVIC</td>
<td>4</td>
<td>119</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>macro name</td>
<td>macro nr.</td>
<td>instr. nr.</td>
<td>elan instructions generated</td>
<td>meaning of macro name</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>------------</td>
<td>-----------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>EXIT</td>
<td>50</td>
<td>1</td>
<td>GOTO (M[−1])</td>
<td>exit</td>
</tr>
<tr>
<td>EXITB</td>
<td>125</td>
<td>1 v</td>
<td>B = M[M[par]]</td>
<td>exit block</td>
</tr>
<tr>
<td>EXITC</td>
<td>126</td>
<td>1 v</td>
<td>B = M[M[par]] SUBC (:FREE CHAIN)</td>
<td>exit block using counter stack</td>
</tr>
<tr>
<td>EXITIS</td>
<td>45</td>
<td>1</td>
<td>GOTO (:EXITIS)</td>
<td>exit implicit subroutine</td>
</tr>
<tr>
<td>EXITP</td>
<td>71</td>
<td>1</td>
<td>GOTO (:EXITP)</td>
<td>exit procedure</td>
</tr>
<tr>
<td>EXITPC</td>
<td>72</td>
<td>1</td>
<td>GOTO (:EXITPC)</td>
<td>exit procedure using counter stack</td>
</tr>
<tr>
<td>EXITSV</td>
<td>127</td>
<td>1 v</td>
<td>S = M[M[par]] GOTO (M[S])</td>
<td>exit subscripted variable</td>
</tr>
<tr>
<td>EXP</td>
<td>80</td>
<td>1</td>
<td>SUBC (:EXP)</td>
<td>exponential function</td>
</tr>
<tr>
<td>FAD</td>
<td>39</td>
<td>1</td>
<td>GOTO (:FAD)</td>
<td>finish address description</td>
</tr>
<tr>
<td>FADCV</td>
<td>46</td>
<td>1</td>
<td>GOTO (:FADCV)</td>
<td>finish addr descr of controlled variable</td>
</tr>
<tr>
<td>FIXP</td>
<td>133</td>
<td>1</td>
<td>SUBC (:FIXP)</td>
<td></td>
</tr>
<tr>
<td>FLOP</td>
<td>135</td>
<td>1</td>
<td>SUBC (:FLOP)</td>
<td></td>
</tr>
<tr>
<td>HAND</td>
<td>142</td>
<td>1</td>
<td>SUBC (:HAND)</td>
<td></td>
</tr>
<tr>
<td>IAD</td>
<td>63</td>
<td>1</td>
<td>SUB3 (:IAD)</td>
<td>integer array declaration</td>
</tr>
<tr>
<td>IDI</td>
<td>6</td>
<td>1</td>
<td>SUBC (:IDI)</td>
<td>integer division</td>
</tr>
<tr>
<td>LJU</td>
<td>104</td>
<td>1 n</td>
<td>GOTO (M[par])</td>
<td>indirect jump</td>
</tr>
<tr>
<td>LJU1</td>
<td>105</td>
<td>1 p</td>
<td>GOTO (M[par + 1])</td>
<td>indirect jump</td>
</tr>
<tr>
<td>IMP</td>
<td>17</td>
<td>1</td>
<td>Y, B = 1 N, S = −M[−1], P</td>
<td>logical 'implies'</td>
</tr>
<tr>
<td>INCRB</td>
<td>115</td>
<td>1 v</td>
<td>B + :M[par]</td>
<td>increase B</td>
</tr>
<tr>
<td>ISUBJ</td>
<td>109</td>
<td>1 p</td>
<td>SUBC (:M[par])</td>
<td>indirect subroutine jump</td>
</tr>
<tr>
<td>JU</td>
<td>102</td>
<td>1 np</td>
<td>GOTO (:M[par])</td>
<td>jump</td>
</tr>
<tr>
<td>JU1</td>
<td>103</td>
<td>1 p</td>
<td>A = :M[par] GOTO (:MA[1])</td>
<td>jump</td>
</tr>
<tr>
<td>JUA</td>
<td>74</td>
<td>1</td>
<td>GOTO (:JUA)</td>
<td>jump via A</td>
</tr>
<tr>
<td>macro name</td>
<td>B</td>
<td>instr. react.</td>
<td>part</td>
<td>par.</td>
</tr>
<tr>
<td>------------</td>
<td>---</td>
<td>--------------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>EXIT</td>
<td>4</td>
<td>61</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>EXITB</td>
<td>4</td>
<td>178</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>EXICT</td>
<td>4</td>
<td>178</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>EXITIS</td>
<td>13</td>
<td>56</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>EXITP</td>
<td>4</td>
<td>82</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>EXITPC</td>
<td>4</td>
<td>83</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>EXITSV</td>
<td>13</td>
<td>180</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>EXP</td>
<td>4</td>
<td>93</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>FAD</td>
<td>4</td>
<td>50</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>FADCV</td>
<td>4</td>
<td>57</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>FEXP</td>
<td>0</td>
<td>166</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>FLOP</td>
<td>0</td>
<td>188</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>HAND</td>
<td>4</td>
<td>195</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>IAD</td>
<td>10</td>
<td>74</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>IDI</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>IJU</td>
<td>4</td>
<td>155</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>IJU1</td>
<td>4</td>
<td>156</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>IMP</td>
<td>3</td>
<td>21</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>INCRB</td>
<td>4</td>
<td>165</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>ISUBJ</td>
<td>4</td>
<td>160</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>JU</td>
<td>4</td>
<td>154</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>JU1</td>
<td>4</td>
<td>143</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>JUA</td>
<td>4</td>
<td>85</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>macro name</td>
<td>macro instr.</td>
<td>elan instructions</td>
<td>meaning of macro name</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>-------------------</td>
<td>----------------------</td>
<td></td>
</tr>
<tr>
<td>LAD 120 1 v</td>
<td>S = '[Mpar]'</td>
<td>SUB (';LAD')</td>
<td>label declaration</td>
<td></td>
</tr>
<tr>
<td>LAST 119 1 p</td>
<td>-(A - M[par])</td>
<td></td>
<td>last element of label list</td>
<td></td>
</tr>
<tr>
<td>LES 10 1</td>
<td>F = MC[-2], P</td>
<td>less</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 X, F = 0, P</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LESIC - 1 c</td>
<td>G = M[par], P</td>
<td>less than integer constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 N, F = F, Z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 S = 'F', P</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LESIV - 1 n</td>
<td>G = Mp[q], P</td>
<td>less than integer variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 N, F = F, Z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 S = 'F', P</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LESRC - 1 c</td>
<td>F = M[par], P</td>
<td>less than real constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 N, F = F, Z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 S = 'F', P</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LESRV - 1 n</td>
<td>F = Mp[q], P</td>
<td>less than real variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 N, F = F, Z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 S = 'F', P</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LESSIC - 1 v</td>
<td>F = '[Mpar]'</td>
<td>less than small integer constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 N, F = F, Z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 S = 'F', P</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LN 81 1</td>
<td>SUBC (';LN')</td>
<td>natural logarithm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LNC 147 1 v</td>
<td>S = '[Mpar]'</td>
<td>set line counter</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 line counter = S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOS 70 1</td>
<td>loose string = B</td>
<td>loose string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LST 13 1</td>
<td>F = MC[-2], Z</td>
<td>at least</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 N, S = 'F', E</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSTIC - 1 c</td>
<td>G = M[par], P</td>
<td>at least integer constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 N, F = F, Z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSTIV - 1 n</td>
<td>G = Mp[q], P</td>
<td>at least integer variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 N, F = F, Z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSTRC - 1 c</td>
<td>F = M[par], P</td>
<td>at least real constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 N, F = F, Z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSTRV - 1 n</td>
<td>F =Mp[q], P</td>
<td>at least real variable</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 N, F = F, Z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>macro name</td>
<td>B</td>
<td>instr. part</td>
<td>par. part</td>
<td>instr. nбр.</td>
</tr>
<tr>
<td>------------</td>
<td>---</td>
<td>-------------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>LAD</td>
<td>4</td>
<td>175</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>LAST</td>
<td>4</td>
<td>174</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LES</td>
<td>2</td>
<td>10</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>LESIC</td>
<td>4</td>
<td>124</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LESIV</td>
<td>4</td>
<td>124</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LESRC</td>
<td>4</td>
<td>124</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LESRV</td>
<td>4</td>
<td>124</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LESSIC</td>
<td>4</td>
<td>127</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LN</td>
<td>4</td>
<td>94</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>LNC</td>
<td>4</td>
<td>202</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>LOS</td>
<td>4</td>
<td>81</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>LST</td>
<td>2</td>
<td>15</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>LSTIC</td>
<td>4</td>
<td>124</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LSTIV</td>
<td>4</td>
<td>124</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LSTRC</td>
<td>4</td>
<td>121</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LSTRV</td>
<td>4</td>
<td>121</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>macro name</td>
<td>macro nr.</td>
<td>instr. nr.</td>
<td>elan instructions generated</td>
<td>meaning of macro name</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>------------</td>
<td>-----------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>LSTSIC</td>
<td>– 1 v</td>
<td>F - :M[par], P</td>
<td>2 N, F = F, Z</td>
<td>at least small integer constant</td>
</tr>
<tr>
<td>MDR</td>
<td>12 1</td>
<td>F - M[-2], P</td>
<td>2 N, F = F, Z</td>
<td>more</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MDRIC</td>
<td>– 1 c</td>
<td>G - M[par], P</td>
<td>3 S = -T, P</td>
<td>more than integer constant</td>
</tr>
<tr>
<td>MDRIV</td>
<td>– 1 n</td>
<td>G - M[p], P</td>
<td>2 Y, F = 0, P</td>
<td>more than integer variable</td>
</tr>
<tr>
<td>MORRC</td>
<td>– 1 c</td>
<td>F - M[par], P</td>
<td></td>
<td>more than real constant</td>
</tr>
<tr>
<td>MORRV</td>
<td>– 1 n</td>
<td>F - M[p], P</td>
<td>2 Y, F = 0, P</td>
<td>more than real variable</td>
</tr>
<tr>
<td>MORSIC</td>
<td>– 1 v</td>
<td>F - :M[par], P</td>
<td></td>
<td>more than small integer constant</td>
</tr>
<tr>
<td>MST</td>
<td>11 1</td>
<td>F - M[-2], P</td>
<td>2 N, S = -1, E</td>
<td>at most integer constant</td>
</tr>
<tr>
<td>MSTIC</td>
<td>– 1 c</td>
<td>G - M[par], Z</td>
<td>2 N, F = F, Z</td>
<td>at most integer constant</td>
</tr>
<tr>
<td>MSTIV</td>
<td>– 1 n</td>
<td>G - M[p], Z</td>
<td>2 N, S = -1, E</td>
<td>at most integer variable</td>
</tr>
<tr>
<td>MSTRC</td>
<td>– 1 c</td>
<td>F - M[par], Z</td>
<td>2 N, S = -1, E</td>
<td>at most real constant</td>
</tr>
<tr>
<td>MSTRV</td>
<td>– 1 n</td>
<td>F - M[p], Z</td>
<td>2 N, S = -1, E</td>
<td>at most real variable</td>
</tr>
<tr>
<td>MTSIC</td>
<td>– 1 v</td>
<td>F - :M[par], Z</td>
<td>2 N, S = -1, E</td>
<td>at most small integer constant</td>
</tr>
<tr>
<td>MJL</td>
<td>4 1</td>
<td>F × M[-2]</td>
<td></td>
<td>multiply</td>
</tr>
<tr>
<td>MJULIC</td>
<td>– 1 c</td>
<td>G × M[par]</td>
<td></td>
<td>multiply by integer constant</td>
</tr>
<tr>
<td>MJULIV</td>
<td>– 1 n</td>
<td>G × M[p], Z</td>
<td></td>
<td>multiply by integer variable</td>
</tr>
<tr>
<td>MJULRC</td>
<td>– 1 c</td>
<td>F × M[par]</td>
<td></td>
<td>multiply by real constant</td>
</tr>
<tr>
<td>MJULRV</td>
<td>– 1 n</td>
<td>F × M[p], Z</td>
<td></td>
<td>multiply by real variable</td>
</tr>
<tr>
<td>MJLSIC</td>
<td>– 1 v</td>
<td>F × :M[par]</td>
<td></td>
<td>multiply by small integer constant</td>
</tr>
<tr>
<td>macro name</td>
<td>B</td>
<td>instr.</td>
<td>part react.</td>
<td>part</td>
</tr>
<tr>
<td>------------</td>
<td>---</td>
<td>--------</td>
<td>------------</td>
<td>------</td>
</tr>
<tr>
<td>LSTSCIC</td>
<td>4</td>
<td>127</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MOR</td>
<td>2</td>
<td>12</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>MOPIC</td>
<td>4</td>
<td>138</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MORIV</td>
<td>4</td>
<td>138</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MORBC</td>
<td>4</td>
<td>136</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MORRV</td>
<td>4</td>
<td>136</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MORSCIC</td>
<td>4</td>
<td>127</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MST</td>
<td>2</td>
<td>12</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>MSTIC</td>
<td>4</td>
<td>132</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MSTIV</td>
<td>4</td>
<td>132</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MSTRC</td>
<td>4</td>
<td>130</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MSTRVC</td>
<td>4</td>
<td>130</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MSTSIC</td>
<td>4</td>
<td>134</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>MUL</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>MULIC</td>
<td>4</td>
<td>110</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MULIV</td>
<td>4</td>
<td>110</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MULRC</td>
<td>4</td>
<td>109</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MULRV</td>
<td>4</td>
<td>109</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MULSIC</td>
<td>4</td>
<td>111</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Macro name</td>
<td>Macro nr.</td>
<td>Elan instructions</td>
<td>Meaning of macro name</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>-------------------</td>
<td>-----------------------</td>
<td></td>
</tr>
<tr>
<td>NSEG</td>
<td>1</td>
<td>( P = \overline{F} )</td>
<td>invert sign</td>
<td></td>
</tr>
<tr>
<td>NIL</td>
<td>118</td>
<td>( A + M[\text{par}] )</td>
<td>not last element in label list</td>
<td></td>
</tr>
<tr>
<td>NCON</td>
<td>15</td>
<td>( S = \overline{T}, P )</td>
<td>logical 'not'</td>
<td></td>
</tr>
<tr>
<td>OBAD</td>
<td>68</td>
<td>SUB3 (:OBAD)</td>
<td>own boolean array declaration</td>
<td></td>
</tr>
<tr>
<td>OIAD</td>
<td>67</td>
<td>SUB3 (:OIAD)</td>
<td>own integer array declaration</td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>18</td>
<td>( Y, B = 1 ) ( N, S = M[\text{-1}], P )</td>
<td>logical 'or'</td>
<td></td>
</tr>
<tr>
<td>ORAD</td>
<td>66</td>
<td>SUB3 (:ORAD)</td>
<td>own real array declaration</td>
<td></td>
</tr>
<tr>
<td>OSTDAD</td>
<td>69</td>
<td>SUB3 (:OSTAD)</td>
<td>own string array declaration</td>
<td></td>
</tr>
<tr>
<td>PUHEP</td>
<td>141</td>
<td>SUBC (:PUHEP)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUNCH</td>
<td>136</td>
<td>SUBC (:PUNCH)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUNLCR</td>
<td>137</td>
<td>SUBC (:PUNLCR)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PUSPACE</td>
<td>138</td>
<td>SUBC (:PUSPACE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QVL</td>
<td>16</td>
<td>( S = T ) ( S = M[\text{-1}], E )</td>
<td>logical 'equivalent'</td>
<td></td>
</tr>
<tr>
<td>RAD</td>
<td>62</td>
<td>SUB3 (:RAD)</td>
<td>real array declaration</td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td>132</td>
<td>SUBC (:READ)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REHEP</td>
<td>140</td>
<td>SUBC (:REHEP)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REJST</td>
<td>73</td>
<td>SUBC (:REJST)</td>
<td>reject string</td>
<td></td>
</tr>
<tr>
<td>RLNC</td>
<td>146</td>
<td>( S = M[p + 2] )</td>
<td>restore line counter</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{line counter} = S )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RUNCUT</td>
<td>139</td>
<td>SUBC (:RUNCUT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAS</td>
<td>37</td>
<td>stock 3 = S</td>
<td>save S</td>
<td></td>
</tr>
<tr>
<td>SIN</td>
<td>129</td>
<td>SUBC (:SIN)</td>
<td>sine</td>
<td></td>
</tr>
<tr>
<td>SIGN</td>
<td>77</td>
<td>( F = F, Z ) ( N, F = 1, E ) ( N, F = -1 )</td>
<td>sign</td>
<td></td>
</tr>
<tr>
<td>macro name</td>
<td>B</td>
<td>instr. react.</td>
<td>part</td>
<td>par.</td>
</tr>
<tr>
<td>------------</td>
<td>---</td>
<td>--------------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>NEJ</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NIL</td>
<td>4</td>
<td>173</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NON</td>
<td>4</td>
<td>9</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>GBAD</td>
<td>10</td>
<td>79</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>GIAAD</td>
<td>10</td>
<td>78</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>OR</td>
<td>3</td>
<td>23</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>GORAD</td>
<td>10</td>
<td>77</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>GSTDAD</td>
<td>10</td>
<td>80</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>PUNSEP</td>
<td>4</td>
<td>194</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>PUNCH</td>
<td>4</td>
<td>189</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>PUNLCR</td>
<td>4</td>
<td>190</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>PUSPACE</td>
<td>4</td>
<td>191</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>QVL</td>
<td>3</td>
<td>19</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>RAD</td>
<td>10</td>
<td>73</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>READ</td>
<td>4</td>
<td>185</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>RENSEP</td>
<td>4</td>
<td>193</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>REJST</td>
<td>4</td>
<td>84</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>RLNC</td>
<td>4</td>
<td>200</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>RUNOUT</td>
<td>4</td>
<td>192</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SAS</td>
<td>4</td>
<td>48</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SIN</td>
<td>4</td>
<td>182</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SIGN</td>
<td>4</td>
<td>88</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Macro Name</td>
<td>Macro Nr.</td>
<td>Instr. Nr.</td>
<td>Elan Instructions</td>
<td>Meaning of Macro Name</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>------------</td>
<td>-------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>SLNC</td>
<td>145</td>
<td>1</td>
<td>S = line counter</td>
<td>save line counter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Mp[q + 2] = S</td>
<td></td>
</tr>
<tr>
<td>SQRT</td>
<td>79</td>
<td>1</td>
<td>SUBC (:SQRT)</td>
<td>square root</td>
</tr>
<tr>
<td>SSTI</td>
<td>96</td>
<td>1</td>
<td>Mp[q] = G</td>
<td>simple store integer</td>
</tr>
<tr>
<td>SSTS1</td>
<td>31</td>
<td>1</td>
<td>SUB2 (:SSTS1)</td>
<td>simple store subscripted integer</td>
</tr>
<tr>
<td>STAAM</td>
<td>20</td>
<td>1</td>
<td>MC = A</td>
<td>stack A</td>
</tr>
<tr>
<td>STAB</td>
<td>14</td>
<td>1</td>
<td>S = T</td>
<td>stack boolean</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>MC = S</td>
<td></td>
</tr>
<tr>
<td>STAD</td>
<td>65</td>
<td>1</td>
<td>SUB3 (:STAD)</td>
<td>string array declaration</td>
</tr>
<tr>
<td>STACK</td>
<td>0</td>
<td>1</td>
<td>MC = F</td>
<td>stack F</td>
</tr>
<tr>
<td>STB</td>
<td>97</td>
<td>1</td>
<td>S = T</td>
<td>store boolean</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Mp[q] = S</td>
<td></td>
</tr>
<tr>
<td>STFSU</td>
<td>34</td>
<td>1</td>
<td>SUB2 (:STFSU)</td>
<td>store formal subscripted of unknown type</td>
</tr>
<tr>
<td>STI</td>
<td>95</td>
<td>1</td>
<td>S = F, Z</td>
<td>store integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>N, SUBC (:FND)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Mp[q] = G</td>
<td></td>
</tr>
<tr>
<td>STOP</td>
<td>144</td>
<td>1</td>
<td>SUBC (:STOP)</td>
<td></td>
</tr>
<tr>
<td>STR</td>
<td>94</td>
<td>1</td>
<td>Mp[q] = F</td>
<td>store real</td>
</tr>
<tr>
<td>STSB</td>
<td>32</td>
<td>1</td>
<td>SUB2 (:STSB)</td>
<td>store subscripted boolean</td>
</tr>
<tr>
<td>STSI</td>
<td>30</td>
<td>1</td>
<td>SUB2 (:STSI)</td>
<td>store subscripted integer</td>
</tr>
<tr>
<td>STSR</td>
<td>29</td>
<td>1</td>
<td>SUB2 (:STSR)</td>
<td>store subscripted real</td>
</tr>
<tr>
<td>STSST</td>
<td>33</td>
<td>1</td>
<td>SUB2 (:STSST)</td>
<td>store subscripted string</td>
</tr>
<tr>
<td>STST</td>
<td>96</td>
<td>1</td>
<td>F = Mp[q]</td>
<td>store string</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>SUB2 (:STST)</td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td>3</td>
<td>1</td>
<td>F = -F</td>
<td>subtract</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>F + MC[-2]</td>
<td></td>
</tr>
<tr>
<td>SUBIC</td>
<td>-</td>
<td>1</td>
<td>G = M[par]</td>
<td>subtract integer constant</td>
</tr>
<tr>
<td>SUBIV</td>
<td>-</td>
<td>1</td>
<td>G = Mp[q]</td>
<td>subtract integer variable</td>
</tr>
<tr>
<td>SUBJ</td>
<td>108</td>
<td>1</td>
<td>SUBC (:M[par])</td>
<td>subroutine jump</td>
</tr>
<tr>
<td>macro name</td>
<td>B</td>
<td>instr. req.</td>
<td>par. part</td>
<td>instr. part</td>
</tr>
<tr>
<td>------------</td>
<td>---</td>
<td>-----------</td>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>SMLC</td>
<td>4</td>
<td>198</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>SQRT</td>
<td>4</td>
<td>92</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SSTI</td>
<td>4</td>
<td>148</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SSTSI</td>
<td>10</td>
<td>42</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>STAA</td>
<td>5</td>
<td>27</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>STAB</td>
<td>5</td>
<td>17</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>STAD</td>
<td>10</td>
<td>76</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>STACK</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>STB</td>
<td>4</td>
<td>14</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>STFSU</td>
<td>10</td>
<td>45</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>STI</td>
<td>4</td>
<td>146</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>STDP</td>
<td>4</td>
<td>197</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>STR</td>
<td>4</td>
<td>145</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>STSB</td>
<td>10</td>
<td>43</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>STSI</td>
<td>10</td>
<td>41</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>STSR</td>
<td>10</td>
<td>40</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>STSST</td>
<td>10</td>
<td>44</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>STSST</td>
<td>4</td>
<td>96</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>SUB</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>SUBIC</td>
<td>4</td>
<td>107</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SUBIV</td>
<td>4</td>
<td>107</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SUBJ</td>
<td>4</td>
<td>159</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>macro name</td>
<td>macro nr.</td>
<td>instr. nr.</td>
<td>elan instructions generated</td>
<td>meaning of macro name</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>------------</td>
<td>------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>SUBRC</td>
<td>- 1 c</td>
<td></td>
<td>$F - M[\text{par}]$</td>
<td>subtract real constant</td>
</tr>
<tr>
<td>SUBRV</td>
<td>- 1 n</td>
<td></td>
<td>$F - M[pq]$</td>
<td>subtract real variable</td>
</tr>
<tr>
<td>SUBSIC</td>
<td>- 1 v</td>
<td></td>
<td>$F - M[\text{par}]$</td>
<td>subtract small integer constant</td>
</tr>
<tr>
<td>SWP</td>
<td>124 1 v</td>
<td></td>
<td>$M[p] = B$</td>
<td>set working space pointer</td>
</tr>
<tr>
<td>TAA</td>
<td>123 1 n</td>
<td></td>
<td>$A = M[pq]$</td>
<td>take address of array reference</td>
</tr>
<tr>
<td>TAK</td>
<td>92 1 n</td>
<td></td>
<td>$A = M[pq]$</td>
<td>take array key</td>
</tr>
<tr>
<td>TASSB</td>
<td>42 1</td>
<td></td>
<td>GOTO (:TASSB)</td>
<td>take actual subscripted boolean</td>
</tr>
<tr>
<td>TASI</td>
<td>41 1</td>
<td></td>
<td>GOTO (:TASI)</td>
<td>take actual subscripted integer</td>
</tr>
<tr>
<td>TASR</td>
<td>40 1</td>
<td></td>
<td>GOTO (:TASR)</td>
<td>take actual subscripted real</td>
</tr>
<tr>
<td>TASST</td>
<td>43 1</td>
<td></td>
<td>GOTO (:TASST)</td>
<td>take actual subscripted string</td>
</tr>
<tr>
<td>TASU</td>
<td>44 1</td>
<td></td>
<td>GOTO (:TASU)</td>
<td>take actual subscripted of unknown type</td>
</tr>
<tr>
<td>TAV</td>
<td>60 1</td>
<td></td>
<td>SUB4 (:TAV)</td>
<td>transport array called by value</td>
</tr>
<tr>
<td>TBC</td>
<td>89 1 v</td>
<td></td>
<td>$S = M[\text{par}]$, $Z$</td>
<td>take boolean constant</td>
</tr>
<tr>
<td>TBL</td>
<td>112 1</td>
<td>2 v</td>
<td>$A = M_C$</td>
<td>take block level</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$F = M_D[\text{par}]$</td>
<td></td>
</tr>
<tr>
<td>TBV</td>
<td>88 1 n</td>
<td></td>
<td>$S = M[pq]$, $P$</td>
<td>take boolean variable</td>
</tr>
<tr>
<td>TCST</td>
<td>28 1</td>
<td></td>
<td>SUBC (:TCST)</td>
<td>take constant string</td>
</tr>
<tr>
<td>TDA</td>
<td>121 1 v</td>
<td></td>
<td>$S = M[\text{par}]$</td>
<td>take dimension of array</td>
</tr>
<tr>
<td>TDL</td>
<td>116 1 v</td>
<td></td>
<td>$F = M_A[\text{par}]$</td>
<td>take display level</td>
</tr>
<tr>
<td>TEST1</td>
<td>51 1</td>
<td></td>
<td>$F = M[C-2]$, $Z$</td>
<td>test for list element exhausted</td>
</tr>
<tr>
<td>TEST2</td>
<td>52 1</td>
<td>2 v</td>
<td>$N, F = F, E$</td>
<td>test for list element exhausted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$N, F = F, Z$</td>
<td></td>
</tr>
<tr>
<td>TFD</td>
<td>36 1</td>
<td></td>
<td>$S = M_S[1]$</td>
<td>take formal display pointer</td>
</tr>
<tr>
<td>TPSL</td>
<td>27 1</td>
<td></td>
<td>SUBC (:TPSL)</td>
<td>take formal subscripted label</td>
</tr>
<tr>
<td>TPSU</td>
<td>25 1</td>
<td></td>
<td>SUB2 (:TPSU)</td>
<td>take formal subscripted of unknown type</td>
</tr>
<tr>
<td>TIAV</td>
<td>61 1</td>
<td></td>
<td>SUB4 (:TIAV)</td>
<td>transport integer array called by value</td>
</tr>
<tr>
<td>macro name</td>
<td>B</td>
<td>instr. react.</td>
<td>par. part</td>
<td>instr. part</td>
</tr>
<tr>
<td>------------</td>
<td>---</td>
<td>--------------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>SUBRC</td>
<td>4</td>
<td>106</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SUBRV</td>
<td>4</td>
<td>106</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SUBSIC</td>
<td>4</td>
<td>108</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SWP</td>
<td>4</td>
<td>177</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TAA</td>
<td>4</td>
<td>143</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TAK</td>
<td>4</td>
<td>142</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TASB</td>
<td>4</td>
<td>53</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TASI</td>
<td>4</td>
<td>52</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TASR</td>
<td>4</td>
<td>51</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TASST</td>
<td>4</td>
<td>54</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TASI</td>
<td>4</td>
<td>55</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TAV</td>
<td>4</td>
<td>71</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TBC</td>
<td>4</td>
<td>141</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TBL</td>
<td>4</td>
<td>163</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>TBE</td>
<td>4</td>
<td>140</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TCHT</td>
<td>4</td>
<td>39</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TDA</td>
<td>4</td>
<td>175</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TDL</td>
<td>4</td>
<td>170</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TESI</td>
<td>2</td>
<td>8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TEST2</td>
<td>4</td>
<td>62</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>TFD</td>
<td>4</td>
<td>47</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TFSI</td>
<td>4</td>
<td>38</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TFSI</td>
<td>11</td>
<td>36</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TIAS</td>
<td>4</td>
<td>72</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Macro name</td>
<td>Macro nr.</td>
<td>Instr. nr.</td>
<td>Elan instructions generated</td>
<td>Meaning of macro name</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>------------</td>
<td>----------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>TIC</td>
<td>86</td>
<td>1</td>
<td>$G = M[par]$</td>
<td>take integer constant</td>
</tr>
<tr>
<td>TISCV</td>
<td>48</td>
<td>1</td>
<td>GOTO (:TISCV)</td>
<td>take integer subscripted controlled variable</td>
</tr>
<tr>
<td>TIV</td>
<td>84</td>
<td>1</td>
<td>$G = M[p][q]$</td>
<td>take integer variable</td>
</tr>
<tr>
<td>TLV</td>
<td>91</td>
<td>1</td>
<td>$A = M[p][q]$</td>
<td>take label variable</td>
</tr>
<tr>
<td>TNA</td>
<td>122</td>
<td>1</td>
<td>$F = M[par]$</td>
<td>take number of arrays</td>
</tr>
<tr>
<td>TNIC</td>
<td>1</td>
<td>1</td>
<td>G = $\neg M[par]$</td>
<td>take negative integer constant</td>
</tr>
<tr>
<td>TNIV</td>
<td>1</td>
<td>1</td>
<td>G = $\neg M[p][q]$</td>
<td>take negative integer variable</td>
</tr>
<tr>
<td>TNRC</td>
<td>1</td>
<td>1</td>
<td>F = $\neg M[par]$</td>
<td>take negative real constant</td>
</tr>
<tr>
<td>TNRV</td>
<td>1</td>
<td>1</td>
<td>F = $\neg M[p][q]$</td>
<td>take negative real variable</td>
</tr>
<tr>
<td>TNSIC</td>
<td>1</td>
<td>1</td>
<td>F = $\neg M[par]$</td>
<td>take negative small integer constant</td>
</tr>
<tr>
<td>TSC</td>
<td>85</td>
<td>1</td>
<td>F = $M[par]$</td>
<td>take real constant</td>
</tr>
<tr>
<td>TRSCV</td>
<td>47</td>
<td>1</td>
<td>GOTO (:TRSCV)</td>
<td>take real subscripted controlled variable</td>
</tr>
<tr>
<td>TTV</td>
<td>83</td>
<td>1</td>
<td>$F = M[p][q]$</td>
<td>take real variable</td>
</tr>
<tr>
<td>TSB</td>
<td>23</td>
<td>1</td>
<td>SUB1 (:INDB)</td>
<td>take subscripted boolean</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>S 'x', MA, 2</td>
<td></td>
</tr>
<tr>
<td>TSCVU</td>
<td>49</td>
<td>1</td>
<td>GOTO (:TSCVU)</td>
<td>take subscripted variable of unknown type</td>
</tr>
<tr>
<td>TSI</td>
<td>22</td>
<td>1</td>
<td>SUB (:IND)</td>
<td>take subscripted integer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>G = MA</td>
<td></td>
</tr>
<tr>
<td>TSIC</td>
<td>87</td>
<td>1</td>
<td>$F = M[par]$</td>
<td>take small integer constant</td>
</tr>
<tr>
<td>TSL</td>
<td>26</td>
<td>1</td>
<td>SUBC (:TSL)</td>
<td>take subscripted label</td>
</tr>
<tr>
<td>TSR</td>
<td>21</td>
<td>1</td>
<td>SUB (:IND)</td>
<td>take subscripted real</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>F = MA</td>
<td></td>
</tr>
<tr>
<td>TSST</td>
<td>24</td>
<td>1</td>
<td>SUB (:IND)</td>
<td>take subscripted string</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>A = MA</td>
<td></td>
</tr>
<tr>
<td>TSTV</td>
<td>90</td>
<td>1</td>
<td>$A = M[p][q]$</td>
<td>take string variable</td>
</tr>
<tr>
<td>TSWE</td>
<td>93</td>
<td>1</td>
<td>$A = M[par]$</td>
<td>take switch entry</td>
</tr>
<tr>
<td>TTP</td>
<td>7</td>
<td>1</td>
<td>SUBC (:TTP)</td>
<td>to the power</td>
</tr>
<tr>
<td>macro</td>
<td>B</td>
<td>instr.</td>
<td>par.</td>
<td>instr.</td>
</tr>
<tr>
<td>-------</td>
<td>---</td>
<td>--------</td>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td>name</td>
<td>4</td>
<td>react.</td>
<td>part</td>
<td>part</td>
</tr>
<tr>
<td>TIC</td>
<td>4</td>
<td>97</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TISCV</td>
<td>4</td>
<td>59</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TIV</td>
<td>4</td>
<td>97</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TLV</td>
<td>4</td>
<td>143</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TWA</td>
<td>4</td>
<td>98</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TMIC</td>
<td>4</td>
<td>101</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TNIV</td>
<td>4</td>
<td>101</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TNRC</td>
<td>4</td>
<td>100</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TNRV</td>
<td>4</td>
<td>100</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TNSIC</td>
<td>4</td>
<td>102</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TRC</td>
<td>4</td>
<td>96</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TRSCV</td>
<td>4</td>
<td>58</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TRV</td>
<td>4</td>
<td>96</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TSB</td>
<td>11</td>
<td>32</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>TSCVU</td>
<td>4</td>
<td>60</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TSI</td>
<td>11</td>
<td>30</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>TSIC</td>
<td>4</td>
<td>98</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TSL</td>
<td>4</td>
<td>37</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TSB</td>
<td>11</td>
<td>28</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>TSST</td>
<td>11</td>
<td>34</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>TSTV</td>
<td>4</td>
<td>142</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TSWE</td>
<td>4</td>
<td>143</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TTP</td>
<td>2</td>
<td>7</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Macro Name</td>
<td>Macro Instr.</td>
<td>Elan Instructions</td>
<td>Meaning of Macro Name</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>-------------------</td>
<td>-----------------------</td>
<td></td>
</tr>
<tr>
<td>UQU</td>
<td>9 1</td>
<td>F = M[-3], Z</td>
<td>unequal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>S = -T , P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UQUIC</td>
<td>- 1 c</td>
<td>G = M[par], Z</td>
<td>unequal to integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>S = -T , P</td>
<td>constant</td>
<td></td>
</tr>
<tr>
<td>UQUIV</td>
<td>- 1 n</td>
<td>G = M[pq], Z</td>
<td>unequal to integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>S = -T , P</td>
<td>variable</td>
<td></td>
</tr>
<tr>
<td>UQURC</td>
<td>- 1 c</td>
<td>F = M[par], Z</td>
<td>unequal to real</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>S = -T , P</td>
<td>constant</td>
<td></td>
</tr>
<tr>
<td>UQRV</td>
<td>- 1 n</td>
<td>F = M[pq], Z</td>
<td>unequal to real</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>S = -T , P</td>
<td>variable</td>
<td></td>
</tr>
<tr>
<td>UQUSIC</td>
<td>- 1 v</td>
<td>F = M[par], Z</td>
<td>unequal to small</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>S = -T , P</td>
<td>integer constant</td>
<td></td>
</tr>
<tr>
<td>XEEN</td>
<td>143 1</td>
<td>SUBC (:XEEN)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>YCOJU</td>
<td>107 1 p</td>
<td>Y, GOTO (:M[par])</td>
<td>yes-conditional jump</td>
<td></td>
</tr>
<tr>
<td>macro name</td>
<td>B</td>
<td>instr react.</td>
<td>par. part</td>
<td>instr part</td>
</tr>
<tr>
<td>------------</td>
<td>---</td>
<td>--------------</td>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>UQJ</td>
<td>2</td>
<td>8</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>UQJIC</td>
<td>4</td>
<td>117</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>UQJIV</td>
<td>4</td>
<td>117</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>UQJRC</td>
<td>4</td>
<td>115</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>UQJRV</td>
<td>4</td>
<td>115</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>UQJUSIC</td>
<td>4</td>
<td>119</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>XQEN</td>
<td>4</td>
<td>196</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>YQJU</td>
<td>4</td>
<td>158</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
4.3 INSTRUCTION TABLE

The instruction table given below is taken from the ELAN version of the compiler. It contains all machine instructions needed by the macro processor (in particular by Produce (page 79)) to build up the object program from the macros generated by the compiler.
INSTR LST[0]:

MC = F  " DIV, STACK
F = MC[-1]
F / MC
F = - F
F + MC[-2]
F × MC[-2]
SUBC (:IDI)
SUBC (:TTP)
F - MC[-2], Z
S = T, P
F - MC[-2], P
Y, F = 0, P
F = MC[-2], P
N, F = F, Z
S = T, P
F - MC[-2], Z
N, S = -1, E
S = T
MC = S
S = T, P
S = MC[-1], E
Y, B = 1
N, S = - MC[-1], P
Y, B = 1
N, S = MC[-1], P
Y, B = 1
N, S = MC[-1], P
MC = A
SUB (:IND)
" STAA
F = MA
SUB (:IND)
G = MA
SUB1 (:INDB)
S 'x' MA, Z
SUB (:IND)
" STS
A = MA
SUB2 (:TSU)
" TFSU
SUBC (:TSL)
" TSU
SUBC (:TFSL)
" TFSU
SUBC (:TST)
" TST
SUB2 (:STS)
" STSR
SUB2 (:STSI)
" STSI
SUB2 (:STSII)
" STSII
SUB2 (:STSB)
" STSB
SUB2 (:STSSST)
" STSSST
SUB2 (:STSSS)
" STSSS
SUB2 (:STSSU)
" STSSU
SUB2 (:ENTRIS)
" ENTRIS
S = MS[1]
" TFD
stock3 = S
S - 2
" DECS
INSTR LIST[50]:

GOTO (:FAD)  " FAD
GOTO (:TASR)  " TASR
GOTO (:TASI)  " TASI
GOTO (:TASB)  " TASB
GOTO (:TASS)  " TASS
GOTO (:TASU)  " TASU
GOTO (:EXITIS)  " EXITIS
GOTO (:FADC)  " FADC
GOTO (:TISD)  " TRSCV
GOTO (:TISC)  " TISC
GOTO (:TSCV)  " TSCV

GOTO (:TSCV)  " TSCV
GOTO (:EXIT)  " EXIT
N, F = F, E  " TEST2
N, F = F, Z
SUBC (:Crv)  " CRV
SUBC (:CIV)  " CIV
SUBC (:CBE)  " CBE
SUB1 (:CSTY)  " CSTY
SUBC (:CLY)  " CLY
SUB (:CEN)  " CEN

SUB1 (:CLPN)  " CLPN
SUB4 (:TAV)  " TAV
SUB1 (:TAY)  " TAY
SUB1 (:RAD)  " RAD
SUB3 (:TAD)  " IAD
SUB3 (:BAD)  " BAD
SUB3 (:STD)  " STD
SUB3 (:ORAD)  " ORAD
SUB3 (:GIA)  " GIAD
SUB3 (:GRAD)  " GBAD

SUB3 (:OSTAD)  " OSTD
loose string = B  " LG8
GOTO (:EXITF)  " EXITF
GOTO (:EXITFC)  " EXITFC
SUBC (:NOBST)  " NOBST
GOTO (:JUA)  " JUA
F = F, P  " ABS
N, F = F
F = F, Z  " SIGN
N, F = 1, E

N, F = 1
SUBC (:ENTER)  " ENTER
SUBC (:SQRT)  " SQRT
SUBC (:EXP)  " EXP
SUBC (:IN)  " LN
GOTO (:END RUN)  " END
F = M[0]  " TV, TC
G = M[0]  " TV, TC
F = 0  " TSIC, TNA, STST
SUB2 (:STST)
INSTR LST[100]:

F = - M[0]  " TNRV, TNRC
G = - M[0]  " TNIV, TNIC
F = - O     " TNSIC
F + M[0]    " ADDRV, ADDRC
G + M[0]    " ADDIV, ADDIC
F + O       " ADDIC
F - M[0]    " SUBRV, SUBRC
G - M[0]    " SUBIV, SUBIC
F - O       " SUBIC
F x M[0]    " MULRV, MULRC

G x M[0]    " MULIV, MULIC
G x O       " MULIC
F / M[0]    " DIVRV, DIVRC
G / M[0]    " DIVIV, DIVIC
F / O       " DIVIC
F - M[0], Z " EQURV, EQURC, UQRV, UQRC
S = T, P    " EQUIC, UQVIC
S = -T, P   " EQVIC, UQVIC
F - O, Z    " EQJSIC, USIC

S = -T, P   " LSVRV, LSVRC, LSVIV, LSVIC
F, F = F, Z " LSVIC
S = -T, P   " LSVIC
G - M[0], P " LSVIC, LSTIC, LSTRC
N, F = F, Z " LSVIC, LSTIC
S = -T, P   " LSVIC, LSTIC, LSTIC
F - O, P    " LSSIC, LSTIC, MIRCIC
N, F = F, Z " LSSIC, LSTIC, MIRCIC
S = -T, P   " LSSIC, LSTIC, MIRCIC

F - M[0], Z " MSTRV, MSTRC
N, S = -1, E " MSTIV, MSTIC
N, S = -1, E " MSTIC
F - O, Z    " MSTIC
N, S = -1, E " MSTIV, MSTIC
F - M[0], P " MSTIV, MSTIC
Y, F = -O, P " MSTIV, MIRCIC
Y, F = -O, P " MSTIV, MIRCIC

S = M[0], P " TBV
S = O, Z   " TBC
A = M[0]   " TSIV, TAK
A = O      " TSLV, TSW, TAA, JUI
GOTO (:IMA[1])
M[0] = F   " STR
S = F, Z   " STI
N, SUBC (:IND)
M[0] = G   " SSTIT
S = T      " STB
INSTR LIST(150):

M[0] = S
DOS (M[0])
DOS (M[2])
DOS (M[3])
GOTO (:M[0])
GOTO (M[0])
GOTO (M[1])
N, GOTO (:M[0])
Y, GOTO (:M[0])
SUBC (:M[0])

SUBC (M[0])
B = 0
DO (M[0])
A = :MC
F = :MD[0]
B = 0
SUBS (:ENTRB)
S = D
'0327 77776'
SUB (:DPTR)

F = :MA[0]
B = 0
ENTRB
SUBS (:ENTRBP)
A + M[0]
A - M[0]
S = 0
SUB (:LAD)
MD[0] = B
SUB (:FREE CHAIN)

S = :MC[0]
GOTO (ME)
SUBC (:SIN)
SUBC (:COS)
SUBC (:ATAN)
SUBC (:READ)
SUBC (:FIXP)
SUBC (:ABSFIXP)
SUBC (:FLOP)
SUBC (:PUNCH)

SUBC (:PUNCH)
SUBC (:PUSH)
SUBC (:RUNOUT)
SUBC (:REHSP)
SUBC (:PUSH)
SUBC (:HAND)
SUBC (:XHSP)
SUBC (:STOP)
S = line counter
M[2] = S
INSTR LST[200]:  
S = M[2]  " RLNC
line counter = S
S = 0  " LNC
line counter = S
4.4 ACTUAL PARAMETER DESCRIPTORS

In the object program every actual parameter is characterized by an Actual Parameter Descriptor. These APD's directly follow the procedure call, one for each parameter, in the order of the actual parameter list. Each APD uses one 27-bits word (d26, d25, ..., d0) as follows:

- d0 - d14 contain an address or a value,
- d15 - d17 0,
- d18 is 1 when the address in d0 - d14 is a dynamic one, else 0,
- d19 0,
- d20 - d25 contain a number (s32) indicating the type of the actual parameter,
- d26 0.

Some types of actual parameters are too complicated to be described fully by an APD. For these parameters there also exists an implicit subroutine (ISR), that is a piece of object program, preceding the procedure call. The address in the APD then gives the (start) address of the ISR. Again the order of the ISR's is that of the actual parameter list. If there is at least one ISR, the first of them is preceded by a "jump over the implicit subroutines", that is a jump instruction leading directly to the procedure call.

In the following table the correspondence between the type of an actual parameter and its APD is given.
<table>
<thead>
<tr>
<th>actual parameter</th>
<th>AFD</th>
</tr>
</thead>
<tbody>
<tr>
<td>real variable identifier (r)</td>
<td>( (0 \times d20 + \text{address } r) )</td>
</tr>
<tr>
<td>integer variable identifier (i)</td>
<td>( (1 \times d20 + \text{address } i) )</td>
</tr>
<tr>
<td>Boolean variable identifier (b)</td>
<td>( (2 \times d20 + \text{address } b) )</td>
</tr>
<tr>
<td>string variable identifier (st)</td>
<td>( (3 \times d20 + \text{address } st) )</td>
</tr>
<tr>
<td>floating point constant (fp)</td>
<td>( (4 \times d20 + \text{address } fp) )</td>
</tr>
<tr>
<td>integer constant (n)</td>
<td>( (5 \times d20 + \text{address } n) )</td>
</tr>
<tr>
<td>logical value (lv)</td>
<td>( (6 \times d20 + \text{if } lv \text{ then 0 else 1}) )</td>
</tr>
<tr>
<td>small positive integer (s)</td>
<td>( (7 \times d20 + s) )</td>
</tr>
<tr>
<td>real array identifier (ar)</td>
<td>( (8 \times d20 + \text{address } ar) )</td>
</tr>
<tr>
<td>integer array identifier (ai)</td>
<td>( (9 \times d20 + \text{address } ai) )</td>
</tr>
<tr>
<td>Boolean array identifier (ab)</td>
<td>( (10 \times d20 + \text{address } ab) )</td>
</tr>
<tr>
<td>string array identifier (ast)</td>
<td>( (11 \times d20 + \text{address } ast) )</td>
</tr>
<tr>
<td>switch identifier (sw)</td>
<td>( (12 \times d20 + \text{address } sw) )</td>
</tr>
<tr>
<td>small negative integer (-s)</td>
<td>( (13 \times d20 + s) )</td>
</tr>
<tr>
<td>label identifier (lb)</td>
<td>( (14 \times d20 + \text{address } lb) )</td>
</tr>
<tr>
<td>formal identifier (f)</td>
<td>( (15 \times d20 + \text{address } f) )</td>
</tr>
<tr>
<td>actual parameter</td>
<td>APL</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>real array element</td>
<td>((16 \times d20 + \text{address ISR}))</td>
</tr>
<tr>
<td>integer array element</td>
<td>((17 \times d20 + \text{address ISR}))</td>
</tr>
<tr>
<td>Boolean array element</td>
<td>((18 \times d20 + \text{address ISR}))</td>
</tr>
<tr>
<td>string array element</td>
<td>((19 \times d20 + \text{address ISR}))</td>
</tr>
<tr>
<td>not assignable arithmetic expression</td>
<td>((20 \times d20 + \text{address ISR}))</td>
</tr>
<tr>
<td>not assignable Boolean expression</td>
<td>((22 \times d20 + \text{address ISR}))</td>
</tr>
<tr>
<td>not assignable string expression</td>
<td>((23 \times d20 + \text{address ISR}))</td>
</tr>
<tr>
<td>real procedure identifier</td>
<td>((24 \times d20 - \text{address ISR}))</td>
</tr>
<tr>
<td>integer procedure identifier</td>
<td>((25 \times d20 - \text{address ISR}))</td>
</tr>
<tr>
<td>Boolean procedure identifier</td>
<td>((26 \times d20 - \text{address ISR}))</td>
</tr>
<tr>
<td>string procedure identifier</td>
<td>((27 \times d20 - \text{address ISR}))</td>
</tr>
<tr>
<td>designational expression</td>
<td>((28 \times d20 + \text{address ISR}))</td>
</tr>
<tr>
<td>integer constant or designational expression</td>
<td>((29 \times d20 + \text{address ISR}))</td>
</tr>
<tr>
<td>procedure identifier</td>
<td>((30 \times d20 + \text{address ISR}))</td>
</tr>
<tr>
<td>not assignable expression of unknown type</td>
<td>((31 \times d20 + \text{address ISR}))</td>
</tr>
<tr>
<td>possibly assignable expression of unknown type</td>
<td>((32 \times d20 + \text{address ISR}))</td>
</tr>
</tbody>
</table>
5. STRUCTURE OF THE NAME LIST

In this chapter we briefly describe the structure of the name list in order to facilitate an analysis of the name-list procedures. No advanced techniques whatsoever are used: no hash tables, no binary search etc. The main design objectives have been compactness and simplicity, and never efficiency. In spite of this compile time has never been felt to be a bottle-neck to the system, except for ALOOK programs like the one presented in this tract. In the sequel we discuss the structure of a name cell first; then the overall structure of the name list and the look-up process for an identifier will be discussed.

5.1 NAME CELLS

Each name cell consists of:
1) a variable number of negative words, containing the component letters and digits of an identifier;
2) a descriptor of one, two or at most three positive words.

The word length of the EL-X8 is 27 bits; the digits 0:9 have internal representation 0:9, the letters a:z have 10:35, the letters A:Z have 37:62. In a name cell these values have been increased by 1. Each letter or digit is coded in 6 bits, and each word can contain 4 letters or digits. For the representation of an identifier of k letters and digits, \((k+3) + 4\) (negative) words are needed. The identifier "a0123" is coded in octal:

\[
\begin{align*}
\text{space [nl base - pointer]} & : '013010203' \\
\text{space [nl base - pointer - 1]} & : '00000004'
\end{align*}
\]

(cf. read identifier, page 11).

In this way identifiers of any length can be represented and discriminated.

The first word of any descriptor has the following lay-out:

\[
\begin{array}{ccc}
0 & \text{code} & m & \text{address} \\
\hline
d26 & d19 & d18 & d0
\end{array}
\]
in which:

code: a number specifying information on the nature of the object identified by the identifier;

m: for formal parameters called by name: whether the formal parameter occurs as left part somewhere in the procedure body or not, for value formal parameters: always C,
for non-formal identifiers: whether the address is a static address (in the array space) or a dynamic address (in the run-time stack); addresses of formal parameters always are dynamic (cf. Assigned to and Dynamic on page 84);

address: the address assigned to the identifier (in the case of a label identifier this address is the address of a pseudo label variable; however, not every label has such a variable assigned to it, cf. section 0.2).

The following codes can occur in a name cell:

1) for simple variables, labels, arrays, switches and procedures:

<table>
<thead>
<tr>
<th>decimal</th>
<th>octal</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>'000'</td>
<td>real variable identifier</td>
</tr>
<tr>
<td>1</td>
<td>'001'</td>
<td>integer variable identifier</td>
</tr>
<tr>
<td>2</td>
<td>'002'</td>
<td>Boolean variable identifier</td>
</tr>
<tr>
<td>3</td>
<td>'003'</td>
<td>string variable identifier</td>
</tr>
<tr>
<td>6</td>
<td>'006'</td>
<td>label identifier</td>
</tr>
<tr>
<td>8</td>
<td>'010'</td>
<td>real array identifier</td>
</tr>
<tr>
<td>9</td>
<td>'011'</td>
<td>integer array identifier</td>
</tr>
<tr>
<td>10</td>
<td>'012'</td>
<td>Boolean array identifier</td>
</tr>
<tr>
<td>11</td>
<td>'013'</td>
<td>string array identifier</td>
</tr>
<tr>
<td>14</td>
<td>'016'</td>
<td>switch identifier</td>
</tr>
<tr>
<td>16</td>
<td>'020'</td>
<td>real procedure identifier</td>
</tr>
<tr>
<td>17</td>
<td>'021'</td>
<td>integer procedure identifier</td>
</tr>
<tr>
<td>18</td>
<td>'022'</td>
<td>Boolean procedure identifier</td>
</tr>
<tr>
<td>19</td>
<td>'023'</td>
<td>string procedure identifier</td>
</tr>
<tr>
<td>24</td>
<td>'030'</td>
<td>procedure identifier</td>
</tr>
<tr>
<td>32</td>
<td>'040'</td>
<td>own real variable identifier</td>
</tr>
<tr>
<td>33</td>
<td>'041'</td>
<td>own integer variable identifier</td>
</tr>
<tr>
<td>34</td>
<td>'042'</td>
<td>own Boolean variable identifier</td>
</tr>
<tr>
<td>35</td>
<td>'043'</td>
<td>own string variable identifier</td>
</tr>
</tbody>
</table>
40 '050' own real array identifier
41 '051' own integer array identifier
42 '052' own Boolean array identifier
43 '053' own string array identifier

2) for value parameters:

<table>
<thead>
<tr>
<th>decimal</th>
<th>octal</th>
<th>specified</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>'100'</td>
<td>real or real procedure</td>
</tr>
<tr>
<td>65</td>
<td>'101'</td>
<td>integer or integer procedure</td>
</tr>
<tr>
<td>66</td>
<td>'102'</td>
<td>Boolean or Boolean procedure</td>
</tr>
<tr>
<td>67</td>
<td>'103'</td>
<td>string or string procedure</td>
</tr>
<tr>
<td>70</td>
<td>'106'</td>
<td>label</td>
</tr>
<tr>
<td>72</td>
<td>'110'</td>
<td>array or real array</td>
</tr>
<tr>
<td>73</td>
<td>'111'</td>
<td>integer array</td>
</tr>
<tr>
<td>74</td>
<td>'112'</td>
<td>Boolean array</td>
</tr>
<tr>
<td>75</td>
<td>'113'</td>
<td>string array</td>
</tr>
</tbody>
</table>

3) for name parameters:

3.1) occurring in expressions or left parts as "primaries":

<table>
<thead>
<tr>
<th>decimal</th>
<th>octal</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>'140'</td>
<td>real</td>
</tr>
<tr>
<td>97</td>
<td>'141'</td>
<td>integer</td>
</tr>
<tr>
<td>98</td>
<td>'142'</td>
<td>Boolean</td>
</tr>
<tr>
<td>99</td>
<td>'143'</td>
<td>string</td>
</tr>
<tr>
<td>100</td>
<td>'144'</td>
<td>arithmetic (real or integer)</td>
</tr>
<tr>
<td>101</td>
<td>'145'</td>
<td>non-designational (real, integer, Boolean or string)</td>
</tr>
<tr>
<td>102</td>
<td>'146'</td>
<td>designational</td>
</tr>
</tbody>
</table>

3.2) occurring as array or switch identifier:

<table>
<thead>
<tr>
<th>decimal</th>
<th>octal</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
<td>'150'</td>
<td>real array</td>
</tr>
<tr>
<td>105</td>
<td>'151'</td>
<td>integer array</td>
</tr>
<tr>
<td>106</td>
<td>'152'</td>
<td>Boolean array</td>
</tr>
<tr>
<td>107</td>
<td>'153'</td>
<td>string array</td>
</tr>
<tr>
<td>108</td>
<td>'154'</td>
<td>real or integer array</td>
</tr>
<tr>
<td>109</td>
<td>'155'</td>
<td>array of unknown type</td>
</tr>
</tbody>
</table>
3.3) occurring as procedure identifier:

<table>
<thead>
<tr>
<th>decimal</th>
<th>octal</th>
</tr>
</thead>
<tbody>
<tr>
<td>112</td>
<td>'160'</td>
</tr>
<tr>
<td>113</td>
<td>'161'</td>
</tr>
<tr>
<td>114</td>
<td>'162'</td>
</tr>
<tr>
<td>115</td>
<td>'163'</td>
</tr>
<tr>
<td>116</td>
<td>'164'</td>
</tr>
<tr>
<td>117</td>
<td>'165'</td>
</tr>
<tr>
<td>120</td>
<td>'170'</td>
</tr>
</tbody>
</table>

3.4) otherwise:

<table>
<thead>
<tr>
<th>decimal</th>
<th>octal</th>
</tr>
</thead>
<tbody>
<tr>
<td>127</td>
<td>'177'</td>
</tr>
</tbody>
</table>

Specification of name parameters leads in *prescanU* to octal codes equal to '142' (Boolean), '143' (string), '144' (real or integer), '146' (label), '152' (boolean array), '153' (string array), '154' (real array or integer array), '155' (array), '156' (switch), '162' (boolean procedure), '163' (string procedure), '164' (real procedure or integer procedure) or '170' (procedure).

The other codes can result only from code transitions during *prescanI* by virtue of the contexts in which the parameter is used.

The following direct transitions can occur in this process:

- '140' to '150' or '160';
- '141' to '151' or '161';
- '142' to '152' or '162';
- '143' to '153' or '163';
- '144' to '140', '141', '154' or '164';
- '145' to '140', '141', '142', '143', '144', '155' or '165';
- '146' to '156';
- '154' to '150' or '151';
- '155' to '150', '151', '152', '153' or '154';
- '157' to '150', '151', '152', '153', '154', '155' or '156';
'165' to '162', '163' or '164';
'170' to '162', '163', '164' or '165';
'177' to '140', '141', '142', '143', '144', '145', '146', '157' or '170'.

Several of these transitions may occur one after another; one and the same context can induce two transitions in succession. In
procedure p(x, y); x := y + 2;...
the initial code '177' for x is transformed into '145' first by a call of Assigned to (page 40) and later on into '144' by a call of Insert (page 30) with type = 4; the initial code '177' for y is transformed into '144' by a call of Arithmetic (page 39).

For simple <local or own type> variables the descriptor consists of one word only. All other name cells have a second descriptor word. The layout depends on the nature of the object identified by the identifier (as given by its code in the first descriptor word) in the following way:

1) for array identifiers and for formal parameters specified or used as array identifier, switch identifier of procedure identifier:

```
<table>
<thead>
<tr>
<th>0</th>
<th>ll</th>
</tr>
</thead>
<tbody>
<tr>
<td>d18</td>
<td>d0</td>
</tr>
</tbody>
</table>
```

ll = 0 if the dimension or the number of parameters is unknown,
ll = 1 + (dimension or number of parameters), otherwise;

2) for all other formal parameters:

```
<table>
<thead>
<tr>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>d0</td>
</tr>
</tbody>
</table>
```

3) for switch identifiers:

```
<table>
<thead>
<tr>
<th>0</th>
<th>d</th>
<th>0</th>
<th>ll</th>
</tr>
</thead>
<tbody>
<tr>
<td>d20</td>
<td>d19</td>
<td>d18</td>
<td>d3</td>
</tr>
</tbody>
</table>
```

ll = 1 + number of entries in the switch list,
d = 1 after assignment of a "program address" to the switch identifier,
i.e. after analysis of the switch list in the third scan;
4) for label identifiers:

```
<table>
<thead>
<tr>
<th>d</th>
<th>fc</th>
<th>d</th>
<th>s</th>
<th>program address</th>
</tr>
</thead>
<tbody>
<tr>
<td>d26</td>
<td>d20</td>
<td>d19</td>
<td>d18</td>
<td>d0</td>
</tr>
</tbody>
</table>
```

- **fc** = (for count =) the number of nested for statements the label "declaration" is enveloped in (this number is used for a very rough check against jumps from outside into a for statement),
- **d** = 1 after assignment of a "program address" to the label, i.e. after its occurrence as label during the third scan,
- **s** = 1 for labels to which no pseudo label variable is assigned.

5) for procedure identifiers:

```
<table>
<thead>
<tr>
<th>d</th>
<th>p</th>
<th>c</th>
<th>e</th>
<th>a</th>
<th>d</th>
<th>0</th>
<th>0</th>
<th>ll</th>
</tr>
</thead>
<tbody>
<tr>
<td>d26</td>
<td>d18</td>
<td>d18</td>
<td>d0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- **ll** = 1 + number of parameters,
- **l** = 1 for library procedures (not declared in the program),
- **c** = 1 for code procedures (declared in the program with code body),
- **p** = 1 for some library procedures like abs that are compiled as monadic operator,
- **e** = 1 except during the analysis of the procedure body in the third scan,
- **a** = 1 after the occurrence of the assignment of a function value to the procedure identifier within the body of a type procedure during the third scan,
- **d** = 1 after the assignment of a "program address" to the procedure in the third scan.

A third descriptor word exists for type procedure identifiers (codes '20', '21', '22' or '23') only. It contains the descriptor of a simple variable of corresponding type. This variable is assigned to the procedure to record the function value (cf. the compiler procedure Local position, page 85).

The declaration `array a [I:10]` in a program leads to the following name cell for `a`:
space [nl base - pointer]  :  '000000013'
space [nl base - pointer - 1]  :  '021002k07' (code='10',m=1)
space [nl base - pointer - 2]  :  '000000002' (ll=2)

assuming a dynamic address '002407' for a.

For integer labels a special name cell is constructed with a normal descriptor preceded by two negative words containing together the integer value of the label (cf. in name list, page 12). No identifier can produce the same name cell as an integer label on account of a maximum value for integers of $2 + 26 - 1$.

The result of the procedure Identifier, both in proarray and in translate, always points to the first descriptor word.

5.2 BLOCK CELLS

To each block in the program and to each procedure declaration corresponds a block cell in the name list. A block cell consists of the following sections:

1) a block head of 4 positive words containing general information about the block or procedure body;
2) a negative word acting to the name-look-up procedure as a jump instruction and leading to the fifth section of the block cell (with the name cells of local identifiers);
3) the name cells of the formal parameters of the block cell, simply one after another without any separators (for blocks and for procedures without parameters this section is empty);
4) a negative word acting as a jump and leading to the fifth word of the block cell corresponding to the smallest embracing block (or procedure). This fifth word itself points to the locals of that block as described in section 2;
5) the name cells of the local identifiers of the block, simply one after another without any separators. In between these name cells, however, block cells may occur, either for procedures or for inner blocks. These block cells are each preceded by a jump over the block cell;
6) a number of consecutive one-word descriptors of pseudo for variables, coded as simple variables of type integer (i.e. code = '1');
7) a negative word acting as a jump and leading to the third section of
the block cell (with the name cells of formal identifiers).

Some of these sections can be empty, viz. section 3 (see above), section 5
(for procedures without locals) and section 6 (for blocks or procedures
without for statements).

A jump pointing to a word in the array space with index nl base - i is
coded as - (2*25 + i); it can thus be distinguished from negative words
of name cells, never that negative (cf. next pointer, page 11).

A typical block cell (for a procedure) is visualized in the following
picture displaying the contents of space [nl base - i] for i = bcp (1)
bcp + 43:

```
+ bcp
  + bcp + 1
    + bcp + 2
      + bcp + 3
        + (2*25+bcp+12)
          + bcp + 4
            + bcp + 5
              + bcp + 6
                + bcp + 7
                  + bcp + 8
                    + bcp + 9
                      + bcp + 10
                        + bcp + 11
                          + bcp + 12
                            + bcp + 13
                              + bcp + 14
                                + bcp + 15
                                  + bcp + 16
                                    + bcp + 17
                                      + (2*25+bcp+38)
                                        + bcp + 18
                                          + block cell with
                                            formal parameters
                                              and locals of the
                                                procedure...
```
In this picture bcp is the "address" of the block cell.

The search process by which an identifier is looked up in the name list during `preasm1` or `translate` is rather simple and direct: the search is started at the fifth word of the appropriate block cell, leading to the first local identifier of the block cell. To skip an inapplicable name cell the momentary value of the pointer used in the search process is decreased by one until the first word of the next name cell is found (jumps encountered during this skip are honoured, of course). When the locals are exhausted the search is automatically continued with the formals of the same block and next with the locals of the smallest embracing block and so on until the identifier is found (due to a trick to be discussed below it will certainly be found). For identifiers occurring inside the bound pair list of an array declaration the search is started with the formal identifiers of the appropriate block cell rather than with the locals (cf. the procedure `look up`, page 12).

For the program as a whole, an outermost block cell is constructed starting at `space [nl base]` (its "address", therefore, equals 0), which is open-ended: section 7 is lacking. In this block cell labels global to all blocks of the program are inserted as well as, during `preasm1`, identifiers for which no declaration can be found. At the end of this block cell (found through the global compiler variable `nil`), `read identifier` (page 11) assembles identifiers. If an identifier is looked up that has not been encountered before, the search process automatically ends at this temporary cell at the end of the name list.

During `preasm0`, the name list is constructed as an abstract of block structure and declarations. Identifiers to be added to the name list are
looked up (see Process identifier, page 23) for the purpose of detecting
doubly declared identifiers. Identifiers occurring in the value part or
in the specification part of a procedure declaration are looked up as well.
In both cases the search is, of course, restricted to the current block
 cell which, then, is still open-ended.

The global compiler variable block cell pointer always contains the address
in the name list of the current block cell. It has to be updated both at
block entrance and at block exit. During prescan and translate this is
performed in the following way: at block entrance to block cell pointer is
assigned the value of another global compiler variable next block cell
pointer. This last one is updated with the contents of the least signifi-
cant part of the first word of the new block cell (all block heads in the
name list are linked together in a forward chain); at block exit block cell
pointer is given the value of the most significant part of the first word
of the (old) block cell (each block head in the name list points to the
block head of the smallest embracing block, if any). These chains are con-
structed during prescan which uses a different method for updating block
cell pointer.

We do not specify the contents of the other three words of a block head,
except for the most significant part of its third word (called status),
which points to one of the descriptors of pseudo for variables (if any)
in the block cell and which is decreased or increased by one before or
after the analysis of the statement after do of a for statement during
translate. In this way parallel for statements share the same pseudo for
variable but nested for statements use different ones.
6. CROSS-REFERENCE TABLE

The cross-reference table is meant to be an aid to the study of the compiler. To all relevant procedure identifiers occurring in the ALGOL text, an order number is added. The identifiers are listed alphabetically, each item preceded by its order number and the number of the page where its declaration occurs, and followed by a list or order numbers of the procedures from where it is directly called.
<table>
<thead>
<tr>
<th>nr.</th>
<th>proc.</th>
<th>identifier</th>
<th>calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Actual</td>
<td>parameter</td>
<td>176</td>
</tr>
<tr>
<td>2</td>
<td>Actual</td>
<td>parameter</td>
<td>175</td>
</tr>
<tr>
<td>3</td>
<td>Address</td>
<td>6 21 117 240</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Address</td>
<td>2 187 261</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Address</td>
<td>description</td>
<td>2 83 252</td>
</tr>
<tr>
<td>6</td>
<td>Address</td>
<td>description</td>
<td>33 183</td>
</tr>
<tr>
<td>7</td>
<td>Add type</td>
<td>11 17 25 36 70 111 198 245</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Aassign</td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Arboolexp</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Arboolrest</td>
<td>215</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Arboast</td>
<td>88 209</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Arboast</td>
<td>2 55 175</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Ardesexp</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Arithconstant</td>
<td>2 179 226</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Arithexp</td>
<td>15 47 85 216 254 272</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Arithexp</td>
<td>8 9 16 48 80 83 110 179 186 200 253</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Arithmetic</td>
<td>85 106 209 216 219 225</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Arithmetic</td>
<td>2 10 44 55 73 81 171 175 184</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Arithname</td>
<td>8 10 44 81 110 179 184 200 264</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Arithoperator last symbol</td>
<td>10 43 44 81 164 207</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Array declaration</td>
<td>21 63</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Array declaration</td>
<td>22 64</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Ask librarian</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Assign</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Assigned to</td>
<td>Assign</td>
<td>209</td>
</tr>
<tr>
<td>26</td>
<td>Assigned to</td>
<td>Assign</td>
<td>74 156 175 184</td>
</tr>
<tr>
<td>27</td>
<td>Assignstat</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Assignstat</td>
<td>236</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Assign function designator</td>
<td>127 212</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Begin statement</td>
<td>193 237</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Bit string</td>
<td>6 26 51 70 72 75 87 94 101</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Block</td>
<td>30 62</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>Block</td>
<td>194 238</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>Block</td>
<td>195 236</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>Boolean</td>
<td>35 73</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>Boolean</td>
<td>106 209 219 225</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>Boolean</td>
<td>2 10 55 73 77 81 171 175 184</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>Boolean</td>
<td>38 79 85 98 272</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>Boolean</td>
<td>9 16 35 39 68 80 83 99 249</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>Boolean</td>
<td>46 135</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>Boolean</td>
<td>10 44 81 184</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>Boolean</td>
<td>10 81 164 207 209 215 225 226</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>Boolean</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>Boolean</td>
<td>35 43</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>Boolean</td>
<td>35 136</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>Boolean</td>
<td>100 137</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>Bound pair list</td>
<td>21 47</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>Bound pair list</td>
<td>22 48</td>
<td></td>
</tr>
<tr>
<td>nr.</td>
<td>proc.</td>
<td>identifier</td>
<td>calls</td>
</tr>
<tr>
<td>-----</td>
<td>-------</td>
<td>------------</td>
<td>-------</td>
</tr>
<tr>
<td>49</td>
<td>82</td>
<td>B reaction</td>
<td>188</td>
</tr>
<tr>
<td>50</td>
<td>85</td>
<td>Change</td>
<td>127</td>
</tr>
<tr>
<td>51</td>
<td>82</td>
<td>Character</td>
<td>1 18</td>
</tr>
<tr>
<td>52</td>
<td>41</td>
<td>Check dimension</td>
<td>21</td>
</tr>
<tr>
<td>53</td>
<td>87</td>
<td>Check dimension</td>
<td>5</td>
</tr>
<tr>
<td>54</td>
<td>87</td>
<td>Check list length</td>
<td>175</td>
</tr>
<tr>
<td>55</td>
<td>87</td>
<td>Check type</td>
<td>175</td>
</tr>
<tr>
<td>56</td>
<td>82</td>
<td>Code bits</td>
<td>4 53</td>
</tr>
<tr>
<td>57</td>
<td>19</td>
<td>Compound tail</td>
<td>30</td>
</tr>
<tr>
<td>58</td>
<td>31</td>
<td>Compound tail</td>
<td>33</td>
</tr>
<tr>
<td>59</td>
<td>69</td>
<td>Compound tail</td>
<td>36</td>
</tr>
<tr>
<td>60</td>
<td>69</td>
<td>Constant string</td>
<td>224</td>
</tr>
<tr>
<td>61</td>
<td>15</td>
<td>corresponding block cell pointer</td>
<td>70</td>
</tr>
<tr>
<td>62</td>
<td>20</td>
<td>Declaration list</td>
<td>32</td>
</tr>
<tr>
<td>63</td>
<td>34</td>
<td>Declaration list</td>
<td>33</td>
</tr>
<tr>
<td>64</td>
<td>76</td>
<td>Declaration list</td>
<td>34</td>
</tr>
<tr>
<td>65</td>
<td>8</td>
<td>declarator last symbol</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>195</td>
</tr>
<tr>
<td>66</td>
<td>85</td>
<td>Declared</td>
<td>133</td>
</tr>
<tr>
<td>67</td>
<td>27</td>
<td>Desigexp</td>
<td>67</td>
</tr>
<tr>
<td>68</td>
<td>54</td>
<td>Desigexp</td>
<td>68</td>
</tr>
<tr>
<td>69</td>
<td>55</td>
<td>Designame</td>
<td>81</td>
</tr>
<tr>
<td>70</td>
<td>40</td>
<td>Designational</td>
<td>92</td>
</tr>
<tr>
<td>71</td>
<td>82</td>
<td>Designtional</td>
<td>2</td>
</tr>
<tr>
<td>72</td>
<td>14</td>
<td>display level</td>
<td>6</td>
</tr>
<tr>
<td>73</td>
<td>57</td>
<td>Distribute on type</td>
<td>8</td>
</tr>
<tr>
<td>74</td>
<td>84</td>
<td>Dynamic</td>
<td>2 4</td>
</tr>
<tr>
<td>75</td>
<td>15</td>
<td>entrance block</td>
<td>33</td>
</tr>
<tr>
<td>76</td>
<td>3</td>
<td>ERROMESSAGE</td>
<td>5</td>
</tr>
<tr>
<td>77</td>
<td>48</td>
<td>Evaluation of</td>
<td>2</td>
</tr>
<tr>
<td>78</td>
<td>15</td>
<td>exit block</td>
<td>6 34</td>
</tr>
<tr>
<td>79</td>
<td>27</td>
<td>Exp</td>
<td>1 219</td>
</tr>
<tr>
<td>80</td>
<td>35</td>
<td>Exp</td>
<td>2 13</td>
</tr>
<tr>
<td>81</td>
<td>57</td>
<td>Expresnt</td>
<td>2 226</td>
</tr>
<tr>
<td>82</td>
<td>46</td>
<td>Factor</td>
<td>138</td>
</tr>
<tr>
<td>83</td>
<td>70</td>
<td>For list</td>
<td>86</td>
</tr>
<tr>
<td>84</td>
<td>84</td>
<td>Formal</td>
<td>2 19</td>
</tr>
<tr>
<td>85</td>
<td>32</td>
<td>Forstat</td>
<td>238</td>
</tr>
<tr>
<td>86</td>
<td>69</td>
<td>Forstat</td>
<td>236</td>
</tr>
<tr>
<td>87</td>
<td>81</td>
<td>Forward jumping macro</td>
<td>132</td>
</tr>
<tr>
<td>88</td>
<td>41</td>
<td>Function</td>
<td>90</td>
</tr>
<tr>
<td>89</td>
<td>83</td>
<td>Function</td>
<td>19 83</td>
</tr>
<tr>
<td>90</td>
<td>30</td>
<td>Function designator</td>
<td>209</td>
</tr>
<tr>
<td>91</td>
<td>60</td>
<td>Function designator</td>
<td>8 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>250</td>
</tr>
<tr>
<td>nr.</td>
<td>page</td>
<td>proc.</td>
<td>identifier</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>-------</td>
<td>------------</td>
</tr>
<tr>
<td>92</td>
<td>31</td>
<td>Gotostat</td>
<td>238</td>
</tr>
<tr>
<td>93</td>
<td>69</td>
<td>Gotostat</td>
<td>236</td>
</tr>
<tr>
<td>94</td>
<td>84</td>
<td>Id1</td>
<td>29 66 165 173 259</td>
</tr>
<tr>
<td>95</td>
<td>23</td>
<td>Identifier</td>
<td>32 66</td>
</tr>
<tr>
<td>96</td>
<td>42</td>
<td>Identifier</td>
<td>21 85 92 183 194 209 211 216 219</td>
</tr>
<tr>
<td>97</td>
<td>87</td>
<td>Identifier</td>
<td>2 8 22 35 43 83 93 110 179</td>
</tr>
<tr>
<td>98</td>
<td>32</td>
<td>Ifclause</td>
<td>15 38 67 238 248</td>
</tr>
<tr>
<td>99</td>
<td>69</td>
<td>Ifstatement</td>
<td>236</td>
</tr>
<tr>
<td>100</td>
<td>49</td>
<td>Implication</td>
<td>141</td>
</tr>
<tr>
<td>101</td>
<td>14</td>
<td>In code</td>
<td>183 184</td>
</tr>
<tr>
<td>102</td>
<td>15</td>
<td>Init</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>87</td>
<td>Increase status</td>
<td>86</td>
</tr>
<tr>
<td>104</td>
<td>84</td>
<td>In library</td>
<td>2 139 175 182 184 191</td>
</tr>
<tr>
<td>105</td>
<td>12</td>
<td>in name list</td>
<td>2 113 114 194 195 221 222 225 226</td>
</tr>
<tr>
<td>106</td>
<td>30</td>
<td>Insert</td>
<td>209</td>
</tr>
<tr>
<td>107</td>
<td>81</td>
<td>Instruct number</td>
<td>192</td>
</tr>
<tr>
<td>108</td>
<td>82</td>
<td>Instruct part</td>
<td>192</td>
</tr>
<tr>
<td>109</td>
<td>5</td>
<td>Insymbol</td>
<td>109 134</td>
</tr>
<tr>
<td>110</td>
<td>58</td>
<td>Intassign</td>
<td>73 110</td>
</tr>
<tr>
<td>111</td>
<td>39</td>
<td>Integer</td>
<td>106</td>
</tr>
<tr>
<td>112</td>
<td>82</td>
<td>Integer</td>
<td>2 19 73 77 83 184 242</td>
</tr>
<tr>
<td>113</td>
<td>35</td>
<td>Int lab declaration</td>
<td>193 237</td>
</tr>
<tr>
<td>114</td>
<td>35</td>
<td>Intlabelled statement</td>
<td>238</td>
</tr>
<tr>
<td>115</td>
<td>84</td>
<td>In value list</td>
<td>139 184</td>
</tr>
<tr>
<td>116</td>
<td>22</td>
<td>Label declaration</td>
<td>113 193 237</td>
</tr>
<tr>
<td>117</td>
<td>35</td>
<td>Label declaration</td>
<td>114 119 194</td>
</tr>
<tr>
<td>118</td>
<td>77</td>
<td>Label declaration</td>
<td>195 236</td>
</tr>
<tr>
<td>119</td>
<td>35</td>
<td>Labelled statement</td>
<td>238</td>
</tr>
<tr>
<td>120</td>
<td>76</td>
<td>Label list</td>
<td>34 184</td>
</tr>
<tr>
<td>121</td>
<td>67</td>
<td>Line</td>
<td>22 236</td>
</tr>
<tr>
<td>122</td>
<td>67</td>
<td>Linel</td>
<td>86 121</td>
</tr>
<tr>
<td>123</td>
<td>41</td>
<td>List length</td>
<td>90 190 251</td>
</tr>
<tr>
<td>124</td>
<td>86</td>
<td>List length</td>
<td>22 53 54 158 182 184 186 261</td>
</tr>
<tr>
<td>125</td>
<td>79</td>
<td>Load</td>
<td>132</td>
</tr>
<tr>
<td>126</td>
<td>15</td>
<td>Local label</td>
<td>92 93</td>
</tr>
<tr>
<td>127</td>
<td>85</td>
<td>Local position</td>
<td>178 184</td>
</tr>
<tr>
<td>128</td>
<td>85</td>
<td>Local position!</td>
<td>127 210</td>
</tr>
<tr>
<td>129</td>
<td>14</td>
<td>Local space</td>
<td>34 184</td>
</tr>
<tr>
<td>130</td>
<td>12</td>
<td>Look up</td>
<td>96 97 105 185</td>
</tr>
<tr>
<td>131</td>
<td>78</td>
<td>Macro</td>
<td>2 8 22 35 45 48 77 83 93 110</td>
</tr>
<tr>
<td>132</td>
<td>78</td>
<td>Macro2</td>
<td>2 5 8 9 10 14 16 19 22 34</td>
</tr>
<tr>
<td>133</td>
<td>86</td>
<td>Mark position in name list</td>
<td>118 184 261</td>
</tr>
<tr>
<td>134</td>
<td>4</td>
<td>next basic symbol</td>
<td>145</td>
</tr>
<tr>
<td>nr.</td>
<td>page</td>
<td>proc. identifier</td>
<td>calls</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>-----------------</td>
<td>-------</td>
</tr>
<tr>
<td>135</td>
<td>49</td>
<td>Next boolfac</td>
<td>46</td>
</tr>
<tr>
<td>136</td>
<td>50</td>
<td>Next boolsec</td>
<td>40</td>
</tr>
<tr>
<td>137</td>
<td>49</td>
<td>Next boolterm</td>
<td>100</td>
</tr>
<tr>
<td>138</td>
<td>46</td>
<td>Next factor</td>
<td>2</td>
</tr>
<tr>
<td>139</td>
<td>87</td>
<td>Next formal identifier</td>
<td>175</td>
</tr>
<tr>
<td>140</td>
<td>12</td>
<td>next identifier</td>
<td>6</td>
</tr>
<tr>
<td>141</td>
<td>49</td>
<td>Next implication</td>
<td>141</td>
</tr>
<tr>
<td>142</td>
<td>87</td>
<td>Next local label</td>
<td>120</td>
</tr>
<tr>
<td>143</td>
<td>11</td>
<td>next pointer</td>
<td>130</td>
</tr>
<tr>
<td>144</td>
<td>47</td>
<td>Next primary</td>
<td>2</td>
</tr>
<tr>
<td>145</td>
<td>4</td>
<td>next symbol</td>
<td>2</td>
</tr>
<tr>
<td>146</td>
<td>7</td>
<td>next tape symbol</td>
<td>109</td>
</tr>
<tr>
<td>147</td>
<td>46</td>
<td>Next term</td>
<td>147</td>
</tr>
<tr>
<td>148</td>
<td>83</td>
<td>Nonarithmetic</td>
<td>8</td>
</tr>
<tr>
<td>149</td>
<td>67</td>
<td>Nonassignable</td>
<td>175</td>
</tr>
<tr>
<td>150</td>
<td>83</td>
<td>Nonboolean</td>
<td>35</td>
</tr>
<tr>
<td>151</td>
<td>75</td>
<td>Nondesexp</td>
<td>80</td>
</tr>
<tr>
<td>152</td>
<td>83</td>
<td>Nondesignational</td>
<td>55</td>
</tr>
<tr>
<td>153</td>
<td>15</td>
<td>nonformal label</td>
<td>70</td>
</tr>
<tr>
<td>154</td>
<td>84</td>
<td>Nonfunction</td>
<td>91</td>
</tr>
<tr>
<td>155</td>
<td>83</td>
<td>Noninteger</td>
<td>110</td>
</tr>
<tr>
<td>156</td>
<td>84</td>
<td>Nonproc</td>
<td>175</td>
</tr>
<tr>
<td>157</td>
<td>83</td>
<td>Nonreal</td>
<td>200</td>
</tr>
<tr>
<td>158</td>
<td>84</td>
<td>Nonsimple</td>
<td>175</td>
</tr>
<tr>
<td>159</td>
<td>83</td>
<td>Nonstring</td>
<td>55</td>
</tr>
<tr>
<td>160</td>
<td>84</td>
<td>Nonsubsvvar</td>
<td>175</td>
</tr>
<tr>
<td>161</td>
<td>83</td>
<td>Nontype</td>
<td>55</td>
</tr>
<tr>
<td>162</td>
<td>67</td>
<td>Number descriptor</td>
<td>2</td>
</tr>
<tr>
<td>163</td>
<td>87</td>
<td>Number of local labels</td>
<td>120</td>
</tr>
<tr>
<td>164</td>
<td>9</td>
<td>operator last symbol</td>
<td>2</td>
</tr>
<tr>
<td>165</td>
<td>84</td>
<td>Operator like</td>
<td>182</td>
</tr>
<tr>
<td>166</td>
<td>89</td>
<td>Operator macro</td>
<td>186</td>
</tr>
<tr>
<td>167</td>
<td>81</td>
<td>Optimizable operator</td>
<td>132</td>
</tr>
<tr>
<td>168</td>
<td>79</td>
<td>Optimize</td>
<td>132</td>
</tr>
<tr>
<td>169</td>
<td>81</td>
<td>Opt number</td>
<td>168</td>
</tr>
<tr>
<td>170</td>
<td>78</td>
<td>Order counter</td>
<td>2</td>
</tr>
<tr>
<td>171</td>
<td>51</td>
<td>Ordinal number</td>
<td>2</td>
</tr>
<tr>
<td>172</td>
<td>91</td>
<td>output</td>
<td></td>
</tr>
<tr>
<td>173</td>
<td>84</td>
<td>Outside declaration</td>
<td></td>
</tr>
<tr>
<td>174</td>
<td>7</td>
<td>outsymbol</td>
<td>134</td>
</tr>
<tr>
<td>175</td>
<td>62</td>
<td>Parameter list</td>
<td>182</td>
</tr>
<tr>
<td>176</td>
<td>30</td>
<td>Parlist</td>
<td>90</td>
</tr>
<tr>
<td>177</td>
<td>82</td>
<td>Par part</td>
<td>192</td>
</tr>
<tr>
<td>178</td>
<td>57</td>
<td>Prepare</td>
<td>8</td>
</tr>
<tr>
<td>179</td>
<td>47</td>
<td>Primary</td>
<td>82</td>
</tr>
<tr>
<td>180</td>
<td>41</td>
<td>Proc</td>
<td>88</td>
</tr>
<tr>
<td>nr.</td>
<td>page</td>
<td>proc. identifier</td>
<td>calls</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>------------------</td>
<td>-------</td>
</tr>
<tr>
<td>181</td>
<td>83</td>
<td>Proc</td>
<td>2 91 158 160 161 171 175 191</td>
</tr>
<tr>
<td>182</td>
<td>61</td>
<td>Procedure call</td>
<td>91 131</td>
</tr>
<tr>
<td>183</td>
<td>34</td>
<td>Procedure declaration</td>
<td>63</td>
</tr>
<tr>
<td>184</td>
<td>74</td>
<td>Procedure declaration</td>
<td>64</td>
</tr>
<tr>
<td>185</td>
<td>23</td>
<td>Process identifier</td>
<td>95 193 237</td>
</tr>
<tr>
<td>186</td>
<td>67</td>
<td>Process operator</td>
<td>182</td>
</tr>
<tr>
<td>187</td>
<td>81</td>
<td>Process parameter</td>
<td>192</td>
</tr>
<tr>
<td>188</td>
<td>80</td>
<td>Process stack pointer</td>
<td>192</td>
</tr>
<tr>
<td>189</td>
<td>14</td>
<td>proc level</td>
<td>4 117</td>
</tr>
<tr>
<td>190</td>
<td>30</td>
<td>Procstat</td>
<td>238</td>
</tr>
<tr>
<td>191</td>
<td>60</td>
<td>Procstat</td>
<td>236</td>
</tr>
<tr>
<td>192</td>
<td>79</td>
<td>Produce</td>
<td>132 192 276</td>
</tr>
<tr>
<td>193</td>
<td>17</td>
<td>Program</td>
<td>193</td>
</tr>
<tr>
<td>194</td>
<td>35</td>
<td>Program</td>
<td>194</td>
</tr>
<tr>
<td>195</td>
<td>77</td>
<td>Program</td>
<td>195</td>
</tr>
<tr>
<td>196</td>
<td>86</td>
<td>Program address</td>
<td>4 133 187</td>
</tr>
<tr>
<td>197</td>
<td>11</td>
<td>read identifier</td>
<td>95 96 97 193 237</td>
</tr>
<tr>
<td>198</td>
<td>39</td>
<td>Real</td>
<td>106</td>
</tr>
<tr>
<td>199</td>
<td>82</td>
<td>Real</td>
<td>2 73 171</td>
</tr>
<tr>
<td>200</td>
<td>58</td>
<td>Realassign</td>
<td>73 200</td>
</tr>
<tr>
<td>201</td>
<td>51</td>
<td>Relation</td>
<td>43 204 208</td>
</tr>
<tr>
<td>202</td>
<td>89</td>
<td>Relatmacro</td>
<td>201</td>
</tr>
<tr>
<td>203</td>
<td>8</td>
<td>relatoperator last symbol</td>
<td>10 14 81 164 201 207</td>
</tr>
<tr>
<td>204</td>
<td>52</td>
<td>Rest of arboolexp</td>
<td>2 10 81 215 226</td>
</tr>
<tr>
<td>205</td>
<td>52</td>
<td>Rest of arithexp</td>
<td>8 110 200 204 208</td>
</tr>
<tr>
<td>206</td>
<td>52</td>
<td>Rest of boolexp</td>
<td>2 10 35 81 204 215 226</td>
</tr>
<tr>
<td>207</td>
<td>28</td>
<td>Rest of exp</td>
<td>209 219 225</td>
</tr>
<tr>
<td>208</td>
<td>51</td>
<td>Rest of relation</td>
<td>43 14</td>
</tr>
<tr>
<td>209</td>
<td>29</td>
<td>Right hand side</td>
<td>27 209</td>
</tr>
<tr>
<td>210</td>
<td>75</td>
<td>Save and restore loc</td>
<td>184</td>
</tr>
<tr>
<td>211</td>
<td>42</td>
<td>Scan code</td>
<td>183</td>
</tr>
<tr>
<td>212</td>
<td>85</td>
<td>Set inside declaration</td>
<td>184</td>
</tr>
<tr>
<td>213</td>
<td>83</td>
<td>Simple</td>
<td>19 41 69 81 158 175 178 250</td>
</tr>
<tr>
<td>214</td>
<td>83</td>
<td>Simple</td>
<td>156 160 213</td>
</tr>
<tr>
<td>215</td>
<td>52</td>
<td>Simple arboolexp</td>
<td>9 226</td>
</tr>
<tr>
<td>216</td>
<td>25</td>
<td>Simple arithexp</td>
<td>15 207 219 225</td>
</tr>
<tr>
<td>217</td>
<td>45</td>
<td>Simple arithexp</td>
<td>16 43 201 215</td>
</tr>
<tr>
<td>218</td>
<td>81</td>
<td>Simple arithmetic take macro</td>
<td>132</td>
</tr>
<tr>
<td>219</td>
<td>26</td>
<td>Simple boolean</td>
<td>30 207 225</td>
</tr>
<tr>
<td>220</td>
<td>49</td>
<td>Simple boolean</td>
<td>39 215 226</td>
</tr>
<tr>
<td>221</td>
<td>27</td>
<td>Simple desigexp</td>
<td>67</td>
</tr>
<tr>
<td>222</td>
<td>51</td>
<td>Simple desigexp</td>
<td>68</td>
</tr>
<tr>
<td>223</td>
<td>26</td>
<td>Simple stringexp</td>
<td>225 248</td>
</tr>
<tr>
<td>224</td>
<td>53</td>
<td>Simple stringexp</td>
<td>226 249</td>
</tr>
<tr>
<td>225</td>
<td>28</td>
<td>Simplexp</td>
<td>79</td>
</tr>
<tr>
<td>226</td>
<td>56</td>
<td>Simplexp</td>
<td>80</td>
</tr>
<tr>
<td>227</td>
<td>13</td>
<td>skip identifier</td>
<td>21 62 227 232</td>
</tr>
<tr>
<td>228</td>
<td>88</td>
<td>Skip parameter list</td>
<td>184</td>
</tr>
<tr>
<td>229</td>
<td>13</td>
<td>skip rest of statement</td>
<td>58 59 229</td>
</tr>
<tr>
<td>230</td>
<td>13</td>
<td>skip specification list</td>
<td>183 184 230</td>
</tr>
<tr>
<td>231</td>
<td>13</td>
<td>skip string</td>
<td>62 229 237</td>
</tr>
<tr>
<td>232</td>
<td>13</td>
<td>skip type declaration</td>
<td>63 64 228 230 232 233</td>
</tr>
<tr>
<td>233</td>
<td>13</td>
<td>skip value list</td>
<td>183 184</td>
</tr>
<tr>
<td>nr.</td>
<td>page</td>
<td>proc.</td>
<td>identifier</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>-------</td>
<td>--------------------</td>
</tr>
<tr>
<td>234</td>
<td>9</td>
<td>specifier</td>
<td>last symbol</td>
</tr>
<tr>
<td>235</td>
<td>67</td>
<td>Start</td>
<td>implicit subroutine</td>
</tr>
<tr>
<td>236</td>
<td>68</td>
<td>Stat</td>
<td>236 239 274</td>
</tr>
<tr>
<td>237</td>
<td>32</td>
<td>Statement</td>
<td>57</td>
</tr>
<tr>
<td>238</td>
<td>58 114 119 183 238</td>
<td></td>
<td></td>
</tr>
<tr>
<td>239</td>
<td>68</td>
<td>Statement</td>
<td>59 85 99 184</td>
</tr>
<tr>
<td>240</td>
<td>38</td>
<td>Static addressing</td>
<td>6</td>
</tr>
<tr>
<td>241</td>
<td>14</td>
<td>status</td>
<td>6 83 86 240</td>
</tr>
<tr>
<td>242</td>
<td>70</td>
<td>Store macro</td>
<td>83</td>
</tr>
<tr>
<td>243</td>
<td>23</td>
<td>Store numerical constant</td>
<td>62 237</td>
</tr>
<tr>
<td>244</td>
<td>70</td>
<td>Store preparation</td>
<td>83</td>
</tr>
<tr>
<td>245</td>
<td>39</td>
<td>String</td>
<td>223</td>
</tr>
<tr>
<td>246</td>
<td>59</td>
<td>Stringassign</td>
<td>73 247 267</td>
</tr>
<tr>
<td>247</td>
<td>59</td>
<td>Stringassign</td>
<td>73 247 267</td>
</tr>
<tr>
<td>248</td>
<td>26</td>
<td>Stringexp</td>
<td>234 247 272</td>
</tr>
<tr>
<td>249</td>
<td>53</td>
<td>Stringexp</td>
<td>204 224 247 249</td>
</tr>
<tr>
<td>250</td>
<td>54</td>
<td>Stringname</td>
<td>81 184 224 247</td>
</tr>
<tr>
<td>251</td>
<td>25</td>
<td>Subscripted variable</td>
<td>27 35 92 209 216 219 221 223 225</td>
</tr>
<tr>
<td>252</td>
<td>47</td>
<td>Subscripted variable</td>
<td>8 28 35 43 93 110 118</td>
</tr>
<tr>
<td>179</td>
<td>200 215 222 224 226 247 261 273</td>
<td></td>
<td></td>
</tr>
<tr>
<td>253</td>
<td>48</td>
<td>Subscript list</td>
<td>5 253</td>
</tr>
<tr>
<td>254</td>
<td>25</td>
<td>Subscriptlist</td>
<td>251 254</td>
</tr>
<tr>
<td>255</td>
<td>40</td>
<td>Subscrvar</td>
<td>251 254</td>
</tr>
<tr>
<td>256</td>
<td>63</td>
<td>Subscrvar</td>
<td>200 236 242 244 247 252 254 254 273</td>
</tr>
<tr>
<td>257</td>
<td>77</td>
<td>Subst2</td>
<td>83 257 258</td>
</tr>
<tr>
<td>258</td>
<td>77</td>
<td>Substitute</td>
<td>9 16 39 54 68 80 83 86 99</td>
</tr>
<tr>
<td>133 175 224 249</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>259</td>
<td>85</td>
<td>Super local</td>
<td>120</td>
</tr>
<tr>
<td>260</td>
<td>32</td>
<td>Switch declaration</td>
<td>63</td>
</tr>
<tr>
<td>261</td>
<td>72</td>
<td>Switch declaration</td>
<td>64</td>
</tr>
<tr>
<td>262</td>
<td>41</td>
<td>Switch length</td>
<td>260</td>
</tr>
<tr>
<td>263</td>
<td>32</td>
<td>Switchlist</td>
<td>260 263</td>
</tr>
<tr>
<td>264</td>
<td>70</td>
<td>Take macro</td>
<td>83</td>
</tr>
<tr>
<td>265</td>
<td>46</td>
<td>Term</td>
<td>2 147 217</td>
</tr>
<tr>
<td>266</td>
<td>86</td>
<td>Test forcount</td>
<td>93</td>
</tr>
<tr>
<td>267</td>
<td>16</td>
<td>Test pointers</td>
<td>145 192 267</td>
</tr>
<tr>
<td>268</td>
<td>7</td>
<td>the underline symbol</td>
<td>199 258</td>
</tr>
<tr>
<td>269</td>
<td>14</td>
<td>top of display</td>
<td>6 184 188</td>
</tr>
<tr>
<td>270</td>
<td>88</td>
<td>Translate code</td>
<td>184</td>
</tr>
<tr>
<td>271</td>
<td>14</td>
<td>type bits</td>
<td>12 18 37 71 112 148 150 152 155 157 159 161 199 209 225 246 275</td>
</tr>
<tr>
<td>272</td>
<td>27</td>
<td>Type exp</td>
<td>79 209</td>
</tr>
<tr>
<td>273</td>
<td>60</td>
<td>Unassign</td>
<td>73</td>
</tr>
<tr>
<td>274</td>
<td>63</td>
<td>Unconditional statement</td>
<td>99</td>
</tr>
<tr>
<td>275</td>
<td>26</td>
<td>Unknown</td>
<td>2 81</td>
</tr>
<tr>
<td>276</td>
<td>79</td>
<td>Unload</td>
<td>132 168</td>
</tr>
<tr>
<td>277</td>
<td>11</td>
<td>unsigned integer</td>
<td>211 270 278</td>
</tr>
<tr>
<td>278</td>
<td>10</td>
<td>unsigned number</td>
<td>62 193 194 216 221 225 237 239 279</td>
</tr>
<tr>
<td>279</td>
<td>99</td>
<td>Unsigned number</td>
<td>2 179 195 222 226 236 261</td>
</tr>
<tr>
<td>280</td>
<td>14</td>
<td>use of counter stack</td>
<td>34 183 184</td>
</tr>
<tr>
<td>281</td>
<td>81</td>
<td>Value like</td>
<td>187</td>
</tr>
</tbody>
</table>