REPORT 164 MA

PROCESS FOR AN ALGOL TRANSLATOR

PART ONE:

THE TRANSLATOR
REPORT 164 MA

PROCESS FOR AN ALGOL TRANSLATOR

PART ONE:

THE TRANSLATOR

BY: G. VAN DER MEY

WITH THE CO-OPERATION OF: W.L. VAN DER POEL
P.A. WITMANS
G.G.M. MULDER

JULY 1962
1.0 General comment

1.1 Procedure S0 : stores word in object program on address P.

1.2 Procedure S0a: extracts from list L and conjugates.

1.3 Procedure S0c: stores word in list L on address S.

1.4 Procedure S0d: equips object programme with storing and
extracting a partial result.

1.5 Procedure S0e: stores declaration list I.

1.6 Procedure S0f: looks up identifier f in list I.

1.7 Procedure S0g: shifts list L to the mid of the range P...T.

1.8 Procedure S0h: completes dynamic introduction of a block.

1.9 Procedure S0i: reserves address P of the object programme
for a pass instruction.

1.10 Procedure S0k: places extract normally instruction.

1.11 Procedure S0l: contra declares a label.

1.12 Procedure S0m: stores word in list L on address So-1.

1.13 Procedure S0n: stores contra declaration and internal
equivalent of an identifier.

1.14 Procedure S0p: contra declares name of function having no
parameters.

1.15 Procedure S0r: inverts gression bit of instruction d.

1.16 Procedure S0s: makes contra declaration of subscripted
actual parameter.

1.17 Progressive and regressive after-actions of the opening symbols.

1.18 Compound statement "Entry".

1.19 Compound statement S1: Entry for delimiters being either an
operator, or a separation or closing symbol.

1.20 Compound statement S2: Pre-action of the opening symbol if.

1.21 Compound statement S2a: After-actions of the opening symbol if.

1.22 Compound statement S3 : Pre-action of the opening symbol begin.

1.23 Compound statement S3a: Entry for declarators and specificators.

1.24 Compound statement S3b: After-action of the opening symbol begin.

1.25 Compound statement S3c: Entry for colon.

1.26 Compound statement S4 : Entry for identifiers.

1.27 Compound statement S4a: Entry for constants.

1.28 Compound statement S5 : Pre-action of the opening symbol (.

1.29 Compound statement S5a: After-actions of the opening symbol (.
1.30 Compound statement S5b: Action of the delimiter lsq.
1.31 Compound statement S6: Pre-action of the opening symbol [.
1.32 Compound statement S6a: After-actions of the opening symbol [.
1.33 Compound statement S7: Pre-action of the opening symbol for.
1.34 Compound statement S7a: After-actions of the opening symbol for.
1.35 Compound statement S8: Pre-action of the opening symbol go to.
1.36 Compound statement S8a: After-actions of the opening symbol go to.
1.37 Compound statement S8b: After-actions of the opening symbol switch.
1.38 Compound statement S8d: After-actions of the opening symbol procedure.
1.40 Compound statement S9a: After-actions of the opening symbol :=.
1.0 General comment

ALGOL Translator

1.0.1

begin

comment

Restrictions in action:

1 In any block head within a text to be translated, the declarations of simple variables and arrays are supposed to precede those of procedures and switches.

2 The bounds of own arrays are supposed to be integral constants.

3 The controlled variable of a for statement may not be a subscripted variable.

Notations:

1 Above each labeled line, all references to the label are gathered, represented by "approximate" labels.

2 Within the programme, many constants are written in a semi-binary notation, in which powers of 2 occur as factors and terms.

C Crosses x on the right-hand side of the pages mark the lines, in which such non-variable expressions appear which, of course, should better be replaced in the text by their values.

3 Other constants, listed in table 1E, are conveniently referred to by names which, in praxis, are to be replaced in the text by the appropriate values. Crosses + on the right-hand side of the pages mark the lines concerned.

Composition:

The translator, a block, contains procedure wrong, the procedures of the S0-series, the switches sw and sw, and the compounds s1, s2, s2a, s3, s3a, s3b, s3c, s4, s4a, s4b, s5, s5a, s5b, s6, s6a, s7, s7a, s8, s8a, s8b, s8d, s9, s9a, input and input1.
input, depending on the kind of string taken from the tape, goes to either $S_4L_1$ (for an identifier), or to $S_4aL_1$ (for an unsigned number), or to one of the labels mentioned in tables 1A and 1B. It assigns the value of the string to variable $f$ of table $W_4$, which is, however, not necessary when the string is a delimiter $: = ( [ \begin{array}{l} \text{begin for go to if lesq} \end{array}$.

When $f$ is a number, input assigns the value $0000000t010\ldots0$ to variable $g$. The bit $t$ indicates the representation of the number $f$.

input1 (cf. S5b).

Non-ALGOL features:

1. The translator considers boolean type to be the same as integer type. Thus variables declared boolean may assume integral values. Internally, true and false are represented by 0 and -1 respectively. Then the significance of operator or can be extended as follows:

2. When applied to integers $p$ and $q$, $p$ or $q$ denotes the logical product of $p$ and $q$ which, of course, might also be obtained by a procedure logical product (par1, par2).

3. The ZEBRA, for which this translator has been developed, is a binary machine with 8192 locations in its store, each of which containing 33 bits $b_0, b_1 \ldots b_{32}$. $b_0$ is the sign digit: $b_0=0 \rightarrow \text{pos.}, 1 \rightarrow \text{neg}$. Then $x < 0$ is a short notation for logical product $(x, 2^{32}) \neq 0$;
own integer array st[PO: QO];

integer a, b, c, d, e, f, g, D, mark, P, q, R, S, S accent, S0, T, T1, T2, u, v;

procedure wrong;

comment
In this space, object programmes are built up, and it also contains the lists I and L needed in translation time (cf. variables P, T, S of table 4A);

cf. table 4A;

This hardware programme stops and prints address, from which it has been invoked. That address corresponds to the occurred kind of misery listed in table 5 as a function of a variable assuming labels as values;
procedure S0(F);
begin
st[P] := F; P := P + 1;
if P > S0-2 then S0g;
end S0;

comment
stores word in object programme on address P;

for shifting list L away from object programme
procedure S0a;
comment extracts from list L and conjugates;
begi
  c := st[S - 3];
d := st[S - 2];
e := st[S - 1];
a := 480 or d; b := 480 or f; 480 is = 511 - 31, thus b is = 32 x rank of
delimiter f (cf. tables 1 A and B)
end S0a;
procedure S0c(F);
begin
    st[S] := F; S := S + 2;
    if S + 1 > T then S0g;
end S0c;

comment
stores word in list L on address S;
for shifting list L away from I
**comment**

**procedure** S0d;

equips object programme with storing and extracting a partial result;

begin
    st[Saccent - 1] :=
    2\text{\( \uparrow \)}23 \times 3 + 2\text{\( \uparrow \)}26 \times 63;
    S0(partres)
    end S0d;
procedure SOe(F)
begin
    st[T - 1]: = e;
    st[T]: = F; T.: = T - 2;
    if S + 1 > T then SOg;
end SOe;

comment
stores declaration in list I;
is internal equivalent of identifier F;
for shifting list L away from I (cf. SOc)
procedure SOf;

looks up identifier f in list I on addresses T1 + 2, T1 + 4, ... T2.

If f is found, then c := lowest address where f occurs, and d := corresponding internal equivalent, which is positive.

Otherwise d := -1.

If f is found to be a simple variable or formal parameter, then e := d.

Otherwise e := 2^{32} + f, which is the contra-identifier of f, being negative.

If f is found to be an array identifier, then g := the arl1 instruction which is the internal equivalent of the factor identifier, being neg.(cf.S6L8) and u := d, which is used on S6L17

Otherwise g := 0;

begin
    g := 0; c := T1;
    S0fL1: if T2 = c then go to S0fL3;
    c := c + 2;
    if st[c] ≠ f then go to S0fL1;
    d := e := st[c - 1];
    if 2 × d > 0 then begin
        if 4 × d > 0 then go to S0fL4;
        e := c - 1;
    end

    Then identifier has not been found;

    for extracting the next identifier;

    which is the internal equivalent of identifier f;

    for label, switch- and procedure identifier;

    f is array identifier;
comment

S0fL2 : e := e - 2; g := st[e];
    if g > 0 then go to S0fL2;
u := d; go to S0fL4;
S0fL3 : d := -1;
S0fL4 : e := f + 2132
    end
    end S0f;

Then an instruction is not yet found;

This compound statement is omitted in the case of a formal parameter or simple variable
procedure S0g;
begin
a := (P + T - S - SO) ÷ 2;

Saccent := Saccent + a;
S := S + a;
SO := SO + a;
if P + 2 > SO then
  S0gL1: wrong;

if a > 0 then
  for b := S step -1 until SO
  do st[b] := st[b - a]
else
  for b := SO step 1 until S
  do st[b] := st[b - a]
end S0g;

comment
shifts list L to the mid of the range P ... T;
The distance from the mid of SO...S to that of P...T is = a + 1 with either l = 0, or 1/2, or -1/2;
Then translator is short of working space.
Otherwise P + 2 is ≤ SO
and, because of abs(l) ≤ 1/2, also S + 1 ≤ T
as required in procedures S0, S0c, S0e, and S0m;
procedure SOh;

begin
  a := st[S - 7];
  c := st[S - 3];  d := st[S - 2];
  e := st[S - 1];
  if d = 322 and c ≠ 0 and a ≥ 0 then
    begin
      if a > 0 then st[a] := P + 2*23 + 2*26 * 121;
      if D < 0 then a := D else a := 0;
      st[S - 7] := a;
      if a < 0 then
        begin
          q := q - 1;
        end
        end
      end
    
This only happens, when translation of a block head is still running;

Then pass instruction (cf. table 1D) is inserted on address a;

Dynamic introduction of block is completed.
q is smallest address reserved for simple variable which is local to block;
Thus, when \( st[S - 2] = 322 \) which is the value of \begin{itemize} \item \text{begin} \item \text{st[S - 3] \neq 0, (cf. S3 and S3a), then} \item \text{st[S - 7] can be the address where to insert a pass instr. in dynamic introduction of block at next call of procedure S0h (cf. S0i), or 0 when no pass instr. is to be inserted, or negative, when introduction is ready;} \end{itemize}
procedure S01;

if st[S - 7] = 0 then
  begin
    st[S - 7] := P;
    S0(0);
  end
end S01;

comment
is invoked on S4L10, S4L14, S4L24, S4L37.
Unless there has already been reserved an address, it
reserves address $P$ of the object programme for a pass
instruction, which is to be inserted later by
procedure S0h;

A preliminary zero is inserted;
comment

procedure S0k;

if g = 0 then
    begin

if e ≠ 0 then
    begin
    if e > 0 then S0(e + 2126 × 98)
    else S0p
    end if e > 0;
    end if g = 0
else
    begin

S0 (g + 2126 × 98);
S0(e);

end else
    end S0k;


e, if > 0, is the internal equivalent of either a simple variable or formal parameter. If < 0, e is the contra-identifier of a function name. e = 0 occurs only at a call from S1L10. Then nothing is added to the object programme;

Representation of constant e is indicated by bit t of g = 0 000000t01 0...0;

Thus, in the object programme, constant e is subsequent to the extract normally instruction concerned;
procedure SOL;

if g = 0 then SOn(2↑23 × 127)
else
  if e < 0 or 2↑24 ≤ e then
SOLL1: wrong;
else
  begin
    a := e + 2↑24 × 63 + 2↑32;
    e := P + 2↑23 × 127;
    S0e(a);
    S0(0)
  end SOL1;

comment
contra-declares a label, which is either identifier st[S] or, when g ≠ 0, the constant e.
For 127 = 0 001111111 cf. table 2A;

x

x

cf. table 5

x

e + 2↑24 × 63 is the identifier form of a constant label e;
procedure $S_0m(F)$;
begin
$S_0 := S_0 - 1$; $st[S_0] := F$;
if $P + 2 > S_0$ then $S_0g$;
end $S_0m$;

comment
stores word in list $L$ on address $S_0 - 1$;
compare procedure $S_0$
procedure SOn\(P\); 

begin 
 e := P + F; 
 S0e (st[S] + 2↑32); 
 S0(0); 

end SOn;

comment stores contra-declaration of identifier st [S] 
(cf. S4L23) in list I, with P + F as internal equivalent;

Thus a preliminary zero is stored in the place of 
an instruction which is inserted later
procedure S0p;
begin
  a := e;
  e := P + 2*23 \times 31;
  S0e(a);
  S0(0)
end S0p;

comment
contra-declares name of function having no parameters;
which is the contra-identifier of the function name;
procedure S0r;

begin
  g := d or 2^26;
  if g = 0 then
    begin
      a := d or 2^26 * 126;
      if a = 2^26 * 64 or a = 2^26 * 68
         or a = 2^26 * 70 or a = 2^26 * 74
         or a = 2^26 * 80 or a = 2^26 * 82
         or a = 2^26 * 88
        then g := - 2^26
    end
  end

  d := d - g
end S0r;

comment

inverts gression bit of instruction d;

Then operator (cf. table 1A) is not commutative.
When the operator is commutative, the bit d_6 = 0
is not replaced by 1;
procedure S0s;

begin
  a := st[P];
  e := 2↑23;
  if a ≠ 0 then e := e or a;
  if e = 0 then a := a + 2↑23 + 2↑26 x (108-125);
  S0(a);
  SOn(2↑23 x 255 - 1);

end S0s

comment

This procedure makes a contra-declaration of identifier st[S] according to the value p = 0 011111111
listed in table 2A together with an explanation. The procedure is invoked only on S2aL19 and S5aL27 and S2aL19;

Then e = 0 indicates that a is an ar2 instruction (cf. S6aL61) referring to a formal parameter;

Cf. VERIFY and ar2 in table 1D;
Thus either a VERIFY instruction is stored, or an ar2 instruction or 0 is restored;
Then P is again increased, and a zero is stored on address P - 1, while the contra-declaration made refers to the address P - 2
comment

Progressive after-actions of opening symbols (cf. table 1 B).
S1L6;

\[\text{switch sw : = }\]
S5aL1, S6aL1, S3bL1, S9aL1, S7aL1,
S8aL2, S2aL1, S8dL0, S8bL2;

Regressive after-actions of opening symbols.
S1L21;

\[\text{switch SW : = }\]
S5aL3, S6aL5, S3bL2, S9aL2, S7aL2,
S8aL1, S2aL13, S8dL1, S8bL1;

The entries correspond to the after-actions of
\( \text{[ begin : = for go to if procedure switch; } \)
begin

entry: \( P := P0 + 1; \)

\( q := Q0; \ T1 := T0 := Q00-2xh; \)
\( T2 := Q00; \)
\( S0 := L0; \)
\( S := 1 + S0; \)
\( st[S0 - 1] := R := 0; \)
\( \text{go to S6aL11; } \)

end entry;

\textbf{comment}

Compound statement entry.

The constants \( L0, P0, Q0, \) and \( Q00 \) of table 1E are the initial values of the address variables.

When, on S3bL8, the translation finishes, the first word of the object programme is inserted on address \( P0; \)

Then \( \text{Saccent} := 8191, \)
\( \text{st[S0]} := g := \text{mark} := 0. \)

As the opening symbol \textbf{begin} occurs in front of any text, input goes to compound S3.

Procedure \( 30h \) cannot do any harm, for \( \text{st}[S - 2] = 0 \) differs from the internal value 322 of \textbf{begin}.\)
begin

comment
Compound statement S1.
After reading a delimiter \( f \) which is either an operator (table 1A) or a separation or closing symbol (table 1B),
procedure input goes to here;
Then \( c = st[S - 3], d = st[S - 2], e = st[S - 1], \)
\( a = 32 \times \text{rank } r_d \) of delimiter \( d, \)
\( b = 32 \times \text{rank } r_f \) of delimiter \( f; \)
cf. S3aL4 and S5aL5. Then the identifier or constant preceding delimiter \( f \) in the text need not be examined;
Then identifier \( st[S] = e - 2^{32} \) represents an array (cf. S4L22). As this significance of the identifier is nonsens here, it must be non-local, and another local declaration of the identifier may be expected later.
Only exception: identifier is actual parameter, representing array. Then \( g : = 0 \) does no harm;
Then delimiter \( f \) is preceded in the text by the 0-identifier (there is no identifier and no constant);
Operator \textbf{not} is preceded by an identifier or constant.
\( S1L2; \)

if \( g < 0 \) then \( g : = 0; \)

if \( e = 0 \) and \( g = 0 \) then \( S1L3; \)
if \( f \neq 32 \times 5 + 2^{16} \times 97 \)
then \( S1L4; \)

\( S1L2a: \) wrong;

\( S1L3: \) if \( a < 128 \) then\n\( S1L3a: \) wrong;

if \( f = 32 \times 3 + 2^{16} \times 72 \)
then \( \text{go to input}; \)
\textbf{S1L3: wrong;}
\begin{align*}
f & : = 32 \times 3 + 2 \times 26 \times 96; \\
\end{align*}

\textbf{S1L4: if} b < a \textbf{ then go to S1L26;}

\textbf{S1L5: if} \text{Saccent} = S \textbf{ then go to S1L21;}

\textbf{S1L6: if} a = 320 \textbf{ then go to sw [d-319];}

\textbf{S1L7: if} \text{Saccent} + 2 < S \textbf{ then} \text{SCd;}

\textbf{S1L8: a : = d or 2 \times 23 \times 5;}

\textbf{S1L9: d : = d - a + g;}

\textbf{S1L10: if} \text{Saccent} + 2 = S \textbf{ then go to S1L11;}
\begin{align*}
g & : = a; \\
a & : = e; e : = c; c : = a; \\
\end{align*}

\textbf{S1L11: comment}
for a dummy statement. Then the after-action of \text{d} with \text{f = semi-colon or end} is considered to be regressive; for operator \text{not};

\textbf{X}
Then an identifier or constant has been omitted; \textbf{X}
In a = - b etc. - is treated this way.
\textbf{X}
\textbf{S1L2;}
Then reading continues and \text{d} is a regressive operator or \text{opening symbol.}
Otherwise reading is interrupted:
Then \text{d} is either a regressive operator, or an opening symbol which is regressive with respect to the after-action with \text{f} (cf. \text{S5aL5} and \text{S3aL4}).
In general, however, an operator or opening symbol \text{d} which is translated first after interrupting the reading, is progressive:
for progressive after-actions of opening symbols;
Then object programme must be provided with storing and extracting a partial result;
Then \text{a} is either \text{= 0} or \text{a = 0 \ldots 0 and c a programme const. whose representation is indicated by the bit t. For g and e it is analogous(cf. S1L26 and S1L29); which is thus associated to e instead of c; Otherwise object programme must be equipped with filling the accumulator:;}
c and e interchange their values;
Thus previous value of \( e \) is restored.

**S1L10, S1L25:**

Then instruction \( d \) is ready and \( e \) is a programme constant to be used by instr. \( d \);

Then \( e \) is the internal equivalent of either a simple variable or formal parameter.

Otherwise \( e - 2^{32} \) is an identifier (cf. S4L22) which, in this position, can only be the name of a function having no parameters; ;

Otherwise \( d \) is an operator not or - such as does not, in operation time, require to store a previous partial result: ;

Compare S1L13;

S1L12;

for storing previous partial result;

Thus a contra-declaration of identifier \( c - 2^{32} \) is made according to \( p = 0 \ 000011111 \) which constant is listed in table 2A together with an explanation;

For, in operation time, instruction \( d \) must extract a partial result;

Thus the gression bit \( d_6 \) is inverted. When, for example, function \( - p \times q + ... \) is translated, then operator - is regressive according to the general rule. Thus \( p \times q \) is translated first. Before, in operation time, function may be invoked, the object programme must at first store the partial result \( \text{accu} \) which is \( p \times q \). Therefore the minus instruction is to be translated in the form \( \text{accu} := \text{accu} - (\text{partial result}) \) which is progressive instead of regressive.
S1L14: e := e - 2^26 × 63 + 
(d or 2^26 × 127);  
go to S1L16;
S1L15: S0(d);  
S1L16: S0(e);

S1L17: Saccent := S := S - 2;
S1L18: S0a;
S1L19: if b < a then go to S3L5;

S1L21: if a = 320 then go to 
SW[d - 319];
S1L22: if d = 32 × 5 + 2^26 × 97 
then go to S1L23;
e := take inversion;

go to S1L16;
S1L23: if d = 32 × 3 + 2^26 × 96 
then go to S1L24;
e := take complement;

go to S1L16;

comment
S1L12;

S1L11;
S1L14, S1L22, S1L23;
Thus instruction or programme constant is stored 
in the object programme.
S3bL14, S7aL18, S8aL3, S9aL2;
Operator st[S] has been translated and will be 
overwritten now;
cf. S1L1;
Then translator again proceeds to reading. 
Otherwise translation continues (cf. S1L4).
S1L5, S1L3;
for regressive after-actions of opening symbols 
(cf. S1L6).
S1L12;

That code instruction of table 1E is the regressive 
version, and extract inversion of table 1C is the 
progressive version of operator not;
S1L22;
That code instruction of table 1E is the regressive 
version, and extract complement of table 1C which 
has been introduced at the end of S1L3 is the 
progressive version of operator --, when -- is preceded 
by the 0-identifier;
S1L23;
That procedure sets the gression bit \( d_6 \) which is still \( = 0 \), to \( 1 \), for operator \( d \) is regressive; If that is a programme constant, the indication of its representation is already recorded in \( d \); \nS1L4; 
with either \( g = 0 \), or \( g = 00000001 \ 0...0 \), the bit \( t \) indicating the representation of \( e \) which is a programme constant (cf. S1L29).

If, at the next action of this compound \( S_1 \), operator \( st[S - 2] \) just listed in \( L \) is found to be progressive, the g-part is removed from it and replaced by the next value of \( g \), on S1L9.

S3L5, S5aL6, S6aL11;
cf. table 4C.

S3aL4, S3cL6, S5L9, S6aL31, S6aL61, S8L2, S8dL8, S9L4;
If the translator proceeds to S4L23 or S4aL2, this 0 is replaced by something else.

S4L21;
After reading a constant, procedure input goes to compound S4a with \( g = 000000010...0 \) in which bit \( t \) indicates the representation of the constant \( f \) read.

On S4L22, occasionally an art instruction is assigned to variable \( g \) as value;
begin

S2L1: f := 326
S0h;
S2L2: S0c(mark);
go to S3L3;

end S2;

comment

Compound statement S2.
Pre-action of the opening symbol if;
Cf. if in table 1B;
Cf. S3L2;

Then, for translating an if-expression, there is listed in L:
st[S - 4] = mark (cf. table 4C),
st[S - 3] = 0,
from 1st after-action: minus address P'
where to store a test instruction,
from 2nd after-action: plus address P''
where to store a pass instruction.
st[S - 2] = value 326 of if,
st[S - 1] = 0.

Signal mark is set to 0. The listed value,
st[S - 4], is re-introduced on the 1st and 2nd after-actions of if;
comment

Compound statement S2a.

After-actions of the opening symbol if.
Delimiter f is either then or else or the
closing symbol of a conditional expression or
- statement E. On $S_{4L4}$, the value 0 or $-P'$ or
+$P''$, mentioned in compound S2, has already been
assigned to variable c.

Progressive:
The expression, preceding delimiter f, is a
single identifier or constant i
$S_{4L6}$;
being the value of mark retained by compound
statement S2;
Then the 2nd or 3rd after-action of if is
beginning, f being else or the closing symbol,
and, in addition, d is = 1 or 2.
Then i may be a label.
Otherwise i is no label;
Thus object programme is equipped with filling
the phantom accumulator;
$S_{4L2}$;
Then i is a label (or a formal parameter that
may only represent a label), and E is a
designational expression.
Otherwise expression E is an actual parameter;
Then i is the constant e.
Otherwise i is an identifier ;
Then $i = e - 2^{32}$ is found to be no simple
variable and no formal parameter (cf.$S_{4L22}$).

S2aL1: d := st [S - 4];

S2aL2: if c * d ≠ 0 then go to S2aL4;

S2aL3: S0k;

go to S2aL20;

S2aL4: if d = 2 then go to S2aL7;

S2aL5: if g ≠ 0 then go to S2aL11;

S2aL6: if e < 0 then go to S2aL9;
S2aL7: \textbf{if} \ e - 2^{23} \times 505 > 0 \ \textbf{then go to S2aL18};

S2aL8: \ g : = e + 2^{23} + 2^{26} \times (110-63); \textbf{which is a verify instruction} (\textit{cf.} \textbf{table 1D}).

In operation time, the bit \( g_9 = 1 \) of instruction \( g \) makes the test on S10L13 fail, and the interpreter proceeds to S10L140. There the instruction \( X \) whose formal parameter represents the actual parameter in which expression \( E \) is contained, is examined. If \( X \) is no jump instruction, then the formal parameter \( i \) to which the verify instruction refers, cannot represent a label so that the verify instruction is to be interpreted as the extract normally instruction referring to \( i \).
S2aL6;
S2aL18;
being the contra-identifier of \( i \) \textbf{(cf. S4L23)};

The contra-declaration \( g, e \) to be stored now is made according to \( p = 0111111111 \) which constant is listed in table 2A together with an explanation. It refers to the address \( x = P - 1 \), where the object programme contains either the mentioned verify or extract normally instruction or 0.
S2aL5;

\textbf{Comment:}
Identifier \( i \) is found to be a simple variable or formal parameter with the internal equivalent \( e \); Then \( i \) is a simple variable, and an extract-normally instruction is added to the object programme. Otherwise \( i \) is a formal parameter;
Then \( st[P - 2] \) is the extract normally instr. which, in operation time, extracts the programme constant \( st[P - 1] = 1 \);

Then constant \( i \) is not suitable for being a label. Constant \( i \) may occur as a label;

which is the contra-identifier of a constant label \( i \);

The contra-declaration \( g, e \) to be stored now is made according to \( p = 1 \times 100000000 \) which constant is listed in table 2A together with an explanation. It refers to the address \( x = P - 2 \), where the object programme contains the extract normally instruction. The internal equivalent \( e \) is negative.

thus the contra-declaration is stored now;

Regressive:

The expression I preceding delimiter \( f \) is non-trivial (no identifier and no constant).

Then expression I cannot be a switch designator (cf. S2aL4) and has already been translated. Otherwise \( I = i[E] \) is either a subscripted variable or a switch designator. The subscript \( E \) has been translated in compound S6a;

Then \( st[P] \) is either the ar2 instruction referring to \( i \), in which case \( i \) has been found to be either an array identifier or a formal parameter, or 0 (cf. S6aL61);
S2aL16: \textbf{if} \ d = 1 \ \textbf{then} \ \textbf{go to} \ S2aL19;

S2aL17: \textbf{S01;}
\textbf{go to} \ S2aL20;
S2aL18: \textbf{S0k;}
\textbf{go to} \ S2aL10;
S2aL19: \textbf{S0s;}
S2aL20: \textbf{mark} : = \ d;

S2aL21: \textbf{e} : = \ P;
S2aL22: \textbf{if} \ c < 0 \ \textbf{then} \ \textbf{go to} \ S2aL28;
S2aL23: \textbf{if} \ c \neq 0 \ \textbf{then} \ \textbf{go to} \ S2aL31;

S2aL24: \textbf{e} : = \ - \ e;
S2aL25: \textbf{st}[S - 3] : = \ e;
S2aL26: \textbf{S0(0);}
S2aL27: \textbf{go to} \ S1L28;
S2aL28: \textbf{if} \ f \neq 326 \ \textbf{then} \ \textbf{a} : = \ e - 1;
S2aL29: \textbf{st}[-c]: =e+1+2\times 23+2\times 26\times 111;
S2aL30: \textbf{if} \ f \neq 326 \ \textbf{then} \ \textbf{go to} \ S2aL32; \textbf{Then the separation symbol else is not present;}
\textbf{go to} \ S2aL25;
S2aL31: \textbf{st}[c]: = e+2\times 23+2\times 26\times 121;
S2aL32: \textbf{S} : = \ S - 2;
\textbf{go to} \ S8aL3
\textbf{end} \ S2a;

\textbf{comment}

Then 1 \ [E] \ is contained in an actual parameter.
1 \ [E] \ is a switch designator thus 1 a switch identifier:
S2aL4;
Label or switch identifier is contra-declared;
S2aL7;
S2aL16;
S2aL3, S2aL11, S2aL12, S2aL14, S2aL17;
At the after-actions with \textbf{f = then} \ or \textbf{else} signal
mark is restored to the value retained by compound S2;

for the 2nd after-action;
for the 3rd after-action.
1st after-action:
S2aL30;
At the next after-action, this 0 is replaced by a
test or pass instruction;
S2aL22;
Cf. \textbf{else} \ in \text{table 1B;}
Cf. \text{test in table 1D;}
S2aL23;
Cf. \text{pass in table 1D.}
S2aL30;
Thus, at the last after-action of \textbf{if}, mark is set to 0
comment

Compound statement $s_3$.
Pre-action of the opening symbol \texttt{begin};
Cf. \texttt{begin} in table \ref{table:1B};
Cf. D in table \ref{table:2};
Perhaps opening symbol \texttt{begin} immediately follows a block head. Then procedure S0h completes the dynamic introduction of the block.
S2L2, S6L9;
S7L2, S9L10;
S1L19, S5L4, S6L19;
After pre-action of opening symbol, reading continues.

For translating a compound statement there is listed in L:
\[ st[s - 3] = 0, \]
\[ st[s - 2] = \text{value 322 of } \texttt{begin}, \]
\[ st[s - 1] = 0 \]
begin

S3aL1: D := D or f;

if (D or 2128) = 0 then go to S3aL4;

S3aL2: S0a;

if d ≠ 322 then
S3aL2a: wrong;

S3aL3: if c ≠ 0 then go to S3aL4

st[S - 1] := q;
S0c(P);
S0(0);
st[S - 1] := T;
S0c(322);
S3aL4: Saccent := B;
go to S1L28;

end S3a;

comment

Compound statement S3a.
After reading a declarator or specifier (table 2),
procedure input goes to here;
Thus declaration is replaced by its logical product
with f;
Then f is a specifier or specifying declarator;
for extracting opening symbol from list L;
Then declarator occurs in the wrong place;
Then block has already been prepared in L by a
previous declarator.
First declarator of a block:

S3aL1, S3aL3, S3bL2, S4L30, S5L2;
For translating a block there has been listed in L:
st[S - 7] = 0, which 0 may be replaced through
procedure S0i by an address to inform procedure S0h,
st[S - 5] = q', being the highest address occupied by
a local variable,
st[S - 4] = address P' where the block has its object
programme beginning,
st[S - 3] = T', being the highest address in the list I,
occupied by any identifier declared in the block,
st[S - 2] = value 322 of opening symbol begin,
st[S - 1] = 0
begin

S3bL1: S0h;
SO\textup{n}(2^{123} \times 31);

S3bL2: if \( f \neq 330 \) then go to S3bL3;
\hspace{1em} if \( D > 0 \) then go to S3aL4;

S3bL2a: wrong;

S3bL3: T_1 = T;
S3bL4: D = 2^{123} \times 1023;
S3bL5: if \( f = 332 \) then go to S6aL11;
\hspace{1em} if \( f \neq 322 \) then
S3bL5a: wrong;

S3bL6: if \( c \neq 0 \) then go to S3bL9;

comment

Compound statement S3b
After-actions of opening symbol begin
Delimiter \( f \) last read is either comma or semi-colon or end.
Progressive:
S1L6;
A block head may be followed by a procedure statement consisting of a single procedure identifier (cf. S3L2);
A contra-declaration of the procedure identifier is made according to \( p = 0 \ 00001111 \) which constant is listed in table 2a together with an explanation.
Regressive:
S1L21;
Then \( f \) is no comma (cf. table 1B);
Then comma occurs in list of identifiers to be declared;
Comma occurs in a block as separator of statements.
S3bL2;
This bound is observed by procedure S0f;
as happens also on S3L1;
for reading next declaration or statement of compound statement or block, which is subsequent to semi-colon \( f \);
Then a declaration or statement of a compound statement or block is followed by a delimiter, differing from semi-colon and end.
After-action of begin with end:
On S1L1, st[S - 3], as left by either compound S3 or S3a, has been assigned to variable \( c \);
Then a block is closing.
A compound statement is closing;


\[ e : = \text{QO} - 2 \times h; \]

which is the highest value \( T \) can have;  

\[ \text{if} \ S = S0 + 3 \ \text{then go to S3bL13}; \]

Then the text is a compound statement instead of a block. Labels may occur in it.

S3bL7: \( \text{if} \ S > S0 + 3 \ \text{then go to input}; \)

input, in this comment situation, looks for the next delimiter \( f \) which is equal to either semi-colon or end or else, then going with \( f \) to S1L17. Then the delimiter preceding begin in the list L, is going to after-act with \( f \).

The whole text has been read;

Thus last word of object programme is 0, but may be a jump to any stop of the machine;

Thus 1st word of object programme is address where +

working space begins;

Object programme is ready!

S3bL6;

Now \( \text{st}[S - 4] \) is the delimiter preceding begin in the list L. For \( c = q' \) etc. cf. compound S3a;

Thus, in object programme of block, an adjust instruction is the first word;

Thus in object programme of block, a restore instruction \( \times \) (cf. table 1D) is the last word.

S8dL7;

Thus value of \( a \), which was present when first declarator of block was read, is restored.

S3bL6;

On S3bL3, the value of \( T \) was also assigned to \( T_1 \).

S3bL15, S3bL18, S3bL20, S3bL34;

Then the contra-identifier occurring next on the addresses \( a+2, a+4, \ldots \) e is looked up.
All contra-identifiers have been considered;
Then in the list it is the highest address,
occupied by an identifier which is local to the
translated block;
Cf. table 1E;

Then it is a block that closes,
st[S - 3] having been made negative by procedure
SOh after reading the block head.
A procedure is closing:
Thus semi-colon f of compound S8d is restored;
as ultimately happens too with a block
(cf. S3bL7).
S3bL14;
Then st [a] is a normal (positive) identifier.

Then identifier f corresponding to contra-
identifier st [a] is looked for on the addresses
T+2, T+4, ...T2;
being the internal equivalent of the contra-
identifier;

Then procedure SOf has found identifier f on
address c, d = st[c - 1] being the internal
equivalent (cf. table 2) of f. Then contra-identifier
st [a] can be satisfied.
Identifier f has not been found so that the
contra-identifier remains unsatisfied;
Thus the value of e as it was on S3bL16, is
restored.

T1 := T := e;

T2 := Q00;
if st [S - 3]<0 then go to S3bL7;

f := 332;
go to S114;

S3bL15: a := a + 2;
f := st [a] + 2*32;
if f < 0 then go to S3bL14;
S3bL16: T2 := e; SOf;

b := st[a - 1];

S3bL17: if d > 0 then go to S3bL21;

S3bL18: e := T2;
if \( S > S_0 + 3 \) then go to S3bL19;

if \( b < 0 \) then go to S3bL14;

S3bL18a: wrong;

S3bL19: \[ \begin{align*}
    st[a - 1] &:= st[e - 1]; \\
    st[a] &:= st[c]; \\
    st[e - 1] &:= b; \\
    st[e] &:= f + 2^{13}; \\
    a &:= a - 2; \\
    e &:= e - 2;
\end{align*} \]

Thus 2 declarations are interchanged; The non-satisfied contra-identifier is considered to be no longer local. Thus \( e \) is again the highest address where a local identifier is listed in \( I \); S3bL17; which is used on S3bL33; being address \( + 2^{13} \times \text{rank} \); being the address where either an instruction or the main word of a parameter key is to be inserted. For internal equivalent \( b \) cf. table 2A; Then \( b = 0 \) indicates, that the contra-declaration refers to the key of an actual parameter that is a single identifier. There must be formed either \( d = \text{main word} \), and
S3bL23: if \( b \neq 0 \) then go to S3bL35;

S3bL24: if \( 2 \times d < 0 \) then go to S3bL29;

S3bL25: if \( 4 \times d < 0 \) then go to S3bL26;

S3bL26: if \( 8 \times d < 0 \) then go to S3bL30;

S3bL27: \( d := e \text{ or } (2^{13} - 2^{13}); \)
\( e := e + 1 - d; \)
\( d := d + 2^{13} \times 57; \)
go to S3bL31;
S3bL28: \( e := g - e - 2^{13} \times 2^{25} \times 126; \)
go to S3bL31;

S3bL29: if \( (d \text{ or } 2^{13}) = 0 \) then
\( d := e + 2^{26} \times 3; \)

S3bL30: \( e := 0; \)

S3bL31: \( st[e - 1] := e; \)

Comment:
e = by-word of the key (cf. table 3) of an actual parameter that is an identifier or a constant, or \( d = \) instruction;
for a constant parameter or an instruction.
Parameter is an identifier:\;
Then parameter represents either a formal parameter or a simple variable (cf. table 2);
Then parameter represents an array;
Then parameter represents a label or switch.
Parameter represents a procedure:\;
being \( 2^{13} \times \) rank of procedure identifier;
being \( 1 \) + address where procedure has its object programme beginning;
S3bL25;
Cf. arr in table 1D;
As procedure SOf assigned an arr instruction (cf. S6L8) to variable \( g \) as value, \( e \) is a difference of addresses now.
S3bL24;
Then actual parameter represents a formal parameter
S3bL26;
Thus, for a parameter representing a simple variable or formal parameter or label or switch, the by-word is cleared, which is facultative.
S3bL27, S3bL28;
Thus by-word is stored.
The key main word of a parameter, representing a simple
comment

variable or array or label or switch i, is equal to
the internal equivalent of i.

S3bL47;
Thus key main word or instruction is stored.
S3bL42, S3bL43;
Cf. S3bL24;

Any identifier has been defined twice in the same
block or , or in the same procedure outside the
blocks contained in it;
Thus the value of T1 as it was on S3bL14, is
restored;
Ditto;
Satisfaction of contra-identifier st[a] is ready.
S3bL23;

Then g is the address taken from the declaration.
Now the kind of the object referred to by the contra-
declaration (cf. table 2a) is examined. It must be in
accordance with that of the object referred to by the
corresponding declaration (cf. table 2);
Then a constant occurring in an actual parameter (or
being an actual parameter) is found to be used as a label
too - the declaration of which having been found;

Then st[c] is a verify or extract normally instruction
or 0, occurring in the object programme of an actual
parameter (cf. S2aL6);

Then st[c] is a VERIFY or nr2 instruction or 0,
occurring in the object programme of an actual
parameter (cf. S2aL15);
S3bl39: if \( 8 \times b < 0 \) then go to S3bl45; Then a jump instruction is to be inserted in the object programme of a switch or go to statement, identifier \( f \) having been contra-declared on S8aL2 for example.

In the object programme of a function designator or procedure statement, the 1st word must be adjusted, identifier \( f \) having been contra-declared on S5aL6 or S1L13 or S3bL1 for example;

Then identifier \( f \) may only have the significance of a procedure identifier.

Identifier \( f \) has been found in the significance of a formal parameter.;

Cf. extract normally and prostat in tables 1 C and 1 D;

S3bl40;

A procedure statement or function designator invokes nonsense;

A function designator or procedure statement having actual parameters, invokes a function or procedure having no formal parameters.

S3bl38;

Then identifier \( f \) is a formal parameter;
A subscript has been attached to a simple variable;
Then $f$ is an array identifier;

Then $st[c]$ is replaced by a jump instruction;

A subscript has been attached to a procedure identifier.

S3BL37;

Then identifier $f$ is either a simple variable or a formal parameter and $st[c]$ is not changed;

An array identifier occurring within an expression is not followed by a subscript;

Then $f$ is a switch identifier or a label;

Identifier $f$ is a function or procedure name.
The function designator or procedure statement has no actual parameters:

S3BL41;

A function designator or procedure statement having no actual parameters, invokes a function or procedure having formal parameters.

S3BL41;

Cf. table 1E;

On address $c$, a code instruction is stored which, in operation time, jumps to address $g$ where the object programme of the function or procedure begins.

S3BL39;
S3bL45: \texttt{if } b = d \texttt{ then go to S3bL46;}

\[ g : = 2 \times 26 \times 122; \]

\texttt{if } d = 2 \times 26 \times 63 \texttt{ then go to S3bL47;}

S3bL45a: wrong;

S3bL46: \[ g : = 2 \times 23 + 2 \times 26 \times 122; \]

S3bL47: \[ d : = e + g; \]

\texttt{go to S3bL32;}

S3bL48: \[ g : = \texttt{if } 2 \times b < 0 \texttt{ then } 2 \times 23 + 2 \times 26 \times 109 \texttt{ else } 2 \times 23 \times 89; \]

\texttt{go to S3bL47}

\texttt{end S3b;}

\textbf{comment}

Then \( f \) is a label or switch identifier.
Identifier \( f \) is a formal parameter;
Cf. jump in table 1D;

Then \( f \) is indeed a formal parameter;
nonsense, referred to in a designational
expression.
S3bL42, S3bL43;
Cf. jump in table 1D.
S3bL40, S3bL44, S3bL45, S3bL48;

for storing instruction or key main word.
S3bL36;

Cf. Verify in table 1D and key main word of
constant parameter in table 3;
begin

S3cL1: \textbf{if} (D or 2124) \neq 0 \textbf{then go to} S3cL2;

\textbf{if} e = 0 \textbf{then}
S3cL3a: wrong;
S3cL4: \textbf{f} := \text{st}[8];

S3cL5: a := P + R + 2123 \times 127;
S0e(f);
S3cL6: S0 (a + R + 2123 + 2126 \times 119);
\textbf{go to} S1L28;
S3cL7: \textbf{if} e < 0 \textbf{or} 2124 < = e \textbf{then}
S3cL7a: wrong;
S3cL8: \textbf{f} := e + 2124 \times 63;
\textbf{go to} S3cL5
\textbf{end} S3c;

\textbf{comment}

Compound statement S3c.
Colon goes to here from input;

Then colon declares the label preceding it in the text.
Otherwise pattern D contains the type indication of an array declaration, colon being the separation symbol between a lower and upper bound;

Cf. colon in table 1B;
S3cL1;
Cf. S3L2;
Then label is an integer (compare procedure S0L);

Label has been omitted;
Cf. S4L23.
S3cL8;
Cf. label in table 2;
Thus declaration of label is listed;
Cf. restore in table 1D;
S3cL3;
Cf. table 5;
being the internal value of constant label f;
begin

comment

Compound statement S4.
After reading an identifier f, procedure input goes to here.
Then identifier f must be looked for in the list I.
Identifier f must be declared or specified (cf. table 2);
Then identifier f is declared a simple variable;
array identifier;
switch identifier;
procedure identifier;

Then f is either a value parameter, or is to be specified.

Identifier f is a formal parameter to be listed in I:

S4L1: if \( D < 0 \) then go to S4L22;

S4L2: if \( 2 \times D < 0 \) then go to S4L9;
S4L3: if \( 4 \times D < 0 \) then go to S4L11;
S4L4: if \( 8 \times D < 0 \) then go to S4L37;
S4L5: if \( 16 \times D < 0 \) then go to S4L24;
S4L6: if \((D \text{ or } 2^{13}) \neq 0\)
then go to S4L32;

S4L7: \( P := P - 1; \)
e := st[P];
S0(D); S0(e);
then go to S4L32;

S4L8: e := q + 2^{126} \times 63;
q := q - 2;
go to S4L19;

S4L9: if \( 8 \times D < 0 \) then go to S4L18;

S4L10: S01;
go to S4L15;
S4L11: a := st[S - 1];
S4L12: if \( 8 \times D < 0 \) then go to S4L16;

Then st[P - 2] = D, and st[P - 1] = -2^{13} \times \text{rank of procedure (cf. S4L26)};

Relative addresses q + 3 and q + 2 are reserved for the main — and by-word of the parameter key.

S4L2;

Then variable is not own.
Variable is own;
Cf. S4L24;
S4L3;

Then array is not own.
Array is own;
S4L13: if c ≠ 0 then go to S4L15;

S4L14: S0i;
    st[S - 1] := P;

S4L15: e := P + D + 2*29;

S0(0);

go to S4L20;

S4L16: if c ≠ 0 then go to S4L18;

S4L17: S0h;

   st[S - 1] := q;

S4L18: e := q + D; q := q - 1;

S4L19: e := e + R;

cmment

Then f is not the 1st identifier of the list.
1st identifier of the list:

Cf. S4L24;
Then st[S - 1] is no longer = 0.
Compound S6 will insert pre-value [a] of a = 1st array
of declaration (cf. explanation)
on address P.
S4L10, S4L13;
Then the bit e3 of the internal equivalent = 1, x
as it is in the case of a variable or array being
not own;
A place for the simple variable or the pre-value of
the array is reserved in the object programme;
S4L12;
Then f is not the first identifier of the list.
1st identifier of the list:
Thus the space of own variables and - arrays and of
the object programmes of procedures and switches,
which eventually precede the non-own array declaration,
is bridged by inserting a pass instruction in the
dynamic introduction of the block;
The relative address q is reserved for the
pre-value [a] (cf. S4L14), that is an internal
(non-declared) variable of the object programme.
S4L9, S4L16;
Thus internal equivalent e of identifier f
contains the address reserved.
S4L8;
S4L15;
S4L20: \( T_1 := T; \)
    \( T_2 := st[S - 3]; \)
    \( S0e(f); \)
    \( S0f; \)
    \textbf{if} \( d > 0 \) \textbf{then go to} S3bL33a;

\textbf{comment}

That is the address \( T' \) mentioned in compound S3a;
Thus declaration is listed in I;

Then identifier \( f \) is defined twice in the same
block or procedure heading.

\textbf{S4L21:} \( T_2 := Q00; \)
\textbf{go to} S1L29;

\textbf{S4L22:} \( S0f; \)

\textbf{S4L23:} \( st[S - 1] := e; st[S] := f; \)
\textbf{go to} input;

Because of the possible necessity to make a
contra-declaration, identifier \( f \) remains available
on address \( S \), until the next delimiter is stored
there in the list \( L \).

\textbf{S4L24:} \( S0i; \)

\textbf{S4L25:} \( e := P + R + D; \)

\textbf{S4L26:} \( R := R + 2^{13}; \)

As the object programme of the procedure which is
going to be translated now, is to be passed by the
dynamic introduction of the block, a location is
reserved for the pass instruction needed, unless this
reservation has already been made earlier on
\textbf{S4L10, S4L14, S4L24 or S4L37};

Thus the address \( P \) where the procedure has its
object programme beginning, is contained in the
internal equivalent \( e \) of procedure identifier \( f \);

\( R \) is cleared in compound entry, and decreased on
\textbf{S8dL3}. \( r = 2^{-13} \times R \) is the rank of the procedure,
\( r - 1 \) is the rank of the procedure identifier;
comment

Cf. table 1B;

The patterns of eventual formal parameters are
inserted between instruction X and constant -R so
that the latter must be shifted then (cf. S4L7);
Cf. procedure in table 1B.
S4L39;

Cf. S4L20;
For translating a procedure, there has been listed
in L:

\[ st[S - 5] = \text{value } q' \text{ of } q \text{ which was present when} \]
declarator procedure was read,
\[ st[S - 4] = 2 + \text{address } P' \text{ where instruction } X \text{ has} \]
been stored,
\[ st[S - 3] = \text{address } T' \text{ where procedure identifier } f \]
has been listed in I,
\[ st[S - 2] = \text{value 327 of opening symbol procedure,} \]
\[ st[S - 1] = 0. \]
to be compared with the information listed by
compound S3a.
S4L6;

being the number of the formal parameters
(cf. \( P' \) in S4L31);

On that address the 1st parameter identifier is
listed in I;
Thus identifier \( f \) is looked up in parameter list;

S4L27: S0(X);
S4L28: S0(-R);
S4L29: \( st[S - 1] = q; \)
\[ q := Q0 - 1; \]
\[ d := 327; \]
S4L30: S0c(P);
\[ st[S - 1] = T; \]
S0c(d);
S0c(f);
S4L31: go to S3aL4;

S4L32: \( e := P - st[S - 4]; \)
T2 := \( R + e + e; \)
S0f;
if d < 0 then
SL4L33: wrong;

SL4L34: e := P - 1 + (T - c) ÷ 2;

SL4L35: st[e] := st[e] or D;
SL4L36: go to SL4L21;
SL4L37: S01;
SL4L38: D := 2123 × 1023;
    T1 := T;
    q := q - 1;

SL4L39: d := 328;
    go to SL4L30;

comment

Then specified identifier does not occur in the parameter list of the procedure;
On that address the pattern of the parameter to be specified, is listed in the object programme;

SL4L4;
Cf. SL4L24;

Cf. S3bL 3 and 4. The expressions of the switch list must be translated now;
This is compensated by S8bL9.
As there is supposed that the declarations of all local variables and arrays are precedent to those of switches and procedures, q + 1 is the lowest address, occupied by a local variable (namely the internal variable which is used by the adjust and restore instructions) the switch list may contain deliberate statements (jumping or not) as its elements;
Within the list L, this 0 will be preceded by the pass instructions leading to the object programmes of the switch list elements, until the list of these pass instructions can be stored in the object programme of the switch (cf. S8bL7);

For translating a switch, there is to be listed now in L:
st[S - 4] = address where the switch list element to be read now has its object programme beginning,
comment

st[S - 3] = address where switch identifier is
listed in I, the internal equivalent being stored
in I on S8bL10,
st[S - 2] = value 328 of opening symbol switch,
st[S - 1] = 0

end S4;
begin

comment
Compound statement S4a.
After reading a constant \( f \), input goes to here with \( f' \) and \( g = 2^{23} \times 0000000001 \) in which the bit indicates the representation of \( f \): \( t = 1 \rightarrow f \) is real, \( 0 \rightarrow \) integral or boolean;

Then a constant occurs within a list of identifiers to be declared;

The delimiter which is subsequent to the constant is read

S4aL1: \textbf{if} D > 0 \textbf{then}
S4aL1a: wrong;

S4aL2: st[S - 1] := f;
\textbf{go to} input;

end S4a;
begin

S5L1: S0h;
    if D < 0 then go to S5L3;

if d ≠ 327 then
S5L1a: wrong;

S5L2: D := 2*23 * 30;
    go to S3aL4;

S5L3: if g > 0 then
S5L3a: wrong;

S5L4: if Saccent < S then S0d;

S5L5: Saccent := 8191;
    f := 320;
    if e = 0 then go to S3L5;

S5L6: SOn(2*23 * 63);

comment

Compound statement S5.
Pre-action of the opening symbol (;
Cf. S3L2;

Then either a function designator or a procedure statement or an expression enclosed within parentheses is going to be translated.
A formal parameter list is beginning.
Procedure S0h has assigned the value of opening symbol procedure (cf. table 1B) to variable d;

Then (occurs in an identifier list of a block head;
S8dL8;
Cf. table 2;

The parentheses of a formal parameter part are not listed in L.
S5L1;

Then ( is preceded by a constant;
Compare this with S1L7, where the operator st[S - 2] which is on the point of being translated, has already been listed in L;
as happens too on S6aL11;
Cf. ( in table 1B;
Then an expression enclosed within parentheses is beginning in the text.
function designator or procedure statement:;
Thus identifier f is contra-declared according to p = 0001111111, which constant is listed in table 2A together with an explanation;
S5L7: S0m(0); S0m(0);

S5L8: S0c(P);
    S0(0);
    S0c(f);
S5L9: s[S - 3] = P;
    mark := 1;
    go to S1L28;

end S5;

comment
The keys of the actual parameters will be listed in L, until all actual parameters will have been translated (cf. S5aL17);

S5aL15;

cf. table 4C;
For translating a function designator or procedure statement, having actual parameters, there has been listed in L:
s[S - 4] = address where to store the key address of the function designator or procedure statement (cf. S5aL23).
s[S - 3] = address where the object programme (it may be empty) of the next actual parameter begins,
s[S - 2] = value 320 of (,
s[S - 1] = 0
comment

begin

Compound statement S5a.

After-actions of the opening symbol (, Delimiter f read last is either a comma or ).

On S1L1 the value of st[S - 3], as left by compound S5, has already been assigned to variable c.

Progressive:

Delimiter f is preceded in the text by p which is either a string or a trivial expression (identifier or constant).

S1L6;

Then p is an actual parameter.

p, though enclosed within parentheses, is no parameter;

S5aL1: if c ≠ 0 then go to S5aL7;

S5aL2: S0k;

Thus the object programme is equipped with filling the accumulator.

Regressive:

Delimiter f is preceded in the text by a non-trivial expression p.

S1L21;

Then p is an actual parameter.

p is no actual parameter;

S5aL3: if c ≠ 0 then go to S5aL24;

if f = 321 then go to S5aL5;

Then expression p is enclosed within brackets (cf. S6L12).

S5aL23;

S5aL4: if f ≠ 320 then

S5aL4a: wrong;

Actual parameter, or expression intended to be enclosed within parentheses (or brackets), is followed by a wrong delimiter.

S5aL3, S6aL26, S6aL42;
S5aL5: \texttt{Saccent : = S : = S - 2};
S5aL6: \texttt{go to S1L27};

S5aL7: \texttt{if g > 0 then go to S5aL10;}
\hspace{1cm} \texttt{if c > 0 then go to S5aL8;}

\hspace{1cm} \texttt{d : = -c; e : = P;}
\hspace{1cm} \texttt{go to S5aL14;}

S5aL8: \texttt{c : = st [S] + 2132;}

S5aL9: \texttt{d : = T;}
\hspace{1cm} \texttt{go to S5aL13;}

---

\texttt{comment}

Delimiter \texttt{st[S22]} is regressive, for it is precedent to parentheses or brackets in the text. If its rank should be not greater than that of the delimiter to be read next regressivity is detected on S1L5.

S5aL1;

Then \texttt{p} is a constant, \texttt{e}, and \texttt{g = 0 000000001...0}, bit \texttt{t} indicating the representation of \texttt{e};

Then \texttt{p} is an identifier, \texttt{st [S]} (cf. S4L23), \texttt{c} being the address listed in \texttt{L} on S5L8.

\texttt{p} is a string, \texttt{c} being the complement of the mentioned address (cf. compound S5b):

Thus the object programme of the string occupies the addresses \texttt{d, d + 1, ... e - 1.}

\texttt{d, e} becomes the string key (cf. table 3).

S5aL7;

As the parameter may be a label or a procedure- or switch identifier declared later in the text, the parameter key may (and possibly can) not yet be formed. Thus a contra-declaration of identifier \texttt{st [S]} will be made according to \texttt{p = 0 000000000} which constant is listed in table 2A together with a general explanation;

The contra-declaration is to point to the address \texttt{x} where the key main word will be inserted in the object programme. As \texttt{x} is still unknown in this stage of the translation, the address \texttt{d} where contra-identifier
S5aL10: \( d : = g + 2^{26} \times 63; \)
S5aL11: \[
\text{if } e < 0 \text{ or } 2^{24} \leq e \\
\text{then go to S5aL14;}
\]
S5aL12: \( c : = e + 2^{24} \times 63 + 2^{32}; \)
\( d : = d + T; \)
S5aL13: \( S0e(c); \)
S5aL14: \[
\text{if } f \neq 330 \text{ then go to S5aL16;}
\]
S5aL15: \( S0m(d); S0m(e); \)
\( \text{go to S5L9; } \)

**comment**

c will be listed in \( I \) as a preliminary key main word to be used on S5aL20 for storing the internal equivalent of \( c \).

S5aL7;

which is similar to the internal equivalent of a simple variable (cf. table 2);

Then constant \( e \) is either no integer or anyhow unsuitable for being used as a label. Thus no contra-declaration is made.

Otherwise the constant parameter \( e \) will be contra-declared according to \( p = 1 00000000 \) which constant is listed in table 2A together with an explanation:

which is a contra-identifier;

Again (cf. S5aL9) the preliminary key main word points to an address in \( I \).

S5aL9;

Thus contra-identifier \( c \) is listed in \( I \).

On S5aL20 its internal equivalent is stored.

S5aL7, S5aL11, S5aL26;

Then delimiter \( f \) is no comma, thus the actual parameter part is closing in the text.

\( f \) is a comma (cf. table 1B):

Then there is in \( L \):

\[ \text{st}[s0] = \text{by-word } e, \text{ and} \]
\[ \text{st}[s0 + 1] = \text{main word } d \text{ of parameter key, and} \]

the next actual parameter is read.

S5aL14;
Then, in the object programme, \( st[P-1] = 0 \), \( st[P] \) becoming the by-word of the parameter which is the last element of the parameter list.

Transport cycle:

Then parameter is a non-trivial expression and the key is ready (cf. S5aL25);

The key main word is to be stored on address \( g \);
Then \( c \) is either \( = 0 \), or the address where the required contra-declaration is to be listed in I, or the address where the object programme of the string begins;

Then parameter is either a single identifier or a string.
Parameter is a constant;
Then the key of the parameter looks as follows:
\( st[P]= \) constant \( e \),
\( st[P+1] = d = P + 0 \overline{111111t01} \ 0 \ldots 0 \);
which \( = P + 1 + 2^{32} \). If \( c \neq 0 \), then constant \( e \) may occur as a label (cf. table 2A).
S5aL19;

which is the internal equivalent of the contra-identifier stored in I on S5aL13. It refers to the main word of the parameter key which is to be formed or modified, when the translator will have arrived on S3bL23.
S5aL17;
Thus there has been added to the object programme:
\[ \text{st}[P - 2] = \text{by-word } e, \] and
\[ \text{st}[P - 1] = \text{main word } d \] of the parameter key;
Thus next parameter key is extracted from L;

Then the next key is transported.
All keys have been stored in the object programme;
Then \( d \) is the address denoted with \( \text{st}[S - 4] \)
on S5L9;
which is the key address of the function designator
or procedure statement;
S5aL3;

Then mark = i, the parameter having the form \( i[E] \),
in which identifier \( i = \text{st}[S] \).
i is expected to be either an array- or switch identifier.
Parameter is a non-trivial expression differing from \( i[E] \):

\[ \text{cf. table } iE. \] In the object programme of a
 parameter \( i[E] \) which is a subscripted variable or
 switch designator, the last word = 0 instead of return.
S5aL27;

\[ \text{cf. table } iE; \]
\( d, e \) is the key of an actual parameter which is
a non-trivial expression; (cf. table 3);
S5aL24;

Then \( \text{st } [P] \) is either an ar2 instruction or 0
(cf. S6aL61);
S0s;

comment
The mentioned identifier i is contra-declared because the ar2 instruction might, and a zero must be replaced later by a jump instruction. A zero is added to the parameter's object programme;

go to S5aL25
end S5a;
comment

Compound S5b.
Action of delimiter 'leg' (cf. table 1B);
Then \( d = \text{delimiter } st[S - 2] \), the parenthesis
having the value 320;
Otherwise the string beginning in the text is an
actual parameter;
Thus the address \( P \) listed in \( L \) on S5L8 is replaced
by its complement;
Then the string is read. For storing the successive
words, procedure SO is recommended, because it also
pays attention to the position of the list \( L \).
After skipping delimiter 'req', input1 goes to S1L28;
This string is no actual parameter thus perhaps a
piece of code programme to be included in the object
programme?

begin

S5bL1: S0a;

if \( d \neq 320 \) or \( c = 0 \)
then go to S5bL2;
st[S - 3]: = - c;
go to input1;

S5bL2: wrong;

end S5b;
begin

S6L1: \( f := 321; \)

\textbf{if} \( D < 0 \) \textbf{then go to} S6L10;

S6L2: \( s0a; \)

\textbf{if} \( e = 0 \) \textbf{then}

S6L2a: \textbf{wrong;}

S6L3: \( D := D + 2 \times 32; \)

S6L4: \textbf{if} \( D \text{ or } 2 \times 29 \neq 0 \) \textbf{then go to} S6L6;

S6L5: \( e := P; \)

\textbf{go to} S6L7;

S6L6: \( e := q; \)

T1 := c;

---

comment

Compound statement S6.

Pre-action of the opening symbol [.

cf. [ in table 1B;

Then a subscript list is beginning in the text.

- bound pair list;

Then the following assignments are performed:

\( c = \text{value, variable of the 1st declarator} \) of the block was read (cf. S3aL5),

\( d = \text{value 322 of the opening symbol} \begin{array}{l}
\text{begin,} \\
\text{e = address, reserved for the pre-value [a] of the} \\
\text{1st array, called a in the explanations (cf. S4L14} \\
\text{and S4L16);}
\end{array} \\

\text{Then a list of array identifiers has been omitted;}

\text{for D must be negative during the translation of the}

\text{expressions of the bound pairs. Then the tests on}

S6L1 and S4L1 succeed;

Then the arrays are not \textbf{own}.

Arrays are \textbf{own};

which = 1 + address where pre-value [c] of \( c \), the

last array of the list, is to be stored later

(cf. S6aL41);

S6L4;

That address is reserved for the internal variable \( u \).

Thus address \( e + 1 \) is reserved for pre-value [c] of \( c \),

the last array of the list;

for, within the bounds, there only occur non-local

variables.
S6L5;
S6L7: S0c(e);
S6L8: e := e + R + 2^23 - 2 + 2^126 × 126;
      S0e(2^24 × 62);

S6L9: go to S3L3;

comment

In the case of an own declaration, the internal equivalent of the factor identifier 2^{24} × 62 is not e and will be inserted later on S6aL38.
Then, for translating a bound pair list, there is listed in L:

st[S - 5] = address reserved for the pre-value [a] of a = i^{st} array of the list mentioned in the explanations;

st[S - 4] = i + address reserved for the pre-value [c] of c = last array of the list;

st[S - 3] = -1 + number of bound pairs already translated;

st[S - 2] = value 321 of [, 
st[S - 1] = 0.

S6L2;

for a procedure statement with actual parameters can be the 1st statement in the compound tail of a block (cf. S3L2);

Then [ is preceded by an array identifier;
Then there is beginning an expression enclosed within brackets, which is no subscript.
[ is preceded by either a formal parameter or a switch identifier;

cf. ar2 in table 1D.

Then [ is not preceded by a formal parameter. Thus it is preceded by a switch identifier whose declaration may, of course, occur later in the text;

S6L10: S0h;

S6L11: if g ≠ 0 then go to S6L15;
S6L12: if e = 0 then go to S5L4;

S6L13: if e or (2^32 + 2^23) ≠ 0 then
       e := 2^126 × (63 - 125);
S6L14: \( \text{st}[S - 1] := e + 2^{26} \times (126 - 63); \)

goto S6L18;
S6L15: if \( g > 0 \) then
S6L15a: wrong;
S6L16: \( \text{st}[S - 1] := g; \)

S6L17: \( e := U + 2^{31}; \)

S6L18: \( S := S + 2; \)
    if \( S + 1 > T \) then S0g;
    \( \text{st}[S - 1] := e + 2^{26} \times (125 - 63); \)
S0c (- mark);
    \( \text{st}[S - 1] := 0; \)
S6L19: goto S3L5;

Comment
That is either the \( \text{ar1} \) instruction referring to a formal parameter, or a positive and harmless constant;
S6L14;

Then \([\) is preceded by a number;
which is the \( \text{ar1} \) instruction listed in I on S6L8 and referring to the 1st factor of the array declaration;

Then bit \( e_1 = 1 \) as it is in the case that \( e \) is the internal equivalent of a formal parameter.
S6L14;

as happens in procedure S0c;

as is indicated in table 4C;

Then, for translating a subscript list, there is listed in \( L: \)
\( \text{st}[S - 7] = \) an \( \text{ar1} \) instruction or a positive constant,
\( \text{st}[S - 6] = \) the array- or switch identifier or the formal parameter, preceding \([\) in the text,
\( \text{st}[S - 5] = \) an \( \text{ar2} \) instruction of '0',
\( \text{st}[S - 4] = \) minus the value of \text{mark} which was present when \([\) was read,
comment

st[S - 3] = -1 + number of subscripts already translated,

st[S - 2] = value 321 or [,

st[S - 1] = 0

end S6;
begin

comment

Compound statement S6a.

After-actions of the opening symbol [. Delimiter f read last is either a comma or a colon or ]. On S1L1 (or S1L18) the value of st[5 - 3] as left by compound S6, has already been assigned to variable c.

Progressive:

In the text, delimiter f is preceded by a trivial expression p (a constant or identifier). S6L1;

S6L1: d := st[S - 4];
S6L2: if d <= 0 then go to S6aL43;
S6L3: if D or t29 = 0 then go to S6aL27;
S6L4: S0k;

Compare S6L9 with S6L19;
Then p is a subscript.
p is a bound:;
Then declaration is own.
Declaration is not own;
Thus p is translated.
Regressive:
expression p is not trivial and is already translated.
S1L21;
cf. S6aL1;
S6L5: d := st[S - 4];
S6L6: if d <= 0 then go to S6aL53;
S6L7: if D or t29 = 0 then S6aL7a: wrong;
S6L8: if c /= 0 then d := d + 1;

Please only constant bounds in an own array declarations;
Thus, in the case of the 1st bound pair, d is the address of variable u, otherwise d is the address of variable v = [c], as is indicated in the explanations;
S6aL9: if \( f \neq 331 \) then go to S6aL12;

S6aL10: \( S0(d+R+2\times 23+2\times 26\times 117) \);

S6aL11: Saccent := 8191;
go to S1L27;

S6aL12: if \( c = 0 \) then \( q := q - 1 \);

S6aL13: \( S0(d+R+2\times 23+2\times 26\times 74) \);

S6aL14: \( S0(q+R+2\times 23+2\times 26\times 115) \);
S6aL15: if \( c = 0 \) then \( e := 0 \) else \( e := c + 2 \);
S6aL16: \( S0(e) \);

S6aL17: \( q := q - 1 \);

S6aL18: \( st[S-3] := c := c + 1 \);

S6aL19: if \( f = 330 \) then go to S6aL11;

*comment*

Then delimiter \( f \) is no colon.

colon;

cf. store accu in table 1D.
Thus \( u := \text{accu} \) or \( v := \text{accu} \) is translated.
entry, S3bL5, S7aL9, S7aL17, S6aL19;

As long as \( S' \) has the value 8191, the tests on S1L7, S5L4 and S6al4 cannot succeed.
S6aL9; the latter being the same as \( q := d - 1 \).
Then \( q \) is the address to be reserved for the variable \( H_k \), the value of which is the number of array elements;

cf. -- in table 1A.
Thus \( \text{accu} := \text{accu} - u \) or \( \text{accu} := \text{accu} - v \) is translated;

cf. store factor in table 1B;

else \( e := d - q \) may be written instead;
Thus the constant 0 or \( k - 1 + 2 (i < k, \text{cf. explanation}) \) is subsequent to the store factor instruction;

That address is reserved for the variable to be introduced next which is possibly another factor \( h_{i-1} \).
S6aL36, S6aL58;
Thus the number of bound pairs or subscripts is counted;

Then \( f \) is a comma;
if f ≠ 321 then
S6aL19a: wrong;

S6aL20: D := D + 2↑32;
S6aL21: S := S - 2;
S6aL22: if D or 2↑29 = 0 then go to S6aL37;
S6aL23: Soa;

S6aL24: S0(c+R+2↑23+2↑26×114);
S6aL25: S0(d - c);

S6aL26: go to S5aL5;

S6aL27: if g = 0 then go to S6aL7a;

S6aL28: if g + 2↑25 × 127 < 0 then go to S6aL7a;
S6aL29: if f ≠ 331 then go to S6aL32;
S6aL30: if c ≠ 0 then v := e
else u := e;
S6aL31: go to S1L28
S6aL32: if c ≠ 0 then e := e - v
else e := e - u;
S6aL33: if e < 0 then
S6aL33a: wrong;
e := e + 1;

Then a subscript, or a bound of an array declaration, is not followed by the required comma, ], or colon, which compensates for S6L3;

for own;

Then is (cf. S6L9):
c = address reserved for variable [a],
d = address reserved for u;
cf. store pre-value in table 1D;

In the object programme, the store pre-value instruction is followed by the constant which is the difference (address of u minus that of [a]);
The brackets are exhausted.
S6aL3;
Then a variable bound occurs within a declaration of own arrays;
Then the bound has not the integer representation;

S6aL29;
Then lower bound is greater than upper bound;
In the store, the calculated constants must appear in the same order as have the factor variables of the arrays being not own. Storing in L through procedure SOm is a means of reversing the order;

which is the next value of $H_k$;

which is the next value of $u$.

Thus the constants $h_1$, ..., $h_{k-1}$, and $H_k$ are listed in the object programme, $g$ being $H_k$ at the end;

which is the correction announced on S6L8; x

Constant $u$ is stored;

cf. S6aL23.

Addresses $c$ and $d-1$ are reserved for the first and last pre-value constants.

Thus space for own arrays is reserved in object programme;

Then the identifier or constant $p$ to be translated is not the first subscript of the list.

First subscript;
S6aL44: if Saccen + 2 < S then S0d;
    S0k;
    go to S6aL54;
S6aL45: P := P - 1;
S6aL46: if g ≠ 0 then go to S6aL50;
S6aL47: if e < 0 then go to S6aL52;
S6aL48: e := e + 2126 × (72 - 63);
S6aL49: go to S6aL51;
S6aL50: S0(g + 2126 × 72);
S6aL51: S0(e);
    go to S6aL54;
S6aL52: P := P + 1;
    S0p;
S6aL53: if c ≠ 0 then
    S0(2123 × 3 + 2126 × 72);
S6aL54: if f=321 then go to S6aL59;

comment

as happens also on S1L7;
Thus 1st subscript is translated;
S6aL43;
Then st[P] = instruction partres, stored at the
previous action of this compound S6a on S6aL57;
Then g indicates the representation of e,
e being a constant subscript
Then subscript p is a procedure identifier namely
no simple variable and no formal parameter.
Simple variable or formal parameter;
cf. + in table 1A;
x
S6aL46;
S6aL49;
x
S6aL49;
not: to 53.
S6aL47;
Thus instruction partres remains in the object
programme;
Thus a contra-declaration of function name p is made.
The test on S6aL53 succeeds.
S6aL6;
cf. + in table 1A. This instruction
accu := accu + (partial result) corresponds to the x
instruction partres, added to the object programme
during the previous action of this compound S6a on
S6aL57.
S6aL44, S6aL51;
cf. ] in table 1B. Then the last subscript has been
translated.
S6aL55: \( d := st[S - 7]; \)

\[ \text{if } d > 0 \text{ then} \]

S6aL55a: \text{wrong;}

S6aL56: \[ \text{if } d - 2^23 - 2^16 \times 126 < 0 \text{ then } S0(d) \text{ else } c := d - c; \]

S6aL57: \( S0(\text{partres}); \)

\( S0(\text{partres}); \)

\( \text{go to S6aL18; } \)

S6aL58: \( \text{go to S6aL18; } \)

S6aL59: \( \text{mark} := - d; \)

S6aL60: \( \text{Saccent} := S := S - 6; \)

S6aL61: \( S0(st[S + 1]); \)

\text{comment}

\text{Subscript list is not yet exhausted; ;}

\text{That is either an ar1 instruction referring to an}

\text{array identifier or formal parameter, or positive;}

\text{Then 2 or more subscripts are attached to an identifier}

\text{which is no array identifier and no formal parameter;}

\text{Thus the ar1 instruction just stored either refers}

\text{itself to the factor } h_i \text{ (} i < k \text{) required, or it}

\text{refers to the key of a formal parameter which refers}

\text{to the factor } h_{k-1} \text{ of the array represented by the}

\text{parameter, while ar1 is followed, in the object}

\text{programme, by the constant } k - 1 - 1; \)

\text{cf. table 1E}

\text{This instruction partres corresponds to the}

\text{instruction}

\text{accu := accu + (partial result)}

\text{which will be added to the object programme during}

\text{the next action of this compound (cf. S6aL53).}

\text{However, often both instructions are dispensable;}

S6aL54:

\text{Thus the value of mark, listed in L on S6L19, is}

\text{restored;}

\text{Compare this with S5aL5;}

\text{Then } st[P - 1] \text{is the ar2 instruction or 0, mentioned}

\text{on S6L19.}
go to S1L28
end S6e;

comment
st[S] is the identifier to which the subscripts are attached in the text;
begin

S7L1: S0h;
S7L2: st[S - 1] := 0;
S0c(0);
st[S - 1] := -P;
f := 324;
go to S3L4;

end S7;

comment
Compound statement S7.
Pre-action of the opening symbol for
cf. S3L2;
Thus an identifier or constant preceding opening
symbol for is skipped;

Then, for translating a for statement, there is
listed in L:
st[S - 5] = 0,
st[S - 4] = 0 to be replaced by the internal
equivalent of the controlled variable v on
S9L4,
st[S - 3] = minus address P' where the object
programme of the for list element to be read next
is to begin,
st[S - 2] = value 324 of the opening symbol for,
st[S - 1] = 0
begin

S7aL1: S0k;

S7aL2: S := S - 2;
S0a;

if d ≤ 0 then
S7aL2a: wrong;

S7aL3: if c < 0 then go to S7aL18;

S7aL4: S := S + 2;
if f ≠ 324 then go to S7aL10;
S7aL5: if e < 0 then go to S7aL7;

S7aL6: st[e] := a+2×26 ×(123-63);

comment

Compound statement S7a.

After-actions of the opening symbol for.

Delimiter f read last may be a separation symbol
step, until, while, comma, or do, or the closing
symbol of the for statement.

Progressive:
Expression p preceding delimiter f in the text
is trivial (an identifier or constant).
S1L6;
for translation of p.

Regressive:
Expression p is not trivial.
S1L21;

Thus internal equivalent of controlled variable is
assigned to variable d
(cf. st[S - 4] in compound S7);

Then either controlled variable is no simple variable
and no formal parameter,
or even := has been omitted;

Then after-action of for with do has already taken
place so that f is the closing symbol (compare
st[S - 5] in S7aL16 with that in S7aL15 or S7L2);
Thus previous value of S is restored;
Then f is either comma or do (cf. 324 in table 1B);
Then delimiter f is either step or while.

Delimiter f = until;

cf. for1 in table 1D;
S0(for3);
  c := 0;
  go to S7aL8;
S7aL7: c := P;
  S0:(-e);

S0(d + 2*26 * (117 - 63));

S7aL8: st[S - 3] := c;
S7aL9: go to S6aL11;

S7aL10: if e > 0 then go to S7aL12;

S7aL11: if e ≠ 0 then
  begin
  S0(d + 2*26 * (117 - 63));
  b := for2;
  end
else b := d + 2*26 * (74-63);
  S0(b);
  go to S7aL14;

S7aL12: b := st[e];

comment

S7aL5;
Thus address where object programme of while- or step element begins, is retained in object programme itself on address c;
cf. store accu in table 1D.
S7aL6, S7aL15;

S7aL4;
Then value of e has been listed in L during previous action of this compound on S7aL7-8.
It is a while element whose object programme is to be completed;

element is an expression

S7aL14;

Object programme of step- or expression element is ready.
S7aL10;
That is the address c of S7aL7. Element has the form: F while G.
S7aL13;
S7aL13: st[e] := st[e - 1];
   e := e - 1;
   if e ≠ b then go to S7aL13;
   st[b] := for2;

Thus object programme of expression F is shifted over one place;

cf. table 1E.
Thus instruction for2 precedes object programme of F.
S7aL11;

cf. do in table 1B;

S7aL14: if f = 323 then go to S7aL16;
   if f ≠ 330 then
S7aL14a: wrong;
   S0(c);
   c := - P;
   go to S7aL8;

Then for statement contains wrong delimiter;
That is the address where to insert a for instruction
later (cf. S7aL16);
On address st[S - 5] the previous address where to
insert a for instruction, is stored;

Thus c is the complemented address where the object
programme of the next for list element is to begin.
S7aL14;

cf. for in table 1D;
Thus last for instruction is stored;
That is the complemented address where to store a
pass instruction later (cf. S7aL18);
For the time being, 0 is inserted.
S7aL17;
Then all for instructions have been inserted;

S7aL16: e := P + 2 + 2↑23 + 2↑26×124;
   S0(e);
   st[S - 5] := - P;
   S0(0);

S7aL17: if c = 0 then go to S6aL11;
   b := c; c := st[c];
   st[b] := e;
   go to S7aL17;

Thus next for instruction is inserted on address b;
S7aL3;
S7aL18: S0(for0);

Thus object programme of for statement is concluded by adding the word for0;

st[-c] = P + 2↑23 + 2↑26 × 121; Thus a bridging pass instruction (cf. table 1D) is inserted;

go to S1L17;

There variable S is again (cf. S7aL2) decreased with 2, and the delimiter preceding for in L, is going to after-act with delimiter f

end S7a
comment

Compound statement S8.
Pre-action of the opening symbol `go to';
cf. S3L2;
Then $a_t[s - 2]$ is the value 325 of `go to'
(cf. table 1B)
S8L6, S9L3;
cf. table 4C;
as happens also on S6aL11;
begin

S8aL1: if mark = 0 then go to S8aL3;
P := P - 1;

S8aL2: S01;
S8aL3: mark := 0;
go to S1L17;

end S8a;

comment

Compound statement S8a.
After-actions of the opening symbol go to.

Regressive (cf. S8bL1):
S1L21;

Progressive (cf. S8bL2):
S1L6;
S2aL32, S8aL1, S8bL11;

Delimiter st[S - 4] is going to after-act with delimiter f
begin

comment
Compound statement S8b.

After-action of the opening symbol **switch**.

f = delimiter read last is either the separation
symbol comma or the closing symbol semicolon of the
switch declaration.

Regressive:
The designational expression p preceding delimiter
f in the text, is not a single label.

S1L21;

Then p contains more than a switch designator and
has already been translated and contra-declared.
p is a switch designator i[E]. Translation of E is
ready; st[P - 1] = 0 or an ar2 instruction
(cf. S6aL61) which is to be replaced by a jump
instruction later.

Identifier i is available as st[S] and g = 0;

Progressive:
p is a label.

S1L6;

Thus there is made a contra-declaration of the label
or switch identifier, referring to address P - 1
where a jump instruction will be inserted later
(cf. S3bL39).

S3bL1;

Now d is the address where the designational
expression preceding delimiter f in the text, has
it object programme beginning (cf. S8bL6 and S4L39);
which is the pass instruction referring to d;

S8bL1: if mark = 0 then go to S8bL3;

P := P - 1;

S8bL2: S01;

S8bL3: S := S - 2; S0a;

S8bL4: c; = d + 2123 + 2126 × 121;
S8bL5: \textbf{if} f \neq 330 \textbf{then} go to S8bL7; \\
S8bL6: \textbf{S} := S + 2; \\
\textbf{st}[S - 4] := P; \\
\textbf{S0m(c);} \\
\textbf{go to S8L2;}

S8bL7: d := 0;

S8bL8: d := d + 1; \\
\textbf{S0(c); S0 := S0 + 1;} \\
c := \text{st}[S0 - 1]; \\
\textbf{if c \neq 0 then} go to S8bL8;

S8bL9: q := q + 1; \\
S8bL10: \text{st}[e - 1] := P+R+2^{13}\times127; \\
S8bL11: S0(q + R + 2^{13} - 1 + 2^{26}\times112); \text{cf. switch in table 1D;}
\textbf{S0(d);} \\
\textbf{go to S8aL3;}
end S8b;
begin

S8dL0: wrong;

S8dL1: if D > 0 then go to S8dL8;

if f ≠ 332 then
S8dL1a: wrong;
S8dL2: S := S - 2; S0a;

S8dL3: R := R - 2 + 13;
S8dL4: if (st[e - 1] or 2124) = 0 then S0(extract procedure);

S8dL5: S0(Y);

S8dL6: if st[d - 1] < 0 then
    st[d - 2] := X1;
S8dL7: e := e - 2; go to S3bL12;
S8dL8: if f = 332 then go to S8dL9;
    if (D or 2123) = 0 then go to S5L2;
    go to S1L28;

comment

Compound statement S8d.
After-actions of the opening symbol procedure.
Progressive:
S1L6;
Procedure body is a dummy or a single identifier or constant.
Regressive:
S1L21;
Then body must still be read.
Body has been translated:

Then procedure body is not followed by a semi-colon;
Then c = q', d = P' + 2 and e = T' are the values mentioned on S4L31;
for compensating the action of S4L26;
cf. table 1E.
Thus instruction extract procedure only appears in
the object programme of a type procedure;
Thus, in object programme of any procedure,

Thus, in object programme of a procedure having no
parameters, instruction X1 of table 1E is 1st word;
for semi-colon;

for reading the next formal parameter;
for reading parameter to be next specified.
S8dL8;
S8dL9: \( T_1 = T; \ D = 2^{23} \times 543; \)  
\[ \text{go to S6aL11} \]
\[ \text{end S8d}; \]
expressions}

\begin{align*}
\text{begin} \\
S9L1: & \text{if } g \neq 0 \text{ then} \\
S9L1a: & \text{wrong}; \\
S9L2: & \text{S0h}; \\
S9L3: & \text{if } d = 328 \text{ then go to S8L2;} \\
S9L4: & \text{if } d \neq 324 \text{ then go to S9L5;} \\
& \text{st}[S - 4] : = e; \\
& \text{go to S1L28;} \\
S9L5: & \text{if } e < 0 \text{ then go to S9L11;} \\
& \text{if } e \neq 0 \text{ then go to S9L7;} \\
& \text{if } (\text{st}[P - 1] \text{ or } 2 \times 26 \times 127) \neq 2 \times 26 \times 125 \text{ then} \\
S9L5a: & \text{wrong;} \\
S9L6: & e : = - 1; \\
& \text{go to S9L9;} \\
S9L7: & e : = e + 2 \times 26 \times (117 - 63); \\
S9L8: & \text{if } e - 2 \times 23 - 2 \times 26 \times 117 > 0 \text{ then go to S9L10;} \\
\end{align*}

\text{comment}

\begin{align*}
\text{Compound statement S9.} \\
\text{Pre-action of the opening symbol : =;} \\
\text{Then : = is preceded by a constant;} \\
as happens also on S3L2. The procedure performed the following assignments: \\
d : = \text{st}[S - 2], \ e : = \text{st}[S - 1]; \\
\text{Then the delimiter listed last in } L \text{ is } \text{switch} \\
(\text{cf. S4L39}); \\
\text{Otherwise the delimiter listed last in } L \text{ is } \text{for}; \\
as was announced in compound S7; \\
S9L4; \\
\text{Then : = is preceded in the text by the identifier of a type procedure;} \\
\text{Then either a simple variable or a formal parameter is precedent to : =;} \\
\text{A subscripted variable is precedent to : = ;} \\
\text{Then, within an assignment statement, : = is not preceded by a (simple or subscripted) variable or formal parameter or identifier of a type procedure;} \\
\text{S9L5;} \\
\text{cf. store accu in table 1D;} \\
\text{Then a simple variable is precedent to : =} \\
\text{Formal parameter;}
e := e + 2↑26;

S9L9: S0(e);

e := extract address;

S9L10: st[S - 1] := e;

f := 323;
go to S3L4;

S9L11: f := e + 2↑32; S0f;
e := (d or 2↑26-2↑13)+2↑13 + 2↑26×1↑13+3+Q0;
if (d or 2↑24 + 2↑29) = 0
then go to S9L10;
go to S9L5a
end S9;

cf. store address in table 1D.
S9L6;
Then st[P - 1] is either a store address instruction or the constant -1;
cf. table 1E.
S9L8, S9L11;
Thus either a store accu-or store procedure- or extract address instruction is listed in L;
cf. := in table 1B.
S9L5;
Thus procedure identifier is again looked up;
cf. store procedure and Q0 in tables 1D and 1E;
Then identifier signifies a type procedure;
Compound statement S9a.

After-actions of the opening symbol :=
On S1L1 or S1L18, the value of st[S - 3] has been assigned to variable c.

Progressive:
The expression preceding delimiter f in the text is a single identifier or constant p.
S1L6;

Thus p is translated.

Regressive:
p has already been translated.
S1L24;

Thus st[P - 1] is the instruction listed in L on S9L10;
cf. S8aL3;

input:
begin
:
end

input1:
begin
:
end