REPORT 164 MA

PROCESS FOR AN ALGOL TRANSLATOR

PART TWO:

THE INTERPRETER
REPORT 164 MA

PROCESS FOR AN ALGOL TRANSLATOR

PART TWO:

THE INTERPRETER

BY: G. VAN DER MEY

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

JULY 1962
2.0 General comment
2.1 Compound statement "Entry"
2.2 Compound statement S10 ; main . of the interpreter
2.3 Compound statement S11 ; the big transporter
2.4 Compound statement S11a; the restorer
2.5 Compound statement S12 ; the small transporter
begin

The Interpreter
For special ALGOL features confer the heading of the translator.
For object programmes to be interpreted and for their variables and arrays, the space PO ... QO is available. The object programme of the interpreter is supposed to be on address PO - 1 and to be realized according to the translation process, given above;

As this own array of the interpreter is declar first, it indeed occupies the absolute address PO ... QO and is joined, within the store, by own variables, declared next. On address PO the interpreter contains a pass instruction, leading to the object programme of compound statement ENTRY;

own integer array st[PO, QO];

own integer A, B, chain, chain2, e, e2, I, J, mant, exp, N, E, p, p1, p2, P, Q, Q2, s1, y;
procedure wrong;

procedure real(F, G);

procedure integer(F, G);

Confer table 4B;
This procedure, mentioned in the translator heading, reveals the mistakes, listed in table 5A.
S1OL 24, 25, 27, 123;

Machine code, The procedure transfers integer \( F \) to the real representation, assigning the mantissa, and \( 1 + 2 \times \) the exponent, of the result to \( F \) and \( G \)
S1OL 27, 62, 65, 91, 122, 128;

Machine code. \( f \) be the value of the number with the mantissa \( F \) and the exponent \( (G - 1) \div 2 \).
Then \( G := 0 \) and \( F := \text{entier}(f + 0.5) \).
Switch for the non-extractive instructions. In Table 1D shows the reversed order.
S1OL15;
switch sw1 :=
S1OL135, S1OL142, S1OL140, S1OL130, S1OL128, S1OL120, S1OL96, S1OL91, S1OL70, S1OL121, S1OL144, S1OL101, S1OL145, S1OL58, S1OL55, S1OL106, S1OL108, S1OL65, S1OL62, S1OL100;
switch sw2 :=
S1OL88, S1OL87, S1OL81, S1OL144, S1OL126;

As the arithmetic part of the interpreter is not described below, the labels in the next 2 switch lists are imaginary.
The operator $\uparrow$ is mentioned on S1OL22.
Switch for applying the operators (table 1A) to integral values mant and $N$.
The first division operator delivers a real result mant, exp. Otherwise an integral result is assigned to mant, for example:
true = 0 or false = -1.
S1OL23;

switch sw3 :=
multiplication, division, integer division, plus, minus, equal, not $\bot$, greater, greater or equal, and, or, implies, equivalent;

Switch for applying the operators to real val. mant, exp. and $N$. E.
Unless a type mistake occurs (cf. table 5A), a result is obtained, whose mantissa is assigned to mant, while $1 + 2 \times$ the exponent is assigned to exp. The relational operators, combined, replace extraction instruction $\uparrow$ $J$ by an appropriate jump instruction. 
return separated by a jump to the minus section, $\uparrow$ $J$, perform their test and, combined,
S1OL26;
switch sw1 : =

procedure JUMP (F);

MULTIPLICATION, JT 10L147, PLUS, MINUS,
EQUAL, NOT EQUAL, GREATER, GREATER OR EQUAL,
S10L147, S10L147, S10L147, S10L147;
S10L2, S10L3;

Machine code. The procedure only executes the machine
code jump instruction F, which is a positive wr-
in ZEBRA code in contrary to the extraction in-
tion of the interpreter, which is negative;
begin
ENTRY:  
A := read;

B := p := e2 := 0;

Q := Q0;
P := st[P0];
c := P0 + E0;
go to S10L1
end ENTRY;

comment
Compound statement ENTRY;
which = \(2^i - 1\), when exponents are wanted to occupy
i digits (the right-most i digits of a location);
B = st[Q0 + 2] may also be regarded as a permanent
constant 0 of the interpreter which need not be
cleared here;
cf. table 1E;
which is the address P1 of table 4B;
\begin{verbatim}
begin

S1OL0: Q := y := Q - 2;
    if Q < P then wrong

S1OL0a: st[y + 1] := mant;
        st[y + 2] := exp;

S1L1: e := e + 1;

S1L2: if e ≥ 0 then JUMP(e);

    I := st[e - EO];
    N := st[e + 1 - EO];
S1L3: if I ≥ 0 then JUMP (I);

S1L4: y := 8191 or I;
    if y ≠ 0 then go to S1OL9;
S1L5: J := I ÷ 2^26;
S1L6: if (I or 2^24) = 0 then go to
    S1OL8;

comment

Compound statement S10.
S1OL3, the instruction partres:
S1OL 41, 71;
Addresses Q + 1 and Q + 2 are reserved for storing a partial result;
cf. table 5A.
S1OL124;
Thus mantissa and exponent of a partial result stored separately.
S1OL 22, 63, 81, 89, 90, 99, 100, 104, 111, 130, 131,
ENTRY;
Extraction instruction is increased with 1.
S1OL 44, 59, 95, 109; S11aL9;
Then e is a machine code instruction and is normally executed.
Otherwise 2 consecutive words are extracted from the object programme;
For EO cf. table 1E;
+ x
Then instruction I is normally executed. For example, instruction I = partres jumps to S1OL0.
A negative instruction I is interpreted as follows:
S1OL 73, 113, 139;
which is the address part of instruction I;
Otherwise address y is 0;
which is analogous to S1OL14;
Then N is a programme constant to be extracted by
the instruction I preceding it in the object programme.
\end{verbatim}
S10L7: \( Q : = Q + 2; N : = st[Q - 1]; \)
\( E : = st[Q]; \)
\textbf{go to S10L20;} 

S10L8: \( e : = e + 1; \)
\textbf{go to S10L17;} 

S10L9: \( J : = I \text{ or } 2^{\cdot}23 - 2^{\cdot}13; \)
\textbf{if } \( J \neq 0 \) then \textbf{go to S10L10;} 

\( p1 : = 0; \)
\textbf{go to S10L14;} 

S10L10: \( p1 : = p; s1 : = \text{chain}; \)

S10L11: \textbf{if } \( s1 < J + 8192 \) then \textbf{go to S10L12;} 

\( s1 : = st[p1 + Q0 + 1]; \)
\( p1 : = st[p1 + Q0 + 2]; \)
\textbf{go to S10L11;} 

S10L12: \( y : = y + p1; \)

S10L13: \textbf{if } \( I \text{ or } 2^{\cdot}23 = 0 \) then \textbf{go to S10L30;} 

\begin{center}
\textbf{comment}
\end{center}

Instruction I must extract partial result from addresses \( Q + 1 \) and \( Q + 2 \) (cf. S10L0); 
Addresses \( Q - 1 \) and \( Q \) are no longer occupied. 
I can only be a calculative instruction, thus:

S10L6; 
for programme constant \( N \) is not the instruction to be extracted next. I can only be an extractive or calculative instruction, thus:

S10L4; 
which is \( 2^{13} \times \text{rank } r0 \) contained in instruction I; 
Then \( r0 \) must be looked up in the rank chain. 
\( r0 = 0 \) is the last element in the rank chain. 
The pre-value 0 corresponds to it, thus:

S10L9; 
\textbf{if } \( s1 < J + 8192 \) then \textbf{go to S10L12;} 
\( s1 : = st[p1 + Q0 + 1]; \)
\( p1 : = st[p1 + Q0 + 2]; \)
\textbf{go to S10L11;} 

S10L12: \( y : = y + p1; \)

S10L13: \textbf{if } \( I \text{ or } 2^{\cdot}23 = 0 \) then \textbf{go to S10L30;} 

\( s1 \) again having the form \( s1 = s + 2^{13} \times r; \)

S10L11; 
which addition is useful only when \( y \) is 
relative address of a variable thus no absolute address of a word of the object programme; 
Then instruction I refers to a formal parameter.
S10L 9, 35, 80, 77;
S10L14: \( J := I \div 2^{16} \);

S10L15: \begin{align*}
\text{if } J + 21 &> 0 \text{ then go to} \\
&\text{sw1}[J + 21];
\end{align*}

S10L16: \( N := \text{st}[y] \);

S10L17: \begin{align*}
\text{if } (I \text{ or } 2^{12}) &= 0 \\
\text{then } E &= 0 \\
\text{else} \\
\text{begin} \\
y &:= A; \ E := y \text{ or } N; \ N := N-E; \\
E &:= 1 + 2 \times (E -(2 \times E \text{ or } y+1)) \\
\text{end}
\end{align*}

S10L18: \begin{align*}
\text{if } (I \text{ or } 2^{12}) &\neq 0 \text{ then go to} \\
&\text{S10L27;}
\end{align*}

S10L19: \begin{align*}
\text{if } J + 33 &> 0 \text{ then go to} \\
&\text{sw2}[J + 33];
\end{align*}

(comment)

Then \( J = -128 + \text{number q} \), mentioned in table 1A, 1C or 1D for instruction I, the regressive version included. The term -128 is introduced by the multiplication, because I is negative. Then instruction I is non-extractive. In tr the numbers 104 to 107 are not yet used. Otherwise an extraction is performed:

S10L18, S11L43;

As \( N \) and \( \text{st}[y] \) are integral variables of the interpreter, this is an assignment without representation. Of course, \( \text{st}[y] \) may be a variable of the object programme whose instruction is running. Yet the value of \( \text{st}[y] \) is literally copied.

For A cf. ENTRY on page 172;

Bit \( I_7 = 0 \) indicates that \( N \) is of integer type. Then \( E := 0 \). Otherwise \( N := \text{mantissa of extracted value} \), and \( E := 1 + 2 \times \text{exponent with the correct sign digit } E_0; \)

Then the representation of the just extracted value of a formal parameter must be transferred. \( x \)

S10L 28, 51;

Then instruction I is extractive and its number \( q \) in table 1C \( \geq 96; \)
S10L20: \textbf{if} \ (J \textbf{ or } 1) = 0 \textbf{ then go to} \ S10L21;

\begin{align*}
I & = N; \ N = \text{mant}; \ \text{mant} = I; \\
I & = E; \ E = \text{exp}; \ \text{exp} = I;
\end{align*}

S10L21: \ J = (J + 64) / 2;

S10L22: \textbf{if} \ J = 0 \textbf{ then go to} \ S10L20;

S10L23: \textbf{if} \ E \neq 0 \textbf{ then go to} \ S10L25;

\begin{align*}
\textbf{if} \ \text{exp} = 0 \textbf{ then go to} \ \text{sw}3[J];
\end{align*}

S10L24: \text{real} \ (N, \ E);

\textbf{go to} \ S10L26;

S10L25: \textbf{if} \ \text{exp} = 0 \textbf{ then real} \ (\text{mant exp});

S10L26: \textbf{go to} \ \text{sw}4[J];

S10L27: \textbf{if} \ E = 0 \textbf{ then real} \ (N, \ E)
\begin{align*}
\text{else integer} \ (N, \ E);
\end{align*}

S10L28: \textbf{go to} \ S10L29;

S10L29: \ E = st[y]; \ J = st[y + 1];

\begin{center}
\textbf{comment}
\end{center}

S10L7:
Then the operation to be carried out is either commutative, or it must be applied to accu and the extracted value taken in the order as given here. Otherwise it must be applied to the values taken in the opposite order;

S10L20:
Thus the gression bit is shifted off and \ J \ is no longer negative;

\begin{center}
exponentiation is performed and a return to S10L1. Otherwise:
\end{center}

Then \ E = 1 + 2 \times \text{exponent}, and \ N = \text{mantissa of a real value};

Then operation is applied to integers mant and \ N;

As accu is real, integer \ N \ must also be given the real representation;

S10L23:
for the other value has also the real representation.

S10L18, 51;

S10L13:
Thus by-word and main word (table 3) of a formal parameter are extracted.

S11L36;
Thus instruction I adopts the eventual type indication of key main word$J$; which is the operation part of instruction S10L 84, 86; which is the address part of key main word$J$;

Then parameter represents an expression;
Otherwise parameter represents simple variable cf. table 5A and jump in table 1D.
S10L35;
Then parameter represents an array;
Then the parameter represents either a switch, or a label, or a constant actual parameter which constant also occurs in the text as a label;
for a string has no type (cf. table 5A).

Formal parameter$ f $represents a procedure;
Then$ f $is either a designator having no actual parameters, or a left part element of an assignment statement.

$ f(...). $ The prostat instruction (cf. table 1D) I which refers to parameter$ f $ and the key address$ N$ are the first 2 words of the object programme of designator$ f(...). $ The procedure, represented by parameter$ f $, has, of course, formal parameters;

cf. table 5A;
st[Q + 5] = chain;
st[Q + 6] = p;
st[Q + 7] = P;
st[Q + 8] = N;
I = E;
E = st[y];
J = st[y + 1];
go to S11L1;

S10L41: if p1 ≠ 2↑26 × 116 then go to S10L42;

mant = J + 2↑27

S10L42: Q = Q - 6;

if Q < P then wrong;
st[Q + 1] = mant;
st[Q + 2] = exp;
st[Q + 3] = chain;
st[Q + 4] = p;
st[Q + 5] = e;
st[Q + 6] = I;

J is the value of chain, belonging to the action of the block to which the procedure to be called is local. That value must be copied. In S11L4, the key address 0 is introduced in chain, which makes, when returning from the procedure, the test on S11aL4 fail. S10L39;

Then a procedure having no formal parameters is called.

Assignment to function name: (cf. store addressable 1D). A store procedure instruction is here instead of the store accu instruction of S10L 34, 41;

cf. table 5A;

The information, stored on addresses Q + 1 to Q + 6, is used for the return on S10L47;
S1OL43: chain := st[y]; p := st[y + 1]; y is the address Z mentioned for procedures and expressions in table 3;

S1OL44: e := E;
    if e < 0 then go to S1OL2;

Then the parameter represents a (non-expression U. The object programme U' may now interpreted.
When U is a subscripted variable, then, at of U', an ar2 instruction with subsequent proceeds to S1OL74 for the return from U'. U is another arithmetic or logical express instruction return at the end of U' jump S1OL46 for the return from U'.
U is a procedure;

S1OL45: I := 1;
        go to S12L1;

The key address 1 is introduced in variable chain, and U is invoked. Instruction Y at the end of object programme U' jumps to S11aL1. The key address 1 makes S11aL3 proceed to S1OL46 for return from U'. S1OL3, the instruction returns:
S1OL44, S11aL3;

Thus the value, obtained by evaluation of the expression or function, now occupies N and E, as does the value of an extracted variable on S1OL17;

S1OL46: N := mant; E := exp;

Thus the values, retained by S1OL42, are restored;
which is analogous to S1OL44;
Then instruction I is calculative or extractive (cf. tables 1A and 1C);
S10L50: wrong;

S10L51: if (I or 2\#24) = 0 then go to S10L19;
        if (I + 2\#25\times E or 2\#25) = 0 then go to S10L19;
        go to S10L27;
S10L52: if p1 ≠ 2\#26 \times 122 then wrong; cf. table 5A.
        E is the address Z mentioned in table 3 for labels and switches;
which are the values to be restored before the jump; only for joining the course coming from S10L12.
S10L53: s1 : = st[E]; p1 : = st[E+1];
S10L54: y : = y + p1;
S10L55: if e2 ≠ 0 then go to S10L57;
        Otherwise the following data are stored, needed only when the jump instruction leads to a switch instruction (cf. S10L128) and, in addition, the subscript concerned is "out of capa then the jump instruction does not lead to a switch instruction, it leads to a restore instruction (cf. S10L101).
        Then e2 is again cleared.
S10L56: chain2 : = chain;
        e2 : = e; p2 : = y;
S10L57: chain : = s1; p : = p1;
S10L58: e : = y - p1 + E0;

S10L59: Then, in the text, no type is specified for parameter f;
        x
Then the representation of the value obtained is in accordance with the type specified for parameter f. Transfer is required;
        x
S10L60:
S10L59: go to S10L2;
S10L60: if p1 ≠ 2^26 × 126 then go to
S10L64;
S10L61: y := E - N;
    e := e + 1;
S10L62: if exp ≠ 0 then
    exp);
S10L63: mant := mant × st,E;
    go to S10L1;
S10L64: if p1 = 2^26 × 125 then go to
S10L65;
    if e > 0 then go to S11L40;
    wrong;
S10L65: if exp ≠ 0 then integer (mant,
    exp);
S10L66: y := mant + st[y];
S10L67: if N = 0 then go to S10L74;
    if N = -1 then go to S10L69;
S10L68: J := -30;

comment

S10L36;

Then I is no ar1 instruction (cf. table 11
st[E] is the first subscript factor,
st[y] is the required subscript factor;
which is analogous to S10L8;
S10L15, the ar1 instruction;;

which is a product of 2 integers;
S10L60;

for an ar2 instruction;
for transport of a value array;
cf. table 5A.
S10L64, S10L15, the ar2 instruction;;

The required subscripted variable is located on
address y;
Then the ar2 instruction is the last instruction in
the object programme of an actual parameter which is
a subscripted variable. Return from that object
programme is arranged;
Then assignment to a subscripted variable is
prepared.
Otherwise a subscripted variable must be extracted;;
which is 2^{-26} × (operation part of extract normally
in table 10C);
go to S1OL16;
S1OL69: e := c + 1;

S1OL70: mant := y + (I or 2125) + 2123 + 2126×117;

S1OL71: go to S1OL0;

S1OL72: Q := Q + 2;
    I := st[Q - 1];
S1OL73: go to S1OL4;

S1OL74: mant := st[Q + 1];
    exp := st[Q + 2];
    chain := st[Q + 3];
    p := st[Q + 4];
    c := st[Q + 5];
    J := st[Q + 6];
    Q := Q + 6;
S1OL75: if(J or 2124) ≠ 0 then go to S1OL78;

S1OL76: I := J + (I or 2124 × 3);

comment
S1OL67;

as happens also on S1OL8.
S1OL15, the store address instruction;
cf. store accu in table 1D. This store accu
instruction is preliminarily stored as a "partial x
result". Its type indication has been copied from
the ar2- or store address instruction I;
The store accu instruction is extracted later on
S1OL72.
S1OL3, the instruction extract address : ;
The store accu instruction from S1OL70 or the
store procedure instruction from S1OL41 is interpreted.
S1OL67;

The store accu instruction from S1OL70 to be compared with S1OL47;
Then, for the formal parameter which represents
subscripted variable, a type has been specified
indicated by the bit J7 (cf. S1OL31 and S14-)
No type has been specified:
Thus, the type of the subscripted varia-
contained in the ar2 instruction, is con-
formed by the formal parameter;
S10L77: go to S10L14;
S10L78: I := (I or 2↑25)-(J or 2↑25);
S10L79: if I = 0 then I := J - 2↑24
else I := J + I;
S10L80: go to S10L14;

S10L81: mant := N; exp :=
go to S10L1;
S10L82: if 16 × J <
S10L83: J := st[y + 1];
S10L84: if (J or 2↑28) ≠ 0 then go to
S10L33;
S10L85: st[y + 1] := J := if
p1 = 2↑26 × 122
then J + 2↑23 × 38
else y + 2↑23 × 505;

S10L86: go to S10L33;
S10L87: N := -N - 1;
go to S10L81;
S10L88: N := -N;
go to S10L81;

S10L75;
which is the difference of the type bit.
Thus either the type bit or the next bit
instruction is inverted;
S10L87, 88,
S10L19, the extract normally instruction:

S10L37;
Then formal parameter represents a label or a
Parameter represents constant st[y],
which constant occurs also as a label;
which is the key main word of the constant
parameter (cf. table 3);

Then the actual parameter key has already been
adjusted earlier on S10L85;

Thus, depending on I being a jump instruction or
not, the actual parameter must be a label or an
arithmetic constant, and the key is adjusted
accordingly. In the case of a label, E is already
the correct key by-word of the formal parameter;
S10L19, the extract inversion instruction:
which is the inversion of the previous value;
S10L19, the extract complement instruction:

S10L3, the instruction take inversion:
S10L89:  mant := - mant - 1;
         go to S10L1;
S10L90:  mant := - mant;
         go to S10L1;
S10L91:  if exp ≠ 0 then integer
          (mant, exp);
S10L92:  if mant < 0 then wrong;
          mant := mant + 1;
S10L93:  st[y] := mant;
          if N = 0 then go to S10L95;
S10L94:  y := y + N;
          st[y - 2] := \begin{array}{c}
          \[ 2 \] \\
          \end{array}
          \times
          st[y - 1] :=
S10L95:  e := e + 2;
          go to S10L2;
S10L96:  N := N + y;
I := st[N - 1];

comment

S10L3, the instruction take complement ;
S10L15, the store factor instruction (cf. S6aL14 and
the explanation);
mant is the difference of the upper and lower bounds
of the bound pair.
k be the number of the bound pairs;
cf. table 5A;

Then y is the address reserved for \( H_k \) = the number
of the array elements. The variable \( u \) and auxiliary
variable \( v \) are located on addresses \( y + 1 \) and \( y + 2 \).
If \( k > 1 \), the addresses \( y - 1 \) to \( y - k + 1 \) are
reserved for the subscript factors \( H_{k-1} \) to \( H_1 \) to be
formed by \( k \) further store factor instructions
when \( y \) is the address of variable \( v \);
which is the next partial value of \( H_k \);
which is the next partial value of \( u \). \( st[y] \) is
the lower bound, calculated last.
S10L 93, 98;

S10L15, the store pre-value instruction;
(cf. S6aL24);
which is the address reserved for variable
addresses \( N + 1 \) to \( y \) are reserved for
pre-values;
which is the number of the array elements;
\[ J := \text{st}[N]; \]

which is \( u. \)

S10L97:
\[ \text{st}[y] := P - J; \]
\[ P := P + I; \quad y := y - 1; \]
\[ \text{if } y \neq N \text{ then go to S10L97;} \]

S10L98:
\[ \text{if } P < Q \text{ then go wrong;} \]

Thus the pre-values are formed and stored.

All pre-values have been stored;

\[ \text{cf. table 5A.} \]

S10L3, the instruction retain:

S10L107, 110, S11L5;

Thus, the value of \( p \) is stored on address

S10L15, the adjust instruction:

\[ \text{cf. table 5A;} \]

S10L15, the restore instruction:

S10L129;

Confere comment on S10L56.

S10L128;

Thus restore a previous value of \( Q; \)

Restore also value of \( P; \)

S10L3, the instruction for2::

may also be another positive value;

S10L15, the for1 instruction::

which plus instruction (cf. table 1A) contains the

type indication of the for1 instruction.

S10L105;

Thus the addresses \( P - 4 \) to \( P - 1 \) are reserved

for the for statement to be carried out;

\[ \text{cf. table 5A;} \]

S10L101:
\[ \text{e2} := 0; \]

S10L102:
\[ Q := y; \]

S10L103:
\[ P := \text{st}[y + 1]; \]

S10L104:
\[ \text{go to S10L1;} \]

S10L105:
\[ \text{mant} := 0; \]

S10L107:
\[ \text{go to S10L107;} \]

S10L106:
\[ I := y + 2^{23} + 2^{26} \times 72 + (I \text{ or } 2^{23} \times 6); \]

\[ \text{if } Q < P \text{ then wrong;} \]
st[P - 4] := e;

go to S10L99;

S10L108: if st[P - 2] < 0 then mant := - mant;
if mant < 0 then go to S10L110;
S10L109: e := y + EC;
go to S10L2;

S10L110: P := P - 4;
go to S10L99;
S10L111: st[P - 2] := mant;
st[P - 1] := exp; go to S10L1;

S10L112: e := st[P - 4]; I := st[P - 3];
mant := st[P - 2]; exp := st[P];
S10L113: if I < 0 then go to S10L4;
S10L114: mant := - 1;
go to S10L1;

S10L120: N := 0;
S10L121: if (I or 2125) = 0 then go to S10L43;
S10L122: if exp # 0 then integer (mant, exp);
go to S10L124;
S10L123: if exp = 0 then real (mant, exp);
S10L124: if N = 0 then go to S10L0a;

I = instruction for 2 is positive, the plus instruction is negative;
S10L15, the for instruction:

Then cycle is no more executed;
cf. table 1E;
Thus execution of cycle begins.
S10L108;
Compare S10L107;
S10L3, the instruction for 3:

Thus the step is stored.
S10L3, the instruction for 0:

S10L15, the store procedure instruction;
S10L15, the store accu instruction;

Then accu must be, or become, real;
Thus accu is an integer;
S10L121;
S10L122;
for assignment to function name;
S11L38;
S10L19, cf. S11L42;

S10L125: N := mant; E := exp;
S10L126: if E = 0 then go to S10L127;

    I := A;
    if E > 0 end E > I then

S10L126a: wrong;
    E := E - 1;
    if E + I <

S10L126b: wrong;

    N := N + (N or I);
    N := N - (N or I);
    if N = 2^32 then N := N - I - 1;
    N := N + (E+2 or I);

S10L127: st[y] := N;
    go to S10L1;

S10L128: if exp ≠ 0 then integer(mant, exp);
    if mant ≤ 0 or mant > N then
goto S10L129;
    e := e - mant - 1;
goto S10L102;

S10L129: chain := chain 2;
    e := e2; p := p2; y := q2;
goto S10L101;

S10L130: if mant = 0 then goto S10L1;
goto S10L58;

S10L131: mant := st[p + Q0 + 4];

comment
Then integer N must be stored;
N = mantissa, and E = 1 + 2 \times \text{exponent} can be stored;
For A cf. ENTRY on page 172;

Then exponent is too large positive;
which is 2 \times \text{the exponent};

Then exponent is too large negative.
I + I = 0 does not occur;
Thus mantissa is rounded;
Thus digits, reserved for exponent, are clear.
Then there is no rounding;
S10L156;

S10L15, the switch instruction;

Then subscript is "out of capacity";
S10L158;

S10L15, the test instruction;
Thus, go to 1 for true = 0, go to 58 for false = -1.
S10L3, the extract procedure instruction;
exp : = st \{ p + 60 + 5 \};
go to S10L1;

S10L135: \ J : = I - 2^{123} - 2^{26} \times 108; 
y : = 2^{26} \times 125;

S10L136: I : = st[Q + 6] or 2^{26} \times 127;

S10L137: I : = if I = 2^{26} \times 122 then J + I else J + y;

S10L138: st[e - 60] : = I;
go to S10L4;

S10L140: \ J : = I - 2^{123} - 2^{26} \times 110; 
y : = 2^{26} \times 98;
go to S10L136;

This value, assigned last to function name, is assigned to accu.
S10L15, the VERIFY instruction:

S10L15, the VERIFY instruction;

Thus value, assigned last to function name, is assigned to accu.
S10L15, the VERIFY instruction;

S10L15, the VERIFY instruction;

cf. VERIFY in table 1D. Then J contains address- and rank part of VERIFY instruction;
cf. ar2 in table 1D.

S10L141, 143;

which is the operation part of instruction st[Q + 6]. That instruction has been stored earlier on address Q + 6 on S10L42, where it was the instr.
I, and it refers to a formal parameter, which represents the expression whose object programme contains the verification instruction which is going to be interpreted now;
Thus, if st[Q + 6] is a jump instruction, the verification instruction is replaced by the same instruction with the same reference as contained in the verification instruction. Otherwise the verification instruction is replaced by the ar2- or extract normally instruction;

which word is definitive;

Instruction I is interpreted. N has still the value, assigned on S10L3.
S10L15, the verify instruction;

To be compared with S10L135;
cf. extract normally in table 1C;
S10L15, the Verify instruction;

To be compared with S10L135;
\[ y := - J + 2123 + 2126 \times 98; \]

S10L143: go to S10L136;
S10L144: wrong;
S10L145: wrong;
S10L146: wrong;

end S10;

comment

When test on S10L137 fails, I beco.
addressless extract normally instruct.

For dummy elements in switch lists.
A prostat instruction did not occur in con.
with a formal parameter, cf table 1D;
Integer operations may not be used with rea.
numbers;
Compound statement S11.
The big transporter.
The object programme of any procedure having formal parameters contains, as its first word, machine code instruction X which jumps to here. There is presented:
I = the machine code instruction which just jumped to X and which is the first word in the object programme of a designator having actual parameters, and key address \( N \) = the second word in that object programme.
The following transport is performed:

res lent value of
\[
\begin{align*}
\gamma & \rightarrow Q - 2 \\
p & \rightarrow Q - 3 \\
\text{chain} & \rightarrow Q - 4
\end{align*}
\]

key main- and by-word of:
\[
\begin{align*}
\text{first param.} & \rightarrow Q - 5, Q - 6 \\
\text{second param.} & \rightarrow Q - 7, Q - 8 \\
\vdots & \\
\text{last param.} & \rightarrow Q' + 3, Q' + 2.
\end{align*}
\]

These keys, derived from the offered actual parameter keys, contain only absolute addr (cf. table 3).
When a value param. does not represent its value becomes the by-word of
When present, value arrays are stored in the lower part of the working space \( \ldots Q \) of the
procedure to be invoked, and the value
P is increased accordingly.
There happens also:
p := Q - 5 - Q0,
chain := N + 2^{13} x rank of procedure to be in.
Q := Q + 1,
st[Q + 1] := new value of P, as happens also if end of the dynamic introduction of a block.
The previous value of P is still available on address p + Q0 + 3.
Assignment to name of type procedure means:
uu on addresses p + Q0 + 4 and p + Q0 + 5

cf. table 5A;
In S11L2, the value of chain will be stored on address Q, as is already mentioned above;
cf. table 1B
S10L40;
Thus, when coming from S11L0, st[Q + 2] is positive = P.
When coming from S10L40, it is negative;

Thus:
st[ st[P] + 4 ] = address I where the procedure's object programme contains the specification pattern of its first formal parameter, being the second word,
\textbf{comment}
\[
\text{st[ st[P] + 3 ]} = N - Q,
\]
which is the difference of the addresses N-1 and Q-1, occupied by the key main words of the first actual respectively first formal parameter, 
\[
\text{st[ st[P] + 2 ]} \text{ is either } = + P \text{ (when coming from S11L0), or negative (when coming from S10L40).}
\]
S11L 35, 38, 39, 41, 45;

\begin{verbatim}
S11L2: Q := Q - 2;
    if Q < P then wrong;
    st[Q + 2] := E;
    st[Q + 3] := J;

S11L3: mant := N := st[P];
    y := st[N + 2];
    J := st[N + 3];
    exp := Q + J;
    E := st[exp];
    I := st[ ];

S11L4: if I \neq 0

if y < 0 then I := N + J;

E := st[N + 4];
\end{verbatim}

\begin{verbatim}
S11L5: p := N - Q0 - 1;
\end{verbatim}

\text{cf. table 5A;}

Thus, first time, chain and p are stored. Next times, the parameter keys are stored.

Cycle for forming and storing parameter keys:

Bein' the retained value of Q;

which is the mentioned difference of address actual parameter key is extracted:

The latter being the main word;

Then actual parameter list is not yet exhausted:

Actual parameter list exhausted:

Q has already the mentioned value Q';

Otherwise I remains = 0 because the programme come from S10L40 instead of S11L0.

I is the next key address;

On address E the procedure's object prog contains \(-2^{13}\) x (rank of procedure)

\text{S12L2; +x}
chain := I - \text{E}^+[\text{E}] \\
\text{if} \ chain < 0 \ \text{then wrong;} \\
e := E + E0; \\
go \text{to S1IL0.}

\text{S1IL6: if } y < 0 \text{ then }

\text{S1IL7: if } I < 0 \text{ then go to S1IL9;}

\text{S1IL8: } y := 8191 \text{ or } I; \\
\quad J := I \text{ or } 2^{13} - 2^{13}; \\
\quad I := I - J - y; \\
\text{S1IL9: if } I \neq 0 \text{ then go to S1IL10;}
\quad J := y; \\
go \text{ to S1IL24;}
\text{S1IL10: if } J \neq 0 \text{ then go to S1IL12;}

comment

being key address I + 2^{13} \times \text{rank of } \text{invoked;}
Then the formal parameter list contains more
than does the actual parameter list (compare S1
cf. table 1E; 
thus object programme of procedure body is execy

\text{S1IL4: thus, when coming from S1IL40; } N \text{ is } \text{mant } + 5'
In both cases, \text{st}[N] \text{ is the value of chain which will}
reside in the which must also be restored after
return from the procedure.
When coming from S1IL40, \text{st}[N - 5] \text{ is the value of }
chain belonging to the action of the block, to
which the procedure to be invoked is local;
Then the actual parameter is an expression.
The key main word \text{J}; \text{ a formal parameter which}
represents an expression, must refer to the address
of the location which contains the value of chain
belonging to the action of the block or procedure
body, in which the expression occurs;
which is the address, contained in the main word
of the actual parameter key;
which is the rank part;
which is characteristic for the kind of the key;
The key is simply copied;
Actual parameter is a string;
Then address N is not necessarily changed;
N := Q0 + 1;
p1 := 0;

go to S11L14;

p1 := p; s1 := chain;

if s1 < J + 8192 then go to S11L14;
N := p1 + Q0 + 1;
s1 := st[N]; p1 := st[N + 1];
go to S11L13;

J := p1 + y;

if 2 x I < 0 + go to 5
if 4 x
if 8 x

if 16 x I < 0 then go to S11L19;
E := st[J]; J := st[J + 1];
go to S11L24;

These values belong to the rank 0. On address Q0 + 2, the interpreter contains the constant 0, being the pre-value corresponding to the rank 0;
S11L10;
S11L13;
Compare S10L11;

Thus the value of chain has been extracted from address N.
S11L11, 13;
Thus, to a relative address y contained in an actual parameter, key, corresponds an absolute address J to be included in the formal param key;
Then actual parameter is a simple variable.
Then actual parameter is an array;
Then actual parameter is either a switch or label, or it is a constant which also occurs the text as a label;
Then actual parameter is a procedure.
Actual parameter refers to a formal parameter any procedure;
Thus the key of that formal parameter copied;
S11L7, 18;
Thus, when a formal parameter represents
a label or a constant which also occurs
in a label, its key by-word E is the address of the
location which contains the value of chain
belonging to the action of the block to which
the label or switch is local. The value of exp.
is the address where the actual programme contains
the key by-word of the actual parameter.
Thus the key main word J of a formal parameter
which represents a constant, refers to that constant, which property is used on S10L82.

Thus, when a formal parameter represents an
array, its key by-word E is an address. On
addresses \( E + 2 \), \( E + 1 \) and \( E \) are available: the
difference (address of first array element minus
pre-value), the number of array elements, and,
if the array has more than 1 dimension, the first
subscript factor.
S11L23: \[ J := I + J; \]

S11L25: \[ (I \text{ or } 2^{24}) \neq 0 \text{ then go to S11L35;} \]

S11L26: \[ \text{if } J \]

S11L27: \[ \text{if } 4 \times x \]

S11L28: \[ 8 \times J < 0 \text{ then wrong;} \]

S11L29: \[ 16 \times J < 0 \text{ then go to S11L32;} \]

S11L30: \[ \text{wrong;} \]

S11L31: \[ 16 \times J < 0 \text{ then wrong;} \]

S11L15:

**In the case of an array**, the key main word \( J \) refers to the pre-value = the absolute address of the array element with all subscripts 0 (that element need not be contained in the space reserved for the array).

S11L9, 18, 19, 21;

That is the specification pattern for the formal parameter, having

\[ 0 \text{...} 0 \times 0 \text{...} 0; \]

Then the formal parameter list contains less elements than does the actual parameter list (r. test in S11L15);

Then no type has been specified for the formal parameter being considered.

A type has been specified:

Then parameter represents an expression:

Then parameter represents a constant:

Then parameter represents a procedure:

A type has been specified for a formal parameter representing a string.

S11L28:

Then a type has been specified for parameter representing a
When a type has been specified for a variable, the type corresponding to the code instruction for jumping to the +x point is stored in the code instruction (except for the +x point, which is stored in the instruction itself). The type is then used to determine the correct type of the variable when the variable is used in an expression. The type of the variable is determined by the type contained in the main word, which is set to 0.

S11L36: \[ i = \text{ANN} + 1 \]

S11L37: \[ \text{if (J - I or 2125) } \neq 0 \text{ then go to } S11L35, \]

S11L38: \[ J = J + 2124; \]

S11L39: \[ \text{if (J - I or 2125) } \neq 0 \text{ then } \]

S11L40: \[ J \text{ copied. The main word is consulted for the value of the variable just extracted.} \]

S11L41: \[ \text{Thus, the bit J of the main word is set to 0.} \]

S11L42: \[ \text{The bit J of the specification bit currently being accessed is then the bit J of the specification bit.} \]

S11L43: \[ \text{Thus the parameter represents now a simple value, not a label, though a type may still be equal to it.} \]

S11L44: \[ \text{The parameter represents now a simple value, not a label, though a type may still be equal to it.} \]
P := P + 1;

go to S10L31;

comment
Then the space P...Q is free,
st[P-1] being the retained value of Q;
Then the extract normally instruction is regarded as to refer to the formal parameter with the key J, E. When the parameter represents a constant, expression, procedure, or variable, accu is given the value obtained by either extraction or evaluation and the interpreter returns through

S10L2) to S11L37 below.

Parameter rep[ ]

of course without extracting the value of the array.
S10L37 (cf. S11L36);
The st[P] is the retained value of Q;

J := if exp = 0
then Q + a
2^23 x 5

if exp = 0 then go to S11L39;

S11L38: y := a
:= i - 1;
go to S10L125;

E := mant;
S11L2;

```
y := P;
P := y

if Q < P then write

st[P] := st[y - 1];

st[y - 1] := I := y - st[E + 2]; i.e. lowest address minus difference address minus pre-value = pre-value;

s1 := st[N] - I;
I := J - N + y - 1;

if (I or 2^24) ≠ 0 then go to S11L42;

S11L41: st[y] := st[s1 + y]; y := y + 1;
if y ≠ P then go to S11L41;
J := I;
go to S11L2;
```
S11L42:  mant := E;
J' := -28;

S11L43:  e := j44 - 1;
N := st[e1 + y];
go to S10L17;

S11L44:  y := y + 1;
if y ≠ P then go to S11L45;

S11L45:  E := mant;
J := if I + 2^23 × 260 < 0
then I + 2^23 × 1018 else
I + 2^24;
go to S11L2
end S11;

---

S10L40;
Thus by-word is put to safety;
Then switch on S10L19 goes to S10L126
(cf. table 1C).
S11L44;
Code instruction j44 jumps to S11L44 below;

---

The bit J_8 of the new main word is 0, while
J_7 differs from I_7;
begin a statement S11a.
The last word in the object program procedure is the be instruction Y, which was resident before.

Then the procedure has been called through a formal parameter and has no formal parameters.

Then the procedure has been called directly (without use of a formal parameter).

Then the procedure has been called through a formal parameter and has formal parameters:

S11aL8: if I = 0 then go to S11aL6;

S11aL9: go to S10L2

end S11a;
BEGIN

E := 1 := I - \pi_0 + 1

A:
N := Q - 4;
Q := N - 2;
if Q < P then wrong;
st[N] := chain;
st[N + 1] := p;
st[N + 2] := e + 1;

GO TO 1
END S12
END

comment
Compound statement S12.
The small transporter.
The object programme of a procedure which has no formal parameters, contains, as its first word, code instruction X1 which jumps to here. For explanation cf. S11;
In object programme of the procedure to be executed rank is the second +x
13 x rank is the second > 1.
cf. Table 5A;

When coming from S12LO, e + 1 is the extra instruction or return.
When coming from S10L45, e + 1 need not be