Technical Report 70-118
NGR 21-002-206

June 1970

DESIGN AND SIMULATION OF AN ALGOL COMPUTER

by

Howard M. Bloom

UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER
COLLEGE PARK, MARYLAND

Technical Report 70-118 June 1970
NGR 21-002-206

DESIGN AND SIMULATION OF AN ALGOL COMPUTER

by

Howard M. Bloom

This research was supported in part by Grant NGR 21-002-206 from the National Aeronautics and Space Administration to the Computer Science Center of the University of Maryland.

ABSTRACT

This report describes the functional design of a stored-program computer which can directly execute a program written in a subset of the programming language, ALGOL 60. For simplicity, the chosen subset does not contain arrays, procedures, switches, or FOR statements. The design is described by the Computer Design Language (CDL).

The report consists of four parts: (a) description of the subset of the ALGOL language using Backus Normal Form, (b) description of the basic elements needed to check the syntax of the program, (c) description of the design of the Algol computer, and (d) simulation of the design. These descriptions include a complete set of sequence charts and a complete description of the design in the Computer Design Language. A sample of output from a simulation run is also shown.

Table of Contents

Design and Simulation of an Algol Computer

Howard M. Bloom

1. Description of the Simplified ALGOL Language

The syntax of the subset of the ALGOL language is taken from that in the revised ALGOL report (3). However, certain definitions are altered so that the language can be constructed from an operator precedence grammar (2). Abbreviations shown in Table 1 are used throughout the report.

1.1 Identifiers and Numbers

<I> ::= A |...| Z | <I> {A|...|Z} | <I> {0|...|9}

Examples:

A ALPHA D10 IDENTIFIER

As defined, the identifier can consist of any sequence of alphabetic or numeric characters beginning with an alphabetic character. However, the proposed ALGOL computer recognizes only the last three characters of each identifier.

<UN> ::= 0|...|9| <UN> {0|...|9}

Examples:

10 654

The only numbers accepted by the computer are integers, as inclusion of the floating point numbers adds complexity without illustrating additional points. The integer of the ALGOL computer is limited to a value less than 2^{18}. For a larger number, only the low-order 18 bits are used.

1.2 Variables and Labels

Variables and labels are the only types of identifiers permissable. They are indistinguishable in the program. All variables must be declared at the beginning of each block. Since the computer only accepts integer numbers, all variables are considered integer variables.

Table 1, Symbols for the Meta-language

Name	Symbol
arithmetic expression	AE
assignment statement	AS
basic statement	BS
block	B
boolean expression	BE
compound statement	CPS
compound tail	CT
conditional statement	CS
declaration	D
factor	F
goto statement	GTS
identifier	I
label	L
primary	P
program	PR
read statement	RS
statement	S
term	T
type list	TL
unconditional statement	UC
unsigned number	UN
variable	V
write statement	WS

Labels are not formally declared but are recognized if they either preceed a colon or follow terminal goto.

<V> ::= <I>

<L> ::= <I>

Examples:

VAR L1

1.3 Arithmetic Expressions

<P> ::= <UN>|<V>|(<AE>)

<F> ::= <P>|<F>↑<P>

<T> ::= <F>|<T>{X|/}<F>

<AE> ::= <T>|{+|-}<T>|<AE>{+|-}<T>

Examples:

Primaries: 43 CAR (A+2/C)

Factors: PR PR↑2 (A+B)↑C

Terms: T C↑A C/D (A+B)/C↑D

Arithmetic Expressions: A A↑B A/B A/B+E

1.4 Boolean Expressions

<BE> ::= <AE>{=|≠}<AE>

The boolean expressions takes the value true if the expression is true; otherwise, it takes the value false.

Example:

A+B=C↑D E≠F

1.5 Assignment Statement

The value of the expression to the right of the assignment symbol is stored as the value of the variable to the left of the symbol.

Example:

A:=B+C

1.6 Goto Statement

<GS> ::= goto <L>

After the statement is recognized, program control is transferred to the statement in the program that has the indicated label.

Example:

goto L1

1.7 Input-Output Statements

<RS> ::= read(<V>)

<WS> ::= write(<V>)

When the read statement is recognized, an integer value is taken from the input channel and stored as the value of the variable. When the write statement is recognized, the value of the variable is placed on the output channel.

Example:

read(A) write(B)

1.8 Declarations

<TL> ::=<V>|<V>,<TL>

<D> ::= integer<TL>

At the beginning of each block, there must be a declaration statement listing all the variables local to that block. All the variables are taken from this list and placed in a table to await assignment of values as determined by the program.

Example:

integer A,B,I

1.9 Conditional Statement

<CS> ::= if<BE>then <US>{Λ|else <US>}| <L> : <CS>

The boolean expression is evaluated and given the value true or false. If it is true, then the unconditional statement following then is

executed and the unconditional statement following else, if it exists, is ignored. If it is false, the unconditional statement following then is not executed and the unconditional statement following else, if it exists, is executed.

Example:

if A=B then C:=D+E else F:=G+H

if Q≠R then L:=M+N

In the first example if A equals B, then C is evaluated but not F. If A does not equal B, then F is evaluated but not C. In the second example, if Q does not equal R then L is evaluated, otherwise nothing is done.

1.10 Program, Block and Statements

<BS> ::= <AS>|<GTS>|<RS>|<WS>|<L>:<BS>

<US> ::= <BS>|<CPS>|<B>

<S> ::= <US>|<CS>

<CT> ::= <S>|<CT>;<S>

<CPS> ::= begin<CT>end| <L>:<CPS>

<B> ::= begin<D>; <CT> end|<L>:<B>

<PR> ::= <B>|<CPS>

Examples:

Basic statement: A:=B+C goto L L1:read(E)

Compound statement: begin <S> ; <S> ;... end

Block: begin integer A,B;<S> ; <S> ; <S>... end

Blocks are distinguishable from compound statements by the fact that blocks have declarations. A block is structured around the begin and end symbols and contains statements enclosed by these two symbols. Any variable declared in a block is local to that block and any blocks may be nested within the given block. It is possible to declare a global variable and then declare the variable within a nested block. The variable can take on new values within the nexted block, but once that block is left, the value of the variable returns to the

global value. Labels are also only defined in the local sense. One cannot transfer from outside a block to some statement within the block.

1.11 Representation of Characters

Because of hardware limitation, only a subset of the ALGOL characters can be used on the computer. The character codes for representing the ALGOL characters are shown in Table 2. In the case of the multicharacter terminals, such as goto represented by 'GOTO', the input is the BCD code for the apostrophe followed by the codes for G, O, T, and O followed by the code for the apostrophe. Since the computer only recognizes a maximum of three characters for a terminal, the input need only have the O, T, and O in addition to the apostrophes. In table 2(b) for the special characters, the BCD codes for the apostrophies are omitted; only the codes for the last three characters are shown.

Table 2(a) Character Codes (simple characters)

(a) Simple characters

Algol Char.	Computer Rep.	Code
0	0	0
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	10
9	9	11
:=	=	13
+	+	20
A	A	21
B	B	22
C	C	23
D	D	24
E	E	25
F	F	26
G	G	27
H	H	30
I	I	31
)	)	34
-	-	40
J	J	41
K	K	42
L	L	43
M	M	44
N	N	45
O	O	46
P	P	47
Q	Q	50
R	R	51
$	$	53
X	*	54
∕	∕	60
/	/	61
S	S	62
T	T	63
U	U	64
V	V	75
W	W	66
X	X	67
Y	Y	70
Z	Z	71
,	,	73
(	(	74

Table 2(b) Character Codes (Special characters)

Algol Char.	Computer	
	Rep.	Code
:	..	3333
;	.,	3373
goto	'GOTO'	466346
if	'IF'	3126
then	'THEN'	302545
else	'ELSE'	436225
	**	5454
=	'EQ'	2550
≠	'NE'	4525
begin	'BEGIN'	273145
end	'END'	254524
integer	'INTEGER'	272551
read	'READ'	252124
write	'WRITE'	316325

2. Syntax Checking and Program Execution

The concepts described in this section are essentially the features of an interpreter program. These features, normally used as a software implementation in a computer system, will be developed and then translated into hardware elements for the ALGOL computer. The reader should be familiar with the concepts of pushdown stacks, dynamic storage tables, languages, grammars and operator precedence as discussed in (2). The concept of syntax-directed interpretering or compiling is described in detail in (4).

2.1 Elements of the Interpreter

(a) Syntax-Checker which checks to see if the input string is correct in syntax, and determines the flow of execution.

(b) Operator Stack which stores the operators or terminals as they are scanned by the syntax-checker.

(c) Operand Stack which stores the operands (or nonterminals) as they are recognized by the syntax-checker.

(d) Dynamic Storage Allocation Table (DSAT) which stores the values of the identifiers as they are being computed or set.

2.2 Interpreter Mechanism

Assume that the program exists as an input string to the interpreter (or computer). It will be of the form:

$$\$S_1S_2\ldots S_n\$$$

where the S_j are characters in the ALGOL language, and the \$ are the markers noting the beginning and end of the string. These characters called terminals are the ones listed in Table 2. The alphabetic and numeric characters when concatinated to form identifiers and integers are called nonterminals or operands. The other terminals are called operators. Eventually, the inter-

preter scans the input string putting operators on the operator stack, and taking the other terminals to form nonterminals and placing them on the operand stack. Using the concept of operator precedence, the operators on the operator stack, or possible future operators in the input string, operate on the operands to form new operands according to the definitions of the language. As these reductions take place, execution of this operand-operator phrase is performed based on its semantics.

As the ALGOL language was defined, it contains an operator precedent grammar. A grammar of this type has the property that no two nonterminals can appear adjacent to one another. An examination of the productions (or definitions) given in Section 1 will show that this is true for this language.

From the set of productions, the precedence between all the operators can be determined in relationship to any given nonterminal. Assume that the input string consists of operators and the other terminals have already been reduced to identifiers or numbers. It can be shown that reduction to identifiers or numbers always take precedent over any other reduction. Define these basic operands as <NT>. Then since the grammar has operator precedence, the operand must be surrounded by operators, e.g. N_i<NT>S_j, where N_i is the operator preceding the nonterminal (it has already been pushed down on the operator stack) and S_j is the present character (i.e. operator) being scanned in the input string. Then one of the following three relationships must hold:

$$N_i \lessdot S_j, \quad N_i \doteq S_j, \quad \text{or} \quad N_i \gtrdot S_j.$$

The first relationship means that S_j has higher precedence than N_i, that is, a reduction involving <NT> and S_j must take place before a reduction

before a reduction involving N_i and <NT>. The second means that N_i and S_j have equal precedence, and the third means that N_i has greater precedence than S_j.

For each nonterminal listed on the left hand side of the productions, it is possible to construct the precedence between the possible operators that can occur adjacent to it in a production. Without describing how precedence is obtained, we will simply give the relations in Table 3. For those cases where a nonterminal can have only one type of operator adjacent to it, that reduction is performed whenever possible. There are many cases of reductions by default when no operators can be combined with a nonterminal.

For example if the nonterminal is <P>, and if N_i is not ↑, then the production <F> ::= <P> is invoked to reduce a primary to a factor.

2.2.1 Operation of the Operator Stack

When the character S_j is scanned, if a reduction cannot take place then the following operation is done.

i := 1 + i

N[i] := S[j]

When a reduction involving the top operator in the stack takes place, the stack operation is the following:

Some execution is done on the string <NT>N_i<NT> or N_i<NT> and then i:= i-1.

2.2.2 Operation of the Operand Stack

When the character S_j is scanned, if S_j is actually in itself a concatenation of alphabetic characters or integers then the following is done:

k:=k+1

O[k]:=S[j]

When a reduction takes place involving the top operator on the stack one of the two operations will occur:

(a) N_i- binary operator

O[k-1]:=O[k-1]N[i]O[k]

Table 3, Precedence among Operators

Nonterminal	Precedence
<F>	$\uparrow \gg X \doteq /$
<T>	$+ \doteq - \lessdot X \doteq /$
<AE>	$(\lessdot + \doteq -$
	$= \doteq \neq \lessdot + \doteq -$
	$+ \doteq - \ll = \doteq \neq$
	$: = \lessdot + \doteq -$
<TL>	integer $\lessdot$,
<CT>	begin $\lessdot$;
	begin $\doteq$ end

(b) Identifier Type

The type is either INTEGER or LABEL

(c) Value

The value will be the numerical value in the case of a variable or the position in the input string S_j where the first character of the statement referenced by the label is stored, in the case of labels.

The pointer T will always be used to have the value of the present top row in the table. Thus T points to the last variable declared in the present block being scanned.

The DSAT has an auxilary stack STORAJ whose index BLKNUM is the degree of block nesting. The value of the top row in STORAJ is the location (or row) in the DSAT of the first identifier declared in the present block being scanned. There are essentially five operations performed with the DSAT:

(a) Entering a block (Recognition of begin)

```
BLKNUM:=BLKNUM+1
STORAJ(BLKNUM):=T+1
```

This operation increases the block nesting index by one, and stores the first possible row of the DSAT (since T is the top row of the previous block) in STORAJ.

(b) Declaring variables

After the declaration has been scanned, all the declared identifiers have been stored in the operand stack. At this point for K identifiers on the stack, do the following:

```
T:=T+1
DSAT(T,1):=0(k)
DSAT(T,2):=INTEGER
DSAT(T,3):=0
```

k:=k-1

(c) Declaring labels

After the block is entered, it is scanned for labels. This process simply looks for colons and stores the preceding identifier (stored as 0(k)) in the table. Any labels declared within a block nested within the given block are not stored. Hence it is necessary to keep a count of the degree of nesting. These labels must not be stored because the language semantics do not permit a transfer into the middle of a block from outside the block. The storage operation is the following. Assume the colon occurs at location j of the input string:

T:=T+1

DSAT(T,1):=0(k)

DSAT(T,2):=LABEL

DSAT(T,3):= j+1

(d) Assignment of values to identifiers

In executing the statements of the program, the value of the identifiers are not stored in the operand stack but instead the row of the DSAT (where the identifier is located) is stored. Thus the operand stack must consist of two columns. The first column contains the information "link" or "value". The information is "link" if the operand is an identifier. For any other nonterminal including unsigned number, the contents will be "value". The second column contains a row number of the DSAT where the identifier is stored if column one is "link", or it contains a numerical value if column one is "value".

(e) Leaving a block (recognition of end)

T:= STORAJ(BLKNUM)-1

BLKNUM:=BLKNUM-1

When the block is left, T is made to point to the top of the previous block and the degree of nesting is reduced by one.

2.3 Syntax Checker

The syntax checker controls the flow of execution of the program while checking to see if the input string is an acceptable program. The checker can be described through the use of a flow chart. Without going into much detail, a few simple concepts will be explained.

The flowchart consists mainly of "nonterminal boxes", i.e. points in the program when it is assumed a given nonterminal has been recognized. Questions asked at these boxes are: What is S_j? or What is N_i? These questions determine the order of reduction based on the precedence relations. There is a unique box called the initial point that begins scanning the first character of any right side of a production. The flow of the interpreter centers around this box. Other boxes are involved in the execution using the operator and operand stacks. At many points in the flowchart, it is possible to recognize illegal strings. At that time, the variable ERROR is set to a given value and the interpetor stops.

The algorithms for the execution phase of the program will be given at the time when the actual computer implementation is described. The syntax checker flowcharts on the following pages will serve as a rough guide as to what is happening in the computer.

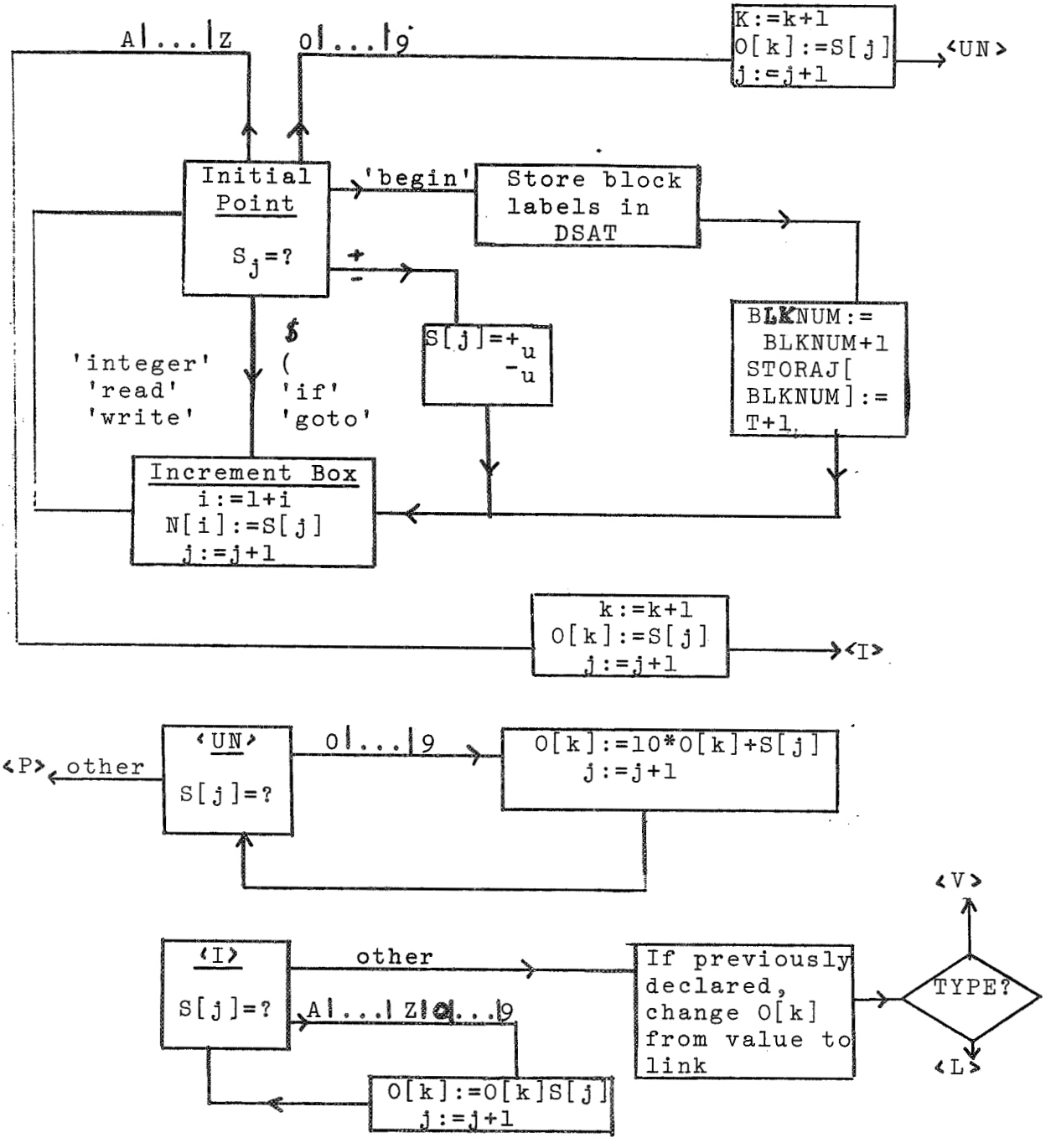

Fig. 1 Syntax Checker and Execution for Initial Point, Increment Box, Identifier, and Unsigned Number

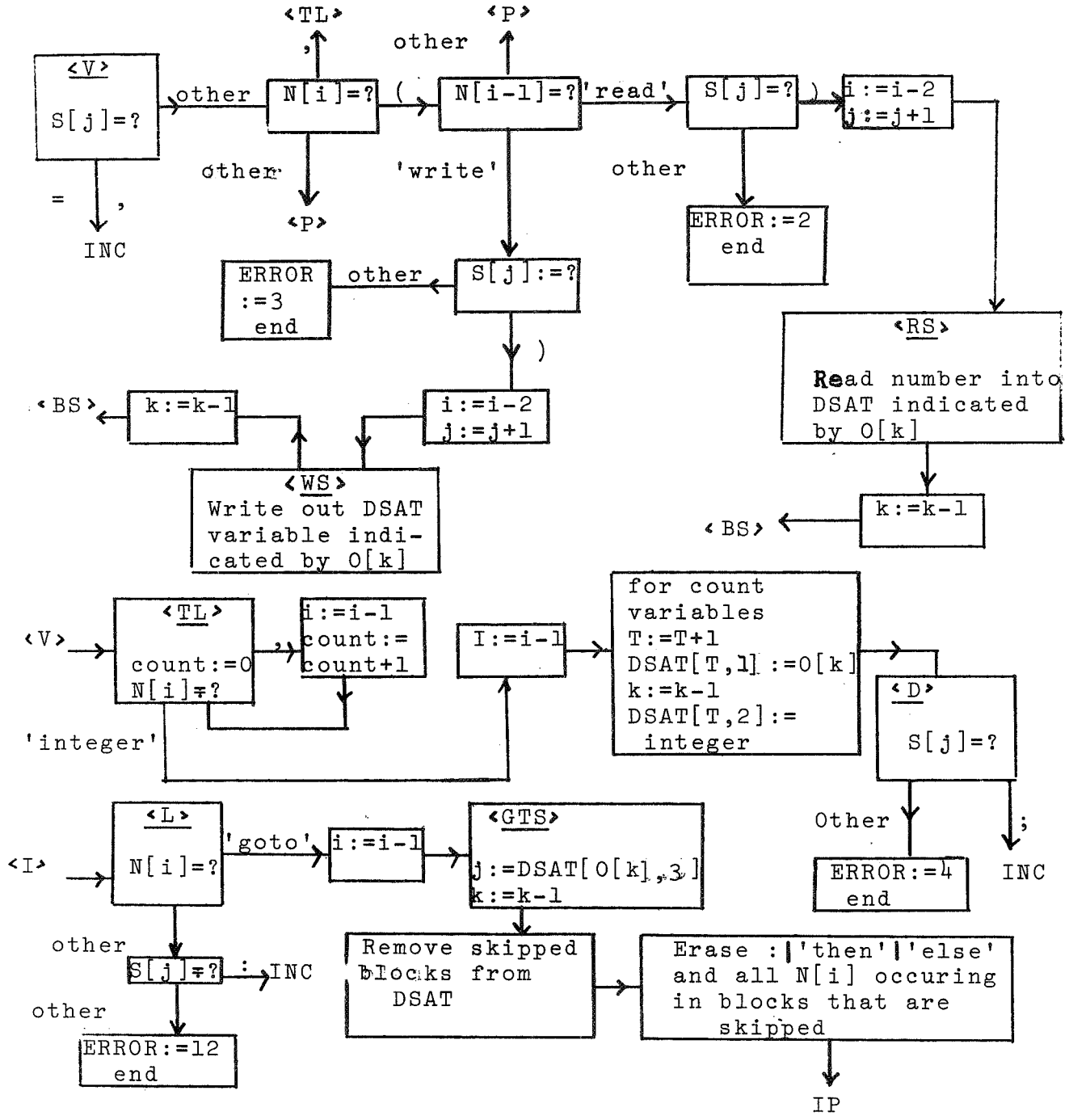

Fig. 2 Syntax Checker and Execution for Variable, Read Statement, Write Statement, Type List, Declaration, Label, and Goto Statement

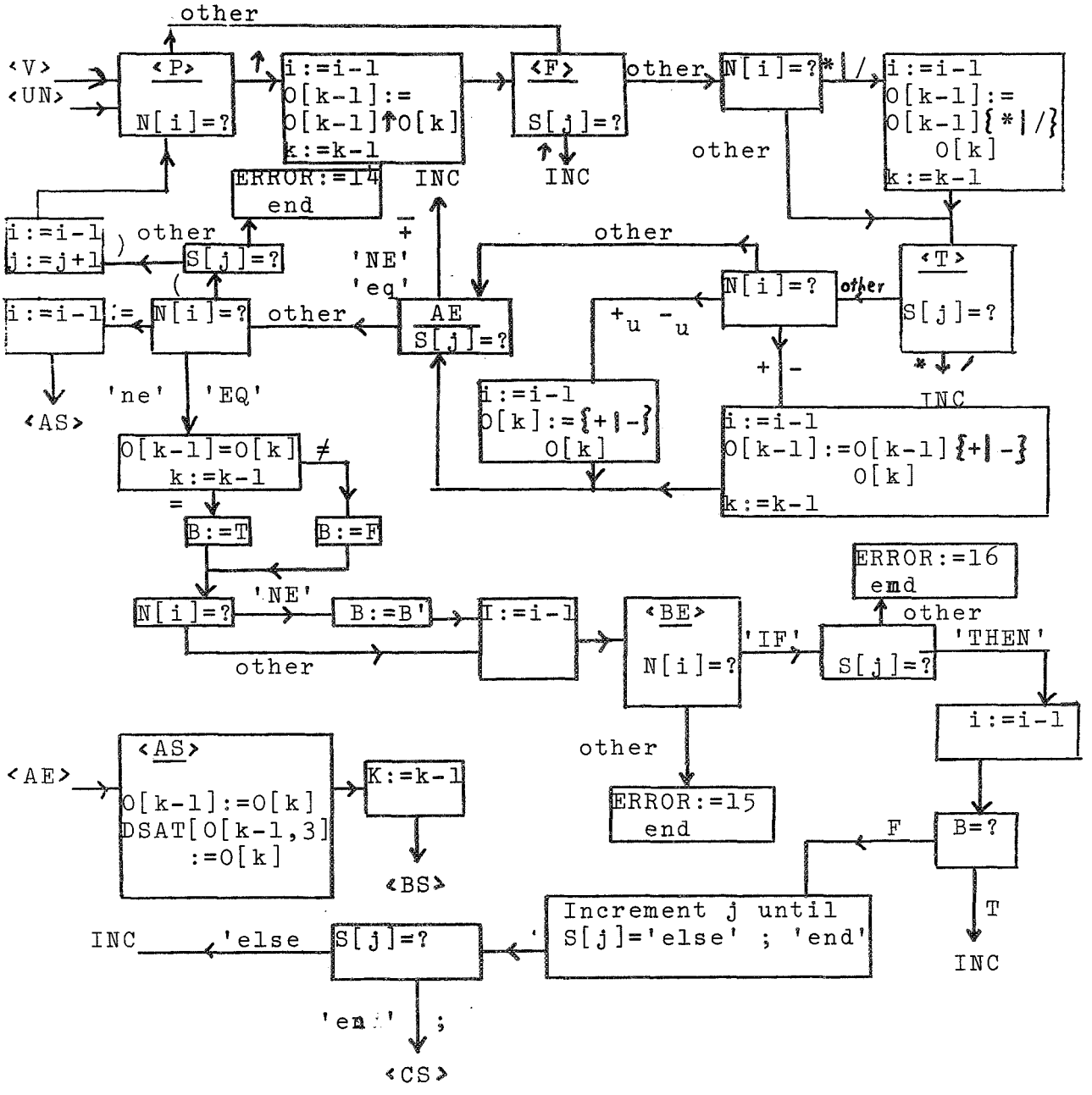

Fig. 3 Syntax Checker and Execution for Primary, Factor, Term, Arithmetic Expression, Boolean Expression, and Assignment Statement

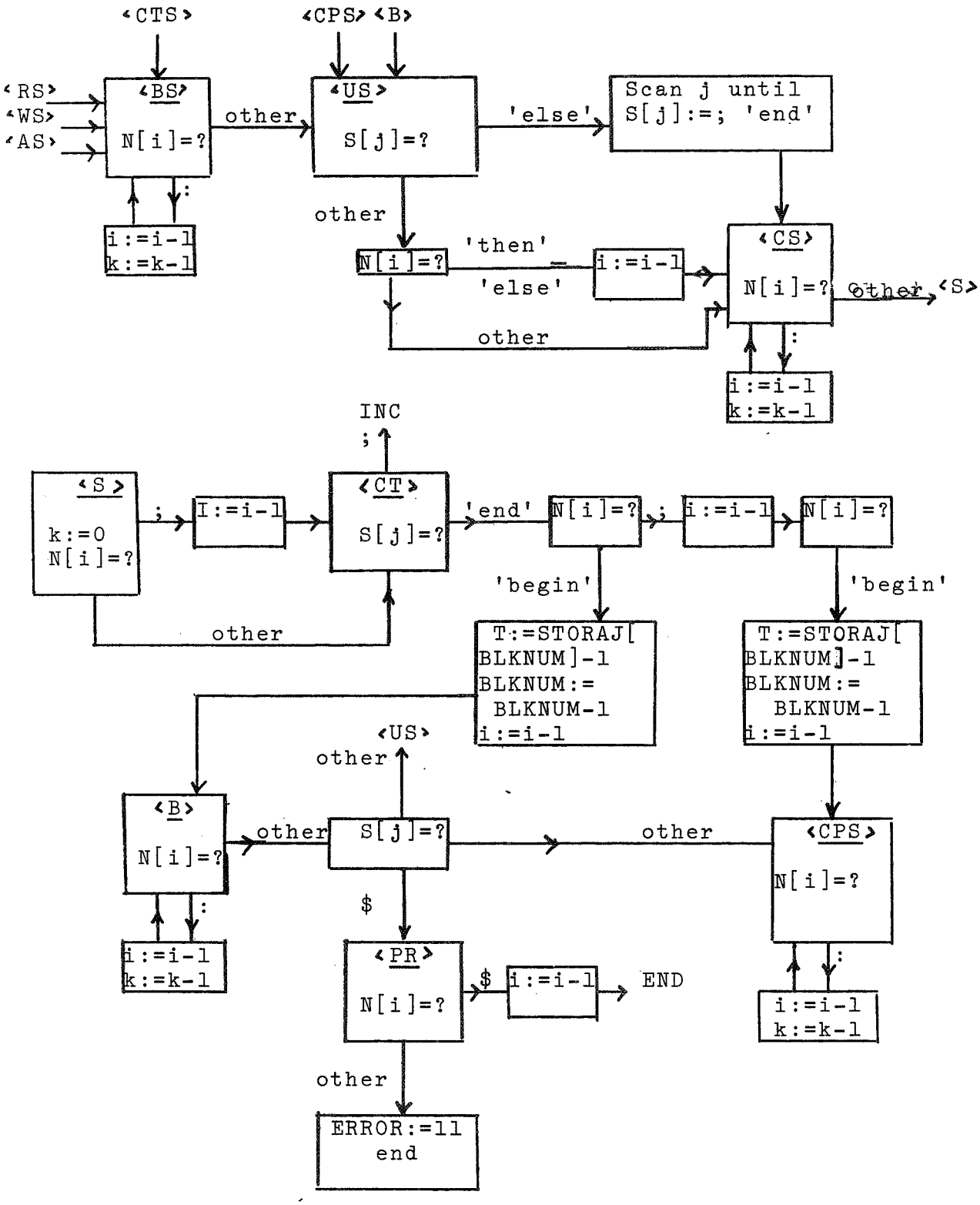

Fig. 4 Syntax Checker and Execution for Basic Statement, Unconditional Statement, Conditional Statement, Block, Compound Statement, and Program

3. CDL Description of the ALGOL Computer

At this point we are ready to transform the interpreter software design described in the last section into the actual computer hardware. A good example of how CDL is used to describe conventional computers that operate on a machine language can be found in (5).

3.1 Configuration

The configuration of the computer is shown in Figs. 5 and 6. Fig. 5 shows the memories, white Fig. 6 shows the other elements.

3.1.1 Memory Configuration

The ALGOL computer contains a main memory, one input channel, one output channel, and four pushdown stacks (Figure 5). Because of limitations in the CDL terminology, these seven elements will all be defined as memories. The elements are defined in the following manner.

(1) MEM - This element is the main memory of the computer. Its contents is the program string for a given ALGOL program. The program is stored one character to a location to correspond to the six bit size of the BCD code. The string begins in location zero. The address register J corresponds to the index of the program string and the buffer register S corresponds to the BCD code of the string character. Special characters such as 'end' that take up more than on memory location are placed in the SA register with only the three lower order characters being stored. The registers S0, S1, and TEMP are used for cascading purposes.

(2) INP - This element is the input channel of the computer system. At loading time, all the input data is stored on the channel in queue form. The address register, IN, is initially zero. After a request for a datum is made through a read statement, IN is incremented. The datum is placed in the buffer INA. The word size permits a sign bit and an integer number with a value less than 2^{18}.

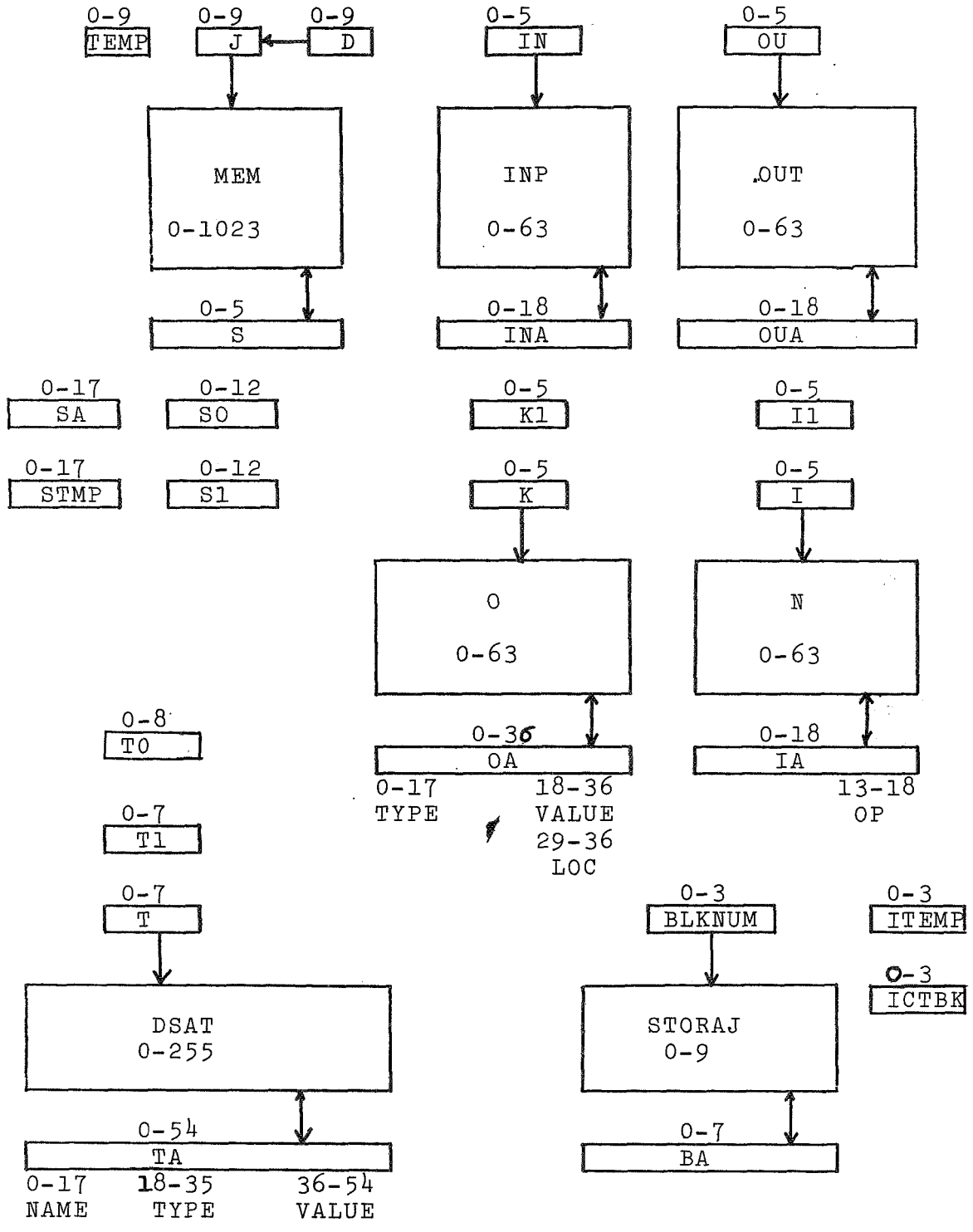

Fig. 5 Configuration of Computer Memory

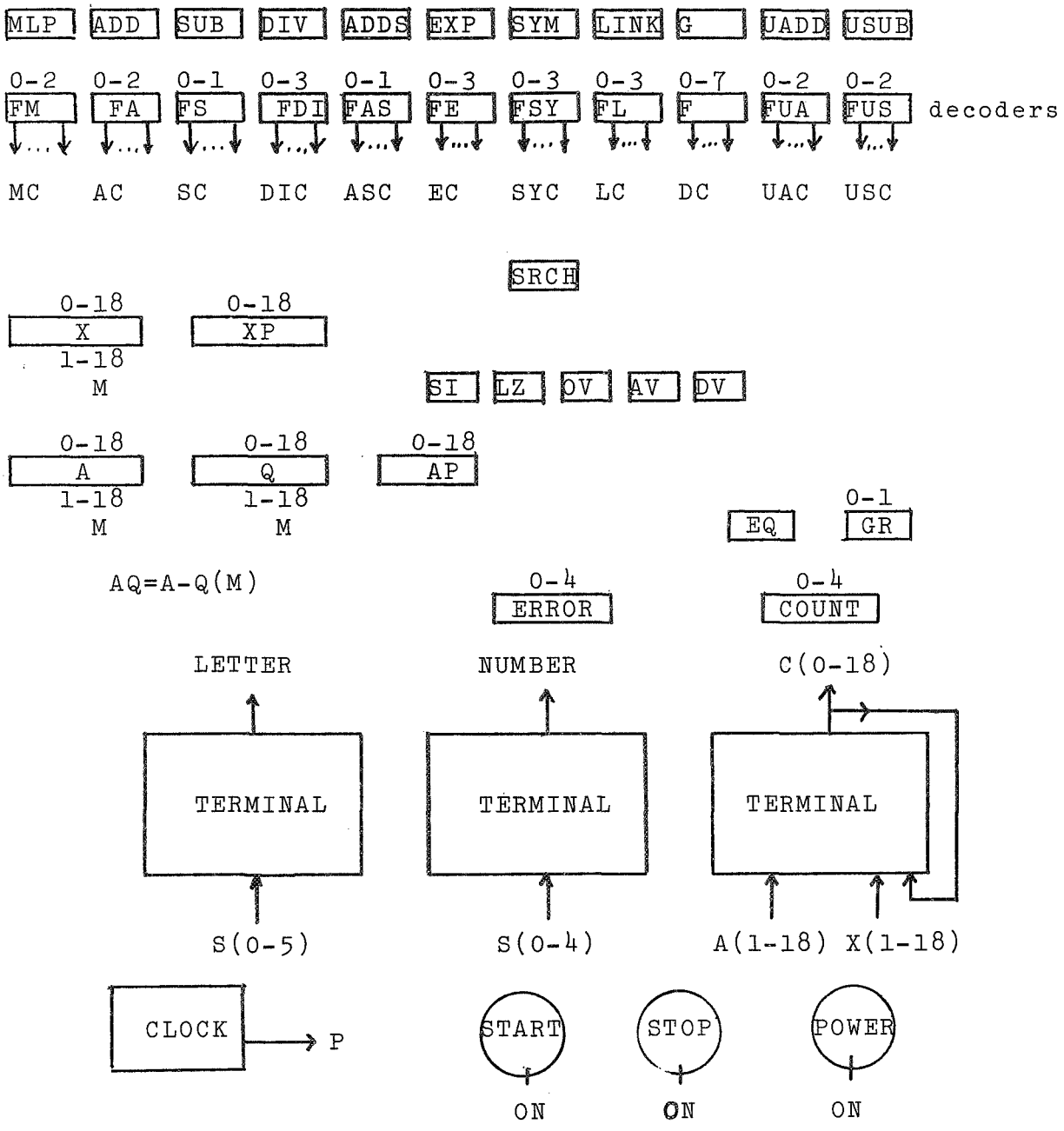

Fig. 6 Configuration of Nonmemory Computer Elements

(3) OUT - This element is the output channel of the computer system. The address register OU is initially set to zero. When the write statement is recognized, the datum corresponding to the variable in the write statement is passed from the DSAT into the buffer register OUA which is then placed onto the channel in queue form.

(4) 0 - The element is the operand pushdown stack and performs the exact operations as defined in Section 2. Its buffer register and consists of two subregisters for storing the type and value inf ..ion. It has the address register K and an auxillary register K1 used to store temporarily in certain operations the value of K.

(5) N - The element is the operator pushdown stack and performs the same operations as defined in Section 2. Its buffer register is IA and its address register is I. There is an auxillary register I1 used to temporarily store the value of I.

(6) DSAT - This is the dynamic storage allocation table (or pushdown stack). All identifiers recognized by the computer are stored in this table. Its address register is T and its buffer register TA is divided into three subregisters to correspond to the name, type, and value of the identifier. The registers T1 and T0 are used for cascading operations.

(7) STORAJ - This is the auxillary stack for the DSAT used in handling block nesting. Its address register is BLKNUM and its buffer register is BA. The registers ITEMP and ICTBK are used in nesting algorithms.

3.1.2 Arithmetic Unit

The arithmetic unit consists of five registers. The algorithm for addition, subtraction, multiplication, and division were taken from (5). The algorithm for exponentiation involves repeated multiplication.

(1) Storage register X - This register is used to store the value of 0[k] in binary operations.

(2) Accumulator A - This register is used to store the value of 0[k-1] in binary operations and also stores the final results of the operations.

(3) Register Q - This is the standard extension register.

(4) Register AP - Auxillary register used in exponentiation.

(5) Register XP - Auxillary register used in exponentiation.

In addition to the five registers there are five special bits to indicate the sign (SI), logical zero (LZ), overflow (OV), addition overflow (AV), and division overflow (DV).

3.1.3 Hardware Subroutines

There are eight hardware subroutines used for performing arithmetic operations. When an operation is required, for example, multiplication, a special bit, MLP, is turned on. At this point the subroutine assumes that the operands have already been stored in X and A. After the operation is completed, the bit is turned off and control returns to that part of the computer that originally caused the bit to be turned on. The eight bits: MLP, ADD, SUB, DIV, ADDS, EXP, UADD, AND USUB; have a corresponding counter and decoder as shown in the figure. The names of the bits are self-explanatory. The sequence charts for these routines appear in Figures 7, 8, 9, and 10.

There is a special hardware subroutine used for determining whether the new character S_j is a simple character such as "/" or a special character such as 'end' that takes more than one location in memory. The bit SYM is turned on each time a call to the memory is made to fetch a new character. Associated with SYM is the counter FSY and decoder SYC. One can see how the fetching is done in the sequence chart (Figure 11). When a character is fetched, a check is made to see if S='. If so, fetching continues until the

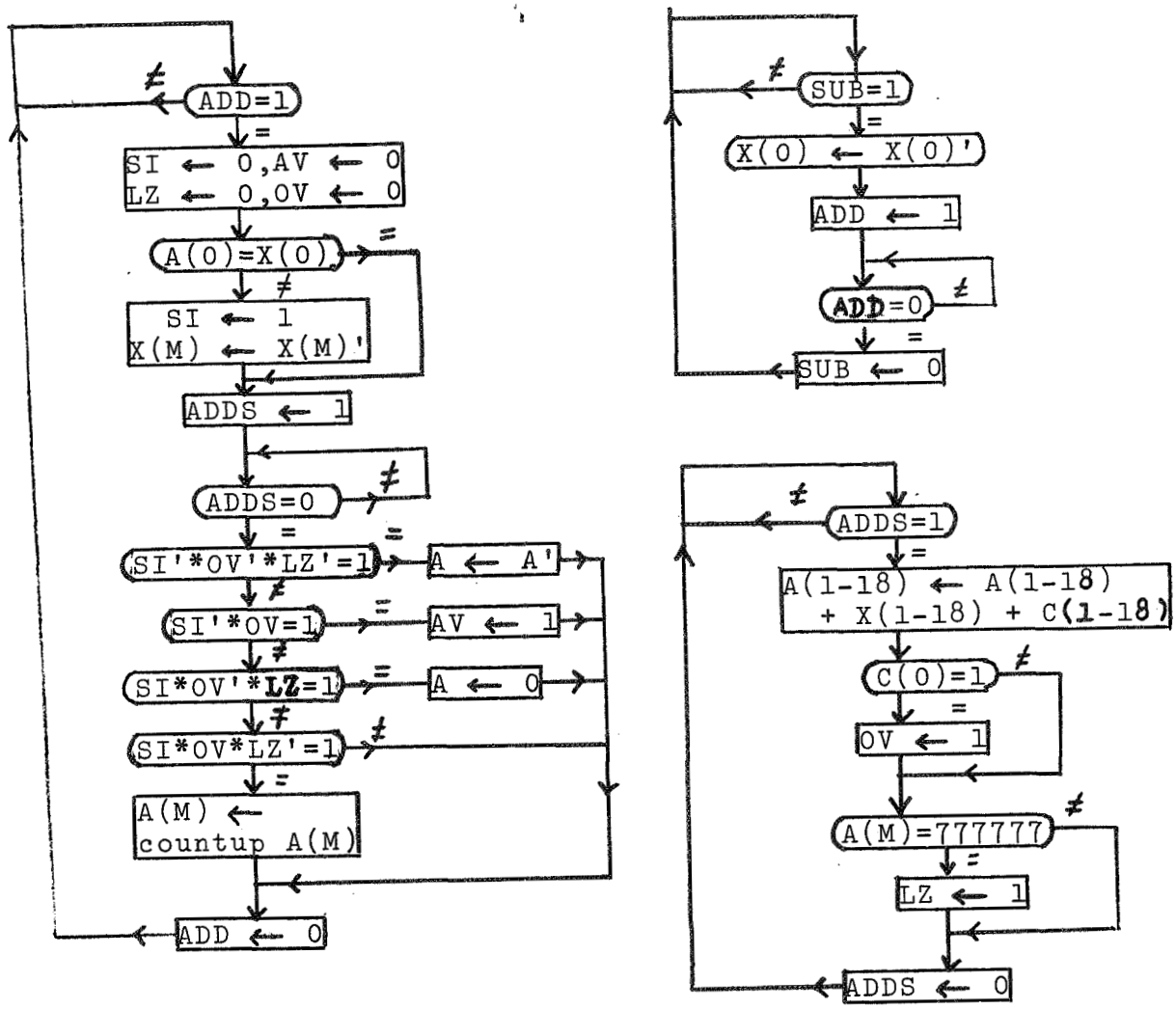

Fig. 7 Sequence Chart for ADD, Subtract, and Add Subsequence

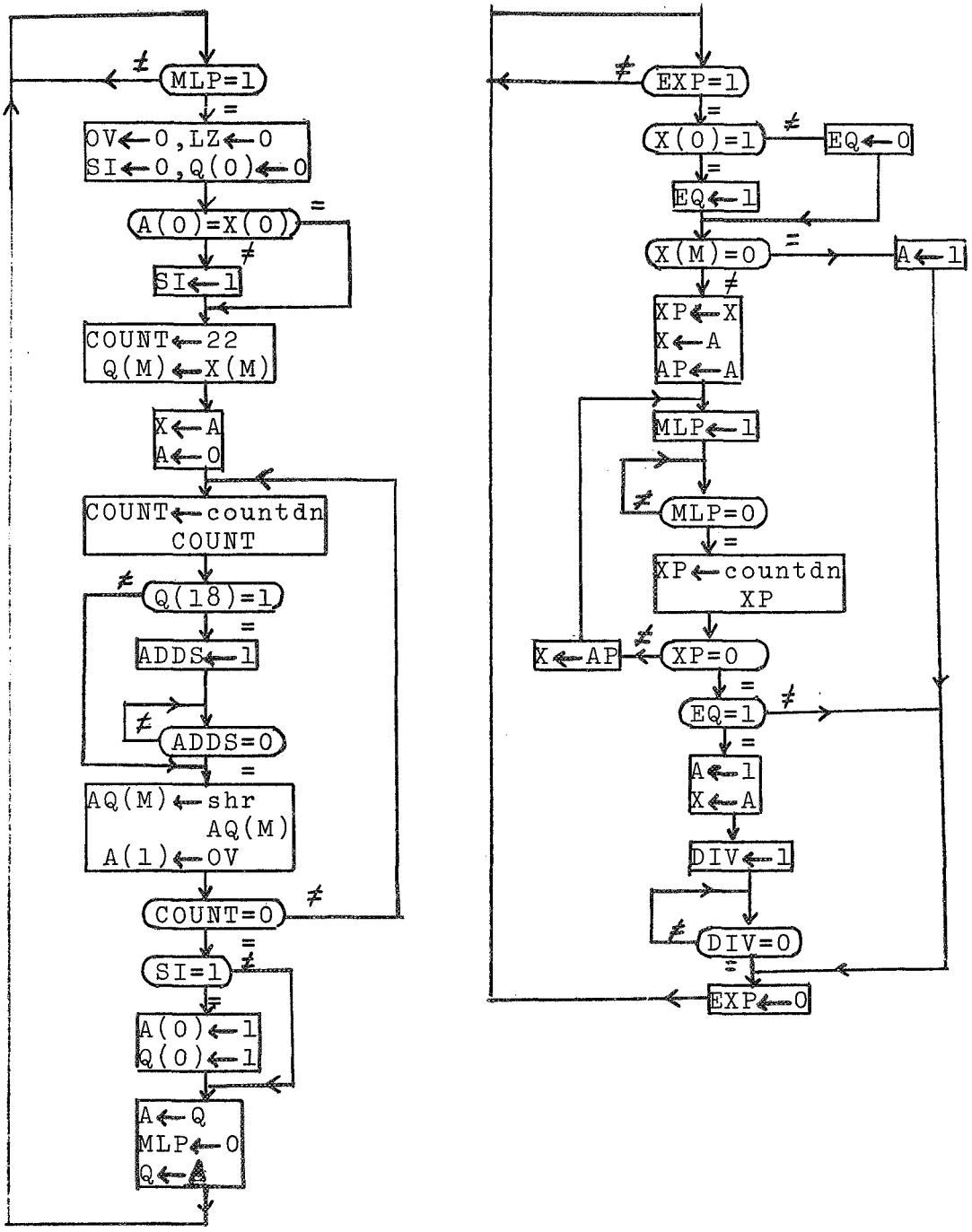

Fig. 8 Sequence Chart for Multiplication and Exponentiation

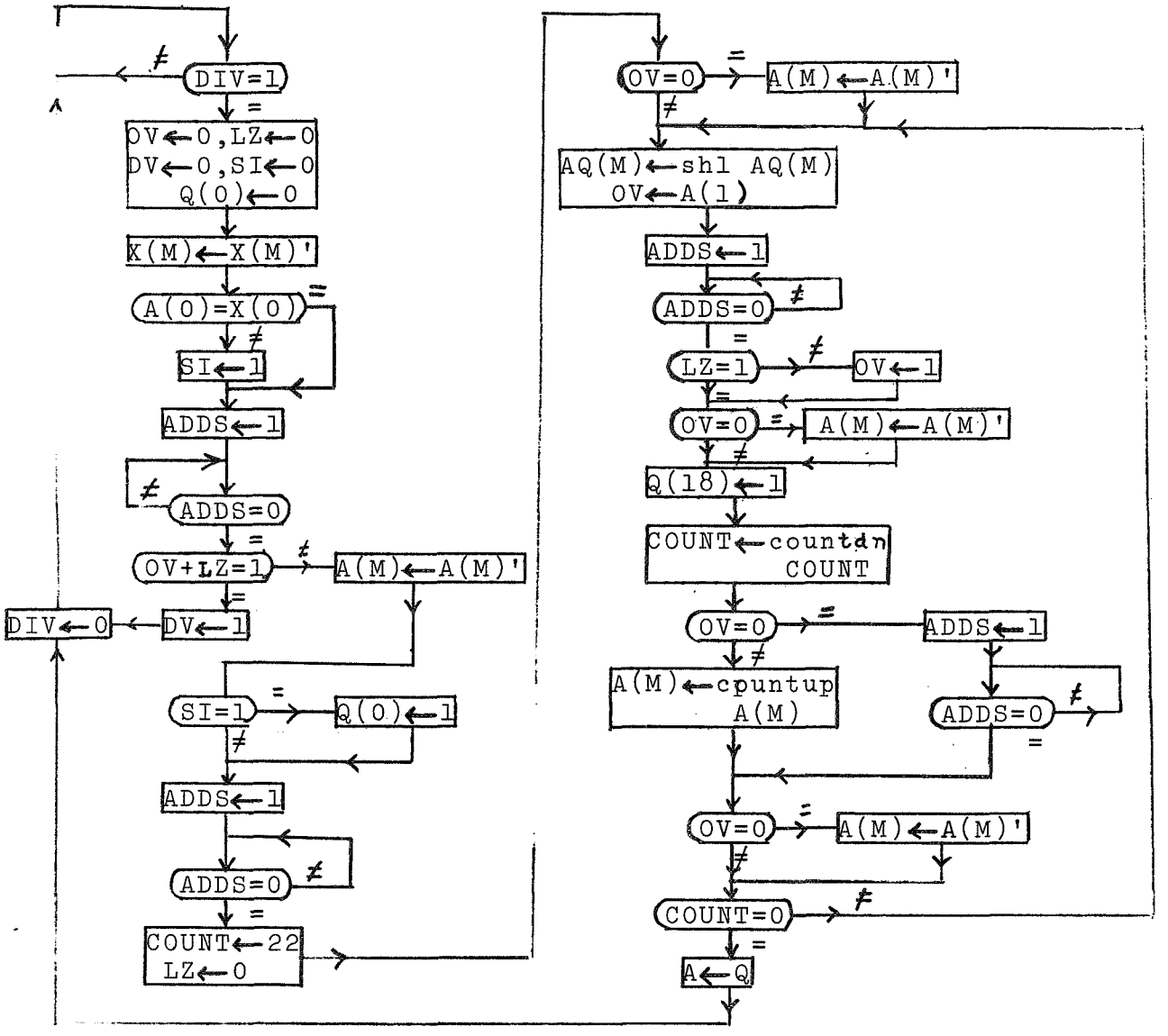

Fig. 9 Sequence Chart for Division

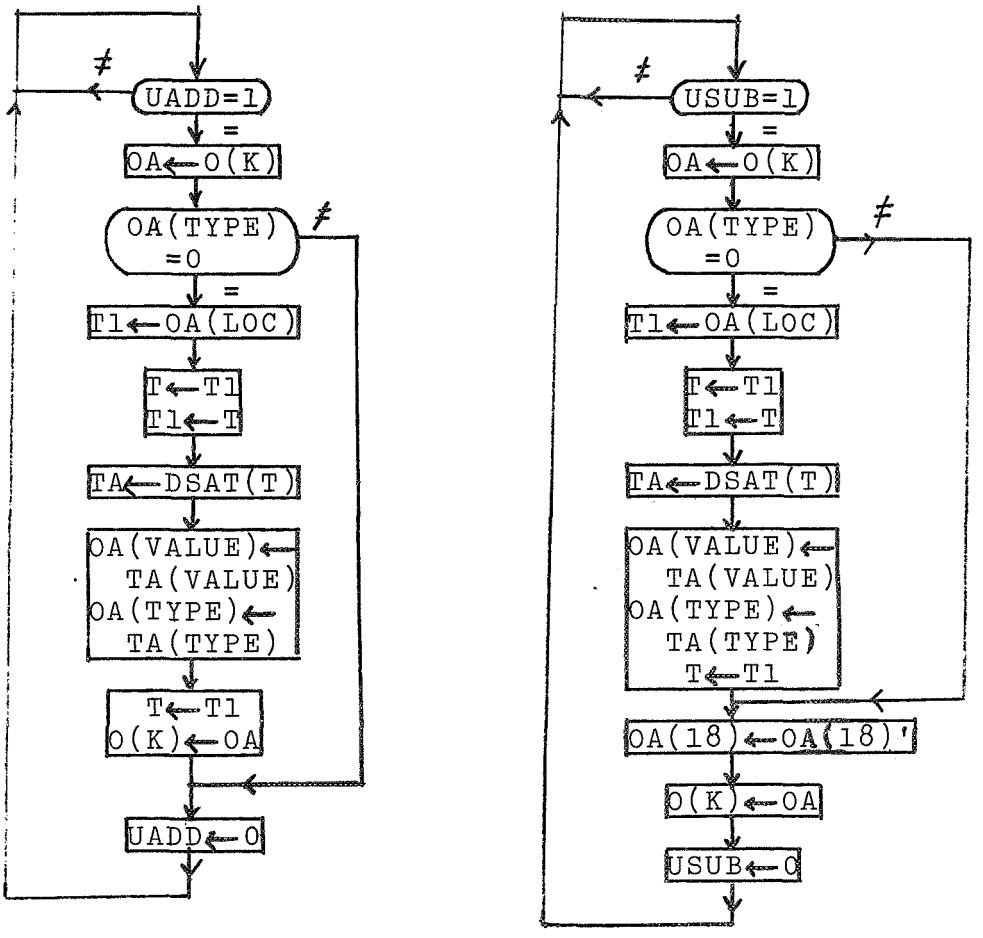

Fig. 10 Sequence Chart for Unary Addition and Unary Subtraction

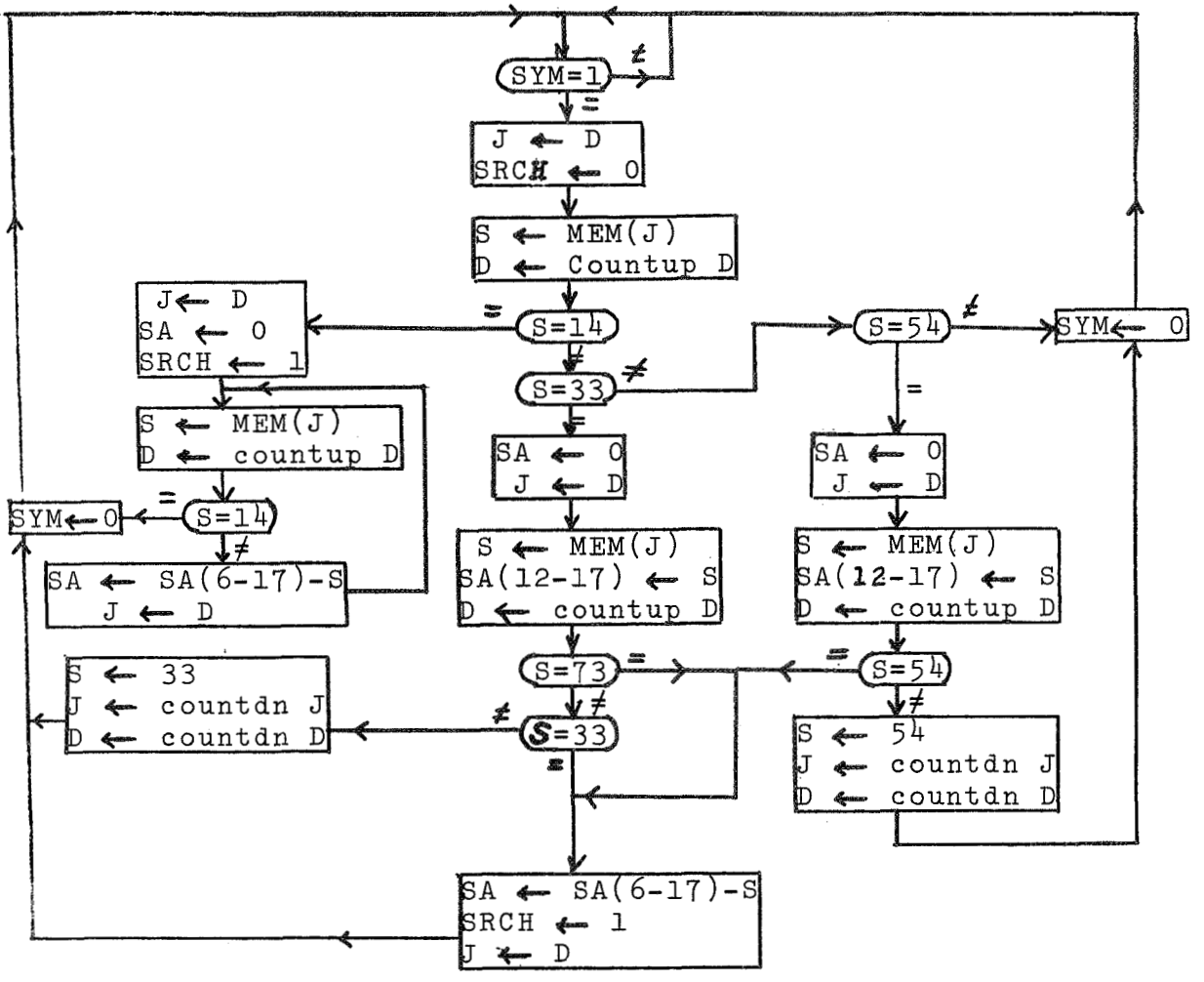

Fig. 11 Sequence Chart for Fetching Special Characters or Simple Characters from Memory

next apostrophe is scanned. The register SA is set to the last three characters fetched and a bit, SRCH, is turned on to indicate that a special character has been fetched from memory. This process is also done if S="." or "*". In this case if new fetching yields the sequences "..", ".,", or "**", the sequence is stored in SA. Otherwise the character remains in S and SRCH remains off.

There is a special hardware routine used before initiating binary operations (Figure 12). The routine begins when the bit LINK is turned on. At this point, it is possible that the value stored in O[k] or O[k-1] may contain the row in the DSAT where the identifier is stored. The routine checks to see if the type of the operand is "link" (indicated by a zero value). If so, the value of the identifier in the DSAT is transferred into the operand stack. At the same time, the value is also placed in X for the top operand, and in A for the next to the top operand.

3.1.4 Terminals

It will be necessary to know whether S is a letter or a number. If we consider S as being made up of six logical variables then $S=S_0S_1S_2S_3S_4S_5$. Using the BCD code for possible letters or integers, one can construct a truth table and then find a boolean expression giving a true value for number or letter.

The boolean expression for number is,

$$\text{number} := S_0'S_1'(S_2' + S_3'S_4').$$

from which the terminal statement is obtained and shown later. The boolean expression for letter is,

$$\text{letter} := S_0'S_1'S_2'(S_3+S_4+S_5)+S_0S_2'(S_3+S_4)+(S_0+S_1)(S_2S_3'S_4')+S_0S_1'S_3'S_4'S_5$$

from which the terminal statement is also obtained.

The other terminal is used to construct the carry expression for binary addition.

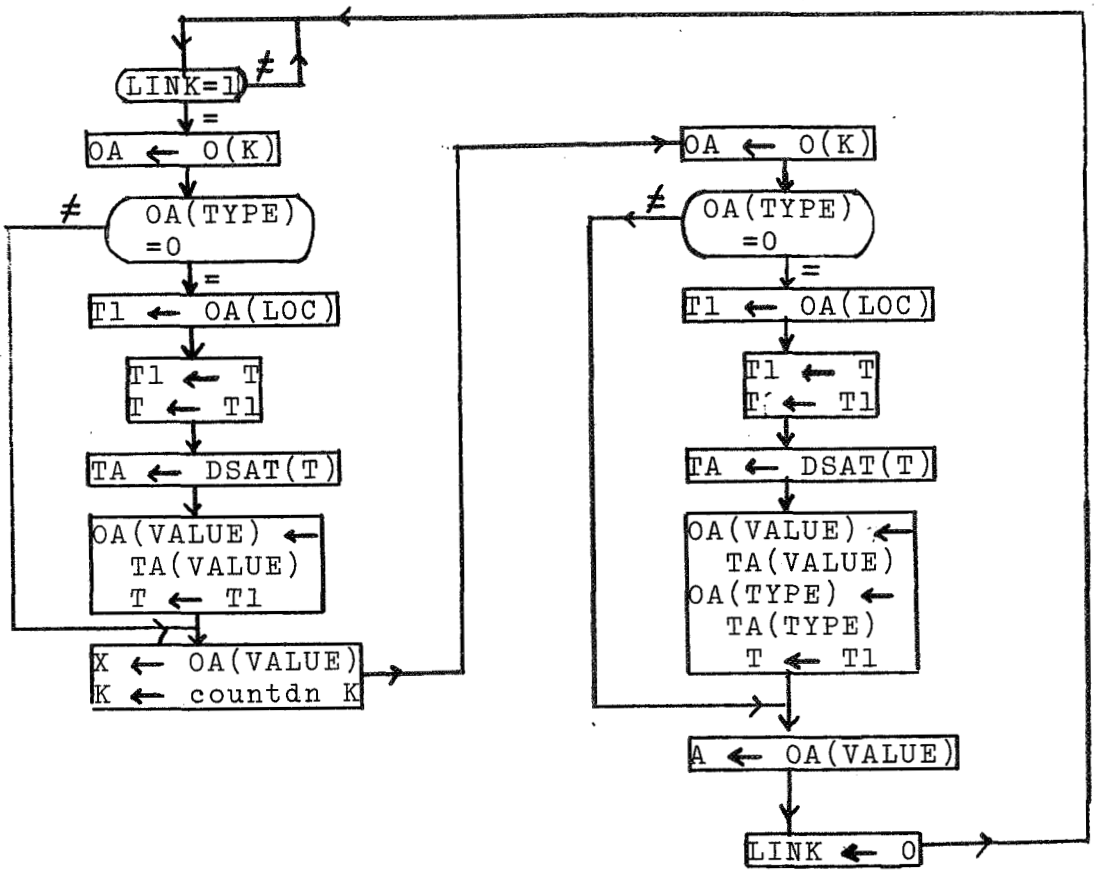

Fig. 12 Sequence Chart for Chainging Operand From Link to Value

3.1.5 Other Elements

The elements EQ and GR, representing equal or greater than, are used in ring operations along with the register COUNT. The register ERROR is used to store a number corresponding to a syntax error. The bit G with the counter F and decoder DC is used for overall program control. The clock P is used for sequencing the microoperations, and the three switches POWER, START, and STOP are used for turning the power on, starting the sequencing and stopping the computer.

3.2 Sequence Charts

Now that the computer elements have been described, we can begin discussion of how the computer operates. The operation can be thought of as two distinct cycles. The first cycle corresponds to the initial point box in the syntax checker flowchart. A symbol in the program string is fetched from memory. Either this symbol is an operator and the symbol is stored on the operator stack and the cycle is repeated, or the symbol is a letter or digit and control goes to the unsigned number or identifier box. This latter case will be defined as the second type of cycle which can be denoted as the execution cycle. We will call all operations centering around nonterminal boxes as occuring during the execution cycle.

3.2.1 Initial Point (or Fetch Cycle)

The sequence chart for the fetch cycle (Figure 13) is an analog to the initial point box and increment box (Figure 1). When the power switch is turned on, all the special bits for the hardware subroutine are set to zero. The addresses for the memories are also set to zero. The computer will then go into a wait cycle until the start switch is pressed. Then the first character is fetched from memory. As will be the case whenever any hardware subroutine is called, the computer will cycle about its present point until the routine operation is finished.

The fetching will continue until a digit or letter is recognized. In reading any sequence chart, it is important to remember that the register S represents a simple character and the register SA represents a special character resulting from the concatenation of simple characters because of keypunch symbol limitations. The two registers are never used at the same time. If the flipflop SRCH is on, that means that SA is used; if SRCH is off, then S is used. In all the charts the octal number representing the BCD code of

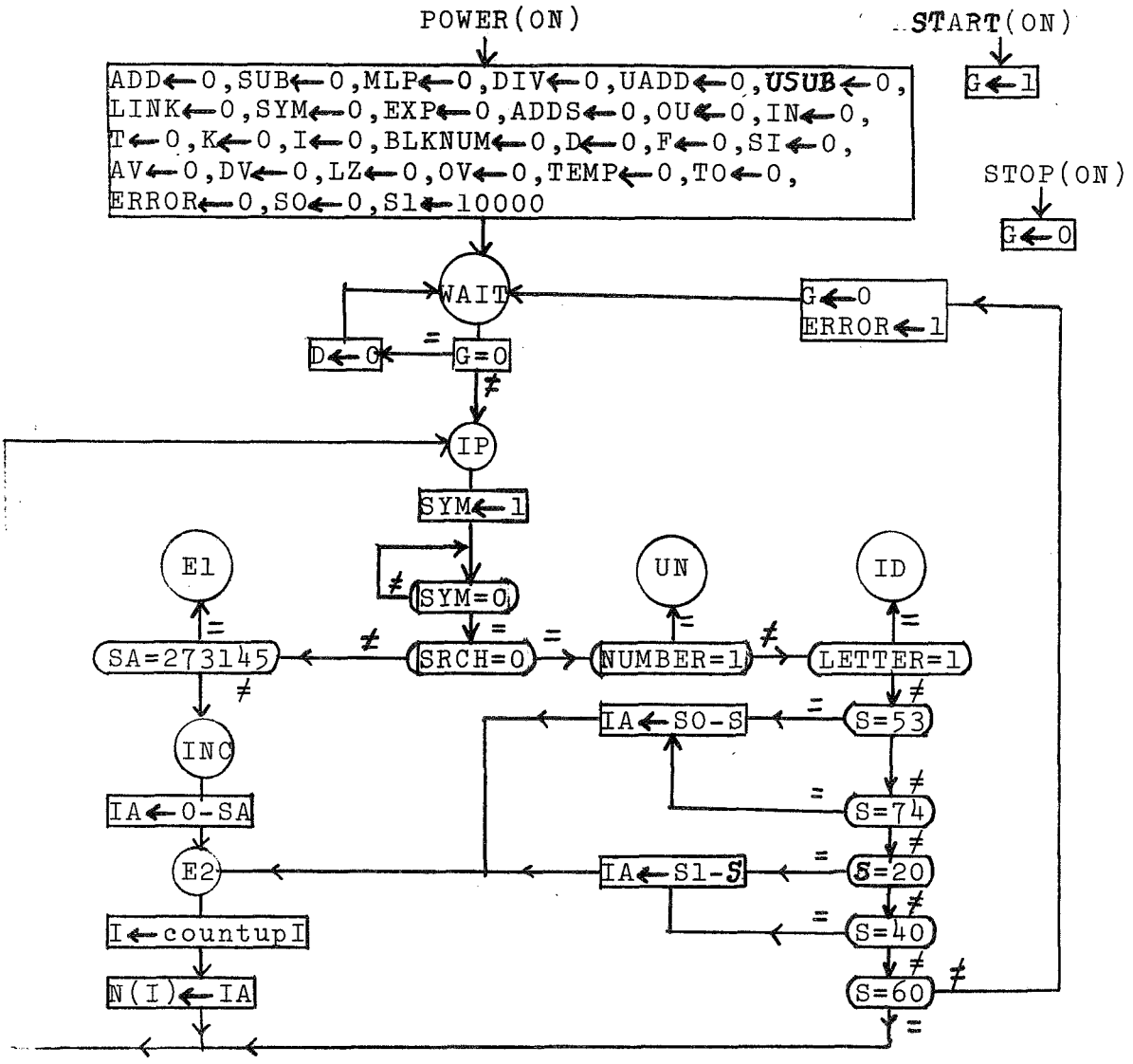

Fig. 13 Sequence Chart for Initial Point and Increment Box

the last three characters of a special symbol is used. The reader should refer to Table 2 whenever he cannot recognize the octal number. The fetching operation can be separated into two cases:

(a) Symbol is not special character - This is the case when SRCH=0. If the character is a number or letter, control goes to the appropriate nonterminal point. If the character is "$" or "(" then it is placed down on the operator stack. A special register S0 containing all zeros is cascaded with S to correspond to the word size of N. If the character is a "+" or "-", this means that the operator is unary and a special register S1, containing a one bit in the first bit position, is cascaded with S before storing in N. If the character is a blank, then it is ignored and another fetch is made. For any other characters that are recognized, the error register is set to a value of one and the computer is placed into the wait cycle.

(b) Symbol is special character - If the special character is not 'begin', then the character is simply pushed down on the operator stack and a new character is fetched.

If the character is 'begin', the operations discussed in Section 2.2.3 are done. The block number is incremented (operation 1) and a fetch cycle is begun that will examine all characters in that block to find the labels (Figure 14).

The operation can be defined by the following steps:

Step 1. Temporarily store the position j in the input string of 'begin' in register TEMP so that when the operations are completed, control can return to that point.

Step 2. Fetch the new character S[j] from memory.

Step 3. Let STMP be a register storing the last three characters fetched. Whenever a ";" is scanned, STMP will contain the name of the label. It cannot

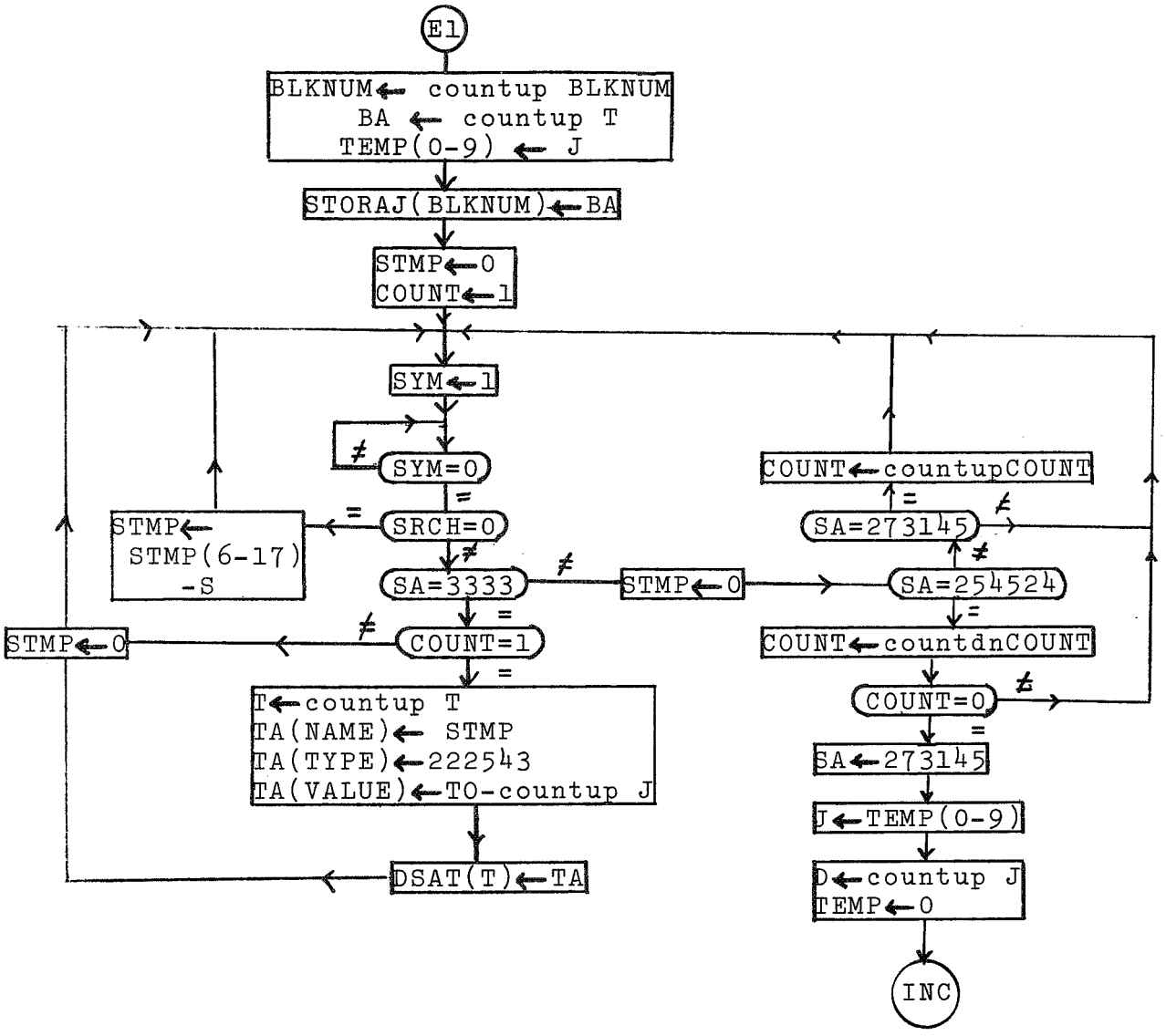

Fig. 14 Sequence Chart for Placing Labels in the DSAT

contain any special characters or operators because the sequence " 'end'<L>: " or " ;<L>: " must occur in recognizing a label and when 'end' or ";" is scanned, STMP is cleared. Thus after the label itself is scanned, STMP must contain either the last three characters in <L> or leading zeros followed by one or two characters that make up the label.

Step 4. If S[j] = 'begin', increment the degree of block nesting, indicated by the COUNT register. If S[j] = 'end', decrement the nesting degree. If COUNT = 0, then the block has been completely scanned and the memory address register J is reset to the position it held before scanning the block. The character 'begin' is placed in the register SA and stored in the stack N. A new fetching cycle begins at the initial point. If S[j] is neither of the above, return to step 2.

Step 5. If S[j] = ":" and the degree of nesting is one, then the label is stored in the DSAT according to operation 3 in Section 2.2.3. Otherwise return to step 2.

3.2.2 Unsigned Number

Upon reaching this point, the symbol S is placed on the buffer register OA (of the operand stack 0) in the value subregister (Figure 15). Its type, 'integer', is stored in the type subregister. Then new symbols are fetched until a symbol is recognized that is not a digit. At this point the number has been computed and stored in the buffer. The contents of the buffer is now put into the operand memory and control goes to the primary logic. This is the result of the production that states that all unsigned numbers are reduced to primaries. The algorithm for computing the number is given below:

Step 1. Set OA = 0

Step 2. For all digits S[j], $OA=12_8 * OA + S[j]$

3.2.3 Identifiers

When the letter has been recognized in the initial point, the production that a letter is reduced to an identifier is imposed and thus control now goes to the identifier section (Figure 16). The letter is stored in the buffer register OA. Fetching is initiated and continued until the new character is neither a digit or a letter. The characters to this point have been ring shifted in the OA register to correspond to the production reducing a sequence of letters and digits to an identifier. Only the last three characters are stored.

Now the identifier whose name is stored in the buffer register is compared with all the identifier names stored in the DSAT until a match is found. The register COUNT is used to keep track of how many times the subregisters OA(VALUE) and TA(NAME) have been ring shifted. Regardless of a match, the shifting must occur 22_8 times so that the names are properly stored after the comparison is completed. If a match is successful, the flip-flop EQ is set to one; otherwise EQ is zero.

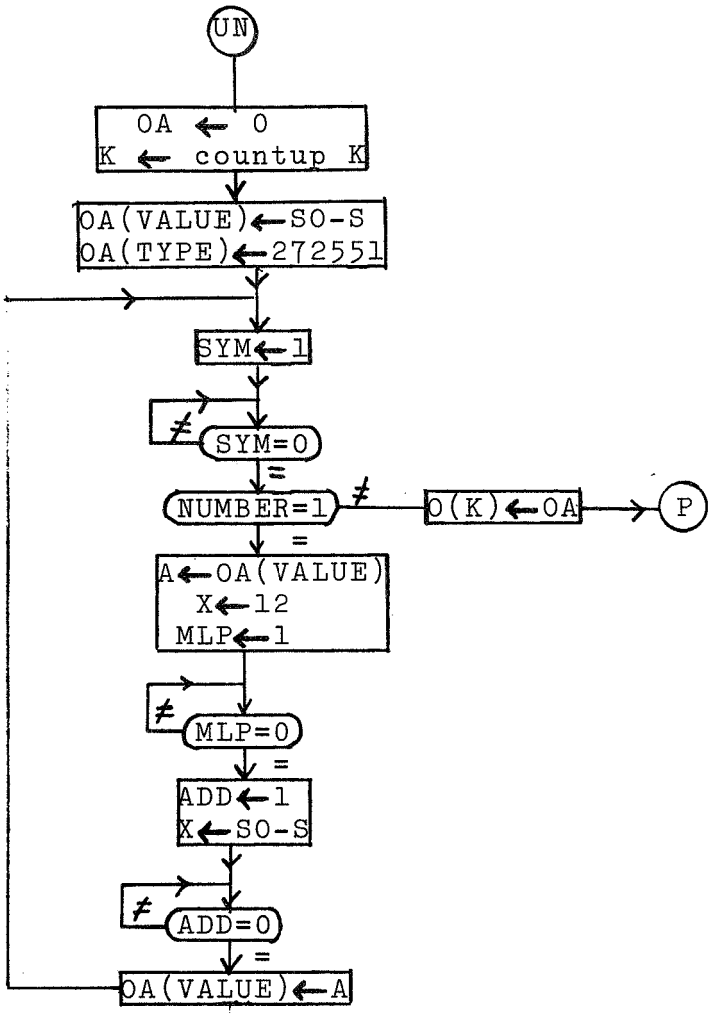

Fig. 15 Sequence Chart for Number Box

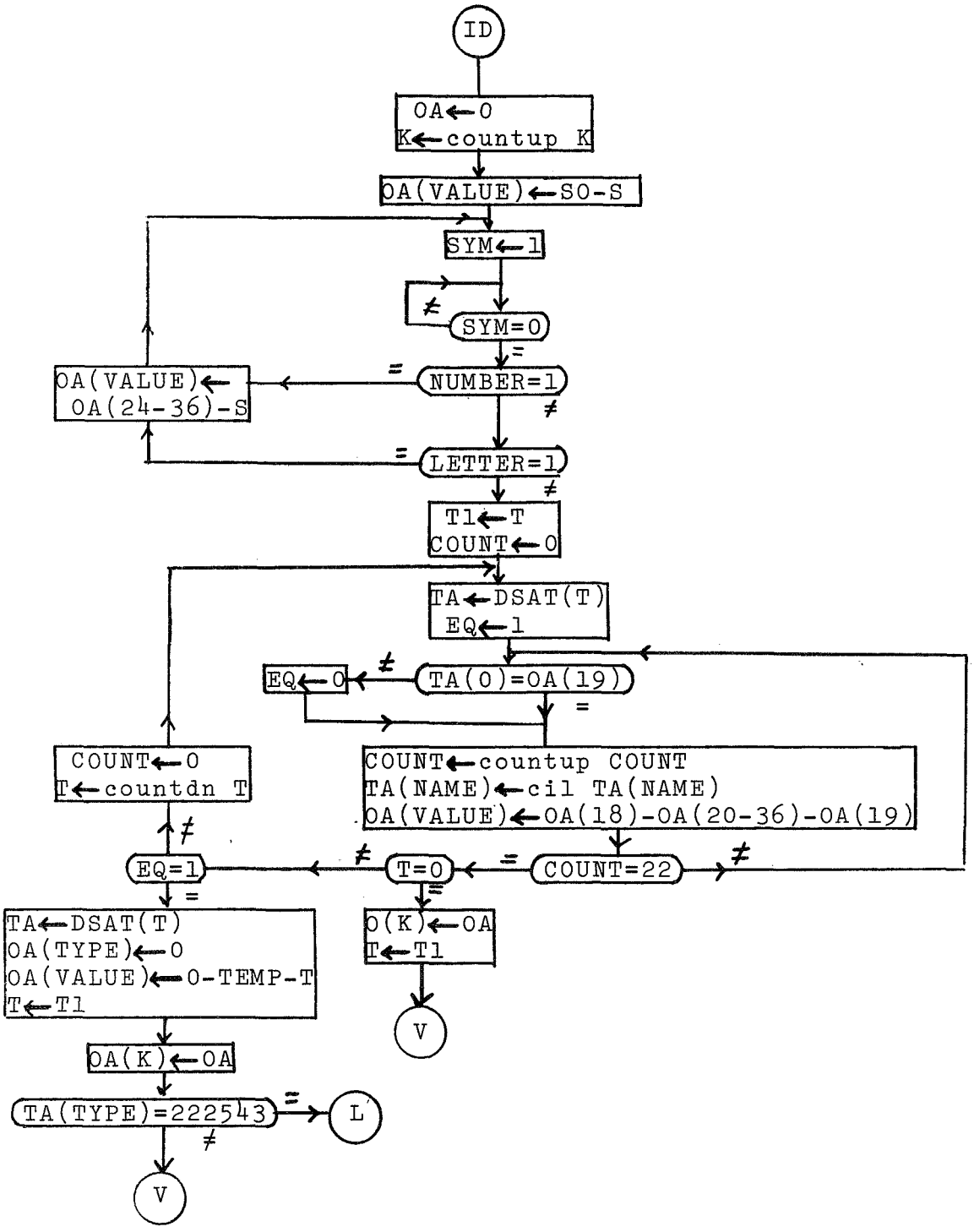

Fig. 16 Sequence Chart for Identifier Box

The reason for finding a match is the following: If the identifier has been previously declared, then the contents of the buffer register should be changed from the name of the identifier to the row T of the DSAT where the identifier is stored. Also from the type subregister of TA, we will know whether the identifier is a label or a variable. This then invokes either the production reducing the identifier to a label or a variable. Before transferring to the new nonterminal, the contents of the buffer register containing the link information is put into the operand memory. If no match was found (T=0), the computer assumes that it is in the middle of a declaration. The identifier is reduced to a variable.

3.2.4 Variable

A variable may appear in one of five productions: primary, read statement, write statement, assignment, and type list. Hence there are also many possible adjacent terminals to consider. They are ":=", "," , "(", " read", "write", ")", and indirectly "integer". Depending on which terminal is present in the N[i] or S[j] position, the computer will proceed as shown in the sequence chart (Figure 17). In the absence of any of the above symbols, the variable will be reduced to a primary.

3.2.5 Read Statement

The read statement sequence (Figure 17) is concerned with bringing an integer number off the input channel and into the DSAT. The address register IN for the memory INP could be thought of as a read mark on a tape. The information is read from the tape and the mark advances. At this point, O[k,2] corresponding to OA(VALUE) holds the row value in the DSAT of the variable whose value is desired from the input channel. The operation can be described as follows:

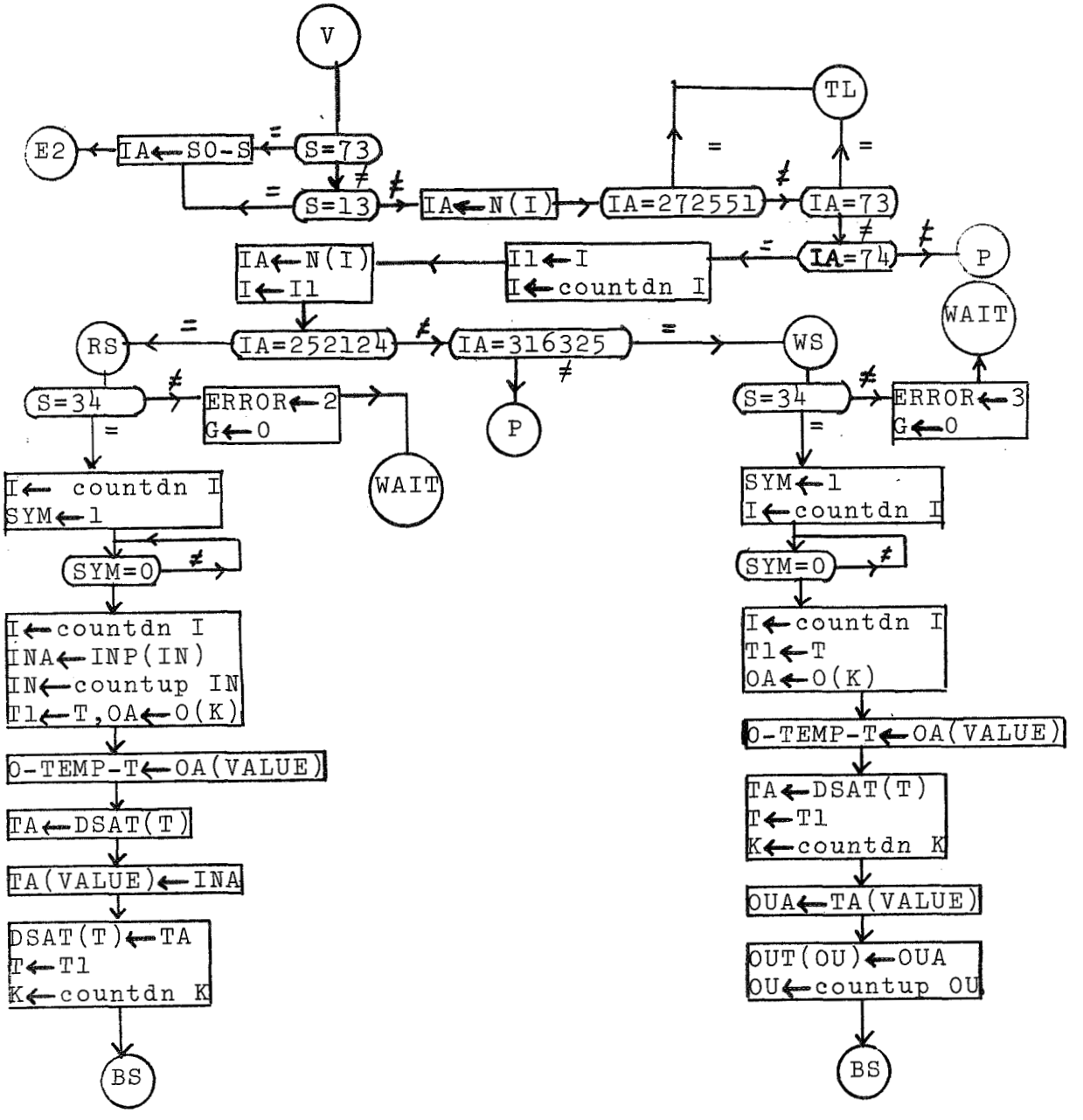

Fig. 17 Sequence Chart for Variable, Read Statement, and Write Statement

DSAT[O[K,2],3]=INP[IN]

IN=IN+1

K=K-1

After reading the next symbol S[j] from memory (SYM←1), because the present S[j]=")" was used in the phrase, the read statement is reduced to a basic statement.

3.2.6 Write statement

The write statement sequence (Figure 17) is concerned with writing an integer number from the DSAT onto the output channel. At this point, the variable, whose value from the DSAT is desired, has its DSAT row stored in O[K,2] or OA(VALUE). This value becomes the new address of the DSAT and its contents is placed in the buffer register whose contents is now transferred to the output channel. The operation can be described as follows:

OUT[OU]=DSAT[O[K,2],3]

OU=OU+1

K=K-1

After reading the next symbol S[j] from memory (SYM←1), because the present S[j]=")" is used in the phrase, the write statement is reduced to a basic statement.

3.2.7 Type List

Before the type list is reduced to declaration, it is necessary to store the names of the variables on the list in the DSAT. The process is shown in the sequence chart (Figure 18). The counter COUNT is used to count the number of variables in the list. This is done by counting the number of commas on the operator stack and adding one for integer. The next sequence is operation 2 described in Section 2.2.3, which is repeated COUNT times. When COUNT is zero, the type list is reduced.

3.2.8 Declaration

There are no execution operations necessary for declarations (Figure 18) except to check and make sure S[j]≠";". If no, there is a syntax error and the computer goes into a wait cycle. If yes, then control goes back to the increment block to continue scanning for the nonterminal block.

3.2.9 Label

The label appears in essentially two productions: (1) Goto statement as a designated transfer and (2) General statements as a reference label. The sequence chart (Fig. 19) shows how a check is first made for N[i]=goto. If yes, then the phrase involving N[i] and the label is reduced to the goto statement. If no, then S[j] must be a colon and is stored in the operator stack.

3.2.10 Goto Statement

The sequence charts (Figs. 19 and 20) show the execution of the goto statement. Before transferring to the statement indicated by the label in the goto statement, two bookkeeping operations must be performed. Since the DSAT is dynamic storage, the variables declared in blocks that are to be in the transfer must be erased from the table. Also the operator stack must

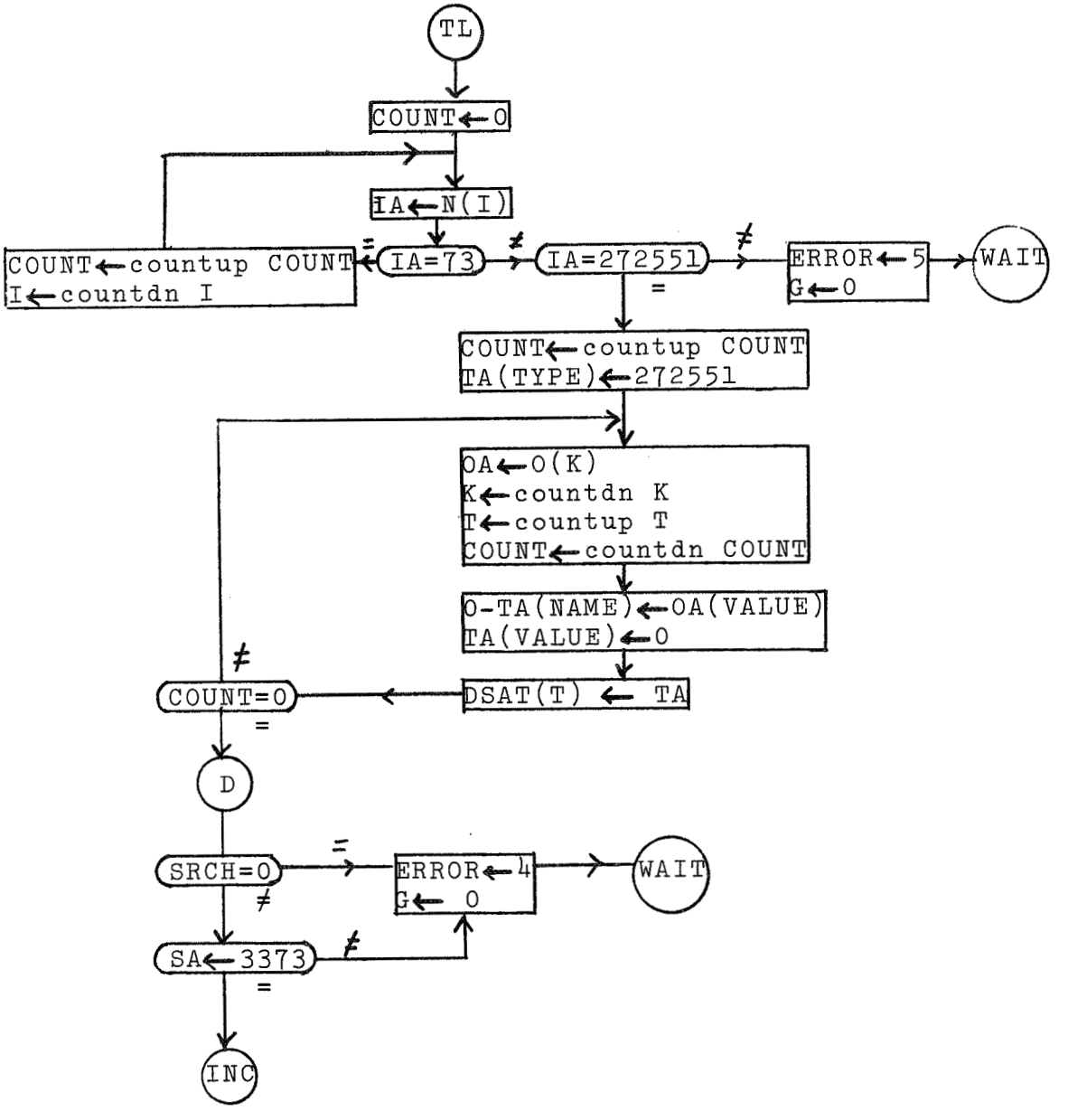

Fig. 18 Sequence Chart for Type List and Declaration

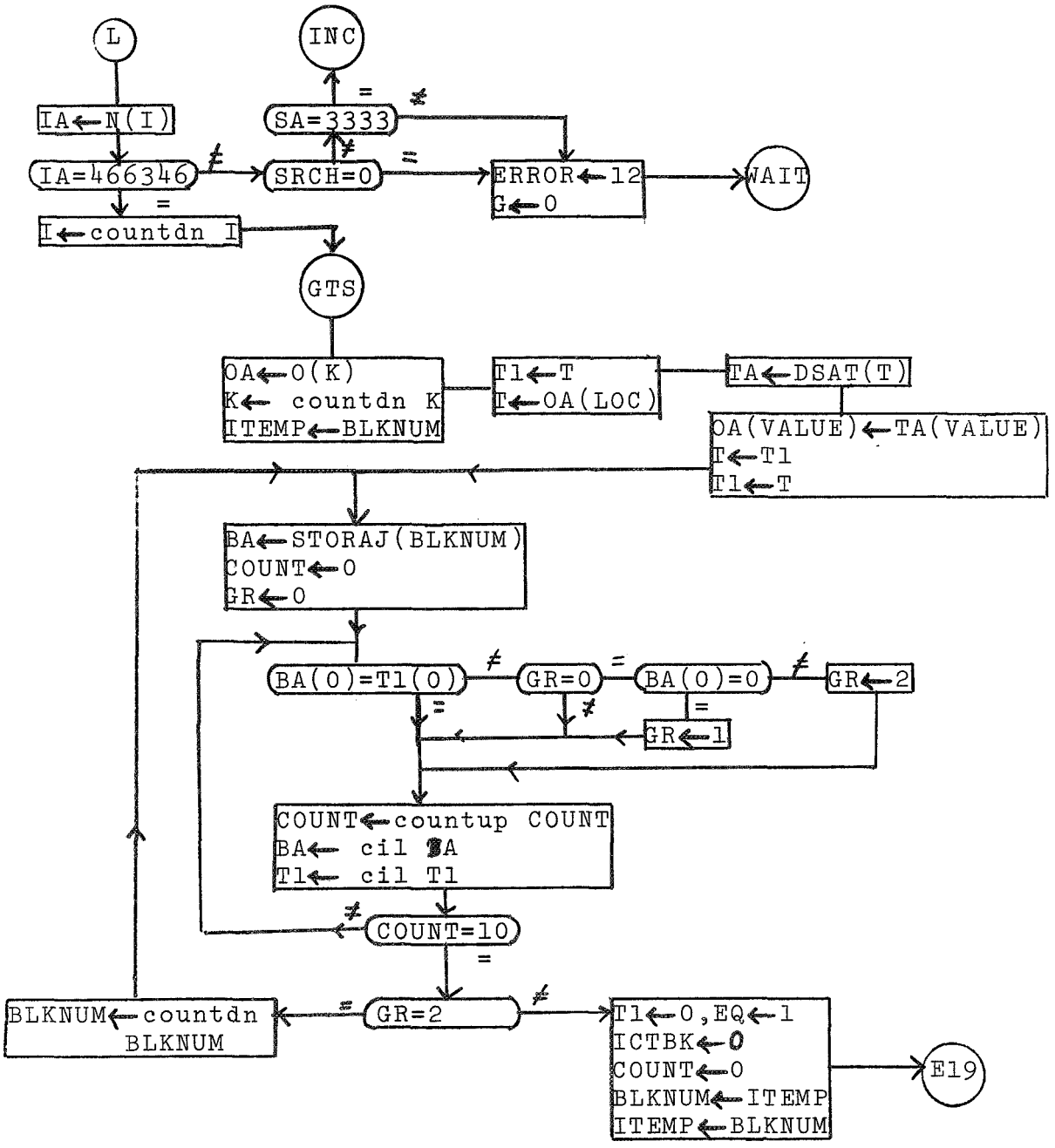

Fig. 19 Sequence Chart for Label and Goto Statement - I

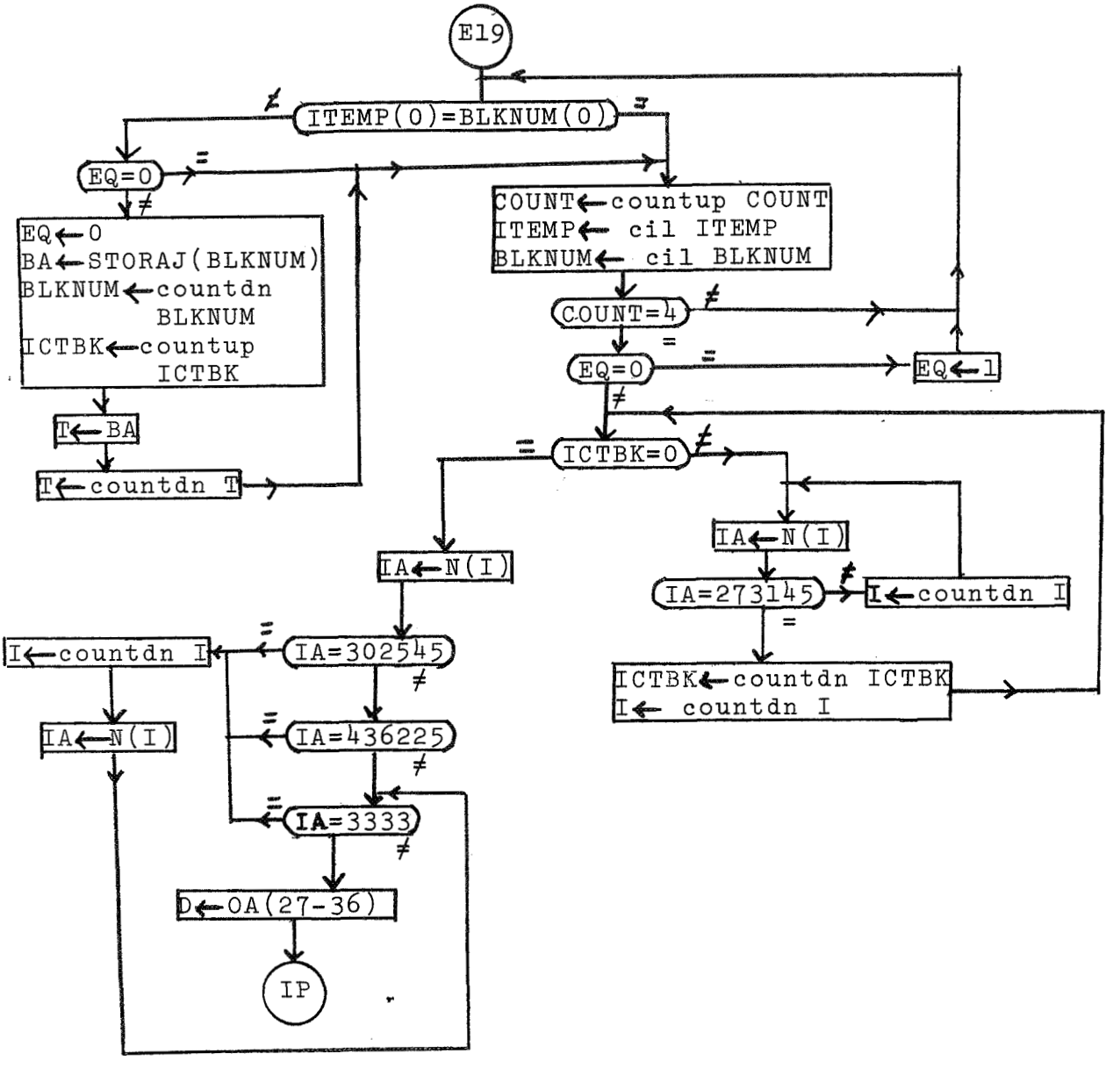

Fig. 20 Sequence Chart for Label and Goto Statement - II

be reduced by the number of begin's that correspond to the number of blocks to be skipped. If the goto statement had a label or was part of a conditional statement, then either ":", or "then", or "else" must also be erased. The operators must be erased so that the syntax checker will operate successfully. The operation of the computer algorithm is described below:

Let the register ICTBK be used to count the number of blocks to be skipped.

Let the register COUNT be used to count the number of bits shifted in BA and T1 or ITEMP and BLKNUM so that the circular shift operation leaves the registers in the original position after the comparison is resolved.

Let T1 be the row in the DSAT where the label from the transfer statement is stored.

Let the register GR be used to indicate whether the contents of BA is greater than (GR=2), equal (GR=0), or less than (GR=1) T1.

Let the register EQ be used to indicate whether the contents of ITEMP (the register stored the initial value of BLKNUM before the transfer operation has begun) is equal (EQ=1) or not equal (EQ=0) to the new value of BLKNUM.

(1) Find the row in the DSAT where the transfer label is stored and place this value in T1. Set OA(VALUE) to the value of the label, i.e., the position j in the program string S.

(2) Find the value of BLKNUM such that BA(BLKNUM) $\leq$ T1. This operation will reset BLKNUM to point to the desired new block after skipping unwanted blocks.

(3) Reset the register T to point to the top of the new block.

(4) If blocks were skipped (ICTBK$\neq$0), then erase all operators from operator stack until ICTBK begin's have been recognized and also erased.

(5) If transfer is from a label or conditional statement, erase the ":", then, or else.

(6) Reset the program counter D to new value in program string and return to initial point.

3.2.11 Primary

Primaries appear in two productions, both part of the nonterminal factor. If N(I)≠"↑", (Fig. 21), then the primary is simply reduced to factor. Otherwise, the arithmetic unit is set up (LINK←1) by changing the top two elements of the operand stack from link to value ans storing these values in the accumulator, A and storage register, X. The exponentiation operation is performed (EXP←1) and the accumulator result placed back on the operand stack.

3.2.12 Factor

Factors are involved in three productions: (1) Simple reduction from factor to term, (2) Phrase that is reduced to term, and (3) Phrase that will be reduced again to factor. Of the three productions, the last one must be considered first because of the operator precedence (Table 3). The sequence chart (Fig. 21) first checks for S[j]=↑. If true, then the symbol is stored. Otherwise a check is made to see if N[i] ={X|/}. If true, the multiplication or division operation is performed and the result is reduced to term. If none of these operators appear, then the factor is reduced to term.

3.2.13 Term

Terms appear in four productions: (1) A phrase that will be reduced to term again, (2) A phrase combined with an addition operator that is reduced to arithmetic expression, (3) a phrase combined with arithmetic expression and an addition operator that is reduced to arithmetic expression, and (4) a simple reduction to arithmetic expression. From the operator precedence (Table 3), the sequence chart (Fig. 21) first check for S[j]={+|-}. This is the first case.

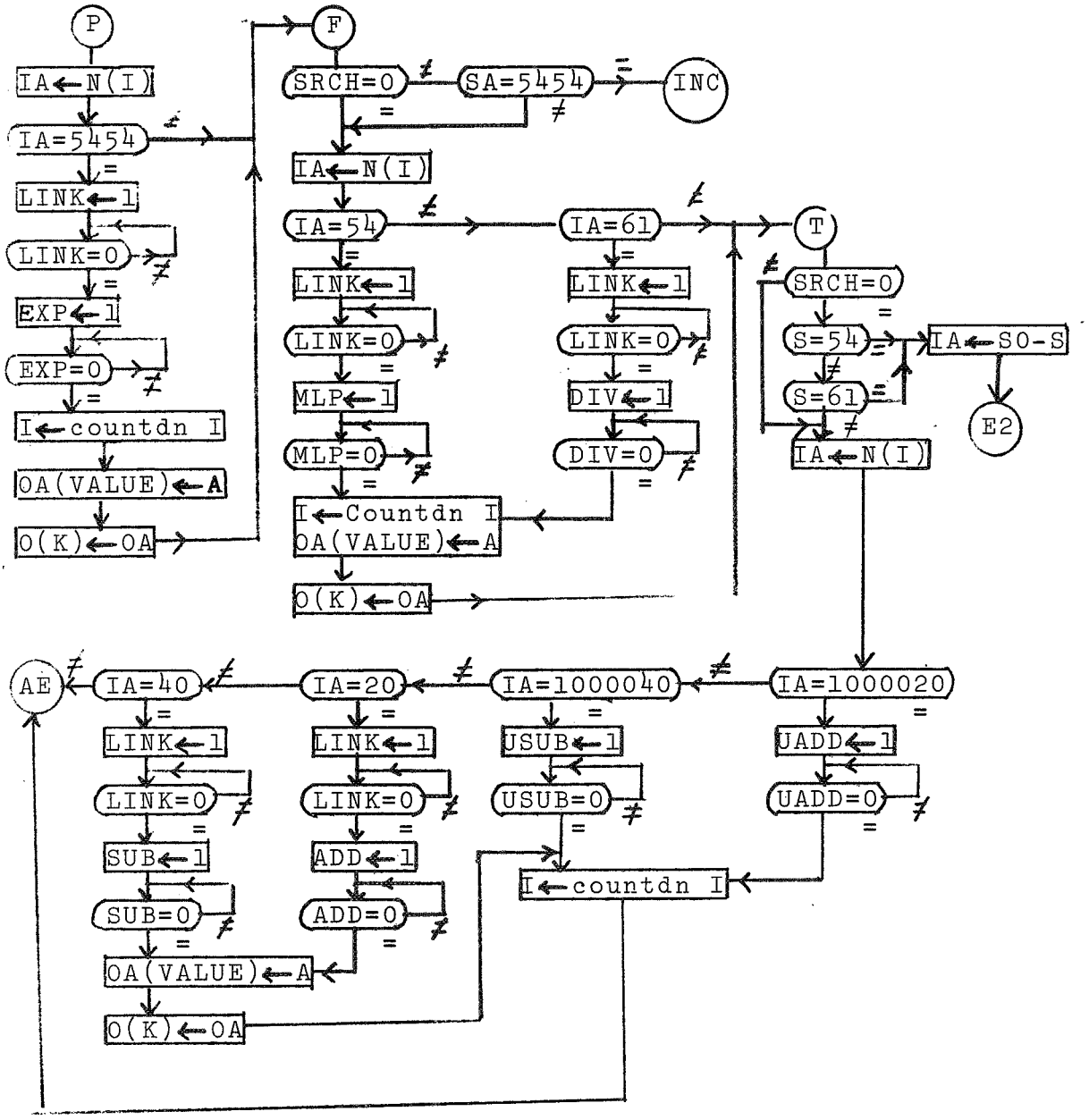

Fig. 21 Sequence Chart for Primary, Factor, and Term

Otherwise the operator stack is checked for N[i] ={+|-|$+_u$|$-_u$}. If one of these cases is true, execution is performed and the phrase reduced to arithmetic expression. Otherwise term is simply reduced to arithmetic expression.

3.2.14 Arithmetic Expression

The arithmetic expression is involved in four productions: (1) Phrase that will be reduced to arithmetic expression again, (2) phrase reduced to primary, (3) phrase reduced to boolean expression, and (4) phrase reduced to assignment statement. Based on the operator precedent (Table 3), the sequence chart (Figure 22) shows that the first check should be for S[j]={+|-}. If yes, S[j] is stored on the operator stack. Otherwise a check is made for S[j] ={'EQ'|'NE'}. Again if yes, the symbol is stored on the operator stack. Otherwise a check is made for the symbol on the operator stack. If N[i] ="(", then if S[j]=")", a reduction is made to primary (otherwise there is an error). If N[i] =":=", then a reduction can be made to an assignment statement.

If N[i] ={'EQ'|'NE'}, a reduction can be made to boolean expression. However some execution must be performed. The top two operands are changed from link to value (LINK←1), and then the result that has been stored in X and A are compared. If they are equal, then the register EQ has the value one, otherwise EQ is set to zero. If N[i] ='NE', the EQ is complemented so that the register represents the truth of the boolean expression.

3.2.15 Boolean Expression

Now that the boolean expression has been evaluated, it must be applied to the conditional statement. The first check is to make sure that N[i]=if (Fig. 23). This operator is erased from the stack and S[j] is tested for then. If no, then there is an error. The computer is now ready to scan the conditional statement based on the boolean expression value. If the register EQ has the value one, the expression is true and the syntax checking continues

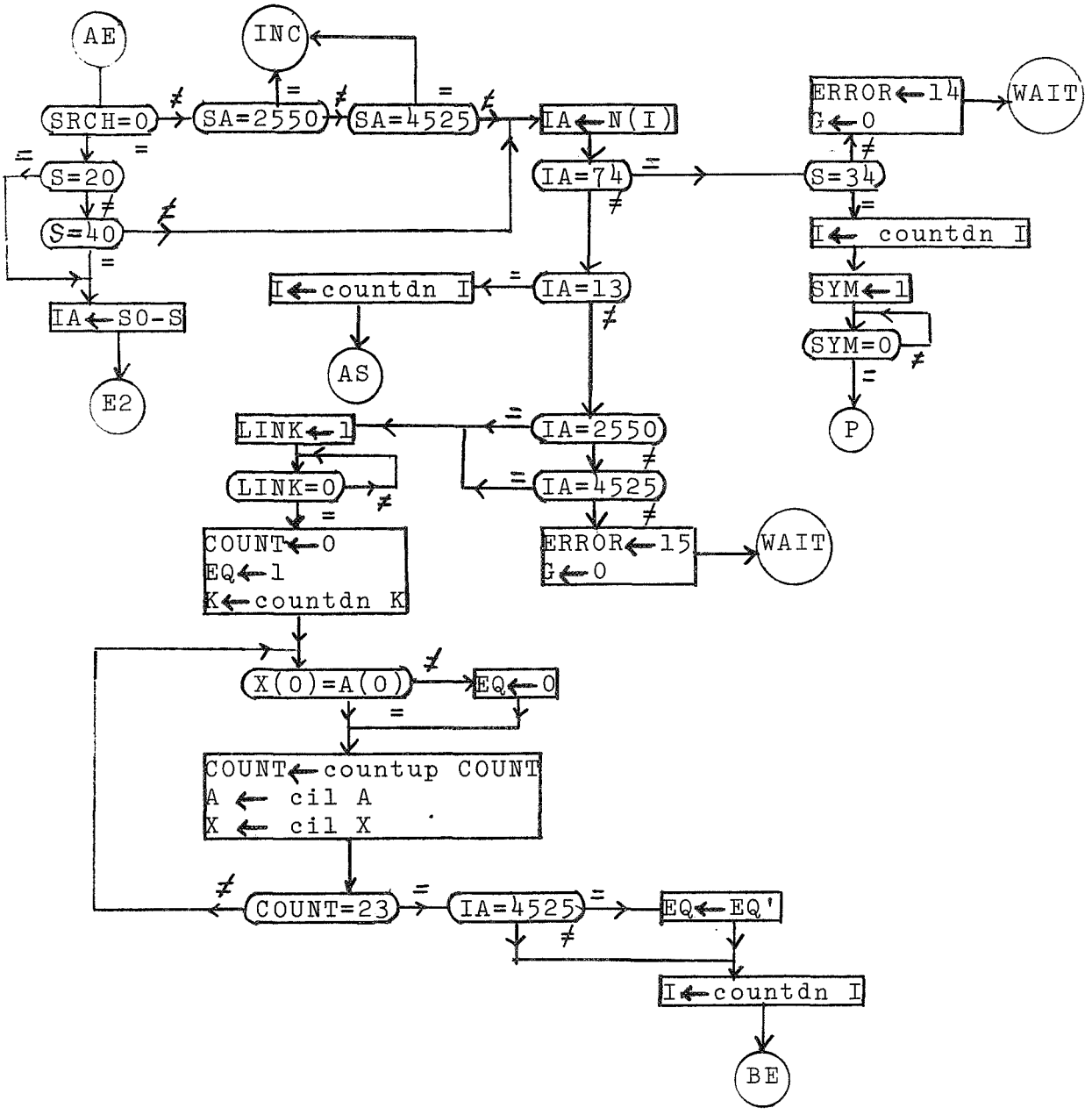

Fig. 22 Sequence Chart for Arithmetic Expression

after storing then on the operator stack. If EQ has the value zero, then the expression was false and the computer will skip symbols to avoid evaluating the then clause until S[j]={else/end/;}. The latter two symbols mean that no else clause existed. Otherwise if the symbol is else, the computer begins syntax checking that clause.

3.2.16 Assignment Statement

At this point, the arithmetic expression has just been evaluated and its value stored in the top row of the operand stack. The variable whose value is to be set has its link stored on the next to the top row of the operand stack. The sequence chart (Fig. 23) now changes the top row of the operand stack from link to value (only occurs if the arithmetic expression was simply a variable) if necessary. The value of the arithmetic expression is stored in the accumulator, A. Then this value is placed in the DSAT. The assignment statement is then reduced to basic statement.

3.2.17 Basic Statement and Conditional Statement

After these statements have been recognized, the sequence chart (Fig. 24) is only concerned with checking for labels. If a label exists, its link is removed from the operand stack and the colon is removed from the operator stack. The statement is then reduced to either unconditional statement or statement.

3.2.18 Unconditional Statement

The unconditional statement can either be reduce directly to a statement or it could be part of the phrase that will be reduced to conditional statement. Hence the first check in the sequence chart (Fig. 24) is for symbols then or else. If S[j]=else, this means that the unconditional statement has just been syntax checked for the then clause and that the else clause should be skipped. Hence symbols are fetched from memory (SYM←-1) until either end or ";" is found and then the phrase is reduced to conditional statement. Otherwise if N[i]={then|else}, the phrase can be reduced immediately to conditional statement. If the adjacent symbol is not {then|else}, then the unconditional statement is reduced to statement.

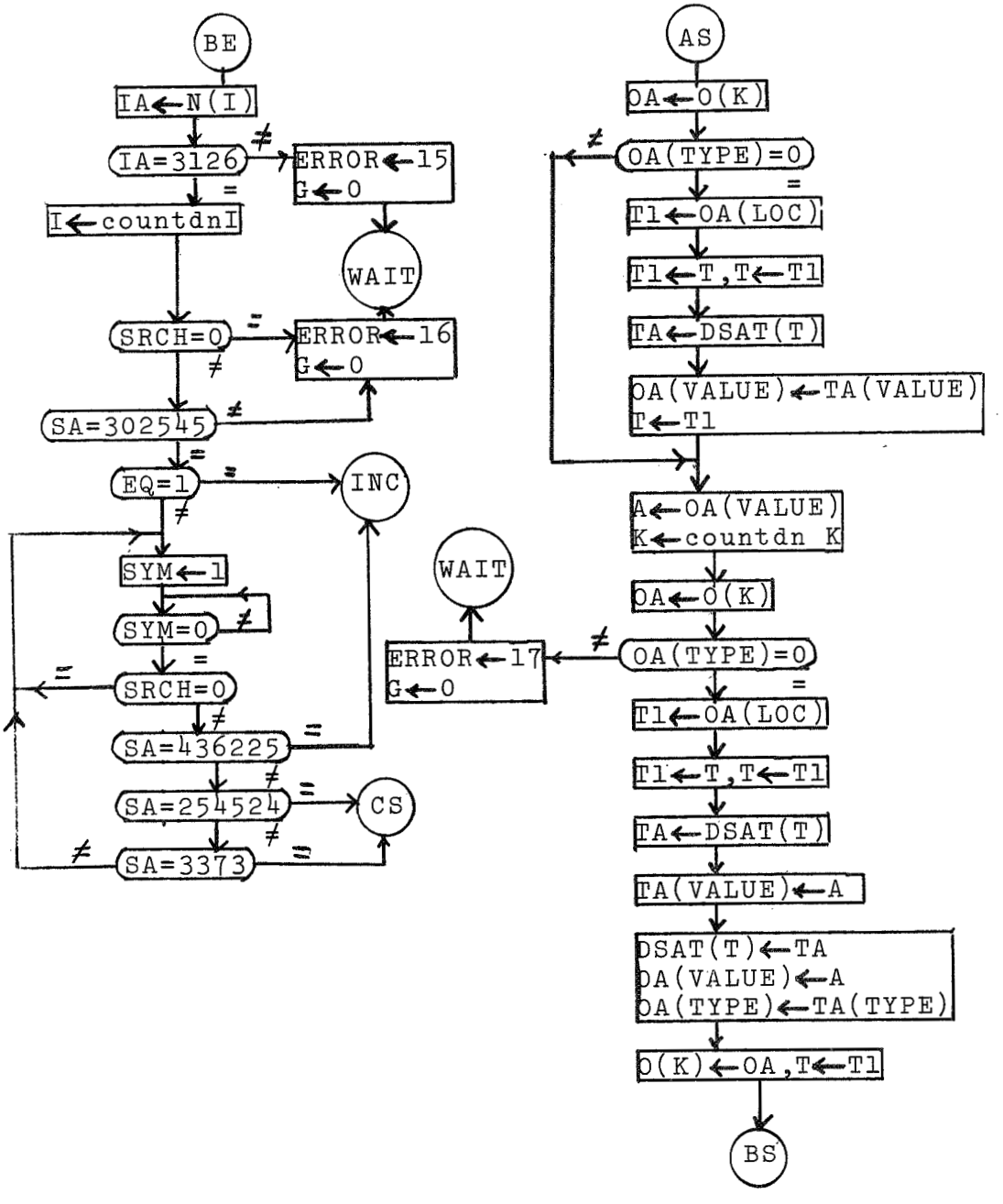

Fig. 23 Sequence Chart for Boolean Expression and Assignment Statement

3.2.19 Statement

The statement is involved in two productions both resulting in reductions to compound tail. If N[i]=";", then a phrase reduction takes place. Otherwise the statement is reduced directly to compound tail. At this point the operand stack is emptied since no references to identifiers are needed.

3.2.20 Compound Tail

The compound tail is part of three productions: (1) Phrase to be reduced to compound tail, (2) Phrase to be reduced to compound statement, and (3) Phrase to be reduced to block. Based on the operator precedence (Table 3), the sequence chart (Fig. 24) first looks for S[j]";". If yes, the symbol is stored on the operator stack and control goes back to initial point. This is the first production mentioned above. Otherwise S[j] must equal end or there is an error. Now there are two cases: (1) N[i]=";" or (2) N[i]=begin. If the latter, then the phrase, compound tail with the **begin** and end, is reduced to compound statement. If the former, the syntax checker looks at the next operator on the stack. If that is begin, the entire phrase is reduced to block.

3.2.21 Compound Statement and Block

Because of their symmetry, both block and compound statement can be represented by the same sequence chart (Fig. 25).

The first execution is that of erasing the block declarations from the DSAT if it is a block. This is operation 5 described in Section 2.2.3. Then if there was a label, it is erased. The next symbol is checked to see if the reduction will be to unconditional statement or program. If S[j]=$, then it is the latter.

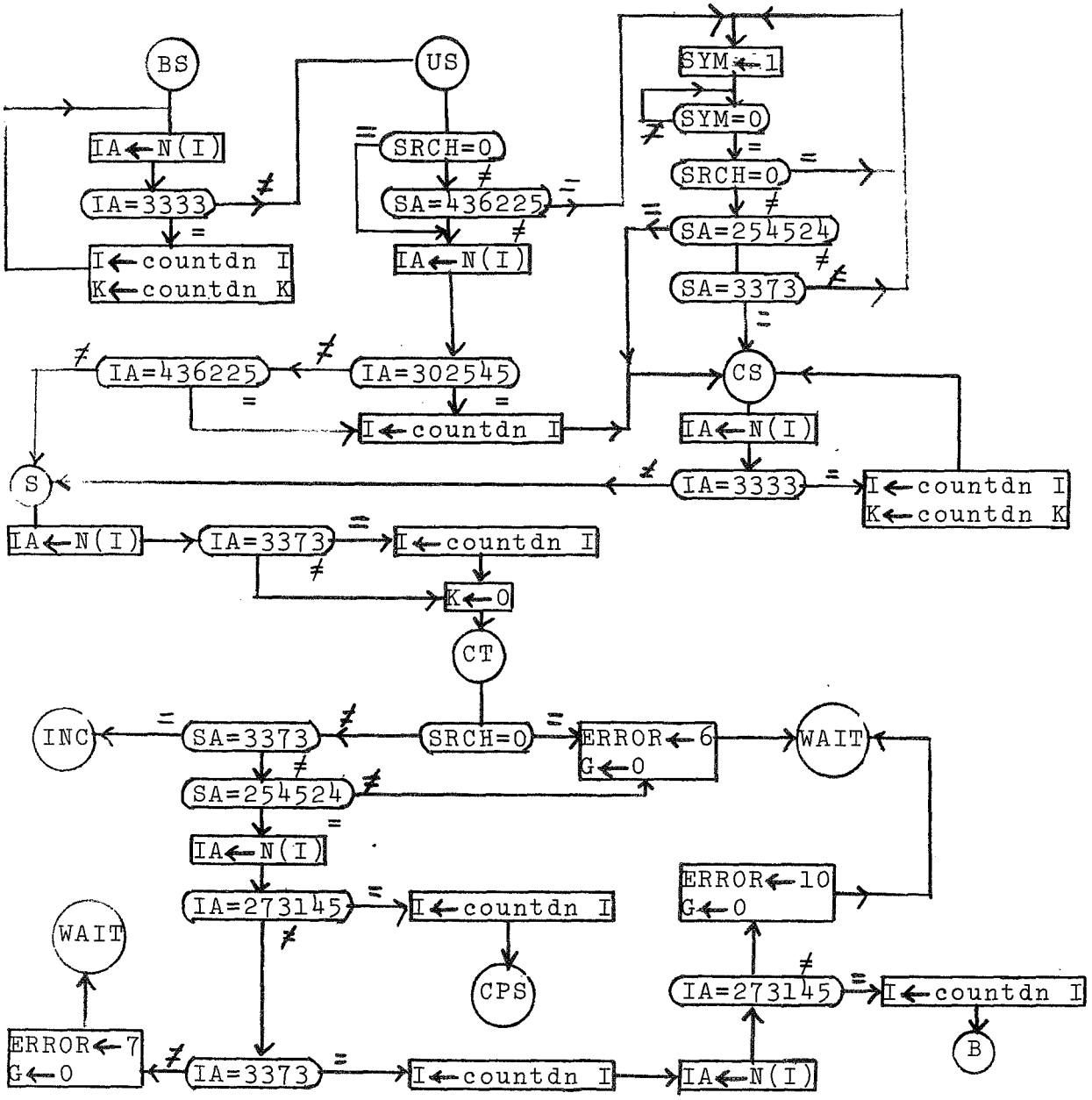

Fig. 24 Sequence Chart for Basic Statement, Unconditional Statement, Conditional Statement, Compound Tail, and Statement

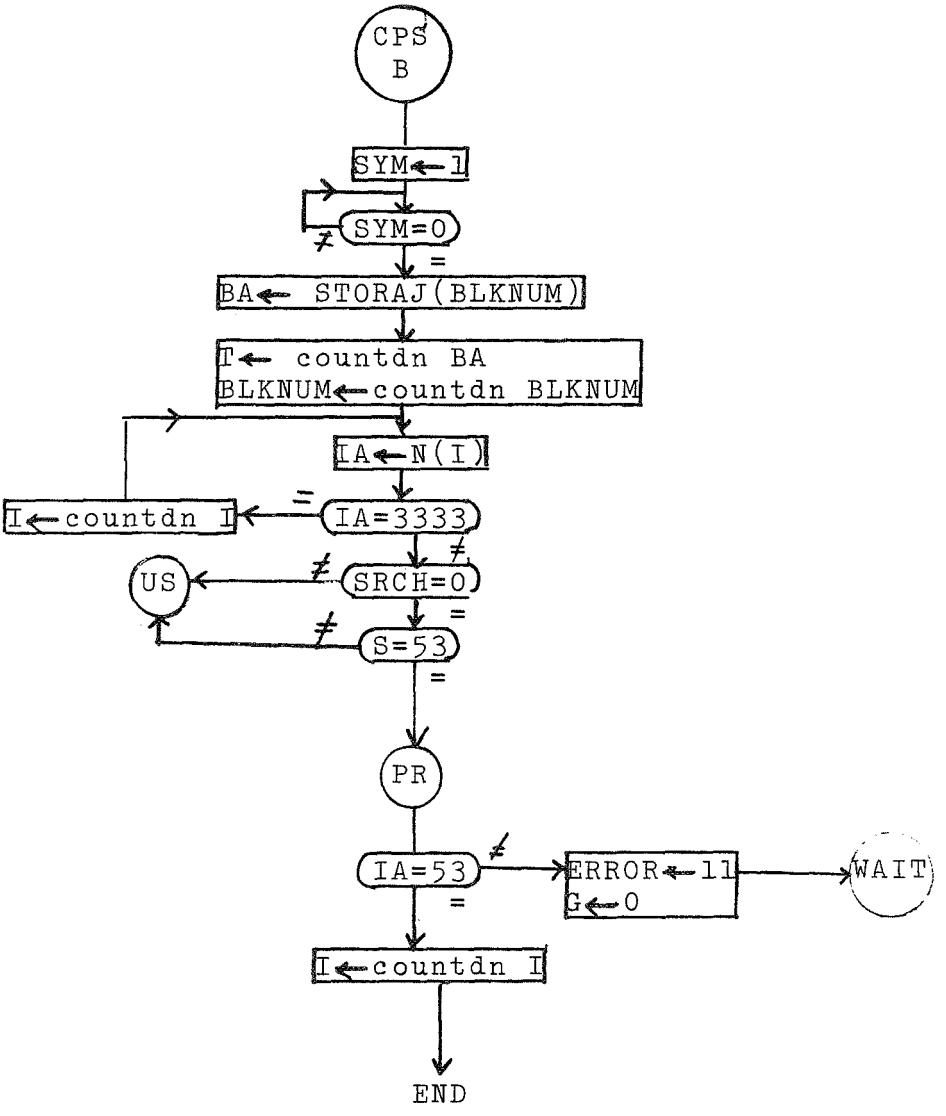

Fig. 25 Sequence Chart for Block, Compound Statement and Program

3.2.22 Program

A check is made to see if N[i]=$. If it is true, then the syntax checking has shown that the program was written correctly.

3.3 Statement Description

From the previously described configuration and sequence charts, the Algol computer is now described by the following CDL statements.

```
Comment,  Description of Algol Computer
Register, J(0-9),                $address register for MEM
          D(0-9),                $next address for MEM
          S(0-5),                $buffer register for MEM
          IN(0-5),               $address register for INP
          INA(0-18),             $buffer register for INP
          OU(0-5),               $address register for OUT
          OUA(0-18),             $buffer register for OUT
          K(0-5),                $address register for 0
          K1(0-5),               $temporary storage for K
          OA(0-5),               $buffer register for 0
          I(0-5),                $address register for N
          I1(0-5),               $temporary storage for I
          IA(0-18),              $buffer register for N
          T(0-7),                $address register for DSAT
          T1(0-7),               $temporary storage for T
          TA(0-54),              $buffer register for DSAT
          BLKNUM(0-3),           $address register for STORAJ
          ITEMP(0-3),            $temporary storage for BLKNUM
          BA(0-7),               $buffer register for STORAJ
          TEMP(0-9),             $temporary storage for J
          S0(0-12),S1(0-12),     $registers used with J
          STMP(0-17),            $register used with S
          ICTBK(0-3),            $counter for BLKNUM
          T0(0-8),               $register used with J and T
```

```
SA(0-17),          $register used with S
A(0-18),           $accumulator
Q(0-18),           $multiplier-quotient register
X(0-18),           $storage register
XP(0-18),          $counter in exponentiation
AP(0-18),          $temporary storage for A
COUNT(0-4),        $general counter
ERROR(0-4),        $error register
GR(0-1),           $greater indicator
EQ,                $equal indicator
SI,                $sign indicator
LZ,                $logical zero indicator
OV,                $add subsequence overflow indicator
AV,                $add overflow
DV,                $divide overflow
MLP,               $multiplication
FM(0-2),           $multiplication control counter
ADD,               $addition
FA(0-2),           $addition control counter
SUB,               $subtraction
FS(0-1),           $subtraction control counter
DIV,               $division
FDI(0-3),          $division control counter
ADDS,              $addition subsequence
FAS(0-1),          $addition subsequence control counter
USUB,              $unary subtraction
FUS(0-2),          $unary subtract control counter
```

```
            EXP,                $exponentiation
            FE(0-3),            $exponentiation control counter
            SYM,                $program symbol
            FSY(0-3),           $program symbol control counter
            SRCH,               $special character indicator
            LINK,               $link to value
            FL(0-3),            $link to value control counter
            G,                  $program control
            F(0-7),             $program control counter
            UADD,               $unary add
            FUA(0-2)            $unary add control counter
Subregister,IA(OP)=IA(13-18),
            OA(TYPE)=OA(0-17),
            OA(VALUE)=OA(18-36),
            OA(LOC)=OA(29-36),
            TA(NAME)=TA(0-17),
            TA(TYPE)=TA(18-35),
            TA(VALUE)=TA(36-54),
            A(M)=A(1-18),
            X(M)=X(1-18),
            Q(M)=Q(1-18),
            AQ(M)=AQ(1-36)
Casregister,AQ(0-36)=A-Q(M)
Memory,     MEM(J)=MEM(0-1023,0-5),
                                $memory containing string of program characters
```

```
          DSAT(T)=DSAT(0-255,0-54),          $dynamic storage allocation table
          O(K)=O(0-63,0-36),                 $operand stack
          N(I)=N(0-63,0-18),                 $operator stack
          INP(IN)=INP(0-63,0-18),            $memory input
          OUT(OU)=OUT(0-63,0-18),            $memory output
          STORAJ(BLKNUM)=STORAJ(0-7,0-7)     $storage table for block numbers
Decoder,  UAC(0-5)=FUA,
          USC(0-6)=FUS,
          MC(0-7)=FM,
          AC(0-6)=FA,
          SC(0-2)=FS,
          DIC(0-13)=FDI,
          ASC(0-3)=FAS,
          EC(0-8)=FE,
          SYC(0-12)=FSY,
          LC(0-14)=FL,
          DC(0-193)=F,
Clock     P
Terminal, LETTER=S(0)'*S(1)*S(2)'*(S(3)+S(4)+S(5))+S(0)*
          S(2)'*(S(3)+S(4))+(S(0)+S(1))*S(2)*S(3)'
          *S(4)'+S(0)*S(1)'*S(3)'*S(4)'*S(5),
          NUMBER=S(0)'*S(1)'*(S(2)'+S(3)'*S(4)'),
          C(18)=0,
          C(0-17)=A(1-18)*X(1-18)+A(1-18)*C(1-18)+X(1-18)*C(1-18)
Switch,   START(ON),STOP(ON),POWER(ON)
```

```
/POWER(ON)/     ADD←0,SUB←0,MLP←0,DIV←0,UADD←0,G←0,USUB←0,LINK←0,
                SYM←0,EXP←0,ADDS←0,OU←0,IN←0,T←0,K←0,I←0,BLKNUM←0,
                D←0,SI←0,AV←0,DV←0,LZ←0,OV←0,TEMP←0,TO←0,ERROR←0,
                SO←0,S1←10000,FM←0,FA←0,FS←0,FDI←0,FAS←0,FE←0,FSY←0,
                FL←0,F←0,FUA←0,FUS←0
/START(ON)/     G←1
/STOP(ON)/      G←0
/DC(0)*P/       IF(G≠0)THEN(F←1)ELSE(D←0)
Comment, Initial Point--begin memory fetch for new character
/DC(1)*P/       SYM←1,F←2
/DC(2)*P/       IF(SYM=0)THEN(F←3)
/DC(3)*P/       IF(SRCH=0)THEN(IF(NUMBER=1)THEN(F←30)ELSE(IF(LETTER=1)THEN(F←40)
                                                    ELSE(F←4)))ELSE(F←11)
/DC(4)*P/       IF(S=53)THEN(F=5)ELSE(IF(S=74)THEN(F←5)ELSE(F←6))
/DC(5)*P/       IA←SO-S,F←13
/DC(6)*P/       IF(S=20)THEN(F←7)ELSE(IF(S=40)THEN(F←7)ELSE(F←10))
/DC(7)*P/       IA←S1-S,F←13
/DC(8)*P/       IF(S=60)THEN(F←1)ELSE(G←0,ERROR←1,F←0)
/DC(9)*P/       IF(SA=273145)THEN(F←15)ELSE(F←12)
Comment, Increment Box--store symbol in operator stack
/DC(10)*P/      IA←0-SA,F←13
/DC(11)*P/      I←countup I,F←14
/DC(12)*P/      N(I)←IA,F←1
Comment, Scan new block for labels-E1
Comment, Store labels in DSAT
/DC(13)*P/      BLKNUM←countupBLKNUM,BA←countupT,TEMP(0-9)←J,
                F←16
```

```
/DC(14)*P/      STORAJ(BLKNUM)←BA,STMP←0,COUNT←1,F←17

/DC(15)*P/      SYM←1,F←20

/DC(16)*P/      IF(SYM=0)THEN(F←21)

/DC(17)*P/      IF(SRCH=0)THEN(STMP←STMP(6-17)-S,F←17)ELSE(F←22)

/DC(18)*P/      IF(SA=3333)THEN(IF(COUNT≠1)THEN(STMP←0,F←17)ELSE(F←23))
                                                ELSE(STMP←0,F←25)

/DC(19)*P/      T←countup T,TA(NAME)←STMP,F←24,TA(TYPE)←222543,
                TA(VALUE)←TO-countup J

/DC(20)*P/      DSAT(T)←TA,STMP←0,F←17

/DC(21)*P/      IF(SA≠254524)THEN(F←17,IF(SA=273145)THEN(COUNT←countup COUNT))
                                ELSE(COUNT←countdn COUNT,F←26)

/DC(22)*P/      IF(COUNT≠0)THEN(F←17)ELSE(SA-273145,J←TEMP(0-9),F←27)

/DC(23)*P/      D←countup J,TEMP←0,F←12

Comment, Unsigned Number - compute  number from string of BCD symbols

/DC(24)*P/      OA←0,K←countup K,F←31

/DC(25)*P/      OA(VALUE)←SO-S,OA(TYPE)←272551,F←32

/DC(26)*P/      SYM←1,F←33

/DC(27)*P/      IF(SYM=0)THEN(F←34)

/DC(28)*P/      IF(NUMBER≠1)THEN(0(K)←OA,F←136)ELSE(A←OA(VALUE),
                X←12,MLP←1,F←35)

/DC(29)*P/      IF(MLP≠0)THEN(F←36)

/DC(30)*P/      ADD←1,X←SO-S,F←37

/DC(31)*P/      IF(ADD=0)THEN(OA(VALUE)←A,F←32)

Comment, Identifier - concatenate string of symbols

/DC(32)*P/      OA←0,K←countup K,F←41

/DC(33)*P/      OA(VALUE)←SO-S,F←42

/DC(34)*P/      SYM←1,F←43
```

```
/DC(35)*P/      IF(SYM=0)THEN(F←44)
/DC(36)*P/      IF(NUMBER+LETTER=1)THEN(OA(VALUE)←OA(24-36)-S,F←42)
                                        ELSE(T1←T,COUNT←0,F←45)
/DC(37)*P/      TA←DSAT(T),EQ←1,F←46
/DC(38)*P/      IF(TA(0)≠OA(19))THEN(EQ←0),F←47
/DC(39)*P/      COUNT←countup COUNT,TA(NAME)←cil TA(NAME),OA(VALUE)←OA(18)-
                OA(20-36)-OA(19),F←50
/DC(40)*P/      IF(COUNT≠22)THEN(F←46)ELSE(IF(T=0)THEN(0(K)←OA,T←T1,F←53)
                                                    ELSE(F←51))
/DC(41)*P/      IF(EQ≠1)THEN(COUNT←0,T←countdn T,F←45)ELSE(TA←DSAT(T),
                OA(TYPE)←0,OA(VALUE)←0-TEMP-T,T←T1,F←52)
/DC(42)*P/      0(K)←OA,IF(TA(TYPE)=222543)THEN(F←107)ELSE(F←53)
Comment, Variable - check for correct reduction
/DC(43)*P/      IF(S=73)THEN(IA←SO-S,F←13)ELSE(IF(S=13)THEN(IA←SO-S,F←13)
                                            ELSE(IA←N(I),F←54))
/DC(44)*P/      IF(IA=272551)THEN(F←76)ELSE(IF(IA=73)THEN(F←76)
                ELSE(IF(IA≠74)THEN(F←136)ELSE(F←55)))
/DC(45)*P/      I1←I,I←countdn I,F←56
/DC(46)*P/      IA←N(I),I←I1,F←57
/DC(47)*P/      IF(IA=252124)THEN(F←60)ELSE(IF(IA≠316325)THEN(F←136)
                                        ELSE(F←67))
Comment, Read Statement - read variable value from input channel
/DC(48)*P/      IF(S≠34)THEN(ERROR←2,G←0,F←0)ELSE(I←countdn I,SYM←1,F←61)
/DC(49)*P/      IF(SYM=0)THEN(F←62)
/DC(50)*P/      I←countdn I,INA←INP(IN),IN←countup IN,T1←T,OA←0(K),F←63
/DC(51)*P/      0-TEMP-T←OA(VALUE),F←64
/DC(52)*P/      TA←DSAT(T),F←65
```

```
/DC(53)*P/      TA(VALUE)←INA,F←66
/DC(54)*P/      DSAT(T)←TA,T←T1,K←countdn K,F←247
Comment, Write Statement - write variable value on output channel
/DC(55)*P/      IF(S≠34)THEN(ERROR←3,G←0,F←0)ELSE(I←countdn I,SYM←1,F←70)
/DC(56)*P/      IF(SYM=0)THEN(F←71)
/DC(57)*P/      I←countdn I,T1←T,OA←0(K),F←72
/DC(58)*P/      O-TEMP-T←OA(VALUE),F←73
/DC(59)*P/      TA←DSAT(T),T←T1,K←countdn K,F←74
/DC(60)*P/      OUA←TA(VALUE),F←75
/DC(61)*P/      OUT(OU)←OUA,OU←countup OU,F←247
Comment, Type List - store variables in DSAT
/DC(62)*P/      COUNT←0,F←77
/DC(63)*P/      IA←N(I),F←100
/DC(64)*P/      IF(IA=73)THEN(COUNT←countup COUNT,I←countdn I,F←77)ELSE(F←101)
/DC(65)*P/      IF(IA≠272551)THEN(ERROR←5,G←0,F←0)ELSE(COUNT←countup COUNT,
                TA(TYPE)←272551,F←102)
/DC(66)*P/      OA←0(K),K←countdn K,T←countup T,COUNT←countdn COUNT,F←103
/DC(67)*P/      O-TA(NAME)←OA(VALUE),TA(VALUE)←0,F←104
/DC(68)*P/      DSAT(T)←TA,IF(COUNT≠0)THEN(F←102)ELSE(F←105)
Comment, Declaration - continue block scan
/DC(69)*P/      IF(SRCH=0)THEN(F←106)ELSE(IF(SA≠3373)THEN(F←106)ELSE(F←12))
/DC(70)*P/      ERROR←4,G←0,F←0
Comment, Label - check for label statement or goto statement
/DC(71)*P/      IA←N(I),F←110
/DC(72)*P/      IF(IA=466346)THEN(I←countdn I,F←112)ELSE(IF(SRCH=0)THEN(F←111)
                ELSE(IF(SA≠3333)THEN(F←111)ELSE(F←12)))
/DC(73)*P/      ERROR←12,G←0,F←0
```

```
Comment, Goto Statement - perform bookkeeping and then transfer
/DC(74)*P/      OA←0(K),K←countdn K,ITEMP←BLKNUM,F←277
/DC(191)*P/     T1←T,T←OA(LOC),F←300
/DC(192)*P/     TA←DSAT(T),F←301
/DC(193)*P/     OA(VALUE)←TA(VALUE),T←T1,T1←T,F←113
/DC(75)*P/      BA←STORAJ(BLKNUM),COUNT←0,GR←0,F←114
/DC(76)*P/      IF(BA(0)≠T1(0))THEN(IF(GR=0)THEN(IF(BA(0)=0)THEN(GR←1)
                                ELSE(GR←2))),F←115
/DC(77)*P/      COUNT←countup COUNT,BA←cil BA,T1←cil T1,F←116
/DC(78)*P/      IF(COUNT≠10)THEN(F←114)ELSE(F←117)
/DC(79)*P/      IF(GR=2)THEN(BLKNUM←countdn BLKNUM,F←113)ELSE(ICTBK←0,
                COUNT←0,EQ←1,BLKNUM←ITEMP,T1←0,ITEMP←BLKNUM,F←120)
/DC(80)*P/      IF(ITEMP(0)=BLKNUM(0))THEN(F←124)ELSE(IF(EQ=0)THEN(F←124)
                                        ELSE(F←121))
/DC(81)*P/      EQ←0,BA←STORAJ(BLKNUM),ICTBK←countup ICTBK,BLKNUM←countdn BLKNUM,
                F←122
/DC(82)*P/      T←BA,F←123
/DC(83)*P/      T←countdn T,F←124
/DC(84)*P/      COUNT←countup COUNT,ITEMP←cil ITEMP,BLKNUM←cil BLKNUM,F←125
/DC(85)*P/      IF(COUNT≠4)THEN(F←120)ELSE(IF(EQ=0)THEN(EQ←1,F←120)
                                        ELSE(F←126))
/DC(86)*P/      IF(ICTBK≠0)THEN(F←127)ELSE(F←131)
/DC(87)*P/      IA←N(I),F←130
/DC(88)*P/      IF(IA≠273145)THEN(I←countdn I,F--127)ELSE(I←countdn I,
                ICTBK←countdn ICTBK,F←126)
/DC(89)*P/      IA←N(I),F←132
/DC(90)*P/      IF(IA=302545)THEN(F←134)ELSE(IF(IA=436225)THEN(F←134)ELSE(F←133))
/DC(91)*P/      IF(IA=3333)THEN(F←134)ELSE(D←OA(27-36),F←1)
```

```
/DC(92)*P/      I←countdn I,F←135
/DC(93)*P/      IA←N(I),F←133
Comment, Primary - perform exponentiation if appropriate
/DC(94)*P/      IA←N(I),F←137
/DC(95)*P/      IF(IA≠5454)THEN(F←145)ELSE(LINK←1,F←140)
/DC(96)*P/      IF(LINK=0)THEN(F←141)
/DC(97)*P/      EXP←1,F←142
/DC(98)*P/      IF(EXP=0)THEN(F←143)
/DC(99)*P/      I←countdn I,OA(VALUE)←A,F←144
/DC(100)*P/     O(K)←OA,F←145
Comment, Factor - perform multiplication or division if appropriate
/DC(101)*P/     IF(SRCH≠0)THEN(IF(SA=5454)THEN(F←12)ELSE(F←146))ELSE(F←146)
/DC(102)*P/     IA←N(I),F←147
/DC(103)*P/     IF(IA=54)THEN(F←150)ELSE(IF(IA=61)THEN(F←156)ELSE(F←162))
/DC(104)*P/     LINK←1,F←151
/DC(105)*P/     IF(LINK=0)THEN(F←152)
/DC(106)*P/     MLP←1,F←153
/DC(107)*P/     IF(MLP=0)THEN(F←154)
/DC(108)*P/     I←countdn I,OA(VALUE)←A,F←155
/DC(109)*P/     O(K)←OA,F←162
/DC(110)*P/     LINK←1,F←157
/DC(111)*P/     IF(LINK=0)THEN(F←100)
/DC(112)*P/     DIV←1,F←161
/DC(113)*P/     IF(DIV=0)THEN(F←154)
Comment, Term - perform addition or subtraction if appropriate
/DC(114)*P/     IF(SRCH≠0)THEN(F←164)ELSE(IF(S=54)THEN(F←163)ELSE(IF(S=61)
                        THEN(F←163)ELSE(F←164)))
/DC(115)*P/     IA←SO-S,F←13
```

```
/DC(116)*P/     IA←N(I),F←165
/DC(117)*P/     IF(IA=1000020)THEN(F←166)ELSE(IF(IA=1000040)THEN(F←171)ELSE
                (IF(IA=20)THEN(F←173)ELSE(IF(IA=40)THEN(F--201)ELSE(F←205))))
/DC(118)*P/     UADD←1,F←167
/DC(119)*P/     IF(UADD=0)THEN(F←170)
/DC(120)*P/     I←countdn I,F←205
/DC(121)*P/     USUB←1,F←172
/DC(122)*P/     IF(USUB=0)THEN(F←170)
/DC(123)*P/     LINK←1,F←174
/DC(124)*P/     IF(LINK=0)THEN(F←175)
/DC(125)*P/     ADD←1,F←176
/DC(126)*P/     IF(ADD=0)THEN(F←177)
/DC(126)*P/     OA(VALUE)←A,F←200
/DC(128)*P/     O(K)←OA,F←170
/DC(129)*P/     LINK←1,F←202
/DC(130)*P/     IF(LINK=0)THEN(F←203)
/DC(131)*P/     SUB←1,F←204
/DC(132)*P/     IF(SUB=0)THEN(F←177)
Comment, Arithmetic Expression - compute  boolean expression if appropriate
/DC(133)*P/     IF(SRCH=0)THEN(IF(S=20)THEN(F←206)ELSE(IF(S=40)THEN(F←206)
                ELSE(F←207)))ELSE(IF(SA=2550)THEN(F←12)ELSE(IF(SA=4525)
                THEN(F←12)ELSE(F←207)))
/DC(134)*P/     IA←SO-S,F←13
/DC(135)*P/     IA←N(I),F←210
/DC(136)*P/     IF(IA=74)THEN(F←211)ELSE(F←213)
/DC(137)*P/     IF(S≠34)THEN(ERROR←14,G←0,F←0)ELSE(I←countdn I,SYM←1,F←212)
/DC(138)*P/     IF(SYM=0)THEN(F←136)
```

```
/DC(139)*P/     IF(IA=13)THEN(I←countdn I,F←232)ELSE(IF(IA=2550)THEN(F←214)
                ELSE(IF(IA=4525)THEN(F←214)ELSE(ERROR←15,G←0,F←0)))
/DC(140)*P/     LINK←1,F←215
/DC(141)*P/     IF(LINK=0)THEN(F←216)
/DC(142)*P/     COUNT←0,EQ←1,K←countdn K,F←217
/DC(143)*P/     IF(X(0)≠A(0))THEN(EQ←0),F←220
/DC(144)*P/     COUNT←countup COUNT,A←cil A,X←cil X,F←221
/DC(145)*P/     IF(COUNT≠23)THEN(F←217)ELSE(IF(IA=4525)THEN(EQ←EQ'),
                I←countdn I,F←222)
Comment, Boolean Expression - check conditional statement
/DC(146)*P/     IA←N(I),F←223
/DC(147)*P/     IF(IA≠3126)THEN(ERROR←15,G←0,F←0)ELSE(I←countdnI,F←224)
/DC(148)*P/     IF(SRCH=0)THEN(F←225)ELSE(IF(SA≠302545)THEN(F←225)ELSE(F←226))
/DC(149)*P/     ERROR←16,G←0,F←0
/DC(150)*P/     IF(EQ=1)THEN(F←12)ELSE(F←227)
/DC(151)*P/     SYM←1,F←230
/DC(152)*P/     IF(SYM=0)THEN(IF(SRCH=0)THEN(F←227)ELSE(F←231))
/DC(153)*P/     IF(SA=436225)THEN(F←12)ELSE(IF(SA=254524)THEN(F←260)ELSE(IF(SA=3373)
                                THEN(F←260)ELSE(F←227)))
Comment, Assignment Statement - store assigned value in DSAT
/DC(154)*P/     OA←O(K),F←233
/DC(155)*P/     IF(OA(TYPE)≠0)THEN(F←237)ELSE(T1←OA(LOC),F←234)
/DC(156)*P/     T1←T,T←T1,F←235
/DC(157)*P/     TA←DSAT(T),F←236
/DC(158)*P/     OA(VALUE)←TA(VALUE),T←T1,F←237
/DC(159)*P/     A←OA(VALUE),K←countdn K,F←240
/DC(160)*P/     OA←O(K),F←241
/DC(161)*P/     IF(OA(TYPE≠0)THEN(ERROR←17,G←0,F←0)ELSE(T1←OA(LOC),F←242)
```

```
/DC(162)*P/     T1←T,T←T1,F←243

/DC(163)*P/     TA←DSAT(T),F←244

/DC(164)*P/     TA(VALUE)←A,F←245

/DC(165)*P/     DSAT(T)←TA,OA(VALUE)←A,OA(TYPE)←TA(TYPE),F←246

/DC(166)*P/     O(K)←OA,T←T1,F←247

Comment, Basic Statement

/DC(167)*P/     IA←N(I),F←250

/DC(168)*P/     IF(IA≠3333)THEN(F←251)ELSE(I←countdn I,K←countdnK,F←247)

Comment, Unconditional Statement

/DC(169)*P/     IF(SRCH=0)THEN(F←252)ELSE(IF(SA=436225)THEN(F←255)ELSE(F←252))

/DC(170)*P/     IA←N(I),F←253

/DC(171)*P/     IF(IA=302545)THEN(F←254)ELSE(IF(IA=436225)THEN(F←254)ELSE(F←262)

/DC(172)*P/     I←countdn I,F←260

/DC(173)*P/     SYM←1,F←256

/DC(174)*P/     IF(SYM=0)THEN(IF(SRCH=0)THEN(F←255)ELSE(F←257)

/DC(175)*P/     IF(SA=254524)THEN(F←260)ELSE(IF(SA=3373)THEN(F←260)ELSE(F←255))

Comment, Conditional Statement

/DC(176)*P/     IA←N(I),F←261

/DC(177)*P/     IF(IA=3333)THEN(I←countdn I,K←countdn K,F←260)ELSE(F←262)

Comment, Statement

/DC(178)*P/     IA←N(I),F←263

/DC(179)*P/     IF(IA=3373)THEN(I←countdn I),K←0,F←264

Comment, Compound Tail

/DC(180)*P/     IF(SRCH=0)THEN(F←265)ELSE(IF(SA=3373)THEN(F←12)ELSE(IF(SA≠254524)
                        THEN(F←265)ELSE(F←266)))

/DC(181)*P/     ERROR←6,G←0,F←0

/DC(182)*P/     IA←N(I),F←267
```

```
/DC(183)*P/     IF(IA=273145)THEN(I←countdn I,F←272)ELSE(IF(IA≠3373)THEN
                (ERROR←7,G←0,F←0)ELSE(I←countdn I,F←270))
/DC(184)*P/     IA←N(I),F←271
/DC(185)*P/     IF(IA=273145)THEN(I←countdn I,F←272)ELSE(ERROR←10,G←0,F←0)
Comment, Block and Compound Statement
/DC(186)*P/     BA←STORAJ(BLKNUM),F←273
/DC(187)*P/     SYM←1,T←countdn BA,BLKNUM--countdn BLKNUM,F←274
/DC(188)*P/     IA←N(I),IF(SYM=0)THEN(F←275)
/DC(189)*P/     IF(IA=3333)THEN(I←countdn I,F←274)ELSE(IF(SRCH≠0)THEN(F←252)
                ELSE(IF(S=53)THEN(F←276)ELSE(F←252)))
Comment, Program
/DC(190)*P/     IF(IA≠53)THEN(ERROR←11,G←0,F←0)ELSE(I←countdn I)
Comment, Here Begins Special Routines
Comment, Fetch character from memory
/SYC(0)*P/      IF(SYM=1)THEN(J←D,SRCH←0,FSY←1)
/SYC(1)*P/      S←MEM(J),D←countup D,FSY←2
/SYC(2)*P/      IF(S=14)THEN(FSY=3)ELSE(IF(S=33)THEN(FSY←6)ELSE(IF(S=54)THEN(FSY←12)
                                        ELSE(SYM←0,FSY←0)))
/SYC(3)*P/      J←D,SA←0,SRCH←1,FSY←4
/SCY(4)*P/      S←MEM(J),D←countup D,FSY←5
/SYC(5)*P/      IF(S=14)THEN(SYM←0,FSY←0)ELSE(SA←SA(6-17)-S,J←D,FSY←4)
/SYC(6)*P/      SA←0,J←D,FSY←7
/SYC(7)*P/      S←MEM(J),D←countup D,FSY←10,SA(12-17)←S
/SYC(8)*P/      IF(S=73)THEN(FSY←11)ELSE(IF(S=33)THEN(FSY←11)ELSE
                (S←33,J←countdn J,D←countdn D,SYM←0,FSY←0))
/SYC(9)*P/      SA←SA(6-17)-S,J←D,SYM←0,FSY←0,SRCH←1
/SYC(10)*P/     SA←0,J←D,FSY←13
```

```
/SYC(11)*P/     S←MEM(J),D←countup D,FSY←14,SA(12-17)←S
/SYC(12)*P/     IF(S=54)THEN(FSY←11)ELSE(S←54,J←countdn J,D←countdn D,
                SYM←0,FSY←0)
Comment, Change operand from link to value
/LC(0)*P/       IF(LINK=1)THEN(FL←1)
/LC(1)*P/       OA←0(K),FL←2
/LC(2)*P/       IF(OA(TYPE=0)THEN(FL←3)ELSE(FL←7)
/LC(3)*P/       T1←OA(LOC),FL←4
/LC(4)*P/       T1←T,T←T1,FL←5
/LC(5)*P/       TA←DSAT(T),FL←6
/LC(6)*P/       OA(VALUE)←TA(VALUE),T←T1,FL←7
/LC(7)*P/       X←OA(VALUE),K←countdn K,FL←10
/LC(8)*P/       OA←0(K),FL←11
/LC(9)*P/       IF(OA(TYPE)=0)THEN(FL←12)ELSE(FL←16)
/LC(10)*P/      T1←OA(LOC),FL←13
/LC(11)*P/      T1←T,T←T1,FL←14
/LC(12)*P/      TA←DSAT(T),FL←15
/LC(13)*P/      OA(VALUE)←TA(VALUE),OA(TYPE)←TA(TYPE),T←T1,FL←16
/LC(14)*P/      A←OA(VALUE),LINK←0,FL←0
Comment, Add Sequence
/AC(0)*P/       IF(ADD=1)THEN(FA←1)
/AC(1)*P/       SI←0,OV←0,LZ←0,AV←0,FA←2
/AC(2)*P/       IF(A(0)≠X(0))THEN(SI←1,X(M)←X(M)'),FA←3
/AC(3)*P/       ADDS←1,FA←4
/AC(4)*P/       IF(ADDS=0)THEN(FA←5)
/AC(5)*P/       IF(SI*OV'*LZ'=1)THEN(A←A'),IF(SI'*OV=1)THEN(AV←1),IF(SI*OV'*LZ=1)
                THEN(A←0),IF(SI*OV*LZ'=1)THEN(A(M)←countup A(M)),FA←6
/AC(6)*P/       ADD←0,FA←0
```

```
Comment, Subtract Sequence
/SC(0)*P/       IF(SUB=1)THEN(FS←1)
/SC(1)*P/       X(0)←X(0)',ADD←1,FS←2
/SC(2)*P/       IF(ADD=0)THEN(SUB←0,FS←0)
Comment, Add Subsequence
/ASC(0)*P/      IF(ADDS=1)THEN(FAS←1)
/ASC(1)*P/      A(1-18)←A(1-18)⊕X(1-18)⊕C(1-18),FAS←2
/ASC(2)*P/      IF(C(0)=1)THEN(OV←1),FAS←3
/ASC(3)*P/      IF(A(M)=777777)THEN(LZ←1),ADDS←0,FAS←0
Comment, Multiplication Sequence
/MC(0)*P/       IF(MLP=1)THEN(FM←1)
/MC(1)*P/       OV←0,LZ←0,SI←0,Q(0)←0,FM←2
/MC(2)*P/       IF(A(0)≠X(0))THEN(SI←1),FM←3
/MC(3)*P/       COUNT←22,Q(M)←X(M),X←A,A←0,FM←4
/MC(4)*P/       COUNT←countdn COUNT,IF(Q(18)≠1)THEN(FM←6)ELSE(ADDS←1,FM←5)
/MC(5)*P/       IF(ADDS=0)THEN(FM←6)
/MC(6)*P/       AQ(M)←shr AQ(M),A(1)←OV,IF(COUNT≠0)THEN(FM←4)ELSE(FM←7)
/MC(7)*P/       IF(SI=1)THEN(A(0)←1,Q(0)←1),A(M)←Q(M),Q(M)←A(M),MLP←0,FM←0
Comment, Division Sequence
/DIC(0)*P/      IF(DIV=1)THEN(OV←0,LZ←0,DV←0,SI←0,Q(0)←0,FDI←1)
/DIC(1)*P/      X(M)←X(M)',IF(A(0)≠X(0))THEN(SI←1),ADDS←1,FDI←2
/DIC(2)*P/      IF(ADDS=0)THEN(FDI←3)
/DIC(3)*P/      IF(OV+LZ=1)THEN(DV←1,DIV←0,FDI←0)ELSE(A(M)←A(M)',FDI←4)
/DIC(4)*P/      IF(SI=1)THEN(Q(0)←1),ADDS←1,FDI←5
/DIC(5)*P/      IF(ADDS=0)THEN(COUNT←22,LZ←0,FDI←6)
/DIC(6)*P/      IF(OV=0)THEN(A(M)←A(M)'),FDI←7
/DIC(7)*P/      AQ(M)←shl AQ(M),OV←A(1),ADDS←1,FDI←10
```

```
/DIC(8)*P/      IF(ADDS=0)THEN(IF(LZ=1)THEN(OV←1),FDI←11)
/DIC(9)*P/      IF(OV=0)THEN(A(M)←A(M)')ELSE(Q(18)←1),FDI←12
/DIC(10)*P/     COUNT←countdn COUNT,IF(OV=0)THEN(ADDS←1,FDI←13)
                ELSE(A(M)←countup A(M),FDI←14)
/DIC(11)*P/     IF(ADDS=0)THEN(FDI←14)
/DIC(12)*P/     IF(OV=0)THEN(A(M)←A(M)'),FDI←15
/DIC(13)*P/     IF(COUNT≠0)THEN(FDI←7)ELSE(A←Q,DIV←0,FDI←0)
Comment, Exponentiation Sequence
/EC(0)*P/       IF(EXP=1)THEN(FE←1)
/EC(1)*P/       IF(X(0)=1) THEN (EQ←1) ELSE (EQ←0),FE←2
/EC(2)*P/       IF(X(M)=0)THEN(A←1,EXP←0,FE←0)ELSE(FE←3)
/EC(3)*P/       XP←X,X←A,AP←A,FE←4
/EC(4)*P/       MLP←1,FE←5
/EC(5)*P/       IF(MLP=0)THEN(XP←countdn XP,FE←6)
/EC(6)*P/       IF(XP≠0)THEN(X←AP,FE←4)ELSE(FE←7)
/EC(7)*P/       IF(EQ≠1)THEN(EXP←0,FE←0)ELSE(A←1,X←A,DIV←1,FE←10)
/EC(8)*P/       IF(DIV=0)THEN(EXP←0,FE←0)
Comment, Unary Addition Sequence
/UAC(0)*P/      IF(UADD=1)THEN(OA←O(K),FUA←1)
/UAC(1)*P/      IF(OA(TYPE)≠0)THEN(UADD←0,FUA←0)ELSE(T1←OA(LOC),FUA←2)
/UAC(2)*P/      T←T1,T1←T,FUA←3
/UAC(3)*P/      TA←DSAT(T),FUA←4
/UAC(4)*P/      OA(VALUE)←TA(VALUE),OA(TYPE)←TA(TYPE),FUA←5
/UAC(5)*P/      T←T1,O(K)←OA,UADD←0,FUA←0
Comment, Unary Subtraction Sequence
/USC(0)*P/      IF(USUB=1)THEN(OA←O(K),FUS←1)
/USC(1)*P/      IF(OA(TYPE)≠0)THEN(FUS←5)ELSE(T1←OA(LOC),FUS←2)
/UCS(2)*P/      T←T1,T1←T,FUS←3
```

```
/USC(3)*P/      TA←DSAT(T),FUS←4
/USC(4)*P/      OA(VALUE)←TA(VALUE),OA(TYPE)←TA(TYPE),T←T1,FUS←5
/USC(5)*P/      OA(18)←OA(18)',FUS←6
/USC(6)*P/      O(K)←OA,USUB←0,FUS←0
                END
```

4. Simulation

The description of the Algol computer given in the previous chapter is in terms of the general form of CDL. However, there exists an actual implementation of CDL, called CDL3, that is available at the University of Maryland [6]. This chapter describes the changes in the CDL simulation program that were necessary to use the CDL3 program. A sample program is given that illustrates how the ALGOL computer functions.

4.1 Changes in Program

In order to run the CDL simulation on the IBM 7094, it was necessary to make certain types of changes in the program:

(1) Differences between CDL statements and operators and CDL3 statements and operators must be corrected. For example, the operation "countup A" becomes "A.COUNT.". However, the biggest change is that the indices that were decimal numbers must now be expressed as octal numbers.

(2) In CDL, there is no provision for actually exhibiting external control of a program. All operations must be done during the clock cycles. However, one would like to load the program of the simulated computer into memory before the clock cycling begins. This operation is possible in CDL3. Also the switches are set by special external control cards to correspond to an operator switching on a computer console.

(3) CDL3 provides a output format for printing register and switch values. In order to print out information at every cycle, sets of CDL statements have been combined into one statement whenever possible.

(4) CDL assumes an unlimited amount of space is available for the program. However, the CDL3 program is greatly limited. Hence for this example, certain hardware subroutines (division and exponentiation) that are not needed have been omitted.

4.2 Sample Program

The following Algol program was chosen to illustrate the functioning of the computer.

```
$'BEGIN' 'INTEGER' A,B; 'READ'(A); B=A+2; 'WRITE'(B) 'END'$
```

This program string is loaded into memory in a slightly altered form because of the CDL3 format limitation requiring octal input. Hence, M(0)=53,M(1)=14,etc., corresponds to reading in the octal BCD equivalent to the alphanumeric character. Notice that the characters are read in one at a time and stored one character to a memory word. It is also necessary to set up the input channel INP since numbers are read from this channel by the simulated computer. For this example, there is only one read instruction and thus only one number is needed. The value for A was chosen to be octal 15 and thus INP(0)=15. After the simulation is over, one expects the output channel OUT to have octal 17 as its only value.

4.3 Description of program operation

The listing of the simulated computer program appears in Figures 26-31. In the beginning of the listing, there are the standard twelve control cards common to all CDL3 programs. Following these cards, is the CDL3 description of the ALGOL computer. Finally after the "$SIMULATE" is a set of control cards describing the output format, switching, and presimulation status of the computer.

To get a true feeling for the computer operation, it would be best to examine the output at every clock pulse. The simulation does have this output but it would be too costly to include it in this paper. However, one can summarize the operation in a few steps (Table 4). The table is divided into four columns. The first column lists the range of the clock time during which the series of operations occured. The second column shows the flow of control between the non-

```
$IBSYS
$*     MOUNT TAPE 1090 ON A9, ,RING OUT AND SAVE
$*     THANK YOU
$PAUSE
$ATTACH          A9
$AS              SYSLB4
$REWIND          SYSLB4
$LXECUTE         USER
$ID    BLOOM*001/11/728*5M*75P$
$CDL3
$TRANSLATE
*MAIN
COMMENT,SIMULATION OF ALGOL COMPUTER IN CDL
 REGISTER,      J(0-11),D(0-11),S(0-5),IN(0-5),INA(0-22),OU(0-5),OUA(0-22)
 REGISTER,       K(0-5),K1(0-5),DA(0-44),I(0-5),I1(0-5),IA(0-22),T(0-7)
 REGISTER,      T1(0-7),TA(0-66),BLKNUM(0-3),ITEMP(0-3),BA(0-7),TEMP(0-11)
 REGISTER,       S0(0-14),S1(0-14),STMP(0-21),ICTBK(0-3),TO(0-10)
 REGISTER,      A(0-22),Q(0-22),X(0-22),AR(0-22),AP(0-22),XP(0-22)
 REGISTER,      CNT(0-4),ERROR(0-4),GR(0-1),EQ,SI,LZ,OV,AV,DV,MLP
 REGISTER,      FM(0-2),ADD,FA(0-2),SUB,FS(0-1),DIV,FDI(0-3),ADDS
 REGISTER,      FAS(0-1),USUB,FUS(0-2),EXP,FE(0-3),SYM,FSY(0-3),SRCH
 REGISTER,      LINK,FL(0-3),G,F(0-7),UADD,FUA(0-2),SA(0-21)
 SUBREGISTER,  IA(OP)=IA(15-22),DA(TYPEO)=DA(0-21),DA(VALUED)=DA(22-44)
 SUBREGISTER,  TA(NAME)=TA(0-21),TA(TYPE)=TA(22-43),TA(VALUE)=TA(44-66),
1              A(M)=A(1-22),X(MX)=X(1-22),Q(MQ)=Q(1-22)
 MEMORY,        MEM(J)=MEM(0-1777,0-5),
1               DSAT(T)=DSAT(0-377,0-66),
1               C(K)=C(0-77,0-44),
1               N(I)=N(0-77,0-22)
 MEMORY,        INP(IN)=INP(0-77,0-22),
1               OUT(OU)=OUT(0-77,0-22),
1               STORAJ(BLKNUM)=STORAJ(0-7,0-7)
 DECODER,       UAC(0-5)=FUA,USC(0-6)=FUS,MC(0-7)=FM,AC(0-6)=FA
 DECODER,       SC(0-2)=FS,DIC(0-15)=FDI,ASC(0-3)=FAS,EC(0-8)=FE,
1               SYC(0-14)=FSY,LC(0-16)=FL,DC(0-302)=F
 CLOCK,         P
 SWITCH,        START(ON),STOP(ON),POWER(ON)
 TERMINAL,      LETTER=S(0)'*S(1)*S(2)'*(S(3)+S(4)+S(5))+S(0)*S(2)'*(S(3)+
1         S(4))+(S(0)+S(1))*S(2)*S(3)'*S(4)'+S(0)*S(1)'*S(3)'*S(4)'*S(5),
1               NUMBER=S(0)'*S(1)'*(S(2)'+S(3)'*S(4)')
 /POWER(ON)/    ADD=0,SUB=0,MLP=0,DIV=0,UADD=0,G=0,USUB=0,LINK=0,SYM=0
 /POWER(ON)/    EXP=0,ADDS=0,OU=0,IN=0,T=0,K=0,I=0,BLKNUM=0,D=0
 /POWER(ON)/    TEMP=0 ,SI=0,AV=0,DV=0,LZ=0,OV=0,S1=10000
 /POWER(ON)/    TO=0,ERROR=0,S0=0,          FM=0,FA=0,FS=0,FDI=0,FAS=0
 /POWER(ON)/    FE=0,FSY=0,FL=0,F=0,FUS=0,FUA=0,TA=0,DA=0,IA=0,J=0
 /START(ON)/    G=1
 /STOP(ON)/     G=0
 /DC(0)*P/      IF(G.NE.0) THEN(F=1)ELSE(J=0)
COMMENT,INITIAL POINT
 /DC(1)*P/      SYM=1,F=2
 /DC(2)*P/      IF(SYM.EQ.0)THEN(IF(SRCH.EQ.0)THEN(IF(NUMBER.EQ.1)THEN(F=
                30)ELSE(IF(LETTER.EQ.1)THEN(F=40)ELSE(F=4)))ELSE(IF(SA.EQ.
                273145)THEN(F=15)ELSE(IA=0-SA,I=I.COUNT.,F=14)))
 /DC(4)*P/      IF((S.EQ.53)+(S.EQ.74))THEN(IA=S0-S,I=I.COUNT.,F=14)ELSE(
                IF((S.EQ.20)+(S.EQ.40))THEN(IA=S1-S,I=I.COUNT.,F=14)ELSE(
                IF(S.EQ.60) THEN(F=1)ELSE(G=0,ERROR=1,F=0)))
COMMENT,INCREMENT BCX
 /DC(13)*P/     I=I.COUNT.,F=14
 /DC(14)*P/     N(I)=IA,F=1
COMMENT,SCAN NEW BLOCK FOR LABELS-E1
```

Fig. 26 CDL3 Program listing 1

```
 /DC(15)*P/    BLKNUM=BLKNUM.COUNT.,BA=T.COUNT.,TEMP(0-11)=J,F=16
 /DC(16)*P/    STORAJ(BLKNUM)=BA,STMP=0,CNT=1,SYM=1,F=20
 /DC(20)*P/    IF(SYM.EQ.0) THEN(
               IF(SRCH.EQ.0)THEN(STMP=STMP(6-21)-S,SYM=1)ELSE(
               IF(SA.EQ.3333)THEN(IF(CNT.NE.1)THEN(STMP=0,SYM=1)ELSE(F=23
              ))ELSE(STMP=0,
               IF(SA.NE.254524)THEN(F=20,SYM=1 ,IF(SA.EQ.273145)THEN(CNT=
               CNT.COUNT.))ELSE(CNT=(CNT'.COUNT.)',F=26))))
 /DC(23)*P/    T=T.COUNT.,TA(NAME)=STMP,F=24,TA(TYPE)=222543,
               TA(VALUE)=T0-J
 /DC(24)*P/    DSAT(T)=TA,STMP=0,F=20,SYM=1
 /DC(26)*P/  IF(CNT.NE.0)THEN(SYM=1,F=20)ELSE(SA=273145,J=TEMP(0-11),
               TEMP=0,F=14,TA=273145,I=I.COUNT.)
COMMENT,UNSIGNED NUMBER
 /DC(30)*P/    OA=0,K=K.COUNT.,F=31
 /DC(31)*P/    OA(VALUED)=S0-S,OA(TYPED)=272551,F=32
 /DC(32)*P/    SYM=1,F=33
 /DC(33)*P/    IF(SYM.EQ.0)THEN(IF(NUMBER.NE.1)THEN(O(K)=OA,F=136)ELSE(A=
               OA(VALUED),X=12,MLP=1,F=35))
 /DC(35)*P/    IF(MLP.NE.0) THEN( ADD=1,X=S0-S,F=37)
 /DC(37)*P/    IF(ADD.EQ.0)THEN(OA(VALUED)=A,F=32)
COMMENT,IDENTIFIER
 /DC(40)*P/    OA(TYPED)=0,K=K.COUNT.,OA(VALUED)=S0-S,F=42
 /DC(42)*P/    SYM=1,F=43
 /DC(43)*P/    IF(SYM.EQ.0)THEN(IF((NUMBER+LETTER).EQ.1)THEN(OA(VALUED)=
               OA(30-44)-S,F=42)ELSE(T1=T,CNT=0,F=45))
 /DC(45)*P/    TA=DSAT(T),EQ=1,F=46
 /DC(46)*P/    IF(TA(NAME).NE.OA(23-44))THEN(EQ=0),F=50
 /DC(50)*P/    IF(T.EQ.0)THEN(O(K)=OA,T=T1,F=53)ELSE(
               F=53)ELSE(
               IF(EQ.NE.1)THEN(CNT=0,T=(T'.COUNT.)',F=45)ELSE(TA=DSAT(T),
               OA(TYPED)=0,OA(VALUED)=0-TEMP-T,T=T1,F=52))
 /DC(52)*P/    O(K)=OA,IF(TA(TYPE).EQ.222543)THEN(F=107)ELSE(F=53)
COMMENT,VARIABLE
 /DC(53)*P/    IF((SRCH.EQ.0)*((S.EQ.73)+(S.EQ.13)))THEN(IA=S0-S,F=13)
               ELSE(IA=N(I),F=54)
 /DC(54)*P/    IF(IA.EQ.272551)THEN(F=76)ELSE(IF(IA.EQ.73)THEN(F=76)
               ELSE(IF(IA.NE.74)THEN(F=136)ELSE(
               I1=I,I=(I'.COUNT.)',F=56)))
 /DC(56)*P/    IA=N(I),I=I1,F=57
 /DC(57)*P/    IF(IA.EQ.252124)THEN(F=60)ELSE(IF(IA.NE.316325)THEN(F=136)
               ELSE(F=67))
COMMENT,READ STATEMENT
 /DC(60)*P/    IF(S.NE.34)THEN(ERROR=2,G=0,F=0)ELSE(I=(I'.COUNT.)',
               SYM=1,F=61)
 /DC(61)*P/    IF(SYM.EQ.0)THEN(
               I=(I'.COUNT.)',INA=INP(IN),IN=IN.COUNT.,T1=T,OA=O(K),F=63)
 /DC(63)*P/    T=OA(35-44),F=64
 /DC(64)*P/    TA=DSAT(T),F=65
 /DC(65)*P/    TA(VALUE)=INA,F=66
 /DC(66)*P/    DSAT(T)=TA,T=T1,K=(K'.COUNT.)',F=247
COMMENT,WRITE STATEMENT
 /DC(67)*P/    IF(S.NE.34)THEN(ERROR=3,G=0,F=0)ELSE(I=(I'.COUNT.)',
               SYM=1,F=70)
 /DC(70)*P/    IF(SYM.EQ.0)THEN(F=71)
 /DC(71)*P/    I=(I'.COUNT.)',T1=T,OA=O(K),F=72
 /DC(72)*P/    T=OA(35-44),F=73
 /DC(73)*P/    TA=DSAT(T),T=T1,K=(K'.COUNT.)',F=74
 /DC(74)*P/    OUA=TA(VALUE),F=75
 /DC(75)*P/    OUT(OU)=OUA,OU=OU.COUNT.,F=247
```

Fig. 27 CDL3 Program Listing 2

```
COMMENT,TYPE LIST
 /DC(76)*P/    CNT=0,F=77
 /DC(77)*P/    IA=N(I),F=100
 /DC(100)*P/   IF (IA.EQ.73)THEN(CNT=CNT.COUNT.,I=(I'.COUNT.)',F=77)ELSE(
               IF(IA.NE.272551)THEN(ERROR=5,G=0,F=0)ELSE(CNT=CNT.COUNT.,
               TA(TYPE)=272551,F=102,I=(I'.COUNT.)'))
 /DC(102)*P/   OA=O(K),K=(K'.COUNT.)',T=T.COUNT.,CNT=(CNT'.COUNT.)',F=103
 /DC(103)*P/   TA(NAME)=OA(23-44),TA(VALUE)=0,F=104
 /DC(104)*P/   DSAT(T)=TA,IF(CNT.NE.0)THEN(F=102)ELSE(F=105)
COMMENT,DECLARATION
 /DC(105)*P/   IF((SRCH)*(SA.EQ.3373))THEN(F=13,IA=O-SA)ELSE(ERROR=4,
               G=0,F=0)
COMMENT,LABEL
 /DC(107)*P/   IA=N(I),F=110
 /DC(110)*P/   IF(IA.EQ.466346)THEN(I=(I'.COUNT.)',F=112)ELSE(IF(SRCH.EQ.
               0)THEN(ERROR=12,G=0,F=0)ELSE(IF(SA.NE.3333)THEN(ERROR=12,
               G=0,F=0)ELSE(F=13,IA=O-SA)))
COMMENT,GOTO STATEMENT
 /DC(112)*P/   OA=O(K),K=(K'.COUNT.)',ITEMP=BLKNUM,F=277
 /DC(277)*P/   T1=T,T=OA(35-44),F=300
 /DC(300)*P/   TA=DSAT(T),F=301
 /DC(301)*P/   OA(VALUEO)=TA(VALUE),T=T1,T1=T,F=113
 /DC(113)*P/   BA=STORAJ(BLKNUM),CNT=0,GR=0,F=114
 /DC(114)*P/   IF(BA(0).NE.T1(0) )THEN(IF(GR.EQ.0)THEN(IF(BA(0).EQ.0)
               THEN(GR=1)ELSE(GR=2))),
               CNT=CNT.COUNT.,BA=BA(1-7)-BA(0),T1=T1(1-7)-T1(0),F=116
 /DC(116)*P/   IF(CNT.NE.10)THEN(F=114)ELSE(
               IF(GR.EQ.2)THEN(BLKNUM=(BLKNUM'.COUNT.)',F=113)ELSE(ICTBK=
               0,CNT=0,EQ=1,BLKNUM=ITEMP,ITEMP=BLKNUM,F=120))
 /DC(120)*P/   IF(ITEMP(0).EQ.BLKNUM(0))THEN(F=124)ELSE(IF(EQ.EQ.0)THEN(F
               =124)ELSE(EQ=0,BA=STORAJ(BLKNUM),ICTBK=ICTBK.COUNT.,BLKNUM
               =(BLKNUM'.COUNT.)',F=122))
 /DC(122)*P/   T=(BA'.COUNT.)',F=124
 /DC(124)*P/   CNT=CNT.COUNT.,ITEMP=ITEMP(1-3)-ITEMP(0),
               BLKNUM=BLKNUM(1-3)-BLKNUM(0),F=125
 /DC(125)*P/   IF(CNT.NE.4)THEN(F=120)ELSE(IF(EQ.EQ.0)THEN(EQ=1,F=120)
               ELSE(F=126))
 /DC(126)*P/   IF(ICTBK.NE.0)THEN(F=127)ELSE(F=132,IA=N(I))
 /DC(127)*P/   IA=N(I),F=130
 /DC(130)*P/   IF(IA.NE.273145)THEN(I=(I'.COUNT.)',F=127)
                 ELSE(I=(I'.COUNT.)',ICTBK=(ICTBK.COUNT.)',F=126)
 /DC(132)*P/   IF((IA.EQ.302545)+(IA.EQ.436225))THEN(I=(I'.COUNT.)',
               F=135)ELSE(F=133)
 /DC(133)*P/   IF(IA.EQ.3333)THEN(I=(I'.COUNT.)',F=135)ELSE(J=OA(33-44),
               F=1)
 /DC(135)*P/   IA=N(I),F=133
COMMENT,PRIMARY
 /DC(136)*P/   IA=N(I),F=137
 /DC(137)*P/   IF(IA.NE.5454)THEN(F=145)ELSE(LINK=1,F=140)
 /DC(140)*P/   IF(LINK.EQ.0)THEN(EXP=1,F=142)
 /DC(142)*P/   IF(EXP.EQ.0)THEN(I=(I'.COUNT.)',OA(VALUEO)=A,F=144)
 /DC(144)*P/   O(K)=OA,F=145
COMMENT,FACTOR
 /DC(145)*P/   IF((SRCH.NE.0)*(SA.EQ.5454))THEN(F=13,IA=O-SA)ELSE(
               IA=N(I),F=147)
 /DC(147)*P/   IF(IA.EQ.54)THEN(LINK=1,F=151)ELSE(IF(IA.EQ.61)THEN(
               LINK=1,F=157)ELSE(F=162))
 /DC(151)*P/   IF(LINK.EQ.0)THEN(MLP=1,F=153)
 /DC(153)*P/   IF(MLP.EQ.0)THEN(F=154)
 /DC(154)*P/   T=(T'.COUNT.)',OA(VALUEO)=A,F=155
```

Fig. 28 CDL3 Program Listing 3

```
 /DC(155)*P/  O(K)=OA,F=162
 /DC(157)*P/  IF(LINK.EQ.0)THEN(DIV=1,F=161)
 /DC(161)*P/  IF(DIV.EQ.0)THEN(F=154)
COMMENT,TERM
 /DC(162)*P/  IF((SRCH.EQ.0)*((S.EQ.54)+(S.EQ.61)))THEN(IA=S0-S,F=13)
              ELSE(IA=N(I),F=165)
 /DC(165)*P/  IF(IA.EQ.1000020)THEN(UADD=1,F=167) ELSE(IF(IA.EQ.1000040)
              THEN(USUB=1,F=172)ELSE(IF(IA.EQ.20)THEN(LINK=1,F=174)
              ELSE(IF(IA.EQ.40)THEN(LINK=1,F=202)ELSE(F=205))))
 /DC(167)*P/  IF(UADD.EQ.0) THEN(I=(I'.COUNT.)',F=205)
 /DC(172)*P/  IF(USUB.EQ.0)THEN(I=(I'.COUNT.)',F=205)
 /DC(174)*P/  IF(LINK.EQ.0)THEN(ADD=1,F=176)
 /DC(176)*P/  IF(ADD.EQ.0)THEN(OA(VALUEO)=A,F=200)
 /DC(200)*P/  O(K)=OA,I=(I'.COUNT.)',F=205
 /DC(202)*P/  IF(LINK.EQ.0)THEN(SUB=1,F=204)
 /DC(204)*P/  IF(SUB.EQ.0)THEN(OA(VALUEO)=A,F=200)
COMMENT,ARITHMETIC EXPRESSION
 /DC(205)*P/  IF(SRCH.EQ.0)THEN(IF((S.EQ.20)+(S.EQ.40))THEN(IA=S0-S,
              F=13)ELSE(IA=N(I),F=210))ELSE(IF((SA.EQ.2550)+(SA.EQ.
              4525))THEN(F=13,IA=0-SA)ELSE(IA=N(I),F=210))
 /DC(210)*P/  IF(IA.EQ.74)THEN(IF(S.NE.34)THEN(ERROR=14,G=0,F=0)
              ELSE(I=(I'.COUNT.)',SYM=1,F=212))ELSE(
              IF(IA.EQ.13)THEN(I=(I'.COUNT.)',F=232)ELSE(IF( (IA.EQ.2550
              )+(IA.EQ.4525))THEN(LINK=1,F=215)ELSE(ERROR=15,G=0,F=0)))
 /DC(212)*P/  IF(SYM.EQ.0)THEN(F=136)
 /DC(215)*P/  IF(LINK.EQ.0)THEN(CNT=0,EQ=1,K=(K'.COUNT.)',F=217)
 /DC(217)*P/  IF(X(0).NE.A(0))THEN(EQ=0),
              CNT=CNT.COUNT.,A=A(1-22)-A(0),X=X(1-22)-X(0),F=221
 /DC(221)*P/  IF(CNT.NE.23)THEN(F=217)ELSE(IF(IA.EQ.4525)THEN(EQ=EQ'),
              I=(I'.COUNT.)',F=222)
COMMENT,BOOLEAN EXPRESSION
 /DC(222)*P/  IA=N(I),F=223
 /DC(223)*P/  IF(IA.NE.3126)THEN(ERROR=15,G=0,F=0)ELSE(I=(I'.COUNT.)',
              F=224)
 /DC(224)*P/  IF(SRCH.EQ.0)THEN(ERROR=16,G=0,F=0)ELSE(IF(SA.NE.302545)
              THEN(ERROR=16,G=0,F=0)ELSE(IF(EQ.EQ.1)THEN(F=13,IA=0-SA)
              ELSE(SYM=1,F=230)))
 /DC(230)*P/  IF(SYM.EQ.0)THEN(IF(SRCH.EQ.0)THEN(SYM=1)ELSE(
              IF(SA.EQ.436225)THEN(F=13,IA=0-SA)ELSE(IF(SA.EQ.254524)
              THEN(F=260)ELSE(IF(SA.EQ.3373)THEN(F=260)ELSE(SYM=1))))
COMMENT,ASSIGNMENT STATEMENT
 /DC(232)*P/  OA=O(K),F=233
 /DC(233)*P/  IF(OA(TYPEO).NE.0)THEN(F=237)ELSE(T1=OA(35-44),F=234)
 /DC(234)*P/  T1=T,T=T1,F=235
 /DC(235)*P/  TA=DSAT(T),F=236
 /DC(236)*P/  OA(VALUEO)=TA(VALUE),T=T1,F=237
 /DC(237)*P/  A=OA(VALUEO),K=(K'.COUNT.)',F=240
 /DC(240)*P/  OA=O(K),F=241
 /DC(241)*P/  IF(OA(TYPEO).NE.0)THEN(ERROR=17,G=0,F=0)ELSE(T1=OA(35-44)
              ,F=242)
 /DC(242)*P/  T1=T,T=T1,F=243
 /DC(243)*P/  TA=DSAT(T),F=244
 /DC(244)*P/  TA(VALUE)=A,F=245
 /DC(245)*P/  DSAT(T)=TA,OA(VALUEO)=A,OA(TYPEO)=TA(TYPE),F=246
 /DC(246)*P/  O(K)=OA,T=T1,F=247
COMMENT,BASIC STATEMENT
 /DC(247)*P/  IA=N(I),F=250
 /DC(250)*P/  IF(IA.NE.3333)THEN(F=251)ELSE(I=(I'.COUNT.)',
              K=(K'.COUNT.)',F=247)
COMMENT,UNCONDITIONAL STATEMENT
```

Fig. 29 CDL3 Program Listing 4

```
 /DC(251)*P/   IF(SRCH.EQ.0)THEN(IA=N(I),F=253)ELSE(IF(SA.EQ.436226)
               THEN(SYM=1,F=256)ELSE(IA=N(I),F=253))
 /DC(253)*P/   IF((IA.EQ.302545)+(IA.EQ.436226))THEN(I=(I'.COUNT.)',
               F=260)ELSE(F=262)
 /DC(256)*P/   IF(SYM.EQ.0)THEN(IF(SRCH.EQ.0)THEN(SYM=1)ELSE(
               IF(SA.EQ.254524)THEN(F=260)ELSE(IF(SA.EQ.3373)THEN(F=260)
               ELSE(SYM=1))
COMMENT,CONDITIONAL STATEMENT
 /DC(260)*P/   IA=N(I),F=261
 /DC(261)*P/   IF(IA.EQ.3333)THEN(I=(I'.COUNT.)',K=(K'.COUNT.)',F=260)
               ELSE(F=262)
COMMENT,STATEMENT
 /DC(262)*P/   IA=N(I),F=263
 /DC(263)*P/   IF(IA.EQ.3373)THEN(I=(I'.COUNT.)'),K=0,
               IF(SRCH.EQ.0)THEN(ERROR=6,G=0,F=0)ELSE(IF(SA.EQ.3373)
               THEN(F=13,IA=0-SA)ELSE(IF(SA.NE.254524)THEN(ERROR=6,F=0,G=
               0)ELSE(F=266)))
 /DC(266)*P/   IA=N(I),F=267
 /DC(267)*P/   IF(IA.EQ.273145)THEN(I=(I'.COUNT.)',F=272)ELSE(IF
               (IA.NE.3373)THEN(ERROR=7,G=0,F=0)ELSE(I=(I'.COUNT.)',
               IA=N(I),F=271))
 /DC(271)*P/   IF(IA.EQ.273145)THEN(I=(I'.COUNT.)',F=272)ELSE(ERROR=10,
               G=0,F=0)
COMMENT,BLOCK AND COMPOUND STATEMENT
 /DC(272)*P/   BA=STCRAJ(BLKNUM),F=273
 /DC(273)*P/   T=(BA'.COUNT.)',BLKNUM=(BLKNUM'.COUNT.)',F=274,SYM=1
 /DC(274)*P/   IA=N(I),IF(SYM.EQ.0)THEN(F=275)
 /DC(275)*P/   IF(IA.EQ.3333)THEN(I=(I'.COUNT.)',F=274)ELSE(IF(SRCH.NE.0)
               THEN(F=251)ELSE(IF(S.EQ.53)THEN(F=276)ELSE(F=251)))
COMMENT,PROGRAM
 /DC(276)*P/   IF(IA.NE.53)THEN(ERROR=11,G=0,F=0)ELSE(I=(I'.COUNT.)')
COMMENT,HERE BEGINS SPECIAL ROUTINES
C
COMMENT,FETCH CHARACTER FROM MEMORY
 /SYM*SYC(0)*P/ IF(SYM.EQ.1)THEN(SRCH=0,S=MEM(J),J=J.COUNT.,FSY=2,SA=0)
 /SYC(2)*P/    IF(S.EQ.14)THEN(J=J.COUNT.,SRCH=1,FSY=5,S=MEM(J))ELSE(
               IF(S.EQ.33)THEN(FSY=10,J=J.COUNT.,S=MEM(J),SA=33)ELSE(
               IF(S.EQ.54)THEN(FSY=14,J=J.COUNT.,S=MEM(J),SA=54)ELSE(
               SYM=0,FSY=0)))
 /SYC(5)*P/    IF(S.EQ.14)THEN(SYM=0,FSY=0)ELSE(SA=SA(6-21)-S,S=MEM(J),
               J=J.COUNT.)
 /SYC(10)*P/    IF((S.EQ.73)+(S.EQ.33))THEN(SA=SA(6-21)-S,SYM=0,FSY=0,
               SRCH=1)ELSE(S=33,J=(J'.COUNT.)',SYM=0,FSY=0)
 /SYC(14)*P/   IF(S.EQ.54)THEN(SA=SA(6-21)-S,SYM=0,FSY=0,SRCH=1)ELSE(
               S=54,J=(J'.COUNT.)',SYM=0,FSY=0)
COMMENT,CHANGE OPERAND FROM LINK TO VALUE
 /LINK*LC(0)*P/  IF(LINK.EQ.1)THEN(OA=0(K),FL=2)
 /LC(2)*P/     IF(OA(TYPEO).EQ.0)THEN(T1=OA(35-44),FL=4)ELSE(FL=7)
 /LC(4)*P/     T1=T,T=T1,FL=5
 /LC(5)*P/     TA=DSAT(T),FL=6
 /LC(6)*P/     OA(VALUEO)=TA(VALUE),T=T1,FL=7
 /LC(7)*P/     X=OA(VALUEO),K=(K'.COUNT.)',FL=10
 /LC(10)*P/    OA=0(K),FL=11
 /LC(11)*P/    IF(OA(TYPEO).EQ.0)THEN(T1=OA(35-44),FL=13)ELSE(FL=16)
 /LC(13)*P/    T1=T,T=T1,FL=14
 /LC(14)*P/    TA=DSAT(T),FL=15
 /LC(15)*P/    OA(VALUEO)=TA(VALUE),OA(TYPEO)=TA(TYPE),T=T1,FL=16
 /LC(16)*P/    A=OA(VALUEO),LINK=0,FL=0
COMMENT,ADD SEQUENCE
 /ADD*AC(0)*P/   IF(ADD.EQ.1)THEN(SI=0,OV=0,L7=0,AV=0,FA=2)
```

Fig. 30 Program Listing 5

```
 /AC(2)*P/     IF(A(0).NE.X(0))THEN(SI=1,X(MX)=X(MX)',ADDS=1,FA=4
 /AC(4)*P/     IF(ADDS.EQ.0)THEN(
               IF((SI *OV'*LZ').EQ.1)THEN(A=A'),
               IF((SI'*OV).EQ.1)THEN(AV=1),
               IF((SI*OV'*LZ).EQ.1)THEN(A=0),
               IF((SI*OV*LZ').EQ.1)THEN(A(M)=A(M).COUNT.),
               ADD=0,FA=0)
COMMENT,SUBTRACT SEQUENCE
 /SUB*SC(0)*P/  IF(SUB.EQ.1) THEN(X(0)=X(0)',ADD=1,FS=2)
 /SC(2)*P/     IF(ADD.EQ.0)THEN(SUB=0,FS=0)
COMMENT,ADD SUBSEQUENCE
 /ADDS*ASC(0)*P/  IF(ADDS.EQ.1)THEN(AR=(0-A(M)).ADD.(0-X(MX)),FAS=2
 /ASC(2)*P/    A(M)=AR(1-22),IF(AR(0).EQ.1)THEN(OV=1),FAS=3
 /ASC(3)*P/    IF(A(M).EQ.777777)THEN(LZ=1),ADDS=0,FAS=0
COMMENT,MULTIPLICATION SEQUENCE
 /MLP*MC(0)*P/  IF(MLP.EQ.1)THEN(OV=0,LZ=0,SI=0,Q(0)=0,FM=2)
 /MC(2)*P/     IF(A(0).NE.X(0))THEN(SI=1),CNT=22,Q(MQ)=X(MX),X=A,A=0,FM=4
 /MC(4)*P/     CNT=(CNT'.COUNT.)',IF(Q(22).NE.1)THEN(FM=6)ELSE(ADDS=1,
               FM=5)
 /MC(5)*P/     IF(ADDS.EQ.0)THEN(FM=6)
 /MC(6)*P/     A(2-22)=A(1-21),Q(MQ)=A(22)-Q(1-21),
               IF(CNT.NE.0)THEN(FM=4)ELSE(
               IF(SI.EQ.1)THEN(A(0)=1,Q(0)=1),A(M)=Q(MQ),Q(MQ)=A(M),
               MLP=0,FM=0)
               END
$SIMULATE
*OUTPUT    CLOCK(1)=AV,OV,ERROR,J,K,I,T,BLKNUM,S,TA,OA,IA,SA,OUT(0),
          OUT(1),OUA,INA
*SWITCH    1,POWER=ON
*SWITCH    2,START=ON
*LOAD
  INP(0)=15,
  MEM(0- )=53,14,27,31,45,14,14,27,25,51,14,21,73,22,33,73,
  MEM(20- )=14,25,21,24,14,74,21,34,33,73,22,13,21,20,2,33,73,
  MEM(41- )=14,31,63,25,14,74,22,34,14,25,45,24,14,53
*SIM       1000,13
9999
$IBSYS
$RESTORE
9999
```

Fig. 31 Program Listing 6

Table 4 Flow of the Sample Program

CT	Nonterminals	Symbols	Comments
2-7	IP-INC	$	Store $ into operator stack
8-101	IP-E1-INC	'BEGIN'	After recognizing 'BEGIN', scan block for labels and store 'BEGIN' on operator stack.
102-154	IP-INC-IP-I I-V-INC-IP- I-V-TL-D-INC	'INTEGER' A,B;	Recognize declaration. Store ";" on operator stack. A and B in DSAT.
155-203	IP-INC-IP- INC-IP-I-V- RS-BS-US- S-INC	'READ'(A);	Recognize read statement and store value of A in DSAT.
204-301	IP-I-V-INC-IP-I -V-P-F-T-AE-INC -IP-UN-P-F-T-AE -AS-BS-US-S-INC	B=A+2;	Recognize assignment statement. Compute A+2 and store result in B's location in DSAT.
302-357	IP-INC-IP-INC- IP-I-V-WS-BS- US-S	'WRITE'(B)	Recognize write statement. Write value of B on output channel.
358-364	B-PR	'END'	Recognize program

terminal boxes as the computer is scanning the program. The third column shows the symbols or symbol being scanned during this point and the fourth column describes the essential operation. Using this table as a guide, the reader is encouraged to go through the computer operation himself for the given program input.

5. Acknowledgement

The preparation of this paper was stimulated by the two courses offered at the Computer Science Center of University of Maryland: "Structure of Programming Languages"instructed by Dr. Victor Schneider where the ALGOL Interpreter concepts and structure were outlined, and "Simulation of Digital Computers" instructed by Dr. Yaohan Chu where a methodology of computer-structure description and simulation were presented. This work has been an outgrowth of a project of the above simulation course. The author would like to acknowledge Dr. Yaohan Chu's encouragement and effort which has made this report a reality and Nancy A. Nowell's effort in typing this report.

6. References

[1] Y. Chu, "Introduction to Computer Organization", Prentice Hall, Inc., 1970.

[2] P. Wegner, "Programming Languages, Information Structures, and Machine Organization, "McGraw Hill, New York, 1968, Chapter 4 and Appendix 1.

[3] P. Naur (ed), Revised Algol report. Comm. of ACM, Jan., 1963.

[4] S. Rosen (ed), "Programming Systems and Languages, "McGraw Hill, New York, 1967, Part 3.

[5] Y. Chu, The Bi-Tran Six Computer Organization, University of Maryland, Computer Science Center, July 1968.

[6] C. Mesztenyi, Computer Design Language, Simulation and boolean translation, University of Maryland, Computer Science Center, Technical Report 68-72, June 1968.