N 70 42366

CR 114159

Design of an ALGOL Machine

by

Thomas F. Signiski

UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER
COLLEGE PARK, MARYLAND

CASE FILE
COPY

Technical Report 70-131
NGR-21-002-206

September 1970

Design of an ALGOL Machine

by

Thomas F. Signiski

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.

Table of Contents

Abstract

1. Introduction

    1.1 Organization of the report
    1.2 The subset of ALGOL
    1.3 Design Assumptions

        1.3.1 Input Sequence
        1.3.2 Output Sequence
        1.3.3 Error Sequence

2. Configuration of the Machine

    2.1 Memories

        2.1.1 Allocation of Memory M1

            2.1.1.1 BLOCK NUMBER COUNTERS
            2.1.1.2 DELIMITER STACK
            2.1.1.3 FLAG STACK
            2.1.1.4 INITIAL STACK
            2.1.1.5 COUNT STACK

        2.1.2 Allocation of Memory M2

            2.1.2.1 NAME TABLE
            2.1.2.2 PROGRAM AREA
            2.1.2.3 INPUT QUEUE
            2.1.2.4 OUTPUTSTRING
            2.1.2.5 OPERAND LIST
            2.1.2.6 LINK TO FORLIST STACK

    2.2 Registers
    2.3 CLD Description

3. Program Execution

    3.1 An ALGOL Program
    3.2 Machine Conditions at the Start of Execution
    3.3 Execution of the Example Program

        3.3.1 Label search
        3.3.2 Declarations
        3.3.3 The READ Statement
        3.3.4 The First Conditional Statement
        3.3.5 Initial Value Assigned to TEMP
        3.3.6 The Iteration

            3.3.6.1 Process the For List Element
            3.3.6.2 Process the Statement Contained in the Iteration
            3.3.6.3 Repeat the Iteration

        3.3.7 The Boolean Assignment Statement
        3.3.8 The Second Conditional Statement
        3.3.9 Exiting the Program

4. Sequence Charts

4.1 Initial Point Sequence
4.2 Output String Initialization Sequence
4.3 Number Processing Sequence
4.4 Block Entry Sequence
4.5 Block Exit Sequence
4.6 Delimiter TRUE Sequence
4.7 Delimiter FALSE Sequence
4.8 Iteration Initialization Sequence
4.9 Declaration Initialization Sequence
4.10 NAME TABLE Activity Sequence

4.10.1 Hash Coding a Name
4.10.2 NAME TABLE Entry

4.10.2.1 Handling Collisions
4.10.2.2 Entering a Name

4.10.3 NAME TABLE Search

4.11 Label Processing Sequence
4.12 Variable Processing Sequence
4.13 Factor Sequence

4.13.1 Multiplication
4.13.2 Division

4.14 Term Sequence

4.14.1 Addition and Subtraction
4.14.2 Unary Minus

4.15 Sum Sequence
4.16 Logical Expression Sequence
4.17 Arithmetic Expression Sequence
4.18 Assignment Sequence
4.19 Unconditional Statement Sequence
4.20 End of Statement Sequence
4.21 Program Body Sequence
4.22 Iteration Control Sequence
4.23 Read/Write Execution Sequence
4.24 Error Sequence

5. Acknowledgements

6. References

Appendix: Description of the Subset of ALGOL

A.1 Elements of the Subset

A.1.1 Character Set
A.1.2 Delimiters
A.1.3 Names
A.1.4 Numbers
A.1.5 Logical Values

A.2 Expressions

A.2.1 Variables
A.2.2 Arithmetic Expressions
A.2.3 Logical Expressions
A.2.4 Designational Expressions

A.3 Statement

A.3.1 Unconditional Statement

A.3.1.1 Assignment
A.3.1.2 Transfer
A.3.1.3 Communication

A.3.2 Conditional Statement
A.3.3 Iteration
A.3.4 Empty Statement

A.4 Program Structure
A.5 Declarations

Abstract

This report describes the design of a syntax-directed machine whose language is a subset of ALGOL to be referred to as ALGOL for simplicity. This machine is described in the Computer Design Language (CDL). The subset of ALGOL is first described. Computer elements such as memories and registers as well as the allocations of the memories for serving functional elements such as stacks are then described. Program execution by the machine is then illustrated in great detail with an example. This is followed by the presentation of a set of sequence charts to implement the syntax-directed algorithm.

This work aims to demonstrate the feasibility of a high-level language processor and to illustrate the use of the CDL to describe it.

Design of an ALGOL Machine

T. F. Signiski

1. Introduction

Algebraic languages such as FORTRAN and ALGOL allow a user to write programs in a format which is close to his natural language. However, when such languages are implemented on a general-purpose computer, the user's programs must be translated into machine code before they can be executed. To the users, the time required for the translation is not directly productive since it produces no output data. It is greatly desirable that the translation be eliminated.

One solution for accomplishing this is to build a machine which executes an algebraic language directly. In other words, a machine which has the algebraic language as its machine language. Such a machine would be a hardware interpreter of the language that it executes. To demonstrate the feasibility of such a machine, the design of a machine that executes a subset of ALGOL is described in this report. This machine is a syntax-directed (1,2) computer described in the Computer Design Language (CDL) (3,4).

1.1 Organization of the Report

This report is divided into four sections. This first section defines the subset of ALGOL and describes the machine sequences that are assumed to be available but are not implemented in this report. Section two describes the configuration of the machine except for the control part to be described in a separate report. Section three gives a step by step description of the machine's operations in executing an ALGOL program. Section four presents the sequence charts of the machine's sequences and a detailed description of these sequences. A list of references is provided. An informal but detailed description of the subset of ALGOL is attached as an Appendix.

1.2 The Subset of ALGOL

The subset of ALGOL described in Appendix A of this report was selected because it provides a nontrivial example of a higher level language machine and is small enough so that the description of the machine design is not overly complicated. The major features of ALGOL that are not included in this subset are procedures, arrays, switches, real variables, comment statements, logical operators, and the arithmetic operation of exponentiation. The subset does include iterations, relational operators and unconditional statements.

1.3 Design Assumptions

In the interest of simplifying the machine description, certain machine sequences have not been implemented although it is assumed the machine executes these sequences. These sequences are discussed below.

1.3.1 Input Sequence

It is assumed that the ALGOL program which is to be executed has already been read into memory and is ready for execution. In the interest of execution speed it is also assumed that some pre-processing has taken place during the input phase. This pre-processing consists of eliminating the blank spaces from the program and placing the remaining program constituents* into separate locations in memory. Strings of characters which are to be printed out during the execution of a WRITE statement are placed seven characters per word in successive memory locations. The constants in the program are converted from BCD to 36-bit, fixed point binary form and are identified with the special character 17_8 in the left most character position of their memory location. All other program constituents are stored left-adjusted in their respective locations. Table 1 shows a labeled assignment statement and a WRITE statement as they would

---

*A program constitutent is any name, number, delimiter, or string of characters enclosed by apostrophes as specified in the Appendix, Sections A.1 and A.3.1.3.

Table 1 Example of ALGOL Statements Stored in Memory

Program Constitutent	Memory Contents (octal)
L	43000000000000
:	15000000000000
ARC	21512300000000
=	13000000000000
B	22000000000000
+	20000000000000
3	17000000000003
$	53000000000000
WRITE	66513163250000
(	74000000000000
'	14000000000000
THE VAL	63302560652143
UE OF A	64256046266021
RC IS	51236031626060
'	14000000000000
,	73000000000000
ARC	21512300000000
)	53000000000000
$	53000000000000

appear stored in the machine. Each line in the table corresponds to a memory location. The two statements shown in the Table are:

```
L:ARC=B+3$
  WRITE('THE VALUE OF ARC IS',ARC)$
```

It is also assumed that the data which is to be read in during execution of the program has already been read in, converted to binary form, and stored in a separate buffer area.

1.3.2 Output Sequence

When a WRITE statement is to be executed, the strings of characters and the values of the variables that are to be printed out are prepared for output but the output is not implemented. The strings of characters are assembled in a separate buffer area of memory called the OUTPUTSTRING area. The values of the variables are fetched from memory and converted to BCD form. The machine then transfers to another sequence.

1.3.3 Error Sequence

The machine transfers to this sequence whenever a syntax error is detected. In this sequence it is assumed the machine identifies the error, prints out a diagnostic message, and halts.

2. Configuration of the Machine

The computer elements selected to implement the ALGOL machine are shown in Figure 1 except for the control elements to be described in a separate report.

2.1 Memories

The machine is designed with two memories, M1 and M2. M1 is a 128-word, 6-bit-per-word memory and M2 is a 4096-word, 42-bit-per-word memory. Both are random-access core memories. M1 has address register AR1 and buffer register BR1. M2 has address register AR2 and buffer register BR2. Both memories are treated as though they contain several distinct areas. These areas are described below. Tables 2 and 3 list the names of these areas, their memory locations and their sizes.

2.1.1 Allocation of Memory M1

In memory M1, five areas are allocated for use as counters and stacks. These are described below.

2.1.1.1 BLOCK NUMBER COUNTERS

Each counter, which comprises one memory word, corresponds to a block level in a nesting of blocks as depicted in Figure 2. During execution of a program, the counters contain current counts of the number of blocks which have been entered at their corresponding block levels.

2.1.1.2 DELIMITER STACK

This stack dynamically stores the delimiters (see Appendix, Section A.1.2) encountered while scanning the program. The delimiters are placed on and removed from this stack in such a manner as to execute the program in accordance with the syntax and semantics of the allowed subset. The top element of this stack is stored in the six-bit register D1 rather than in memory to reduce the number of memory accesses. The single character delimiters (*, /, +, etc.)

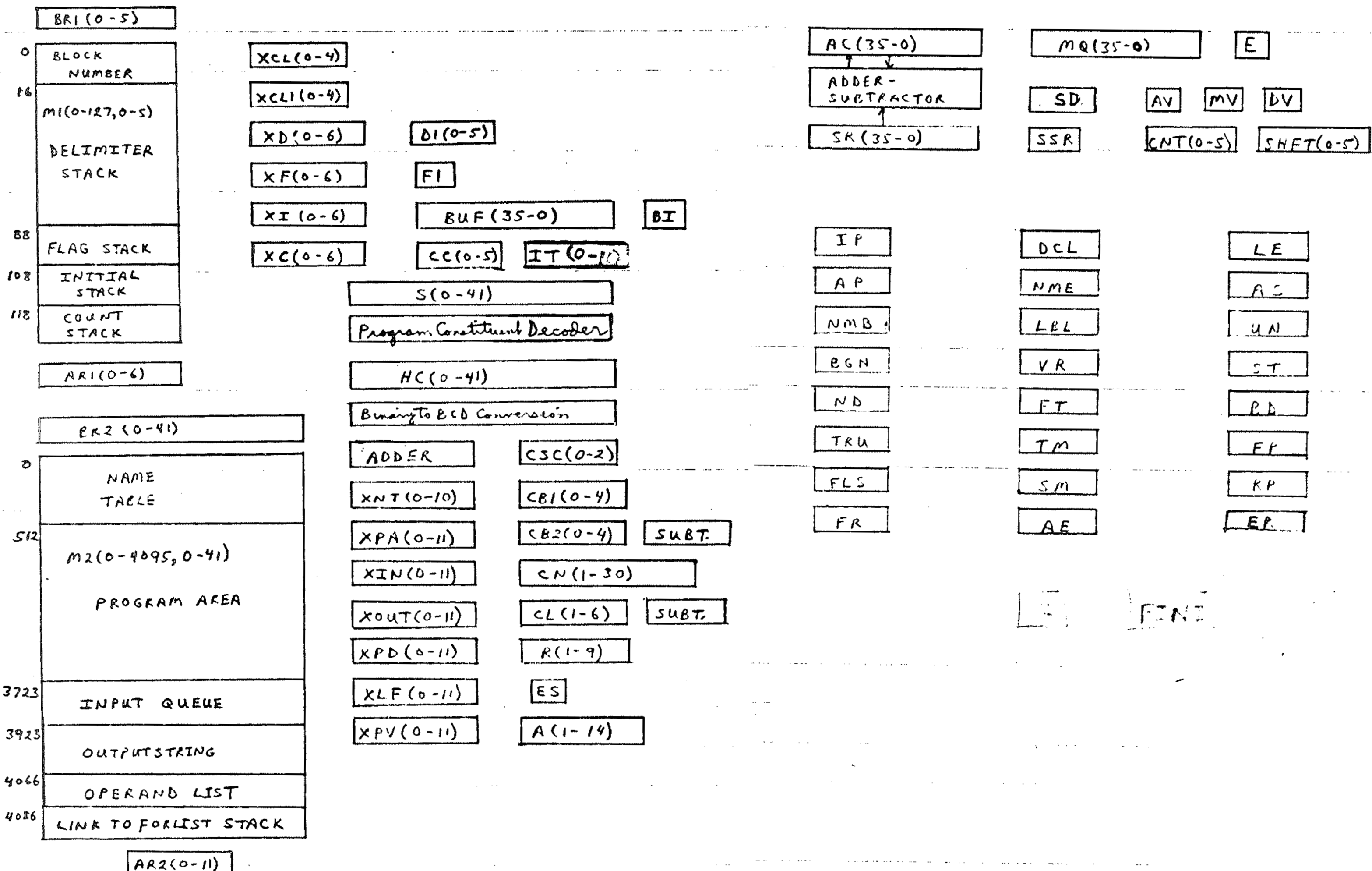


Figure 1 Elements of the ALGOL Machine

Table 2 Areas of Memory M1

Name	Locations	Size
BLOCK NUMBER COUNTERS	0-15	16
DELIMITER STACK	11-87	72
FLAG STACK	88-107	20
INITIAL STACK	108-117	10
COUNT STACK	118-127	10

Table 3 Areas of Memory M2

Name	Locations	Size
NAME TABLE	0-511	512
PROGRAM AREA	512-3722	3211
INPUT QUEUE	3723-3922	20
OUTPUTSTRING	3923-4065	1001
OPERAND LIST	4066-4085	20
LINK TO FORLIST STACK	4086-4095	10

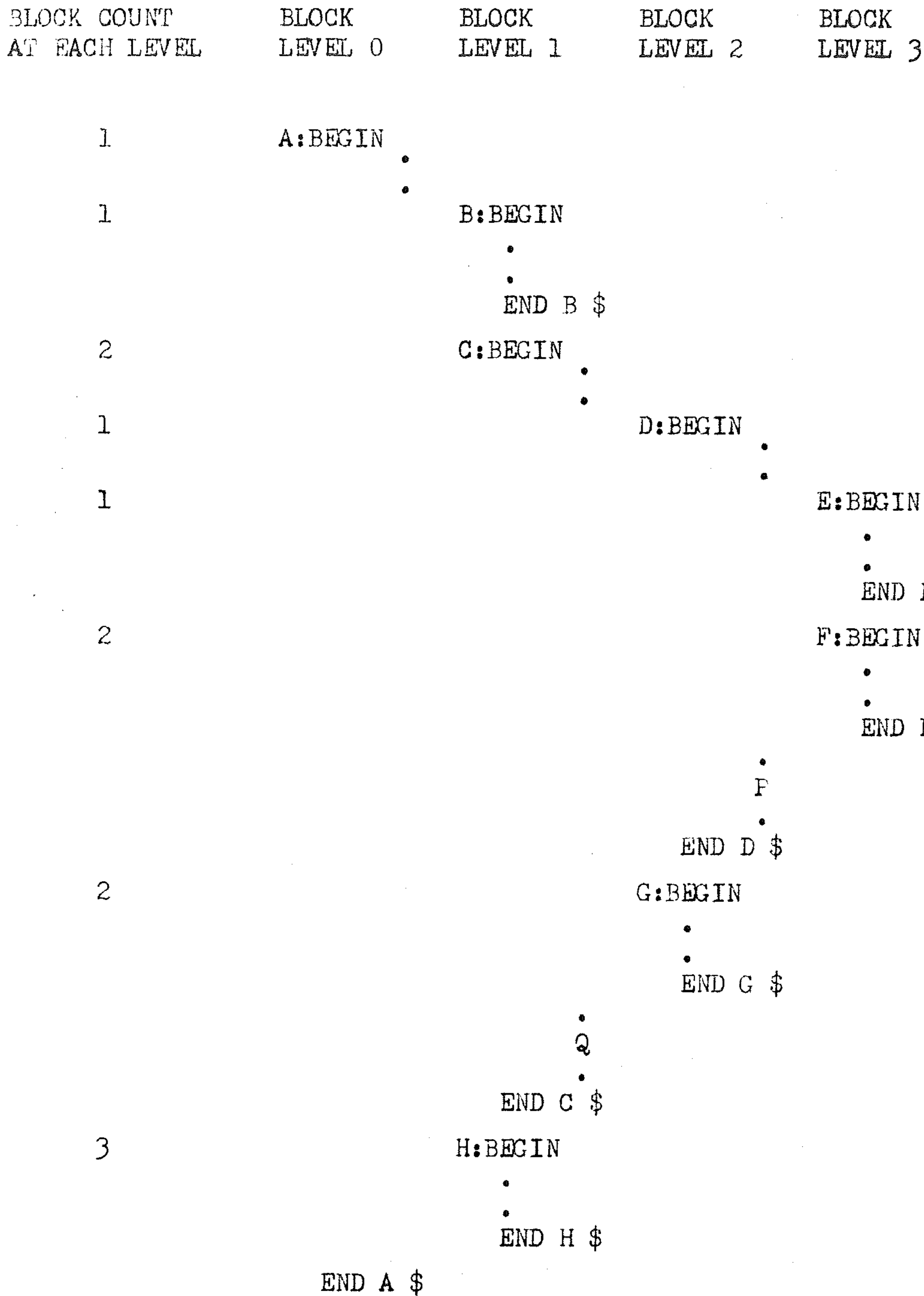


Figure 2: Program illustrating numbering system for Block Levels and Block Numbers.

are stored in their internal form (see Table 4) whereas the multiple character delimiters (BEGIN, GOTO, IT, etc.) are stored as shown in Table 5. The code for unary minus is also shown in Table 5.

2.1.1.3 FLAG STACK

The elements of this stack indicate whether or not certain parts of the program are to be executed. Each time a conditional statement is encountered a new element is placed on this stack. It is removed after the conditional statement has been completely scanned. An element can be assigned one of two values; one or zero. The value of the element is determined respectively by the validity or non-validity of the condition specified in the conditional statement. Only the top element of the stack controls the execution process; this element is referred to as the current flag. If the value of the current flag is one, the program is executed as well as scanned whereas if it is zero, the program is simply scanned. At the start of execution an initial flag is placed on this stack with a value of one. This initial flag controls the program execution while no conditional statements are being processed. Since the value of the current flag is checked frequently, the top element is stored in the single-bit register F1 rather than in memory to reduce the number of memory accesses.

2.1.1.4 INITIAL STACK

The INITIAL STACK elements control the processing of for list elements of the type STEP-UNTIL. If the value of the current flag is one (see Section 2.1.1.3), a new element is added to this stack each time an iteration is encountered. It is removed at the completion of the iteration. These elements are assigned an initial value of one. When a STEP-UNTIL for list element is first encountered, the corresponding INITIAL STACK element (always the top element) is set to zero and the iteration's controlled variable is assigned its initial value for the for list element. Thereafter, when the STEP-UNTIL for list element is scanned, the zero value of the INITIAL STACK element causes the

Table 4 Six-bit Internal Character Codes for the Algol Machine

Letters			
Character	Code (octal)	Character	Code (octal)
A	21	N	45
B	22	O	46
C	23	P	47
D	24	Q	50
E	25	R	51
F	26	S	62
G	27	T	63
H	30	U	64
I	31	V	65
J	41	W	66
K	42	X	67
L	43	Y	70
M	44	Z	71
Digits			
Character	Code (octal)	Character	Code (octal)
0	00	5	05
1	01	6	06
2	02	7	07
3	03	8	10
4	04	9	11
Other Symbols			
Character	Code (octal)	Character	Code (octal)
=	13	$	53
.	14	*	54
:	15	␣	60
+	20	/	61
)	34	,	73
-	40	(	74

Table 5 Codes Used to Store the Multiple Character Delimiters on the DELIMITER STACK

Delimiter	Code (Octal)	Delimiter	Code (Octal)
BEGIN	22	GOTO	27
FOR	26	READ	51
DO	24	WRITE	47
STEP	62	LSS	43
UNTIL	64	LEQ	71
WHILE	66	EQL	50
IF	31	GEQ	67
THEN	63	GTR	70
ELSE	65	NEQ	45
BOOLEAN	21	- (unary minus)	44
INTEGER	46		

value of the controlled variable to be changed by the amount of the increment specified by the STEP-UNTIL for list element.

When an iteration has been executed the number of times specified by its STEP-UNTIL for list element, the corresponding INITIAL STACK element is reset to one and thus is initialized in the event that another STEP-UNTIL for list element is encountered in the iteration's for list.

2.1.1.5 COUNT STACK

A stack of counters. If the value of the current flag is one (see Section 2.1.1.3), a new element is added to this stack each time an iteration is encountered. It is removed at the completion of the iteration. These counters are assigned an initial value of one. As each for list element of a for list is satisfied, the value of the associated counter is incremented by one. Thus, each time the machine returns to an iteration's for list to determine whether or not the iteration's statement should be executed again, it can select the correct for list element for testing by scanning the for list and counting the for list elements and comparing the count to the top element of this stack.

2.1.2 Allocation of Memory M2

In memory M2, six areas are allocated for use as tables, queues, and stacks. These are described below.

2.1.2.1 NAME TABLE

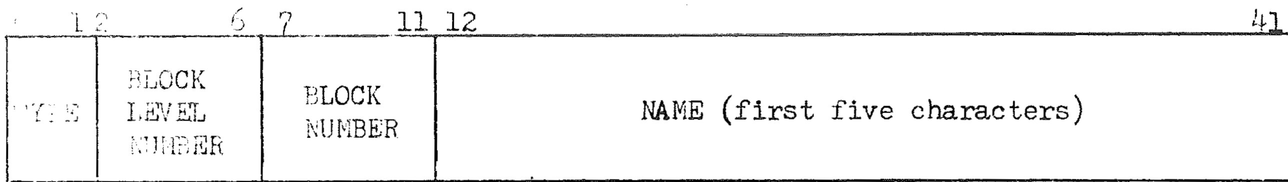
This table stores the variables and labels declared in the program along with their values and additional information which allows the machine to determine in which block of the program they are declared. Scatter storage is used to address the names (5). Each entry in this table is a two word node. The format of the first word of a node is the same for all names entered

into this table. This format is depicted in Figure 3(a). It is divided into four fields. The two-bit field TYPE identifies the name as either a label, an integer variable, or a Boolean variable. The two 5-bit fields BLOCK LEVEL NUMBER and BLOCK NUMBER refer to the numbering system depicted in Figure 2 and contain the values of the block in which the name is declared. The 30-bit field NAME contains the first five characters of the name. If the name is a label, the format of the second word of its node is as shown in Figure 3(b). This word also contains the five-bit field BLOCK LEVEL NUMBER which is contained in the first word. The 11-bit field L(LABEL) contains the address of the location in the PROGRAM AREA of M2 which contains the colon following the label.

Figure 3(c) depicts the format of the second word of a node for a name that is a variable. The six-bit field VAI indicates whether or not a value has been assigned to the variable. This field receives the code 77_8 when the variable is entered into the table. This code indicates that the variable has not been assigned a value and therefore cannot be used as an operand in an expression. When a variable is assigned a value, field VAI is set to zero and the value is placed in the right-most 36 bits of the word. For integer variables the value consists of a sign bit and 35 magnitude bits with negative values in two's complement form. For Boolean variables the value is an unsigned one or zero.

2.1.2.2 PROGRAM AREA

This area holds the Algol program that is being executed. Each location of this area contains a single constituent of the program except that strings of characters which are to be printed out during the execution of a WRITE statement are placed seven characters per word in successive words. The constants occuring in the program are stored as 36 bit signed integers and are identified by the special character 17_8 in the left-most character position of their word. The other constituents of the program are stored left justified in BCD form (see Table 1).



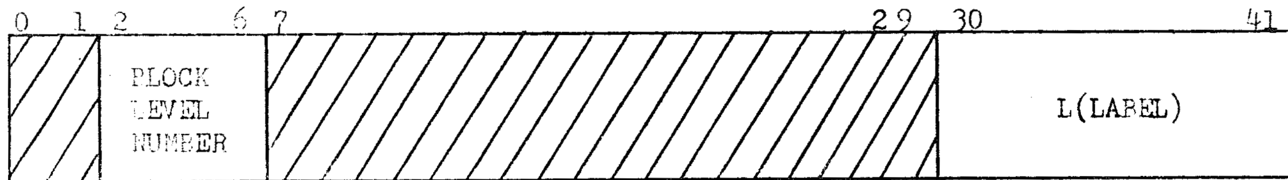
TYPE CODE
11 label

10 integer variable

01 Boolean variable

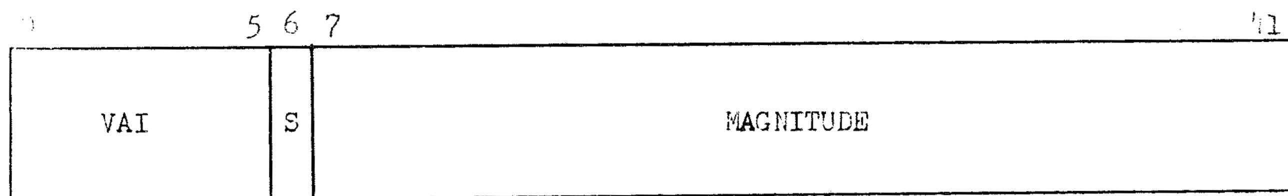
00 undefined

(a) first word of all nodes



L(LABEL) = the address of the location in the PROGRAM AREA of M2 containing the colon which follows the label.

(b) second word of nodes storing labels



VAI = Value Assigned Indicator

(c) second word of nodes storing variables

Figure 3 Word Formats for NAME TABLE area of M2

2.1.2.3 INPUT QUEUE

Values (data) which are to be read in during execution of the program are stored in this area. They are stored in the order in which they are to be read in with the first location containing the first value. Numbers are stored in fixed point, binary form in the right-most 36 bits. Negative numbers are stored in two's complement form. The logical values TRUE and FALSE are stored as 1 and 0 respectively.

2.1.2.4 OUTPUTSTRING

When a string of characters is to be printed out during the execution of the program, it is first assembled in this area of memory. Table 6 shows the contents of this area of memory after a string of characters has been assembled for output. The string shown in the table is 'THIS IS AN EXAMPLE STRING AS IT WOULD APPEAR IN MEMORY AFTER BEING ASSEMBLED FOR OUTPUT'.

2.1.2.5 OPERAND LIST

If the value of the current flag (see Section 2.1.1.3) is one, this list is used to dynamically store the operands and the results of operations on the operands that occur in a statement as it is being scanned. When a constant is encountered, it is stored in this list. When a variable is encountered, the address of its location in the NAME TABLE is stored in this list. Finally, when an operation is to be performed, the top element(s) of this list is fetched and after the operation is complete, the result is placed in this list.

Table 6 Example String Stored in the OUTPUTSTRING area of Memory M2

Memory Address	Memory Contents	
	Character Form	Internal Code
3923	THIS␣IS	63303162603162
3924	␣AN␣EXA	60214560256721
3925	MPLE␣ST	44474325606263
3926	RING␣AS	51314527602162
3927	␣IT␣WOU	60316360664664
3928	LD␣APPE	43246021474725
3929	AR␣IN␣M	21516031456044
3930	EMORY␣A	25444651706021
3931	FTER␣BE	26632551602225
3932	ING␣ASS	31452760216262
3933	EMBLED␣	25442243252460
3934	FOR␣OUT	26465160466463
3935	PUT␣␣␣␣	47646333606060
3936	␣␣␣␣␣␣␣	60606060606060
.	.	.
.	.	.
.	.	.
4065	␣␣␣␣␣␣␣	60606060606060

NOTE: The symbol ␣ indicates a space.

This list operates as a queue when a communication statement (see Appendix, Section A.3.1.3) is being executed and as a stack when any other type of statement is being executed. Figure 4 depicts the formats used in this list.

2.1.2.6 LINK TO FORLIST STACK

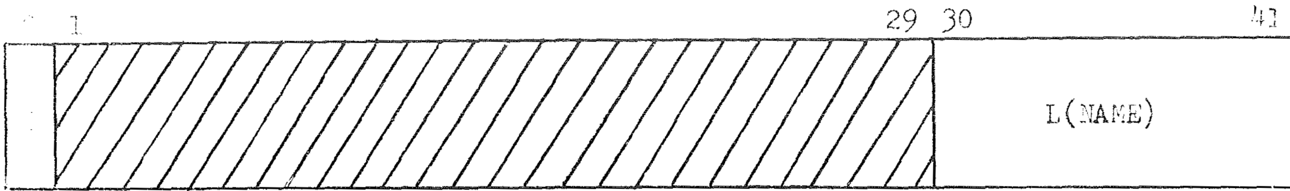
A stack of pointers. If the value of the current flag is one, a new element is added to this stack each time an iteration is encountered. It is removed at the completion of the iteration. These pointers contain the PROGRAM AREA address of the equal sign (=) preceding their corresponding iteration's for list. They enable the machine to return to the for list after the statement contained within the iteration has been executed. The format for this stack is shown in Figure 5.

2.2 Registers

There are 13 index registers, six for Memory M1 and seven for Memory M2. The six index registers for M1 and XCL, XCL1, XD, XF, XI, and XC. Index registers SCL, XD, XF, XI and XC store the current addresses of the BLOCK NUMBER COUNTER, DELIMITER STACK, FLAG STACK, INITIAL STACK, and COUNT STACK areas of M1 respectively. Index register XCL1 indicates the new current address of the BLOCK NUMBER COUNTER area when a transfer statement is executed in a program.

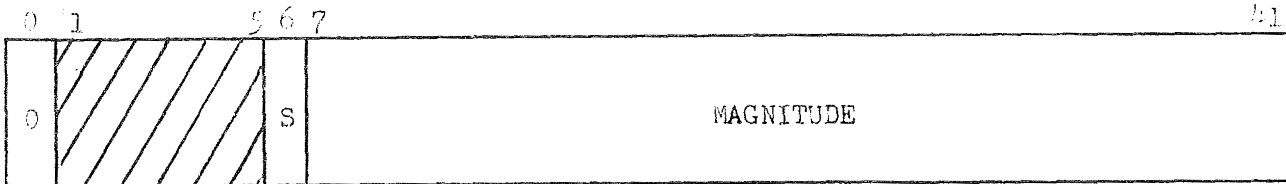
The seven index registers for Memory M2 are XNT, XPA, XIN, XOUT, XPD, XLF, and XPV. Index registers XNT, XPA, XIN, XOUT, XPD, and XLF store the current addresses of the NAME TABLE, PROGRAM AREA, INPUT QUEUE, OUTPUTSTRING, OPERAND LIST, and LINK TO FORLIST STACK areas of M2 respectively. Index register XPV stores the contents of index register XPA at the start of a label search and is used to address the elements of the OPERAND LIST during the execution of a communication statement.

Regsiter D1 is the top element of the DELIMITER STACK. Single-bit register F1 is the top element of the FLAG STACK and is called the current flag.



L(NAME) = the address of the location of the variable in the NAME TABLE

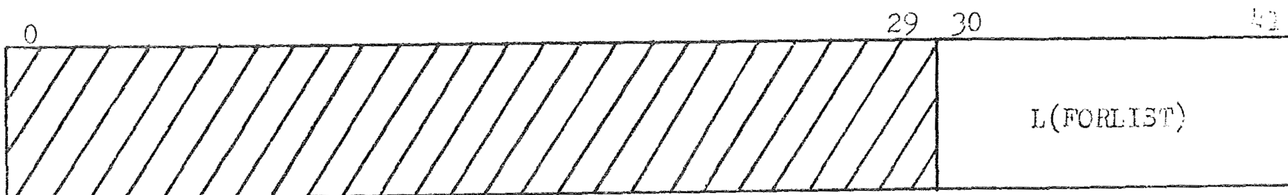
(a) link to the location of a variable in the NAME TABLE



(b) a constant or the result of an operation

Bit "0" indicates whether the word is a link or a value; 1=link, 0=value.

Figure 4 Formats for OPERAND LIST area of memory M2



L(FORLIST) = the address in the PROGRAM AREA of memory M2 of the equal sign (=) preceding an iteration's for list.

Figure 5 Word format for the LINK TO FORLIST STACK area of memory M2

Register BUF is the output register. The single-bit register BI indicates whether a Boolean or an integer variable is being printed out. A value of one indicates a Boolean variable while a value of zero indicates an integer variable.

Register CC counts the commas that are encountered in an iteration's for list when the machine is scanning the for list to return to the for list element currently being processed. By comparing the contents of CC to the contents of the top element of the COUNT STACK, the machine can determine when it reaches the proper for list element. Register IT is set to one at the start of an iteration. If a communication statement occurs within an iteration, the contents of XPD are transferred to this register. It is set to zero at other times.

Register S stores the constituents of the program when they are fetched from the PROGRAM AREA of M2. The program constituent decoder identifies the constituents while they are in this register.

Register HC and CSC are used in conjunction with index register XNT to hash code the declared labels and variables for placement into the NAME TABLE. HC holds the variable and CSC is a counter. XNT receives the hash coded address. HC is also used in conjunction with register AC to convert the values of integer variables from binary to BCD form for output.

Register CB1 stores the count of the number of blocks which have been entered at the current block level during execution of the program. Referring to Figure 2, at point P in the execution of the program shown, register CB1 would contain the value one since only one block (block D) has been entered at block level two. At point Q however, register CB1 would contain the value two since two blocks (blocks B and C) have been entered at block level one. Note that the fact block B has been exited has no effect on the count.

Registers CB2, CN and CL are used to determine if a collision exists when a name is hash coded for entry into the NAME TABLE and the hashed address is non-empty. These registers are also used to find the most recent entry of a name during a search of the NAME TABLE.

Register R is used in conjunction with index register XNT, memory address register AR2, and the 12-bit adder to calculate a new address when a collision occurs while entering a name into the NAME TABLE and when a search is being made for the most recent entry of a name.

Single-bit register ES indicates whether a name is being entered or searched for in the NAME TABLE. If ES contains a one, an entry is being made whereas, if ES contains a zero, a search is being conducted. When a search of the NAME TABLE is being conducted, the type (label, Boolean, or integer) of the name and the address of its location in the NAME TABLE are placed in register A.

Registers AC, MQ and SR are used to perform the arithmetic operations of addition, subtraction, multiplication, and division. Registers AC and SR are also used to form the difference of the two operands when a relational operation is executed. Additionally, AC is used in conjunction with register HC to convert the values of integer variables from binary to BCD form for output. A full adder-subtractor is wired between registers AC and SR. The single-bit register SD indicates whether addition or subtraction is to occur. If SD contains a one, addition is specified; otherwise subtraction. The single-bit register E is a reference flipflop used in multiplication and division. Overflow in the operations of addition, multiplication, and division is indicated by the single-bit registers AV, MV and DV respectively. The single-bit register SSR stores the sign of the contents of register SR during arithmetic

operations. It is used to check for overflow in addition and subtraction and to set the sign of the quotient in division.

Register CNT counts the number of bits processed during multiplication and division and also indicates whether or not an inner block is being scanned during a label search and during a scan to the end of an iteration.

Register SHFT counts the number of bits that the divisor (register SR) is shifted left (scaled) so that the division can take place when the divisor is smaller than the quotient.

The twenty-four single-bit registers, IP, AP, NMB, etc., are control registers for calling the various sequences of the machine. During execution only one of these registers is set to one at any given time.

Single-bit register G is the run/stop control flipflop. Light FINI indicates when program execution is complete.

2.3 CDL Description

```
Comment, configuration of the Algol Machine.
Register, AR1(0-6),    $address register for M1
          AR2(0-11),   $address register for M2
          BR1(0-5),    $buffer register for M1
          BR2(0-41),   $buffer register for M2
Comment, the index registers used with memory M1.
Register, XCL(0-4),    $store address for BLOCK NUMBER COUNTERS in M1
          XCL1(0-4),   $transfer statement control register
          XD(0-6),     $store address for DELIMITER STACK of M1
          XF(0-6),     $store address for FLAG STACK of M1
          XI(0-6),     $store address for INITIAL STACK of M1
          XC(0-6),     $store address for COUNT STACK of M1
Comment, the index registers used with memory M2.
Register, XNT(0-10),   $store address for NAME TABLE of M2
```

```
Register, XPA(0-11),   $store address for PROGRAM AREA of M2
          XIN(0-11),   $store address for INPUT QUEUE of M2
          XOUT(0-11),  $store address for OUTPUTSTRING of M2
          XPD(0-11),   $store address for OPERAND LIST of M2
          XLF(0-11),   $store address for LINK TO FORLIST STACK of M2
          XPV(0-11),   $store address for OPERAND LIST of M2 during execution
                       of communication statements
Comment, the remaining registers
Register, D1(0-5),     $top element of DELIMITER STACK
          F1,          $top element of FLAG STACK
          BUF(35-0),   $output register
          BI,          $output control flipflop
          CC(0-5),     $count commas in for list
          IT(0-11),    $iteration control register
          S(0-41),     $identify constituents of the program
          HC(0-41),    $hold names for hash coding and convert integer values
                       from binary to BCD for output
          CSC(0-2),    $control counter used in hash coding of names
          CB1(0-4),    $stores block number count for current Block Level
          CB2(0-4),    $stores block number count for Block Level of name
                       fetched from NAME TABLE
Comment, the registers used when entering a name into or searching for a name
        in the NAME TABLE
Register, CN(1-30),    $compare the name being entered or searched for with
                       the name found in the NAME TABLE
          CL(1-6),     $compare current Block Level with the Block Level of
                       the name found in the NAME TABLE
```

```
Register,    R(1-9),       $generate random number to add to NAME TABLE address
                            if a collision occurs
             ES,           $control flipflop for entry or search of NAME TABLE
             A(1-14),      $store type and NAME TABLE address of variable found
                            in the NAME TABLE during a search
Comment, the registers used for arithmetic operations.
Register,    AC(35-0),     $accumulator
             MQ(35-0),     $multiplier-quotient register
             SR(35-0),     $storage register used in arithmetic operations
             SD,           $add-subtract control flipflop (add when 1)
             E,            $reference flipflop used in multiplication and division
             AV,           $add overflow indicator
             MV,           $multiply overflow indicator
             DV,           $divide overflow indicator
             SSR,          $store sign of register SR during arithmetic operations
             CNT(0-5),     $control counter
             SHFT(0-5),    $count leftshifts of register SR when setting up for
                            division
Casregister, AQ=AC-MQ
Comment, the registers which enable the sequences of the machine.
Register,    IP,           $control register
             AP,           $control register
             NMB,          $control register
             BGN,          $control register
             END,          $control register
             TRU,          $control register
             FLS,          $control register
             FR,           $control register
```

```
Register,   DCL,        $control register
            NME,        $control register
            LBL,        $control register
            VR,         $control register
            FT,         $control register
            TM,         $control register
            SM,         $control register
            AE,         $control register
            LE,         $control register
            AS,         $control register
            UN,         $control register
            ST,         $control register
            BD,         $control register
            FP,         $control register
            RP,         $control register
            ER,         $control register
            G,          $run/stop control register
Light,      FINI
Comment, description of the memories.
Memory,     M1(AR1)=M1(0-127,0-5),
            M2(AR2)=M2(0-4095,0-41)
Comment, description of the parallel adder-subtractor
Terminal, C(36-1)=(SD⊙AC(35-0))*SR(35-0)+(SD⊙AC(35-0))*C(35-0)+SR(35-0)*C(35-0),
          C(0)=0,
          SUM(35-0)=AC(35-0)⊙SR(35-0)⊙C(35-0)
Comment, description of the binary to BCD converter
```

```
Subregister, HC0(3-0)=HC(38-41),
             HC1(3-0)=HC(34-37),
             HC2(3-0)=HC(30-33),
             HC3(3-0)=HC(26-29),
             HC4(3-0)=HC(22-25),
             HC5(3-0)=HC(18-21),
             HC6(3-0)=HC(14-17),
             HC7(3-0)=HC(10-13),
             HC8(3-0)=HC(6-9),
             HC9(3-0)=HC(2-5),
Comment, these terminals indicate whether or not the contents of the above
         subregisters are greater than or equal to five.
Terminal,    AD30=HC0(3)+(HC0(2)*(HC0(1)+HC0(0))),
             AD31=HC1(3)+(HC1(2)*(HC1(1)+HC1(0))),
             AD32=HC2(3)+(HC2(2)*(HC2(1)+HC2(0))),
             AD33=HC3(3)+(HC3(2)*(HC3(1)+HC3(0))),
             AD34=HC4(3)+(HC4(2)*(HC4(1)+HC5(0))),
             AD35=HC5(3)+(HC5(2)*(HC5(1)+HC5(0))),
             AD36=HC6(3)+(HC6(2)*(HC6(1)+HC6(0))),
             AD37=HC7(3)+(HC7(2)*(HC7(1)+HC7(0))),
             AD38=HC8(3)+(HC8(2)*(HC8(1)+HC8(0))),
             AD39=HC9(3)+(HC9(2)*(HC9(1)+HC9(0)))
Comment, description of the program constituent decoder
Decoder,     S1(0-63)=S(0-5)    $decodes single character program constituents
                                and the first letter of multiple character pro-
                                gram constituents
Comment, single character program constituents
```

```
Terminal,     SEQ=S1(11),
              SAP=S1(12),
              SCO=S1(13),
              SNUM=S1(15),
              SPL=S1(16),
              SLP=S1(28),
              SMI=S1(32),
              SDLR=S1(43),
              SSTR=S1(44),
              SDI=S1(49),
              SCMA=S1(59),
              SRP=S1(60),
Comment, decoders for the remaining characters of multiple character constituents
Decoder,      S21(2-6)=S(6-8),
              S20(0,1,3,5,6)=S(9-11),
              S31(2-6)=S(12-14),
              S30(0-7)=S(15-17),
              S41(2-4,6)=S(18-20),
              S40(0-7)=S(21-23),
              S51,(2,4,6)=S(24-26),
              S50(0,3,5,7)=S(27-29),
              S61(2,6)=S(30-32),
              S60(0,1,5)=S(33-35),
              S71(4-6)=S(36-38),
              S70(0,1,5)=S(39-41)
Comment, the multiple character reserved names
Terminal,     BEGIN=S1(18)*S21(2)*S20(5)*S31(2)*S30(7)*S41(3)*S40(1)*S51(4)*
                    S50(5)*S61(6)*S60(0)*S71(6)*S70(0),
```

```
Terminal, END=S1(21)*S21(4)*S20(5)*S31(2)*S30(4)*S41(6)*S40(0)*S51(6)*
              S50(0)*S61(6)*S60(0)*S71(6)*S70(0),
          IF=S1(25)*S21(6)*S20(6)*S31(6)*S30(0)*S41(6)*S40(0)*S51(6)*S50(0)*
              S61(6)*S60(0)*S71(6)*S70(0),
          EQL=S1(21)*S21(5)*S20(0)*S31(4)*S30(3)*S41(6)*S40(0)*S51(6)*
              S50(0)*S61(6)*S60(0)*S71(6)*S70(0),
          GTR=S1(23)*S21(6)*S20(3)*S31(5)*S30(1)*S41(6)*S40(0)*S51(6)*
              S50(0)*S61(6)*S60(0)*S71(6)*S70(0),
          GEQ=S1(23)*S21(2)*S20(5)*S31(5)*S30(0)*S41(6)*S40(0)*S51(6)*
              S50(0)*S61(6)*S60(0)*S71(6)*S70(0),
          NEQ=S1(37)*S21(2)*S20(5)*S31(5)*S30(0)*S41(6)*S40(0)*S51(6)*
              S61(6)*S60(0)*S71(6)*S70(0),
         READ=S1(41)*S21(2)*S20(5)*S31(2)*S30(1)*S41(2)*S40(4)*S51(6)*
              S50(0)*S61(6)*S60(0)*S71(6)*S70(0),
        WRITE=S1(54)*S21(5)*S20(1)*S31(3)*S30(1)*S41(6)*S40(3)*S51(2)*
              S50(5)*S61(6)*S60(0)*S71(6)*S70(0),
         TRUE=S1(51)*S21(5)*S20(1)*S31(6)*S30(4)*S41(2)*S40(5)*S51(6)*
              S50(0)*S61(6)*S60(0)*S71(6)*S70(0),
        FALSE=S1(22)*S21(2)*S20(1)*S31(4)*S30(3)*S41(6)*S40(2)*S51(2)*
              S50(5)*S61(6)*S60(0)*S71(6)*S70(0),
      BOOLEAN=S1(18)*S21(4)*S20(6)*S31(4)*S30(6)*S41(4)*S40(3)*S51(2)*
              S50(5)*S61(2)*S60(1)*S71(4)*S70(5),
         THEN=S1(51)*S21(3)*S20(0)*S31(2)*S30(5)*S41(4)*S40(5)*S51(6)*
              S50(0)*S61(6)*S60(0)*S71(6)*S70(0),
         ELSE=S1(21)*S21(4)*S20(3)*S31(6)*S30(2)*S41(2)*S40(5)*S51(6)*
              S50(0)*S61(6)*S60(0)*S71(6)*S70(0),
          FOR=S1(22)*S21(4)*S20(6)*S31(5)*S30(1)*S41(6)*S40(0)*S51(6)*
              S50(0)*S61(6)*S60(0)*S71(6)*S70(0),
```

```
    STEP=S1(50)*S21(6)*S20(3)*S31(2)*S30(5)*S41(4)*S40(7)*S51(6)*
         S50(0)*S61(6)*S60(0)*S71(6)*S70(0),
   UNTIL=S1(52)*S21(4)*S20(5)*S31(6)*S30(3)*S41(3)*S40(1)*S51(4)*
         S50(3)*S61(6)*S60(0)*S71(6)*S70(0),
   WHILE=S1(54)*S21(3) *S20(0)*S31(3)*S30(1)*S41(4)*S40(3)*S51(2)*
         S50(5)*S61(6)*S60(0)*S71(6)*S70(0),
      DO=S1(20)*S21(4)*S20(6)*S31(6)*S30(0)*S41(6)*S40(0)*S51(6)*
         S50(0)*S61(6)*S60(0)*S71(6)*S70(0),
     LSS=S1(35)*S21(6)*S20(2)*S31(6)*S30(2)*S41(6)*S40(0)*S51(6)*
         S50(0)*S61(6)*S60(0)*S71(6)*S70(0),
     LEQ=S1(35)*S21(2)*S20(5)*S31(5)*S30(0)*S41(6)*S40(0)*S51(6)*
         S50(0)*S61(6)*S60(0)*S71(6)*S70(0),
 INTEGER=S1(25)*S21(4)*S20(5)*S31(6)*S30(3)*S41(2)*S40(5)*
         S51(2)*S50(7)*S61(2)*S60(5)*S71(5)*S70(1),
    GOTO=S1(23)*S21(4)*S20(6)*S31(6)*S30(3)*S41(4)*S40(6)*S51(6)*
         S50(0)*S61(6)*S60(0)*S71(6)*S70(0),
Comment, code the delimiters for input to the DELIMITER STACK
Terminal,     SDEL0=SMI+SSTR+SDI+THEN+ELSE+STEP+UNTIL+WHILE+LSS+LEQ+EQL+GTR+GEQ+
                   NEQ+READ+WRITE+INTEGER,
              SDEL1=SPL+SDI+SLP+BEGIN+IF+THEN+ELSE+FOR+STEP+UNTIL+WHILE+DO+
                   LEQ+GTR+GEQ+BOOLEAN+GOTO,
              SDEL2=SEQ+SAP+SSTR+SLP+IF+LEQ+EQL+GTR+READ,
              SDEL3=SAP+SSTR+SLP+ELSE+FOR+UNTIL+WHILE+DO+GEQ+NEQ+WRITE+INTEGER+
                   GOTO+SMI*IP,
              SDEL4=SEQ+BEGIN+THEN+FOR+STEP+WHILE+LSS+GEQ+WRITE+INTEGER+GOTO,
              SDEL5=SEQ+SDI+IF+THEN+ELSE+LSS+LEQ+GEQ+NEQ+READ+WRITE+BOOLEAN+GOTO,
              SDEL(0-5)=SDEL0-SDEL1-SDEL2-SDEL3-SDEL4-SDEL5
Comment, register S contains a name used as a label or a variable whenever charac-
```

ter position one of S contains a letter and none of the reserved name terminals listed above are activated. The following terminal is activated by these conditions

```
Terminal, SVAR=(S1(17)+S1(18)+S1(19)+S1(20)+S1(21)+S1(22)+S1(23)+S1(24)+
          S1(25)+S1(33)+S1(34)+S1(35)+S1(36)+S1(37)+S1(38)+S1(39)+
          S1(40)+S1(41)+S1(50)+S1(51)+S1(52)+S1(53)+S1(54)+S1(55)+S1(56)+
          S1(57))*(SDEL0'*SDEL1'*SDEL2'*SDEL3'*SDEL4'*SDEL5')
Comment, the relational operators
Terminal, SRO=LSS+LEQ+EQL+GEQ+GTR+NEQ
```

3. Program Execution

This section describes how the machine executes an ALGOL program stored in the memory.

3.1 An ALGOL Program

Figure 6 depicts a program written in the previously-described subset of ALGOL. This program reads a value for variable N and calculates N! if this value is greater than or equal to zero. If this value is less than zero, the program indicates to the programmer that the value for N is illegal.

3.2 Machine Conditions at the Start of Execution

The memories of the machine are first cleared (all locations set to zero). Then, the program is read in the stored in the PROGRAM AREA of memory M2 as shown in Table 7. The value(s) to be assigned to N when the READ statement is executed is also read in with the program and stored in the INPUT QUEUE area of memory M2. The registers of the machine are next initialized and execution begins. Tables 8, 9, and 10 list the contents of the different areas of both memories and those of the registers at the start of execution. The index registers which normally contain the current addresses of the areas of both memories are initially set to a value which is one less than their corresponding area's first address. The current flag, F1, is set to one; thus causing the machine to start executing as well as scanning the program. The special character 57_8 (symbolized by @ in Tables 11 to 28) is placed on the DELIMITER STACK (D1). This character insures that a character other than zero is found the first time the DELIMITER STACK is referenced.

3.3 Execution of the Example Program

The actions of the ALGOL Machine in executing the program shown in Figure 7 will now be described. For this example N will receive a value of three. The reader should refer to Table 7 in following this description.

```
BEGIN INTEGER N, TEMP, V $
      BOOLEAN X $
      READ (N) $
      IF N LSS O THEN
      BEGIN X=FALSE $
         L1:GOTO L
      END $
      TEMP=1 $
      FOR V = N STEP -1 UNTIL 1 DO
      TEMP = TEMP*V $
      X=TRUE $
    L: IF X THEN WRITE ('N FACTORIAL=', TEMP)
       ELSE WRITE ('ILLEGAL VALUE FOR N') $
END
```

Figure 6 An Algol program for calculating N!

Table 7 PROGRAM AREA of Memory M2 showing the ALGOL program

Memory Address	Memory Contents	
	Character Form	Internal Code
512	BEGIN	22252731450000
513	INTEGER	31456324272551
514	N	45000000000000
515	,	73000000000000
516	TEMP	73254447000000
517	,	73000000000000
518	V	65000000000000
519	$	53000000000000
520	BOOLEAN	22464643254145
521	X	67000000000000
522	$	53000000000000
523	READ	51252124000000
524	(	74000000000000
525	N	45000000000000
526	)	34000000000000
527	$	53000000000000
528	IF	31260000000000
529	N	45000000000000
530	LSS	43626200000000
531	Ø	17000000000000
532	THEN	63302545000000
533	BEGIN	22252731450000
534	X	67000000000000
535	=	13000000000000
536	FALSE	26214362250000
537	$	53000000000000
538	L1	43010000000000

Table 7 Continued

Memory Address	Memory Contents	
	Character Form	Internal Code
539	:	15000000000000
540	GOTO	27466346000000
541	L	43000000000000
542	END	25452400000000
543	$	53000000000000
544	TEMP	63254447000000
545	=	13000000000000
546	1	17000000000001
547	$	53000000000000
548	FOR	26465100000000
549	V	65000000000000
550	=	13000000000000
551	N	45000000000000
552	STEP	62632547000000
553	-	40000000000000
554	1	17000000000001
555	UNTIL	64456331430000
556	1	17000000000001
557	DO	24460000000000
558	TEMP	63254447000000
559	=	13000000000000
560	TEMP	63254447000000
561	*	54000000000000
562	V	65000000000000
563	$	53000000000000
564	X	67000000000000
565	=	13000000000000

Table 7 Continued

Memory Address	Memory Contents	
	Character Form	Internal Code
556	TRUE	63516425000000
567	$	53000000000000
568	L	43000000000000
569	:	15000000000000
570	IF	31260000000000
571	X	67000000000000
572	THEN	63302545000000
573	WRITE	66513163250000
574	(	74000000000000
575	'	14000000000000
576	N FACTO	45602621236346
577	RIAL=	51312143136060
578	'	14000000000000
579	,	73000000000000
580	TEMP	63254447000000
581	)	34000000000000
582	ELSE	25436225000000
583	WRITE	66513163250000
584	(	74000000000000
585	'	14000000000000
586	ILLEGAL	31434325272143
587	VALUE	60652143642560
588	FOR N	26465160456060
589	'	14000000000000
590	)	34000000000000
591	$	53000000000000
592	END	25452400000000

Table 8 Initial State of Memory M1

Area name	Contents
BLOCK NUMBER COUNTER	All locations contain zero
DELIMITER STACK	All locations contain zero*
FLAG STACK	All locations contain zero*
INITIAL STACK	All locations contain zero
COUNT STACK	All locations contain zero

*The top elements of these stacks are not kept in memory, see registers D1 and F1 in Table 10.

Table 9 Initial State of Memory M2

Area name	Contents
NAME TABLE	All locations contain zero
PROGRAM AREA	The program to be executed
INPUT QUEUE	The value(s) to be read in as data
OUTPUTSTRING	All locations contain zero
OPERAND LIST	All locations contain zero
LINK TO FOR LIST STACK	All locations contain zero

Table 10 Initial State of the Registers*

Register	Contents (octal)
XCL	37
XD	17
XF	127
XI	153
XG	165
XNT	3777
XPA	0777
XIN	7212
XOUT	7522
XPD	7741
XLF	7765
F1	1
D1	57
CB1	0
ES	0
IT	0

*The registers not listed in this table are not set to a specific state at the start of execution.

3.3.1 Label Search

At the start of execution, the machine increments index register XPA by one (to a value of 512), places its contents into address register AR2, fetches the first program constituent from the PROGRAM AREA and places this constituent in register S for identification. The output of the program constituent decoder identifies the constituent. Upon recognizing the contents of S as the delimiter BEGIN, the machine places these contents on the DELIMITER STACK by transferring them to register D1. It then increments the count for block level zero in the BLOCK NUMBER COUNTER area of memory M1, places the count for block level zero in register CB1, stores the contents of index register XPA in index register XPV, and initializes registers CNT and ES for a label search. CNT is set to zero and ES is set to one. Since register XPA is used in the label search, register XPV provides the means to return to the start of the block when the label search is finished.

During the label search, the block just entered is scanned for colons. The machine scans the block by sequentially fetching program constituents from the PROGRAM AREA and placing these constituents in register S for identification. If a colon is found, a check is made to determine whether it is in an inner block (a block contained within the block just entered) or in the block being scanned. If the colon is in an inner block, it is ignored; otherwise, the label preceding it is fetched from the PROGRAM AREA, hash coded, and stored in the NAME TABLE along with the PROGRAM AREA address of the colon and other information used to identify it. The method of determining whether or not a colon is in an inner block is described in Section 4.4. The hash coding and NAME TABLE entry algorithms are described in detail in Section 4.10. The label search continues until the end of the block is reached. Table 11 shows the contents of the BLOCK NUMBER COUNTERS, the DELIMITER STACK, and the NAME TABLE at the end of the label search. Note that label L1 has not been placed in the NAME

Table 11 Contents of the BLOCK NUMBER COUNTERS, the DELIMITER STACK and the NAME TABLE after the label search of the outer block of the example program.

BLOCK NUMBER COUNTER	
Address	Contents
0	1
1	0
⋮	⋮
15	0

Contents of Address Register XCL: 0

DELIMITER STACK	
Address	Contents
16	@
17	
⋮	
87	

Contents of Address Register XD: 20₈

Top Element of DELIMITER STACK in Register D1: BEGIN

NAME TABLE				
Address*	Contents**			
H(L)	3	0	1	L
H(L)+1		0		1071
⋮			⋮	

Contents of Address Register XNT: H(L)*

---

*The notation H(NAME) represents the hash coded address of NAME.
**See Section 2.1.2.1 and Figure 3 for the description of these contents.

TABLE since it occurs in an inner block.

3.3.2 Declarations

At the end of the label search, the contents of index register XPV are transferred to index register XPA. XPA is then incremented by one (to a value of 513) and the next program constituent is fetched from the PROGRAM AREA and placed in S. The output of the program constituent decoder indicates that this constituent is the declarator INTEGER and the machine places it on the DELIMITER STACK (see Table 12). The machine then scans for the variables declared in the declaration. It increments index register XPA by one and fetches the name N from the PROGRAM AREA. It recognizes N as a name, hash codes it, and places it in the NAME TABLE. It then increments index register XPA by one and fetches the comma following N from the PROGRAM AREA. The comma causes the machine to look for another declared name. The name TEMP is then fetched, hash coded, and placed in the NAME TABLE. The comma following TEMP causes the machine to process V in the same manner. The $ in memory location 519 indicates the end of the integer declaration and causes the machine to erase the declarator INTEGER from the DELIMITER STACK. The machine does this by settingregister D1 to zero. The Boolean declaration which follows the integer declaration is processed in the same manner. Table 13 shows the contents of the DELIMITER STACK and the NAME TABLE at the completion of the Boolean declaration. At this time, index register XPA contains 522.

3.3.3 The READ Statement

Continuing on, the machine increments index register XPA and fetches the input operator READ from the PROGRAM AREA and places this operator on the DELIMITER STACK. It then fetches the left parenthesis in memory location 524. Then, realizing that a READ statement is being processed, the machine skips past the parenthesis by incrementing index register XPA by one and fetching the next program constituent. The machine fetches the name N in memory location 525.

Table 12 Contents of the DELIMITER STACK after the declarator INTEGER is recognized.

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	
⋮	⋮
87	

Contents of Address Register XD: 218

Top Element of DELIMITER STACK in Register D1: INTEGER

Table 13 Contents of the DELIMITER STACK and the NAME TABLE After the Declarations Have Been Processed.

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	
⋮	
87	

Contents of Register D1: 0

Contents of Address Register AD: 20

NAME TABLE				
Address*	Contents**			
H(L)	3	0	1	L
H(L)+1		0		1071
H(N)	2	0	1	N
H(N)+1	77			
H(TEMP)	2	0	1	TEMP
H(TEMP)+1	77			
H(V)	2	0	1	V
H(V)+1	77			
H(X)	1	0	1	X
H(X)+1	77			

Contents of Address Register XT: H(X)

*The notation H(NAME) represents the hash coded address of NAME.

**See Section 2.1.2.1 and Figure 3 for the description of these [illegible].

Recognizing this constituent as a name, the machine examines the current flag, register F1. Since F1 contains one, the machine increments the OPERAND LIST address in index register XPD by one and then uses this address to store the address of N's location in the NAME TABLE in the OPERAND LIST. The machine then fetches the right parenthesis in memory location 526. The right parenthesis causes the machine to examine the top element on the DELIMITER STACK. This element is contained in register D1. Since this element is the operator READ, the machine examines the current flag, register F1. Since F1 contains one, the machine fetches the first element in the INPUT QUEUE (the value 3). It then utilizes the NAME TABLE address of N that was placed in the OPERAND LIST previously and assigns the value 3 to N. Table 14 shows the contents of the DELIMITER STACK, the OPERAND LIST, and the NAME TABLE at this point in the execution. The next program constituent fetched from the PROGRAM AREA is the $ in memory location 527. This $ indicates the end of the READ statement and causes the machine to erase the NAME TABLE address of N from the OPERAND LIST.

3.3.4 The First Conditional Statement

The machine then increments XPA by one (to a value of 528), fetches the delimiter IF, and places it on the DELIMITER STACK. The machine then increments XPA and fetches the variable N. Since F1 contains one, the machine places N's NAME TABLE address in the OPERAND LIST. The relational operator LSS is then fetched and placed on the DELIMITER STACK. The constant zero in memory location 531 is then fetched. This constituent causes the machine to again check the status of the current flag in register F1. Since F1 contains one, the constant zero is placed in the OPERAND LIST. Upon fetching and recognizing the delimiter THEN in memory location 532, the machine executes the relation just scanned and places a new flag on the FLAG STACK. This new flag is placed in register F1 and the old flag is stored in memory M1. The value of the new flag is determined by the outcome of the relation.

Table 14 The Contents of the DELIMITER STACK, the OPERAND LIST, and the NAME TABLE During the Processing of the READ Statement

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	
⋮	
87	

Contents of Address Register XD: 21_8

Top Element of DELIMITER STACK in Register D1: READ

OPERAND LIST	
Address	Contents
4066	H(N)*
4067	
⋮	
4085	

Contents of Address Register XPD: 7742_8

*The notation H(NAME) represents the hash coded address of NAME.

Table 14 Continued

NAME TABLE				
Address*	Contents**			
H(L)	3	0	1	L
H(L)+1		0		1071
H(N)	2	0	1	N
H(N)+1				3
H(TEMP)	2	0	1	TEMP
H(TEMP)+1	77			
H(V)	2	0	1	V
H(V)+1	77			
H(X)	1	0	1	X
H(X)+1	77			

Contents of Address Register XNT: H(N)*

*The notation H(NAME) represents the hash coded address of NAME.
**See Section 2.1.2.1 and Figure 3 for the description of these conten

In executing the relation, the machine fetches the last two elements entered into the OPERAND LIST (The constant zero and the NAME TABLE address of N). When an operand is fetched from the OPERAND LIST, the OPERAND LIST address in index register XPD is decremented by one. This effectively erases the operand from the OPERAND LIST. If either or both operands are NAME TABLE addresses, the machine uses the addresses to fetch the needed values. In this case, the address of N is used to fetch the value of N. The relation is then executed by subtracting zero from 3 (the value of N) and checking the results. Since the value of N is not less than zero, the relation is false and a zero is entered into the OPERAND LIST. The relational operator is then erased from the DELIMITER STACK. The machine then fetches the result of the relation from the OPERAND LIST for examination. Seeing that the result is zero, the machine assigns the new flag the value zero. This new flag is now the current flag and while it remains current, the machine will scan but not execute the program. After raising the new flag, the machine replaces the delimiter IF with the delimiter THEN on the DELIMITER STACK. Table 15 shows the contents of the DELIMITER STACK, the OPERAND LIST, the NAME TABLE, and the FLAG STACK during and after the processing of the relation.

Continuing on, the machine fetches the delimiter BEGIN from memory location 533. After placing this delimiter on the DELIMITER STACK and incrementing the count for block level one in the BLOCK NUMBER COUNTER area of M1 and placing the count for block level one in register CB1, the machine performs a label search. Table 16 shows the contents of the BLOCK NUMBER COUNTERS, the DELIMITER STACK and the NAME TABLE at the end of the label search.

After the label search, the machine increments index register XPA by one and fetches the variable X from memory location 534. Since the value of the current flag is zero, the machine ignores this variable and continues on. The = is fetched and placed on the DELIMITER STACK. The logical value FALSE is

Table 15 Contents of the DELIMITER STACK, the OPERAND LIST, the NAME TABLE and the FLAG STACK While the Relation N LSS 0 is Being Processed.

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	IF
19	
⋮	
87	

Contents of Address Register XD: 22_8

Top Element of DELIMITER STACK in Register D1: LSS

OPERAND LIST	
Address	Contents
4066	H(N)*
4067	0
4068	
⋮	
4085	

Contents of Address Register XPD: 7743_8

FLAG STACK	
Address	Contents
88	
⋮	
107	

Contents of Address Register XF: 127_8

Top Element of FLAG STACK in Register F1: 1

*The notation H(NAME) represents the hash coded address of NAME

Table 15 Continued

NAME TABLE				
Address*	Contents**			
H(L)	3	0	1	L
H(L)+1		0		1071
H(N)	2	0	1	N
H(N)+1				3
H(TEMP)	2	0	1	TEMP
H(TEMP)+1	77			
H(V)	2	0	1	V
H(V)+1	77			
H(X)	1	0	1	X
H(X)+1	77			

Contents of Address Register XNT: H(N)*

(a) prior to execution of the relation

*The notation H(NAME) represents the hash coded address of NAME.
**See Section 2.1.2.1 and Figure 3 for the description of these contents.

Table 15 Continued

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	
⋮	
87	

Contents of Address Register XD: 21_8

Top Element of DELIMITER STACK in Register D1: IF

OPERAND LIST	
Address	Contents
4066	0
4067	
⋮	
4085	

Contents of Address Register XPD: 7742_8

FLAG STACK	
Address	Contents
88	
⋮	
107	

Contents of Address Register XF: 127_8

Top Element of FLAG STACK in Register F1: 1

Table 15 Continued

NAME TABLE				
Address*	Contents**			
H(L)	3	0	1	L
H(L)+1		0		1071
H(N)	2	0	1	N
H(N)+1				3
H(TEMP)	2	0	1	TEMP
H(TEMP)+1	77			
H(V)	2	0	1	V
H(V)+1	77			
H(X)	1	0	1	X
H(X)+1	77			

Contents of Address Register XNT: H(N)*

(b) just after the relation is executed.

*The notation H(NAME) represents the hash coded address of NAME.

**See Section 2.1.2.1 and Figure 3 for the description of these contents.

Table 15 Continued

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
⋮	
87	

Contents of Address Register XD: 21_8
Top Element of DELIMITER STACK in Register D1: THEN

OPERAND LIST	
Address	Contents
4066	
⋮	
4085	

Contents of Address Register XPD: 7741_8

FLAG STACK	
Address	Contents
88	1
89	
⋮	
107	

Contents of Address Register XF: 130_8
Contents of Register D1: 0

Table 15 Continued

NAME TABLE				
Address*	Contents**			
H(L)	3	0	1	L
H(L)+1		0		1071
H(N)	2	0	1	N
H(N)+1				3
H(TEMP)	2	0	1	TEMP
H(TEMP)+1	77			
H(V)	2	0	1	V
H(V)+1	77			
H(X)	1	0	1	X
H(X)+1	77			

Contents of Address Register XNT: H(N)*

(c) after the new flag is placed on the FLAG STACK

*The notation H(NAME) represents the hash coded address of NAME.
**See Section 2.1.2.1 and Figure 3 for the description of these contents.

Table 16 The Contents of the BLOCK NUMBER COUNTERS, the DELIMITER STACK and the NAME TABLE After the Label Search of the Inner Block of the Example Program

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	THEN
19	
⋮	
87	

Contents of Address Register XD: 22_8

Top Element of DELIMITER STACK in Register D1: BEGIN

BLOCK NUMBER COUNTERS	
Address	Contents
0	1
1	1
2	0
⋮	⋮
15	0

Contents of Address Register XCL: 1

Table 16 Continued

NAME TABLE				
Address*	Contents**			
H(L)	3	0	1	L
H(L)+1		0		1071
H(N)	2	0	1	N
H(N)+1				3
H(TEMP)	2	0	1	TEMP
H(TEMP)+1	77			
H(V)	2	0	1	V
H(V)+1	77			
H(X)	1	0	1	X
H(X)+1	77			
H(L1)	3	1	1	L1
H(L1)+1		1		1033

Contents of Address Register XNT: H(L1)*

*The notation H(NAME) represents the hash coded address of NAME.

**See Section 2.1.2.1 and Figure 3 for the description of these contents.

is fetched and ignored because the current flag contains zero. The $ is then fetched. It causes the = to be erased from the DELIMITER STACK. The machine then scans past label L1 and the colon that follows it. It then fetches the delimiter GOTO from memory location 540 and places this delimiter on the DELIMITER STACK. This delimiter is immediately erased from the DELIMITER STACK when the label L is fetched and the current flag is again examined and found to be zero.

The machine then fetches the delimiter END from memory location 542. This delimiter signifies the end of a block and causes the machine to reset register CB1 with the count of block level zero and reset the DELIMITER STACK to its contents prior to entering the block. After the delimiter END is processed, the contents of the BLOCK NUMBER COUNTERS, and the DELIMITER STACK are as shown in Table 17. Note that label L1 remains in the NAME TABLE but it is now inaccessible to the program. The machine next fetches the $ in memory location 543. This $ indicates the end of the conditional statement and causes the machine to erase the delimiter THEN and the current flag from the DELIMITER STACK and the FLAG STACK respectively. The machine does this by setting registers D1 and F1 to zero. The original flag which was assigned a value of one at the start of execution is then fetched from memory M1 and stored in register F1. This flag is again the current flag. Since this flag has a value of one, the program will again be executed as well as scanned. The contents of the DELIMITER STACK and the FLAG STACK at this point are as shown in Table 18.

3.3.5 Initial Value Assigned to TEMP

The machine next fetches the variable TEMP from memory location 544. Since the current flag now has a value of one, the machine places the address of TEMP's location in the NAME TABLE in the OPERAND LIST. It then fetches the = in memory location 545 and places it on the DELIMITER STACK. It then fetches the constant one from memory location 546 and examines the value of the current

Table 17 The Contents of the BLOCK NUMBER Area and the DELIMITER STACK Upon Exiting the Inner Block

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	THEN
19	
⋮	
87	

Contents of Address Register XD: 228

Contents of Register D1: 0

BLOCK NUMBER COUNTER	
Address	Contents
0	1
1	1
2	0
⋮	⋮
15	0

Contents of Address Register XCL: 0

Table 18 Contents of the DELIMITER STACK and the FLAG STACK at the Completion of the Conditional Statement.

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	
⋮	
87	

Contents of Address Register XD: 21_8

Contents of Register D1: 0

FLAG STACK	
Address	Contents
88	
⋮	
107	

Contents of Address Register XF: 127_8

flag in register F1. Since F1 contains one, the machine places the constant one in the OPERAND LIST. It then fetches the $ following the one. The $ causes the machine to perform the assignment indicated by the operator =. This operation is performed on the two elements last entered into the OPERAND LIST. The variable TEMP is set equal to the value 1. The = is then removed from the DELIMITER STACK and the NAME TABLE address of TEMP and the value 1 are removed from the OPERAND LIST. (See Table 19).

3.3.6 The Iteration

At this point the machine fetches the delimiter FOR from memory location 548. Recognizing this delimiter as the beginning of an iteration, the machine sets the variable IT to one and places a one on the COUNT STACK. It then fetches the controlled variable of the iteration from memory location 549. This is the variable V. The machine then checks the contents of the current flag. Since F1 contains one, the machine places the NAME TABLE address of V into the OPERAND LIST. It then fetches the = following the controlled variable and again checks the contents of the current flag. Since F1 still contains one, the PROGRAM AREA address of the = is placed on the LINK TO FORLIST STACK and a one is placed in the INITIAL STACK. Table 20 depicts the contents of these areas of memory at this time.

3.3.6.1 Process the For List Element

The machine then fetches the variable N from memory location 551. Since F1 contains one, the machine places N's NAME TABLE address in the OPERAND LIST. It then fetches the delimiter STEP from memory location 552 and places this delimiter on the DELIMITER STACK (Table 21). The machine then examines the value of the top element on the INITIAL STACK. Since this element contains one and the value of the current flag is one, the NAME TABLE address of N is replaced by the value of N in the OPERAND LIST. This value is then assigned to the controlled

Table 19 Contents of the DELIMITER STACK, the OPERAND LIST, and the NAME TABLE During the Processing of the Assignment TEMP=1$

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	
19	
⋮	
87	

Contents of Address Register XD: 21_8

Top Element of DELIMITER STACK in Register D1: =

OPERAND LIST	
Address	Contents
4066	H(TEMP)*
4067	1
4068	
⋮	
4085	

Contents of Address Registor XPD: 7743_8

*The notation H(NAME) represents the hash coded address of NAME.

Table 19 Continued

NAME TABLE				
Address*	Contents**			
H(L)	3	0	1	L
H(L)+1		0		1071
H(N)	2	0	1	N
H(N)+1				3
H(TEMP)	2	0	1	TEMP
H(TEMP)+1	77			
H(V)	2	0	1	V
H(V)+1	77			
H(X)	1	0	1	X
H(X)+1	77			
H(L1)	3	1	1	L1
H(L1)+1		1		1033

Contents of Address Register XNT: H(TEMP)*

(a) just prior to executing the assignment

* The notation H(NAME) represents the hash coded address of NAME.

**See Section 2.1.2.1 and Figure 3 for the description of these contents.

Table 19 Continued

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	
⋮	
87	

Contents of Address Register XD: 21_8

Contents of Register Dl: 0

OPERAND LIST	
Address	Contents
4066	
⋮	
4085	

Contents of Address Register XPD: 7741_8

Table 19 Continued

NAME TABLE				
Address*	Contents**			
H(L)	3	0	1	L
H(L)+1		0		1071
H(M)	2	0	1	M
H(M)+1				3
H(TEMP)	2	0	1	TEMP
H(TEMP)+1				1
H(V)	2	0	1	V
H(V)+1	77			
H(X)	1	0	1	X
H(X)+1	77			
H(L1)	3	1	1	L1
H(L1)+1		1		1033

Contents of Address Register XNT: H(TEMP)*

(b) after the assignement is executed.

*The notation H(NAME) represents the hash coded address of NAME.

**See Section 2.1.2.1 and Figure 3 for the description of these contents.

Table 20 The Contents of the DELIMITER STACK, the OPERAND LIST, the INITIAL STACK, the COUNT STACK, and the LINK TO FORLIST STACK at the Start of the Iteration in the Example Program

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	
⋮	
87	

Contents of Address Register XD: 21_8

Top Element of DELIMITER STACK in Register D: FOR

OPERAND LIST	
Address	Contents
4066	H(V)*
4067	
⋮	
4085	

Contents of Address Register XPD: 7742_8

INITIAL STACK	
Address	Contents
108	1
109	
⋮	
117	

Contents of Address Register XI: 154_8

*The notation H(NAME) represents the hash coded address of NAME.

Table 20 Continued

COUNT STACK	
Address	Contents
118	1
119	
⋮	
127	

Contents of Address Register XC: 166_8

LINK TO FORLIST STACK	
Address	Contents
4086	1046
4087	
⋮	
4095	

Contents of Address Register XLF: 7766_8

Table 21 The Contents of the DELIMITER STACK and the OPERAND LIST During Execution of the Iteration

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	FOR
19	
⋮	
87	

Contents of Address Register XD: 22_8

Top Element of DELIMITER STACK in Register D1: STEP

OPERAND LIST	
Address	Contents
4066	H(V)*
4067	H(N)*
4068	
⋮	
4085	

Contents of Address Register XPD: 7743_8

*The notation H(NAME) represents the hash coded address of NAME.

variable V in the NAME TABLE. The machine then fetches the unary minus from memory location 553. This delimiter causes the machine to place the special symbol 44_8 on the DELIMITER STACK. Since F1 contains one, the constant one following the unary minus is placed in the OPERAND LIST. The machine then fetches the delimiter UNTIL.

Upon identifying this delimiter, the machine converts the value 1 in the OPERAND LIST to -1 and erases the special symbol 44_8 from the DELIMITER STACK. The machine then replaces the delimiter STEP with the delimiter UNTIL on the DELIMITER STACK. Upon fetching the constant one which follows the delimiter UNTIL from memory location 556, the machine again checks the contents of the current flag. It finds that F1 contains one. The machine therefore places the one in the OPERAND LIST. Table 22 shows the contents of the DELIMITER STACK, the OPERAND LIST and the NAME TABLE at this time.

Upon fetching the delimiter DO from memory location 557, the machine realizes that the entire for list element has been scanned and removes the delimiter UNTIL from the DELIMITER STACK. It then examines the contents of the top element on the INITIAL STACK. Since this element contains one, the machine resets it to zero. It then determines the value of the expression C* sign B - A; where A, B and C are the last three elements placed in the OPERAND LIST respectively. The values of these elements are the values of the three arithmetic expressions which occur in the STEP-UNTIL for list element. If these elements are NAME TABLE addresses, the values of the variables are fetched from the NAME TABLE before the expression is evaluated. If the value of this expression is less than or equal to zero, the statement contained within the iteration is to be executed; otherwise, the execution of the iteration is complete for this for list element. At this point in the example, the resultant value of the expression is -2; thus, the statement contained within the iteration is to be executed. The machine scans to the beginning of the statement by sequentially fetching

Table 22 The Contents of the DELIMITER STACK, the OPERAND LIST, and the NAME TABLE During Evaluation of the For List Element

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	FOR
19	
⋮	
87	

Contents of Address Register XD: 22_8

Top Element of DELIMITER STACK in Register D1: UNTIL

OPERAND LIST	
Address	Contents
4066	H(V)*
4067	3
4068	-1
4069	1
4070	
⋮	
4085	

Contents of Address Register XPD: 7745_8

*The notation H(NAME) represents the hash coded address of NAME.

Table 22 Continued

NAME TABLE				
Address*	Contents**			
H(L)	3	0	1	L
H(L)+1		0		1071
H(N)	2	0	1	N
H(N)+1				3
H(TEMP)	2	0	1	TEMP
H(TEMP)+1				1
H(V)	2	0	1	V
H(V)+1				3
H(X)	1	0	1	X
H(X)+1	77			
H(L1)	3	1	1	L1
H(L1)+1		1		1033

Contents of Address Register XNT: H(V)*

*The notation H(NAME) represents the hash coded address of NAME.

**See Section 2.1.2.1 and Figure 3 for the description of these contents.

program constituents from the PROGRAM AREA until it identifies the delimiter DO. The machine places this delimiter on the DELIMITER STACK. It also erases elements A, B, and C from the OPERAND LIST. The contents of the DELIMITER STACK and the OPERAND LIST at this time are as depicted in Table 23.

3.3.6.2 Process the Statement Contained in the Iteration

The machine fetches the variable TEMP from memory location 558. It then checks the contents of the current flag in register F1. Since F1 contains one, the machine places the NAME TABLE address of TEMP in the OPERAND LIST. It then fetches the = from memory location 559 and places it on the DELIMITER STACK. The second occurrence of TEMP in memory location 560 also results in its NAME TABLE address being placed in the OPERAND LIST. the machine then fetches the * from memory location 561 and places it on the DELIMITER STACK. The NAME TABLE address of V is then placed in the OPERAND LIST. Table 24 depicts the contents of the DELIMITER STACK and the OPERAND LIST at this point in the execution of the program.

The machine then fetches the $ from memory location 563. The $ indicates the end of the statement and causes the machine to check the DELIMITER STACK to determine if the operations which occur in the statement have been completed. The machine finds the operator * on the DELIMITER STACK and fetches the NAME TABLE addresses of TEMP and V from the OPERAND LIST and uses these addresses to fetch the values of TEMP and V from the NAME TABLE. The machine then performs the multiplication operation on these values and places the resultant product in the OPERAND LIST. It then erases the operator * from the DELIMITER STACK (Table 24(b)).

Again checking the DELIMITER STACK, the machine finds the assignment operator =. It then fetches the NAME TABLE address of TEMP and the product of TEMP*V from the OPERAND LIST and uses the address to assign the product as the new value of TEMP. It then places the product back in the OPERAND LIST and erases the = from the DELIMITER STACK. Thus the contents of the DELIMITER STACK, the

Table 23 Contents of the DELIMITER STACK and the OPERAND LIST After the For List Element has been Tested

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	FOR
19	
20	
⋮	
87	

Contents of Address Register XD: 238

Top Element of DELIMITER STACK in Register D1: DO

OPERAND LIST	
Address	Contents
4066	H(V)*
4067	
⋮	
4085	

Contents of Address Register XPD: 7742

*The notation H(NAME) represents the hash coded address of NAME.

Table 24 The Contents of the DELIMITER STACK, the OPERAND LIST, and the NAME TABLE During Execution of the Statement Contained Within the Iteration

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	FOR
19	DO
20	=
21	
⋮	
87	

Contents of Address Register XD: 24_8

Top Element of DELIMITER STACK in Register D1: *

OPERAND LIST	
Address	Contents
4066	H(V)*
4067	H(TEMP)*
4068	H(TEMP)*
4069	H(V)*
4070	
⋮	
4085	

Contents of Address Register XNT: H(V)*

*The notation H(NAME) represents the hash coded address of NAME.

Table 24 Continued

NAME TABLE				
Address*	Contents**			
H(L)	3	0	1	L
H(L)+1		0		1071
H(N)	2	0	1	N
H(N)+1				3
H(TEMP)	2	0	1	TEMP
H(TEMP)+1				1
H(V)	2	0	1	V
H(V)+1				3
H(X)	1	0	1	X
H(X)+1	77			
H(L1)	3	1	1	L1
H(L1)+1		1		1033

Contents of Address Register XNT: H(V)*

(a) just after scanning the statement

*The notation H(NAME) represents the hash coded address of NAME.

**See Section 2.1.2.1 and Figure 3 for the description of these contents.

Table 24 Continued

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	FOR
19	DO
20	=
21	
⋮	
87	

Contents of Address Register XD: 24_8

Contents of Register D1: 0

OPERAND LIST	
Address	Contents
4066	H(V)*
4067	H(TEMP)*
4068	3
4069	
⋮	
4085	

Contents of Address Register XPD: 7744_8

*The notation H(NAME) represents the hash coded address of NAME.

Table 24 Continued

NAME TABLE				
Address*	Contents**			
H(L)	3	0	1	L
H(L)+1		0		1071
H(N)	2	0	1	N
H(N)+1				3
H(TEMP)	2	0	1	TEMP
H(TEMP)+1				1
H(V)	2	0	1	V
H(V)+1				3
H(X)	1	0	1	X
H(X)+1	77			
H(L1)	3	1	1	L1
H(L1)+1		1		1033

Contents of Address Register XNT: H(V)*

(b) after performing the multiplication operation

*The notation H(NAME) represents the hash coded address of NAME.

**See Section 2.1.2.1 and Figure 3 for the description of these contents.

Table 24 Continued

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	FOR
19	DO
20	
⋮	
87	

Contents of Address Register XD: 23_8

Contents of Register D1: 0

OPERAND LIST	
Address	Contents
4066	H(V)*
4067	3
4068	
⋮	
4085	

Contents of Address Register XPD: 7743_8

*The notation H(NAME) represents the hash coded address of NAME.

Table 24 Continued

NAME TABLE				
Address*	Contents**			
H(L)	3	0	1	L
H(L)+1		0		1071
H(N)	2	0	1	N
H(N)+1				3
H(TEMP)	2	0	1	TEMP
H(TEMP)+1				3
H(V)	2	0	1	V
H(V)+1				3
H(X)	1	0	1	X
H(X)+1	77			
H(L1)	3	1	1	L1
H(L1)+1		1		1033

Contents of Address Register XNT: H(TEMP)*

(c) after performing the assignment operation

*The notation H(NAME) represents the hash coded address of NAME

**See Section 2.1.2.1 and Figure 3 for the description of these contents.

OPERAND LIST, and the NAME TABLE at this point are as shown in Table 24(c).

3.3.6.3 Repeat the Iteration

The machine again checks the DELIMITER STACK and finds the delimiter DO. It erases this delimiter from the stack and checks the stack again. This time it finds the delimiter FOR and realizes the iteration statement has been processed. It then erases all elements in the OPERAND LIST which were entered after the NAME TABLE address of the iteration's controlled variable (Table 25).

The machine now must return to the iteration's for list to see if the iteration statement should be executed again. It does this by fetching the top element on the LINK TO FORLIST STACK and placing it in index register XPA.

Once the machine returns to the iteration's for list, it examines the top element on the COUNT STACK. The value of this element indicates which for list element is currently being used to determine the value of the iteration's controlled variable. It also causes the machine to scan the for list until it reaches the indicated for list element. In this instance, the top element on the COUNT STACK has the value one; therefore, the first for list element (in this case the only for list element) is currently being used and no scanning of the for list is required to reach it.

The machine now starts to process the for list element for the second time. Upon fetching the variable N from memory location 551, the machine examines the contents of the current flag and finds that F1 contains one. The machine enters the NAME TABLE address of N into the OPERAND LIST. It then places the delimiter STEP on the DELIMITER STACK. The machine then examines the top element on the INITIAL STACK. Since this element now contains zero, the value of the iteration's controlled variable replaces the NAME TABLE address of N in the OPERAND LIST. The -1 is handled as before and the delimiter UNTIL again replaces the delimiter STEP on the DELIMITER STACK. The constant one following

Table 25: Contents of the DELIMITER STACK and the OPERAND LIST as the Machine Returns to the Iteration's For List

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	
19	
⋮	
87	

Contents of Address Register XD: 21_8

Top Element of DELIMITER STACK in Register D1: FOR

OPERAND LIST	
Address	Contents
4066	H(V)*
4067	
⋮	
4085	

Contents of Address Register XPD: 7742_8

*The notation H(NAME) represents the hash coded address of NAME.

the delimiter UNTIL is again placed in the OPERAND LIST. The delimiter DO again informs the machine that the entire for list element has been scanned; the machine therefore removes the reserved name UNTIL from the DELIMITER STACK and examines the contents of the top element on the INITIAL STACK. Since this element is now zero, the machine adds the values in the B and C elements of the OPERAND LIST (where B and C are defined as before) and stores the result in the C element. It then assigns the result, via the NAME TABLE address in the OPERAND LIST, to the iteration's controlled variable; thus the controlled variable is "stepped" -1. The value of the test expression C* sign B - A is again calculated. This time the result is -1 and the machine executes the statement contained within the iteration as before; thereby changing the value of TEMP to six.

At the completion of execution of the statement, the machine again returns to the start of the iteration's for list and processes the for list element. Since the value of the top element on the INITIAL STACK is still zero, the value of the controlled variable is reduced to one and the result of the test expression is calculated to be zero; therefore, the statement contained within the iteration is executed a third time. Since V now has a value of one, the value of TEMP does not change when the multiplication is performed.

Again returning to the for list, the machine reduces the value of the controlled variable to zero. This value causes the test expression to yield a result of +1. This result indicates the iteration has been executed the number of times specified by the for list element. The machine therefore increments the top element of the COUNT STACK by one, resets the top element of the INITIAL STACK to one, and scans the for list for the start of the next for list element. Upon identifying the reserved name DO the machine realizes there are no more for list elements; hence no further execution of the iteration is required. It therefore erases the delimiter FOR from the DELIMITER

STACK, erases the NAME TABLE address of the iteration's controlled variable V from the OPERAND LIST, erases the top element on the INITIAL STACK, the COUNT STACK, and the LINK TO FORLIST STACK, resets register IT to zero and scans to the end of the iteration. At this point, the contents of these areas of memory are as shown in Table 26.

3.3.7 The Boolean Assignment Statement

Continuing on, the machine fetches the variable X from memory location 564. After finding that the current flag contains one, the machine enters the NAME TABLE address of X into the OPERAND LIST. It then fetches the = from memory location 565 and places it on the DELIMITER STACK. It then fetches the logical value TRUE. This value causes the machine to place a one in the OPERAND LIST. The $ following the logical value causes the machine to execute the indicated assignment and to reset the DELIMITER STACK and the OPERAND LIST to their contents before the assignment statement was entered.

3.3.8 The Second Conditional Statement

Scanning on, the machine by-passes the label L and the colon which follows it. It then fetches the delimiter IF from memory location 570 and places it on the DELIMITER STACK. The variable X is fetched and causes the machine to again examine the contents of the current flag. Since F1 contains one, the machine places the NAME TABLE address of X in the OPERAND LIST. It then fetches the delimiter THEN. This delimiter causes the machine to replace the NAME TABLE address of X with the value of X. The machine then examines this value to determine what value to assign the new flag that is generated for this conditional statement. Since X's value is one, the new flag is assigned the value one. This new flag is the current flag. After the new flag is set, the value of X is erased from the OPERAND LIST and the delimiter IF is replaced by the delimiter THEN on the DELIMITER STACK (Table 27).

Table 26 The Contents of the Areas of Memory After The Iteration is Completely Processed.

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	
⋮	
87	

Contents of Address Register XD: 21_8

Contents of Register D1: 0

OPERAND LIST	
Address	Contents
4066	
⋮	
4085	

Contents of Address Register XPD: 7741_8

COUNT STACK	
Address	Contents
118	
⋮	
127	

Contents of Address Register XD: 165_8

Table 26 Continued

NAME TABLE				
Address*	Contents**			
H(L)	3	0	1	L
H(L)+1		0		1071
H(N)	2	0	1	N
H(N)+1				3
H(TEMP)	2	0	1	TEMP
H(TEMP)+1				6
H(V)	2	0	1	V
H(V)+1				0
H(X)	1	0	1	X
H(X)+1	77			
H(L1)	3	1	1	L1
H(L1)+1		1		1033

Contents of Address Register XNT: H(TEMP)*

INITIAL STACK	
Address	Contents
108	
⋮	
117	

Contents of Address Register XI: 153_8

LINK TO FORLIST STACK	
Address	Contents
4086	
⋮	
4095	

Contents of Address Register XLF: 7765_8

* The notation L(NAME) represents the hash coded address of NAME.

**See Section 2.1.2.1 and Figure 3 for the description of these contents.

Table 27 The Contents of the DELIMITER STACK and the FLAG STACK After Evaluating X and Placing a New Flag for the Conditional Statement.

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	
19	
⋮	
87	

Contents of Address Register XD: 21_8

Top Element of DELIMITER STACK in Register D1: THEN

FLAG STACK	
Address	Contents
88	1
89	
⋮	
107	

Contents of Address Register XF: 130_8

Top Element of FLAG STACK in Register F1: 1

The machine then fetches the output operator WRITE and places it on the DELIMITER STACK. Realizing a WRITE statement is being processed, the machine scans past the left parenthesis and fetches the apostrophe from memory location 575. The machine places the apostrophe on the DELIMITER STACK. This apostrophe indicates that the beginning of a string of characters which is to be printed out has been reached and causes the machine to scan in search of a second apostrophe. Since the current flag has a value of one, the contents of the memory locations that are fetched while the machine is scanning for the second apostrophe are placed in the OUTPUTSTRING area of memory M2. When the second apostrophe is found, the contents of the OUTPUTSTRING area are printed on the line printer and the first apostrophe is removed from the DELIMITER STACK. (The output algorithm is not implemented in this report). The comma following the output string indicates that the WRITE statement has not been completely processed; therefore, the machine continues scanning the program with the output operator WRITE still on the DELIMITER STACK. Upon fetching the variable TEMP from the memory location 580, the machine examines the contents of the current flag in register F1. Since F1 contains one, the machine places the NAME TABLE address of TEMP in the OPERAND LIST. The right parenthesis is then fetched from memory location 581. This delimiter indicates to the machine that there are no more variables or strings of characters in the output list. The machine then fetches the value of TEMP via its NAME TABLE address in the OPERAND LIST and converts this value to BCD form for output. The NAME TABLE address of TEMP and the output operator WRITE are then removed from the OPERAND LIST and the DELIMITER STACK respectively (Table 28).

Upon fetching the delimiter ELSE, the machine changes the value of the current flag from one to zero and replaces the delimiter THEN with ELSE on the DELIMITER STACK. With the current flag now zero, the machine scans to

Table 28 The Contents of the DELIMITER STACK and the OPERAND LIST During Execution of the WRITE Statement Which Outputs the Value of N!

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	THEN
19	
⋮	
87	

Contents of Address Register XD: 22_8

Top Element of DELIMITER STACK in Register D1: WRITE

OPERAND LIST	
Address	Contents
4066	H(TEMP)*
4067	
⋮	
4085	

Contents of Address Register XPD: 7742_8

(a) just before printing the value of TEMP

*The notation H(NAME) represents the hash coded address of NAME.

Table 28 Continued

DELIMITER STACK	
Address	Contents
16	@
17	BEGIN
18	THEN
19	
⋮	
87	

Contents of Address Register XD: 22_8

Contents of Register D1: 0

OPERAND LIST	
Address	Contents
4066	
⋮	
4085	

Contents of Address Register XPD: 7741_8

(b) after the WRITE statement is completely processed

the end of statement symbol ($) in memory location 591, without executing the WRITE statement following the ELSE. Upon reaching the end of statement symbol the machine erases the delimiter ELSE from the DELIMITER STACK and the current flag from the FLAG STACK. The initial flag again becomes the current flag.

3.3.9 Exiting the Program

Finally, the interpretor fetches the reserved name END from memory location 592. After resetting register CB1, the machine realizes that it had been operating at block level zero and thus the outermost block (the program) has been completely processed. The machine then halts.

4. Sequence Charts

The machine operations for executing a program are now put into sequence charts. The operations are organized into the 24 sequences listed in Table 29. The machine beings execution in the initial point sequence. During execution, register S and the top element of the DELIMITER STACK, register D1, determine the machine transfers from one sequence to another. There is a control flip-flop associated with each sequence. When a sequence is completed, its control flipflop is reset to zero. While their control flipflops contain a value of zero, the sequences constantly examine them and wait for their contents to become one. In addition to the sequences, Table 29 also lists the sequence control flipflops and the sequences to which the machine can transfer from each sequence. The sequences are described in detail in the following paragraphs.

4.1 Initial Point Sequence

The machine starts executing an ALGOL program in this sequence. It reenters this sequence whenever a delimiter is placed on the DELIMITER STACK and the program constituent following the delimiter could be one of several types. In this sequence the machine fetches program constituents from the PROGRAM AREA, identifies them, and transfers to the appropriate sequence for processing them. Some constituents require little processing and are processed in this sequence. These are the delimiters GOTO, READ, WRITE, IF, DO, the left parenthesis, the comma, the unary + and -, and character strings which are to be printed out. This sequence is shown in Figures 7A and 7B.

When this sequence is entered, the machine first sets the NAME TABLE activity control flipflop, ES, to zero. It then fetches the next program constituent (at the start of execution, the first program constituent) from the PROGRAM AREA. It does this by incrementing the current PROGRAM AREA address in index register XPA by one, transferring this address to

Table 29 The Machine Sequence for Executing a Program

No.	Sequence	Control Flipflop	Sequence to which Machine Can Transfer
1.	Initial Point Sequence	IP	Output String Initialization Sequence Number Processing Sequence Block Entry Sequence Block Exit Sequence Delimiter TRUE Sequence Delimiter FALSE Sequence Iteration Initialization Sequence Read/write Execution Sequence Declaration Initialization Sequence NAME TABLE Activity Sequence Unconditional Statement Sequence Error Sequence
2.	Output String Initialization Sequence	AP	Initial Point Sequence Read/write Execution Sequence
3.	Number Processing Sequence	NMB	Factor Sequence
4.	Block Entry Sequence	BGN	Initial Point Sequence NAME TABLE Activity Sequence
5.	Block Exit Sequence	ND	
6.	Delimiter TRUE Sequence	TRU	Logical Expression Sequence
7.	Delimiter FALSE Sequence	FLS	Logical Expression Sequence

Table 29 Continued

No.	Sequence	Control Flipflop	Sequence to which Machine Can Transfer
8.	Iteration Initialization Sequence	FR	NAME TABLE Activity Sequence
9.	Declaration Initialization Sequence	DCL	NAME TABLE Activity Sequence
10.	NAME TABLE Activity Sequence	NME	Initial Point Sequence Block Entry Sequence Label Processing Sequence Variable Processing Sequence Unconditional Statement Sequence Error Sequence
11.	Label Processing Sequence	LBL	Initial Point Sequence Unconditional Statement Sequence Error Sequence
12.	Variable Processing Sequence	VR	Initial Point Sequence Factor Sequence Logical Expression Sequence
13.	Factor Sequence	FT	Term Sequence Error Sequence
14.	Term Sequence	TM	Initial Point Sequence Sum Sequence Error Sequence

Table 29 Continued

No.	Sequence	Control Flipflop	Sequence to which Machine Can Transfer
15.	Sum Sequence	SM	Initial Point Sequence Factor Sequence Arithmetic Expression Sequence Logical Expression Sequence Error Sequence
16.	Arithmetic Expression Sequence	AE	Initial Point Sequence Assignment Sequence Iteration Control Sequence Error Sequence
17.	Logical Expression Sequence	LE	Initial Point Sequence Assignment Sequence Iteration Control Sequence Error Sequence
18.	Assignment Sequence	AS	Unconditional Statement Sequence Error Sequence
19.	Unconditional Statement Sequence	UN	Initial Point Sequence End of Statement Sequence Error Sequence

Table 29 Continued

No.	Sequence	Control Flipflop	Sequence to which Machine Can Transfer
20.	End of Statement Sequence	ST	Program Body Sequence Iteration Control Sequence Error Sequence
21.	Program Body Sequence	BD	Initial Point Sequence Block Exit Sequence Error Sequence
22.	Iteration Control Sequence	FP	Initial Point Sequence Block Exit Sequence End of Statement Sequence
23.	Read/write Execution Sequence	RP	Output String Initialization Sequence Unconditional Statement Sequence Error Sequence
24.	Error Sequence	ER	

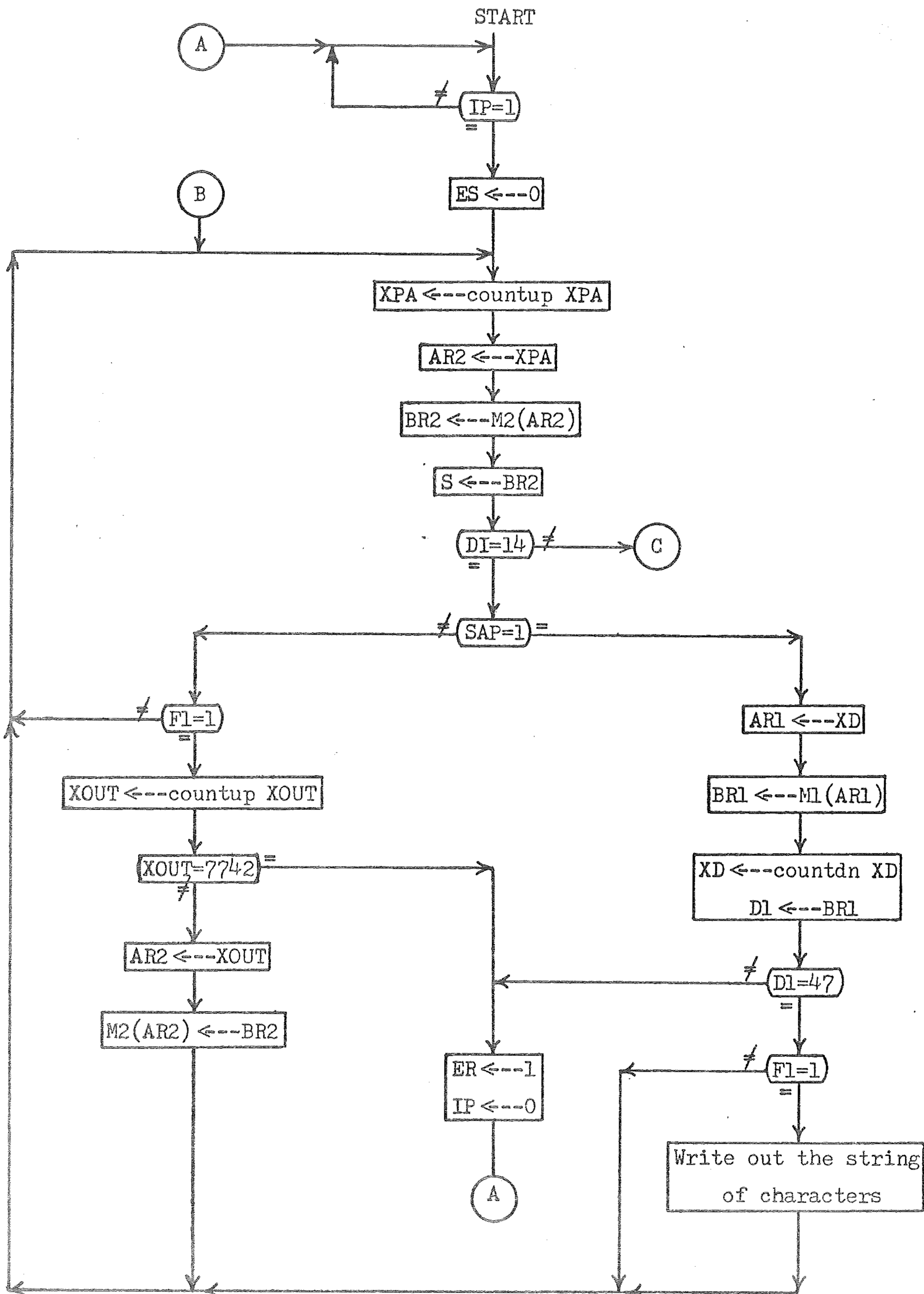


Figure 7a The Initial Point Sequence Chart

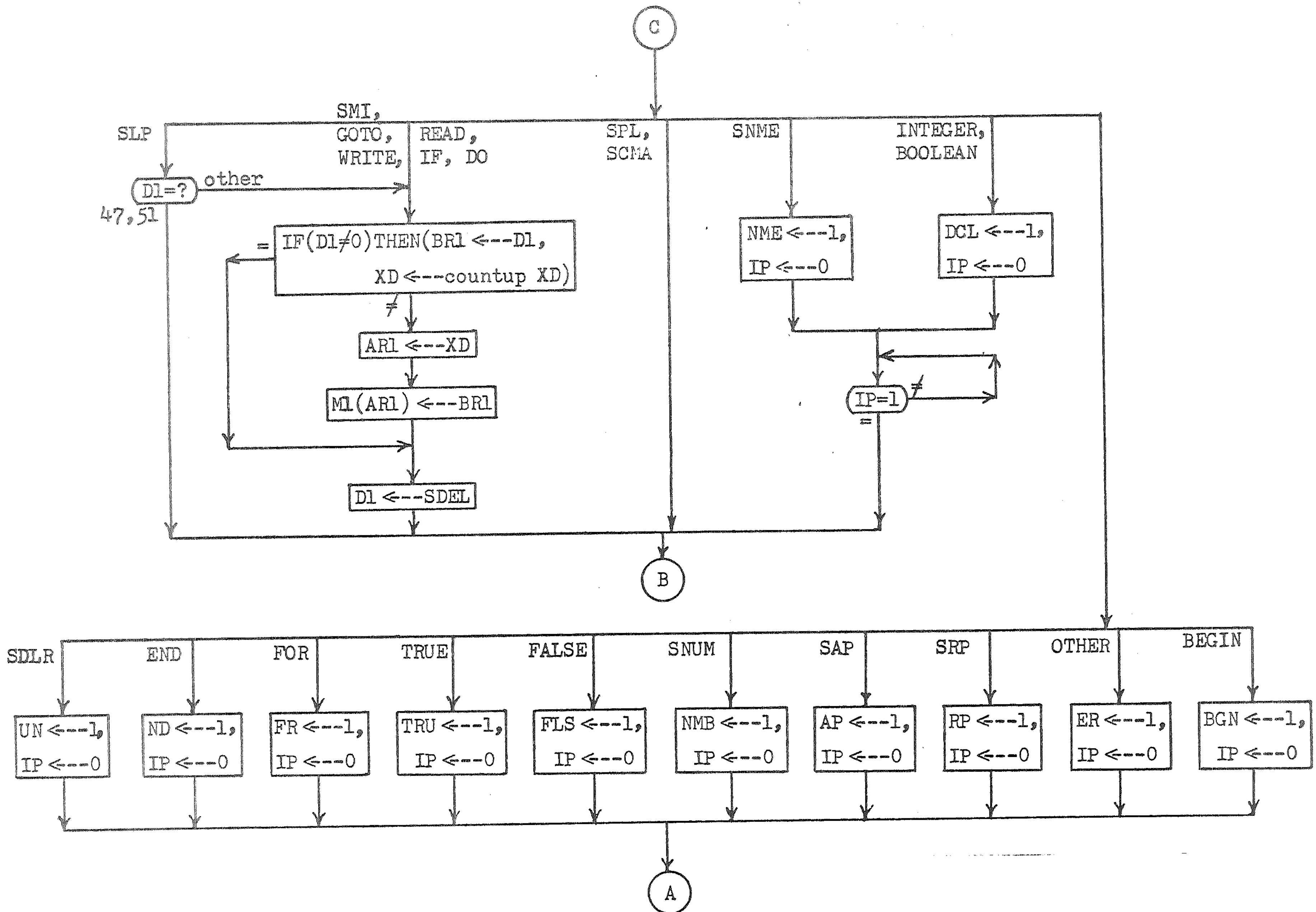


Figure 7b The Initial Point Sequence Chart

to address register AR2, and finally transferring the contents of the memory location to register S through buffer register BR2. The machine then examines the contents of register D1. If register D1 contains an apostrophe ($D1=14_8$) and the constituent just fetched from the PROGRAM AREA is also an apostrophe (SAP is true), then an output character string has been completely scanned. Therefore, the machine erases the apostrophe from the DELIMITER STACK by fetching the next element on the stack from memory M1 and placing it in register D1. Index register XD contains the DELIMITER STACK address. This address is decremented by one after the delimiter is fetched. Register D1 is then examined for the delimiter WRITE. If D1 does not contain this delimiter, the machine transfers to the error sequence; otherwise, it examines the current flag, register F1. If F1 contains a one, the character string has been assembled in the OUTPUTSTRING area of memory M2 and is printed out. The machine then fetches the next program constituent from the PROGRAM AREA. If F1 contains zero, the machine immediately fetches the next program constituent.

If register D1 contains an apostrophe, ($D1=14_8$), but the program constituent fetched from the PROGRAM AREA is not an apostrophe (SAP is false), the contents of register S (also register BR2 at this point) are part of an output string. If F1 contains a one, this part of the output string is placed in the OUTPUTSTRING area of memory M2. The machine then fetches the next program constituent from the PROGRAM AREA. If F1 contains a zero, the machine immediately fetches the next program constituent. In placing the contents of register BR2 in the OUTPUTSTRING area the machine first increments the current address of this area (located in index register XOUT) by one and then, after determining that the length of the string is within the allowed limits (1000 characters), uses the address to transfer the contents to memory M2. The machine then fetches the next program constituent from the PROGRAM AREA. If the output string exceeds the maximum allowable length, the machine transfers to the error sequence.

If register D1 does not contain an apostrophe ($D1 \neq 14_8$), the course of action taken by the machine is determined by the program constituent decoder. If this decoder identifies the contents of S as one of the delimiters GOTO, READ, WRITE, IF, DO, or unary minus, the machine places the delimiter on the DELIMITER STACK and fetches the next program constituent from the PROGRAM AREA. As shown in Figure 7B, in placing a delimiter on the DELIMITER STACK, the machine first determines whether or not register D1 contains zero. If it does, the machine transfers the delimiter to D1. If D1 does not contain zero, the machine first stores it contents in memory M1 and then transfers the delimiter to D1.

If the decoder identifies the contents of register S as a left parenthesis and register D1 does not contain either of the delimiters READ or WRITE, the left parenthesis is placed on the DELIMITER STACK; otherwise it is passed over. In either case, the machine then fetches the next program constituent from the PROGRAM AREA. The delimiters unary + and the comma are always passed over.

If the program constituent decoder identifies the contents of register S as any other program constituent than those mentioned above, the machine transfers to the appropriate sequence as listed in Table 29.

4.2 Output String Initialization Sequence

This sequence is shown in Figure 8. The machine enters this sequence from the initial point sequence whenever the contents of register S are identified as an apostrophe (SAP is true) and the top element of the DELIMITER STACK is not an apostrophe ($D1 \neq 14_8$). Execution of this sequence initializes the machine for assembling an output string in the OUTPUTSTRING area of memory M2. Upon entering this sequence, the machine first determines whether or not there are any variables in the OPERAND LIST whose values are to be printed out. It does this by checking the current OPERAND LIST address in index register XPD and the contents of the iteration control register IT. If the current OPERAND LIST address indicates that the OPERAND LIST is empty ($XPD = 7741_8$) or if register

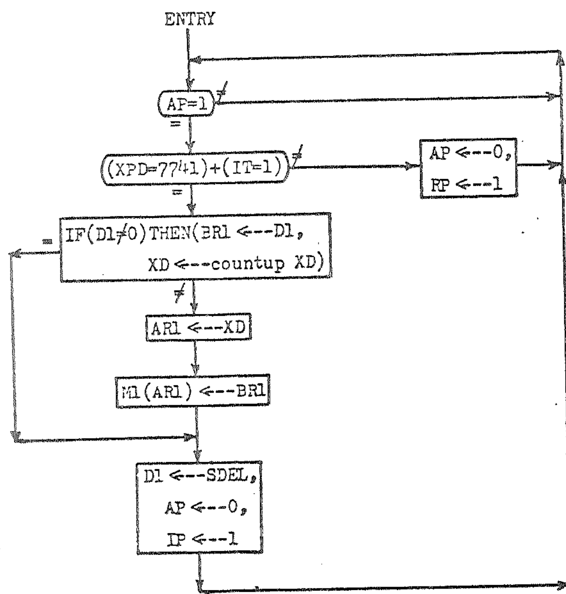


Figure 8 The Output String Initialization Sequence Chart

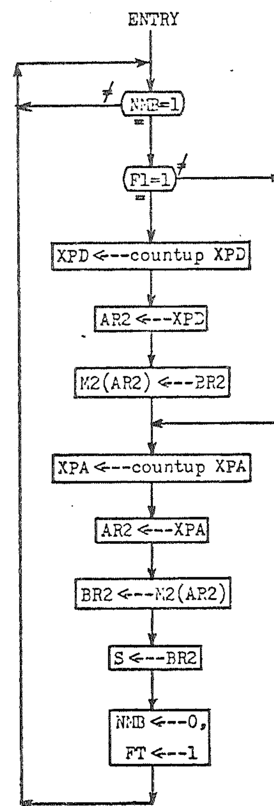


Figure 9 The Number Processing Sequence Chart

IT indicates that the variables in the OPERAND LIST are iteration controlled variables (IT=1), there are no values to be printed and the apostrophe in register S is placed on the DELIMITER STACK. The machine then transfers to the initial point sequence to fetch the next program constituent.

If there are variables in the OPERAND LIST whose values are to be printed out, the machine transfers to the read/write execution sequence. It returns to this sequence after printing out the values. Upon returning, it again examines registers XPD and IT. This time it finds there are no values to be printed out and places the apostrophe in S on the DELIMITER STACK. It then transfers to the initial point sequence.

4.3 Number Processing Sequence

This sequence is shown in Figure 9. The machine enters this sequence from the initial point sequence whenever the contents of register S are identified as a number (SNUM IS TRUE). Upon entering this sequence, the machine first examines the current flag, register F1. If F1 contains a one, the machine increments the current OPERAND LIST address in index register XPD by one and then transfers the number into the memory location of memory M2 specified by the new address. If F1 contains a zero, the number is not entered into the OPERAND LIST. Regardless of the value of the current flag, the machine fetches the next program constituent from the PROGRAM AREA, stores it in register S, and then transfers to the factor sequence.

4.4 Block Entry Sequence

This sequence is shown in Figures 10A and 10B. The machine enters this sequence from the initial point sequence whenever the contents of register S are identified as the delimiter BEGIN. Upon entering this sequence the machine first places the delimiter BEGIN on the DELIMITER STACK. It then increments the current address of the BLOCK NUMBER COUNTER area by one. This address is located

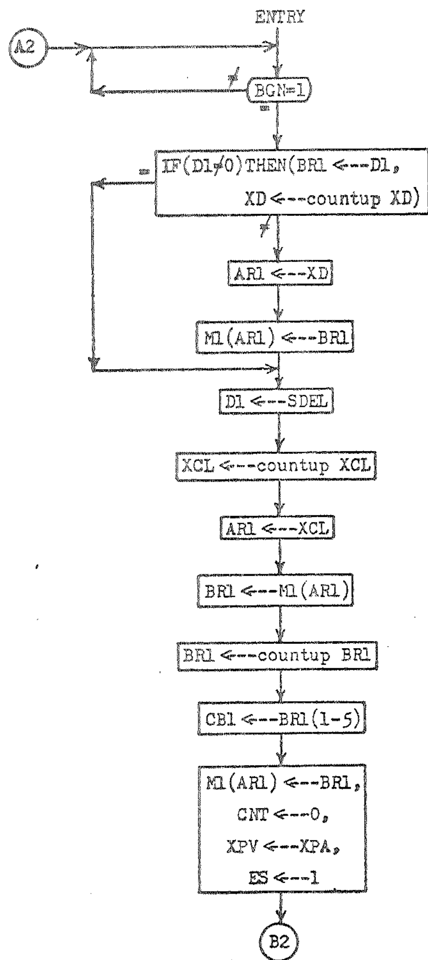


Figure 10a The Block Entry Sequence Chart

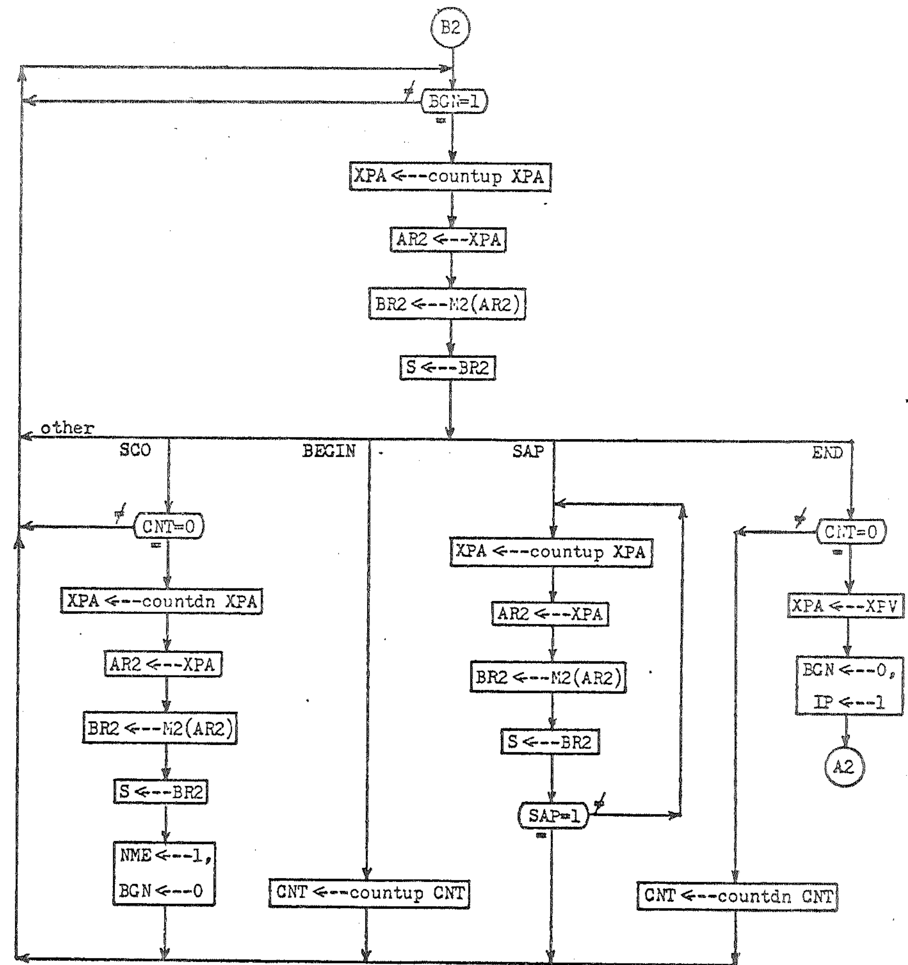


Figure 10b The Block Entry Sequence Chart

in index register XCL. It then fetches the block count for the block level just entered from memory M1 (see Section 2.1.1.1). It then updates this block count by incrementing buffer register BR1 by one. The updated block count is then transferred to the current block count register, CB1, and also into its location in memory M1. The machine then initializes for a label search. It stores the current PROGRAM AREA address in index register XPV, sets the counter CNT to zero, and sets the NAME TABLE activity control register, ES, to one. By saving the current PROGRAM AREA address in register XPV, the machine is able to return to the beginning of the block at the end of the label search. The label search is now executed.

During the label search the machine scans the block just entered in search of colons (:). The program constituents are sequentially fetched from the PROGRAM AREA, placed in register S, and identified. If a colon is found (SCO is true), the machine determines whether it is in an inner block (a block contained within the block just entered) or in the block being scanned. If the colon is in an inner block it is ignored; otherwise, the machine fetches the label preceding the colon, places it in register S, and transfers to the NAME TABLE activity sequence. Since register ES was set to one at the start of the label search, the NAME TABLE activity sequence enters the label into the NAME TABLE. Meanwhile, this sequence is constantly examining its control flipflop BGN while waiting for its contents to become one. After the label has been entered into the NAME TABLE, the machine transfers back to this sequence (BGN←--1) and the label search continues.

The machine determines whether or not a label is in an inner block by the following method. It is noted that at the start of a label search the counter CNT is set to zero. While the label search is being performed, if the delimiter BEGIN is fetched into register S, the contents of counter CNT are incremented by one. If the delimiter END is fetched into register S, the con-

tents of counter CNT are examined and if they are greater than zero, they are decremented by one. Each time a colon is fetched into register S the contents of counter CNT are examined. If CNT has a value greater than zero, the colon occurs in an inner block; otherwise, it occurs in the block just entered. When the delimiter END is fetched and the value in CNT is zero, the machine knows the entire block has been scanned and terminates the label search. At the end of a label search, the machine transfers the contents of index register XPV to index register XPA and then transfers to the initial point sequence.

Whenever an apostrophe is fetched into register S (SAP is true) during a label search, the machine scans without reacting to the constituents until another apostrophe is found. Therefore, any colons which occur in an output character string go unrecognized.

4.5 Block Exit Sequence

This sequence is shown in Figure 11. The machine enters this sequence when the contents of register S are identified as the delimiter END. In executing this sequence the machine first decrements the current address of the BLOCK NUMBER COUNTERS area of memory M1 by one. It then restores the DELIMITER STACK to its contents prior to entering the just completed block. It does this by sequentially fetching elements from memory M1 and transfering them to register D1 until it finds the first occurrence of the delimiter BEGIN ($D1=22_8$). It then erases this delimiter by setting register D1 to zero. After restoring the DELIMITER STACK, the machine determines whether or not it has scanned the entire program; that is, has it just completed the outermost block (block level zero). It examines the contents of index register XCL for a value of 31_8; the value assigned to XCL at the start of execution. If XCL contains 31_8 the entire program has been scanned and the machine halts by setting the run/stop control flipflop, G, to zero. It also turns the light FINI on. If the program has not been completely scanned, the machine increments the current PROGRAM AREA address

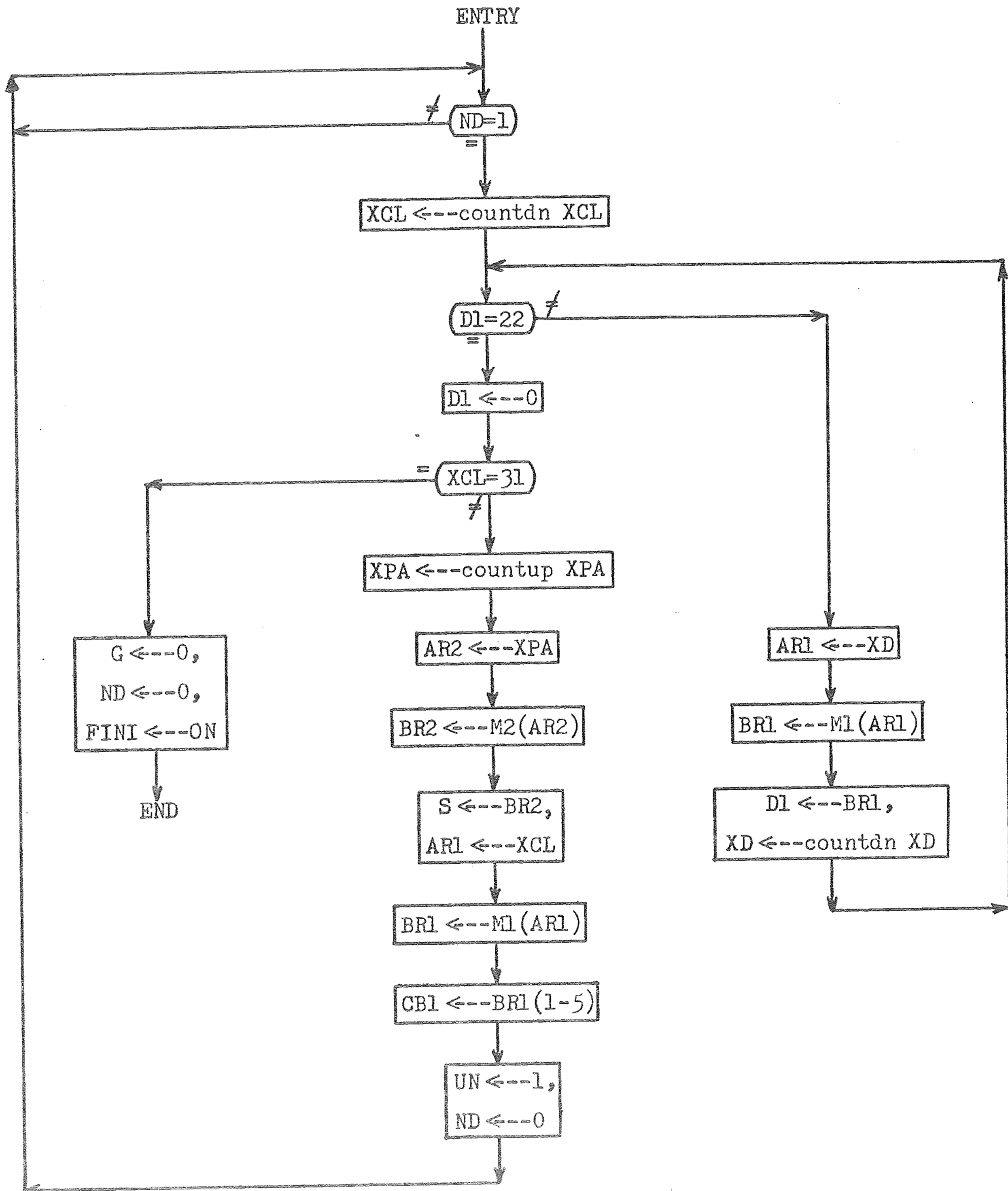


Figure 11 The Block Exit Sequence Chart

by one, fetches the next program constituent, and stores it in register S. It then fetches the block number count for the block level just entered from memory M1 and transfers it to the current block count register, CB1. It then transfers to the unconditional statement sequence.

4.6 Delimiter TRUE Sequence

This sequence is shown in Figure 12. The machine enters this sequence from the initial point sequence whenever the contents of register S are identified as the delimiter TRUE. Upon entering this sequence, the machine first examines the current flag, register F1. If F1 contains a one, the machine increments the current OPERAND LIST address in index register XPD by one and then uses this address to enter a value of one into the OPERAND LIST. It then transfers to the logical expression sequence. If F1 contains a zero, the machine transfers immediately to the logical expression sequence.

4.7 Delimiter FALSE Sequence

This sequence is shown in Figure 13. The machine enters this sequence from the initial point sequence whenever the contents of register S are identified as the delimiter FALSE. Upon entering this sequence, the machine first examines the current flag, register F1. If F1 contains a one, the machine increments the current OPERAND LIST address in index register XPD by one and then uses this address to enter a value of zero into the OPERAND LIST. It then transfers to the logical expression sequence. If F1 contains a zero, the machine transfers immediately to the logical expression sequence.

4.8 Iteration Initialization Sequence

This sequence is shown in Figure 14. The machine enters this sequence from the initial point sequence when the contents of register S are identified as the delimiter FOR. Execution of this sequence initializes the machine for executing the iteration statement just entered. The machine first sets the iteration control register IT to one. It then places the delimiter FOR on

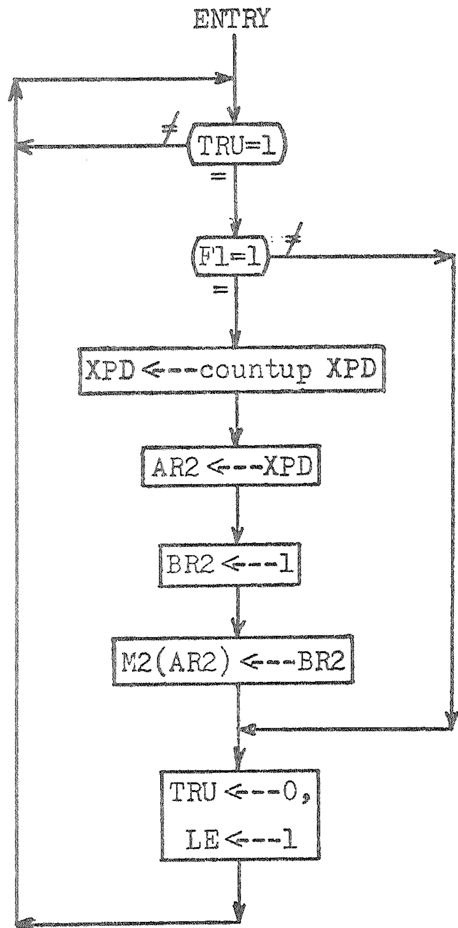


Figure 12 Sequence Chart for Processing the Logical Value TRUE

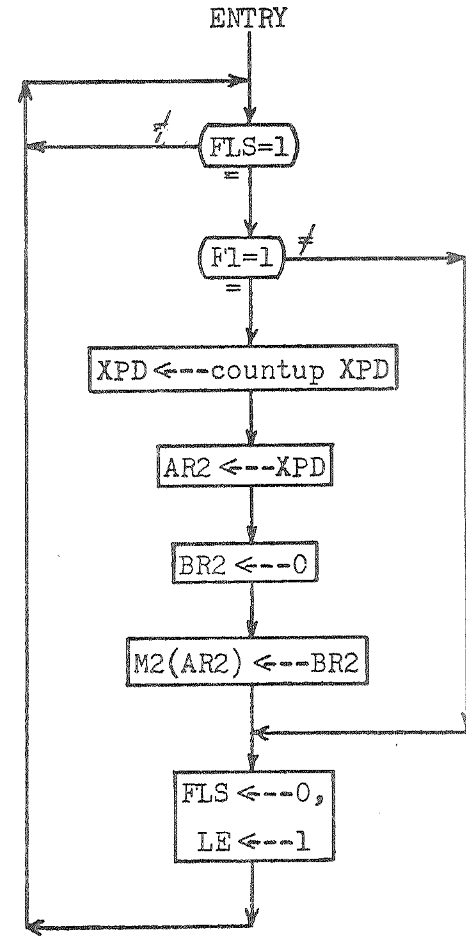


Figure 13 Sequence Chart for Processing the Logical Value FALSE

the DELIMITER STACK. It then increments the current addresses of the COUNT STACK, the INITIAL STACK, and the LINK TO FORLIST STACK by one. These addresses are located in index registers XC, XI, and XLF respectively. It then places the value one on the COUNT STACK. It then increments the current PROGRAM AREA address by one, fetches the next program constituent from memory M2, and stores it in register S. This constituent should be the iteration's controlled variable so the machine transfers to the NAME TABLE activity sequence. At this point the NAME TABLE activity control register, ES, contains a zero. Therefore, a search of the NAME TABLE will be conducted for the location of the iteration's controlled variable.

4.9 Declaration Initialization Sequence

This sequence is shown in Figure 15. The machine enters this sequence from the initial point sequence whenever the contents of register S are identified as one of the declarators BOOLEAN or INTEGER. Upon entering this sequence the machine first sets the NAME TABLE activity control flipflop, ES, to one; thus, the names following the declarator will be entered into the NAME TABLE when the NAME TABLE activity sequence is executed. After setting ES, the machine places the declarator in register S onto the DELIMITER STACK. It then increments the current PROGRAM AREA address by one, fetches the next program constituent from memory M2, and stores it in register S. According to the ALGOL syntax, this constituent should be the first name of the list of variables declared in the declaration. Therefore, the machine transfers to the NAME TABLE activity sequence to enter the name into the NAME TABLE.

4.10 NAME TABLE Activity Sequence

This sequence is shown in Figures 16A-E. The machine enters this sequence when the contents of register S are identified as an unreserved name.

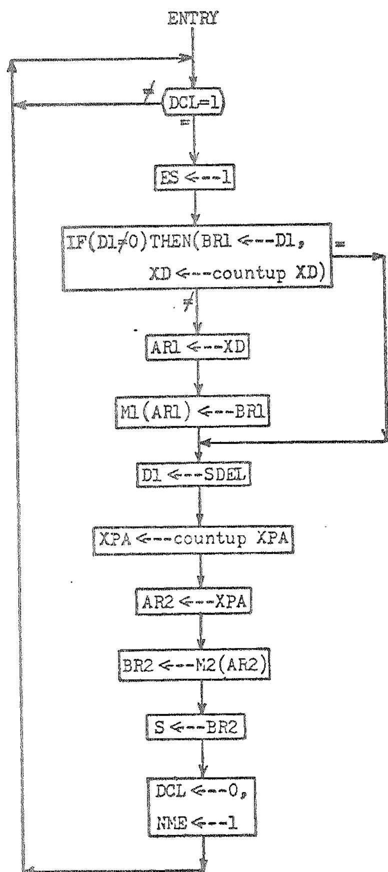


Figure 15 The Declaration Initialization Sequence Chart

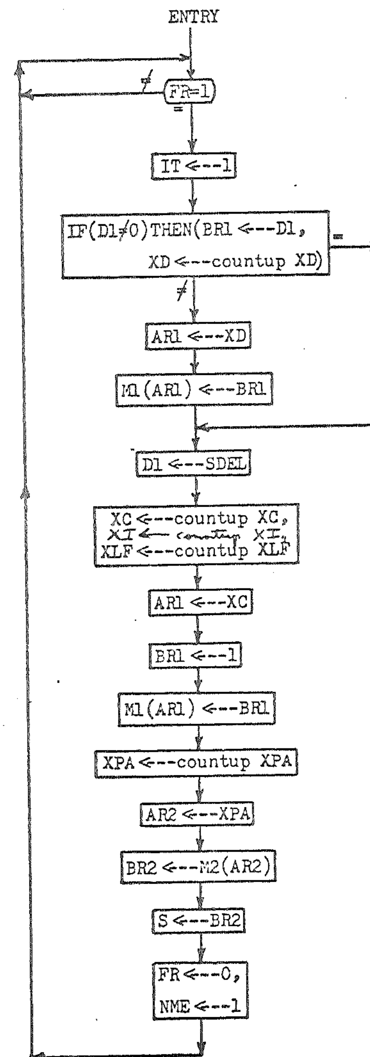


Figure 14 The Iteration Initialization Sequence Chart

Upon entering this sequence, the machine first hash codes the unreserved name to obtain a NAME TABLE address (5). It then examines the control register ES. If register ES contains a one, the name is entered into the NAME TABLE; otherwise, the NAME TABLE is searched for the location of the name. The following sections explain the hash coding, the NAME TABLE entry, and the NAME TABLE search algorithms.

4.10.1 Hash Coding a Name

As shown in Figure 16A, the machine first sets registers CSC and XNT to zero, and sets register R to one. It also transfers the name from register S to register HC. Register CSC is a counter which counts the number of circular leftshifts of register HC in bytes. A byte is seven bits. Index register XNT receives the hash coded address. Register R is used to calculate a new address if a collision occurs. Register HC holds the name while it is being hashed.

After initializing the registers, the machine executes the hash coding loop. In executing this loop the machine performs three operations. First it increments index register XNT by the value contained in subregister HC(0-9). Second, it circularly leftshifts the contents of register HC one byte. Finally, it increments register CSC by one. The machine terminates this loop when the contents of register CSC reach six. At that time register HC will have been circularly shifted 42 bits so that it will again contain the name in its proper form. Register XNT will contain the final hash coded address. This address is transferred to address register AR2. It is noted that the length of the hash coded address is one bit less than the length of register AR2, and that the address is transferred left adjusted to register AR2 with the right-most bit of register AR2 being set to zero. This action allows the machine to effectively hash code only the even addresses and thus reserve the odd addresses for the second word of each node.

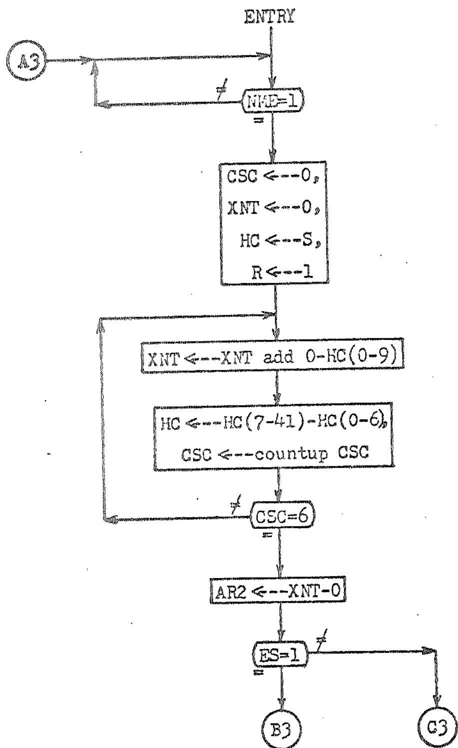


Figure 16a NAME TABLE Activity Sequence Chart

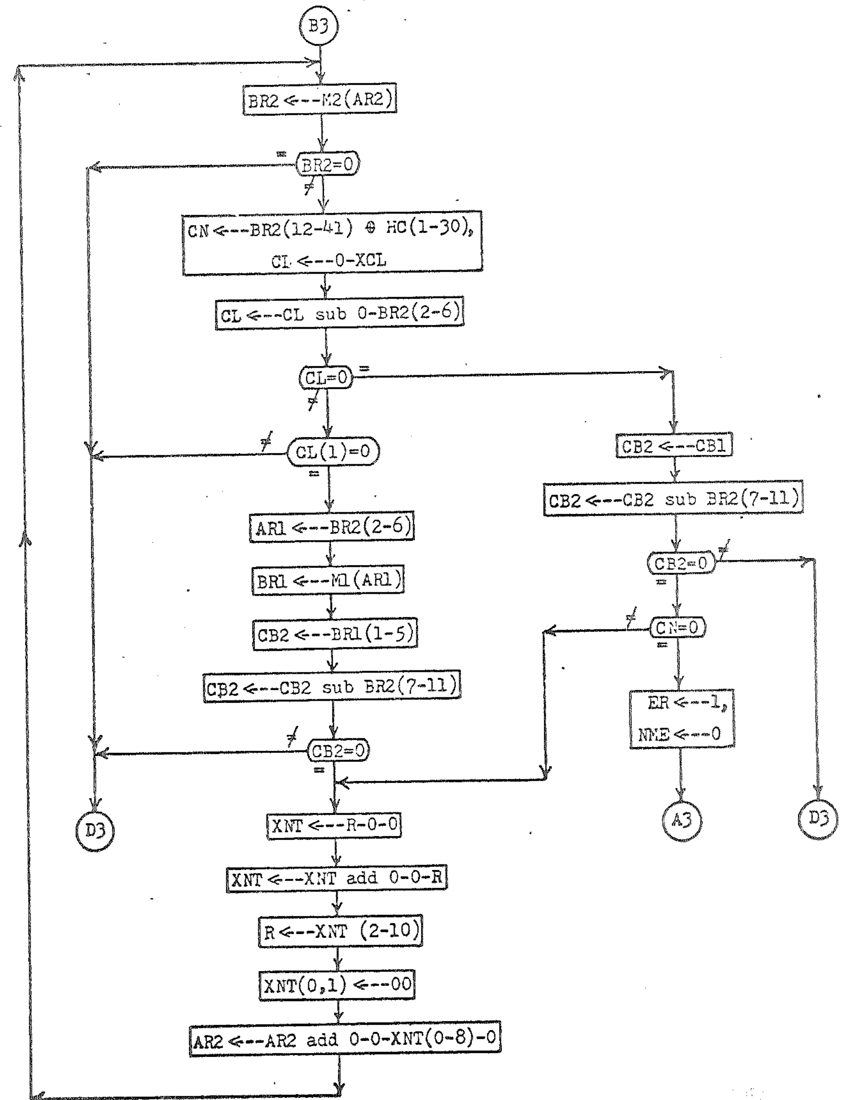


Figure 16 b NAME TABLE Activity Sequence Chart

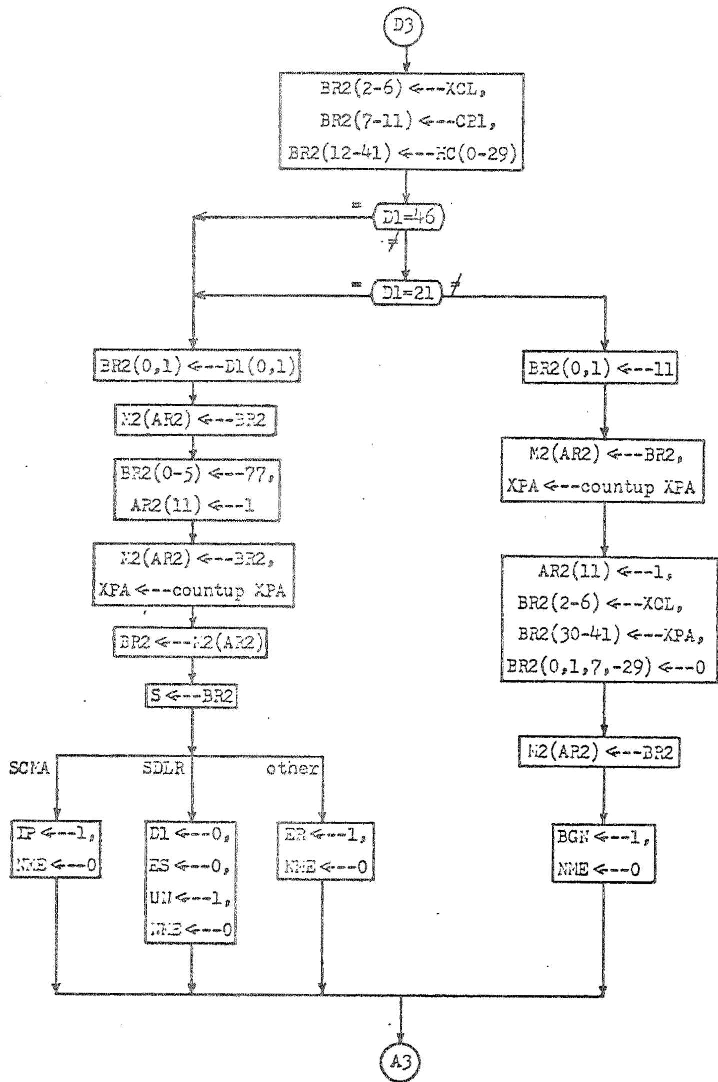


Figure 16c NAME TABLE Activity Sequence Chart

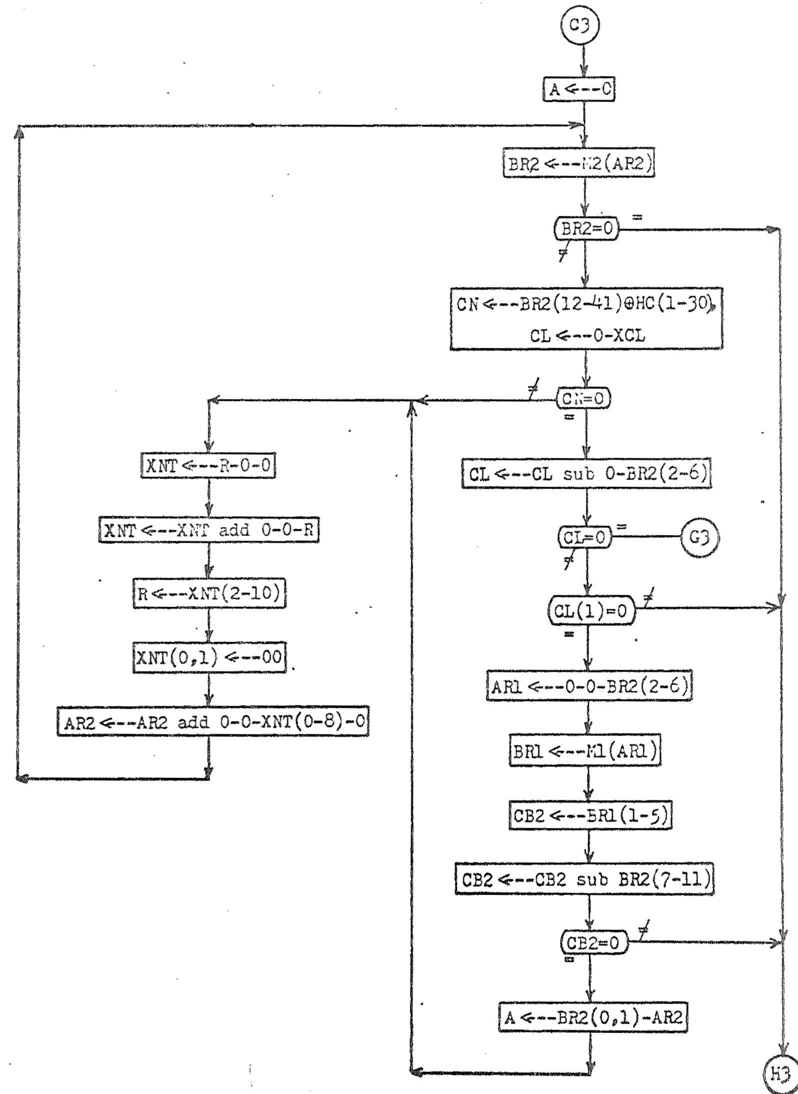


Figure 16d NAME TABLE Activity Sequence Chart

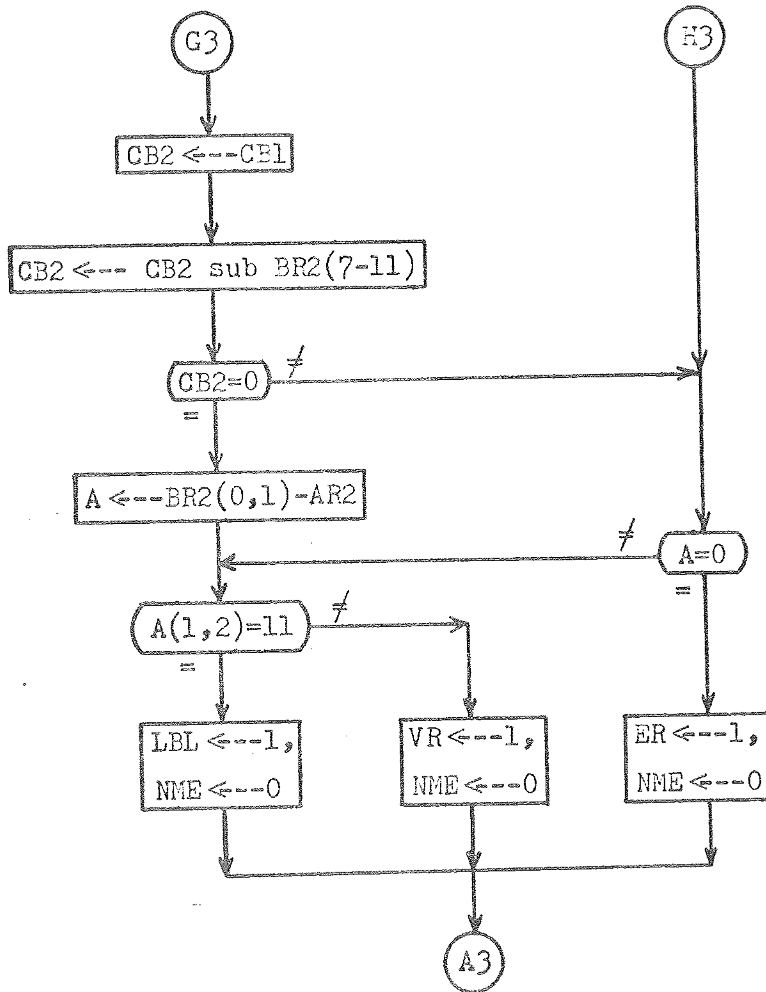


Figure 16e NAME TABLE Activity Sequence Chart

4.10.2 NAME TABLE Entry

As shown in Figure 16B, if register ES contains a one, the contents of the memory location specified by the hash coded address are transferred to buffer register BR2 for examination. If register BR2 contains a zero, the memory location is empty and the machine enters the name into it. This procedure is described in Section 4.10.2.2. If register BR2 contains a non-zero value, the memory location is already occupied and the machine must determine whether the name occupying the location was declared in a block still being processed or in a completed block. If the block is still being processed, the name is still active and a new address must be calculated to enter the name just hashed into the table. If the block has been completed, the name is no longer accessible to the program and can be written over. Referring to Section 2.1.2.1, it is seen that when a name is entered into the NAME TABLE, some additional information is entered with it. This information includes the block level and the block number which were current when the name was entered. The machine compares this information with the current block level and block number to determine whether or not a name is still active.

Referring again to Figure 16B, when the memory location specified by the hash coded address is occupied, the machine compares the name occupying the location with the name being entered and stores the results of the comparison in register CN. This result is examined later in the sequence if the block level and block number information indicates that the two names were declared in the same block. At this time, the machine also transfers the current block level (located in index register XCL) to register CL. It then decrements register CL by the contents of subregister BR2(2-6). This is the block level of the name in the NAME TABLE. The machine then examines register CL. If register CL contains zero, the two block levels are the same and the machine must compare the block numbers to finally determine whether or not the name

is still active. To perform the block number comparison, the machine transfers the current block number (located in register CB1) to register CB2. It then decrements register CB2 by the contents of subregister BR2(7-11). This is the block number of the name in the NAME TABLE. It then examines register CB2. If CB2 contains a non-zero value, the name in the NAME TABLE was declared in a block already completed and is no longer accessible to the program. Therefore, the machine enters the name that was just hashed into this location (see Section 4.10.2.2). Referring to Figure 2, if execution of the program has proceeded to point Q, any names declared in block B would fit this condition.

If register CB2 contains zero, the name in the NAME TABLE and the name just hashed have been declared in the same block. Therefore, the machine examines register CN to determine whether or not they are the same name. If they are, an error condition exists since a name cannot be declared more than once in a block. Therefore, the machine transfers to the error sequence. If the two names are not the same, a collision exists and a new address must be calculated to enter the name just hashed. The procedure for doing this is described in Section 4.10.2.1.

If register CL contains a non-zero value, the current block level and the block level of the name in the NAME TABLE differ. Therefore, the machine examines the left-most bit of register CL, CL(1), to determine whether the name in the NAME TABLE was declared at a block level that is higher or lower than the current block level. If CL(1) contains a one, the name in the NAME TABLE was declared at a higher block level and is no longer accesssible to the program. Therefore, the machine enters the name that was just hashed into this location (see Section 4.10.2.2). Referring to Figure 2, if execution of the program has proceeded to point P, any names declared in block E or block F fit this condition.

If CL(1) contains a zero, the name was declared at a lower block level and the machine must compare the block numbers to finally determine whether or not the name is still active. To perform the comparison, the machine first transfers the block level of the name in the NAME TABLE from subregister BR2(2-6) to address register AR1. It then fetches the block number count for this block level from the BLOCK NUMBER COUNTER area of memory M1. It stores this count in register CB2. It then decrements register CB2 by the contents of subregister BR2(7-11); the block number of the name in the NAME TABLE. The machine then examines the content of register CB2. If CB2 contains zero, the name in the NAME TABLE is still active and the machine must calculate a new address to enter the name just hashed (see Section 4.10.2.1). Referring to Figure 2, if execution of the program has proceeded to point P, any names declared in block C would fit this condition.

If register CB2 contains a non-zero value, the name in the NAME TABLE is no longer accessible and the machine enters the name that was just hashed into this location (see Section 4.10.2.2). Referring again to Figure 2, if execution of the program has proceeded to point P, any names declared in block B would fit this condition.

4.10.2.1 Handling Collisions

The collision handling algorithm implemented in this machine is known as the random probing method (5). In this algorithm a pseudorandom number generator is called to provide an offset which is added to the collision address to obtain a new address. The offset is generated in three steps: the previous offset is multiplied by five, the low order n+2 bits of the product are saved while the remaining bits are ignored, and finally, the remaining bits of the product are divided by four. In the machine the offset is stored in register R. It is noted that register R is set to one each time this sequence is entered; thus the same offsets are generated each time a collision occurs. As

shown in Figure 16B, the machine multiplies the offset by four by transferring it left adjusted by two bits to index register XNT. It then effects the required multiplication by five by incrementing register XNT by the contents of register R. It then stores the low order n+2 bits of the product in register R. Finally, it increments register AR2 by the contents of register XNT such that register AR2 is effectively incremented by the product divided by four.

4.10.2.2 Entering a Name

After hash coding a name and determining that the location specified by the hashed address is either empty or occupied by a name no longer required by the program, the machine enters the hashed name into the NAME TABLE. As shown in Figure 16C, the machine transfers the current block level (located in index register XCL), the current block count (located in register CB1), and the first five characters of the name (located in subregister HC(0-29) to subregisters BR2(2-6), BR2(7-11), and BR2(12-41) respectively. It then examines the top element on the DELIMITER STACK, register D1. If register D1 contains a declarator, the name is a variable and the machine transfers the type of the variable from subregister D1(0,1) to subregister BR2(0,1). It then transfers the contents of register BR2 into the NAME TABLE. This is the first word of the node for this name (see Figure 3). The machine then increments the contents of address register AR2 by one and enters the special code 77_8, left-adjusted, into register BR2. The contents of register BR2 are then transferred into the NAME TABLE as the second word of the node for this name. Referring again to Figure 3, the 77_8 is placed in the VAI field. This code indicates that no value has been assigned to the variable. When a value is assigned, either by execution of an assignment statement or execution of a READ statement, the VAI field is set to zero and the assigned value is stored in the rightmost 36 bits of the word.

After entering the variable in the NAME TABLE, the machine increments the current PROGRAM AREA address by one, fetches the next program constituent from memory M2, and stores it in register S. If this constituent is a comma (SCMA is true), the machine transfers to the initial point sequence. If this constituent is an end of statement symbol, $ (SDLR is true), the machine erases the declarator from the DELIMITER STACK by setting register D1 to zero, sets the control register ES to zero, and transfers to the unconditional statment sequence. If this constituent is not a comma or an end of statement symbol, the machine transfers to the error sequence.

If register D1 does not contain a declarator, the name being entered is a label and the machine sets subregister BR2(0,1) to 11. It then transfers the contents of register BR2 into the NAME TABLE. This is the first word of the node for this name (see Figure 3). The machine also increments the PROGRAM AREA address by one. It then increments the contents of register AR2 by one and transfers the PROGRAM AREA address and the current block level to subregisters BR2(30-41) and BR2(2-6) respectively. The contents of register BR2 are then transferred into the NAME TABLE as the second word of the node for this name. The machine then transfers to the block entry sequence.

4.10.3 NAME TABLE Search

As shown in Figure 16D, if register ES contains a zero, the machine sets register A to zero. It then transfers the contents of the memory location specified by the hash coded address to buffer register BR2 for examination. If register BR2 contains a non-zero value, the memory location is occupied and the machine compares the name in the location with the name being search for. It stores the result of this comparison in register CN. At the same time, it transfers the current block level to register CL. It then examines the contents of register CN. If register CN contains a non-zero value, the names are different

and a new address must be calculated to continue the search. The machine operations performed to calculate a new address are described in Section 4.10.2.1.

If register CN contains a zero, the name in the NAME TABLE matches the name being searched for and the machine must determine whether or not this entry of the name is still active; that is, is the block in which the name was declared still being processed. The machine must also determine whether this is the most recent entry of the name in the table or whether the name was declared again in another block. It first compares the block level of the name with the current block level. It decrements register CL by the contents of subregister BR2(2-6). It then examines the contents of register CL. If CL contains a zero, the two block levels are the same. If they are the same, the machine must compare the block numbers to finally determine whether or not the name is still active. Therefore, the machine transfers the current block number to register CB2 as shown in Figure 16E. It then decrements register CB2 by the contents of subregister BR2(7-11); the block number of the name in the NAME TABLE. It then examines the contents of register CB2. If register CB2 contains a zero, this entry of the name is still active. Therefore, its type and NAME TABLE address are transferred to register A. Since this entry occurred at the current block level, the search is complete. Therefore, the machine examines subregister A(1,2) to determine the name's type. If the name is a label, the machine transfers to the label processing sequence; otherwise, it transfers to the variable processing sequence.

As shown in Figure 16D, if the block levels differ (CL$\neq$0), the machine examines subregister CL(1) to determine whether the name in the NAME TABLE was declared at a block level that is higher or lower than the current block level. If subregister CL(1) contains a zero the name was declared at a lower block level and the machine must compare the block numbers to finally determine whether or not the name is still active. The machine transfers the block level

of the name from subregister BR2(2-6) to address register AR1. It then fetches the block number count for this block level from the BLOCK NUMBER COUNTER area of memory M1. It stores this count in register CB2. It then decrements register CB2 by the contents of subregister BR2(7-11); the block number of the name in the NAME TABLE. It then examines the contents of register CB2. If register CB2 contains a zero, this entry of the name is still active. Therefore, its type and NAME TABLE address are transferred to register A. Since this entry of the name occurred at a block level lower than the current block level, the machine calculates a new address in search of a more recent entry of the name. The machine operations performed to calculate a new address are described in Section 4.10.2.1.

If register CB2 contains a non-zero value or if subregister CL(1) indicates that the name in the NAME TABLE was declared at a higher block level than the current block level, the name is no longer accessible to the program. Therefore, the search is complete since a more recent entry would have been entered on top of this entry. The machine now examines the contents of register A. If register A contains a zero, no entry of the name was found. Therefore, the machine transfers to the error sequence. If register A contains some value, the machine examines subregister A(1,2) to determine the name's type. The machine reacts as described previously.

4.11 Label Processing Sequence

This sequence is shown in Figures 17A and 17B. The machine enters this sequence from the NAME TABLE activity sequence when a name is located in the NAME TABLE and identified as a label. Upon entering this sequence, the machine increments the current PROGRAM AREA address in index register XPA by one, fetches the next program constituent from memory M2, and stores it in register S. If this constituent is a colon (SCO is true), the machine transfers to the initial point sequence; thereby passing over the label and the colon.

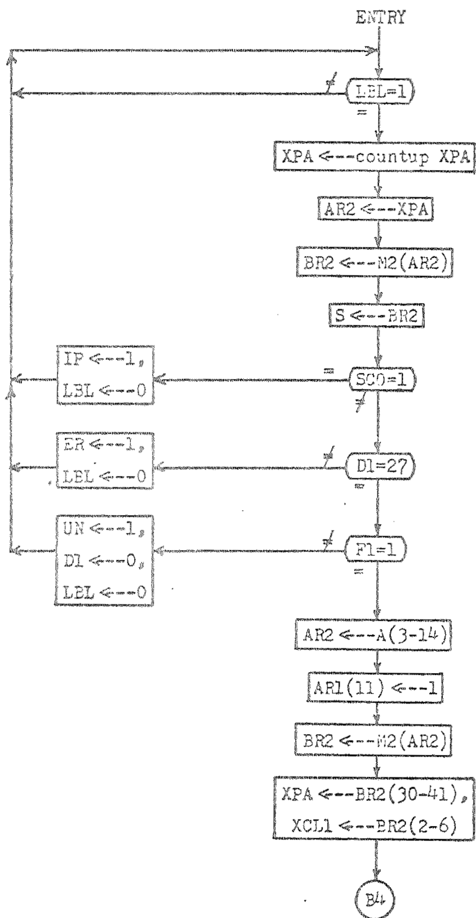


Figure 17a Label Processing Sequence Chart

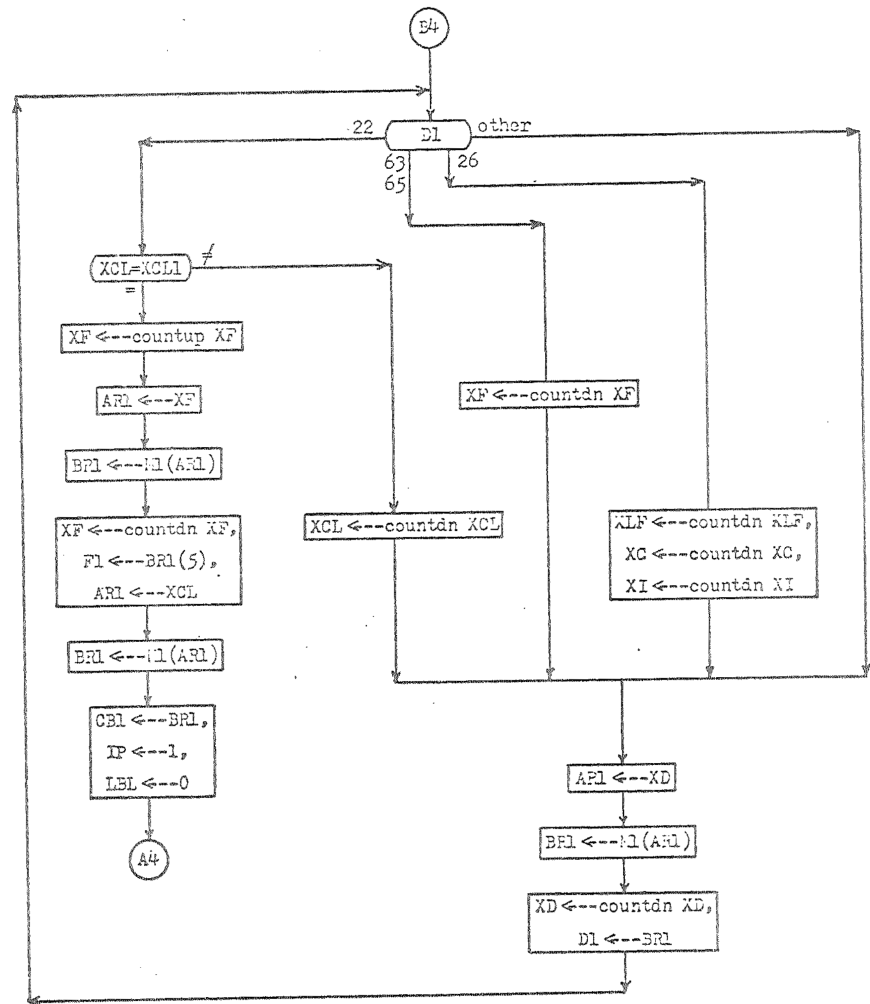


Figure 17b Label Processing Sequence Chart

If the constituent is not a colon the machine examines the top element on the DELIMITER STACK in register D1. If register D1 contains the delimiter GOTO ($D1=27_8$), the machine examines the current flag, register F1. If F1 contains zero, the machine erases the delimiter from register D1 and transfers to the unconditional statement sequence. If F1 contains one, the machine acts to transfer to the location in the program specified by the label. The machine first transfers the label's NAME TABLE address from subregister A(3-14) to address register AR2. This address was placed in register A during execution of the NAME TABLE activity sequence. The machine then increments address register AR2 by one; thus, AR2 contains the address of the second word of the two word node contained in the NAME TABLE for the label. The machine then fetches this word from memory M2. This word contains the PROGRAM AREA address of the label (see Section 2.1.2.1). The machine transfers this address to index register XPA.

Before the machine can begin executing the program at the new address, it must first set its stacks and registers for operation at the block level of the label. To do this it first transfers the label's block level from subregister BR2(2-6) to to index register XCL1. It then examines the contents of register D1. If D1 contains the delimiter FOR ($D1=26_8$), the program has transferred out of an iteration statement and the elements that were placed on the COUNT STACK, the INITIAL STACK and the LINK TO FORLIST STACK for this iteration are no longer needed. Therefore, the current addresses of these stacks (located in registers XC, XI, and XLF respectively) are decremented by one.

If register D1 contains either of the delimiters THEN or ELSE ($D1=63_8$ or $D1=65_8$), the program has transferred out of a conditional statement and the flag associated with the conditional statement is no longer required. Therefore, the current FLAG STACK address in index register XF is decremented by one.

If register D1 contains the delimiter BEGIN ($D1=22_8$), the machine compares the current block level in index register XCL with the block level of the label in index register XCL1. If they are not equal, the current block level is decremented by one. The machine sequentially fetches DELIMITER STACK elements from memory M1, stores them in register D1, and examines them until it recognizes the delimiter BEGIN and finds the current block level equal to the block level of the label. At that time, the machine updates the current flag and the current block number registers by fetching their values from memory M1. It then transfers to the initial point sequence.

4.12 Variable Processing Sequence

This sequence is shown in Figures 18A and 18B. The machine enters this sequence from the NAME TABLE activity sequence when a name is located in the NAME TABLE and identified as a variable. Upon entering this sequence, the machine first examines the current flag, register F1. If F1 contains one, the machine examines the contents of register D1. If register D1 contains either the delimiter READ ($D1=51_8$) or the delimiter WRITE ($D1=47_8$), the machine examines the iteration control register IT. If register IT contains a value of one, it is set to the current OPERAND LIST address located in index register XPD. A value of one in register IT indicates that at least one iteration is being processed. Therefore, the OPERAND LIST contains the NAME TABLE address(es) for the iteration's controlled variable(s). By setting register IT to the OPERAND LIST address before the variables in the communication statement are placed in the list, the machine has a means of determining which variables in the OPERAND LIST are to be processed and removed during execution of the read/write execution sequence. The machine next transfers the NAME TABLE address of the variable from subregister A(3-14) to subregister BR2(30-41) and increments the OPERAND LIST address by one. It then stores the contents of register BR2

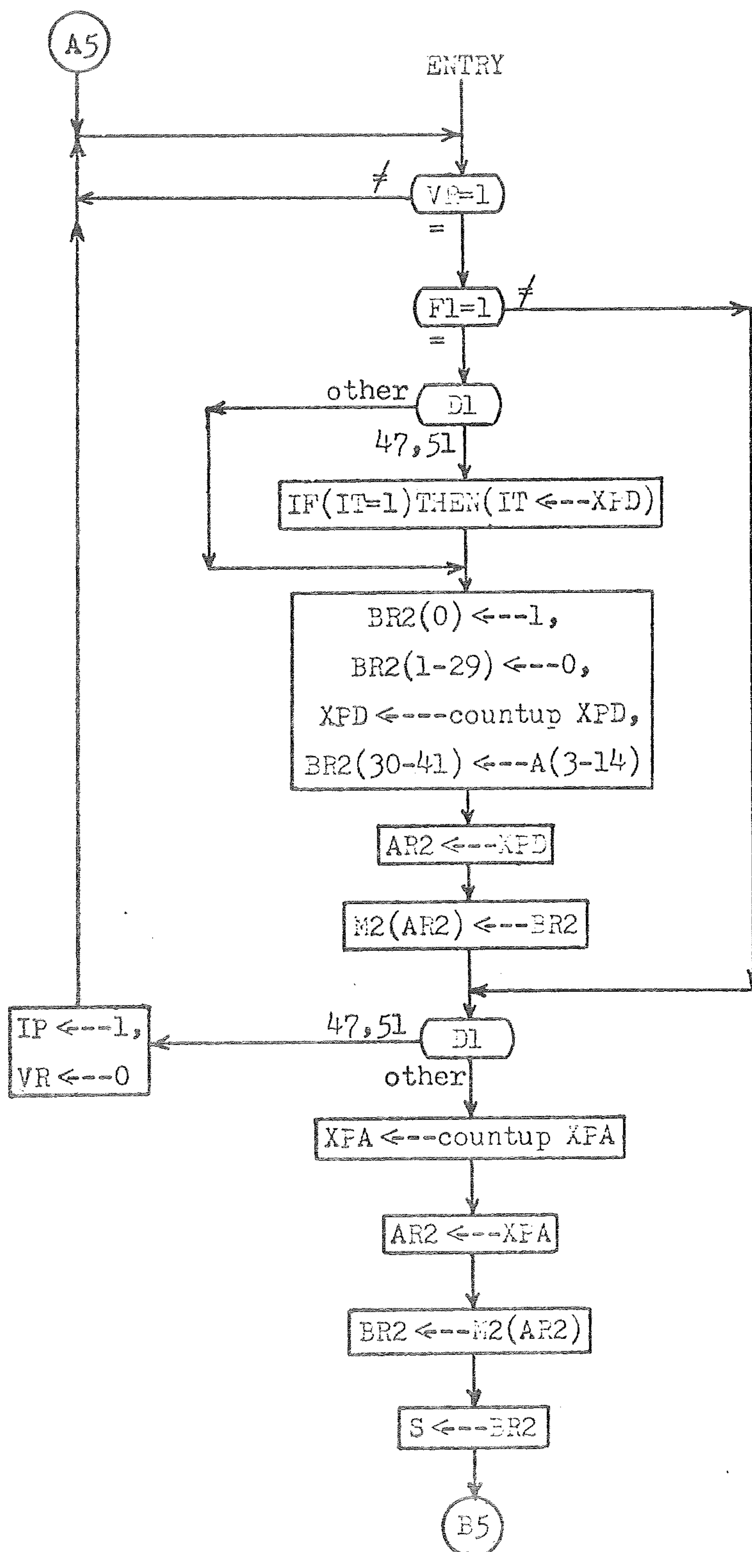


Figure 18a Variable Processing Sequence Chart

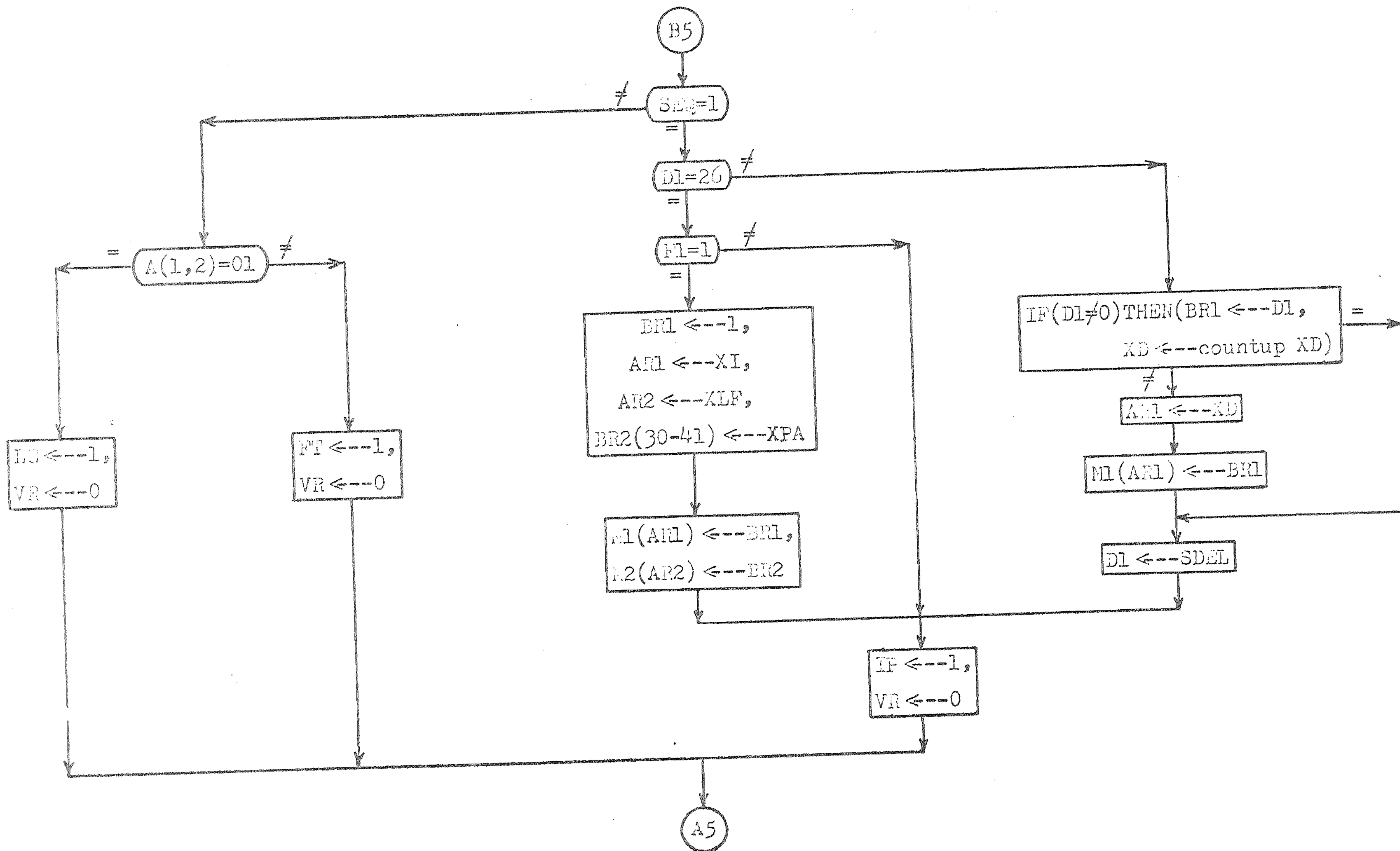


Figure 18b Variable Processing Sequence Chart

in the OPERAND LIST. Figure 4 shows the format for storing an address in the OPERAND LIST. The NAME TABLE address of the variable was placed register in A during execution of the NAME TABLE activity sequence.

After storing the NAME TABLE address, the machine again examines the contents of register D1. If D1 contains either the delimiter READ ($D1=51_8$) or the delimiter WRITE ($D1=47_8$), the machine transfers to the initial point sequence.

If register D1 contains any delimiter other than READ or WRITE, the machine increments the current PROGRAM AREA address by one, fetches the next program constituent from memory M2, and stores it in register S. If this constituent is not an equal sign (SEQ is false), the machine examines subregister A(0,1) to determine the type of the variable. If its type is Boolean, the machine transfers to the logical expression sequence. If its type is integer, the machine transfers to the factor sequence. If this constituent is an equal sign, the contents of register D1 are examined again. If register D1 contains the delimiter FOR ($D1=26_8$), the current flag is examined. If F1 contains a one, the machine stores a one in the top element on the INITIAL STACK. It also stores the current PROGRAM AREA address in the top element on the LINK TO FORLIST STACK. It then transfers to the initial point sequence. If F1 contains a zero, the machine transfers immediately to the initial point sequence.

4.13 <u>Factor Sequence</u>

This sequence is shown in Figure 19A-G. The machine enters this sequence whenever a number, a NAME TABLE address of an integer variable, or the resultant value of a sum enclosed by parenthesis is placed in the OPERAND LIST. In executing this sequence, the machine examines the contents of register D1 to determine whether or not a multiplication or a division operation is specified. If either of these operations is specified and the current flag, register F1, contains one, the machine performs the operation on the top two elements in the OPERAND LIST. After the operation, the two operands are replaced

in the OPERAND LIST by the result. The machine then removes the operator from the DELIMITER STACK and transfers to the term sequence.

As shown in Figure 19A, if register D1 contains zero, the machine transfers the current DELIMITER STACK address in index register XD to address register AR1. It then fetches the top element on the stack from memory M1 and stores it in register D1. At the same time, it decrements register XD by one. It then examines the contents of register D1. If D1 contains a multiplication operator (D1=54_8) or a division operator (D1=61_8), the machine examines the current flag. If F1 contains a one, the machine performs the specified operation. If F1 contains zero, the machine removes the operator from the DELIMITER STACK by setting register D1 to zero. It then transfers to the term sequence.

If register D1 contains any delimiter other than the multiplication or division operator, the machine transfers directly to the term sequence.

4.13.1 Multiplication

The multiplication algorithm implemented in this machine is Booth's algorithm for numbers in signed 2's-complement representation (6). To perform the multiplication, the machine fetches the two operands from the OPERAND LIST as shown in Figure 19B. The current OPERAND LIST address in index register XPD is transferred to address register AR2. The contents of the memory location are then transferred into buffer register BR2. If subregister BR2(0) contains a one, this operand is a NAME TABLE address for a variable. The machine transfers this NAME TABLE address to address register AR2, increments it by one, and transfers the value of the variable to register BR2. The NAME TABLE address is incremented by one to obtain the address of the second word of the variable's two word node. This is done since it is the second word of the node that contains the value of the variable. The VAI field in subregister BR2(0-5) is now examined. If this field contains the special character 77_8, the variable has not been assigned a value and cannot be used as an operand. Therefore, the

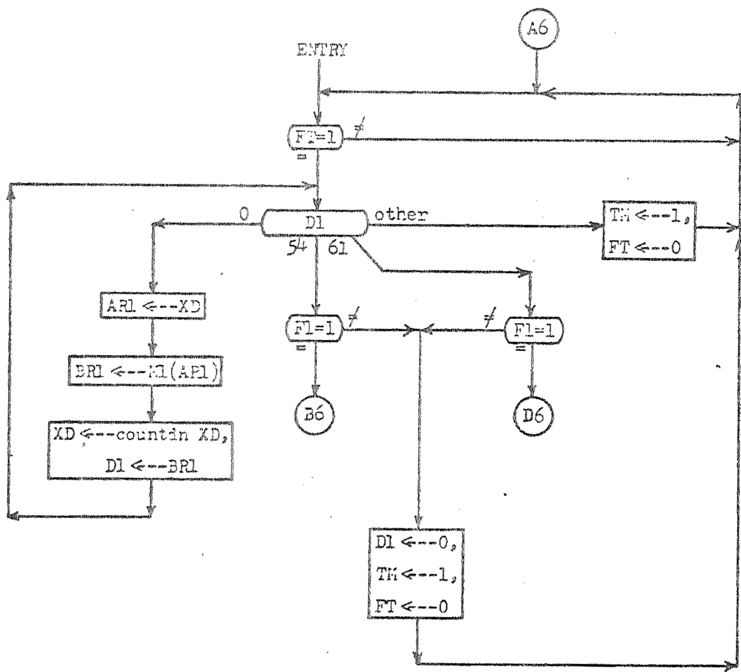


Figure 19a Factor Sequence Chart

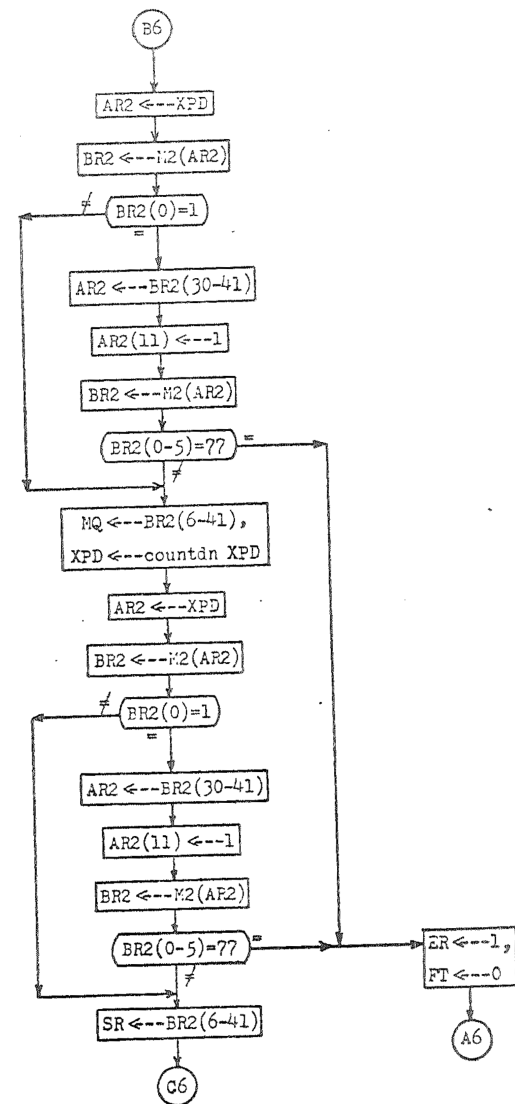


Figure 19b Factor Sequence Chart

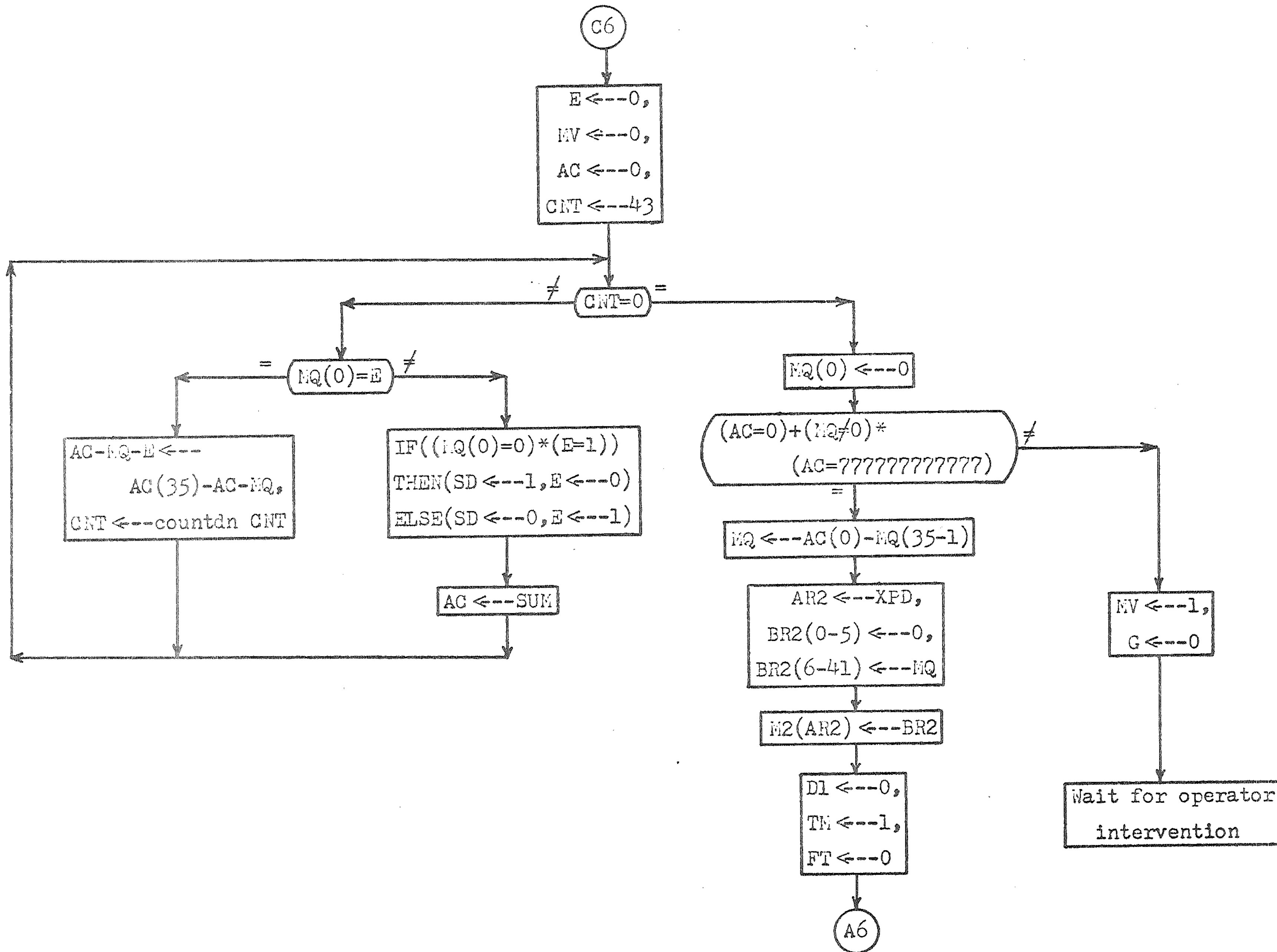


Figure 19c Factor Sequence Chart

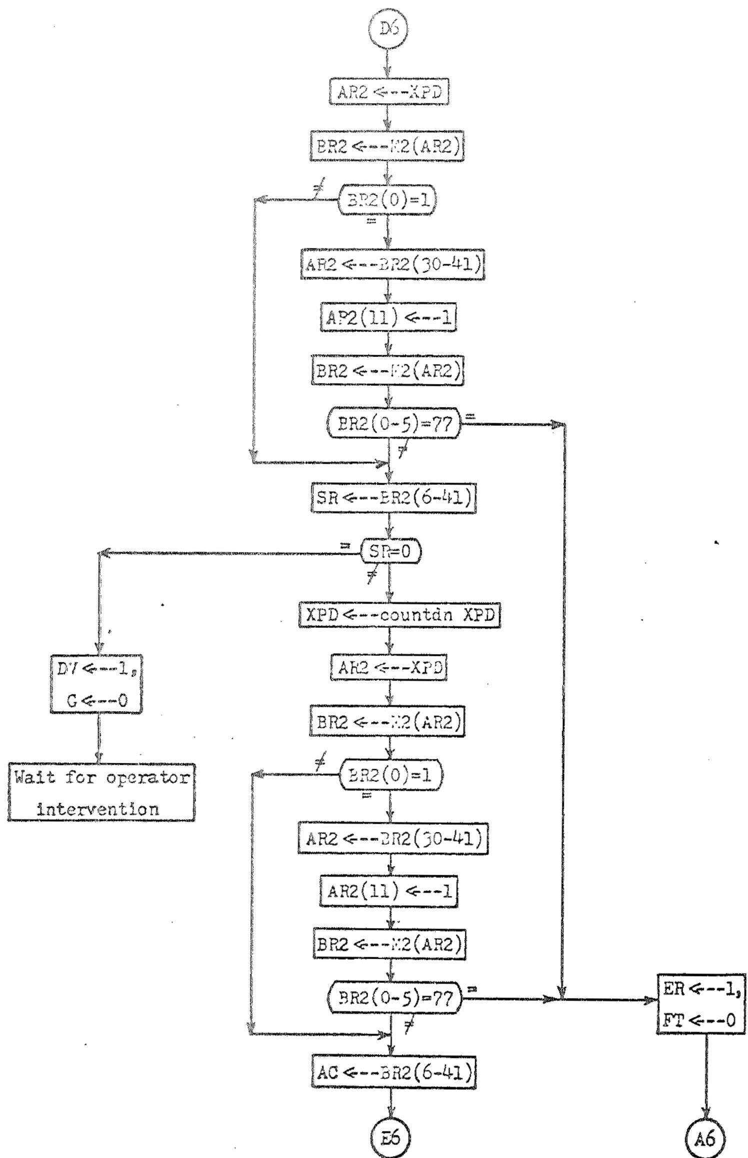


Figure 19d Factor Sequence Chart

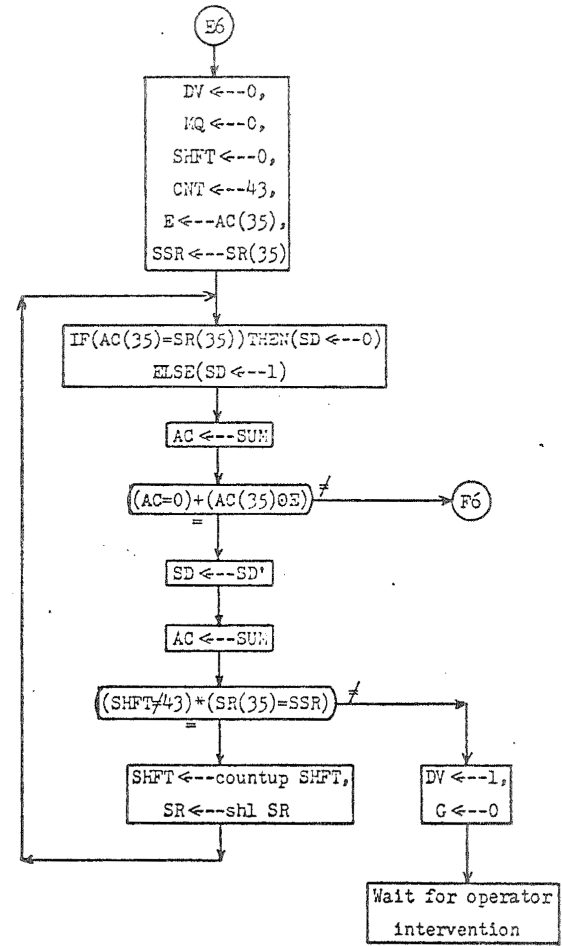


Figure 19e Factor Sequence Chart

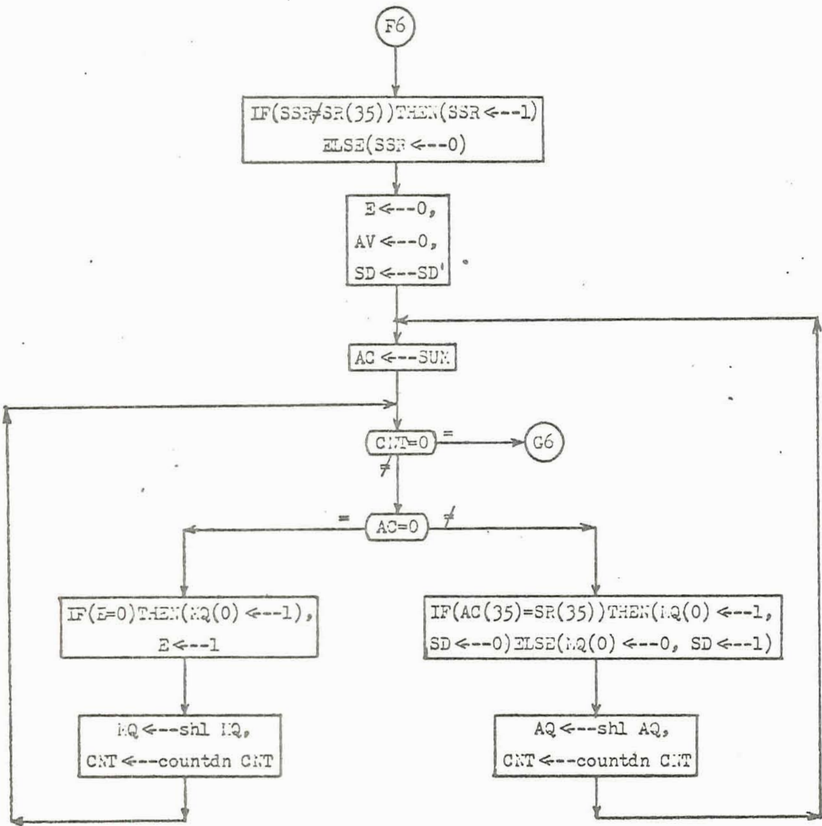


Figure 19f Factor Sequence Chart

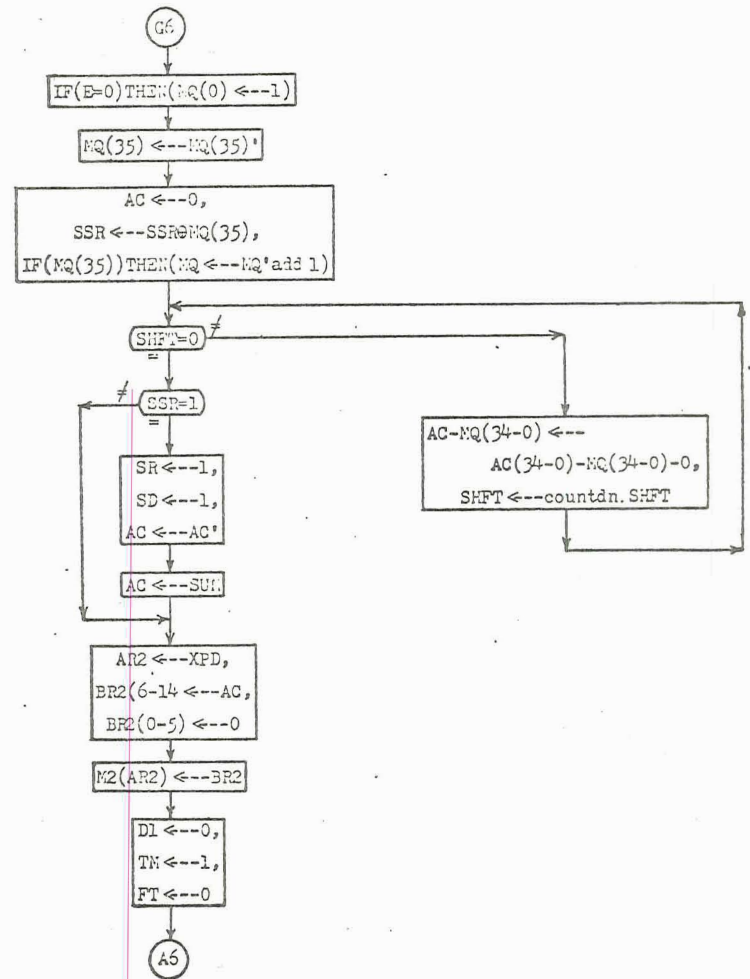
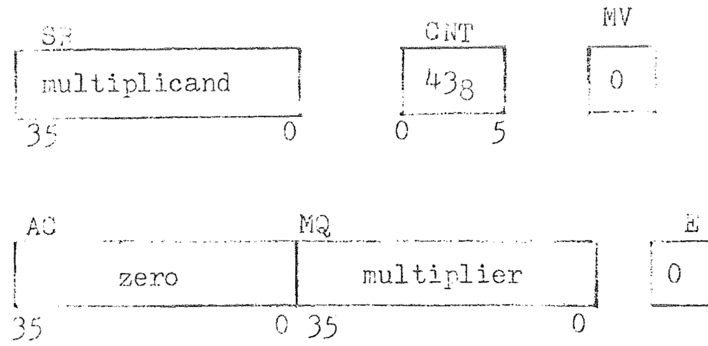
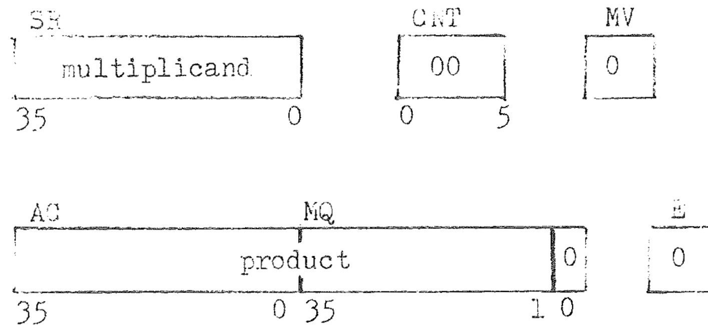


Figure 19g Factor Sequence Chart

machine transfers to the error sequence. If the variable has been assigned a value or if the operand itself is a value, the machine transfers the value to the multiplier-quotient register, MQ. It then decrements the OPERAND LIST address by one and fetches the second operand from the OPERAND LIST. The second operand is the multiplicand and is transferred to register SR. As shown in Figure 19C, the machine next sets the accumulator, AC, the multiply overflow indicator, MV, and the reference flipflop, E, to zero. It also sets the counter CNT to 43_8. The contents of the registers involved in the multiplication process at this time are as shown in Figure 20(a). The multiplication is now performed. The machine first examines the contents of register CNT. If CNT does not contain zero, the multiplication continues; otherwise, it is complete. With the contents of register CNT not equal to zero, the machine compares the contents of subregister MQ(0) with the contents of register E. If they are equal, the combined register AC-MQ-E is shifted one bit to the right and register CNT is decremented by one. The machine then reexamines the contents of register CNT. If the contents of subregister MQ(0) and register E are not equal, the machine either increments or decrements the accumulator by the multiplicand in register SR. Register SD controls the operation and terminals SUM are the resultant value. Which operation is performed depends upon the contents of register E. If register E contains a one, register SD is set to one and the multiplicand is added; otherwise, register SD is set to zero and the multiplicand is subtracted. In either case, register E is complemented. After the addition or subtraction, the machine again examines the contents of register CNT. Since CNT is not decremented when an addition or a subtraction is performed, it can not contain a value of zero. Therefore, subregister MQ(0) and register E are again compared. Since register E was complemented, the comparison indicates they are equal and the machine shifts the combined register



(a) before multiplication



(b) after multiplication

Figure 20 Contents of Registers Involved in the Multiplication Process

AC-MQ-E one bit to the right and decrements register CNT by one. The multiplication process continues until the 35 magnitude bits of the multiplier have been rightshifted out of MQ. At that time, CNT will contain a value of zero and the final product will be contained in registers AC and MQ as shown in Figure 20(b). Note that subregister MQ(0), is not part of the product. This bit is set to zero so as not to cause an error in the overflow test. Since values stored in memory are limited to 36 bits, the machine examines the product to determine if it exceeds 35 magnitude bits in significance. If it does, overflow has occurred and the machine sets register MV to one and the run-stop control register, G, to zero. It then waits for the operator to intervene. If overflow has not occured, the product is right shifted one bit. It is then transferred to subregister BR2(6-41) and finally stored in the OPERAND LIST. The machine then erases the multiplication operator from the DELIMITER STACK and transfers to the term sequence.

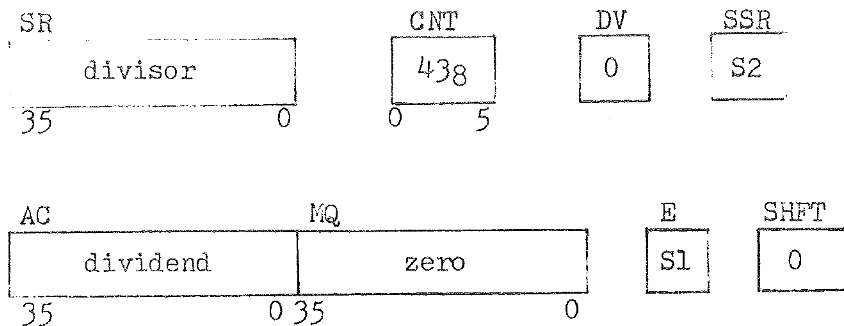
4.13.2 Division

The division algorithm implemented in this machine is von Neumann's nonrestoring method (6). To perform the division, the machine fetches the two operands from the OPERAND LIST as shown in Figure 19D. The machine operations performed to fetch these operands are described in Section 4.13.1. The divisor is fetched first and placed in register SR. If the divisor has a value of zero, the machine sets the divide overflow indicator, DV, to one and the run-stop control flipflop, G, to zero. It then waits for the operator to intervene. If the divisor is not zero, the machine continues on and fetches the dividend. It stores the dividend in the accumulator.

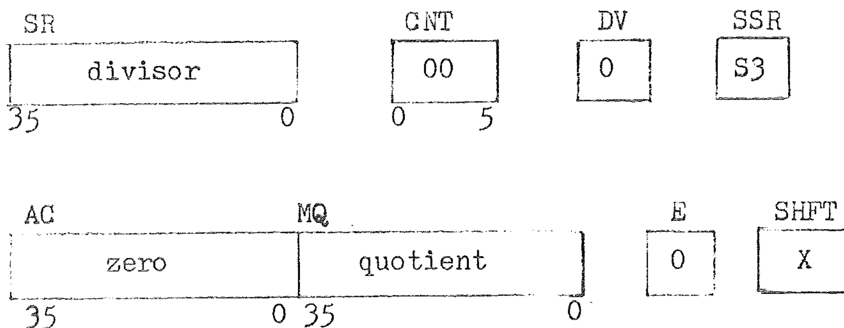
As shown in Figure 19E, after fetching the operands, the machine sets registers SHFT, MQ, and DV to zero. It also stores the sign of the dividend in reference flipflop E, stores the sign of the divisor in register SSR, and

sets register CNT to 43_8. Figure 21(a) depicts the contents of these registers at this time. The machine then tests the divide-stop condition. A dividend larger than or equal to the divisor identifies this condition. In testing for this condition, the machine subtracts the divisor from the dividend and examines the difference. If the two values differ in sign, the machine effectively performs the subtraction by adding the two values. The terminals SUM are the results of this operation. The difference is stored in the accumulator. If the accumulator contains zero or if the sign of the difference is the same as the sign of the dividend (which is stored in register E), the divide-stop condition exists. When this condition exists, the machine attempts to scale the divisor so that it becomes larger than the dividend and the division can continue. The machine first restores the dividend in the accumulator by adding the divisor to the difference. It then determines whether or not the divisor can be scaled. If the divisor has not been shifted to the left 35 bits and if another shift would not cause a significant bit of the divisor's magnitude to be lost, the divisor is shifted to the left one bit and register SHFT is incremented by one. The machine then compares the scaled divisor and the dividend as before. If the divide-stop condition still exists, the machine again attempts to scale the divisor. This process continues until either the divisor becomes larger than the dividend, in which case the division operation is executed, or no further scaling of the divisor can occur. At that point the machine sets register DV to one and register G to zero. It then waits for the operator to intervene.

As shown in Figure 19F, if the divide-stop condition does not occur or it is eliminated by scaling, the division begins by setting register SSR. Note that the sign of the divisor was placed in register SSR earlier in this sequence. If the sign bit in subregister SR(35) changed while the divisor was being scaled, the setting of register SSR records this change. This step is necessary to



(a) before division



(b) after division

S1: sign of dividend
S2: sign of divisor
S3: result of comparison of contents of SSR and sign of quotient
X: number of bits quotient should be shifted to the left

Figure 21 Contents of Registers Used in the Division Process

to insure that the quotient is given the proper sign. The machine next sets reference flipflop E to zero and restores the dividend in the accumulator. It then examines the contents of register CNT. If the value in register CNT is non-zero, the division continues; otherwise, it is complete. Continuing the division, the machine examines the contents of the accumulator. If the accumulator contains a non-zero value, the machine determines the first quotient bit, which is one or zero according to the sign bits of the divisor and the partial remainder being the same or different respectively. This bit is inserted in subregister MQ(0) and the add-subtract flipflop, SD, is set accordingly. The contents of casregister AQ are then shifted one bit to the left and the contents of register CNT are decremented by one. If the quotient bit is one, the contents of register SR are subtracted from the contents of the accumulator; otherwise, the contents of register SR are added to those of the accumulator. If the contents of the accumulator remain non-zero, the machine continues this process until all bit positions in register MQ except MQ(0) contain quotient bits. At that time, register CNT contains zero (see Figure 21(b)). The machine then applies a correction to the partial quotient. As shown in Figure 19G, this correction consists of inserting a one in subregister MQ(0) and complementing subregister MQ(35).

If the value in the accumulator becomes zero at any time during the process, the division is complete and the machine shifts the quotient to the left and decrements the number in register CNT until the number reaches zero. In order to provide the proper correction in this case, a one is inserted in subregister MQ(0) before the quotient is shifted left. Reference flipflop E controls the insertion of the one. Subregister MQ(35) is complemented after the quotient is shifted.

After completing the division process, the machine compares the sign of the quotient with the value in register SSR. It stores the results of this comparison in register SSR. It also sets the accumulator to zero and complements the quotient in register MQ if it is negative. The machine then examines the contents of register SHFT to determine if the divisor was scaled. If it was, the machine shifts the quotient left into the accumulator and decrements the contents of register SHFT. When the contents of register SHFT reach zero, the machine examines the value in register SSR. If this value is one, the machine complements the contents of the accumulator. It then transfers the contents of the accumulator to subregister BR2(6-41) and finally stores them in the OPERAND LIST. The machine then erases the division operator from the DELIMITER STACK and transfers to the term sequence.

Notice that in storing the quotient, the quotient bits in register MQ are truncated. This results in a quotient of zero whenever a division is performed and the divisor is not scaled.

4.14 Term Sequence

This sequence is shown in Figure 22A-D. When the machine enters this sequence it first determines whether or not register S contains a multiplication or a division operator (SSTR is true or SDI is true). If register S contains either of these operators, the machine places the operator on the DELIMITER STACK and transfers to the initial point sequence; otherwise, the machine examines the contents of register D1 to determine what operation to perform. As shown in Figure 22A, if register D1 contains a zero, the machine fetches the top element on the DELIMITER STACK from memory M1 and transfers it to register D1. It then examines the contents of register D1 again. If D1 contains an addition operator ($D1=20_8$) or a subtraction operator ($D1=40_8$), the machine sets register SD to one or zero respectively. It then examines the current flag.

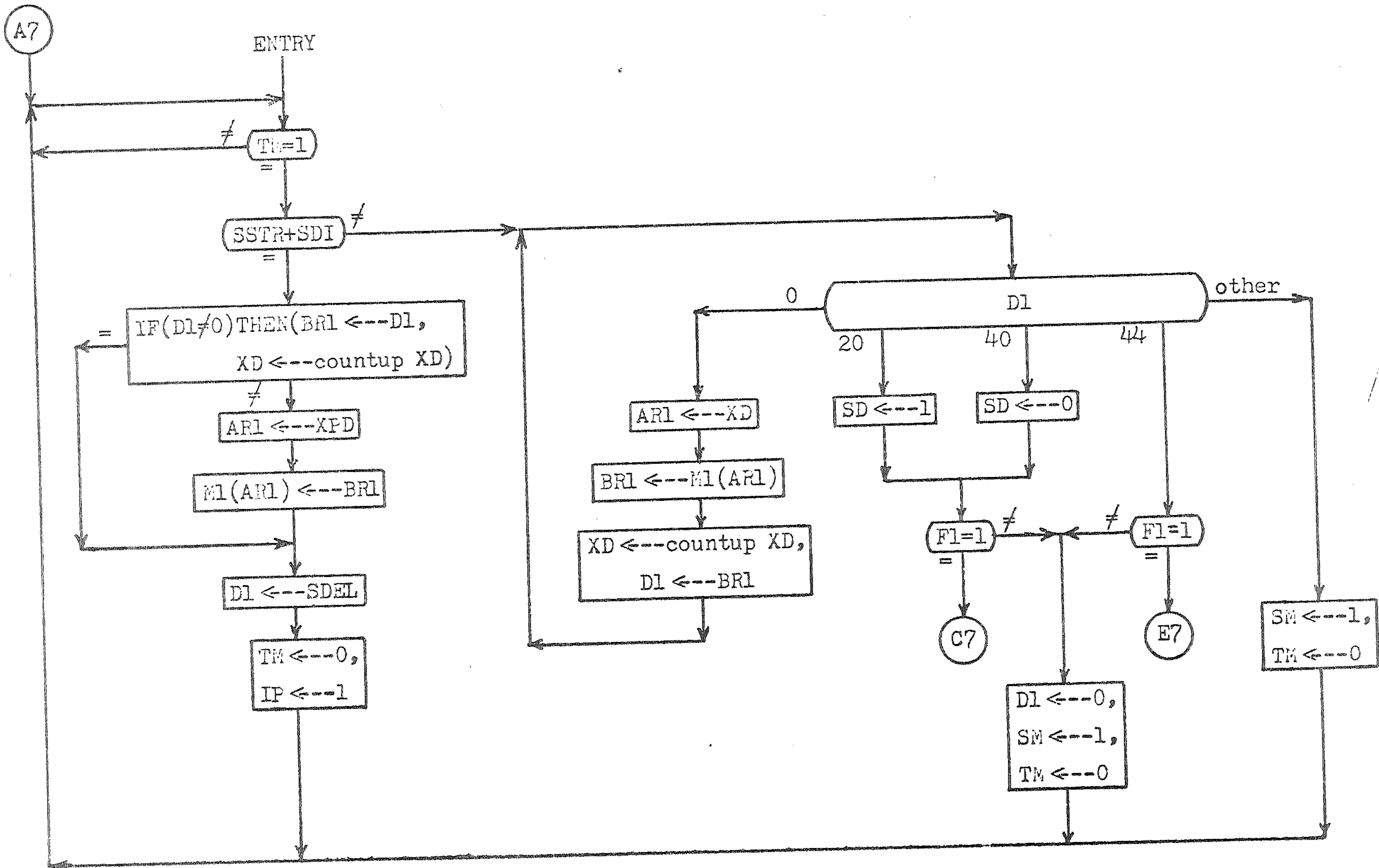


Figure 22a Term Sequence Chart

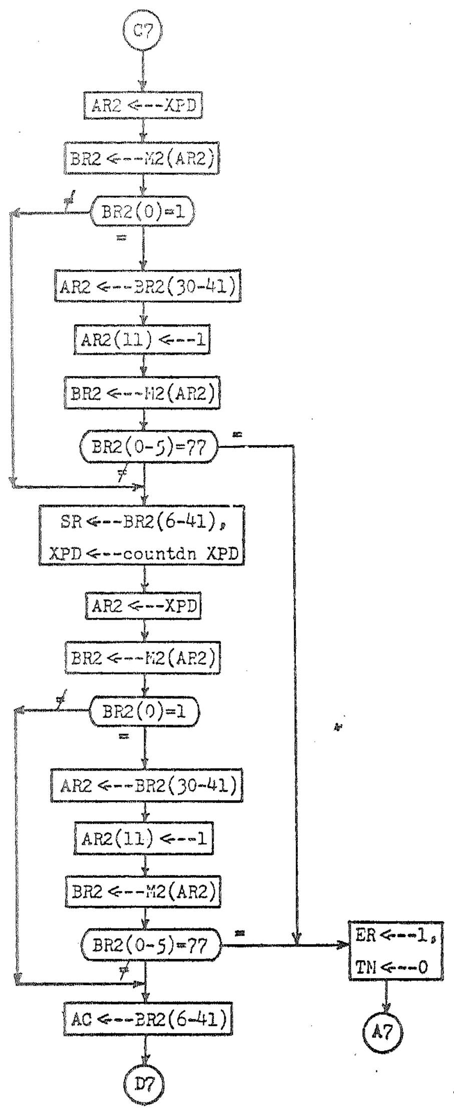


Figure 22b Term Sequence Chart

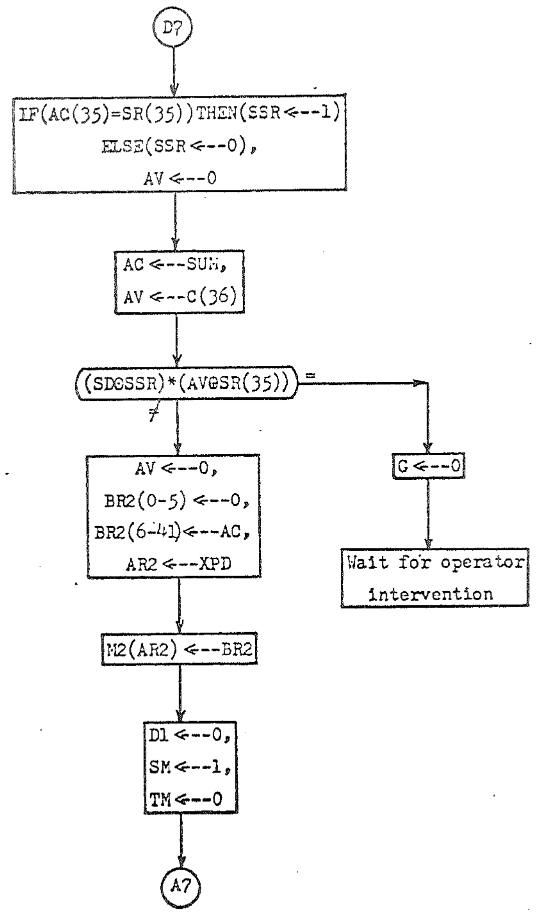


Figure 22c Term Sequence Chart

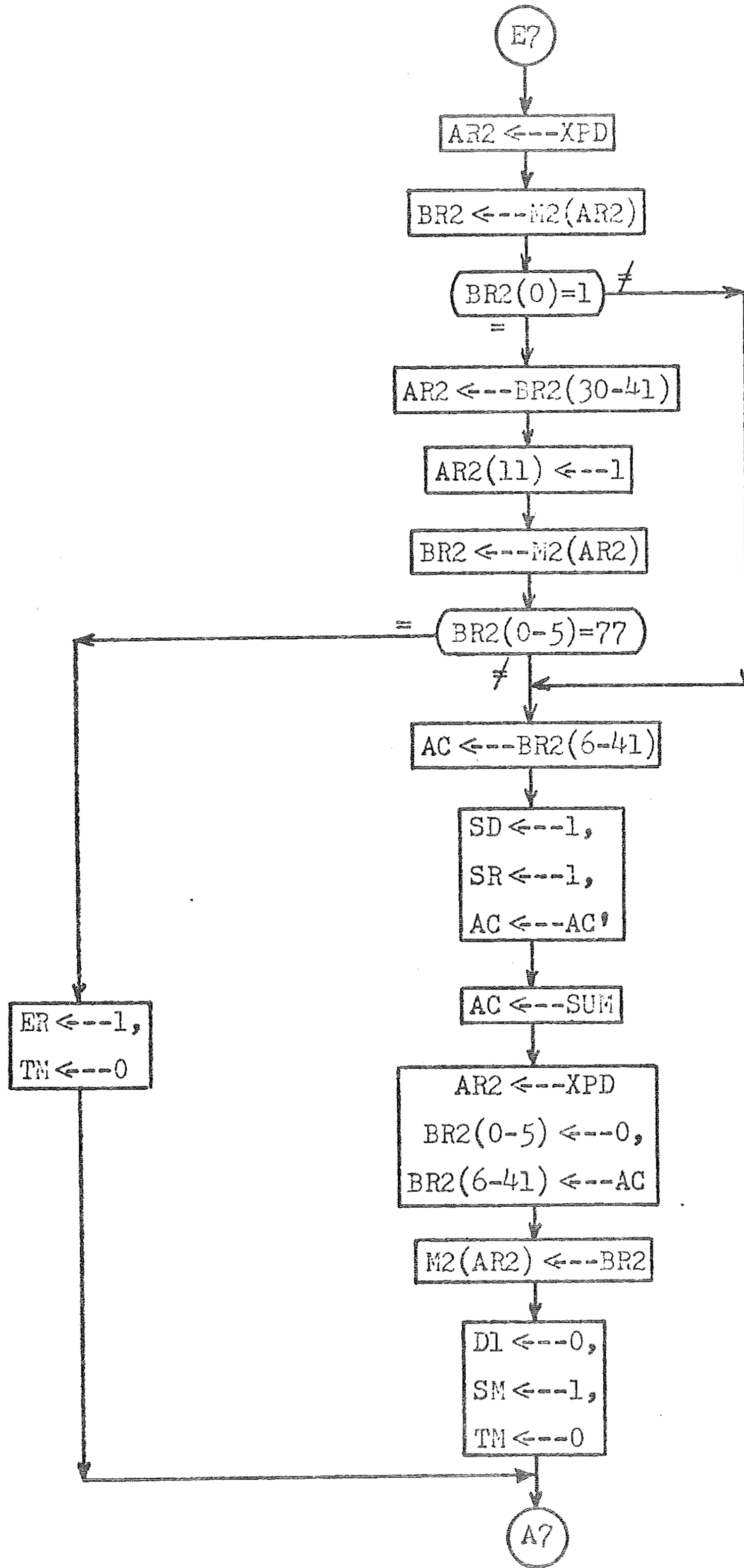


Figure 22d Term Sequence Chart

If F1 contains a one, the machine performs the specified operation. If F1 contains a zero, the machine removes the operator from the DELIMITER STACK by setting register D1 to zero. It then transfers to the sum sequence. If register D1 contains a unary minus operator ($D1=44_8$), the machine examines the current flag and reacts in the same way as described for the addition and subtraction operators. If register D1 contains any other delimiter, the machine transfers directly to the sum sequence.

4.14.1 Addition and Subtraction

Addition and subtraction are implemented in this machine as direct addition and subtraction of numbers in signed 2's-complement representation. A 36-bit parallel adder-subtractor is wired between registers AC and SR. Terminals SUM are the different outputs of this adder-subtractor. The value in register SD determines whether these outputs reflect the sum or difference of the inputs. To perform an addition or a subtraction, the machine fetches the two operands from the OPERAND LIST as shown in Figure 22B. A description of the machine operations performed to fetch these operands is contained in Section 4.13.1. The addend (subtrahend) is fetched first and placed in register SR. The augend (minuend) is then fetched and placed in the accumulator. The machine then compares the signs of the two operands. If they are the same, it sets register SSR to one; otherwise, it sets register SSR to zero. It also sets the add overflow indicator, AV, to zero. It then stores the outputs of the adder-subtractor in the accumulator. The carry from the most significant bit is stored in register AV. The machine then tests for overflow. If overflow has occurred, the machine sets the run-stop control register, G, to zero. It then waits for the operator to intervene. If there is no overflow, the machine sets register AV to zero and transfers the sum (difference) into the

OPERAND LIST. It then transfers to the sum sequence.

4.14.2 Unary Minus

As shown in Figure 22E, the machine fetches an operand from the OPERAND LIST and stores it in the accumulator. The machine operations performed to fetch the operand are described in Section 4.13.1. The machine next complements the contents of the accumulator and stores a one in register SR. It then increments the contents of the accumulator by the contents of register SR. This effectively yields the 2's-complement of the original value of the operand. The machine then stores the operand back in the OPERAND LIST. It then removes the unary minus operator from the DELIMITER STACK and transfers to the sum sequence.

4.15 Sum Sequence

As shown in Figure 23A, the operations performed in this sequence depend upon the output of the program constituent decoder. If this decoder indicates that register S contains either an addition, a subtraction, or a relational operator (SPL, SMI, or SRO is true), the machine places the operator on the DELIMITER STACK and transfers to the initial point sequence. If the decoder indicates that register S contains a right parenthesis (SRP is true), the machine examines register D1 to determine what action to take. A value of zero in register D1 causes the machine to fetch the top element on the DELIMITER STACK and store it in D1. If register D1 contains a left parenthesis ($D1=74_8$), the machine transfers to the factor sequence; otherwise, it transfers to the error sequence.

If the program constituent decoder indicates that register S contains either the delimiter THEN or the delimiter END, the machine examines the contents of register D1 to determine what action to take. Again, a zero causes the machine to fetch the top element on the DELIMITER STACK and store it in register D1. If the top element on the DELIMITER STACK is not a relational operator, the machine examines the output of the program constituent decoder.

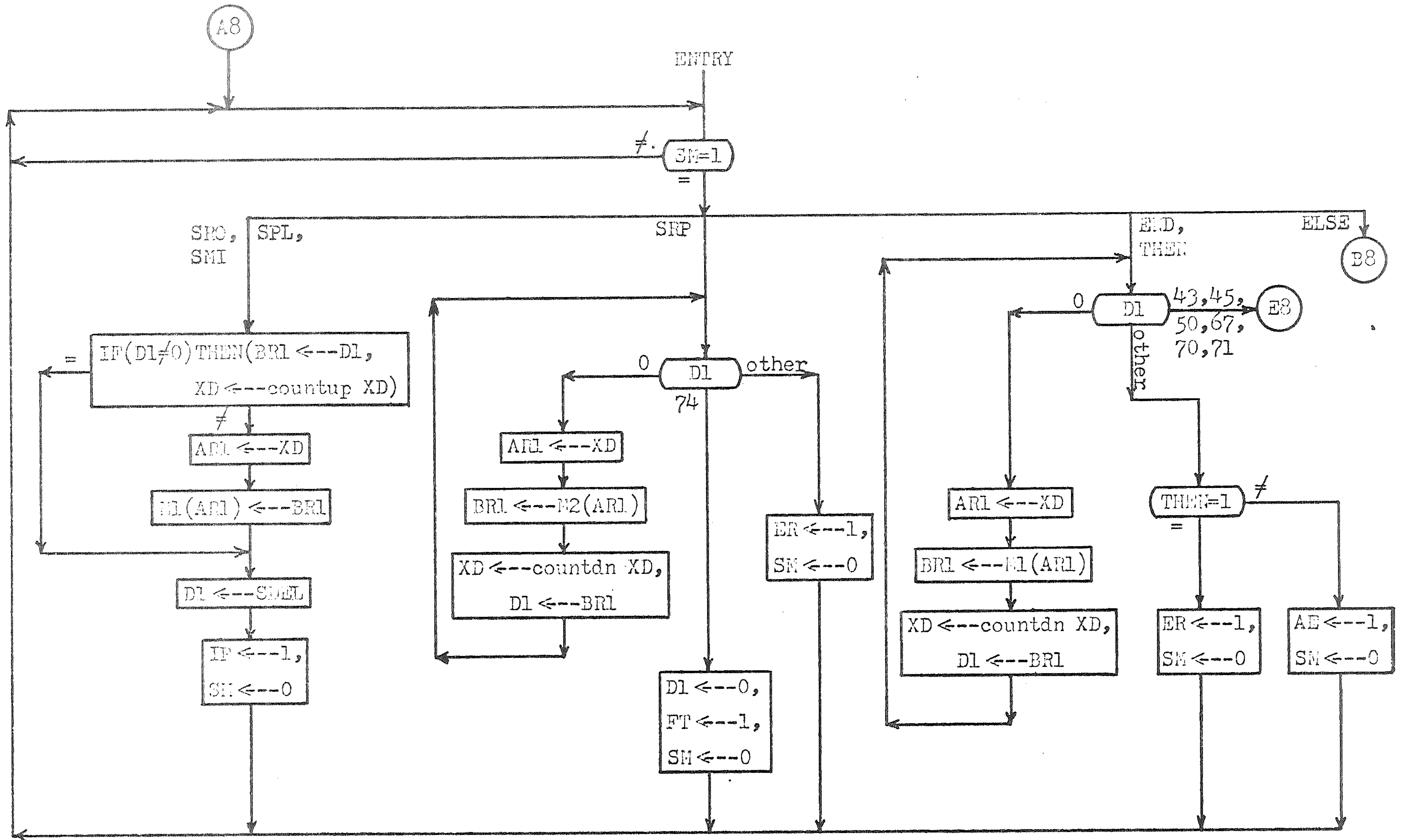


Figure 23a Sum Sequence Chart

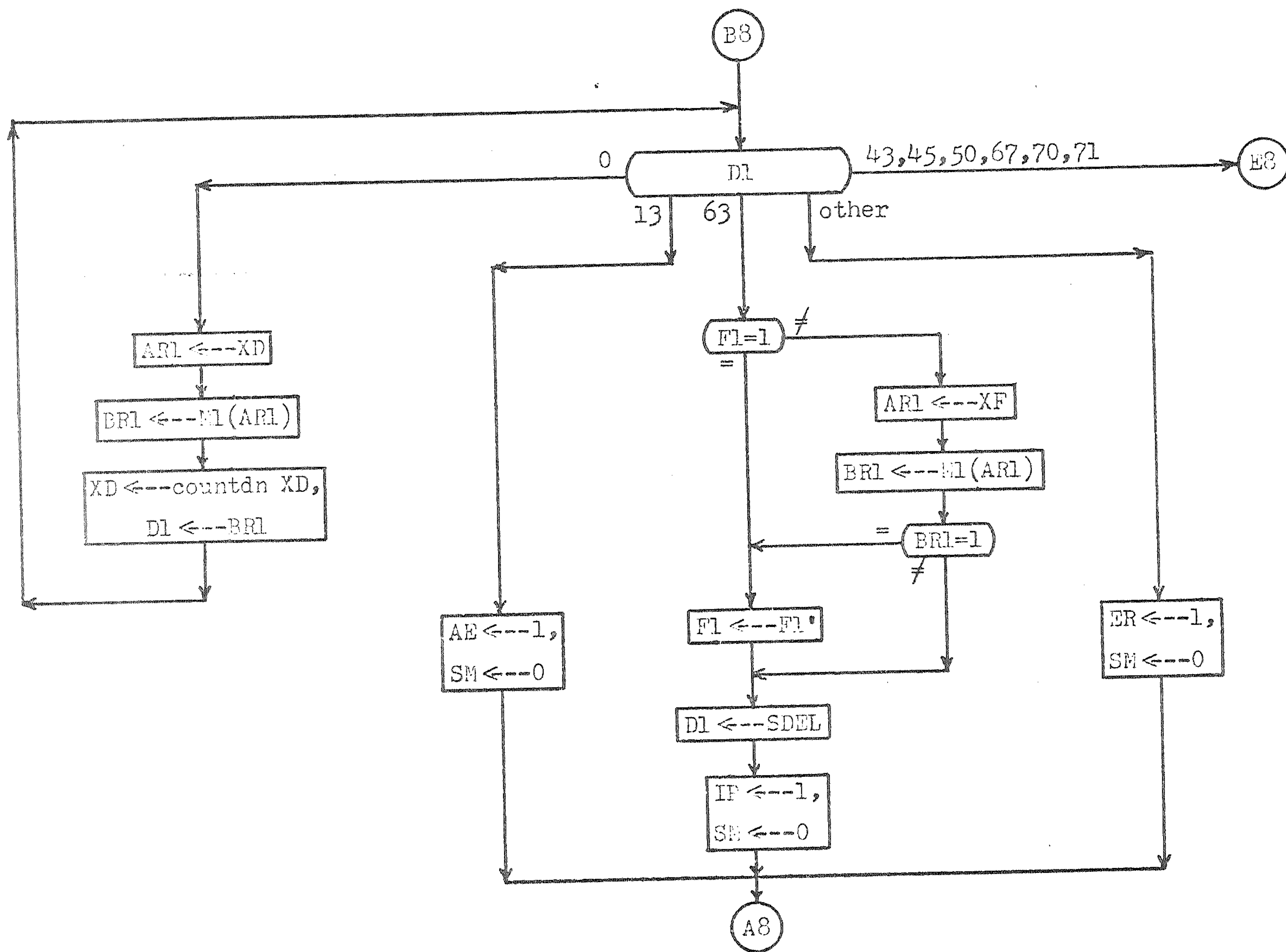


Figure 23b Sum Sequence Chart

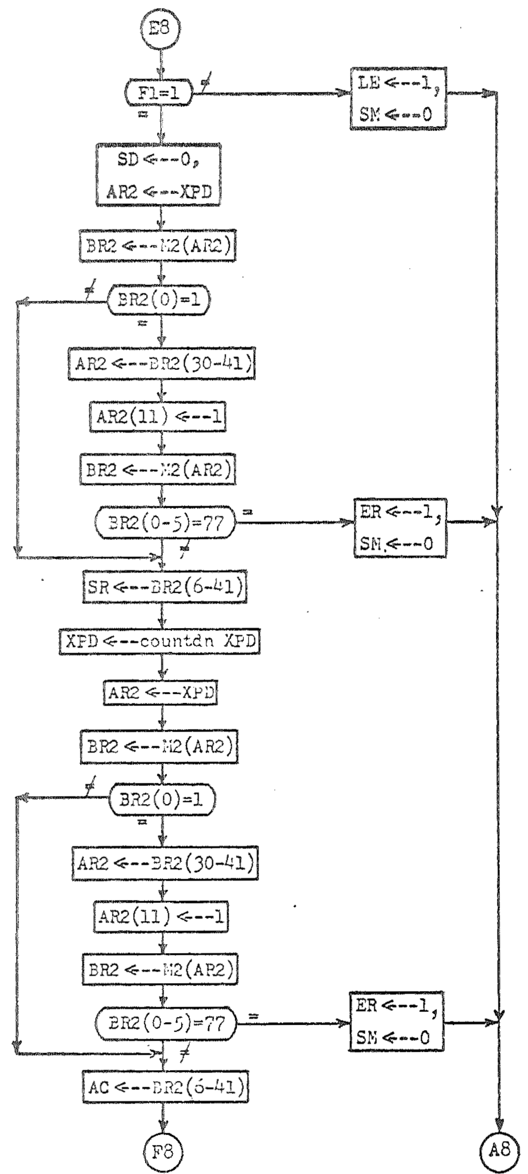


Figure 23c Sum Sequence Chart

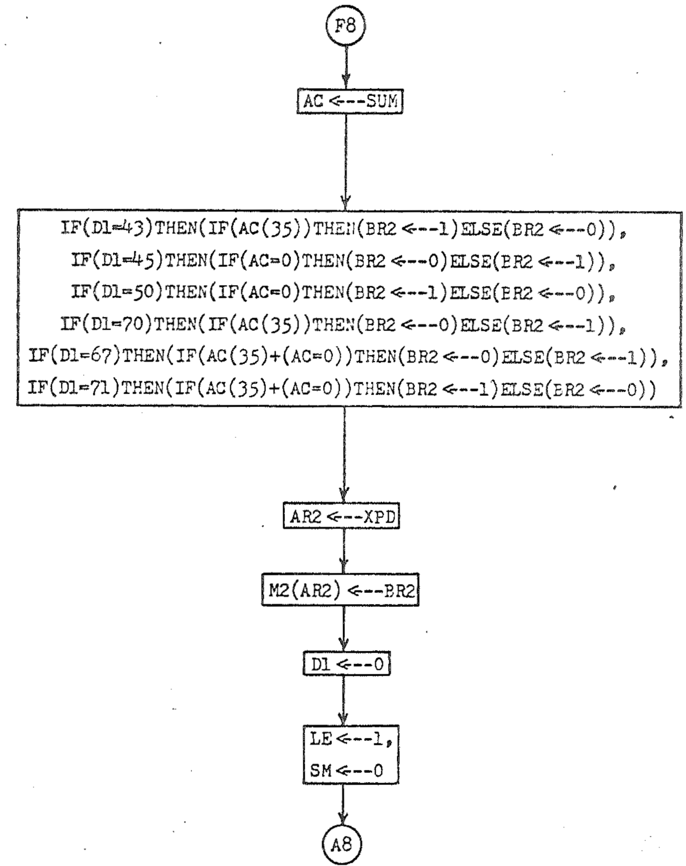


Figure 23d Sum Sequence Chart

If the decoder indicates that register S contains the delimiter THEN, the machine transfers to the error sequence; otherwise, it transfers to the arithmetic expression sequence.

If register D1 contains a relational operator, the machine examines the current flag. As shown in Figure 23C, if F1 contains zero, the machine transfers to the logical expression sequence; otherwise, it performs the specified operation. It does this by subtracting one operand from the other and examining the difference. If the difference indicates that the relation is true, the machine places a one in the OPERAND LIST; otherwise, it places a zero in the OPERAND LIST. As shown in Figure 23C, the machine first sets register SD to zero. It then fetches the two operands from the OPERAND LIST. The machine operations performed to fetch these operands are described in Section 4.13.1. The first operand is stored in register SR and the second operand is stored in the accumulator. As shown in Figure 22D, once the operands are fetched, the machine transfers the outputs of terminals SUM to the accumulator. Since register SD contains zero, these outputs are the difference of the two operands. The machine next examines the accumulator and sets buffer register BR2 accordingly. It then stores the contents of register BR2 in the OPERAND LIST, erases the relational operator from register D1, and transfers to the logical expression sequence.

As shown in Figures 23A and 23B, if the program constituent decoder indicates that register S contains the delimiter ELSE, the machine examines the contents of register D1 to determine what action to take. Again, a zero causes the machine to fetch the top element on the DELIMITER STACK and store it in register D1. If register D1 contains an equal sign ($D1=13_8$) the machine transfers to the arithmetic expression sequence. If register D1 contains the delimiter THEN ($D1=63_8$), the machine examines the current flag, register F1. If F1 contains one, it is complemented; otherwise, the second flag on the FLAG STACK is fetched from memory M1 and examined. If this flag is one, the current flag is

complemented. The machine then replaces THEN with ELSE in register D1 and transfers to the initial point sequence.

If register D1 contains a relational operator, the machine reacts as described above. Any delimiter other than an equal sign, THEN, or a relational operator causes the machine to transfer to the error sequence.

4.16 Logical Expression Sequence

This sequence is shown in Figures 24A-C. The machine enters this sequence when it reaches the end of a logical expression. As shown in Figure 24A, the machine operations performed in this sequence depend upon the contents of register D1. A value of zero in register D1 causes the machine to fetch the top element on the DELIMITER STACK from memory M1 and store it in register D1. If register D1 contains the delimiter IF ($D1=31_8$), the logical expression just scanned is the test condition in a conditional statement and it should be followed by the delimiter THEN. Therefore, the machine examines the output of the program constituent decoder. If this decoder indicates that register S does not contain the delimiter THEN, the machine transfers to the error sequence. If register S does contain the delimiter THEN, the machine places a new flag on the FLAG STACK. As shown in Figure 24B, it first stores the current flag in memory M1. It then places a new flag in register F1. If F1 contains zero, the new flag is also zero. If F1 contains one however, the machine must examine the value of the logical expression to determine the value of the new flag. The value of the logical expression just scanned is the value of the top element in the OPERAND LIST. The machine fetches this operand and stores it in buffer register BR2. If this operand is a NAME TABLE address of a variable (BR2(0)=1), the machine uses the address to fetch the value of the variable from the NAME TABLE. It then examines the VAI field in subregister BR2(0-5). If this field contains 77_8, the variable has not been assigned a value and cannot be used as an operand. Therefore, the machine transfers to the error

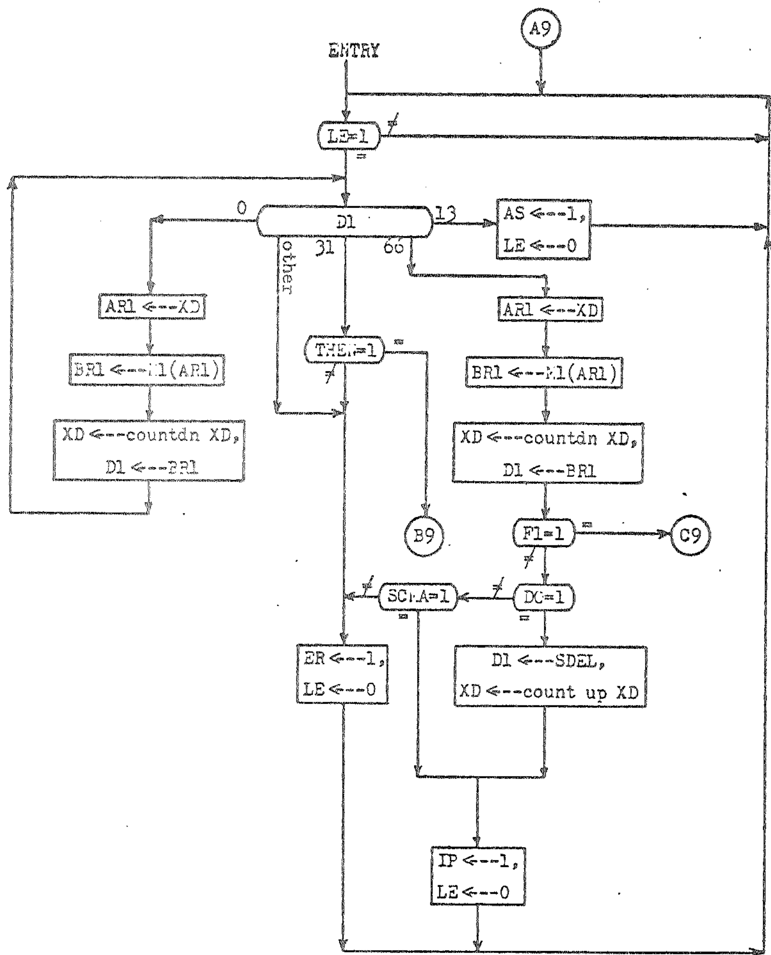


Figure 24a Logical Expression Sequence Chart

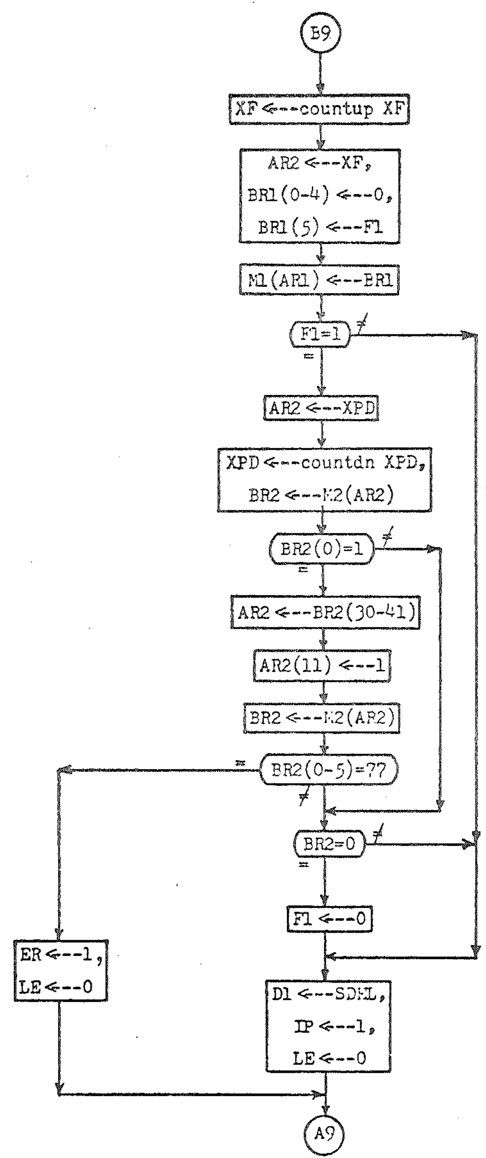


Figure 24b Logical Expression Sequence Chart

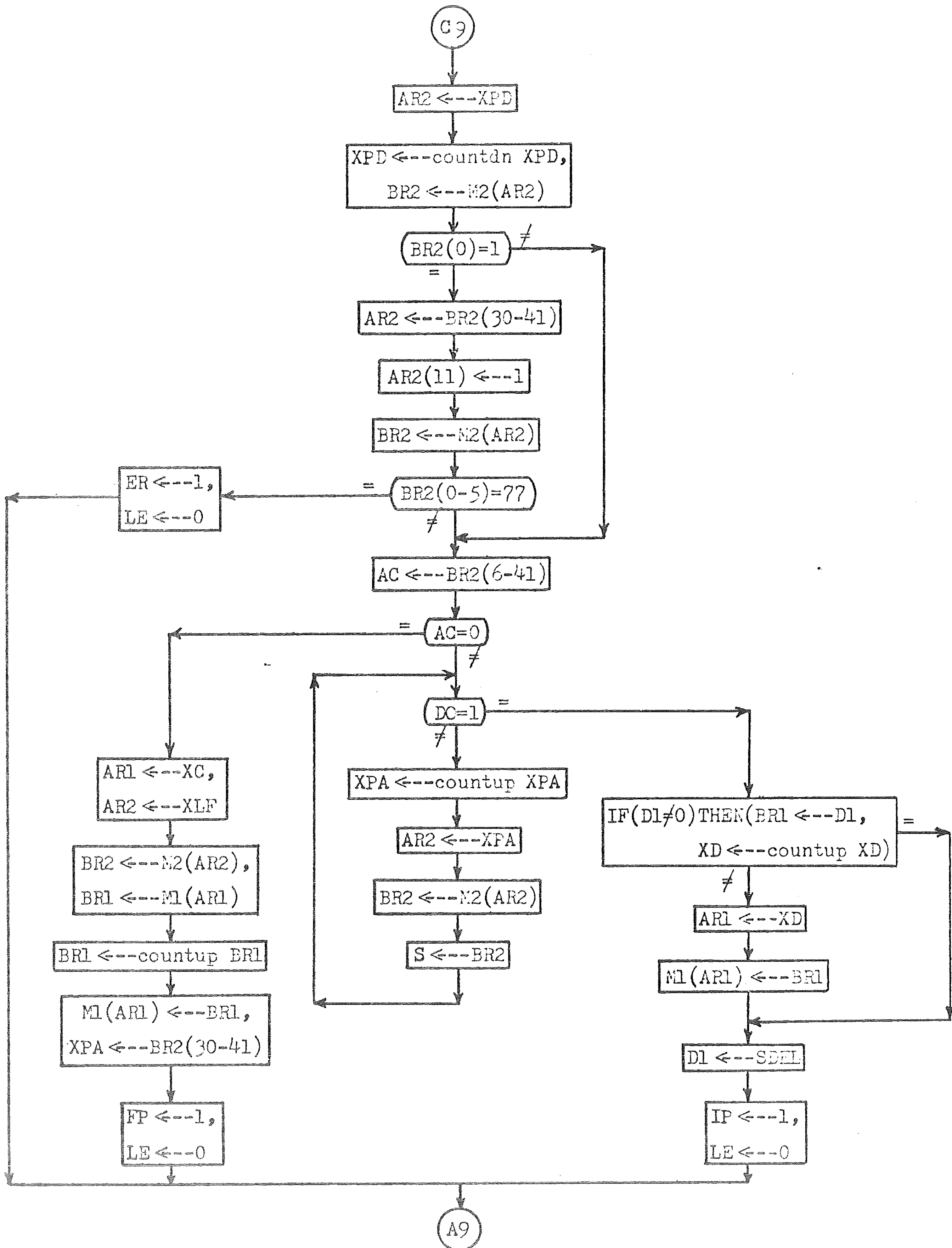


Figure 24c Logical Expression Sequence Chart

sequence. If the variable has been assigned a value or if the operand itself is a value, the machine sets the new flag. If the value is zero, the new flag is set to zero; otherwise, it is set to one. After setting the new flag, the machine replaces the delimiter IF with the delimiter THEN in register D1. It then transfers to the initial point sequence.

Referring again to Figure 24A, if register D1 contains the delimiter WHILE ($D1=66_8$), the logical expression just scanned is the test condition in a for list element. If the current flag, register F1, contains one, the machine must examine the value of the logical expression and determine whether or not to execute the statement contained in the iteration. If the current flag contains zero, however, the machine continues scanning the program without executing the statement.

As shown in Figure 24A, the machine removes the delimiter WHILE from the DELIMITER STACK by fetching the next delimiter on the stack from memory M1 and storing it in register D1. It then examines the current flag. If F1 contains zero, the machine examines the output of the program constituent decoder. If the decoder indicates that register S contains the delimiter DO, the machine places it on the DELIMITER STACK by transferring it to register D1. It also increments the DELIMITER STACK address in register XD by one. This effectively restores the delimiter in register D1 back into memory M1. The machine then transfers to the initial point sequence. A comma in register S (SCMA is true) causes the machine to transfer directly to the initial point sequence while any other program constituent causes the machine to transfer to the error sequence.

As shown in Figure 24C, if register F1 contains one, the machine fetches the top element on the OPERAND LIST from memory M2 and stores it in register BR2. If this operand is a NAME TABLE address of a variable (BR2(0)=1), the machine uses the address to fetch the value of the variable from the NAME

TABLE. It then examines the VAI field in subregister BR2(0-5). If this field contains 77_8, the variable has not been assigned a value and cannot be used as an operand. Therefore, the machine transfers to the error sequence. If the variable has been assigned a value or if the operand itself is a value, the machine stores the value in the accumulator for examination. This value is the result of the logical expression and if it is zero, the execution of the iteration for this for list element is complete. Therefore, the machine fetches the starting address of the iteration's for list from the LINK TO FORLIST STACK in memory M2, enters it into index register XPA as the new PROGRAM AREA address, and then transfers to the iteration control sequence. At the same time, it updates the for list element count for the iteration by fetching the count from the COUNT STACK in memory M1, incrementing this count by one, and returning it to the COUNT STACK. If register AC does not contain zero, the statement contained within the iteration is to be executed. Therefore, the machine scans to the end of the iteration's for list by sequentially fetching program constituents from the PROGRAM AREA of memory M2 until it recognizes the delimiter DO. At that time the machine places the delimiter on the DELIMITER STACK and transfers to the initial point sequence.

As shown in Figure 24A, if register D1 contains an equal sign ($D1=13_8$), the machine transfers to the assignment sequence. If register D1 contains a delimiter other than IF, WHILE, or an equal sign, the machine transfers to the error sequence.

4.17 Arithmetic Expression Sequence

This sequence is shown in Figures 25A-I. The machine enters this sequence when it reaches the end of an arithmetic expression. As shown in Figure 25A, the machine operations performed in this sequence depend upon the contents of register D1. A value of zero in register D1 causes the machine to fetch the top element on the DELIMITER STACK from memory M1 and store it in

register D1. If register D1 contains the delimiter ELSE ($D1=65_8$), the machine is completing a conditional statement and the flag associated with the conditional statement is no longer needed. Therefore, the next flag on the FLAG STACK is fetched from memory M1 and stored in register F1. This flag becomes the current flag. The machine then fetches the next element on the DELIMITER STACK and stores it in register D1.

If register D1 contains the delimiter FOR ($D1=26_8$), the arithmetic expression occurs in a for list element. The machine must determine whether the arithmetic expression is the entire for list element or whether it is the first arithmetic expression in a for list element of the type STEP-UNTIL or the arithmetic expression in a for list element of the type WHILE. As shown in Figure 25E, the machine determines this by examining the output of the program constituent decoder. If this decoder indicates register S contains a comma (SCMA is true) or the delimiter DO, the arithmetic expression is the entire for list element. If register S contains DO, this for list element is the last element in the iteration's for list and the machine erases the delimiter FOR from the DELIMITER STACK by fetching the next delimiter from memory M1 and storing it in register D1. The machine then examines the current flag. If register F1 contains zero, the machine is only scanning the program and transfers to the initial point sequence; otherwise it acts to execute the statement contained in the iteration.

As shown in Figure 25F, the machine fetches the top element on the OPERAND LIST from memory M2. If this operand is a NAME TABLE address of a variable (BR2(0)=1), the machine uses the address to fetch the value of the variable. If no value has been assigned to the variable, the machine transfers to the error sequence; otherwise, it stores the value in the accumulator. This value is the value of the arithmetic expression and it is to be assigned to the iteration's controlled variable. The machine assigns the value to this

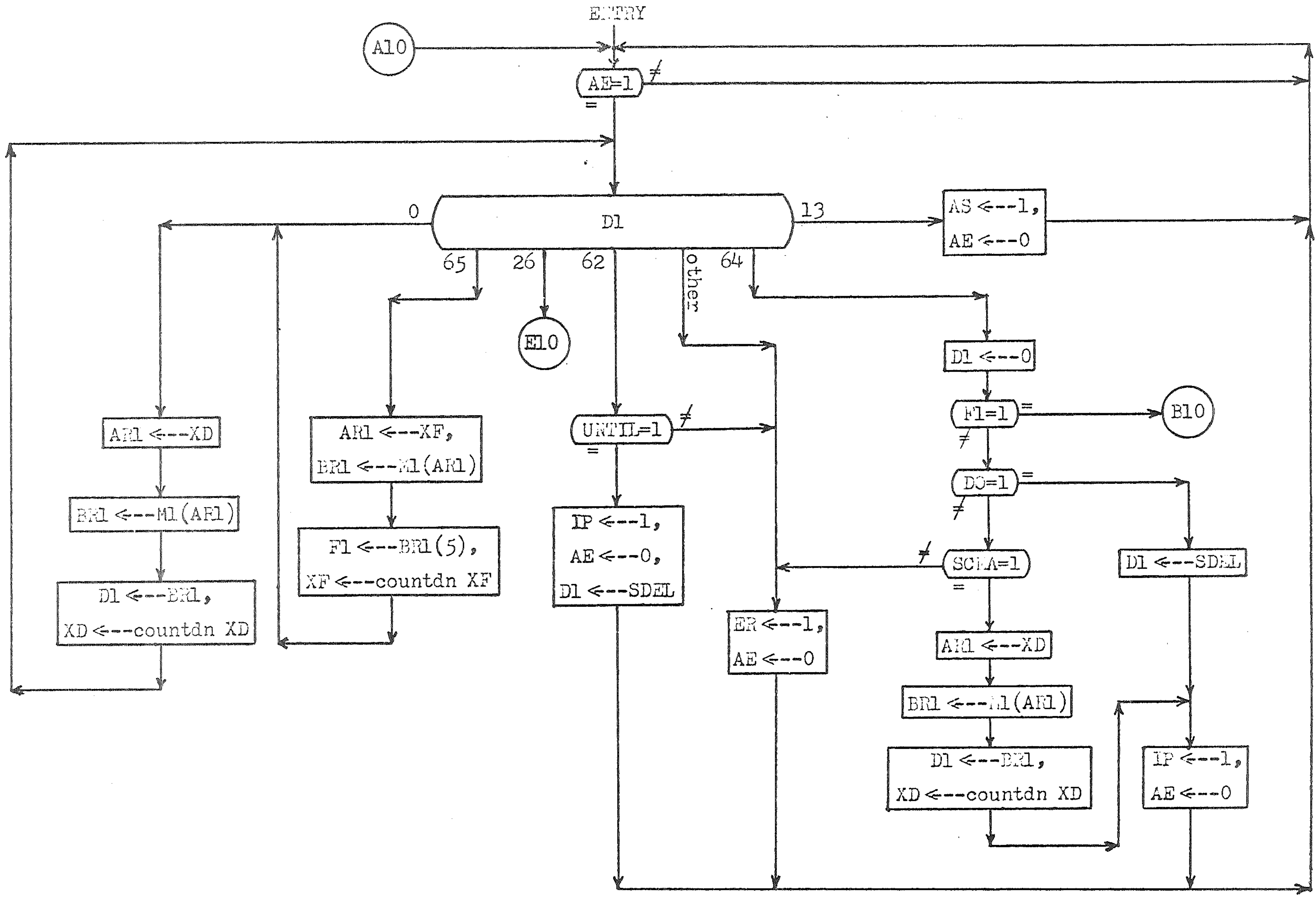


Figure 25a Arithmetic Expression Sequence Chart

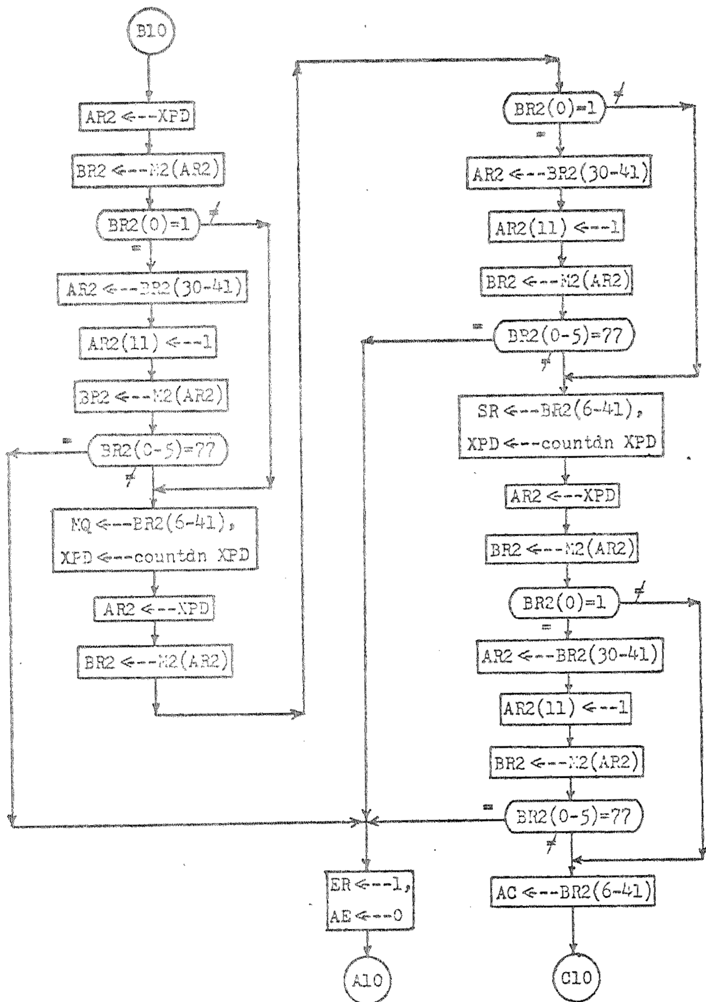


Figure 25b Arithmetic Expression Sequence Chart

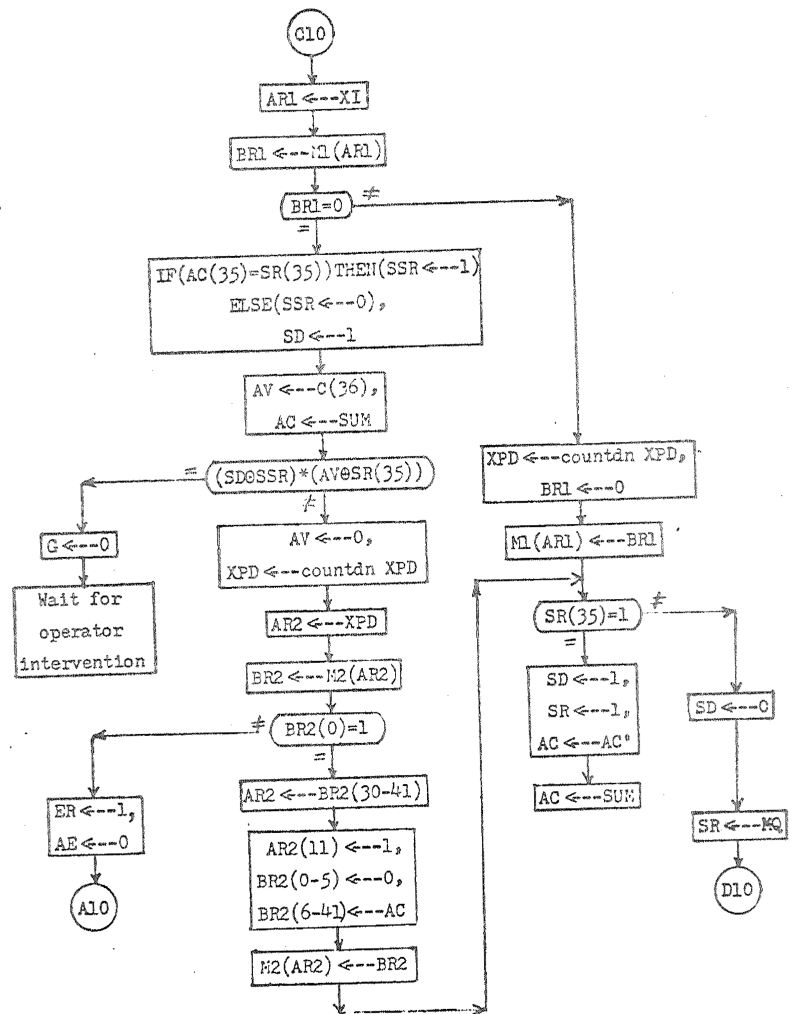


Figure 25c Arithmetic Expression Sequence Chart

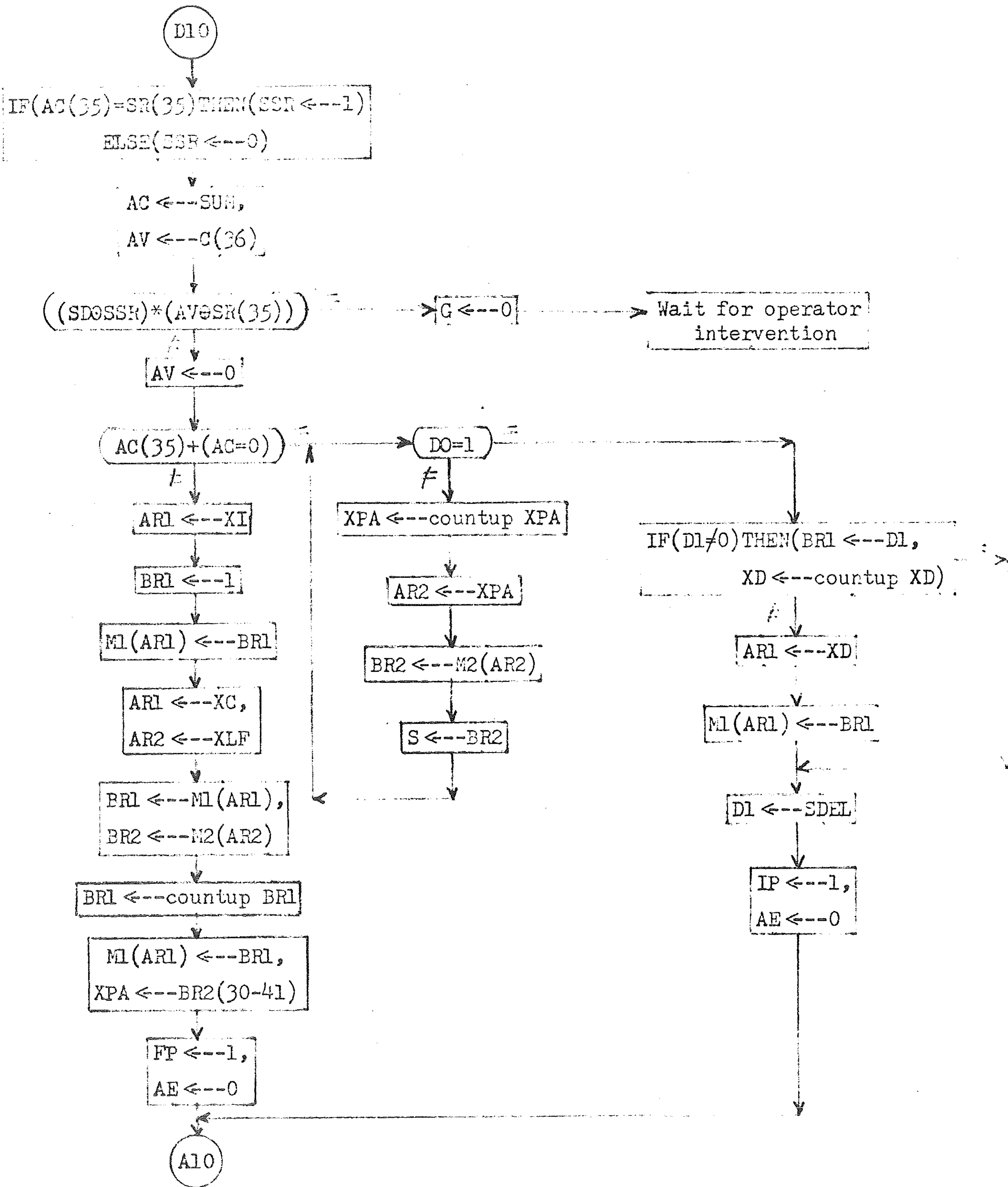


Figure 25d Arithmetic Expression Sequence Chart

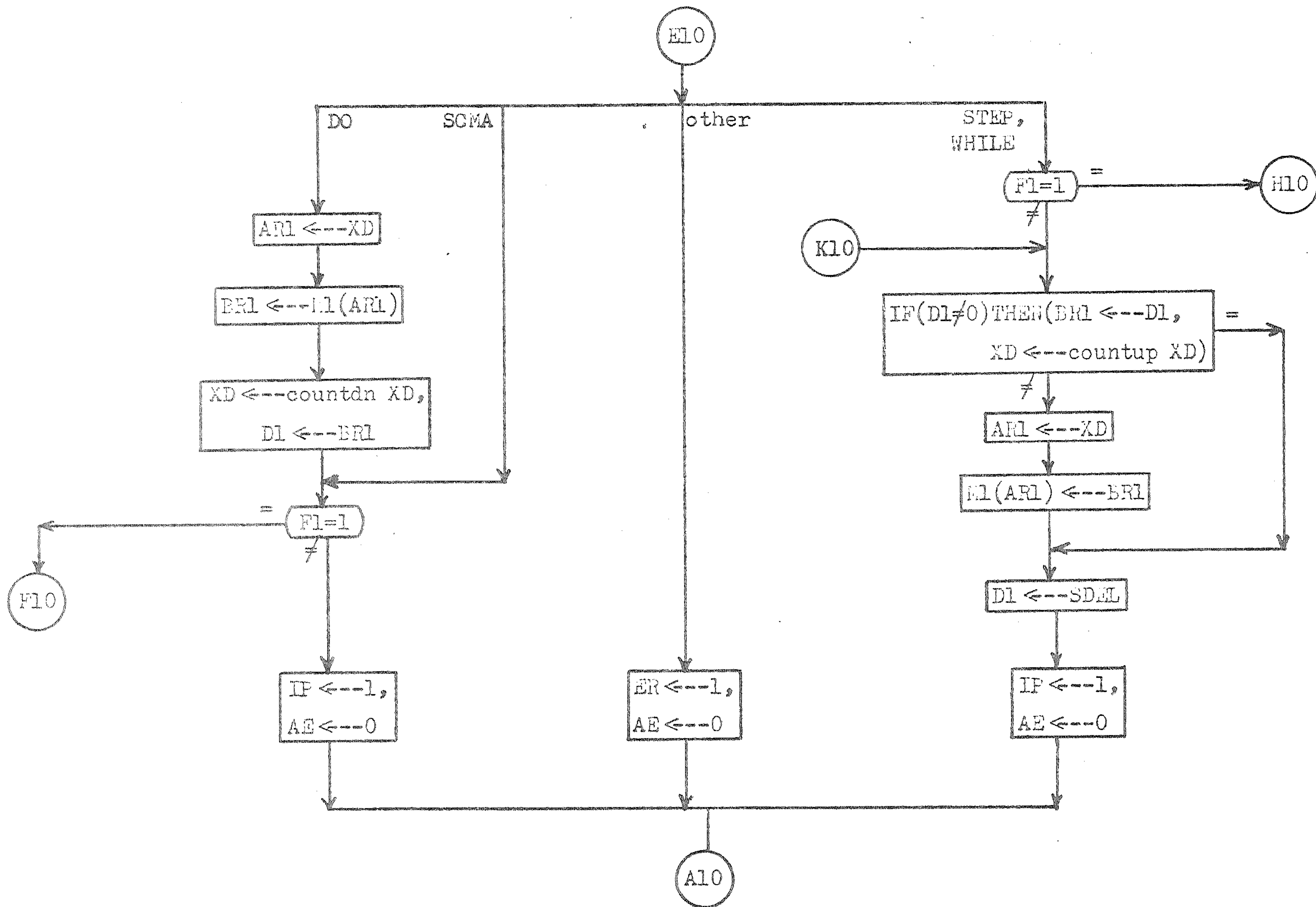


Figure 25e Arithmetic Expression Sequence Chart

F10
AR2 <--XPD
BR2 <--M2(AR2)
BR2(0)=1
AR2 <--BR2(30-41)
AR2(11) <--1
BR2 <--M2(AR2)
BR2(0-5)=77
AC <--BR2(6-41),
XPD <--countdn XPD
AR2 <--XPD
BR2 <--M2(AR2)
BR2(0)=1
AR2 <--BR2(30-41)
AR2(11) <--1,
BR2(0-5) <--0,
BR2(6-41) <--AC
M2(AR2) <--BR2
G10
ER <--1,
AE <--0
A10

Figure 25f Arithmetic Expression Sequence Chart

G10
DC=1
AR1 <--XC
BR1 <--M1(AR1)
BR1 <--countup BR1
M1(AR1) <--BR1
XPA <--countup XPA
AR2 <--XPA
BR2 <--M2(AR2)
S <--BR2
DC=1
IF(D1≠0)THEN(BR1 <--D1,
XD <--countup XD)
AR1 <--XD
M1(AR1) <--BR1
D1 <--SDEL
XC <--countdn XC,
XI <--countdn XI,
XLF <--countdn XLF,
XPD <--countdn XPD
IF(XC=165)THEN(IT <--0)
IP <--1,
AE <--0
A10

Figure 25g Arithmetic Expression Sequence Chart

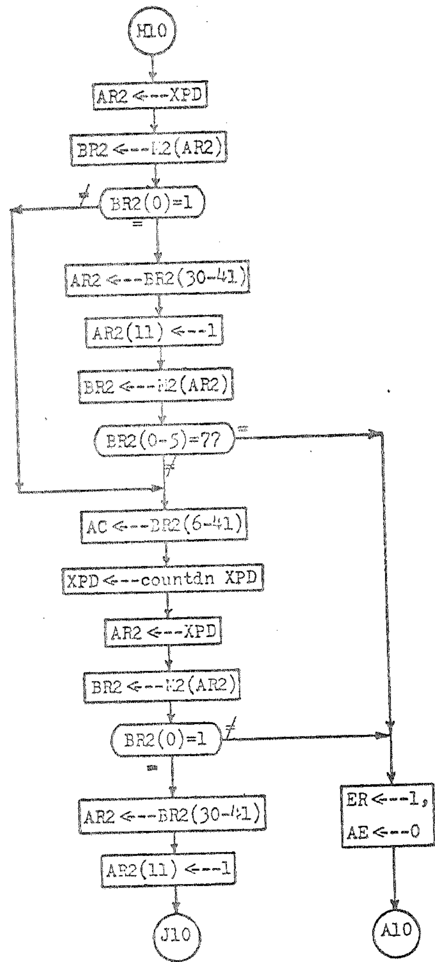


Figure 25h Arithmetic Expression Sequence Chart

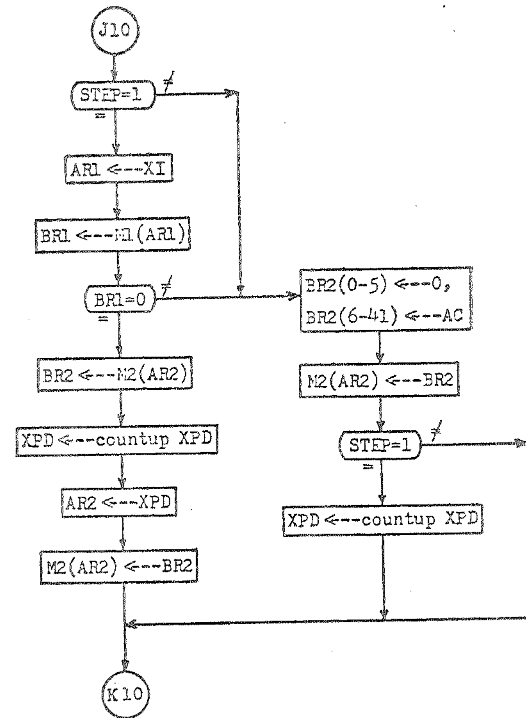


Figure 25i Arithmetic Expression Sequence Chart

variable by fetching the variable's NAME TABLE address from the OPERAND LIST and then using this address to store the value in the NAME TABLE. The machine then scans to the beginning of the statement contained in the iteration as shown in Figure 25G. If the program constituent decoder indicates register S contains DO, the machine is already at the beginning of the statement. Also, as stated earlier, this for list element is the last element of the iteration's for list. Therefore, the machine decrements the COUNT STACK, the INITIAL STACK, and the LINK TO FORLIST STACK addresses by one. This effectively erases the top element on these stacks. The addresses of these stacks are located in index registers XC, XI, and XLF respectively. The machine next examines the COUNT STACK address. If this address indicates the COUNT STACK is empty, the machine sets register IT to zero. It then transfers to the initial point sequence. If register S does not contain the delimiter DO, the machine increments the count in the top element on the COUNT STACK by one. It then scans through the for list by sequentially fetching program constituents from memory M2 until it finds delimiter DO. It places DO on the DELIMITER STACK and then transfers to the initial point sequence.

Referring again to Figure 25E, if register S contains either the delimiter STEP or the delimiter WHILE, the machine examines the current flag. If F1 contains zero, the machine places the delimiter in register S on the DELIMITER STACK and then transfers to the initial point sequence. If F1 contains one, the machine fetches the value of the arithmetic expression from the OPERAND LIST as shown in Figure 25H. The machine stores this value in the accumulator. It then fetches the NAME TABLE address of the iteration controlled variable from the OPERAND LIST and stores the address in register AR2. As shown in Figure 25I, the machine then determines whether or not register S contains the delimiter STEP. If register S does contain STEP, the machine fetches the top element on the INITIAL STACK from memory M1 and examines it. If register S does not con-

tain STEP or if the top element on the INITIAL STACK is not zero, the machine transfers the value of the arithmetic expression to buffer register BR2 and then stores it in the NAME TABLE. This action assigns the value to the iteration controlled variable. The machine again determines whether or not register S contains STEP. If register S does, the machine increments the OPERAND LIST address by one. This effectively restores the arithmetic expression in the OPERAND LIST. The machine then places the delimiter in register S on the DELIMITER STACK as shown in Figure 25E.

If register S contains STEP and the top element on the INITIAL STACK is zero, the machine fetches the value of the iteration controlled variable from the NAME TABLE. It then increments the OPERAND LIST address by one and stores this value in the OPERAND LIST. This value replaces the arithmetic expression just scanned. The machine then places the delimiter STEP on the DELIMITER STACK as shown in Figure 25E.

Referring again to Figure 25A, if D1 contains the delimiter STEP ($D1=62_8$), the arithmetic expression just scanned is the second arithmetic expression in a for list element of the type STEP-UNTIL. This arithmetic expression should be followed by the delimiter UNTIL. Therefore, the machine examines the output of the program constituent decoder. If the decoder indicates that register S does not contain UNTIL, the machine transfers to the error sequence. If register S does contain UNTIL, the machine places it on the DELIMITER STACK by transferring it to register D1. This action also erases the delimiter STEP from the DELIMITER STACK. The machine then transfers to the initial point sequence.

If register D1 contains the delimiter UNTIL ($D1=64_8$), the arithmetic expression just scanned is the third arithmetic expression in a for list element of the type STEP-UNTIL. As shown in Figure 25A, the machine erases UNTIL from the DELIMITER STACK by setting register D1 to zero. It then examines the cur-

rent flag. If register F1 contains zero, the machine examines the output of the program constituent decoder. If the decoder indicates register S contains the delimiter DO, the machine places the delimiter on the DELIMITER STACK by transferring it to register D1. The machine then transfers to the initial point sequence. If the decoder indicates register S contains a comma, the machine fetches the top element on the DELIMITER STACK from memory M1, stores it in register D1, and then transfers to the initial point sequence.

If register F1 contains one, the machine performs a test to determine whether the statement contained in the iteration should be executed or whether execution under control of this for list element is complete. As shown in Figure 25B, the machine fetches the values of the three arithmetic expressions from the OPERAND LIST. It stores the value of the third arithmetic expression in register MQ, the value of the second arithmetic expression in register SR, and the value of the first arithmetic expression in the accumulator. The machine operations performed to fetch these values are described in Section 4.13.1.

As shown in Figure 25C, the machine next fetches the top element on the INITAL STACK from memory M1 for examination. If this element contains one, the machine has not processed this STEP-UNTIL for list element previously and the iteration's controlled variable is not to be "stepped" at this time. The machine then sets the top INITIAL STACK element to zero. If the top element on the INITIAL STACK is zero, the machine increments the value in the accumulator by the value in register SR. It should be noted that since the top element of the INITIAL STACK is zero, the value in the accumulator is the current value of the iteration's controlled variable. The machine then tests for overflow. If overflow has occurred, the machine sets run/stop control flipflop G to zero and then waits for the operator to intervene. If there is no overflow, the machine fetches the NAME TABLE address of the iteration's controlled variable from the OPERAND LIST, transfers this address to address register AR2, and then

assigns the value in the accumulator to the controlled variable. Thus, the value of the iteration's controlled variable is "stepped" by the value of the second arithmetic expression in the for list element.

The machine then determies the value of the expression A*sign B - C; where A represents the iteration's controlled variable and B and C represent the second and third arithmetic expressions in the for list element. The machine first examines the sign bit of the second arithmetic expression in subregister SR(35). If the value of this expression is negative, the machine complements the value of the iteration's controlled variable in the accumulator. It also sets register SD to one. If the value of the second arithmetic expression is positive, the machine sets register SD to zero. The machine then transfers the value of the third arithmetic expression from register MQ to register SR. As shown in Figure 25D, if the sign bits in subregisters AC(35) and SR(35) are equal, the machine sets register SSR to one; otherwise, it sets register SSR to zero. The contents of this register are used to test for overflow. The machine then transfers the value on terminals SUM to the accumulator. These terminals are the outputs of the parallel adder-subtractor wired to registers AC and SR. The machine then tests for overflow. If overflow has occurred the machine sets run/stop control register G to zero and then waits for the operator to intervene. If there is no overflow, the machine sets register AV to zero and then examines the contents of the accumulator. At this time the accumulator contains the results of the previously described expression; A*sign B - C. If the result of this expression is less than or equal to zero, the machine acts to execute the statement contained in the iteration. The machine scans to the end of the iteration's for list by sequentially fetching program constituents from the PROGRAM AREA of memory M2 until it recognizes the delimiter DO. At that time the machine places the delimiter on the DELIMITER STACK and transfers

to the initial point sequence. If the result of the expression is greater than zero, the execution of the iteration is complete for this for list element. Therefore, the machine resets the top element on the INITIAL STACK to one. It then fetches the starting address of the iteration's for list from the LINK TO FORLIST STACK in memory M2 and places it in index register XPA. It also updates the for list element count for the iteration by fetching the count from the COUNT STACK in memory M1, incrementing the count by one, and returning the count to the COUNT STACK. The machine then transfers to the iteration control sequence.

Referring again to Figure 25A, if register D1 contains an equal sign ($D1=13_8$), the machine transfers to the assignment sequence. If register D1 contains any delimiter other than ELSE, FOR, STEP, UNTIL, or an equal sign, the machine transfers to the error sequence.

4.18 Assignment Sequence

This sequence is shown in Figure 26. Upon entering this sequence the machine examines the contents of register D1. If register D1 contains an equal sign ($D1=13_8$) the machine examines the current flag. If F1 contains a one, the machine assigns the value of the top element in the OPERAND LIST to the variable whose NAME TABLE address is the second element in the OPERAND LIST. As shown in Figure 26, the machine fetches the first operand from the OPERAND LIST and stores it in the accumulator. The machine operations performed to fetch the operand are described in Section 4.13.1. It then fetches the second operand. If this operand is not a NAME TABLE address of a variable ($BR2(0)\neq1$), the machine transfers to the error sequence. If it is a NAME TABLE address, the machine transfers it to address register AR2. It then increments the address by one and transfers the contents of the accumulator to subregister BR2(6-41). It should be noted that subregister BR2(0-5) is set to zero. This action erases

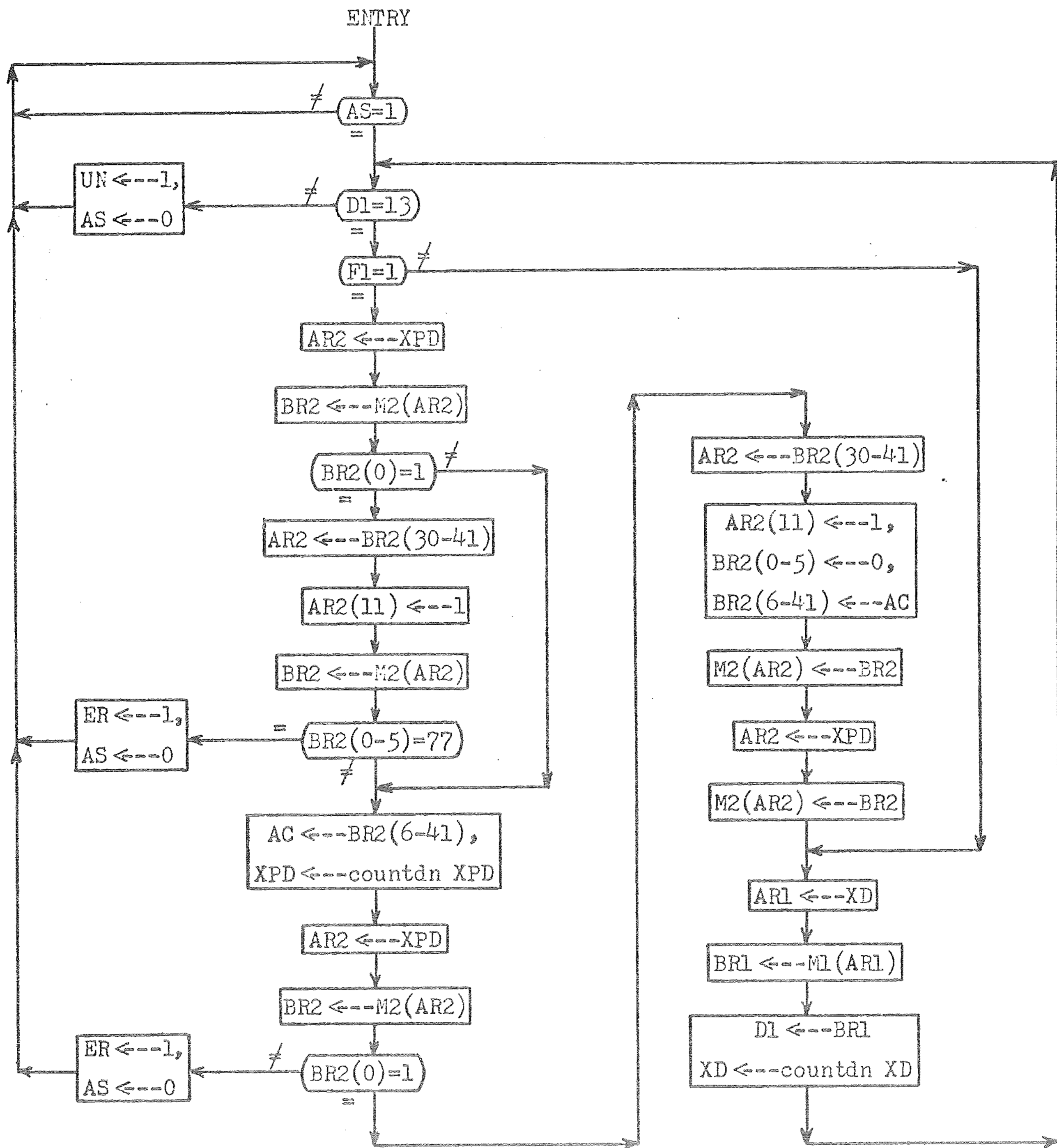


Figure 26 Assignment Sequence Chart

the code 77_8 from the VAI field. The contents of register BR2 are then stored in the NAME TABLE; thus, assigning the value to the variable. The value is also placed back in the OPERAND LIST. The machine then fetches the next element on the DELIMITER STACK from memory M1 and stores it in register D1. It then examines the contents of register D1 to determine whether or not another variable is to receive the value. If register D1 does not contain an equal sign, the machine transfers to the unconditional statement sequence.

4.19 Unconditional Statement Sequence

As shown in Figure 27, upon entering this sequence, the machine examines the contents of register D1. A value of zero causes the machine to fetch the top element on the DELIMITER STACK from memory M1 and store it in register D1. If D1 contains the delimiter THEN(D1=63_8), the machine examines the output of the program constituent decoder. If the decoder indicates that register S contains the delimiter ELSE, the machine examines the current flag, register F1. If F1 contains one, it is complemented; otherwise, the second flag on the FLAG STACK is fetched from memory M1 and examined. If this flag is one, the current flag is complemented. The machine then replaces THEN with ELSE in register D1 and transfers to the initial point sequence. If register D1 contains any other delimiter, the machine transfers to the end of statement sequence.

If the program constituent decoder indicates that register S contains either an end of statement symbol (SDLR is true) or the delimiter END, a conditional statement has just been completed and the flag placed on the FLAG STACK at the start of the statment is no longer required. Therefore, the machine fetches the second flag from memory M1 and stores it in register F1. This action erases the unneeded flag and causes the second flag to become the current flag. The machine then transfers to the end of statement sequence.

Any output of the program constituent decoder other than the end of statement symbol or the delimiters ELSE and END causes the machine to transfer to

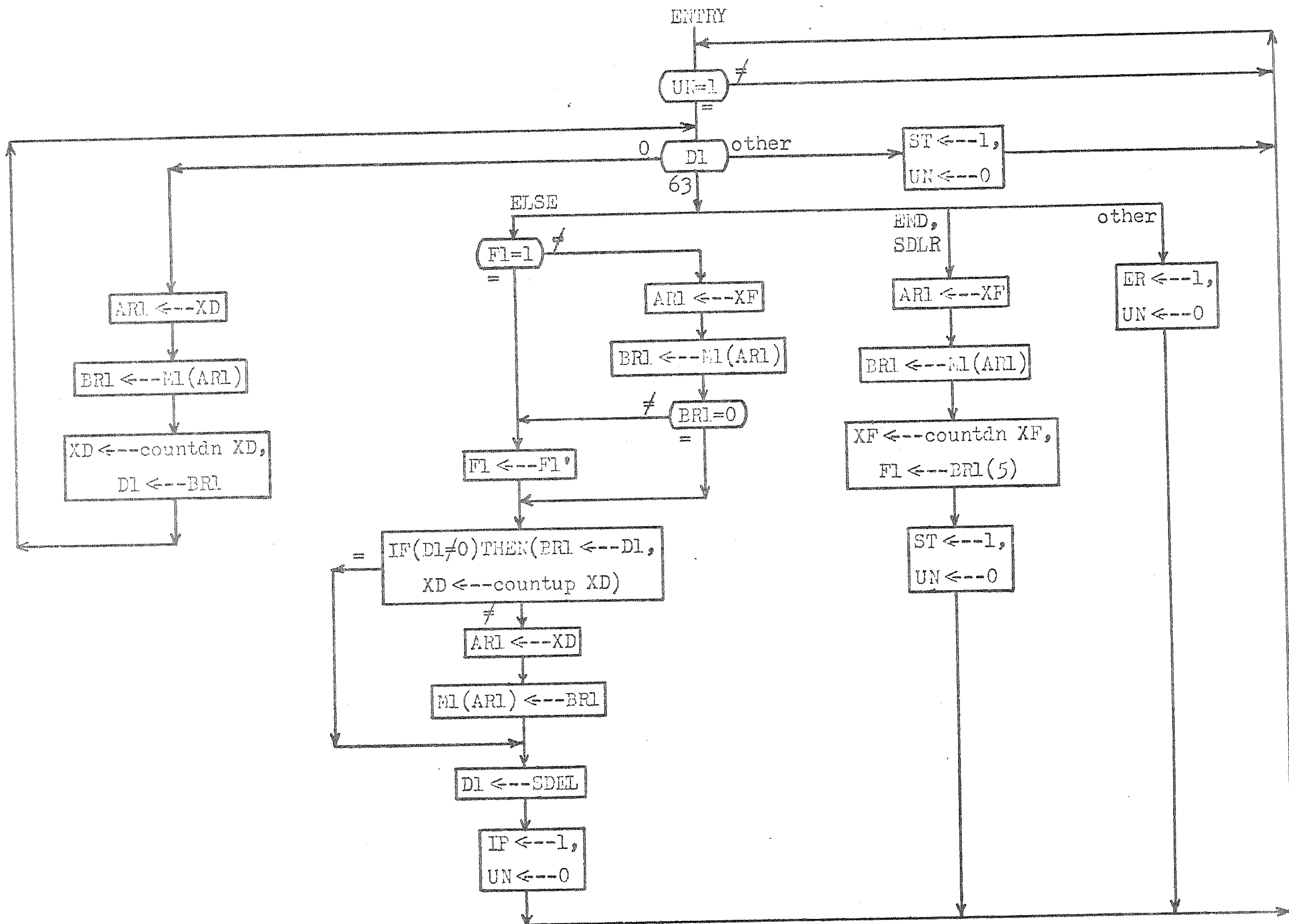


Figure 27 Unconditional Statement Sequence Chart

the error sequence.

4.20 End of Statement Sequence

This sequence is shown in Figures 28A and 28B. Upon entering this sequence, the machine examines the contents of register D1. If register D1 contains a zero, the machine fetches the top element on the DELIMITER STACK from memory M1 and then reexamines register D1. If D1 contains the delimiter ELSE ($D1=65_8$), the machine erases it from the DELIMITER STACK by fetching the next delimiter from memory M1. It also erases the current flag from register F1 by fetching the next flag from memory M1. It then reexamines register D1 to determine whether or not the next element on the DELIMITER STACK is also the delimiter ELSE.

If register D1 contains the delimiter DO ($D1=24_8$), the machine erases this delimiter from the DELIMITER STACK by fetching the next delimiter from memory M1. It then reexamines the contents of register D1. If D1 contains the delimiter FOR, the machine examines the current flag; otherwise, it transfers to the error sequence. If F1 contains a zero, the program is only being scanned and the machine is not required to return to the iteration's for list. Therefore, it sets register D1 to zero and decrements the COUNT STACK, INITIAL STACK, and LINK TO FORLIST STACK addresses (located in registers XC, XI, and XLF respectively) by one. This action returns these stacks to their states before the iteration was encountered. If register XC contains 165_8, the COUNT STACK is empty. This indicates that no iterations are being processed and register IT is set to zero. The machine then transfers to the program body sequence.

As shown in Figure 28B, if register F1 contains one, the machine removes operands from the OPERAND LIST one by one until it finds a NAME TABLE address (BR2(0)=1). If the program has been syntactically correct to this point, this

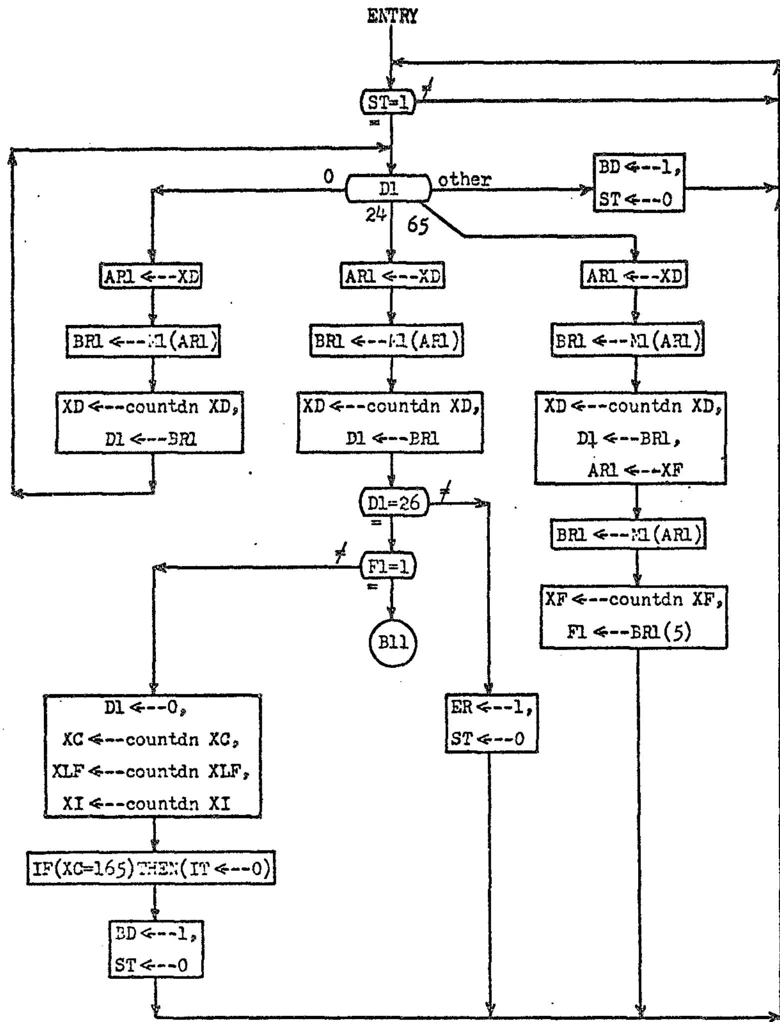


Figure 28a End of Statement Sequence Chart

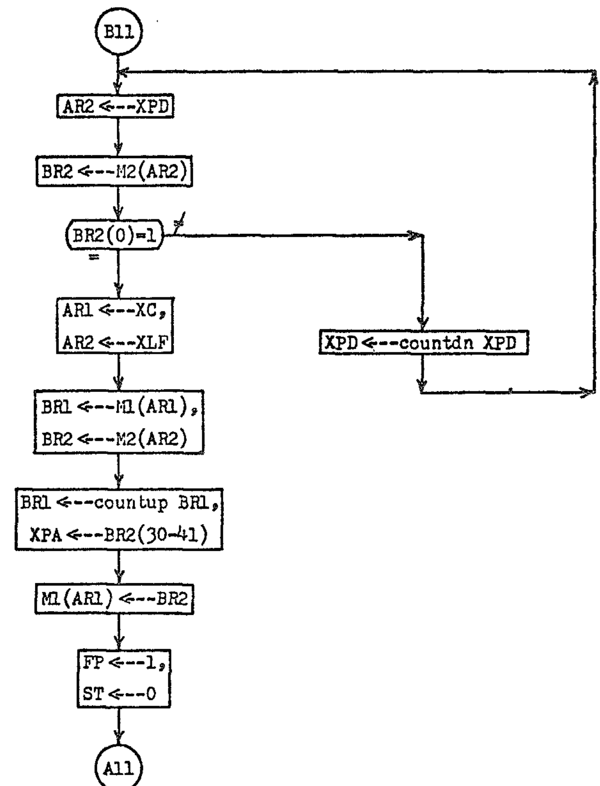


Figure 28b End of Statement Sequence Chart

operand is the NAME TABLE address of the iteration's controlled variable and the machine is set to return to the iteration's for list. It does this by fetching the list's starting address from the LINK TO FORLIST STACK and transferring it to index register XPA. At the same time, it fetches the for list element count for this for list from the COUNT STACK and increments it by one. It then transfers to the iteration control sequence.

4.21 Program Body Sequence

This sequence is shown in Figure 29. The machine operations in this sequence depend upon the output of the program constituent decoder. If this decoder indicates that register S contains an end of statement symbol (SDLR is true) the machine examines the iteration control register IT. If register IT contains a zero, no iterations are being processed and all the operands in the OPERAND LIST are no longer required by the program. Therefore, the machine sets the OPERAND LIST address in register XPD to 7741_8. This is the value the register contains when the OPERAND LIST is empty. The machine then transfers to the initial point sequence.

If register IT contains a non-zero value, at least one iteration is being processed and the NAME TABLE address(es) of the iteration controlled variable(s) that is stored in the OPERAND LIST is still required by the program. If the program has been syntactically correct to this point, all elements in the OPERAND LIST that were entered after the iteration controlled variable(s) should be values rather than NAME TABLE addresses. Therefore, the machine removes the elements one by one until it finds a NAME TABLE address. A NAME TABLE address is recognized when a one appears in subregister BR2(0). The machine then transfers to the initial point sequence.

If register S contains the delimiter END when this sequence is entered, the machine transfers to the block exit sequence. If register S contains any

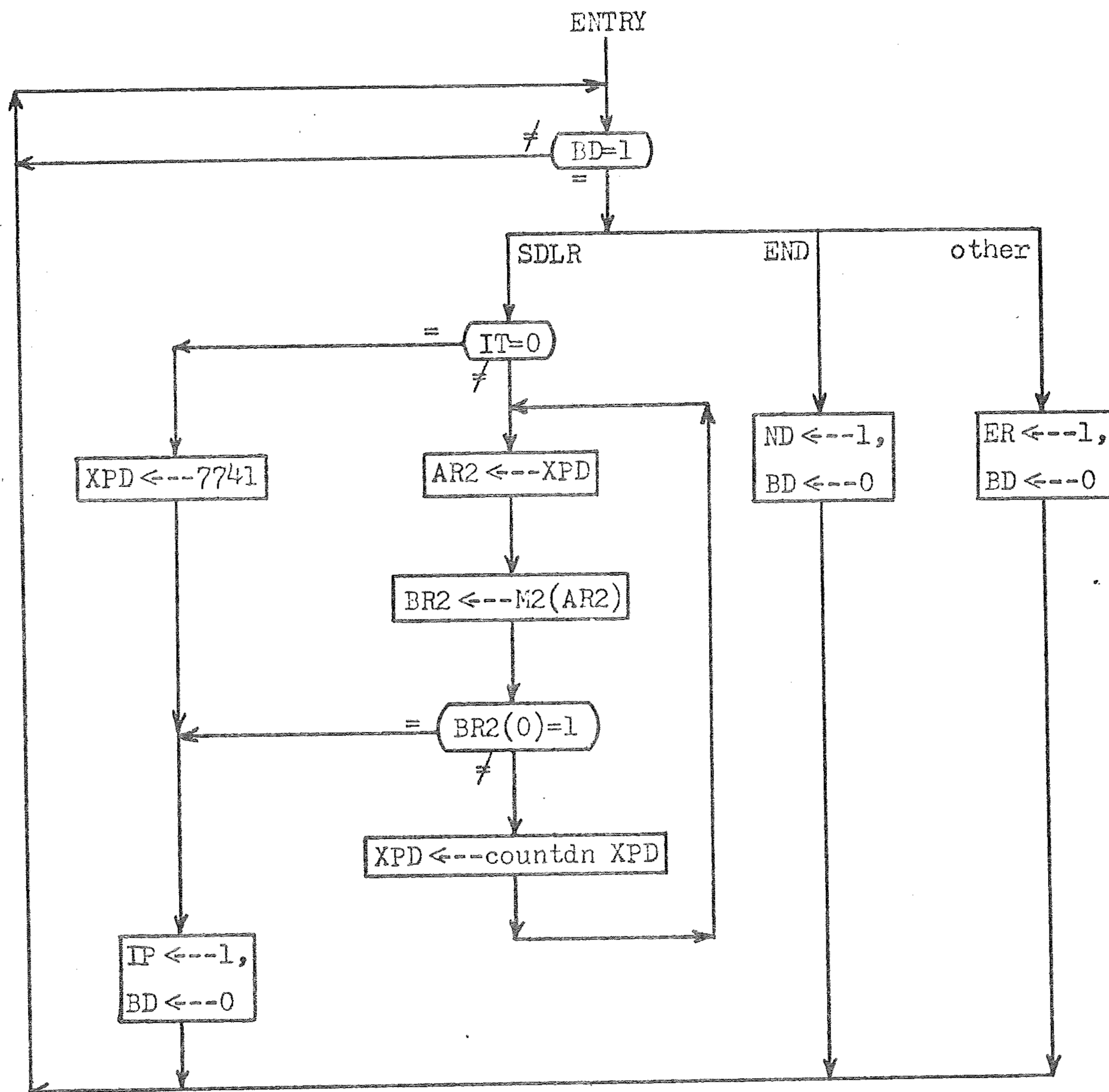


Figure 29 Program Body Sequence Chart

other program constituent, the machine transfers to the error sequence.

4.22 Iteration Control Sequence

This sequence is shown in Figure 30A and 30B. Prior to entering this sequence, the machine sets the PROGRAM AREA address register, XPA, to the start of the iteration's for list. In this sequence the machine scans the for list until it reaches the beginning of the for list element that is currently controlling execution of the iteration. It then transfers to the initial point sequence to process the for list element. The machine determines which for list element is in control by counting the for list elements as it scans and comparing this count with the value of the top element on the COUNT STACK. If the end of the for list is reached before the count equals the value on the COUNT STACK, the processing of the iteration is complete. Therefore, the machine restores its stacks and registers to their contents prior to encountering the iteration. It then scans to the end of the iteration and continues processing the program.

As shown in Figure 30A, upon entering this sequence, the machine transfers the current COUNT STACK address in index register XC to address register AR1. It then fetches the top element on the stack and examines it. If its value is one, the first for list element in the for list is currently controlling the iteration. Since the PROGRAM AREA address register is already set to the beginning of this for list element, the machine transfers immediately to the initial point sequence. If the value of the top element of the COUNT STACK is greater than one, the machine sets register CC to one. It then fetches a program constituent from the PROGRAM AREA of memory M2 and stores it in register S. If the program constituent decoder indicates that register S contains a comma (SCMA is true), the machine increments the count in register CC by one and compares this count to the value of the top element on the COUNT STACK. If

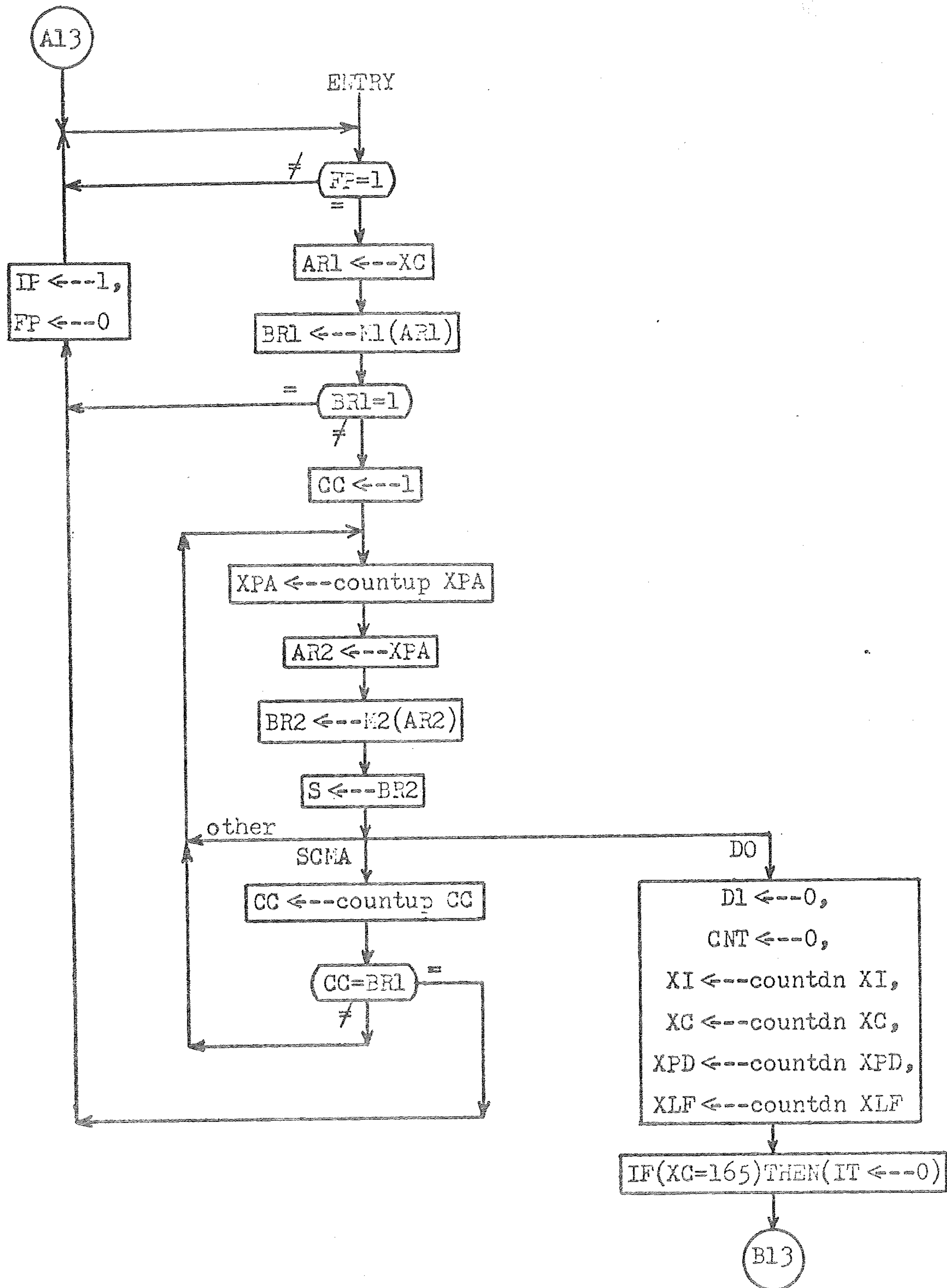


Figure 30a Iteration Control Sequence Chart

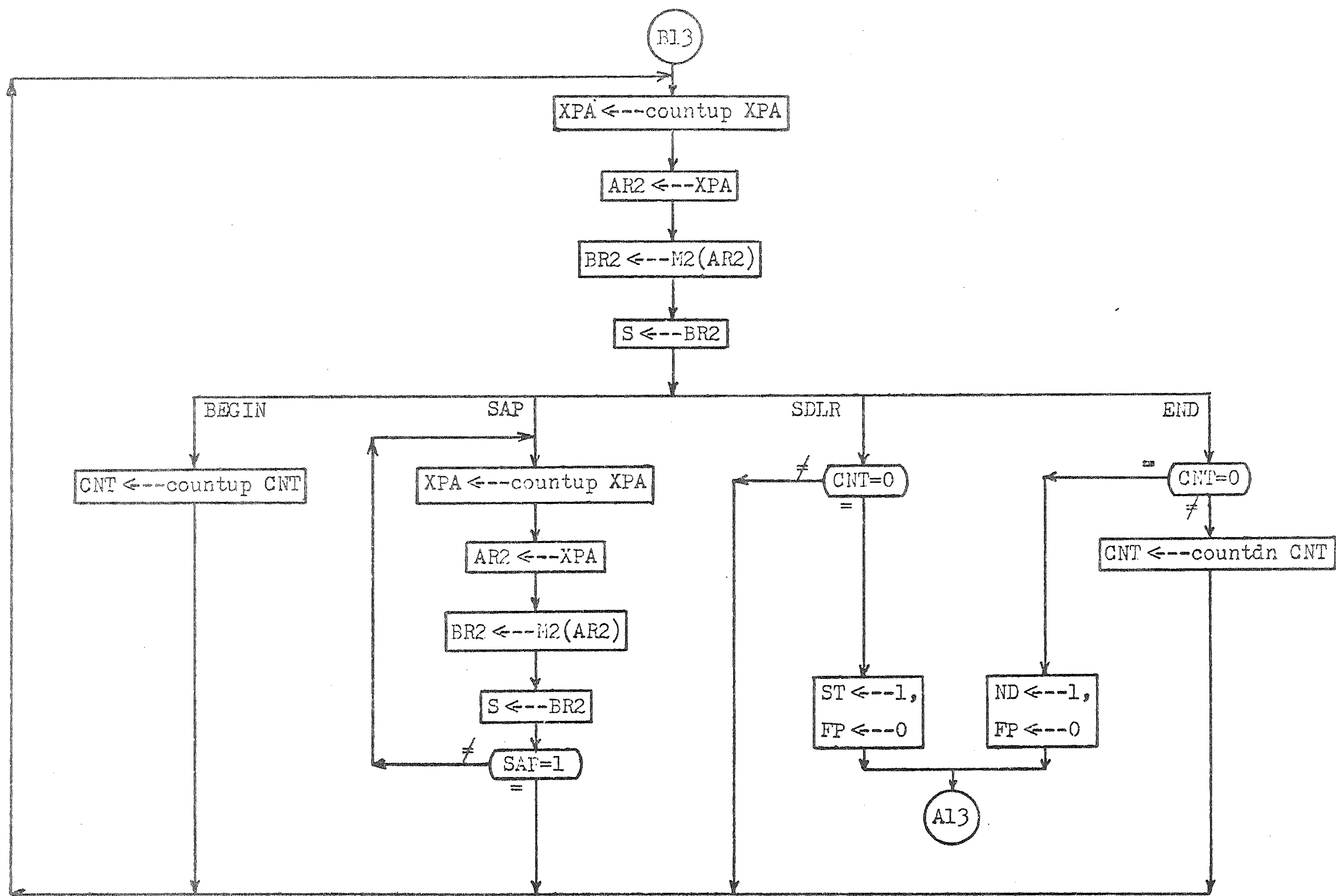


Figure 30b Iteration Control Sequence Chart

the two counts are equal the machine transfers to the initial point sequence; otherwise, it continues to scan the for list. If the decoder indicates that register S contains the delimiter DO, the end of the for list has been reached and processing of the iteration is complete. The machine then erases the delimiter FOR from the DELIMITER STACK by setting register D1 to zero. It also sets register CNT to zero and decrements the OPERAND LIST, COUNT STACK, INITIAL STACK, and LINK TO FORLIST STACK addresses by one. It then examines the COUNT STACK address. If this address indicates the COUNT STACK is empty, register IT is set to zero.

After setting register CNT to zero, the machine is ready to scan to the end of the iteration. As shown in Figure 30B, it scansthrough the program until it finds an end of statement symbol ($) occurring at the current block level or the delimiter END which signifies the end of the block containing the iteration. It sequentially fetches the program constituents from the PROGRAM AREA, stores them in register S, and identifies them. If a $ is found (SDLR is true), the machine determines whether or not it occurs at the current block level by the following method. If the delimiter BEGIN is fetched into register S, the contents of register CNT are incremented by one. If the delimiter END is fetched into register S, the contents of register CNT are examined and if they are greater than zero, they are decremented by one. Each time a $ is fetched into register S the contents of register CNT are examined. If register CNT has a value greater than zero, the $ occurs at a higher block level rather than at the current block level and it is ignored. If register CNT contains zero, the $ occurs at the current block level and the scan terminates. The machine then transfers to the end of statement sequence. When the delimiter END is fetched and the value in register CNT is zero, the machine transfers to the block exit sequence.

Whenever an apostrophe is fetched into register S (SAP is true), the

machine continues scanning without reacting to the constituents until a second apostrophe is found. Therefore any $ which occur in an output character string go unrecognized.

4.23 Read/Write Execution Sequence

This sequence is shown in Figures 31A-C. As shown in Figure 31A, upon entering this sequence, the machine examines the contents of register D1. If register D1 does not contain an input-output operator, READ or WRITE, the machine transfers to the error sequence; otherwise, it continues in this sequence. The input-output operations are described below.

4.23.1 Writing

When register D1 contains the delimiter WRITE ($D1=47_8$), the machine examines the current flag. If register F1 contains one, the machine sets index register XPV to the memory address of the location preceding the location of the first variable whose value is to be printed out. This location varies depending upon the number of iterations being processed at the time the WRITE statement is encountered. If no iteration is being processed (IT=0), register XPV is set to 7741_8. This is the address of the location preceding the OPERAND LIST locations in memory. This address causes the machine to print out the value of all the variables in the OPERAND LIST. When an iteration is being processed and register IT contains a value greater than one, the contents of register IT are transferred to register XPV. Register IT contains the address of the OPERAND LIST location containing the controlled variable of the most recently entered iteration. If register IT contains one, no variables were listed in the WRITE statement and register XPV is set to the contents of register XPD. As will be seen, this value causes the machine to immediately terminate the write operation; thus, unless the WRITE statement contains an output string, it is equivalent to an empty statement.

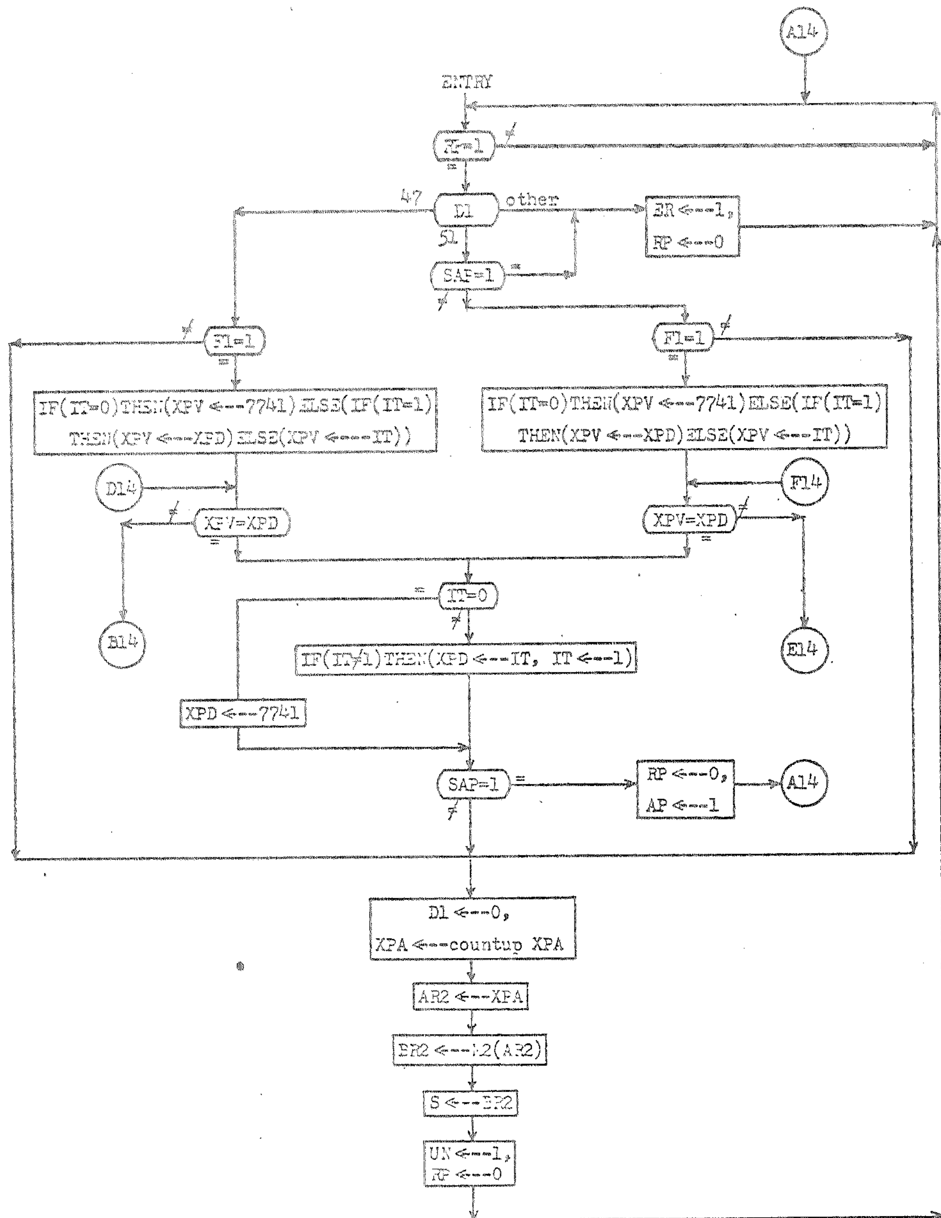


Figure 31a Read/Write Execution Sequence Chart

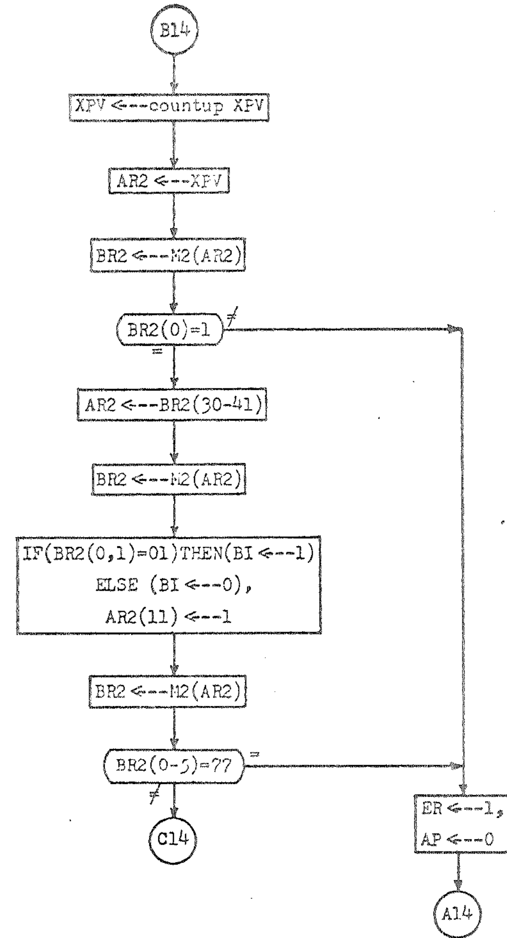


Figure 31b Read/Write Execution Sequence Chart

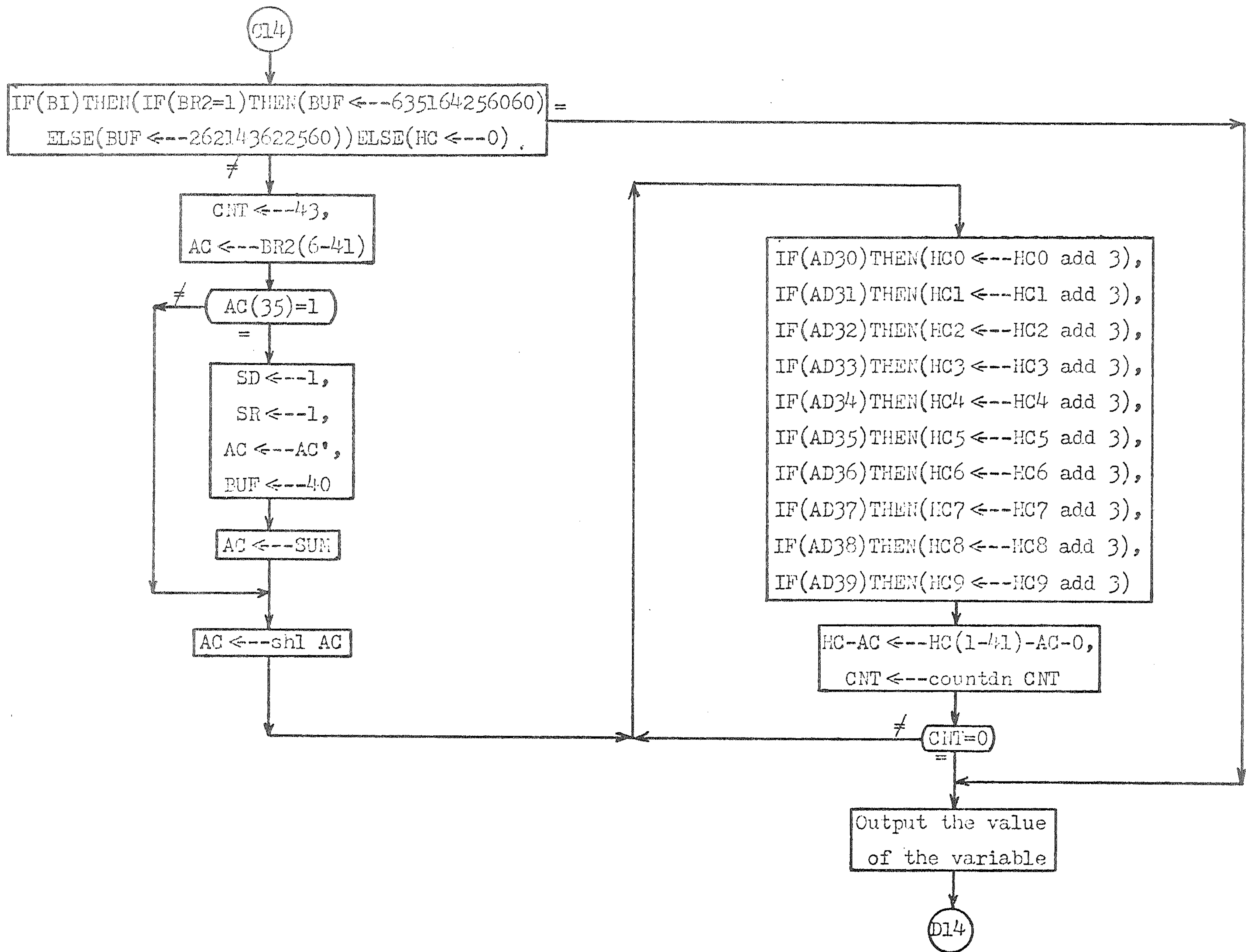


Figure 31c Read/Write Execution Sequence Chart

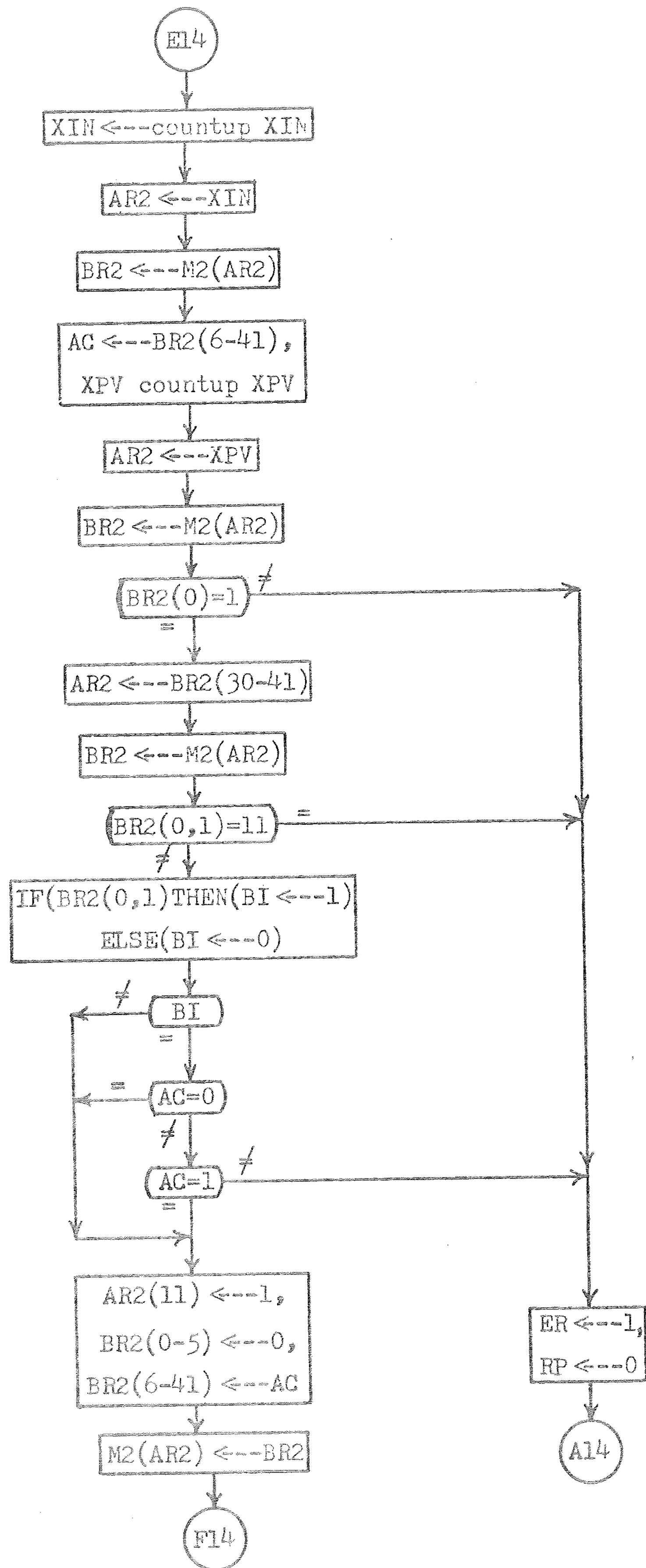


Figure 31d Read/Write Execution Sequence Chart

After setting register XPV, the machine compares its contents with the contents of register XPD. If their contents are not equal, the machine performs the write operation; otherwise the operation is complete. As shown in Figure 31B, the machine first increments register XPV. It then fetches an operand from the OPERAND LIST. If this operand is not a NAME TABLE address, the machine transfers to the error sequence; otherwise it uses the address to fetch the type of the variable from the NAME TABLE. If the variable is a Boolean variable (BR2(0,1)=01), the machine sets register BI to one. If the variable is an integer variable, the machine sets register BI to zero. At the same time it increments address register AR2 by one and fetches the value of the variable from the NAME TABLE. If the VAI field in subregister BR2(0-5) contains 77_8, the variable has not been assigned a value and the machine transfers to the error sequence. If a value has been assigned to the variable, the machine continues the write operation.

As shown in Figure 31C, by examining the contents of register BI, the machine ascertains whether the variable is of type Boolean or type integer. If the variable is Boolean, its internal value (1 or 0) is converted to a logical value (TRUE or FALSE). This value is placed in the output register BUF. It is then printed out. (This part of the sequence has not been implemented.) If the variable is integer, its value is converted from binary to BCD form and then printed out.

The conversion algorithm implemented in this machine is the double-dabble method of conversion (7). This method is implemented as follows. Initially, register CNT is set to 43_8 and the value to be converted to BCD form is stored in the accumulator. If this value is negative, its two's complement is formed and a minus sign is placed in the output register BUF. The value is then left adjusted by one bit in the accumulator. The conversion loop is then

executed. This loop is controlled by register CNT. Register HC receives the BCD value. This register is divided into 10 four-bit subregisters. Each subregister contains one digit of the value when the conversion is complete. In the conversion process the contents of the 10 subregisters are simultnaeously examined by terminals AD. Each subregister that contains a value greater than five is incremented by three. The combined register HC-AC is then left shifted one bit and register CNT is decremented by one. This loop continues until the number is register CNT reaches 0; at that time, the BCD value in register HC is transferred to register BUF and is printed out.

When all the variables listed in the WRITE statement have been processed, the machine erases their NAME TABLE addresses from the OPERAND LIST by resetting index register XPD. If no iteration is being processed, the machine sets register XPD to 7741_8. This effectively empties the OPERAND LIST. If one or more iterations are being processed, the machine sets register XPD to the address of the location containing the controlled variable of the most recently entered iteration. It then examines the output of the program constituent decoder. If this output indicates that register S contains an apostrophe, the machine transfers to the output string initialization sequence; otherwise it removes the delimiter WRITE from the DELIMITER STACK by setting register D1 to zero. It then increments the PROGRAM AREA address in index register XPA by one, fetches a program constituent from memory M2, and stores this constituent in register S. It then transfers to unconditional statement sequence.

4.23.2 Reading

When register D1 contains the delimiter READ ($D1=51_8$), the machine first examines the output of the program constituent decoder to determine whether or not register S contains an apostrophe. If register S does contain an apostrophe,

the machine transfers to the error sequence; otherwise, it examines the current flag. If register F1 contains one, the machine sets register XPV as explained in Section 4.23.1. After setting register XPV, the machine compares its contents with the contents of register XPD. If their contents are not equal, the machine performs the read operation; otherwise the operation is complete. As shown in Figure 31D, in performing the operation the machine first increments the INPUT QUEUE address in index register XIN by one. It then uses this address to fetch an input value from memory M2. It stores this value in the accumulator and increments register XPV by one. It then fetches the operand addressed by the contents of register XPV from the OPERAND LIST. If this operand is not a NAME TABLE address (BR2(0)≠1), the machine transfers to the error sequence; otherwise, it transfers this address to address register AR2 and fetches the name from the NAME TABLE. It then examines the name's type in subregister BR2(0,1). If the name is a label (indicated by 11 in the type field), the machine transfers to the error sequence. If the name is a Boolean variable (indicated by 01 in the type field), the machine sets register BI to one. It then examines the value in the accumulator. If this value is not one or zero and register BI indicates the name is a Boolean variable, the machine transfers to the error sequence; otherwise, it increments register AR2 by one and assigns the value to the name by storing it in the NAME TABLE. It then loops to compare the contents of register XPV with the contents of register XPD.

When all the variables listed in a READ statement have been assigned a value, the machine performs the same operations that it performs upon completing the WRITE statement. These operations are described in Section 4.23.1.

4.24 Error Sequence

The machine transfers to this sequence whenever a syntax error is detected. Although this sequence is not implemented, it is assumed that the machine identifies the error, prints out a diagnostic message, and halts.

5. Acknowledgements

I am indebted to Dr. Yaohan Chu for the help and guidance he provided throughout this report. I would like to acknowledge Drs. Jack Minker and Victor Schneider who taught the courses from which this work evolved.

Special thanks are due my wife and Miss Nancy A. Nowell for their efforts in typing this report.

6. References

1. V. Schneider, "On the Parsing of Context-free Languages by Pushdown Automata", Tech. Report 68-76, Computer Science Center, University of Maryland, August, 1968.

2. V. Schneider, "Syntax-checking and Parsing of Context-free Languages by Pushdown-store Automata", Proc. of the SJCC, 1967, pp. 685-697.

3. Y. Chu, "An Algol-like Computer Design Language", Comm. of the ACM, Oct. 1965, pp. 607-615.

4. Y. Chu, "Introduction to Computer Organization", Prentice-Hall, Inc., 1970.

5. R. Morris, "Scatter Storage Techniques", Comm. of the ACM, Jan. 1968, pp. 38-44.

6. Y. Chu, "Digital Computer Design Fundamentals", McGraw-Hill, N.Y., 1962.

7. Instruction Manual for Supplementary S-PAC Modules and Equipment, Document No. 71-175, Computer Control Company, Inc., Framingham, Massachusetts, 1964, pp. 5-26, 5-27.

8. P. Naur., Editor, "Revised Report on the Algorithmic Language Algol 60", Comm. of the ACM, June, 1963, pp. 1-17.

9. Programmer's Reference Manual for Univac 1108 Algol, Document No. UP-7544, Sperry Rand Corporation, 1967.

APPENDIX

Description of the Subset of ALGOL

The language described herein is the subset of ALGOL (8,9) that is executed by the ALGOL machine. The major features of ALGOL that are not implemented are procedures, arrays, switches, real variables, comment statement, logical operations, and the arithmetic operation of exponentiation.

This description is intended as a reference to this subset of ALGOL only and is not intended for any other purpose.

A.1 Elements of the Subset

A.1.1 Character Set

Programs for this machine are written using the character set consisting of:

letters: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

digits: 0 1 2 3 4 5 6 7 8 9

other symbols: , : $ = + - * / ' ( ) ␣

where ␣ represents a blank space

A.1.2 Delimiters

The delimiters are:

arithmetic operators: + - * /

relational operators: LSS LEQ EQL GEQ GTR NEQ

sequential operators: GOTO IF THEN ELSE FOR DO

input-output operators: READ WRITE

separators: , : $ ␣ STEP UNTIL WHILE

declarators: BOOLEAN INTEGER

brackets: ( ) ' BEGIN END

It is noted that some of the delimiters consist of groups of characters. As will be seen, these delimiters have the same form as names and therefore

they are called reserved names. These names can never be used except in their context as delimiters.

Where it is not obvious, the meaning of a delimiter will be given at the appropriate place in the description.

A.1.3 Names

A.1.3.1 Syntax

A <name> consists of any string of seven or less letters or digits, beginning with a letter.

A.1.3.2 Semantics

A name is chosen by a programmer to identify variables and labels. Names are unique only to their first five characters; that is, two names must differ in their first five characters to be considered different names. All names must be separated from each other by delimiters.

A.1.4 Numbers

A.1.4.1 Syntax

A <number> is an integer. An <integer> consists of a string of one or more digits. A number cannot exceed in magnitude the value $2^{35}-1=34,359,738,367$.

A.1.4.2 Semantics

All numbers are considered to be of the type INTEGER.

A.1.5 Logical Values

The <logical values> are TRUE and FALSE. They are represented internally by 1 and 0 respectively. These values have the same form as names and therefore they are also considered reserved names and cannot be used except in their context as logical values.

A.2 Expressions

An expression is a rule for computing a value. There are three kinds of expressions in this subset: arithmetic, logical, and designational. The constituents of these expressions are logical values, numbers, variables, arithmetic, relational, and certain sequential operators, and other expressions. The value of an arithmetic expression is an integer number. The value of a logical expression is either TRUE or FALSE; these values are represented internally by 1 and 0 respectively. The value of a designational expression is a label.

A.2.1 Variables

A.2.1.1 Syntax

A <variable> is a <name> .

A.2.1.2 Semantics

A variable is a designation given to a single value. The value may change during program execution but the variable name will not. The type of the value of a particular variable is defined in the declaration for the variable (see Section A.5, Declarations).

A.2.2 Arithmetic Expressions

A.2.2.1 Syntax

An <arithmetic expression> is either a <sum> or an expression of the form IF <logical expression> THEN <sum> ELSE <arithmetic expression> .

A <sum> is defined as:

a) <term>

b) + <term>

c) - <term>

d) <sum> + <term>

e) <sum> - <term>

A term is defined as:

a) <factor>

b) <term> * <factor>

c) <term> / <factor>

A factor is defined as:

a) <variable>

b) <number>

c) (<sum>)

A.2.2.2 Semantics

The value of an arithmetic expression that is simply a sum is obtained by performing the indicated arithmetic operations on the values of the factors of the sum. The value of an arithmetic expression containing the delimiters IF, THEN and ELSE is determined by the values of the logical expressions (see Section A.2.3). The logical expressions are evaluated one by one from left to right until an expression having the value TRUE is found. The value of the arithmetic expression is then the value of the sum following this logical expression. All operands in arithmetic expressions, with the exception of the logical expressions which are found between the delimiters IF and THEN, must be of the type INTEGER. Also, the resultant value of all arithmetic expressions is of type INTEGER.

A.2.2.3 Precedence of Operators

Parenthesis may be used to specify the order of operations in arithmetic expressions. If parentheses are not used (or within parentheses) the order of operations is determined from left to right with the following precedence established by the syntax:

a) * /

b) + -

A.2.3 Logical Expressions

A.2.3.1 Syntax

A <logical expression> is a <relation> .

A <relation> is defined as:

a) <logical primary>

b) <sum> <relation operator> <sum>

A <logical primary> is defined as:

a) <variable>

b) <logical value>

A.2.3.2 Semantics

Any variable used as a logical primary must be declared BOOLEAN (see Section A.5, Declarations). A relation of the form <sum> <relational operator> <sum> has the value TRUE if the relation holds and the value FALSE if the relation does not hold.

The meanings of the relational operators are:

Operator	Meaning
LSS	Less then
LEQ	Less than or equal to
EQL	Equal to
GEQ	Greater than or equal to
GTR	Greater than
NEQ	Not equal to

A.2.4 Designational Expressions

A.2.4.1 Syntax

A <designational expression> is a <label>. A <label> is a <name>.

A.2.4.2 Semantics

The value of a designational expression is a label.

A.3 Statement

As in ALGOL 60, the statement is the basic unit of operation in this subset. Normally, statements are executed in the order they are written in the program. However, this sequence can be altered by the execution of certain types of statements. A statement can be an unconditional statement, a conditional statement, an iteration, or an empty statement.

A.3.1 Unconditional Statement

An <unconditional statement> is defined as:

a) an <assignment>

b) a <transfer>

c) a <communication>

d) a <block> (see Section A.4, Program Structure)

A.3.1.1 Assignment

A.3.1.1.1 Syntax

An <assignment> is defined as:

a) <variable>=<arithmetic expression

b) <variable>=<logical expression>

c) <variable>=<assignment>

A.3.1.1.2 Semantics

When executed, an assignment assigns the value of the arithmetic expression or the logical expression to each of the variables preceding the equal sign. If the expression is an arithmetic expression, then all of the variables to the left of the equal sign must be of type INTEGER, whereas, if if the expression is a logical expression then all of the variables to the left of the equal sign must be of type BOOLEAN.

A.3.1.2 Transfer

A.3.1.3.1 Syntax

A <transfer> consists of the sequential operator GOTO followed by a <designational expression>.

A.3.1.2.2 Semantics

When a transfer is executed, the value of the designational expression (which is a label) is determined and execution of the program continues at the start of the statement with this label. A transfer cannot lead from outside into a block.

A.3.1.3 Communication

A.3.1.3.1 Syntax

A <communication> is defined as:

a) WRITE (<list>)

b) READ (<variable list>)

A <list> is defined as:

a) <variable >

b) '<string of characters>'

c) <list>,<variable>

d) <list>,'<string of characters>'

A <variable list> is defined as:

a) <variable>

b) <variable list>,<variable>

A <string of characters> is limited to 1000 characters in length.

A.3.1.3.2 Semantics

The operator WRITE causes the values of the variables and the string of characters that are listed to be printed out on the line printer in the order in which they are listed. Each value and string of characters is printed on a

separate line.

The operator READ causes cards to be read until each variable listed in the variable list has been assigned a value. The variables are assigned values in the order in which they are listed in the variable list. Values on a card are delimited by one or more blanks and by the end of the card. Any values on a card that is read in that are not assigned to a variable are lost; that is, the next read operation begins by reading a new card. Care must be taken that the values on the cards correspond in type to the variables to which they are assigned. A communication which has no elements in its list or variable list is equivalent to an empty statement (Section A.3.4).

A.3.2 Conditional Statement

A.3.2.1 Syntax

A <conditional statement> has one of the two following forms:

a) IF <logical expression> THEN <unconditional>

b) IF <logical expression> THEN <unconditional> ELSE <statement>

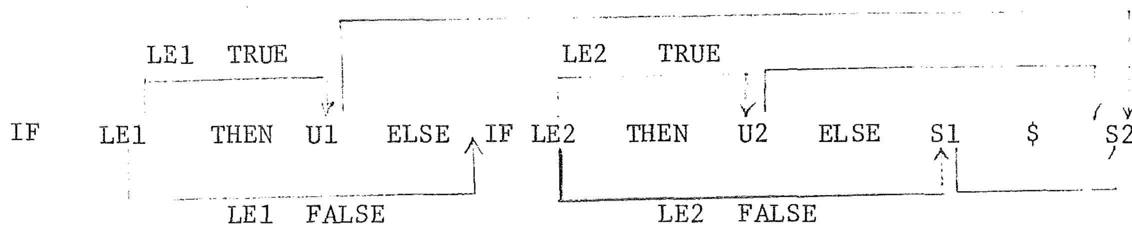
A.3.2.2 Semantics

The first form of the conditional statement means that the unconditional is executed only if the logical expression has the logical value TRUE. Otherwise, the unconditional is skipped and execution begins again with the next statement in sequence.

The second form of the conditional statement menas that if the logical expression has the logical value TRUE then the unconditional is executed and unless it alters the execution sequence, the next statement that is executed is the first statement following the entire conditional statement. If the logical value of the logical expression is FALSE then the statement immediately following the delimiter ELSE is executed and the unconditional is skipped.

The following diagram illustrates the execution sequence for condition-

al statements that are "nested".



A.3.3 Iteration

A.3.3.1 Syntax

An <iteration> consists of a <FOR clause> followed by a <statement>.

A <FOR clause> is: FOR <variable>=<for list> DO.

A <for list> is defined as:

a) <for list element>

b) <for list>,<for list element>

There are three kinds of <for list elements>, they are:

a) <arithmetic expression>

b) <arithmetic expression> STEP <arithmetic expression> UNTIL <arithmetic expression>

c) <arithmetic expression> WHILE <logical expression>

A.3.3.2 Semantics

An iteration causes the statement which it contains to be executed zero or more times. It also causes its controlled variable to take on one or more values in sequence. When all of the for list elements have been processed, execution of the program continues with the statement immediately following the iteration. Each type of for list element is discussed separately below.

A.3.3.3 For List Elements

For list elements are processed in turn from left to right. The sequence of values assigned to the controlled variable is obtained in this way.

A.3.3.3.1 Arithmetic Expression

The value of the arithmetic expression is calculated and assigned to the controlled variable. The statement contained in the iteration is executed once.

A.3.3.3.2 STEP-UNTIL List Element

The assignments and executions that take place when an element of the type A STEP B UNTIL C is processed can best be described as the following ALGOL statements.

```
    VAR = A $
L1:IF (VAR-C) * SIGN(B) LEQ 0 THEN
    BEGIN Statement $
          VAR = VAR + B $
          GOTO L1
    END
```

VAR is the controlled variable. If the test fails initially, then the statement contained in the iteration is not executed at all. Each time the statement is executed, this for list element must be reevaluated to see if the statement should be executed again.

A.3.3.3.3 WHILE Element

The sequence of execution that takes place when an element of the type D WHILE E is processed can best be described by the following Algol statements:

```
L2:VAR = D $
    IF E THEN
    BEGIN Statement $
          GOTO L2
    END
```

Again, VAR is the controlled variable. The iteration continues until the logical expression has a value of FALSE. This for list must also be reevaluated each time the statement is executed.

A.3.3.4 The Value of the Controlled Variable Upon Termination of an Iteration

When an iteration is terminated by execution of a transfer leading out of the iteration, the controlled variable retains the value it had before the transfer was executed. When an iteration is terminated by exhaustion of its for list, the controlled variable retains the value it was last assigned.

A.3.3.5 Transfer Leading Into an Iteration

The effect of a transfer which refers to a label within an iteration is undefined.

A.3.4 Empty Statement

An empty statement executes no operation. It can be used to place a label.

A.4 Program Structure

A.4.1 Syntax

A <program> is defined as a <block>.

A <block> consists of a <block head> followed by a <body> followed by delimiter END.

A <block head> is defined as:

a) BEGIN

b) <block head> declaration $

A <body> is defined as:

a) <label statement>

b) <body>$<label statement>

A <label statement> is defined as:

a) <statement>

b) <label>:<label statement>

A.4.2 Semantics

An ALGOL program written in this subset is a block which may contain other blocks within it. A block constitutes the scope for names declared within it. Names declared in the block head of a block are defined only in that block and have no meaning outside of it. Names (except those used as labels) that appear in a block but are not declared in that block must have been declared in a block which encompasses the block in which they appear. If a name is declared in a block and then is declared again in an inner block, the first declaration of the name is inaccessible inside the inner block. A name can be declared only once in a block; that is, a name can identify only one value in a block. Labels separated from a statement by a colon behave as though declared in the block head of the smallest block encompassing the statement.

A.5 Declarations

A.5.1 Syntax

A <declaration> is defined as a <name declaration>.

A <name declaration> is defined as:

a) INTEGER <name>

b) BOOLEAN <name>

c) <name declaration>,<name>

A.5.2 Semantics

A name declaration defines certain properties of the variables used in a block. Each name must be declared before it is used in a block.