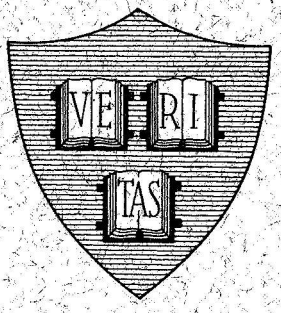
Office of Naval Research

Contract N00014-67-A-0298-0006 NR-372-012

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
Grant NGR 22-007-068

THE MATRIX ALGEBRA PROGRAM A CONVERSATIONAL LANGUAGE FOR NUMERICAL MATRIX OPERATIONS - PART II: REFERENCE MANUAL



By

P. M. Newbold

June 1968

Technical Report No. 562

Division of Engineering and Applied Physics
Harvard University • Cambridge, Massachusetts

Office of Naval Research

Contract N00014-67-A-0298-0006

NR - 372 - 012

National Aeronautics and Space Administration

Grant NGR 22-007-068

THE MATRIX ALGEBRA PROGRAM A CONVERSATIONAL LANGUAGE FOR NUMERICAL MATRIX OPERATIONS - PART II: REFERENCE MANUAL

By

P. M. Newbold

Technical Report No. 562

June 1968

The research reported in this document was made possible through support extended the Division of Engineering and Applied Physics, Harvard University by the U. S. Army Research Office, the U. S. Air Force Office of Scientific Research and the U. S. Office of Naval Research under the Joint Services Electronics Program by Contracts N00014-67-A-0298-0006, 0005, and 0008 and by the National Aeronautics and Space Administration under Grant NGR 22-007-068.

Division of Engineering and Applied Physics

Harvard University • Cambridge, Massachusetts

THE

MATRIX ALGEBRA PROGRAM

A

CONVERSATIONAL LANGUAGE FOR

NUMERICAL MATRIX OPERATIONS –

PART II: REFERENCE MANUAL

By

P. M. Newbold

Division of Engineering and Applied Physics

Harvard University, Cambridge, Massachusetts

June 1968

CONTENTS

1. INTRODUCTION

THIS MANUAL contains detailed information on the structure and operating principles of the MATRIX ALGEBRA PROGRAM, abbreviated MAP. Instructions for the use of the MAP language, and a description of its facilities are given in the MAP USER'S MANUAL [1]. It is assumed that the reader is familiar with the USER'S MANUAL, and also with the SDS 940 Time-Sharing System Manuals.

In particular, since the MAP processor is written exclusively in the SDS 940 Assembly Language ARPAS, the reader is expected to know ARPAS thoroughly.

After reading this manual it should be possible for the reader to insert his own sections of coding into the MAP processor so as to extend the facilities provided by the language.

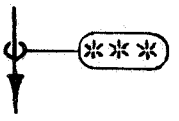
2. NOTATION

IN GENERAL, throughout the manual, words and phrases printed in capitals are to be considered as definitive terms relating to MAP language, or the MAP processor. Many of the terms used are introduced in the MAP USER'S MANUAL.

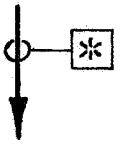
Numbers written in the decimal system have no subscripts. Numbers written in the octal system are subscripted "8" and those in the binary system, "2".

Flow Diagrams

There are many flow diagrams illustrating the operation of sections of the MAP processor. Notation in these is largely conventional or self-explanatory. Points in the flow diagrams labelled thus:



correspond with symbolic locations in the processor possessing the name substituted for *** . Points labelled thus:



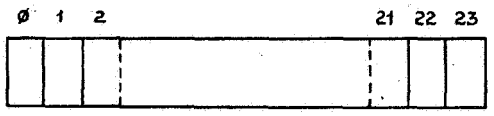
are for reference only and have no counterpart in the processor program. In flow diagrams representing subroutines, the generalised calling sequence is given, together with the contents of the working registers A , B , and X at the times of entry and exit.

The operation of the MAP processor is based on the cross-referencing of various storage tables. The remainder of this section is devoted to their notation in the manual.

Words

A WORD AS DEFINED here is more restricted in sense than is usual. As far as the MAP processor is concerned, a WORD is a unit of storage of data or information. Physically, it is either a

location or two adjacent locations in core storage, or the image of the location(s) on a disk file. A location of storage is a (conventional) word consisting of 24 bits, denoted by:



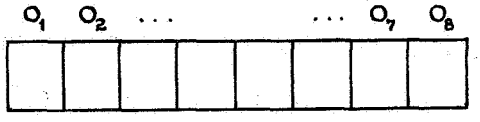
A WORD may or may not be part of a TABLE; if so it is given an INDEX number, and may be called a TABLE ENTRY. If not the WORD is given a symbolic name (which is the symbolic location of the start of the WORD). The contents of the WORD are denoted by the name of the WORD enclosed in braces.

EXAMPLE:

name of location:	INDEX
contents of location:	{INDEX}

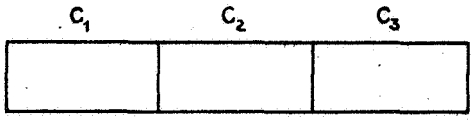
A WORD may be used to store any one of three different kinds of data. The WORD TYPE is dependent on the kind of information stored. It is possible for WORDS of mixed TYPE to exist.

TYPE I — A TYPE I WORD is a one-location WORD in which an integer number is stored in the form of octal digits. Since each octal digit occupies three bits, the WORD contains a number up to eight octal digits long. These digits are labelled $O_1, O_2 \ldots\ldots O_8$. Frequently these digits may constitute several numbers INDEXING other TABLES. A TYPE I WORD is denoted by:



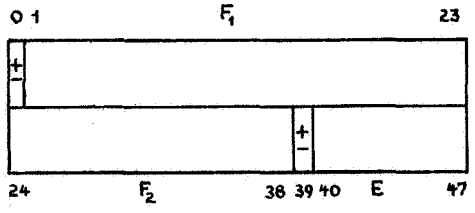
TYPE II

A TYPE II WORD is a one-location WORD in which character codes are stored. Since each octal character code occupies eight bits, the WORD contains three character codes. These are labelled C_1, C_2, and C_3. The two left-most bits of an alphanumeric character code are both zeros. A TYPE II WORD is denoted by:



TYPE III

A TYPE III WORD is a two-location WORD in which a floating point decimal number is stored.[1] The first location of the WORD contains the most significant part of the mantissa. The second location contains the least significant part of the mantissa, and the exponent. These are labelled F_1, F_2, and E respectively. The whole WORD is INDEXED as a single entity. The WORD is denoted by:



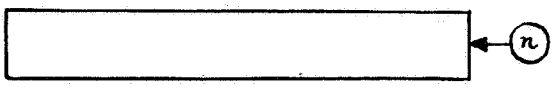
Indices and Tables

A TABLE IS A sequence of WORDS occupying adjacent locations in core storage, or their images on a disk file. Each WORD of the TABLE has an INDEX number denoting its position in the TABLE.

---

1. A more detailed explanation of the floating point arithmetic system of the SDS 940 is to be found in [2], and in the SDS 940 TECHNICAL MANUAL [3].

The value of the INDEX may be either positive or negative. The INDEXING of a TABLE can run either forwards or backwards, usually starting from zero. A TABLE contains WORDS of one TYPE only, unless the WORDS themselves are of a mixed TYPE. A WORD with INDEX number n is denoted by:



In the case of TYPES I & II WORDS the INDEXING of the TABLE coincides with the natural location indexing of the TABLE locations. The name of the TABLE is the symbolic label given to the zeroth location of the TABLE in the processor program (or equivalently, the WORD with INDEX zero).

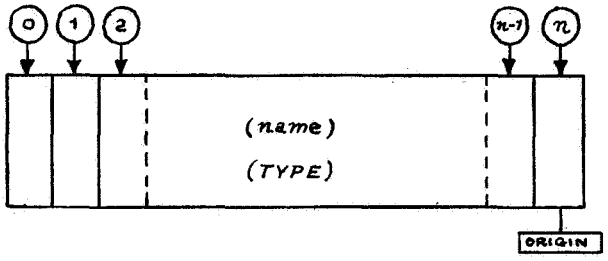
In the case of TYPE III WORDS the INDEXING does not coincide with the natural location indexing of the TABLE locations. The natural index of the first location of a WORD in the TABLE is obtained by doubling the INDEX number of the WORD. The natural index of the second location of the WORD is obtained by adding one to the previous result. The name of the TABLE is the symbolic label given to the zeroth location of the TABLE in the processor program (or equivalently, the first location of the WORD with INDEX zero).

The contents of a WORD of a TABLE are denoted by the name of the TABLE followed by a comma and the INDEX number of the WORD, all enclosed in braces.

EXAMPLE:

name of location containing INDEX number:	INDEX
contents of location:	{INDEX}
INDEXED WORD in ICOM-TABLE:	ICOM, {INDEX}
contents of INDEXED WORD:	{ICOM, {INDEX}}

Most of the TABLES are filled with information as MAP language is being used. TABLES are not always filled starting with INDEX zero. The actual starting point is called the ORIGIN OF STORAGE of the TABLE. A TABLE, its INDEXING, and its ORIGIN OF STORAGE are denoted by:

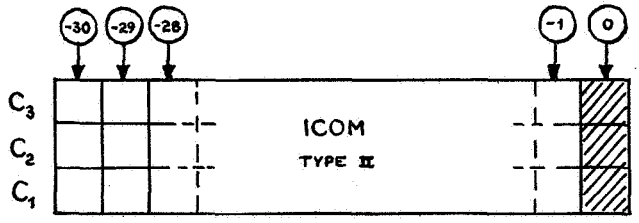


Unused parts of TABLES are shown shaded, and contain zeros.

3. THE STORAGE TABLES OF MAP

THE OPERATION of MAP depends on the storage of data and information in eight major TABLES. By cross-referencing from TABLE to TABLE, MAP decides upon the right course of action at each stage in execution. The TYPE and purpose of each TABLE is explained in turn.

ICOM-TABLE

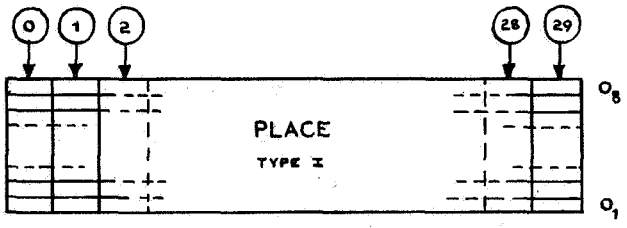


The ICOM-TABLE stores the names of all MAP language COMMANDS. The two left-most bits of

C_3 (bits 16 and 17) of each WORD are used to store the number of ARGUMENTS appropriate to the COMMAND whose name is stored in that WORD:

$\emptyset\emptyset_2$ - no ARGUMENTS
$\emptyset 1_2$ - one ARGUMENT
$1\emptyset_2$ - two ARGUMENTS

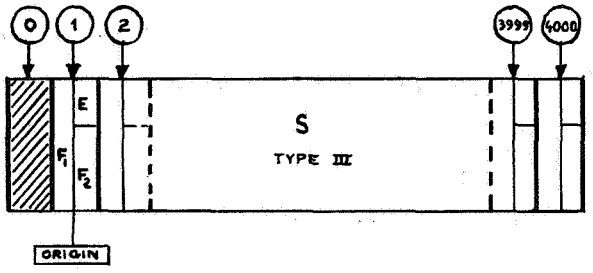
PLACE-TABLE



The PLACE-TABLE stores the locations in the MAP processor to which execution is directed for carrying out the operation represented by the COMMAND named in the corresponding position of the ICOM-TABLE. {PLACE, i} is the transfer location for the COMMAND {ICOM, -1-i}.

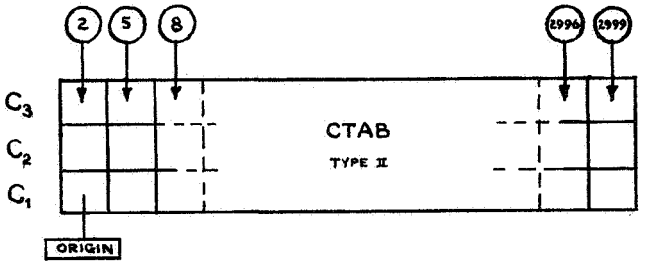
Since the contents of both the ICOM- and PLACE-TABLES are permanent, no ORIGINS OF STORAGE are specified for them. All the following TABLES have specified ORIGINS.

S-TABLE



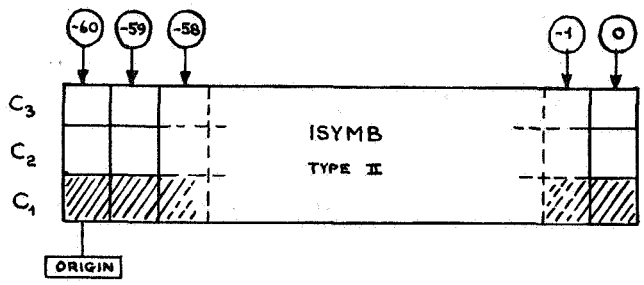
The S-TABLE stores the numerical values of the VARIABLES that the user defines. The TABLE is divided into contiguous BLOCKS of different lengths. Each VARIABLE is assigned one BLOCK of storage, which is of just sufficient length to accommodate the values of all the elements of the VARIABLE (assuming that generally the VARIABLE represents a matrix). The values of the matrix represented are stored row by row. The BLOCKS storing the values of user-defined VARIABLES commence at INDEX number 1ø1, and work forward. The standard output VARIABLE OO is assigned a BLOCK of 144_8 WORDS starting at the ORIGIN. This BLOCK is of constant length in spite of the fact that OO can change its row and column dimensions.

CTAB-TABLE



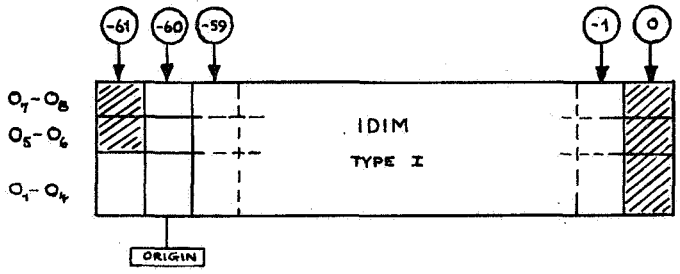
The CTAB-TABLE stores the COMMENTS attached to INDIRECT STATEMENTS of a PROGRAM by the user. The characters of the COMMENTS are stored in a string, three to a WORD, with the ORIGIN at C_1 of the zeroth WORD. This is the only TABLE which is INDEXED by the number of characters from the ORIGIN. C_1 of the zeroth WORD is the zeroth character. All COMMENTS are stored end to end with no empty character positions.

ISYMB-TABLE



The ISYMB-TABLE stores the VARIABLE names that the user defines. C_1 of each WORD is blank. The standard output VARIABLE name OO is permanently stored at the ORIGIN. User-defined names fill up the TABLE from INDEX number -59 forwards.

IDIM-TABLE



Each WORD in the IDIM-TABLE contains numerical data associated with the VARIABLE named in the WORD of the same INDEX in the ISYMB-TABLE. Each WORD is in three parts, each containing an octal number.

O_1 to O_4 contain the INDEX of the last WORD in the BLOCK of storage in the S-TABLE assigned to the storing of the values of the corresponding VARIABLE named in the ISYMB-TABLE. This INDEX will be called the BLOCK INDEX for the VARIABLE.

O_5 to O_6 contain the row dimension of the corresponding VARIABLE named in the ISYMB-TABLE.

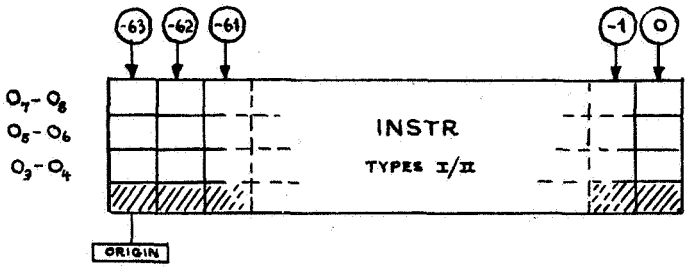
O_7 to O_8 contain the column dimension of the corresponding VARIABLE named in the ISYMB-TABLE.

The numerical data for the standard output VARIABLE OO is permanently stored at the ORIGIN. Before the first use of MAP language, OO is a 1ø by 1ø matrix. The INDEX of the last WORD in the BLOCK assigned to OO in the S-TABLE is 144_8. Hence {IDIM, -6ø} = 1441212_8.

To locate the WORD in the S-TABLE containing the value of the first element of the VARIABLE whose name is {ISYMB, i}, MAP unloads O_1 to O_4 {IDIM, i-1}, and adds one to the result. This is the desired INDEX. To preserve the uniformity of operation for the standard output VARIABLE, O_1 to O_4 {IDIM, -61} are set to zero, representing the INDEX in the S-TABLE of the first WORD of the BLOCK assigned to OO, minus one.

The BLOCKS of the S-TABLE are assigned in the same order as their respective VARIABLE names in the ISYMB-TABLE.

INSTR-TABLE



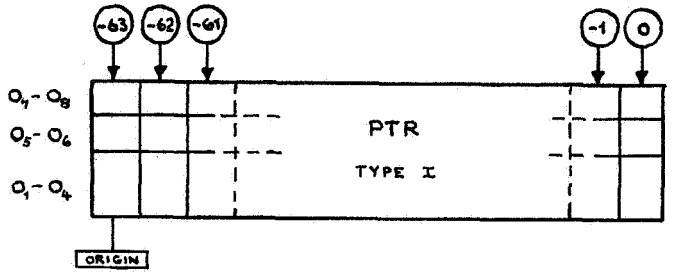
The INSTR-TABLE is a TABLE of WORDS of mixed TYPE storing coded versions of the STATEMENTS that the user types in. DIRECT STATEMENTS are always stored in the zeroth WORD. When the user creates a PROGRAM of INDIRECT STATEMENTS, the STATEMENTS are stored starting at the ORIGIN and working forwards. Each STATEMENT is coded into between two and six octal numbers. Between one and three of these are stored in the INSTR-TABLE.

O_7 to O_8 contain the two's complement of the INDEX of the WORD in the ICOM-TABLE containing the same COMMAND name as the STATEMENT.

O_3 to O_6 contain the coded versions of the ARGUMENTS of the STATEMENT. If the STATEMENT has no ARGUMENTS, O_3 to O_6 are zero. If the STATEMENT has one ARGUMENT, O_3 to O_4 are zero, and O_5 to O_6 contain the coded ARGUMENT. If the STATEMENT has two ARGUMENTS, O_3 to O_4 contain the coded first ARGUMENT, and O_5 to O_6 contain the coded second ARGUMENT.

Depending on the type of the ARGUMENT the codes may be interpreted in two ways. If the ARGUMENT is LABEL-type, each code is the value of the LABEL itself converted to the octal system. If the ARGUMENT is VARIABLE-type, each code is the two's complement of the INDEX of the WORD in the ISYMB-TABLE containing the same VARIABLE name as the ARGUMENT.

PTR-TABLE



Each WORD of the PTR-TABLE contains the remaining two code numbers for the STATEMENT partially coded into the WORD of the same INDEX in the INSTR-TABLE.

O_5 to O_6 contain zero if the STATEMENT is DIRECT, or the LABEL of the STATEMENT converted to the octal system if it is INDIRECT.

O_7 to O_8 contain the number of ARGUMENTS of the STATEMENT if the ARGUMENTS are VARIABLE-type, or zero if they are LABEL-type. This number is called the CHECKNUMBER of the STATEMENT.

O_1 to O_4 contain the INDEX in the CTAB-TABLE of the first character code of the COMMENT attached to the STATEMENT if the latter is INDIRECT and a COMMENT for it exists, or zero otherwise.

In the next section an overall outline of the operation of the MAP processor is given.

4. BASIC OPERATION OF THE PROCESSOR

THE BASIC OPERATION of MAP depends on the coding of DIRECT STATEMENTS and the execution of sequences of DIRECT or INDIRECT STATEMENTS. All operations, including for example, the creation of a VARIABLE LIST or a PROGRAM are carried out as phases in the execution of STATEMENTS.

The section of the MAP processor devoted to these fundamental operations is called the DIRECTIVE SUBPROCESSOR. It consists of a loop of complex form which is iterated every time one STATEMENT is processed. The DIRECTIVE SUBPROCESSOR is shown in Flow Diagram 1. Each phase of its loop will be examined in turn.

On first entry into the MAP processor, the title of the processor is printed out (a). Initialization of the processor follows (b). This process will be discussed later. Next the processor sets an INDEX counter 'INDEX' to zero and asks the user to type in a DIRECT STATEMENT (d). The STATEMENT is input, coded, and stored as $\{INSTR, \emptyset\}$ and $\{PTR, \emptyset\}$ (e). Details of the coding process itself are given later; the contents of the INSTR- and PTR-TABLES on completion of the process are described in the previous section.

If the coding process is unsuccessful because of a user error (f), an error message is printed out (g), and the processor asks for a new DIRECT STATEMENT (d).

If the coding process is successful, the MAP processor moves on to the next phase in execution (h). At this point {INDEX} = $\emptyset$. The processor therefore unloads {INSTR, $\emptyset$} and {PTR, $\emptyset$}. The contents give the following information:

INSTR-TABLE

(I) The INDEX in the ICOM-TABLE of the COMMAND of the coded STATEMENT; and either

(II) the octal equivalents of the LABEL-type ARGUMENTS of the coded STATEMENT; or

(III) the INDICES in the ISYMB-TABLE of the names of the VARIABLE-type ARGUMENTS of the coded STATEMENT.

PTR-TABLE

(IV) The CHECKNUMBER of the STATEMENT.

Knowing (I) the MAP processor obtains the location in the processor program to which control is to be transferred for execution of the COMMAND, from the corresponding entry of the PLACE-TABLE (see the previous section). {INSTR, $\emptyset$} cannot have been an empty WORD, so that MAP proceeds to the next phase (i), effecting the transfer of control and executing the coded DIRECT STATEMENT (j). This phase utilizes (II) through (IV) above.

On return, if execution was deleted because of an irremediable error (k), an error message is printed (g) and a new DIRECT STATEMENT is asked for.

If execution is successful, {INDEX} is incremented by one (ℓ). Two possibilities now arise. If the STATEMENT just executed was any STATEMENT apart from a DIRECT 'BRANCH' STATEMENT, then at (m) {INDEX} > ∅. MAP thus chooses the 'NO' branch, returning to (d) to ask for a new DIRECT STATEMENT.

On the other hand, if the STATEMENT just executed was a DIRECT 'BRANCH' STATEMENT (the user presumably wishing to cause execution of a stored PROGRAM), then at (m) {INDEX} < ∅. The actual value will be the negative of the INDEX of the WORD in the PTR-TABLE containing the coded LABEL which appeared as the ARGUMENT of the DIRECT 'BRANCH' STATEMENT. The 'YES' branch is taken at (m), and unloading of {INSTR, {INDEX}} and {PTR, {INDEX}} follows.

Thus execution of the stored PROGRAM starts with execution of the INDIRECT STATEMENT specified by the user in the DIRECT 'BRANCH' STATEMENT. This time during the unloading process (h), because the STATEMENT is INDIRECT, in addition to (I) through (IV) above, the following information is given:

PTR-TABLE

(V) The INDEX in the CTAB-TABLE of the COMMENT attached to the INDIRECT STATEMENT, if present; and

(VI) the octal equivalent of the LABEL of the INDIRECT STATEMENT.

If the STATEMENT has a COMMENT, it is printed out, and MAP then proceeds as before by obtaining the transfer location for the currently processed STATEMENT from the PLACE-TABLE. At this stage also an empty WORD cannot be present (i), so MAP executes the INDIRECT STATEMENT (j).

The loop (h)(i)(j)(k)(ℓ)(m) is cycled repeatedly, executing the INDIRECT STATEMENTS of the PROGRAM in their LOGICAL SEQUENCE. In the absence of any flow-changing STATEMENTS in the PROGRAM, {INDEX} increase by one at each iteration round the loop. Iteration ceases when one of the following four events takes place.

(I) The MAP processor detects at (i) an empty WORD in the INSTR-TABLE: it considers the PROGRAM to have been completely executed normally, and returns to (d) to ask for a new DIRECT STATEMENT.

(II) The MAP processor detects at (m) that {INDEX} are no longer negative. This can only happen if execution of a maximal length PROGRAM has just been completed normally. Again MAP returns to (d) to ask for a new DIRECT STATEMENT.

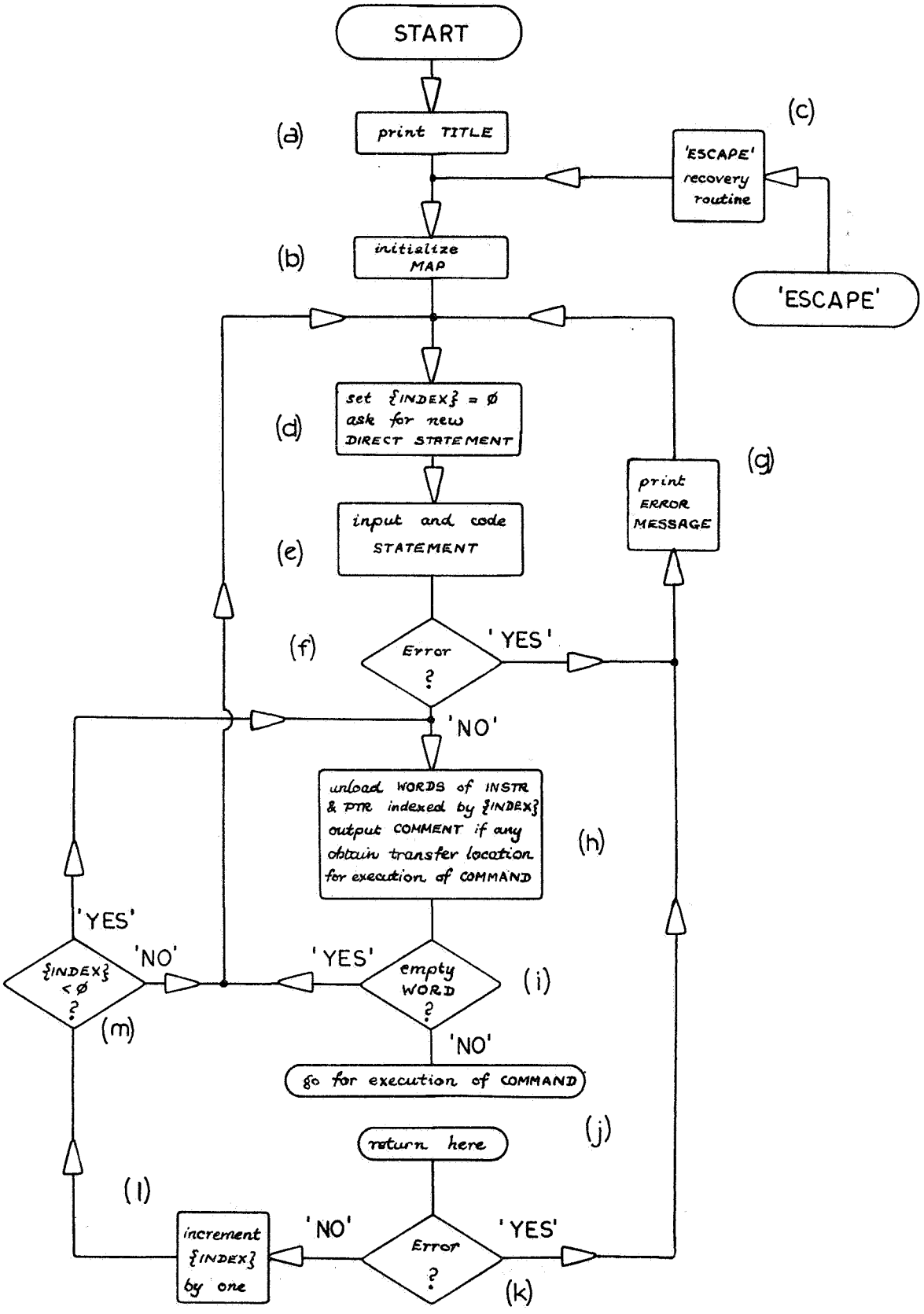
(III) The MAP processor detects at (k) an EXECUTION ERROR and deletes execution. An error message is given (g) and MAP returns to (d) to ask for a new DIRECT STATEMENT.

(IV) The user presses the 'ESCAPE' key. A software interrupt[2] is activated, and MAP is directed to the recovery routine (c) and thence to re-initialize MAP (b).

This completes the discussion of the DIRECTIVE SUBPROCESSOR of MAP.

---

2. The software interrupt system of the SDS 940 is described on p. 8 of the SDS 940 TECHNICAL MANUAL [3].



Flow Diagram 1. The DIRECTIVE SUBPROCESSOR.

5. THE DETAILED PROCESSES OF EXECUTION

THIS SECTION comprises a set of notes designed to supplement the information given in the Flow Diagrams 2 - 8. The reader will also find it useful to refer to a listing of the MAP processor, and to the MAP USER'S MANUAL [1].

There are some general properties of the structure of the processor program that it is useful to bear in mind.

Form of User Subroutines

SUBROUTINES ARE constructed in either of two different ways. If the subroutine has need of only one transmitted parameter apart from the contents of the working registers, then the POP form is employed.[3] The method of call is such that the POP may be used exactly like a machine instruction.

For subroutines requiring more transmitted parameters, or of relatively high complexity, the BRM - BRR type of linkage is employed.[4] A subroutine of this type will from now on be called a BRM for brevity.

---

3. See pages 17-18 of the SDS 940 COMPUTER REFERENCE MANUAL [4], and page 13 of the SDS 940 TAP MANUAL [5].

4. See pages 25-26 of the SDS 940 COMPUTER REFERENCE MANUAL [4].

System Subroutines

There are a large number of system subroutines available on the SDS 940. These are intended to carry out basic operations often required by an ARPAS programmer, (such as input/output operations) to make programming a simpler process. Again there are two forms:

(I) the SYSPOP which is exactly like a POP, except that it is defined within the system; [5]

(II) the BRS, which has a single call instruction, and uses only the working registers to transmit parameters. [6]

For convenience, a list of the BRS's and SYSPOPS used in the MAP processor is given in Appendix A1.

The Directive Subprocessor

THE OPERATION OF the DIRECTIVE SUBPROCESSOR of MAP has already been explained in general terms in the preceding section. More details are given here, and the explanation proceeds more from a programming standpoint than before. The user subroutines employed will be described in following subsections of the manual.

The flowchart of the SUBPROCESSOR is shown again in Flow Diagram 2.

On first entry into the MAP processor at the point ⏀-(WORK), message MS1 is printed out: [7]

5, 6. See the SDS 940 TECHNICAL MANUAL [3].

7. A list of all the messages used is given in Appendix A2.

The 'ESCAPE' interrupt is armed and as a precautionary measure, a BRS closing all disk files is executed.[8] The three characters "<", "space", and "bell" are output to the teletype, and {INDEX} set to zero.

The BRM 'INPUT' is next executed: this handles input of a DIRECT STATEMENT. There are two exits from the BRM. If the user types a mistake the 'BAD' exit is taken, and execution is directed to point ψ-(A1Ø) . Otherwise the 'GOOD' exit is taken. At this point {INSTR, Ø} and {PTR, Ø} constitute the coded STATEMENT. At point ψ-(A11) the MAP processor enters a loop which is cycled when either DIRECT or INDIRECT STATEMENTS are to be executed. The WORDS in the INSTR- and PTR-TABLES containing the coded STATEMENT to be executed are INDEXED by {INDEX}. {INDEX} = Ø for DIRECT STATEMENTS.

First, the CHECKNUMBER O_7O_8 {PTR, {INDEX}} is unloaded, and set into a temporary storage location T+3. Next the BRM 'WDS' is executed. This prints the COMMENT attached to the STATEMENT, if there is one, on the teletype.

A test is now carried out to determine whether {INSTR, {INDEX}} = Ø. A zero value implies that the MAP processor has already executed all the INDIRECT STATEMENTS of a PROGRAM in their LOGICAL SEQUENCE, and no STATEMENT exists in the WORD tested. If the contents of the WORD are zero, execution returns to point ψ-(A1Ø) . If {INSTR, {INDEX}} ≠ Ø the processor unloads and stores the rest of the data for the current STATEMENT in the following way:

---

8. Files are identified by a logical index number. Ø and 1 stand for teletype input or output, and hence no disk files exist with these numbers, and closure does not occur.

$\{T\}$ set to O_3O_4 $\{INSTR, \{INDEX\}\}$
$\{T+1\}$ set to O_5O_6 $\{INSTR, \{INDEX\}\}$
$\{T+2\}$ set to O_7O_8 $\{INSTR, \{INDEX\}\}$

After unloading, $\{T\}$, $\{T+1\}$ are the coded ARGUMENTS of the STATEMENT, and $\{T+2\}$ are the coded COMMAND.

Control of execution is now transferred to that section of the processor executing the specified COMMAND. The *transfer location* is given by $\{PLACE, \{T+2\}-1\}$. The name of the specified COMMAND is given by $\{ICOM, -\{T+2\}\}$. The vector of temporary storage locations T, T+1, T+2, T+3 transmits identifying information during the transfer.

Execution of the STATEMENT now follows. Description of this is deferred until later sections. If an error is detected during execution $\{ERR\}$ are set to some non-zero value identifying the ERROR MESSAGE. Control of execution is transferred to point (A12) on completion.

The MAP processor now makes several tests. The first test determines whether $\{ERR\} = \emptyset$. First suppose that the value is non-zero, implying that an error arose during the execution of the STATEMENT. The processor next makes a test determining whether $\{INDEX\} < \emptyset$. If this is true, then the STATEMENT in which the error occurred was INDIRECT: the processor therefore outputs to the teletype the message MS6:

STOP IN STATEMENT

The LABEL of the incorrectly executed STATEMENT is unloaded from O_5O_6 $\{PTR, \{INDEX\}\}$, and printed by the BRM 'PIC'. Thus the user is provided with information on where execution failed in his PROGRAM.

For all values of {INDEX} execution is now directed to the ERROR MESSAGE subroutine BRM 'ER1'. The ERROR MESSAGE corresponding to the value of {ERR} is printed on the teletype, and {ERR} reset to zero. Execution is then directed back to the point ψ-(A1Ø) .

Alternately, suppose that {ERR} = Ø, implying that no error occurred; then the processor increments {INDEX} by one. A test on {INDEX} is next carried out. If {INDEX} < Ø, then there may still be more INDIRECT STATEMENTS to be executed, and execution is directed to the point ψ-(A11) . Otherwise execution is directed to the point ψ-(A1Ø) .

The condition {INDEX} ≥ Ø could have arisen in two ways. If {INDEX} = Ø this implies that the processor has just successfully completed execution of a PROGRAM which fully occupies the INSTR- and PTR-TABLES. If {INDEX} > Ø this implies that the STATEMENT just executed was DIRECT (but not a flow-changing STATEMENT).

This completes the description of the actual loop itself. One further point remains to be explained in connection with Flow Diagram 2. If at any point the user presses the 'ESCAPE' key the presence of the interrupt mentioned earlier overrides the normal action of the MAP processor, and execution is directed to the BRM 'ESC'. This routine outputs to the teletype a carriage return and line feed, and directs the processor to restart execution of the basic loop at point ψ-[5] .

In the following subsections the operation of the subroutines involved in the DIRECTIVE SUBPROCESSOR is explained.

Routines called by the Directive Subprocessor

SEVERAL SUBROUTINES, all of the BRM type are described in this subsection.

BRM 'WDS' (Flow Diagram 3.)

The BRM 'WDS' is a routine for printing out via the teletype COMMENTS attached to INDIRECT STATEMENTS. On entry {X} is the value of the INDEX in the PTR-TABLE of the WORD containing the starting INDEX of the required COMMENT in the CTAB-TABLE. {A} and {B} are garbage.[9] The calling sequence is:

```
BRM  WDS
[return location]
 .
 .
 .
```

The operation of the routine is as follows. First O_1-O_4 {PTR,{X}} are unloaded, giving the INDEX of the zeroth character of the desired COMMENT. This INDEX number is tested, and if found to be zero, implying that no COMMENT exists for the STATEMENT coded into the WORD in the PTR-TABLE examined, then the exit from the BRM is taken directly.

If the INDEX number is non-zero, then a COMMENT exists. The absolute character location of the zeroth character is calculated, and the characters are then

---

9. The working registers of the SDS 940 are called the A, B, and X registers. X is the index register.

retrieved from the CTAB-TABLE and output to the teletype one by one until a carriage return character is encountered.[10]

A line feed character is then output to the teletype, and the exit from the BRM taken. At exit {X} are the same as at entry, while {A} and {B} are garbage.

BRM 'PIC' (Flow Diagram 4.)

The BRM 'PIC' is a routine which converts an octal number to its character code representation, and prints the result via the teletype. On entry {A} is the two-digit octal number to be converted to its character codes and printed. {B} are garbage, and {X} are unaffected by the BRM. The calling sequence is:

```
BRM  PIC
[return location]
 .
 .
 .
```

The operation of the routine is as follows. First {A} are divided by 12_8. The quotient and the remainder are then the left and right digits respectively of the original number. In turn, these are each converted to their character code form and output to the teletype. The exit from the BRM is then taken.

At exit {A} and {B} are garbage.

---

10. The retrieval of characters from a TABLE or 'string' is one of the processes for which special provision is made in the SDS 940 system. For further information on the 'string processing' system, see the STRING PROCESSING REFERENCE MANUAL [6], and the SDS 940 TECHNICAL MANUAL [3].

BRM 'ER1' (Flow Diagram 5.)

The BRM 'ER1' is a routine which prints out ERROR MESSAGES, and controls the return of execution to a specified location in the processor. At entry {A}, {B} and {X} are garbage. {ERR} constitute an ERROR MESSAGE reference number. The calling sequence is:

```
BRM  ER1
ZRO  [address of return location]
 .
 .
 .
```

The operation of the routine is as follows. First the location of the ERROR MESSAGE corresponding to the value of {ERR} is calculated. The message is printed via the teletype using a BRS 34. {ERR} are reset to zero, and the return location obtained. Lastly the exit from the BRM is taken. At exit {A}, {B} and {X} are garbage.

BRM 'INPUT' (Flow Diagram 6.)

The BRM 'INPUT' is a routine for executing the input of STATEMENTS from the teletype, for checking their acceptability, and for coding them into the INSTR- and PTR-TABLES. It handles the input of DIRECT STATEMENTS, and all parts of INDIRECT STATEMENTS following the colon. On entry {A}, {B} and {X} are garbage. The calling sequence is:

```
BRM  INPUT
[GOOD return location]
ZRO  [address of BAD return location]
ZRO  [address of location containing INDEX
 .           number in INSTR- and PTR-TABLES]
 .
 .
```

The operation of the routine is as follows. On entry the address of the BAD return location is stored in case of its use by the BRM 'ER1'. The contents of two locations used as ARGUMENT and CHECKNUMBER counters are initialized to zero. {X} are set equal to the value of the INDEX of the WORDS in the INSTR- and PTR-TABLES in which the coded STATEMENT is to be put. {INSTR,{X}} are set equal to zero.

Next, the processor checks to see if an error has already occurred. If this is true, so that $\{ERR\} \neq \emptyset$, the 'NO' branch is taken to point ⏀(IN4). If $\{ERR\} = \emptyset$ the 'YES' branch is taken to point ⏀(IN1). This point is the start of a loop which inputs characters three at a time from the teletype into one WORD, and processes them as an entity. The operation of this loop is now examined. First, the three characters of the STATEMENT are input from the teletype into one WORD. If the third character is a comma, then this WORD contains an ARGUMENT; if not, a COMMAND.

Suppose first that the WORD contains an ARGUMENT of the STATEMENT. If the ARGUMENT counter has the value 2, showing that two ARGUMENTS have already been accepted, an error condition arises, and a 'YES' branch is taken to the point ⏀(IN3).

Otherwise the 'NO' branch is taken, and the BRM 'IPC' is next executed. This routine tries to code the ARGUMENT as if it were LABEL-type. If the coding is successful, execution is directed to point ⏀(IN2). If the coding is unsuccessful either the ARGUMENT is VARIABLE-type, or an error in input has taken place.

The BRM 'LOOK' is executed, trying to locate the ARGUMENT in the ISYMB-TABLE. (If the ARGUMENT of the STATEMENT is VARIABLE-type and valid, it will have already been stored in the ISYMB-TABLE by a 'VARIABLES' COMMAND). If no such ARGUMENT exists, it is certain that an error has occurred. {ERR} are set equal to 2, denoting an ERROR MESSAGE reference number, and execution proceeds to point ⏀(IN4). If the ARGUMENT is found in the ISYMB-TABLE, the two's complement of its INDEX in the TABLE is kept; the CHECKNUMBER is incremented by one, and execution proceeds to point ⏀(IN2).

At this point the coded ARGUMENT is placed in position in INSTR,{X}. The ARGUMENT counter is incremented by one, and execution is directed back to point ⏀(IN1) for processing the next three characters of the STATEMENT.

Suppose now that when the characters are processed, they constitute the COMMAND of the STATEMENT. The value of the ARGUMENT counter is merged into bits 16 and 17 of the WORD containing the COMMAND. A BRM 'LOOK' is executed looking up the COMMAND in the ICOM-TABLE. If it is found, the two's complement of its INDEX in the TABLE is loaded in position

in INSTR,{X}. The CHECKNUMBER is loaded into position in PTR,{X}. Execution then proceeds to point ψ-(IN4).

If no such COMMAND is found in the ICOM-TABLE, execution is directed to point ψ-(IN3) where {ERR} are set equal to 9. Execution then proceeds to point ψ-(IN4).

At this point, characters are input one at a time and thrown away, until the first appearance of a carriage return character. If the ARGUMENTS and the COMMAND were successfully coded, the characters input will be those of the COMMAND after the first three. If an error arose at some point, the characters input will in addition include those constituting all the STATEMENT after the occurrence of the error.

On the appearance of a carriage return, a line feed character is output to the teletype. An error check is made, and if none occurred then the 'GOOD' exit from the BRM is taken. If an error did occur, a BRM 'ER1' is executed to print out an ERROR MESSAGE. The 'BAD' exit is then taken from the BRM 'INPUT' to the point specified at the start of the routine.

At either exit {A}, {B} and {X} are garbage.

The two following routines are called during the execution of the BRM 'INPUT'.

BRM 'LOOK' (Flow Diagram 7.)

The BRM 'LOOK' is a routine for examining the WORDS or parts of WORDS in a TABLE for a match with the WORD or part of WORD tested. The BRM is used in many parts of the processor to locate the names of VARIABLES, COMMANDS, and so on. At entry {A} is the WORD to be matched with a TABLE entry. {B} and {X} are garbage. The calling sequence is:

```
BRM  LOOK
ZRO  [mask]
ZRO  [INDEX of TABLE origin]
ZRO  [name of TABLE]
[BAD  return location]
[GOOD  return location]
 .
 .
 .
```

The operation of the routine is as follows. From the arguments of the BRM are obtained the absolute location of the zeroth WORD of the TABLE, the mask through which the match is to be taken, and the INDEX of the ORIGIN of the TABLE. Execution is then at point ⏀—(LK1).

The WORD in the A register is now compared with each WORD in the specified TABLE appropriately masked, starting at the ORIGIN, and stopping at the first match or after the zeroth WORD has been reached. If a match occurred, then the INDEX of the WORD where the match occurred is retained, and the 'GOOD' exit from the BRM taken. If no match occurred, the 'BAD' exit from the BRM is taken.

At a 'BAD' exit {A} are as at entry; and {B} and {X} are garbage. At a 'GOOD' exit {A} is the INDEX of the matched WORD; and {B} and {X} are garbage.

BRM 'IPC' (Flow Diagram 8.)

The BRM 'IPC' is a routine which tests an ARGUMENT of a STATEMENT, and if it is LABEL-type, converts the character codes of the ARGUMENT to a two-digit octal number. At entry {A}, {B} and {X} are garbage. {T} constitute the ARGUMENT to be tested and converted. The calling sequence is:

```
BRM IPC
[EXIT1 return location]
[EXIT2 return location]
.
.
.
```

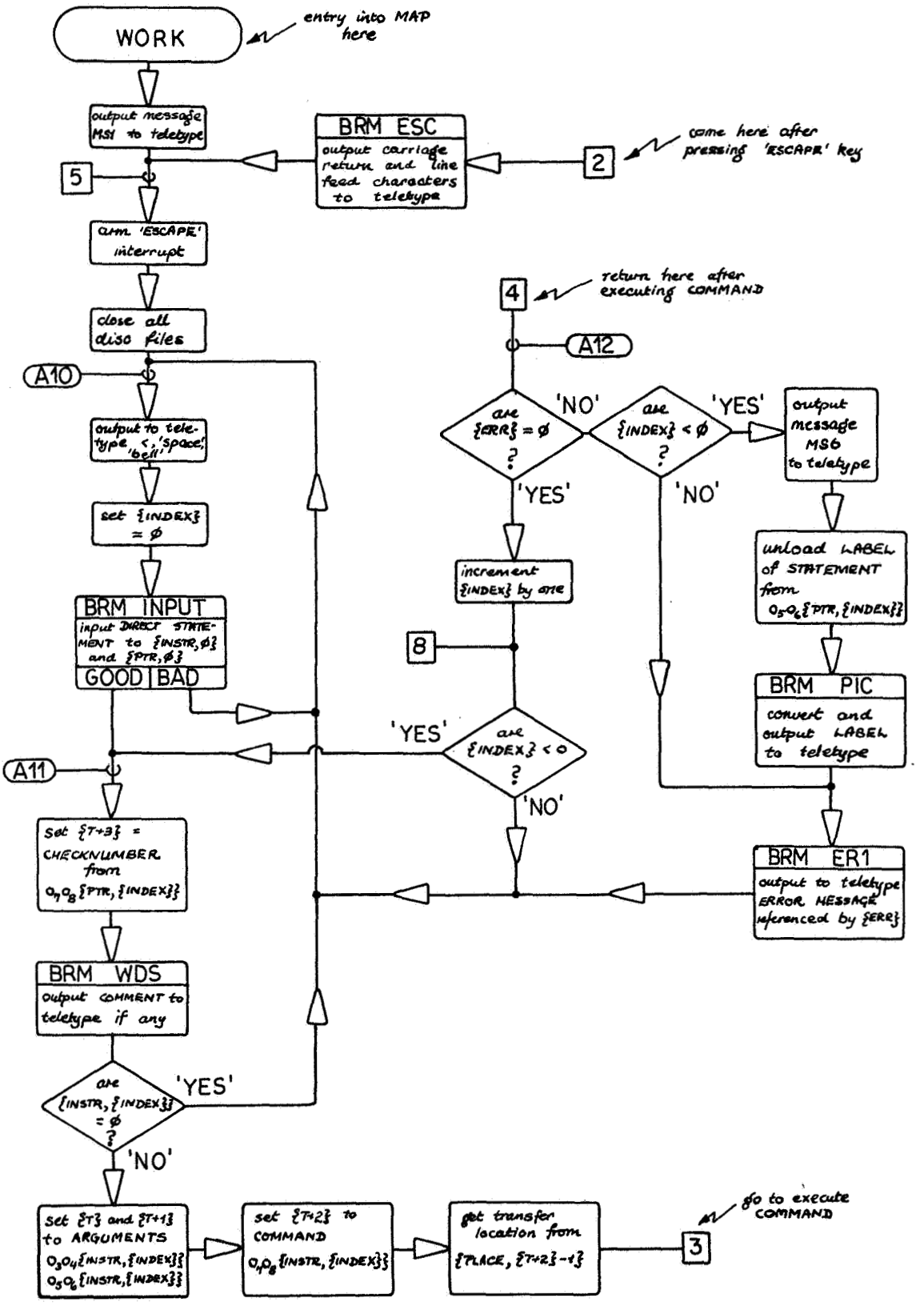
The operation of the routine is as follows. First a location storing the conversion result is initialized to zero. Execution then moves to the start of a loop at point ⏀-(IP1). The current result is multiplied by 12_8. The next leftmost character of the WORD to be converted is selected, and converted to an octal number. If the number is between ∅ and 11, then the character converted was numeric, and the octal number is added into the result. If the number is outside the above range, then the character code was non-numeric and the ARGUMENT was not LABEL-type. EXIT1 is taken from the BRM.

If conversion continues another test is next made. If fewer than two characters have been processed,

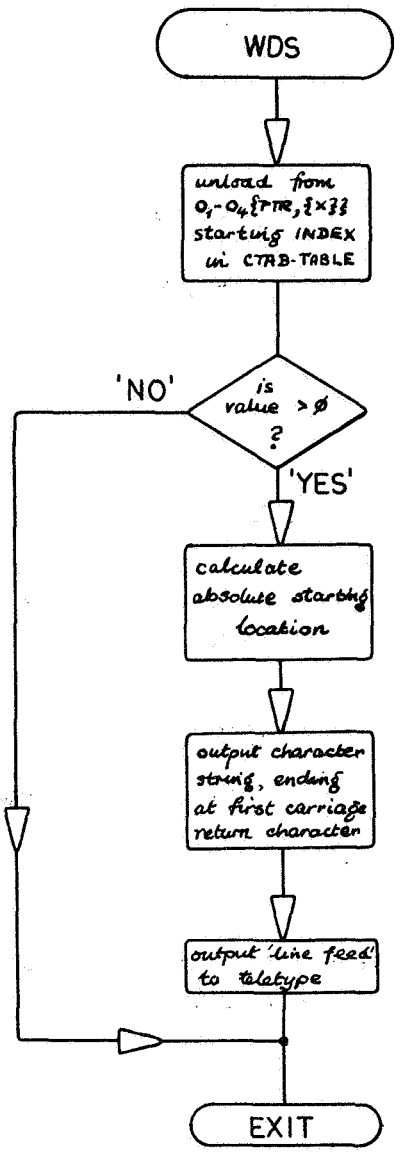
execution returns to point ⟨IP1⟩ and the loop is recycled. If two characters have been processed, the final result has been obtained. If its value is between 1 and 63 inclusive, it constitutes a valid LABEL. EXIT2 is taken from the BRM. If the value is outside this range, the LABEL is invalid, and EXIT1 is taken from the BRM.

At EXIT1 {A}, {B} and {X} are garbage; at EXIT2 {A} constitutes the coded LABEL-type ARGUMENT, and {B} and {X} are garbage.

This completes the detailed discussion of the operation of the DIRECTIVE SUBPROCESSOR. In the following sections the operation of the EXECUTIVE SUBPROCESSOR components will be discussed. These components correspond one for one with the COMMANDS available in MAP language, and carry out their execution.



Flow Diagram 2. Detailed flowchart of DIRECTIVE SUBPROCESSOR.



CALL SEQUENCE:

BRM WDS
[return location]

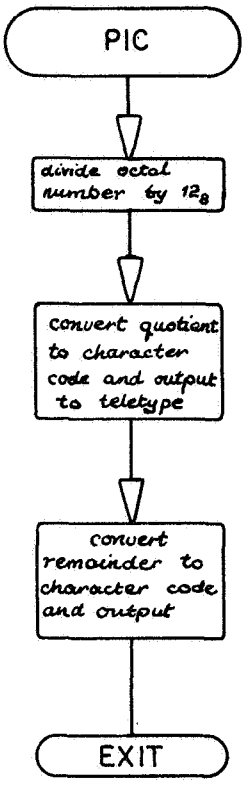
AT ENTRY:

{A},{B} garbage
{X} value of INDEX in PTR-TABLE

AT EXIT:

{A},{B} garbage
{X} unchanged

Flow Diagram 3. Flowchart of BRM 'WDS' for printing the COMMENT attached to an INDIRECT STATEMENT.



CALL SEQUENCE:

BRM PIC
[return location]

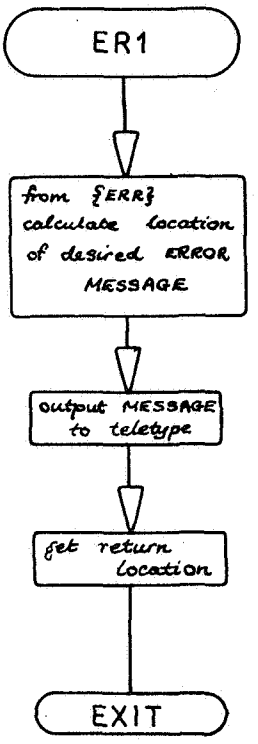
AT ENTRY:

{A} number for conversion
{X} not used
{B} garbage

AT EXIT:

{A},{B} garbage

Flow Diagram 4. Flowchart of BRM 'PIC' for conversion and output of an octal number.



CALL SEQUENCE:

```
BRM ER1
ZRO [address of return
 :          location]
 :
 :
```

AT ENTRY:

{A},{B},{X} garbage
{ERR} ERROR MESSAGE reference number

AT EXIT:

{A},{B},{X} garbage
{ERR} zero

Flow Diagram 5. Flowchart of BRM 'ER1' for printing out ERROR MESSAGES.

CALL SEQUENCE:

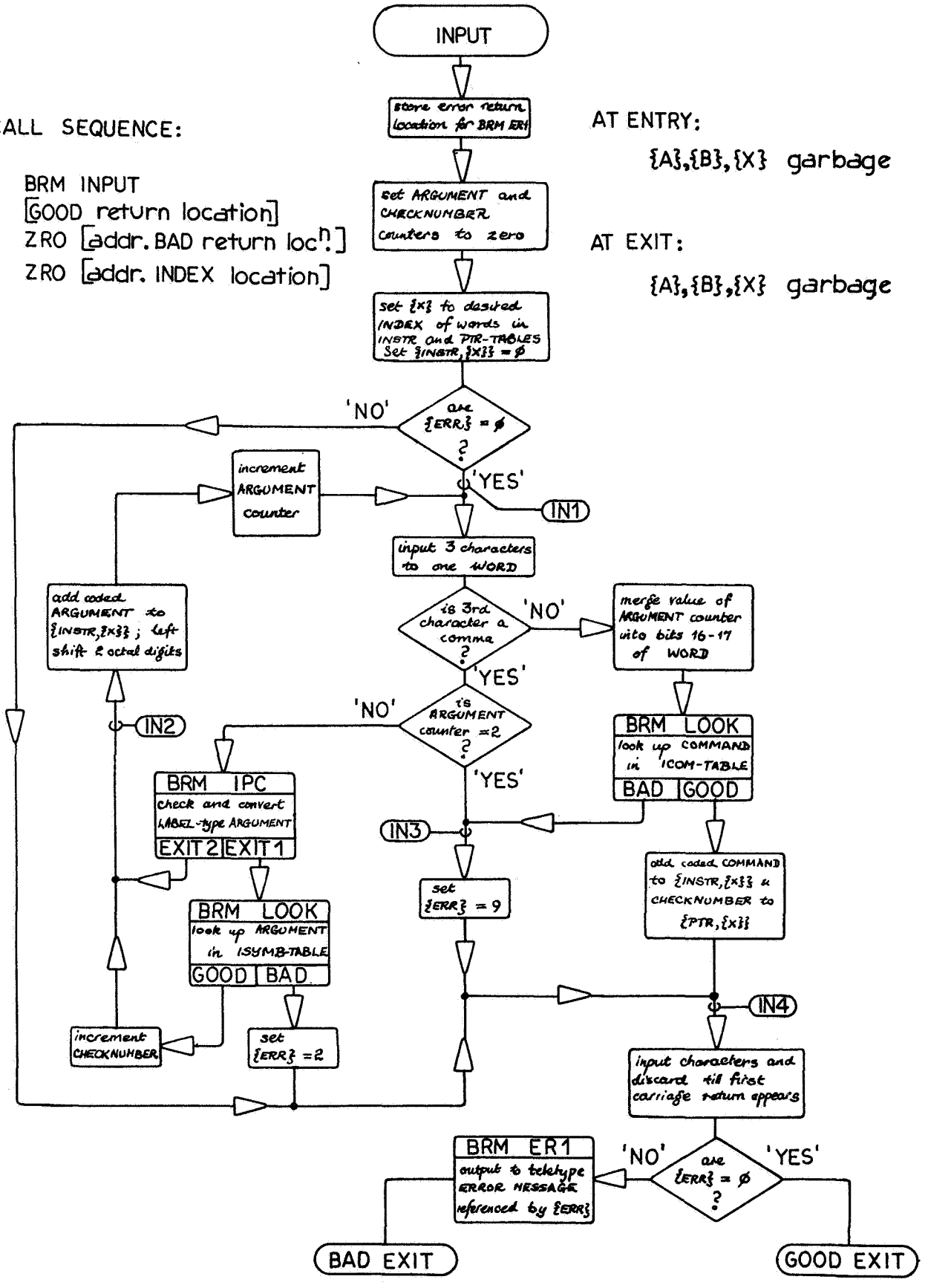
BRM INPUT
[GOOD return location]
ZRO [addr. BAD return loc^{n}.]
ZRO [addr. INDEX location]

AT ENTRY:

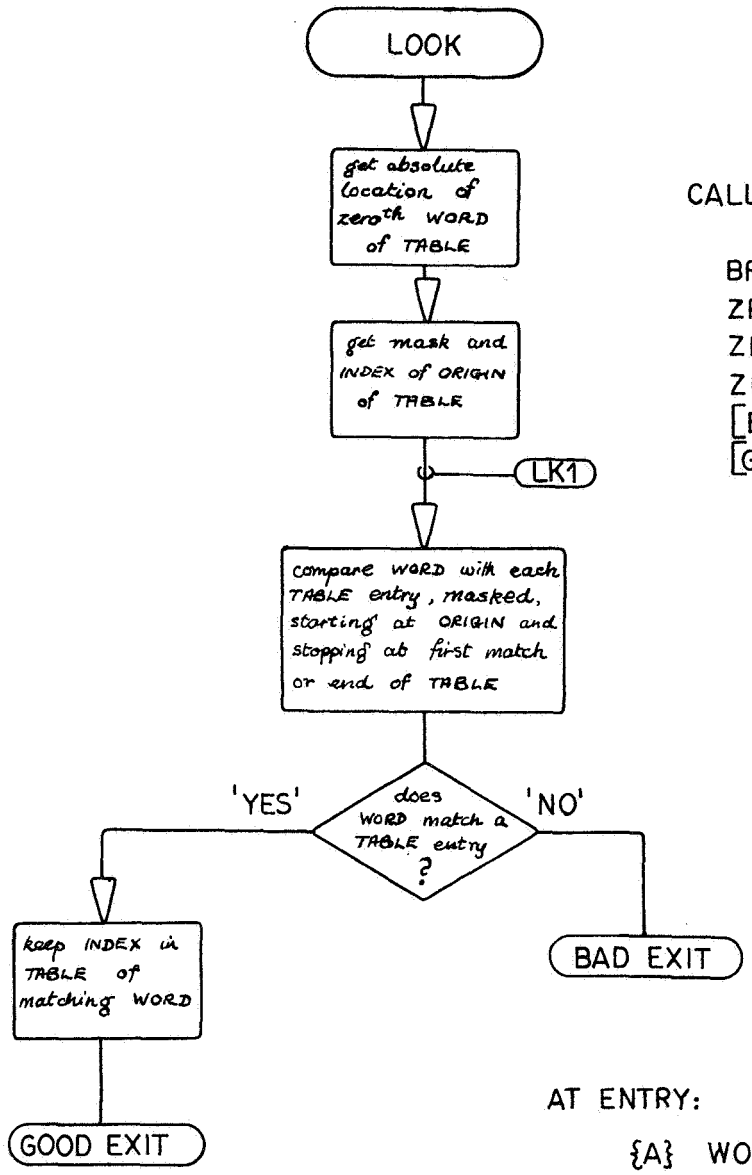
{A},{B},{X} garbage

AT EXIT:

{A},{B},{X} garbage



Flow Diagram 6. Flowchart of BRM 'INPUT' for input and coding of a STATEMENT.



CALL SEQUENCE:

```
BRM LOOK
ZRO [mask]
ZRO [INDEX of TABLE ORIGIN]
ZRO [name of TABLE]
[BAD return location]
[GOOD return location]
  .
  .
  .
```

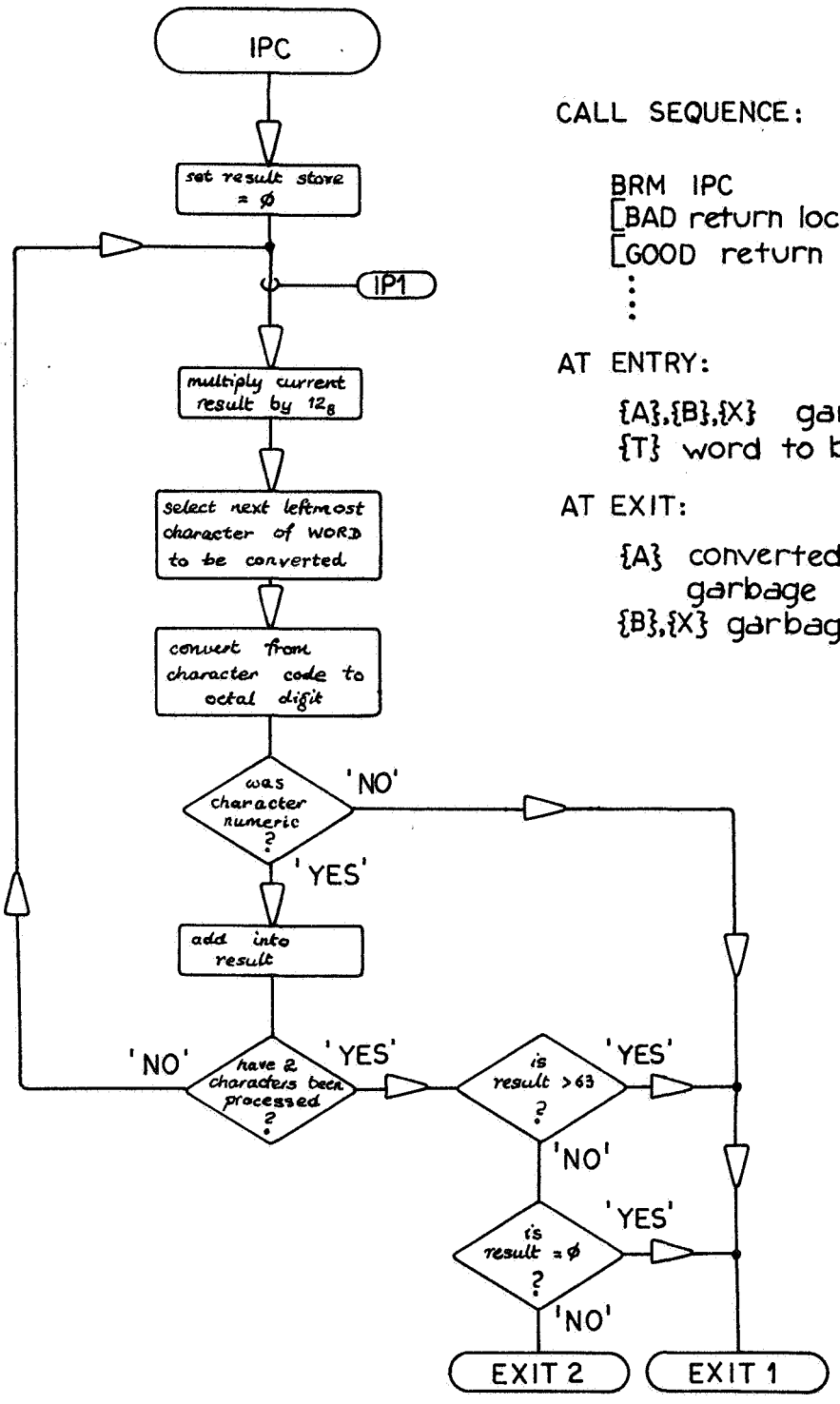
AT ENTRY:

{A} WORD to be matched
{B},{X} garbage

AT EXIT:

{A} INDEX of matching WORD, if one exists
{B},{X} garbage

Flow Diagram 7. Flowchart of BRM 'LOOK' for matching a WORD with a TABLE entry.



Flow Diagram 8. Flowchart of BRM 'IPC' for testing and converting a LABEL-type ARGUMENT to an octal number.

6. THE EXECUTIVE SUBPROCESSOR

THE EXECUTIVE SUBPROCESSOR comprises those components of the MAP processor dealing with the execution of coded STATEMENTS. Control of entry into any of the components is held by the DIRECTIVE SUBPROCESSOR whose operation has been described in the previous sections of this manual. Each component of the EXECUTIVE SUBPROCESSOR corresponds to a MAP language COMMAND. There exist groups of components which are almost entirely independent of each other, except in so far as they may call the same subroutines.

Discussion of the operation of the component groups is divided into two parts. Part I concerns the operation of those components not involving the manipulation of MAP matrices and scalars. Part II concerns all those components which do involve such manipulation. This classification is unrelated to any classification of COMMANDS to be found in the USER'S MANUAL, and is purely for descriptive convenience.

Part I: Components not Involving Manipulation of Matrices

EACH INDEPENDENT COMPONENT or component group will be described in turn, followed by the subroutines which they call. Each description will be titled with the COMMAND name or names. Flowcharts of the components will be found in Flow Diagrams 9-21.

VARiables (Flow Diagram 9.)

'VARiables' is a zero-ARGUMENT COMMAND for adding new VARIABLE names to the LIST of VARIABLES. At entry {LIST} constitute the INDEX

of the WORDS in the ISYMB- and IDIM-TABLES last filled. On first use of a 'VARiables' COMMAND, {LIST} = -6ø.

The operation of the COMMAND is as follows. First a check is made on {LIST} to see if the ISYMB- and IDIM-TABLES have already been completely filled, implied by {LIST} ≥ -1.

If so, a 'NO' branch is taken; the B register is loaded with an ERROR MESSAGE reference number, and execution directed to point —[9] in the 'APPend' component for completion of the error return.

If the TABLES are still only partially filled, the 'YES' branch is taken, and a BRM 'STAR' executed. This routine inputs a character from the teletype. If it is an asterisk denoting that the use of the 'VARiables' COMMAND has been completed, execution is returned to point —[4] in the DIRECTIVE SUBPROCESSOR via EXIT1 from the BRM. If the character is not an asterisk it is assumed that the user is typing another VARIABLE LIST entry. Two more characters are input to complete a whole WORD, and EXIT2 is taken from the BRM.

The processor now has in its possession the first three characters of the new VARIABLE LIST entry being typed in by the user. The first two characters should be the name of the new VARIABLE, and the last character either a carriage return if the VARIABLE is scalar, or "=" if it is a matrix.

Next the VARIABLE name is checked to see if it starts with an alphabetic character. If not the VARIABLE name is illegal and the processor sets {ERR} = 5, denoting an ERROR MESSAGE reference number. If the name is legal, it and its terminating character are separately stored.

The processor now executes a BRM 'LOOK' to find out if a VARIABLE of the same name has previously been stored in the ISYMB-TABLE. If the result is positive, the processor sets {ERR} = 5. If the result is negative, MAP accepts the new VARIABLE name. {ISYMB,{LIST}+1} are set equal to the VARIABLE name.

Now, whether or not an error has been found, the BRM 'DIMS' is entered to input the dimensions of the new VARIABLE. If an irremediable error occurs during execution of the BRM, or occurred before entry, the 'BAD' exit is taken from the BRM, and execution is directed to point —[9] in the 'APPend' component.

If no error occurs {IDIM,{LIST}+1} are appropriately filled, and the 'GOOD' exit taken from the BRM. This completes the input and coding of one entry in the VARIABLE LIST. Because the 'VARiable' COMMAND deals with multiple entries, execution is now directed back to point ⏀—(VAR) ready for a new entry to be accepted.

The section of coding whose operation has just been described calls three BRM'S. BRM 'LOOK' has been described in Section 5. A description of the other two now follows.

BRM 'STAR' (Flow Diagram 1ø.)

The BRM 'STAR' is a routine for input and checking of the first WORD of an entry typed in EDIT MODE by a MAP user. The call sequence is as follows:

```
BRM STAR
[EXIT1 return location]
[EXIT2 return location]
  :
  :
```

At entry the A, B, and X registers contain garbage. At EXIT1 they contain garbage again. At EXIT2 {A} constitute the third character input; {X} = ø ; {B} and {T} are the three characters input.

The operation of the routine is as follows. On entry a "bell" is output to the teletype to inform the user that the processor is waiting for input. The first character is input. If it is an asterisk, the contents of the WORDS of the INSTR- and PTR-TABLES INDEXED by {INSTRX} are set to zero; a carriage return and line feed are output to the teletype, and EXIT1 taken from the BRM.

If the first character input is not an asterisk, the remaining two characters required to fill up a WORD are input. The complete WORD and the last, terminating character are separately stored and EXIT2 taken from the BRM.

BRM 'DIMS' (Flow Diagram 11.)

The BRM 'DIMS' is a routine for the input and coding of the dimensions of a VARIABLE name, and the calculation of the BLOCK INDEX of the BLOCK of storage assigned to it in the S-TABLE. Its call sequence is as follows:

```
BRM DIMS
[BAD return location]
[GOOD return location]
  :
  :
```

At entry the A register contains the terminating character following the VARIABLE name, while {B} and {X} are garbage. At exit {A} and {X} are garbage, and {B} are either garbage, or an ERROR MESSAGE reference number.

The operation of the routine is as follows. First the terminating character is stored in a temporary location T+2. Then, if an error has already occurred prior to entry into the routine, a 'NO' branch is taken to point ψ-(DIM2). Otherwise, a test on the terminating character is next made. Correctly this may either be "=" or a carriage return. If it is not "=" execution is directed to point ψ-(DIM1).

If the character is "=", numbers representing the row and column dimensions of the VARIABLE are input by the user. The terminating character of the first number should be a comma, and of the second a carriage return. If either of these is incorrect, the processor sets {ERR} = 9 and branches to point ψ-(DIM2). If either of the two numbers is greater

than ten the processor sets {ERR} = 6 and branches to point ψ-(DIM2). At the end of this process if no error has occurred, temporary storage locations T+1 and T contain the row and column dimensions of the VARIABLE respectively, and execution has reached point ψ-[14].

Considering now execution from point ψ-(DIM1), the terminating character is either a carriage return or is incorrect. If it is the latter a 'NO' branch is taken; the processor sets {ERR} = 9 and branches to point ψ-(DIM2). If it is the former, then the VARIABLE is a scalar. The contents of both temporary storage locations T and T+1 are therefore set equal to one, and execution again arrives at point ψ-[14].

At this stage in the discussion execution has arrived either at point ψ-[14], or, if an error arose, at point ψ-(DIM2). Continuing from the latter point, all further characters of the LIST entry after the occurrence of the error are input and discarded one by one until the appearance of a carriage return. A line feed is output to the teletype. A BRM 'ER1' is executed printing out the ERROR MESSAGE referenced by {ERR} and a branch to point ψ-[15] made.

Continuing now the discussion of execution from point ψ-[14] the row and column dimensions are stored in IDIM,{LIST}+1. The BLOCK INDEX for the previous VARIABLE, given by O_1-O_4{IDIM,{LIST}} is obtained, and the product of the row and column dimensions added to it. The result is the BLOCK

INDEX in the S-TABLE for the current VARIABLE. If its value shows that use of the VARIABLE would lead to overfilling of the S-TABLE, a 'YES' branch is taken. The processor sets {B} = 4 representing an ERROR MESSAGE reference number and branches to point ⏀—[15].

If no overfilling is indicated {ENDS+1} and O_1-O_4{IDIM, {LIST}+1} are set equal to the new BLOCK INDEX. A line feed is output to the teletype, and {LIST} are incremented by one. The 'GOOD' exit from the BRM is then taken.

If any error occurred during or prior to execution of the BRM, execution arrives at point ⏀—[15]. The processor sets {IDIM, {LIST}+1} and {ISYMB, {LIST}+1} to zero and takes the 'BAD' exit from the BRM.

LISt (Flow Diagram 12.)

'LISt' is a zero-ARGUMENT COMMAND for output to the teletype of the LIST of VARIABLES. The operation merely consists of an execution of the BRM 'LIS1', followed by a branch to point —[4] of the DIRECTIVE SUBPROCESSOR.

BRM 'LIS1' (Flow Diagram 12.)

The BRM 'LIS1' is a routine which outputs to the teletype the LIST of VARIABLES. Its call sequence is as follows:

```
BRM LIS1
[return location]
  :
  :
```

The contents of the A, B, and X registers are garbage at both entry and exit.

The operation of the routine is as follows. First an INDEX counter is initialized to -6ø. A loop is now entered which deals with each entry of the VARIABLE LIST in turn. The INDEXED WORD in the ISYMB-TABLE is output to the teletype, followed by three "spaces". A POP 'UNLO' is executed to unload from the INDEXED WORD in the IDIM-TABLE the row and column dimensions of the VARIABLE, and its BLOCK INDEX in the S-TABLE. (This latter is not used).

The row dimension, a comma, the column dimension, a carriage return, and lastly a line feed are output in turn to the teletype. The INDEX counter is incremented by one.

If all the entries in the ISYMB- and IDIM-TABLES have been treated, exit from the BRM is taken; if not the processor branches back to point ⏀—[16] to handle the next pair of WORDS INDEXED.

POP 'UNLO' (Flow Diagram 13.)

The POP 'UNLO' is a routine for unloading a WORD from the IDIM-TABLE. The calling instruction is as follows:

UNLO [address of storing location]

⋮

At entry the A and B registers contain garbage, and the X register contains the INDEX of the WORD of the IDIM-TABLE to be unloaded. At exit all registers contain garbage.

Operation of the routine is as follows. {B} and {A} are respectively set equal to O_1-O_4{IDIM,{X}-1} and O_5-O_8{IDIM,{X}}. {B}, giving the BLOCK INDEX in the S-TABLE of the corresponding VARIABLE in the ISYMB-TABLE, are stored in the addressed location.

O_5O_6{A}, giving the row dimension of the VARIABLE, are stored in the addressed location plus one. O_7O_8{A}, giving the column dimension of the VARIABLE, are stored in the addressed location plus two. The exit from the POP is then taken.

APPend (Flow Diagram 14.)

'APPend' is a zero-ARGUMENT COMMAND for appending INDIRECT STATEMENTS to a PROGRAM of such STATEMENTS. Here again the structure is highly dependent on the use of a single BRM. At entry {INSTRX} constitute the INDEX of the WORDS in the INSTR- and PTR-TABLES into which the next INDIRECT STATEMENT is to be coded.

Operation of the COMMAND is as follows. Immediately on entry a loop is entered, which inputs a STATEMENT for each iteration. First {INSTRX}

are tested. If the value is negative, the INSTR- and PTR-TABLES have not yet been completely filled and more INDIRECT STATEMENTS can be stored. The 'YES' branch is taken, and the BRM 'PGRM' executed. This routine inputs and codes an INDIRECT STATEMENT typed by the user into INSTR,{INSTRX} and PTR,{INSTRX}. If during the execution of the BRM use of the COMMAND is terminated by the user, or an error occurs such that use of the 'APPend' COMMAND cannot be continued, EXIT1 is taken from the BRM and a branch is made back to point —[4] of the DIRECTIVE SUBPROCESSOR. If the user has indicated that further STATEMENTS are to be input, EXIT2 is taken from the routine and a branch back to point ⏀—(APP) is made.

If at any iteration through the loop described, {INSTRX} become non-negative, a 'NO' branch is taken to point —[10]. The processor sets {B} = 1 and arrives at point —[9]. {ERR} are set equal to {B} and a branch back to point —[4] of the DIRECTIVE SUBPROCESSOR made.

BRM 'PGRM' (Flow Diagram 15.)

The BRM 'PGRM' is a routine for handling the input and coding of an INDIRECT STATEMENT into specified WORDS of the INSTR- and PTR-TABLES. The call sequence is as follows:

```
BRM PGRM
[EXIT1 return location]
[EXIT2 return location]
ZRO [address of location containing INDEX]
 :
 :
```

At entry and both exits the contents of all registers are garbage.

Operation of the routine is as follows. On entry the user is required either to type in a new INDIRECT STATEMENT or an asterisk. A BRM 'STAR' is executed to test the first character input. If an asterisk appears no further INDIRECT STATEMENTS will be input during execution of the COMMAND using the BRM 'PGRM': EXIT1 is taken from the BRM 'STAR', followed by EXIT1 from the BRM 'PGRM'.

Otherwise the new user's input is considered to be a new INDIRECT STATEMENT. The first three characters are input and EXIT2 from BRM 'STAR' taken. The third character is tested; if it is not a colon an error condition has arisen and a 'NO' branch is taken to point ↓PGR1. If the character is a colon, the 'YES' branch is taken and a BRM 'IPC' executed. The first two characters input should be the LABEL of the STATEMENT. The BRM 'IPC' tries to convert the LABEL to its internal form. If unsuccessful, EXIT1 is taken and a branch made to the point ↓PGR1.

If the conversion is successful, the LABEL is provisionally accepted and stored in a temporary storage location. Next a BRM 'LOOK' is employed to check if such a LABEL has been used for a previous STATEMENT. If so, execution is directed to point ↓PGR1, where the processor sets {ERR} = 9, indicating an error; and thence to point ↓17. If not the LABEL is finally accepted as legal, and execution arrives at point ↓17.

At this point the next action is to obtain the address of the location containing the INDEX of the WORDS to be filled in the INSTR- and PTR-TABLES. This address is inserted in the call sequence of a BRM 'INPUT', and the BRM executed. If an error occurs, the 'BAD' exit is taken from BRM 'INPUT' and a branch back to the entry point of the BRM 'PGRM' made.

Otherwise the BRM 'INPUT' completes the input and coding of the INDIRECT STATEMENT, and takes the 'GOOD' exit. On return {INSTR,{**}} and {PTR,{**}} are the coded STATEMENT, where ** is the appropriate INDEX.

Finally, the LABEL is stored in O_5O_6{PTR,{**}}; {INSTRX} are incremented by one, and EXIT2 taken from the routine.

INSert
COMment
BRAnch
DELete (Flow Diagram 16.)

The 'EDIT' component group of the EXECUTIVE SUBPROCESSOR comprises coding for the three one-ARGUMENT COMMANDS 'INSert', 'COMment' and 'BRAnch' and coding for the two-ARGUMENT COMMAND 'DELete'. The processor starts by executing operations that are the same for any of the four COMMANDS, and then a secondary transfer is made to complete execution of the particular COMMAND.

Execution is as follows. First the secondary transfer index number is unloaded and stored. Next a check is made on the CHECKNUMBER for the STATEMENT being executed. If it is non-zero, the STATEMENT has the wrong type of ARGUMENT. The 'NO' branch is taken to point —[7]. The processor sets {B} = 9 denoting an error, and branches to point —[9] in the 'APPend' component of the EXECUTIVE SUBPROCESSOR.

Otherwise the 'YES' branch is taken, and an ARGUMENT counter set to zero. A loop is now entered which takes the ARGUMENTS in turn, ignores them if they are zero, (which only happens if that ARGUMENT does not exist) and by means of a BRM 'LOOK' obtains the INDEX of the WORD in the PTR-TABLE which contains that ARGUMENT in O_5O_6. In other words, the processor finds the STATEMENT which has a LABEL the same as the ARGUMENT. If an ARGUMENT cannot be found in the TABLE an error condition arises. A branch to point ϕ—(ED1) is made; the processor sets {B} = 7, and execution is directed to point —[9] of the 'APPend' component of the EXECUTIVE SUBPROCESSOR. The INDICES obtained are stored, and the secondary transfer is now made.

Each COMMAND will now be dealt with in turn.

INSert

The 'INSert' COMMAND is used to insert one or more INDIRECT STATEMENTS into a PROGRAM immediately following the STATEMENT with the

LABEL specified by the ARGUMENT of the 'INSert' COMMAND. At the point ⏚—(INS) {T+1} constitute the INDEX in the INSTR- and PTR-TABLES of the coded STATEMENT after which the insertion is to occur.

Inserted STATEMENTS are handled one at a time by a loop. Before entry into the loop, two counters are initialized:

{T+2}	are set equal to	{T+1}
{T+3}	are set equal to	{T+1} + 1

The loop is entered, and a test is made on {INSTRX}. If the contents are not less than zero, then the INSTR- and PTR-TABLES are full and insertions are illegal. The 'NO' branch is taken and execution is directed to point —[1Ø] in the 'APPend' component of the EXECUTIVE SUBPROCESSOR.

Otherwise the 'YES' branch is taken. Next, all the WORDS of the INSTR- and PTR-TABLES following the ones INDEXED by {T+2} are moved forward one TABLE entry to accommodate the next STATEMENT to be inserted. A BRM 'PGRM' is executed to input the INDIRECT STATEMENT to be inserted, and code it into INSTR,{T+2} and PTR,{T+2}. From this BRM, EXIT2 is taken if further STATEMENTS are yet to be inserted by the 'INSert' COMMAND. The counters T+2 and T+3 are incremented by one, and a branch back to the beginning of the loop made.

If the insertion is complete, EXIT1 is taken from the BRM 'PGRM'. {INSTRX} are incremented by one. Because the process described above results a gap in the INSTR- and PTR-TABLES immediately after the end of the insertion, the 'DELete' component of the EXECUTIVE SUBPROCESSOR is now used to eliminate it. This process completes execution of the 'INSert' COMMAND.

BRAnch

The 'BRAnch' COMMAND is used to redirect the flow of execution of the INDIRECT STATEMENTS of a PROGRAM, or to enter such a PROGRAM. At the point ⏚—(BRA) {T+1} constitute the INDEX of the WORDS in the INSTR- and PTR-TABLES which contain the coded STATEMENT next to be executed.

Execution of the 'BRAnch' COMMAND is as follows. {INDEX} are set equal to {T+1}. Execution is then directed back to point —[8] of the DIRECTIVE SUBPROCESSOR.

DELete

The 'DELete' COMMAND is used to delete sequences of INDIRECT STATEMENTS from a PROGRAM. At the point ⏚—(DEL) {T} constitute the INDEX in the INSTR- and PTR-TABLES of the WORDS containing the coded STATEMENT first in the NATURAL SEQUENCE to be deleted. {T+1} constitute the INDEX of the WORDS containing the coded STATEMENT last in the NATURAL SEQUENCE to be deleted. If only one STATEMENT is to be deleted from the PROGRAM, {T} = {T+1}.

Execution of the 'DELete' COMMAND is as follows. First {T+1} are incremented by one. If {T+1} $\neq$ {T}, showing that the LABELS given in the ARGUMENTS of the 'DELete' COMMAND have been input in reverse order, a 'NO' branch is taken to point —[7] in the 'EDIT' component group. Otherwise the 'YES' branch is taken, and the deletion takes place.

The deletion is effected by moving backward in turn the sequences of WORDS INSTR,{T+1} through INSTR,-1 and PTR,{T+1} through PTR,-1 so that the sequences start at INDEX {T}. The vacant WORDS appearing at the front of the TABLES are set to zero, while the previous contents of the WORDS with INDICES {T} through {T+1}-1 are obliterated. At the end of this process {INSTRX} are set to the previous value minus the number of entries deleted in each TABLE, and a branch back to point —[4] in the DIRECTIVE SUBPROCESSOR made.

COMment

The 'COMment' COMMAND is used to attach a COMMENT to an INDIRECT STATEMENT. This COMMENT will then be automatically output by the DIRECTIVE SUBPROCESSOR immediately prior to the execution of that STATEMENT.

At the point ϕ—(COM) {T+1} constitute the INDEX of the WORDS in the INSTR- and PTR-TABLES containing the coded STATEMENT to which the COMMENT is to be attached.

Execution of the 'COMment' COMMAND is as follows.[11] The difference of the string pointers of the CTAB-TABLE

[11] It is useful to read the following description in conjunction with the SDS 940 TECHNICAL MANUAL [3] and the STRING PROCESSING REFERENCE MANUAL [6].

is obtained. If the value is greater than 2927 there may not be room for the COMMENT the user wishes to attach. A 'YES' branch is thus taken; the processor sets {ERR} = 1ø, denoting an ERROR MESSAGE reference number, and branches to point —[4] of the DIRECTIVE SUBPROCESSOR.

If the difference is not greater than 2927, the 'NO' branch is taken. O_1-O_4{PTR,{T+1}} are set equal to this difference, and a "bell" is output to the teletype to signal to the user to input the COMMENT characters. The characters are input, counted and stored in the CTAB-TABLE at the end of the existing string until either a carriage return character appears, or the count reaches 72. In either case, a carriage return is added to the end of the COMMENT string.

Finally a line feed is output to the teletype, and execution is directed back to point —[4] of the DIRECTIVE SUBPROCESSOR.

This completes the discussion of this group of four components.

PROgram (Flow Diagram 17.)

'PROgram' is a zero-ARGUMENT COMMAND used to output to the teletype the PROGRAMS of INDIRECT STATEMENTS typed in by the user. Execution is as follows. A loop handles the decoding and output of each pair of WORDS from the INSTR- and PTR-TABLES in turn. The contents of the X register INDEX the pair of WORDS handled. Initially {X} = -63.

First the BRM 'WDS' is executed to output the COMMENT to the teletype, if one is attached to the STATEMENT. Next a test is made: if O_5-O_8{PTR,{X}} are zero implying that a vacant pair of WORDS in the INSTR- and PTR-TABLES has been reached, a 'YES' branch is taken and execution is directed back to point —[4] in the DIRECTIVE SUBPROCESSOR. This condition arises only when all the STATEMENTS of a PROGRAM of less than maximal length have already been output.

If the contents are non-zero the 'NO' branch is taken. A BRM 'PIC' is used to output the LABEL of the STATEMENT coded into O_5O_6{PTR,{X}} to the teletype. This is followed by the output of a colon. The remainder of the contents of the pair of WORDS are then unloaded in the following way:

{T+1}	are set equal to	O_7O_8{INSTR,{X}}
{T+2}	are set equal to	O_7O_8{PTR,{X}}
{A}	are loaded with	O_1-O_6{INSTR,{X}}

A then contains the two coded ARGUMENTS of the STATEMENT, T+1 contains the coded COMMAND, and T+2 the CHECKNUMBER. Temporary storage location T+3 is used as an ARGUMENT counter and initialized to zero.

The CHECKNUMBER is now tested. If it is non-zero, then there is at least one VARIABLE-type ARGUMENT of the STATEMENT; a 'YES' branch is taken to the point ⏀—(PR1). If it is zero then either there are no

ARGUMENTS or there is at least one LABEL-type ARGUMENT. The 'NO' branch is taken. The ARGUMENTS are unloaded from the A register in turn and output to the teletype by means of a BRM 'PIC'. ARGUMENTS with zero value are in actuality non-existent, and are not output. After each ARGUMENT a comma is output. After all ARGUMENTS are output execution is directed to point ⏀—(PR2).

Considering now execution from the point ⏀—(PR1), again each ARGUMENT is unloaded in turn from the A register. Zero ARGUMENTS are again ignored. This time the contents of the WORD of the ISYMB-TABLE indexed by the ARGUMENT are obtained, shifted left one character code, and merged with the character code for a comma. The result is output to the teletype. After all ARGUMENTS have been output in this fashion, execution arrives at point ⏀—(PR2).

At this point the COMMAND is output: {ICOM,{T+1}} are obtained and the value of the ARGUMENT counter reached after the above processes subtracted from bits 16 and 17. The result is output to the teletype, followed by a carriage return and line feed. The X register is incremented by one. If it is still negative there may be further STATEMENTS to be output, and the 'YES' branch is taken back to the beginning of the loop.

If not, the 'NO' branch is taken, and execution is directed back to point —[4] of the DIRECTIVE SUBPROCESSOR. This latter condition can only arise if the PROGRAM is of maximal length.

SAVe (Flow Diagram 18.)

'SAVe' is a zero-ARGUMENT COMMAND used to store on a disk file a PROGRAM of INDIRECT STATEMENTS, together with other data required to retrieve the PROGRAM and restore it to the core satisfactorily. First the contents of the INSTR- and PTR-TABLES are stored; next the contents of the CTAB-TABLE ; and finally the contents of the ISYMB- and IDIM-TABLES.

Operation of the COMMAND is as follows. On entry at point ⏀—(SAV) the BRM'OUTF' is executed to ready the output file. Next the file identifying WORD 31663547_8 is written on the file, followed by {INSTRX}. WORDS from the INSTR- and PTR-TABLES are alternately written on the file, starting with {INSTR, -63} and {PTR, -63} and ending with {INSTR, {INSTRX}} and {PTR, {INSTRX}}. This completes storage of the PROGRAM itself.

The total number of COMMENT characters in all COMMENTS is now calculated and stored on the file. The contents of the CTAB-TABLE are then written on the file WORD by WORD, three characters at a time. This completes storage of the COMMENTS attached to the STATEMENTS of the PROGRAM.

Finally, the LIST of VARIABLES is stored, as follows. {LIST} are written on the disk file, followed by alternate WORDS from the ISYMB- and IDIM-TABLES, starting with {ISYMB, -6Ø} and {IDIM, -6Ø} and ending with {ISYMB, {LIST}} and {IDIM, {LIST}}.

The disk file is then closed and a branch back to point —[4] in the DIRECTIVE SUBPROCESSOR made.

The 'SAVe' COMMAND uses one BRM.

BRM 'OUTF' (Flow Diagram 19.)

The BRM 'OUTF' is a routine for making ready a disk file for output from the core. The call sequence is as follows:

```
BRM OUTF
[return location]
  :
```

Both at entry and at exit all registers contain garbage.

Operation of the routine is as follows. On entry, the message MS5 is output to the teletype:

FILE NAME

A BRS 18 is executed to read the output file name from the teletype. If the file name is acceptable, the 'GOOD' exit is taken. The file type (symbolic) is loaded, and a BRS 19 executed to open the file for output. If the file is successfully opened, the 'GOOD' exit is taken. A carriage return, and line feed are output to the teletype, and the exit from the BRM taken.

If the file name is unacceptable, or cannot be opened, a 'BAD' exit is taken from the BRS concerned. {ERR}

are set equal to 9 denoting an ERROR MESSAGE reference number. A BRM 'ER1' is executed to output an ERROR MESSAGE to the teletype, and execution returns to the start of the BRM to try and introduce a new file name.

REStore (Flow Diagram 2ø.)

'REStore' is a zero-ARGUMENT COMMAND used to recover a PROGRAM of INDIRECT STATEMENTS and other information to make it usable, from a disk file. It is complementary to the 'SAVe' COMMAND. First the contents of the INSTR- and PTR-TABLES are recovered; next the contents of the CTAB-TABLE; and lastly the contents of the ISYMB- and IDIM-TABLES. The previous contents of these TABLES are deleted. On the user's requirement, the VARIABLE LIST may then be redimensioned: this involves the reconstruction of the IDIM-TABLE.

Operation of the 'REStore' COMMAND is as follows. On entry at point ⏀-(REL) the BRM 'INF' is executed to make ready the disk file for input to the core. The first WORD on the file should be the file identifier 31663547_8. If it is not, the file is not one created by a 'SAVe' COMMAND. The 'NO' branch is taken; the file is closed, ERROR MESSAGE 9 output to the teletype, and execution directed back to point ⏀-(REL). If the file identifying WORD is correct, the 'YES' branch is taken and {INSTRX} set equal to the contents of the next file WORD. This gives the number of WORDS in the INSTR- and PTR-TABLES to be filled from the file.

The WORDS of the INSTR- and PTR-TABLES are now filled alternately from successive WORDS of the disk file, starting at {INSTR, -63} and {PTR, -63}, and terminating at {INSTR, {INSTRX}} and {PTR, {INSTRX}}. The remaining WORDS of the INSTR- and PTR-TABLES are set to zero.

The next WORD input from the disk file is the number of COMMENT characters in the CTAB-TABLE. The characters themselves are read into the CTAB-TABLE from the file, three at a time, WORD by WORD. The remainder of the CTAB-TABLE is set to zero. The string pointers for the TABLE are now reinstated.

This completes the input from the file of the PROGRAM and its attached COMMENTS. It now remains to input the VARIABLE LIST. The next WORD input from the file into {LIST} gives the length of the VARIABLE LIST. WORDS are now read from file alternately into the ISYMB- and IDIM-TABLES, starting at {ISYMB, -6ø} and {IDIM, -6ø} and ending at {ISYMB, {LIST}} and {IDIM, {LIST}}. The remaining WORDS of the ISYMB- and IDIM-TABLES are set to zero. The disk file is then closed.

The LIST of VARIABLES is now output to the teletype: first the message MS1ø:

VARIABLES USED

and then the LIST itself by means of a BRM 'LIS1'. Execution now arrives at the point ⏚-(REL1). The user must now decide whether he wants to redimension the VARIABLE LIST. Message MS11 is output to the teletype:

REDIMENSION VARIABLES ?

Characters are now input one by one until a carriage return appears. A line feed is now output to the teletype. If the characters input constitute the word 'NO' a branch to point —[4] of the DIRECTIVE SUBPROCESSOR is made. If the characters constitute neither the word 'NO' nor the word 'YES' execution reaches point ψ—(REL2) and {ERR} are set equal to 9 denoting an ERROR MESSAGE reference number. If the characters constitute the word 'YES' the VARIABLE LIST is now redimensioned.

Each entry in the ISYMB-TABLE is treated in turn excepting the standard output VARIABLE, starting at {ISYMB, -59} and ending at {ISYMB, {LIST}}. The name of the VARIABLE is output from the current WORD in the ISYMB-TABLE, followed by a "bell". The terminating character of the name, either "=" or a carriage return, is input by the user, followed by the new dimensions, the latter by means of the BRM 'DIMS'. At the end of this process the IDIM-TABLE has been completely reconstructed, and execution returns to point —[4] of the DIRECTIVE SUBPROCESSOR.

If the 'BAD' exit was taken from BRM 'DIMS' abnormally stopping the redimensioning process, or if execution passed the point ψ—(REL2) then a BRM 'ER1' is executed to output an ERROR MESSAGE to the teletype. Execution then returns to point ψ—(REL1) to repeat the redimensioning process.

The 'REStore' operation calls four subroutines. Of these only BRM'INF' has not yet been described.

BRM 'INF' (Flow Diagram 21.)

The BRM 'INF' is a routine for making ready a disk file for input to the core. The call sequence is as follows:

```
BRM  INF
[return location]
  :
  :
```

Both at entry and at exit all registers contain garbage.

The operation of the routine is as follows. On entry the message MS5 is output to the teletype:

FILE NAME

A BRS 15 is executed to read the input file name from the teletype. If the file name is acceptable, the 'GOOD' exit is taken. Then a BRS 16 is executed to open the file for input. If the file is successfully opened, the 'GOOD' exit is taken. A carriage return and a line feed are output to the teletype, and the exit from the BRM taken.

If the file name is unacceptable, or cannot be opened, a 'BAD' exit is taken from the BRS concerned. The processor sets {ERR} = 9 denoting an ERROR MESSAGE reference number. A BRM 'ER1' is executed to output the indicated ERROR MESSAGE

to the teletype, and execution returns to the start of the BRM to try and introduce a new file name.

This concludes the description of those COMMANDS not involving the manipulation of the matrices of MAP .

Introduction to Floating-Point Coding

THOSE COMPONENTS of the EXECUTIVE SUBPROCESSOR involving the actual manipulation of MAP matrix and scalar quantities have not yet been described. In these components repeated use is made of certain standard forms of ARPAS coding specially adapted to the processing of double-location WORDS in the S-TABLE and other temporary storage TABLES . In the SDS 940 system all floating-point operations are software generated: a range of BRS's and SYSPOPS handle the operations of addition, subtraction, multiplication, division, and several others[12].

The loading and storing of a floating point number are achieved by the 'LDP' and 'STP' SYSPOPS respectively. F_1 of the WORD is moved to and from the A register; F_2 and E of the WORD are moved to and from the B register.

To carry out an operation on a specific WORD in a TABLE of TYPE I or II WORDS, the INDEX of that WORD is loaded into the X register. The address field of the instruction executing the desired operation contains the name of the TABLE modified by the contents of the X register. To load the X register either the

[12] The operation of these system routines is described in the SDS 940 TECHNICAL MANUAL [3], and the FLOATING POINT SYSTEM MANUAL [2].

instruction 'CAX' (copy A to X) or the instruction 'LDX **' (load X from address **) might be used. This is standard programming practice. In the MAP processor two POPS have been defined so that operations on specific WORDS in TABLES of TYPE III WORDS may be similarly programmed.

POP 'FLDX' (Flow Diagram 22.)

The POP 'FLDX' is the analogue of the 'LDX' instruction. The contents of the addressed location are multiplied by two and placed in the X register. The addressed location, and the A and B registers are left intact. The calling instruction is:

```
FLDX  [address of location containing
 :              value to be loaded]
```

POP 'FCAX' (Flow Diagram 22.)

The POP 'FCAX' is the analogue of the 'CAX' instruction. The contents of the A register are multiplied by two and placed in the X register. The A and B registers contain garbage at the end of the operation. The calling instruction is:

```
FCAX
 :
```

EXAMPLE: To get the WORD INDEXED by {TEMP} from a TABLE -

TYPE I/II	TYPE III
LDX TEMP	FLDX TEMP
LDA TABLE,2	LDP TABLE,2
:	:

In general, since matrix operations consist of repetitive operations on their elements, there will be many nested loops in the part of the EXECUTIVE SUBPROCESSOR which deals with MAP matrix manipulation. These loops are handled in a standard way, in the same manner as loops are handled in a Fortran program. In fact some sections of coding in the processor were written by translating Fortran programs into the standard structures of ARPAS code to be described. This applies in particular to the 'DETerminant', 'INVert' and 'EIGenvalue' COMMANDS.

The process of carrying out a particular operation on each of the elements of a matrix VARIABLE in the S-TABLE in turn will now be described. This is readily generalized to other related processes. Assume the following:

{T} constitute the BLOCK INDEX in the S-TABLE of the VARIABLE occurring immediately before the VARIABLE to be processed. This INDEX is the INDEX of the WORD in the S-TABLE immediately before the first WORD of the BLOCK of storage devoted to the VARIABLE to be processed.

{T+1} is the number of elements of the matrix VARIABLE, that is, the number of WORDS in the BLOCK devoted to that VARIABLE.

T+2 is a temporary storage location.

Then the following section of coding will load and operate on each of the elements of the matrix VARIABLE in turn.

EXAMPLE:

```
            CLA
            STA  T+2
RETN        MIN  T+2
            LDA  T+2
            ADD  T
            FCAX
            LDP  S,2
            [operate on element obtained from TABLE]
            LDA  T+2
            SKE  T+1
            BRU  RETN
             :
             :
```

It is easy to see how this structure may be extended to nested loops and other forms. Sometimes it is useful to generate row and column indices separately, and then to calculate from these the INDEX in the S-TABLE of the element of the VARIABLE. The POP 'CALC' of the processor handles this process.

POP 'CALC' (Flow Diagram 23.)

The POP 'CALC' converts the row and column indices of an element of a matrix VARIABLE to the INDEX of that element in the S-TABLE. The calling instruction is:

```
CALC  [address of location containing
 :                column size]
 :
```

At entry {A} and {B} are the row and column index values, respectively. At exit {A} constitute the INDEX of the element in the S-TABLE, and {B} are garbage. The X register is not used. Operation is as follows: one is subtracted from the row index; the result is multiplied by the column dimension of the matrix, and the column index added. This result is

the INDEX in the S-TABLE relative to the BLOCK INDEX of the previous VARIABLE in the TABLE.

The following example shows the use of the POP 'CALC' in conjunction with a double loop. Each element of the matrix VARIABLE is loaded and operated on in turn row by row. For this example assume the following:

{T} constitute the BLOCK INDEX in the S-TABLE of the VARIABLE occurring immediately before the VARIABLE to be processed.

{T+1} constitute the row dimension of the matrix VARIABLE.

{T+2} constitute the column dimension of the matrix VARIABLE.

T+3 and T+4 are temporary storage locations.

EXAMPLE:

```
        CLA
        STA  T+3
RET1    MIN  T+3
        CLA
        STA  T+4
        MIN  T+4
        LDP  T+3
        CALC T+2
        ADD  T
        FCAX
        LDP  S,2
        [operate on element obtained from TABLE]
        LDA  T+4
        SKE  T+2
        BRU  RET1+3
        LDA  T+3
        SKE  T+1
        BRU  RET1
         :
         :
```

This completes the description of the basic coding structures involved in execution of COMMANDS dealing with the manipulation of MAP matrices. It now remains to describe the operation of the COMMANDS themselves.

Part II: Components Involving the Manipulation of Matrices

THE BASIC CODING structures from which the components of the EXECUTIVE SUBPROCESSOR executing matrix manipulation operations are built up have been described. Flowcharts of the components themselves are shown in Flow Diagrams 24-35. In the description that follows, the constituent basic coding structures are in general not described step by step, but mostly as complete functional units.

OMIt (Flow Diagram 24.)

'OMIt' is a single-ARGUMENT COMMAND used to delete an entry from the LIST of VARIABLES. At the same time the COMMAND erases from the S-TABLE the BLOCK of storage associated with the deleted VARIABLE, and closes up the gap.

Operation of the COMMAND is as follows. At first entry at point ⟨OMI⟩ a BRM 'LAB' is executed to check the type of ARGUMENT. If the ARGUMENT is LABEL-type the 'BAD' exit is taken, and execution proceeds to point [7] in the 'EDIT' component group. If the ARGUMENT is correctly VARIABLE-type, the 'GOOD' exit is taken from the BRM. The

contents of the X register are set equal to the INDEX of the WORD in the ISYMB-TABLE immediately after the one to be deleted.

A check is now made on the value of this INDEX. If it is not greater than -59 a 'NO' branch is taken and execution returns to point —[4] of the DIRECTIVE SUBPROCESSOR. This condition can only arise if the user tries to delete the standard output VARIABLE from the LIST: the action is to ignore the 'OMIt' COMMAND.

If the value of the INDEX is greater than -59 a 'YES' branch is taken and the deletion process takes place. O_1-O_4{IDIM,{X}-2}, O_1-O_4{IDIM,{X}-1} and their difference are temporarily stored in locations Q, Q+1 and T respectively. The last quantity is the number of WORDS by which all the entries in the S-TABLE after the ones deleted must be moved backward to fill the gap created.

Next the ISYMB- and IDIM-TABLES are changed as follows. For all values of ** starting at {X} and ending at -1, {ISYMB,**-1} are replaced with {ISYMB,**}, and {IDIM,**-1} are replaced with {IDIM,**} - {T} if {IDIM,**} $\neq \emptyset$ or zero otherwise. One is subtracted from {LIST}.

Finally the S-TABLE is updated as follows. For all values of ** starting at {Q+1}+1 and ending at {ENDS+1}, {S,**} are moved backward to {S,**-T}, thus obliterating the BLOCK of storage allocated to the deleted VARIABLE.

{ISYMB,Ø} and {IDIM,Ø} are set equal to zero and a branch back to point —[4] of the DIRECTIVE SUBPROCESSOR made.

The 'OMIt' COMMAND uses one routine not already described.

BRM 'LAB' (Flow Diagram 25.)

The BRM 'LAB' is a routine for checking the type of an ARGUMENT of a STATEMENT immediately prior to the execution of its COMMAND. The call sequence is:

```
BRM LAB
[return location]
  :
```

Both at entry and at exit all registers contain garbage.

Operation of the BRM is as follows. On entry {T+3} constitute the CHECKNUMBER of the STATEMENT. This is tested: if it is zero, the ARGUMENT is LABEL-type; the 'NO' branch is taken to point —[7] in the 'EDIT' component group. If it is non-zero, then the ARGUMENT is correctly VARIABLE-type. A 'YES' branch is taken, and {T+1} and {T+2} replaced by their respective two's complements. The exit from the BRM is then taken.

EIGenvalue
DETerminant
INVert
DIAgonal sum (Flow Diagrams 26-29.)

The 'GROUPØ' component group comprises three single-ARGUMENT COMMANDS 'DETerminant',

'INVert' and 'DIAgonal sum', where the result of the operation is placed in the standard output VARIABLE; and one single-ARGUMENT COMMAND 'EIGenvalue' where the standard output VARIABLE is used as working space, and the results are output to the teletype.

On entry, a preliminary section of coding common to any of the four COMMANDS is executed. A secondary transfer is then made to further sections of coding dealing separately with each individual COMMAND.

Flow diagram 26 shows the preliminary section of coding. Operation is as follows. On entry at point ⏀—(GRPØ) the secondary transfer index number is stored, and a BRM 'LAB' executed to check the type of the ARGUMENT. If the ARGUMENT is found to be LABEL-type, the 'BAD' exit is taken, and a branch to point —[7] in the 'EDIT' component group made. If the ARGUMENT is VARIABLE-type the 'GOOD' exit is taken from the BRM.

Next a POP 'UNLO' is executed to unload the dimensions of the VARIABLE and its BLOCK INDEX in the S-TABLE from the IDIM-TABLE. A test is then made on these dimensions: if the VARIABLE is not a square matrix a 'NO' branch is taken; the processor sets {B} = 8 and a branch to point —[9] in the 'APPend' component is made.

If the VARIABLE is a square matrix, the 'YES' branch is taken and a further test made. If the COMMAND executed is not 'INVert', a BRM 'FIX'

is executed to set the row and column dimensions of the standard output VARIABLE equal to unity. Otherwise a BRM 'FIX' is executed to set the row and column dimensions of the standard output VARIABLE to the same as those of the ARGUMENT associated with the COMMAND. Execution converges again to a common path and the secondary transfer is made.

DIAgonal sum — The 'DIAgonal sum' COMMAND is used to find the trace of a square matrix. Its execution is a relatively simple example of the standardized structure of the floating-point coding. Execution proceeds from the point ⍋⊢(DIA). A recursive sum is made of all the diagonal elements of the VARIABLE indicated in the ARGUMENT, the result being accumulated in W,1 [13]. When the sum is complete, {S,1} are set equal to {W,1}. Hence on completion of the COMMAND, the standard output VARIABLE is a scalar, with a value equal to the trace of the VARIABLE operated on.

DETerminant (Flow Diagram 27.) — The 'DETerminant' COMMAND is used to find the determinant of a square matrix. The method of pivotal condensation is used. For an N by N matrix this method consists of N-1 iterations. In each iteration another column of the matrix is operated on so as to bring its sub-diagonal elements to zero.

---

[13] The W-TABLE is a temporary storage TABLE for TYPE III WORDS; its length is 1⍬1 WORDS. It is capable of storing a MAP matrix of any permissible size. The zeroth WORD is not normally used.

In the 'DETerminant' component of the EXECUTIVE SUBPROCESSOR these iterations are carried out by a loop: each iteration of the loop is in two phases. In the first phase the current diagonal (pivotal) element is tested, and made non-zero if the operation is necessary. In the second phase the matrix is operated on to bring the sub-diagonal elements of a new column to zero. On the completion of the iterations, the determinant of the resulting matrix is the product of the diagonal elements.

Execution of the COMMAND is as follows. On entry at point ψ—(DET) a temporary result store Ω is set equal to 1.Ø. A BRM 'COPY' is executed to transfer the elements of the ARGUMENT to the standard output VARIABLE. If the ARGUMENT is scalar, the condensation process is not required and execution proceeds to point ψ—(D4). Otherwise a 'NO' branch is taken. {K+1} are set equal to the dimension of the ARGUMENT *minus one*, and the contents of a loop counter KK initialized to zero. Execution now reaches point ψ—[24] and the main iterative loop is entered.

The first phase is now carried out. {KK} are incremented by one, and the contents of a counter Ω+3 set equal to one. {Ω+3} count the number of times attempts are made to make non-zero a zero pivotal element. Execution now reaches point ψ—[25] at the start of a loop which is iterated once every time an attempt is made to produce a non-zero pivot.

In this loop, first the absolute value of the pivotal element OO(KK, KK)[14] is calculated. If its value is less than about 10^{-7} it is taken as zero, and steps are taken to make it non-zero. Otherwise the first phase in the main loop is finished.

The following action is taken to try to produce a non-zero pivot. {Q+5} are set equal to {Q+3} + {KK} and {Q+3} are incremented by one. A test is now made: if {Q+3} = {K+1} then it is impossible to produce a non-zero pivot and the VARIABLE constitutes a singular matrix. A 'YES' branch is taken to point ψ—(D2). If the equality does not hold, the column {Q+5} of OO is added into the column {KK} at point ψ—(D1). Execution then returns to point ψ—[25] for a new check on the pivot now contained in column {KK}.

Suppose now that the pivotal element OO(KK, KK) has been made non-zero. The second phase of the main loop is now carried out. First {KK+1} are set equal to {KK}. Execution now reaches point ψ—[26] at the start of another inner loop. For each iteration round this loop a row of the matrix is operated upon so as to bring the element in the row in the same column as the pivotal element equal to zero. The following action is taken. {KK+1} are incremented by one. The ratio

---

[14] This new notation is also used in later parts of this section. By OO(*,**) is meant the element of the standard output VARIABLE 'OO' whose row and column indices are {*} and {**} respectively. To obtain OO(*,**) a BRM 'CALC' is first employed to obtain the INDEX of the appropriate WORD in the S-TABLE.

OO(KK+1, KK)/OO(KK, KK) is calculated and stored. Then row {KK} of the matrix is multiplied by the stored ratio, negated and added into the row {KK+1} of the matrix. If further rows of the matrix remain to be treated, execution returns to point ψ—[26]. If not the second phase is complete.

Another test is now made. If {KK} ≠ {K+1} then further iterations of the main loop are required to complete the condensation process. Execution therefore returns to point ψ—[24]. If not, a 'YES' branch is taken; the diagonal elements of the matrix are summed and the result placed in Q. The final result in Q is then transferred to the first WORD in the S-TABLE. Execution now reaches point ψ—(D4), at which point the standard output VARIABLE contains the determinant of the matrix VARIABLE indicated by the ARGUMENT.

If the value is very small or zero, execution proceeds via a 'YES' branch to point ψ—(D2). Otherwise a 'NO' branch is taken, and execution returns to point —[4] in the DIRECTIVE SUBPROCESSOR.

Execution reaches point ψ—(D2) if the matrix is singular. The standard output VARIABLE is set equal to ∅.∅, and message MS4 output to the teletype:

DETERMINANT ZERO

Execution then returns to point —[4] in the DIRECTIVE SUBPROCESSOR.

EIGenvalue (Flow Diagram 28.)

The 'EIGenvalue' COMMAND is used to find the eigenvalues and eigenvectors of a square matrix. The results are output to the teletype, and not stored. The Leverrier-Fadeev method is used to derive the characteristic polynomial of the matrix. This is then solved by Bernoulli's method, to obtain the eigenvalues. The eigenvectors are then obtained using the eigenvalues, and intermediate results from the application of the Leverrier-Fadeev method.

A complete exposition of the method may be found in [7]. For convenience the algorithms are summarized below. Let $\lambda_1, \lambda_2, \ldots, \lambda_n$ be the eigenvalues of the n by n matrix A. Then the characteristic polynomial is

$$\lambda^n - q_1\lambda^{n-1} - q_2\lambda^{n-2} \ldots\ldots q_{n-1}\lambda - q_n = \emptyset \qquad (1)$$

The following recursion gives the coefficients q_i:

$$\left.\begin{array}{lll} A_1 = A & , \text{trace}(A_1) = q_1 & , B_1 = A_1 - q_1 I, \\ A_2 = AB_1 & , \text{trace}(A_2) = 2q_2 & , B_2 = A_2 - q_2 I, \\ \vdots & \vdots & \\ A_{n-1} = AB_{n-2} & , \text{trace}(A_{n-1}) = (n-1)q_{n-1} & , B_{n-1} = A_{n-1} - q_{n-1} I \\ A_n = AB_{n-1} & , \text{trace}(A_n) = nq_n & , B_n = A_n - q_n I \end{array}\right\} \qquad (2)$$

At least one of the columns of the matrix Q_i, where

$$Q_i = \lambda_i^{n-1} I + \lambda_i^{n-2} B_1 + \ldots\ldots + \lambda_i B_{n-2} + B_{n-1} \qquad (3)$$

is the eigenvector corresponding to λ_i, while the others are zero. If the columns are summed, then it is certain that the result is the eigenvector.

The eigenvalues are obtained by finding in turn all the roots of the characteristic polynomial. The roots are obtained in turn, starting with the dominant one. When this is obtained the order of the polynomial is reduced and the process repeated as many times as required. At each stage the following recursion is formed:

$$\left.\begin{aligned} & S_1 - q_1 = 0 \\ & S_2 - S_1 q_1 - 2q_2 = 0 \\ & \cdots\cdots\cdots\cdots\cdots \\ & S_n - S_{n-1} q_1 - \cdots\cdots - S_1 q_{n-1} - n q_n = 0 \end{aligned}\right\} \qquad (4)$$

for $i > n$

$$S_i - S_{i-1} q_1 - \cdots\cdots - S_{i+1-n} q_{n-1} - S_{i-n} q_n = 0 \qquad (5)$$

If the dominant root is a real root λ_1 then

$$\lim_{p \to \infty} \frac{S_p}{S_{p-1}} = \lambda_1 \qquad (6)$$

If the dominant root is a complex pair $\lambda_1 \{\cos \mu_1 \pm i \sin \mu_1\}$ then

$$\lim_{p \to \infty} \frac{S_p S_{p-2} - S_{p-1}^2}{S_{p-1} S_{p-3} - S_{p-2}^2} = \lambda_1^2; \quad \lim_{p \to \infty} \frac{S_p S_{p-3} - S_{p-1} S_{p-2}}{S_{p-1} S_{p-3} - S_{p-2}^2} = 2\lambda_1 \cos \mu_1 \qquad (7)$$

If the dominant root is a real pair of equal magnitude but opposite sign λ_1 , $-\lambda_1$ then

$$\lim_{p \to \infty} \frac{S_p}{S_{p-2}} = -\lambda_1^2 \tag{8}$$

The method will not work if two distinct complex pairs exist with equal absolute value.

The operation of the 'EIGenvalue' COMMAND is now described. It is divided into three phases. In the first phase, the coefficients of the characteristic polynomial are calculated. The second and third phases are contained in a loop iterated once for every real root or complex pair the second phase obtains from the polynomial. The third phase calculates the eigenvector or complex pair corresponding to the root or pair found by the second phase. The third phase also handles output of the results to the teletype, and reduction of the order of the polynomial.

Execution of the first phase is as follows. On entry at point ψ-(EIG) a BRM 'COPY' is executed to transfer the contents of the VARIABLE indicated by the ARGUMENT to the W-TABLE. The TR-TABLE is used to store coefficients of the characteristic polynomial.[15] {TR,Ø} are set equal to 1.Ø. Next the matrix in the W-TABLE is searched to find the

---

[15] The TR-TABLE is a TYPE III TABLE 11 WORDS long used for the temporary storage of floating-point numbers.

two elements having the greatest and least absolute values. {Q} are set equal to the mean of the two absolute values. All the elements of the matrix in the W-TABLE are then normalized by dividing by {Q}.

Actual calculation of the coefficients of the polynomial now takes place. The counter {Q+3} is initialized to zero. Execution now reaches point ψ—[28], the start of a loop which is iterated once for every coefficient calculated. Inside the loop, execution proceeds as follows. First {Q+3} are incremented by one. {TR,{Q+3}} are set equal to ∅.∅. {TR,{Q+3}} are then set equal to the negative of the accumulated sum of the diagonal elements of the matrix in the W-TABLE divided by FLOAT{Q+3}.[16] At the same time the column {Q+3} of the V-TABLE[17] is set to zero. Execution arrives at point ψ—(EI2), at which stage {TR,{Q+3}} are added to the diagonal elements of the matrix in the W-TABLE. All the columns of this matrix are then added into column {Q+3} of the V-TABLE, and at the same time the matrix is premultiplied by the normalized matrix VARIABLE indicated by the ARGUMENT, setting the result back in the W-TABLE.

It can be seen that this process corresponds to one iteration of the recursion shown in Eqn. (2) above. If {Q+3} = {K+1} where {K+1} constitute the dimension of the VARIABLE then all the coefficients have

[16] FLOAT{Q+3} means {Q+3} converted to a floating-point number.

[17] The V-TABLE is a temporary storage TABLE for TYPE III WORDS, 1∅1 WORDS in length. It is identical in form and purpose to the W-TABLE.

been found. Otherwise the loop is iterated once more by returning to point ψ—[28]. On completion, execution arrives at point ψ—(EI4), with {K+1} = {K+2} equal to the dimension of the VARIABLE.

Point ψ—(EI4) is the beginning of the loop containing the second and third phases of execution, and is also the beginning of the second phase. Essentially this phase calculates the dominant root or complex pair of the characteristic polynomial, by use of Eqns. (4) and then by iteration of Eqn. (5) until one of the limits given in Eqns. (6) through (8) has been reached to within some specified accuracy.

First the iteration counter {Q+3} is initialized to zero. {KK}, which indicate the number of times the accuracy criterion has been tentatively met, are set to zero. The W-TABLE is from now on used to store intermediate results. Eventually {W, 97} and {W, 98} will contain the real and imaginary parts of the eigenvalues of the normalized matrix. At this point {W, 98} and {W, 1ØØ} are set equal to Ø.Ø. The inner loop iterating Eqns. (4) and (5) is now entered at point ψ—[29]. {Q+3} are incremented by one, and {KK+1}, denoting either a real or complex root, set to unity.

Either of two courses of action now ensue: if {Q+3} > 1Ø, then {Q+4} are set equal to 1Ø and {W, 1}.....{W, 1Ø} each moved backwards one WORD to W, Ø.... W, 9. If {Q+3} ≤ 1Ø, then {Q+4} are set equal to {Q+3}. In either case {W, Ø} are set to FLOAT{Q+3} and {W, {Q+4}} to Ø.Ø.

Next {Q+6} are set to either {Q+3} or {K+2}, whichever has the smaller value. Execution now reaches point ⏀EI5, where {W,{Q+4}} are set equal to the sum from i = 1 to {Q+6} of {TR,i} multiplied by {W,{Q+4}-i}. At this stage, if more than 1Ø iterations of the loop have been carried out, {W,1Ø}...{W,1} are the ten most recently calculated values S_i in Eqn. (5).

A further test is made. If the value of {Q+3} indicates that fewer than 16 iterations of Eqns. (4) and (5) have been carried out, no testing for convergence is made, and execution proceeds to point ⏀EI6. If more than 15 iterations have been carried out, testing is now started. First a test is made for convergence to the limits shown in Eqns. (7) and (8).

The processor sets

$$\{W,11\} = \{W,9\}\{W,7\} - \{W,8\}^2$$

and calculates $\{W,1Ø\}\{W,8\} - \{W,9\}^2$. This latter, if non-zero divided by {W,11}, is stored in W,12. If the absolute value $[\{W,12\} - \{W,1ØØ\}]/\{W,12\}$ is less than about $1Ø^{-7}$ the convergence criterion is met this iteration, and {KK} are incremented by one. In any event {W,1ØØ} are set equal to {W,12} and a test for convergence to the limit shown in Eqn. (6) follows.

The test is made on {W,11}, divided by $\{W,8\}^2$ if the former is non-zero. If the absolute value

is less than $1\emptyset^{-7}$, then the convergence criterion is met this iteration, and {KK} are decremented by one. In this case {W,97} are set equal to {W,1Ø}, divided by {W,9} if the former is non-zero. If the absolute value is still $> 1\emptyset^{-7}$ then {W,97} are set equal to the absolute value instead. This ends convergence testing for the current iteration.

If convergence is to a real root, then by this process, {KK} increase negatively, as the appropriate criterion is satisfied on successive iterations. If convergence is to a real or complex pair, then {KK} increases positively. Four such iterations are considered sufficient for termination of the iterative process. Accordingly, execution is as follows. If the absolute value of {KK} > 4 then a 'YES' branch is taken to point ψ—[3Ø]. Otherwise a 'NO' branch is taken to point ψ—(EI6). Here, if {Ω+3} ≠ 15ØØ execution returns to point ψ—[29] for a new iteration. If not, no root could be found by the above process. Message MS8 :

COMPUTATION FAILURE

is therefore output to the teletype and a branch made to point —[27] in the 'INVert' component.

In the second phase it now remains to calculate the eigenvalues from the results of the iterative process: execution is at point ψ—[3Ø]. If {KK} ≠ Ø then execution proceeds to point ψ—(EI7). Here if {W,12} are negative a 'NO' branch is taken. There is a real pair of roots. {W,97} are set equal to the square root (as found by a BRM SQRT)

of the negative of {W,12}. Now, or if {KK} < ∅, {W,95} are set equal to {W,97} multiplied by {Q}, and execution branches to point ⏀(EI8). {W,97} is the real eigenvalue of the original matrix VARIABLE.

If in the above process {W,12} had been positive a 'YES' branch would then have been taken. If this happens {W,99} are set equal to [{W,1∅}{W,7} - {W,9}{W,8}]/{W,11}, {W,97} to {W,99}/2.∅, and {W,98} to the square root of {W,12} - {W,97}^{2}. Then {W,95} and {W,96} are set respectively to {W,97} and {W,98} multiplied by {Q}. {KK+1} are set equal to -1 and a branch to point ⏀(EI8) made. {W,97} and {W,98} are the real and imaginary parts of the complex eigenvalue of the original matrix VARIABLE.

The third and last phase of execution is now entered. An inner loop, performed once for a real root, and twice for a complex pair encloses the whole process. {Q+3} a counter indexing the loop are set to unity. On entry into the loop, first n - 1 powers of the eigenvalue are calculated as follows. {W,21} and {W,31} are set to 1.∅ and ∅.∅ respectively. Then for values of i from 2 to {K+1}, {W,2∅+i} are set equal to

$$\{W,97\}\{W,19+i\} - \{W,98\}\{W,29+i\}$$

and {W,3∅+i} are set equal to

$$\{W,97\}\{W,29+i\} + \{W,98\}\{W,19+i\} .$$

Execution now reaches point ⏀-(EI9), at which point {W,21}.....{W,2Ø+{K+1}} and {W,31}.....{W,3Ø+{K+1}} are the real and imaginary parts of the scalar multipliers of the B matrices in Eqn. (3).

The sum of the columns of Ω in Eqn. (3) is now obtained: each column of the matrix in the V-TABLE is multiplied in turn by the real and imaginary parts of the appropriate multipliers previously calculated, and sums of all the real and imaginary columns obtained. The result is the eigenvector corresponding to the eigenvalue already obtained. The real and imaginary parts are placed respectively in {W,41}.....{W,4Ø+{K+1}} and {W,51}.....{W,5Ø+{K+1}}. {K+2} are then decremented by one to show that another root of the polynomial is found and the order has decreased by one.

The eigenvector found is next normalized so that the largest element has a unit modulus, unless the eigenvector is identically zero. It now remains to output the eigenvalue and vector calculated.

First either message MS2:

COMPLEX EIGENVALUE

is output to the teletype if the eigenvalue is complex, or the message MS3:

REAL EIGENVALUE

if the eigenvalue is real. Next the real part of the eigenvalue {W, 95} is output to the teletype. If the imaginary part {W, 96} exists, this is output also. These are followed by a carriage return and two line feeds. The eigenvector is now output. For i = 1 to {K+1}, the following are output: {W, 4Ø+i}; {W, 5Ø+i} if {KK+1} < Ø, and lastly a carriage return and line feed. If {KK+1} < Ø, {W, 96} and {W, 98} are negated.

This completes output of the results. If {Q+3} = {KK+1} it means that the inner loop round the third phase has been iterated the correct number of times (once for a real eigenvalue and twice for a complex pair). If not {Q+3} are decremented by 2 and the loop iterated once more.

Once the processor makes an exit from the inner loop, in this phase of execution it only remains to reduce the order of the polynomial in order to find further eigenvalues, if any. A test is made on {K+2}: if the contents have been reduced to zero, all eigenvalues have been found. A 'YES' branch is taken. A carriage return and line feed are output to the teletype, and execution branches to point —[27] in the 'INVert' component. If {K+2} are non-zero then a 'NO' branch is taken, and the order of the polynomial is reduced as follows.

If {KK+1} are not negative, showing that the eigenvalue just found was real, the order of the

characteristic polynomial is reduced by one, by division by a linear factor. The division is accomplished by recalculating coefficients in this way: for i = 1 to {K+2}, the product {TR,i-1}{W,97} is added into {TR,i}. Execution now returns to point ψ-(EI4) to find the new dominant root of the reduced characteristic polynomial.

If {KK+1} are negative, then the order of the polynomial must be reduced by two by division by a quadratic factor, since a complex pair of eigenvalues were previously obtained. The coefficients of the polynomial are recalculated as follows. {W,99} are added into {TR,1}. If {K+2} are equal to unity then no further calculation is required. Otherwise, for i = 2 to {K+2}, {TR,i-1}{W,99} - {TR,i-2}{W,1ØØ} are added into {TR,i}. Execution now returns to point ψ-(EI4) as in the case of the real eigenvalue.

This completes discussion of the 'EIGenvalue' component of 'GROUPØ'.

INVert
(Flow Diagram 29.)

The 'INVert' COMMAND is used to invert a square non-singular matrix. A standard method using elemental transforms is used. The method has the advantage that it requires a minimum amount of storage space. For an n by n matrix, besides the storage for the matrix itself, only a further 2n integer locations are required.

Briefly for an n by n matrix the algorithm is as follows. Two column n-vectors are defined, to be used as pivot switches. These are initialized to zero. A main loop is executed n times. At each iteration a new pivot is found and its location recorded. The pivot is replaced by its reciprocal. In turn the remaining elements of the matrix are reduced, first the ones off the pivot row and column, and then the ones on the pivot row or column. After n iterations this pivotal reduction is complete. Depending on the final values of the pivot switches, rows and columns of the new matrix are interchanged. The final result is the inverse of the original matrix.

Execution of the 'INVert' COMMAND starts at point ϕ-(INV) . First a BRM 'COPY' is executed to transfer the matrix VARIABLE indicated by the ARGUMENT to the standard output VARIABLE . Next the row pivot switches, {W,1}.....{W,1ϕ} and the column pivot switches {W,11}.....{W,2ϕ} are initialized to zero.[18] A loop index counter {Q+2} is initialized to zero. Execution now arrives at point ϕ-[31] .

The main loop is entered at this point. {K+1} constitutes the dimension of the matrix: the loop is thus iterated {K+1} times. At the start of each iteration {Q+2} are incremented by one and {Q} set equal to ϕ.ϕ . Now all the elements of matrix in the standard output VARIABLE are

[18] Here the W-TABLE is being used to store integer numbers.

searched to find one of largest absolute value, omitting from the search the i^{th} row if $\{W,i\} \neq \emptyset$ or the i^{th} column if $\{W,10+i\} \neq \emptyset$. $\{KK\}$ are set equal to the row index of the element found, and $\{KK+1\}$ to the column index. The value of the element found, the pivotal element, is stored in Q.

If $\{Q\}$ are less than about 10^{-7}, then it is assumed that the matrix is singular and execution proceeds via a 'YES' branch to point ϕ—[32]. Otherwise a 'NO' branch is executed and the pivot reduction process continued.

The elements of the matrix are now reduced. First the pivotal element OO(KK, KK+1) is itself reduced by replacing it with its reciprocal. The new value is also stored in Q+7. Then two pivot switches are set: $\{W,\{KK\}\}$ are set to $\{KK+1\}$, and $\{W,\{KK+1\}\}$ to $\{KK\}$.

Next the elements off the pivot row or column are reduced. From each element OO(i, j) in turn, where $i \neq \{KK\}$ and $j \neq \{KK+1\}$, is subtracted the product

$$\{OO(KK,j)\}\{OO(i,KK+1)\}\{Q+7\}.$$

Lastly the elements in the pivot row and column are reduced. Each element excluding the pivotal element, in row $\{KK\}$ of the matrix is multiplied by $\{Q+7\}$. Each element excluding the pivotal element, in column $\{KK+1\}$ of the matrix is multiplied by $-\{Q+7\}$.

A test is now made: if {Q+2} ≠ {K+1}, then more iterations of the loop described are required and execution proceeds via a 'NO' branch to point ψ—[31]. Otherwise a 'YES' branch is taken. Conditional row and column changes of the matrix in the standard output VARIABLE are now made.

Conditional row changes are made first. {Q+2}, a loop counter, are initialized to zero. A loop checking the row pivot switches is now entered. At each iteration the following process is carried out. {Q+2} are incremented by one. A check is then made on {W,{Q+2}}. If the value is not equal to {Q+2} a 'NO' branch is taken. Row {Q+2} of OO is interchanged with row {W,{Q+2}} of OO, and switch {W,{Q+2}} with switch {W,{W,{Q+2}}}. The test on {W,{Q+2}} is then repeated, and so on. When {W,{Q+2}} becomes equal to {Q+2} a 'YES' branch is taken. If {Q+2} ≠ {K+1}, more row and column interchanges may be required, and a return is made back to the beginning of the loop. If {Q+2} = {K+1}, all the necessary row interchanges have been made.

The column changes follow exactly the same pattern. On the conclusion of this process, the standard output VARIABLE contains the inverse of the original matrix. Execution returns to point —[4] of the DIRECTIVE SUBPROCESSOR.

If the matrix is singular, execution arrives at point —[32]. Message MS7 is output to the teletype:

MATRIX SINGULAR - RANK =

{Q+2} minus one are calculated and output, followed by a carriage return and 2 line feeds. {Q+7} are set equal to the number of elements in the matrix, and execution arrives at point —[27]. All the elements of the matrix in the standard output VARIABLE are set to zero, and then execution returns to point —[4] in the DIRECTIVE SUBPROCESSOR.

This concludes the description of the operation of the four COMMANDS in the 'GROUPØ' component group of the EXECUTIVE SUBPROCESSOR. The component group uses three routines that have not already been described, all BRM's. The BRM SQRT will not be described here, since the same routine is used in the SDS 940 FORTRAN LIBRARY. Slight differences in the two versions can be accounted for by inspection. The remaining two BRM's are described below.

BRM 'FIX' (Flow Diagram 3Ø.)

The BRM 'FIX' is a routine for adjusting the dimensions of the standard output VARIABLE in the IDIM-TABLE. The call sequence is:

```
BRM  FIX
[return location]
ZRO  [address of location containing row dimension]
ZRO  [address of location containing column dimension]
 :
```

Both at entry and exit all registers contain garbage.

Operation of the BRM is as follows. First the row dimension is obtained from its addressed location and placed in O_5O_6{IDIM, -6Ø}. The column dimension

is then obtained from its addressed location and placed in $O_7O_8\{IDIM, -6\emptyset\}$. The exit from the BRM is then taken.

BRM 'COPY' (Flow Diagram 31.)

The BRM 'COPY' is a routine for moving the contents of BLOCKS of storage in and out of the S-TABLE, or for moving them to a new BLOCK of storage of the same size. Its call sequence is:

```
BRM  COPY
[return location]
ZRO  [address of location containing row dimension]
ZRO  [address of location containing column dimension]
ZRO  [address of location containing old BLOCK INDEX]
ZRO  [address of location containing new BLOCK INDEX]
 :
```

Both at entry and at exit all registers contain garbage.

Operation of the BRM is as follows. First the row and column dimensions are obtained, and the number of elements in the matrix calculated. Then elements of the matrix are taken from the old BLOCK in turn and inserted in the new BLOCK in the S-TABLE.

If both BLOCKS are in the S-TABLE this process is straightforward. If, however, the BRM is used to copy the values of a matrix from outside the S-TABLE into the S-TABLE, say from the W-TABLE into the S-TABLE, the old BLOCK INDEX must be calculated in some way. For the purposes of this calculation the S-TABLE is imagined to be extended in the core so that it overlaps the core area designated to the W-TABLE. The BLOCK INDEX is then the INDEX in the expanded

S-TABLE of the zeroth WORD of the W-TABLE. Thus it is of crucial importance in any rearrangement of the MAP processor not to change the relative positions of the temporary storage TABLES and the S-TABLE in the core by changing the ARPAS coding involving the definition of the TABLES.

On the completion of the transfer of the elements of the matrix, the exit from the BRM is taken.

NEGate

NUL1 (Flow Diagram 32.)

The 'GROUP1' component group of the EXECUTIVE SUBPROCESSOR comprises two single-ARGUMENT COMMANDS: 'NEGate', where the result is placed in the standard output VARIABLE; and 'NUL1'. On entry a preliminary section of coding common to either of the COMMANDS is executed. A secondary transfer is then made to further sections of coding dealing separately with each individual COMMAND.

Operation is as follows. On entry at point ⏀-(GRP1) the secondary transfer index number is unloaded and stored. A BRM 'LAB' is executed to check the type of the ARGUMENT associated with the COMMAND. If the ARGUMENT is LABEL-type, the 'BAD' exit is taken to point —[7] in the 'EDIT' component group. If the ARGUMENT is VARIABLE-type, the 'GOOD' exit is taken from the BRM.

Next a POP 'UNLO' is executed to unload the dimensions of the VARIABLE and its BLOCK INDEX in

the S-TABLE from the IDIM-TABLE. The total number of elements in the VARIABLE is calculated and stored. The secondary transfer is then made.

NEGate — The 'NEGate' COMMAND is used to form the negative of a matrix. The result is placed in the standard output VARIABLE. Execution starts at point ⏀—(NEG). A BRM 'FIX' is executed to adjust the dimensions of the standard output VARIABLE in the IDIM-TABLE to the same as those of the ARGUMENT.

Each element of the standard output VARIABLE is in turn set equal to the negative of the corresponding element of the ARGUMENT VARIABLE. A branch to point —[4] in the DIRECTIVE SUBPROCESSOR is then made.

NULl — The 'NULl' COMMAND is used to set the value of any matrix VARIABLE identically to zero. Execution starts at point ⏀—(NUL). Each element of the matrix specified in the ARGUMENT is set equal to $\emptyset.\emptyset$ in turn. A branch back to point —[4] in the DIRECTIVE SUBPROCESSOR is then made.

ADD
SUBtract
SKIp
EQUate (Flow Diagram 33.)

The 'GROUP2' component group of the EXECUTIVE SUBPROCESSOR comprises four double-ARGUMENT COMMANDS. The first two, 'ADD' and 'SUBtract'

are operations where the result is placed in the standard output VARIABLE. The third COMMAND, 'SKIp' never uses the standard output VARIABLE and is a flow-changing operation. In the last COMMAND, 'EQUate', one of the ARGUMENTS is an output VARIABLE. The standard output VARIABLE is only used if it appears as an ARGUMENT.

Operation of this component group is more unorthodox than the others because the secondary transfer occurs inside the loop iterating through the elements of the input matrices.

Operation is as follows. On entry at point ⏀—(GRP2) the secondary transfer index number is unloaded and stored. A BRM 'LAB' is executed to check the types of the ARGUMENTS associated with the COMMAND. If the ARGUMENTS are LABEL-type, the 'BAD' exit is taken to point —[7] in the 'EDIT' component group. If the ARGUMENT is VARIABLE-type, the 'GOOD' exit is taken from the BRM.

Next two POP's 'UNLO' in succession are executed. In turn for the first and second ARGUMENTS respectively, these unload the dimensions of the VARIABLE ARGUMENT and the BLOCK INDEX in the S-TABLE from the IDIM-TABLE. Several tests are now made to ensure compatibility of the ARGUMENTS.

If the COMMAND is not 'EQUate' a 'NO' branch is taken to point ⏀—[18]. If it is 'EQUate' but the first ARGUMENT is not the standard output

VARIABLE, a 'NO' branch is again taken to point ψ—[18]. If the COMMAND is 'EQUate' and the first ARGUMENT is the standard output VARIABLE a 'YES' branch is taken to point ψ—[19]. At point ψ—[18] if either the row dimensions or the column dimensions of each ARGUMENT are not the same, a 'NO' branch is taken to point —[9] in the 'APPend' component of the EXECUTIVE SUBPROCESSOR.

If the row and column dimensions both match each other, then a 'YES' branch is taken. If the COMMAND being executed is either 'ADD' or 'SUBtract' a further 'YES' branch is taken to point ψ—[19]. Otherwise a 'NO' branch is taken to point ψ—[2Ø].

At point ψ—[19] a BRM 'FIX' is executed to adjust the dimensions of the standard output VARIABLE in the IDIM-TABLE to those of the ARGUMENTS. Execution then proceeds to point ψ—[2Ø]. At point ψ—[2Ø] the total number of elements in each matrix VARIABLE is calculated, execution then arriving at point ψ—(GRP21).

At this point the matrix operation itself is executed. If the COMMAND is 'ADD', 'SUBtract', or 'EQUate', the indicated operation is carried out in turn on each element. If the COMMAND is 'SKIp', {INDEX} are incremented by one if the value of the first ARGUMENT is less than or equal to the value of the second. In this last COMMAND if the ARGUMENTS are not scalar a branch to point —[9] in the 'APPend' component is made.

Finally on the completion of this process, a branch is made back to point —[4] of the DIRECTIVE SUBPROCESSOR.

MULtiply
SCAlar multiply
TRAnspose (Flow Diagram 34.)

The 'GROUP3' component group of the EXECUTIVE SUBPROCESSOR comprises two double-ARGUMENT COMMANDS 'MULtiply' and 'SCAlar multiply' and one single-ARGUMENT COMMAND 'TRAnspose'. In all cases the result is placed in the standard output VARIABLE. On entry a preliminary section of coding common to each of the COMMANDS is executed. A secondary transfer is then made to further sections of coding dealing separately with each individual COMMAND.

Operation is as follows. On entry at point ϕ—(GRP3) the secondary transfer index number is unloaded and stored. A BRM 'LAB' is executed to check the type of the ARGUMENTS. If the ARGUMENTS are LABEL-type, the 'BAD' exit is taken to point —[7] in the 'EDIT' component group. If the ARGUMENTS are VARIABLE-type the 'GOOD' exit is taken from the BRM.

Next, two POP's 'UNLO' are executed in succession. In turn for the first and second ARGUMENTS respectively, these unload the dimensions of the VARIABLE and the BLOCK INDEX in the S-TABLE from the IDIM-TABLE. The secondary transfer is then made.

TRAnspose — The 'TRAnspose' COMMAND is used to find the transpose of a matrix. The result is placed in the standard output VARIABLE. Execution starts at point ⏀—(TRA). A BRM 'FIX' is executed to adjust the row dimension of the standard output VARIABLE to the same as the column dimension of the ARGUMENT and vice versa.

By means of a double loop the elements of the matrix ARGUMENT are copied into the W-TABLE, transposing in the process. A BRM 'COPY' is then executed to shift the contents of the W-TABLE into the standard output VARIABLE. A branch back to point —[4] of the DIRECTIVE SUBPROCESSOR is then made.

MULtiply — The 'MULtiply' COMMAND is used to find the matrix product of two matrices. The result is placed in the standard output VARIABLE. Execution starts at point ⏀—(MUL). First a check is made: if the row dimension of the first ARGUMENT is not equal in value to the column dimension of the second ARGUMENT, then a 'NO' branch is taken to point —[9] of the 'APPend' component. Otherwise execution continues.

Next a BRM 'FIX' is executed adjusting the column dimension of the standard output VARIABLE to the same value as the column dimension of the first ARGUMENT, and the row dimension of the standard output VARIABLE to the same value as the row dimension of the second ARGUMENT.

The matrix product is now calculated element by element in the usual way; the result being placed in the W-TABLE. Finally a BRM 'COPY' is used to transfer the result to the standard output VARIABLE, and a branch is made back to point —[4] in the DIRECTIVE SUBPROCESSOR.

SCAlar multiply — The 'SCAlar multiply' COMMAND is used to find the product of a scalar and a matrix. The result is placed in the standard output VARIABLE. Execution starts at point ϕ—(SCA). First the dimensions of the first ARGUMENT are checked. If it is found not to be a scalar, a 'NO' branch is taken to point —[9] in the 'APPend' component.

If it is a scalar, a 'YES' branch is taken and a BRM 'FIX' is executed to set the dimensions of the standard output VARIABLE the same as those of the second ARGUMENT. The value of the first ARGUMENT is then copied into temporary storage.

Each of the elements of the second ARGUMENT in turn are multiplied by the first ARGUMENT and the results stored in the corresponding elements of the standard output VARIABLE. Execution then returns to point —[4] of the DIRECTIVE SUBPROCESSOR.

REAd
PRInt
LOAd
STOre (Flow Diagram 35.)

The 'GROUP4' component group of the EXECUTIVE SUBPROCESSOR is the last group to be described.

It comprises four single-ARGUMENT COMMANDS: 'REAd', 'PRInt', 'LOAd', and 'STOre'. The first two are concerned with input and output of the values of MAP VARIABLES via the teletype, and the second two with input from and output to a disk file. On entry a preliminary section of coding common to each of the COMMANDS is executed. A secondary transfer is then made to further sections of coding dealing separately with each individual COMMAND.

Operation is as follows. On entry at the point ⏀—(GRP4) the secondary transfer index number is unloaded and stored. A BRM 'LAB' is executed to check the type of the ARGUMENT associated with the COMMAND. If the ARGUMENT is LABEL-type, the 'BAD' exit is taken to point —[7] in the 'EDIT' component group. If the ARGUMENT is VARIABLE-type, the 'GOOD' exit is taken from the BRM.

Next a POP 'UNLO' is executed to unload the dimensions and the BLOCK INDEX in the S-TABLE of the VARIABLE ARGUMENT in the IDIM-TABLE. The total number of elements in the VARIABLE is calculated and stored, and the secondary transfer made.

REAd

The 'REAd' COMMAND is used to input the values of a VARIABLE from the teletype. Execution starts at the point ⏀—(REA). Input of VARIABLE values proceeds row by row. Input of each row corresponds to one iteration round the loop to be described.

First the row index is set and a "bell" output to the teletype. Execution arrives at point ψ—[21], the start of an inner loop which is iterated once for every value input by the user. At the start of this loop the INDEX of the WORD in the S-TABLE where the number input is to be stored is obtained. A BRM 'IONUM' is then used to input the number from the teletype. If the number is unintelligible, the 'BAD' exit is taken from the BRM to point ψ—[22]. Otherwise the 'GOOD' exit is taken, and the number and its terminating character are stored.

Several tests are now made. If the end of the row has been reached the terminating character should be a carriage return. If it is not an error has arisen and execution proceeds to point ψ—[22]. If the terminating character is a carriage return, a 'YES' branch is taken. A line feed is output to the teletype, and a check made to see if all elements of the matrix have been read in. If they have execution returns to point —[4] in the DIRECTIVE SUBPROCESSOR. If they have not execution returns to point ψ—(REA) ready for the input of a new row.

If the end of the current row has not been reached, the terminating character should be either a "space" or a line feed. If it is neither an error has arisen, and execution proceeds to point ψ—[22]. If the terminating character is a "space" execution proceeds directly back to point ψ—[21] for the input of the next element in the row. If it is a line feed, a carriage return is output to the teletype before proceeding back to point ψ—[21].

If an error occurs in the input of a row of the matrix, execution reaches point ψ—[22] . The remainder of the row after the number containing the error is input character by character and discarded as unusable until a carriage return appears. A line feed is output to the teletype, and the loop indexing reset so that the next row input by the user is input as a repetition of the row in which the error occurred. Finally a BRM 'ER1' is executed to print out an ERROR MESSAGE, and execution returns to point ψ—(REA) .

PRInt

The 'PRInt' COMMAND is used to output to the teletype the values of a VARIABLE. Execution starts at the point ψ—(PRI). Output of the VARIABLES proceeds row by row. Output of each row corresponds to one iteration round the loop to be described.

First a row index is initialized, and execution arrives at point ψ—[23] , the start of an inner loop iterated once for every number output to the teletype. The pointers of a temporary storage string are reset. The INDEX in the S-TABLE of the WORD to be output is obtained, and the value taken from the TABLE. The latter is converted to character code form and stored in the temporary storage string. The exponent part is stored again separately.

The characters of the string are now output in turn to the teletype until the "E" denoting the start of the exponent part is reached. If the end of the current row has not yet been reached, a 'NO'

branch is taken, and execution returns to point ψ—[23] for output of the next number. If the end of the row has been reached a 'YES' branch is taken. At this point the fractional parts of all the elements of the row have been output, and all the exponent parts have been temporarily stored.

The exponent parts for the row are now output as follows. A carriage return and line feed are output to the teletype. For each element in the row in turn, the processor outputs to the teletype three "spaces", "E", and the appropriate exponent part. Two carriage returns and a line feed are output to the teletype. All the elements of the current row of the matrix VARIABLE have now been fully output.

If all the elements of the VARIABLE have been output, a 'YES' branch is now taken to point —[4] of the DIRECTIVE SUBPROCESSOR. If not a 'NO' branch is taken and execution returns to point ψ—(PRI) to output the next row.

STOre

The 'STOre' COMMAND is used to store the values of a VARIABLE on a disk file. Only one VARIABLE is stored on any one file. Execution starts at point ψ—(STO). On entry a BRM 'OUTF' is executed to make a disk file ready for output. If the user's attempt fails, the 'BAD' exit is taken. A BRM 'ER1' is executed to print out an ERROR MESSAGE and BRM 'OUTF' re-entered. If the file is successfully made ready, the 'GOOD' exit is taken from the BRM. The file identifying WORD 31463146_8, the number of

elements in the VARIABLE, and the number of columns of the VARIABLE are successively written on the file.

Each of the elements of the matrix VARIABLE are now written on the file in turn, row by row. The file is then closed and a branch made back to point —[4] of the DIRECTIVE SUBPROCESSOR.

LOAd

The 'LOAd' COMMAND is used to load the values of a VARIABLE from a disk file. The COMMAND is complementary to the 'STOre' COMMAND. Execution starts at point ψ—(LOA). On entry a BRM 'INF' is executed to make ready the disk file for input. If the user's attempt to specify a file fails, the 'BAD' exit is taken. A BRM 'ER1' is executed to output an ERROR MESSAGE to the teletype, and BRM 'INF' re-entered. If the file is successfully made ready, the 'GOOD' exit is taken from the BRM.

Next, three tests are made in succession. If the first file WORD is not the correct identifier; or if the second file WORD does not match the number of elements of the VARIABLE to be given values; or if the third file WORD does not match the number of columns of that VARIABLE then a 'NO' branch is taken. The file is closed, a BRM 'ER1' executed to output an ERROR MESSAGE, and execution returned to point ψ—(LOA) to specify a new file.

If the three first WORDS on the file are all correct, a 'YES' branch is taken. The values of the elements

are read in one by one from the file into the specified positions in the S-TABLE. The file is then closed, and execution returns to point —[4] of the DIRECTIVE SUBPROCESSOR.

Only one routine is used by the 'GROUP4' component group which has not already been described.

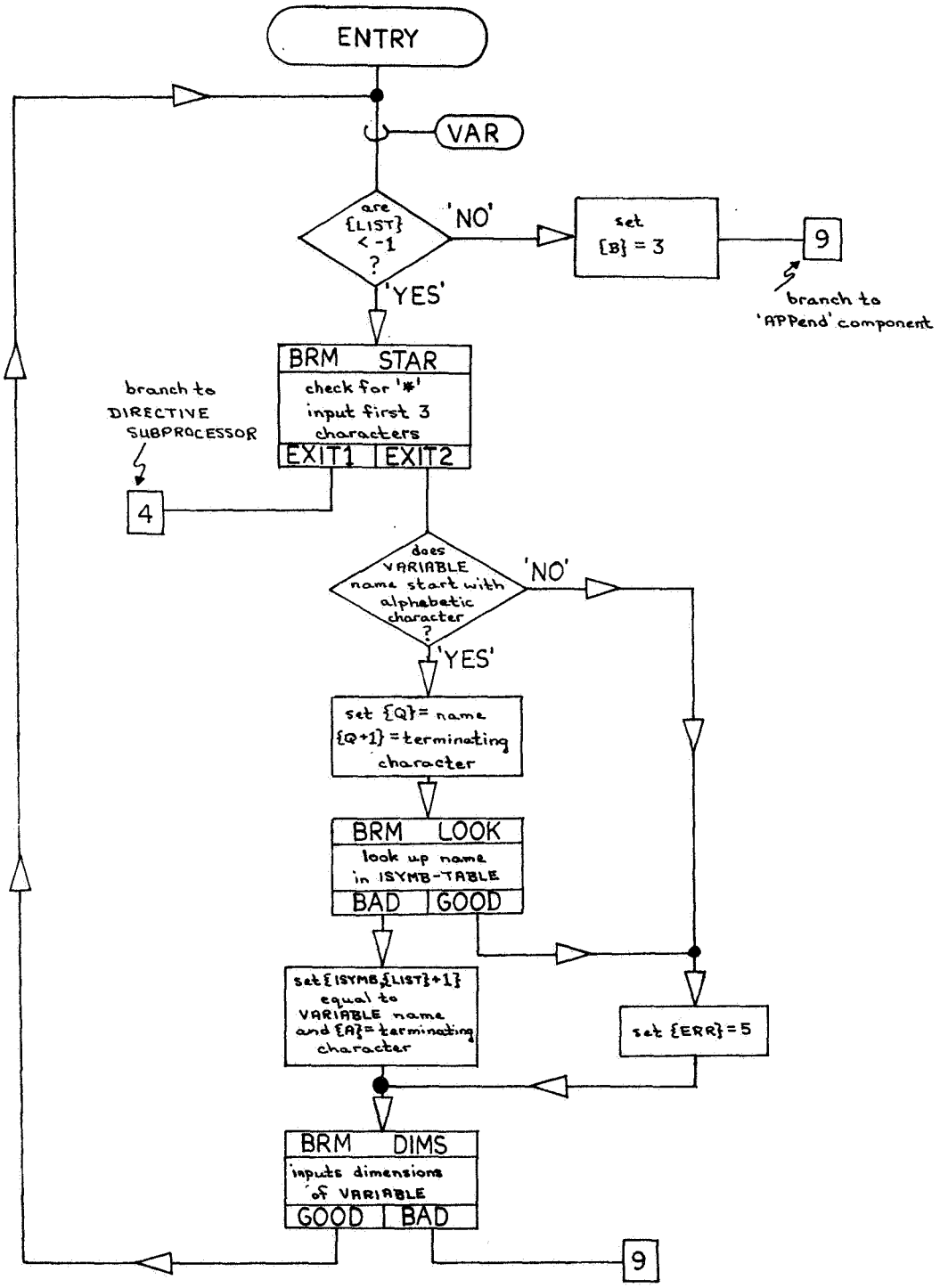
BRM 'IONUM'

The BRM 'IONUM' is used to input a floating-point number from the teletype to the A and B registers. It is identical in form and operation to the BRS 52 and so will not be described here. It was used solely because BRS 52 was not in working order when the MAP processor was developed. The call sequence is as follows:

```
BRM IONUM
[error return]
  :
```

On entry all registers contain garbage. At exit the A and B registers contain the number input, and the X register contains the character with which the user terminated input of the number.

This concludes the description of the components of the processor comprising the EXECUTIVE SUBPROCESSOR. It is also the end of the description of the entire processor. It should now be possible to identify in a symbolic listing of the processor, the purpose and operation of any particular section of coding.



Flow Diagram 9. Flowchart of VARiables operation for adding to the LIST of VARIABLES

CALL SEQUENCE:

```
BRM STAR
[EXIT1 return location]
[EXIT2 return location]
  :
```

AT ENTRY:

{A},{B},{X} garbage

AT EXIT1:

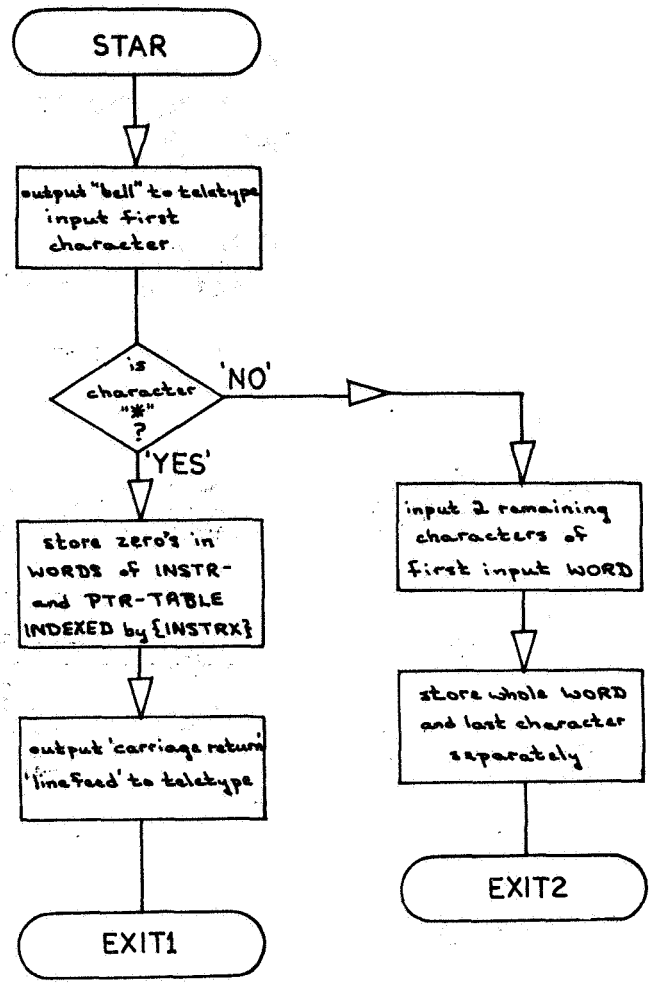
{A},{B},{X} garbage

AT EXIT2:

{B},{T} WORD of three characters

{X} ∅

{A} 3rd character



Flow Diagram 10. Flowchart of BRM STAR for checking first WORD of an EDIT mode expression

CALL SEQUENCE:

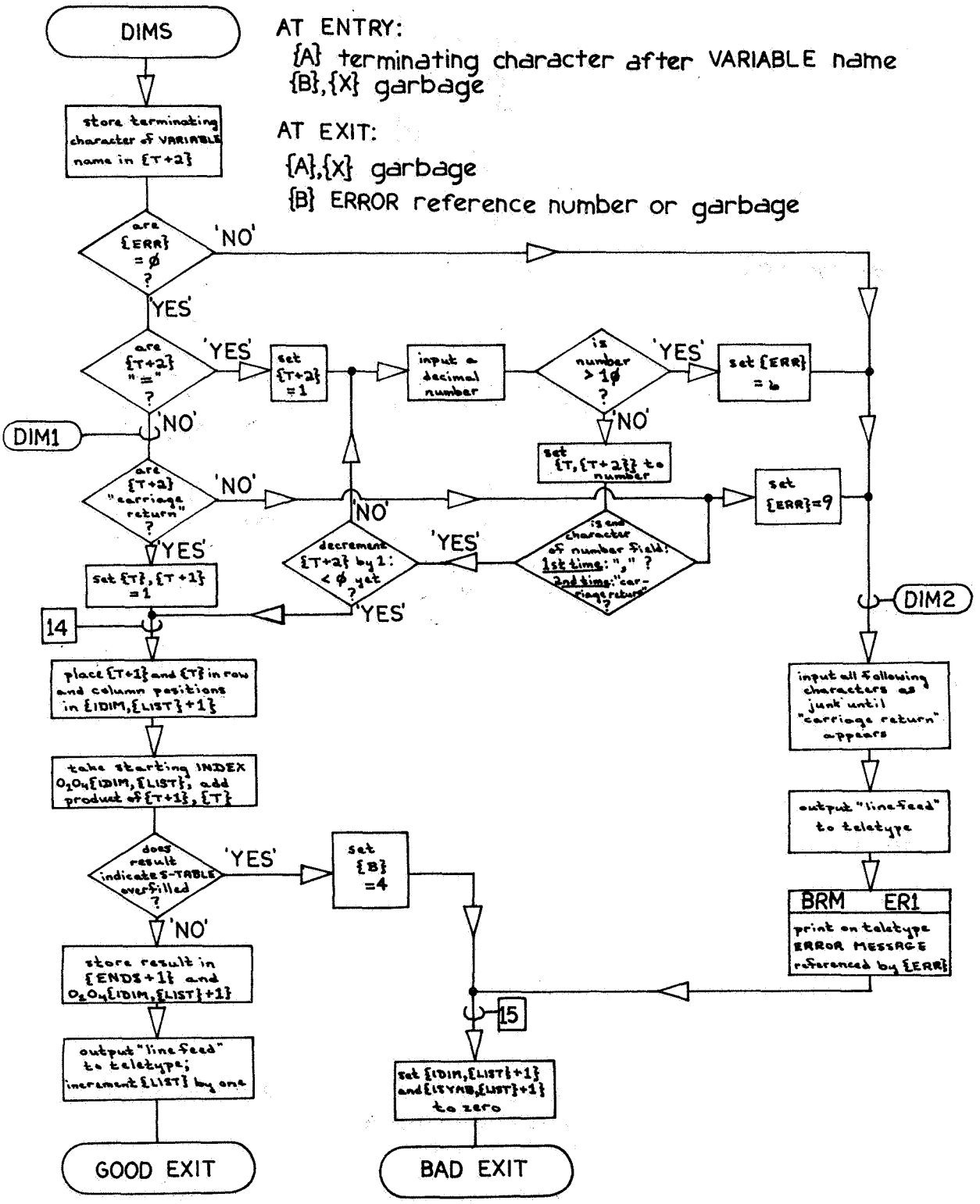
BRM DIMS
[BAD return location]
[GOOD return location]
⋮

AT ENTRY:

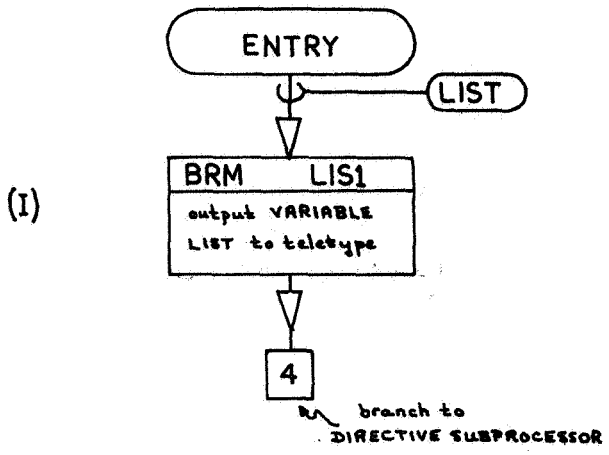
{A} terminating character after VARIABLE name
{B},{X} garbage

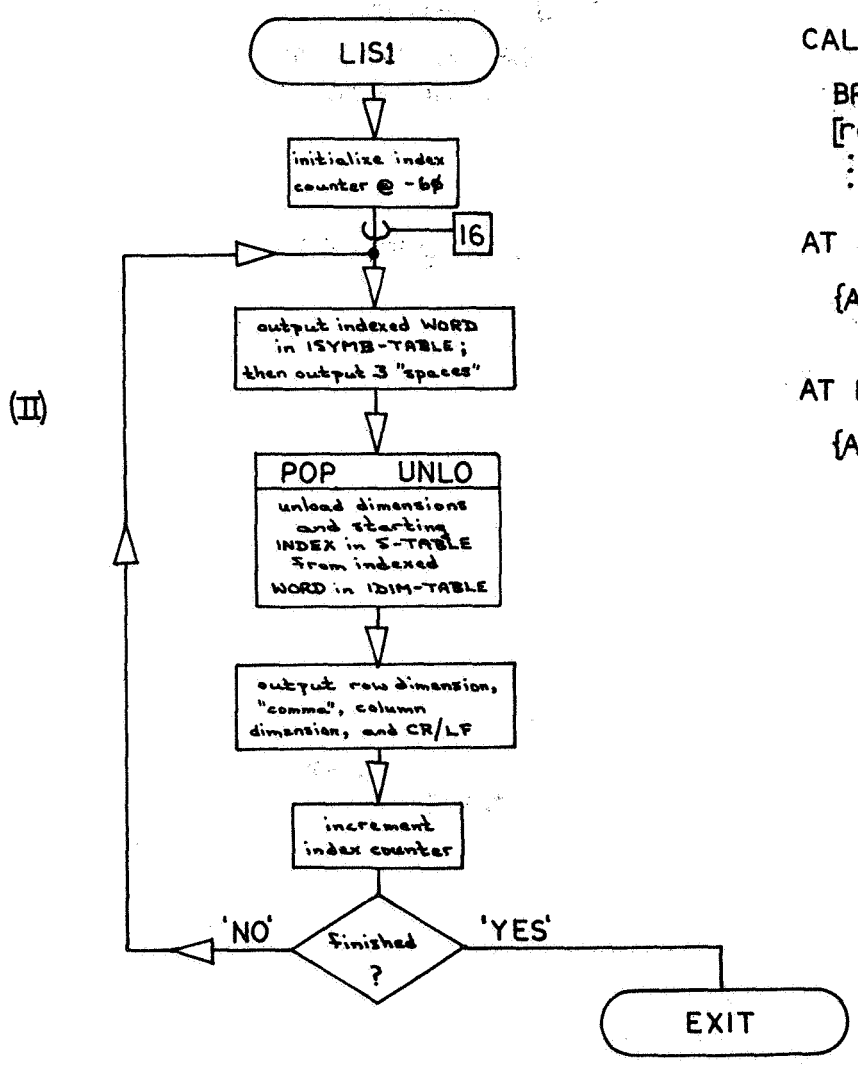
AT EXIT:

{A},{X} garbage
{B} ERROR reference number or garbage



Flow Diagram 11. Flowchart of BRM 'DIMS' for input and coding the dimensions of a VARIABLE





CALL SEQUENCE:

BRM LIS1
[return location]
⋮

AT ENTRY:

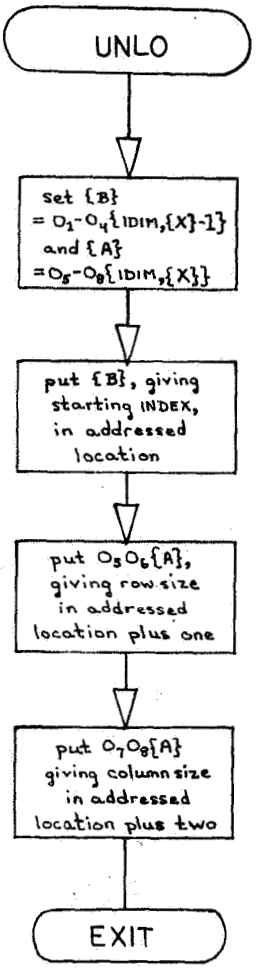
{A},{B},{X} garbage

AT EXIT:

{A},{B},{X} garbage

Flow Diagram 12. I Flowchart of calling sequence

II Flowchart of BRM LIS1 for printing a PROGRAM on the teletype



CALL SEQUENCE:

UNLO [addr. of storing location]

⋮

AT ENTRY:

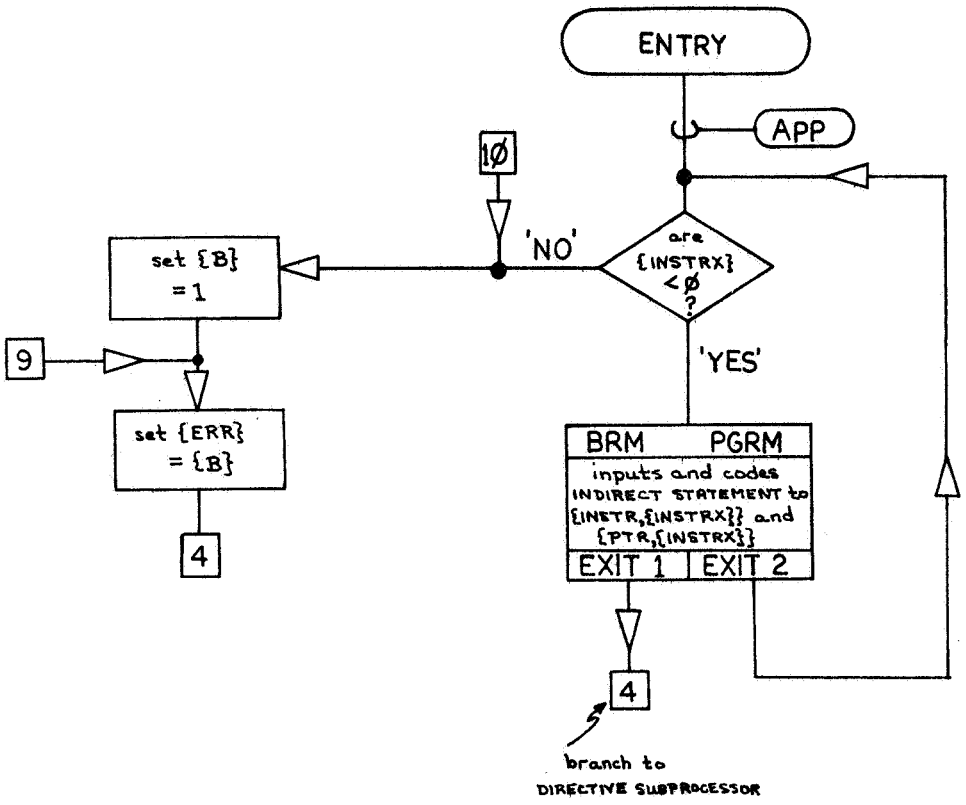
{A},{B} garbage

{X} INDEX of WORD of IDIM-TABLE to be unloaded

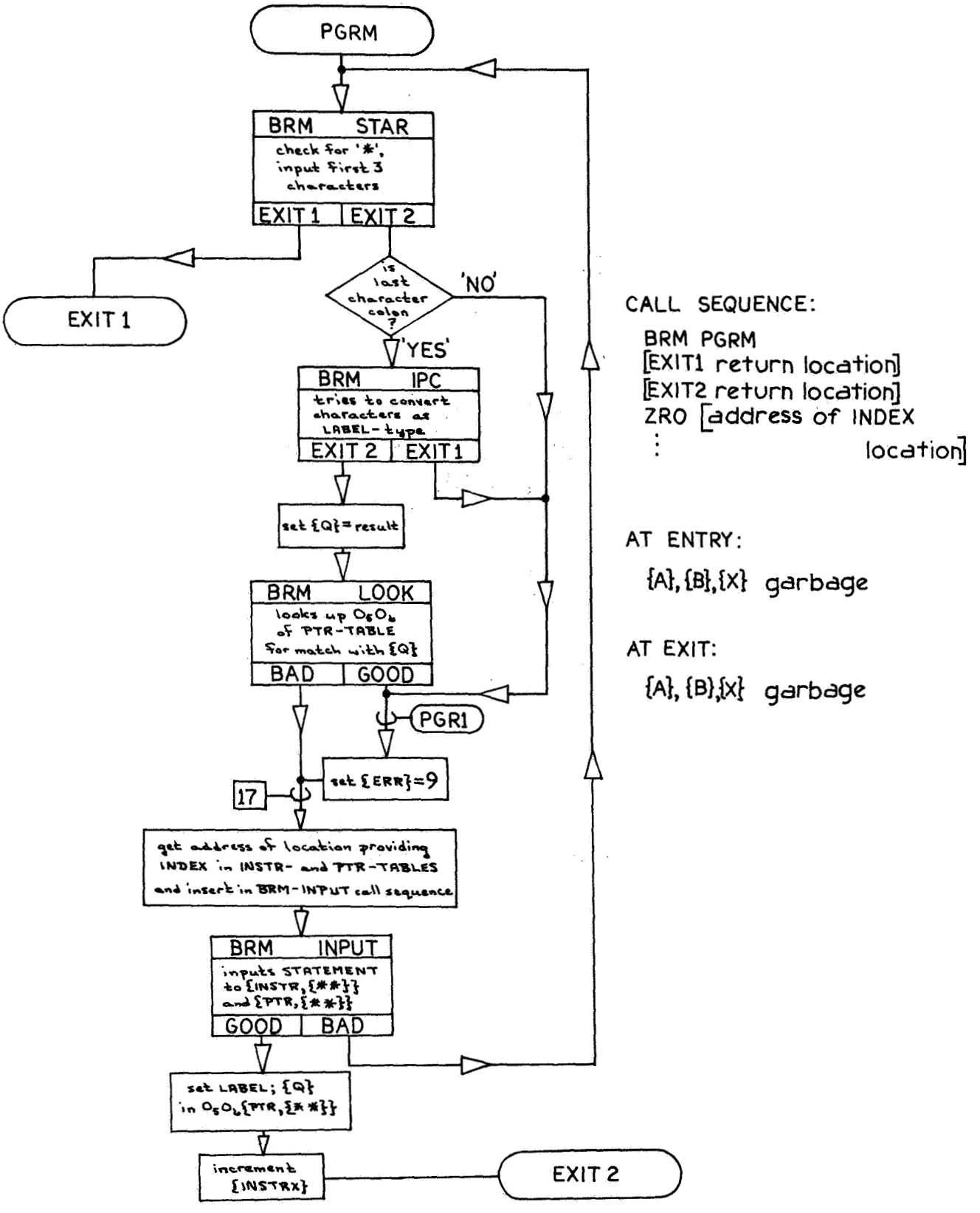
AT EXIT:

{A},{B},{X} garbage

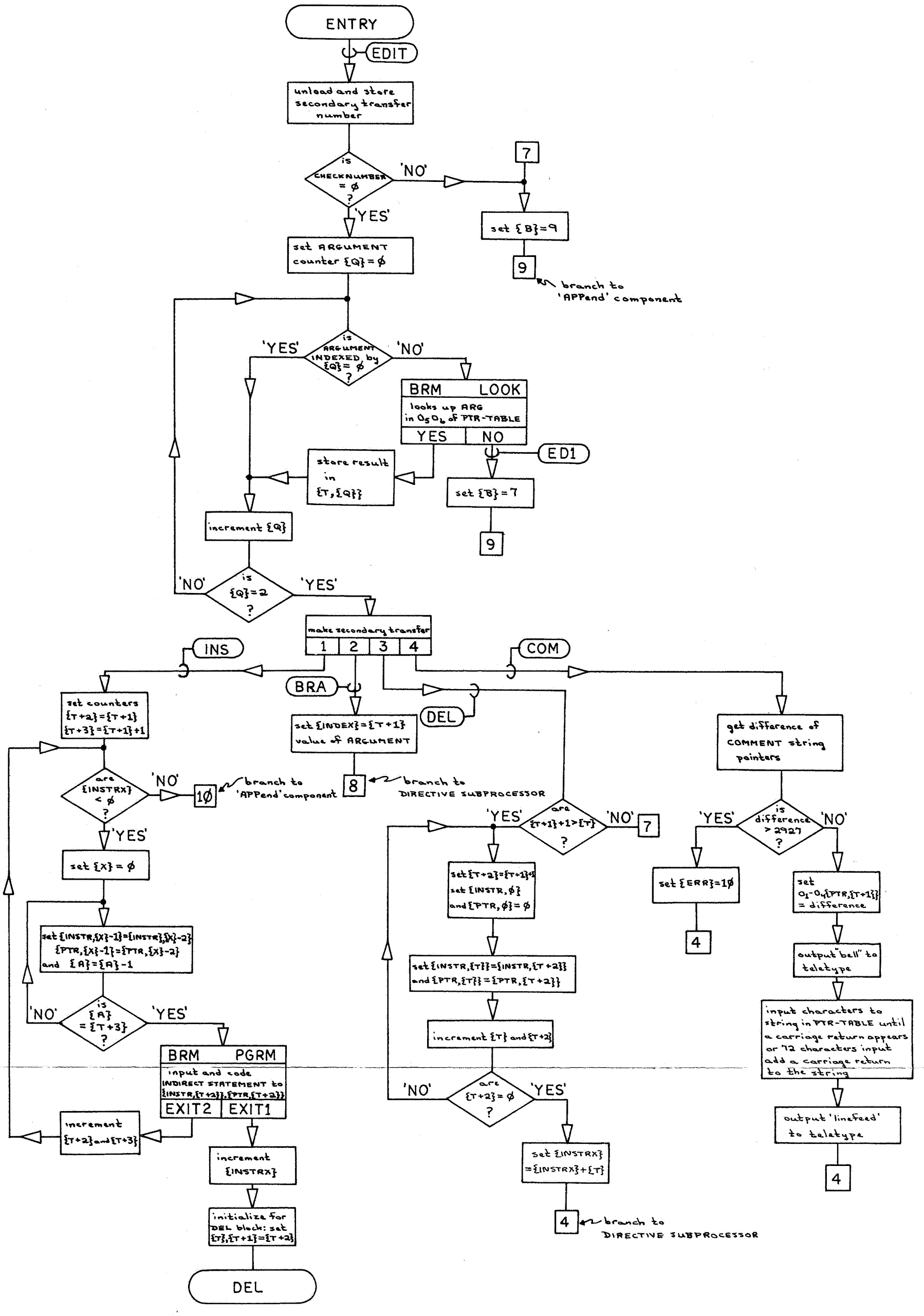
Flow Diagram 13 Flowchart of POP 'UNLO' for unloading a WORD from the IDIM-TABLE



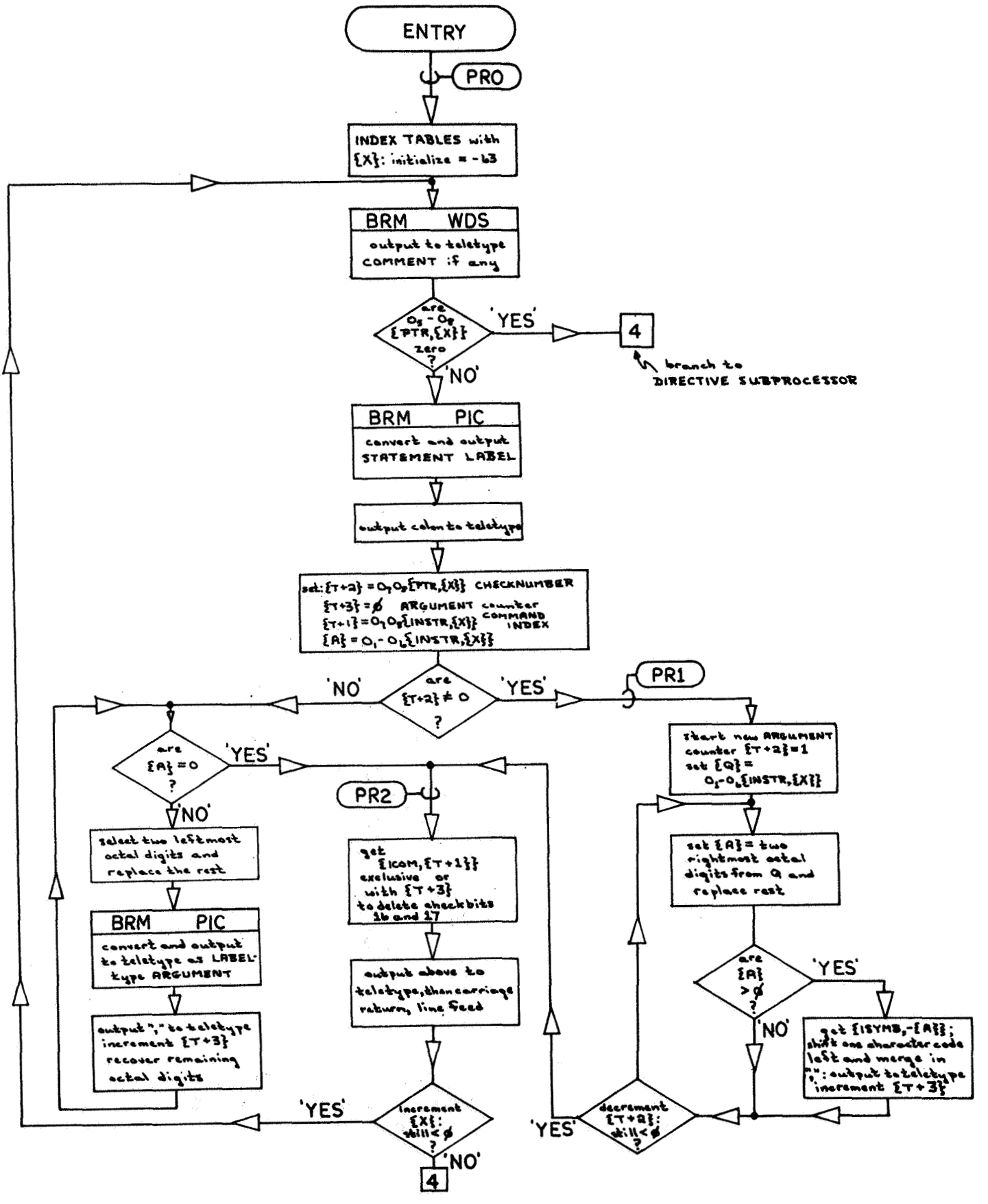
Flow Diagram 14. Flowchart for APPend operation appending an INDIRECT STATEMENT to a PROGRAM.



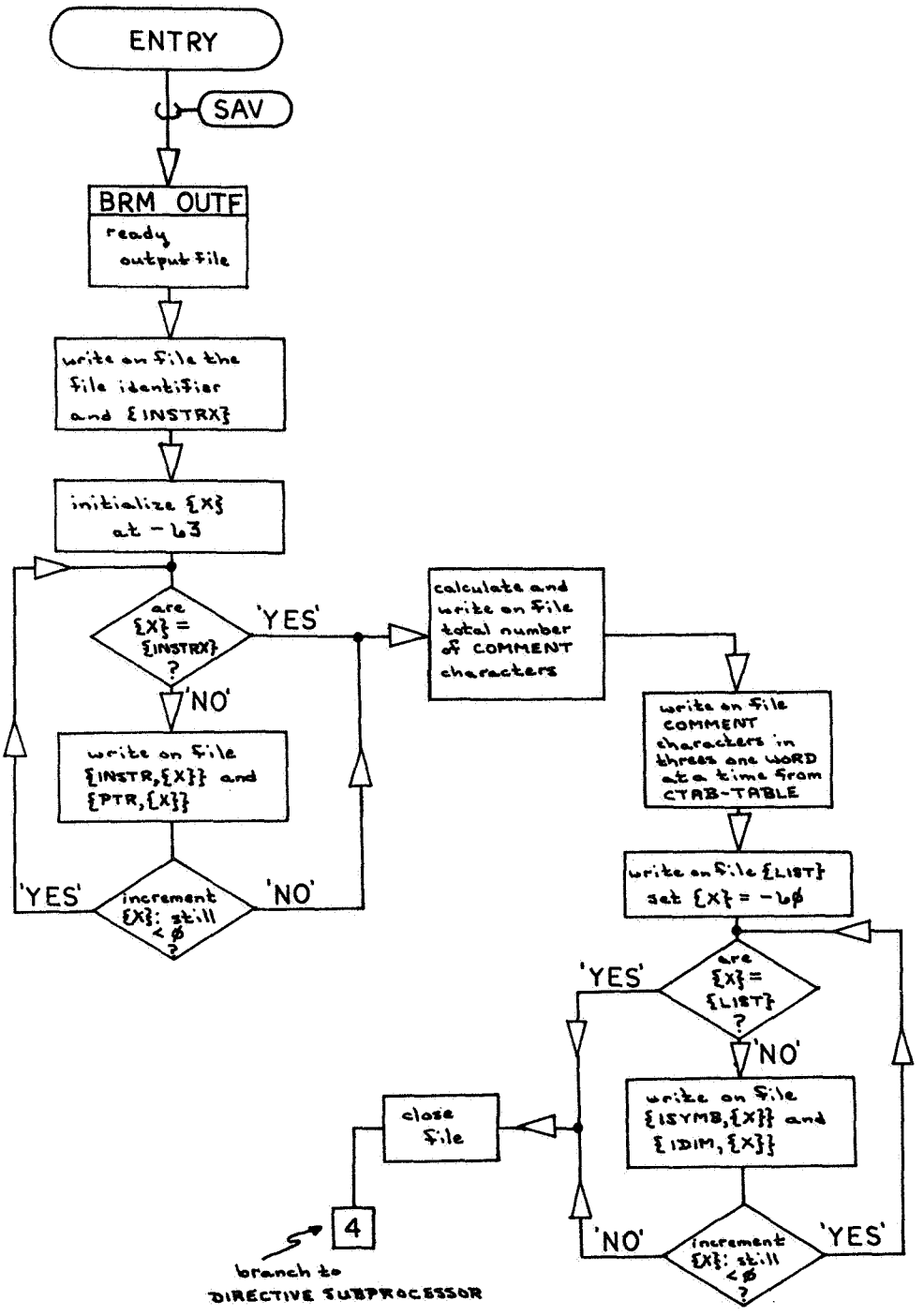
Flow Diagram 15. Flowchart of BRM PGRM coding an INDIRECT STATEMENT into specified WORDS of INSTR- and PTR-TABLES.



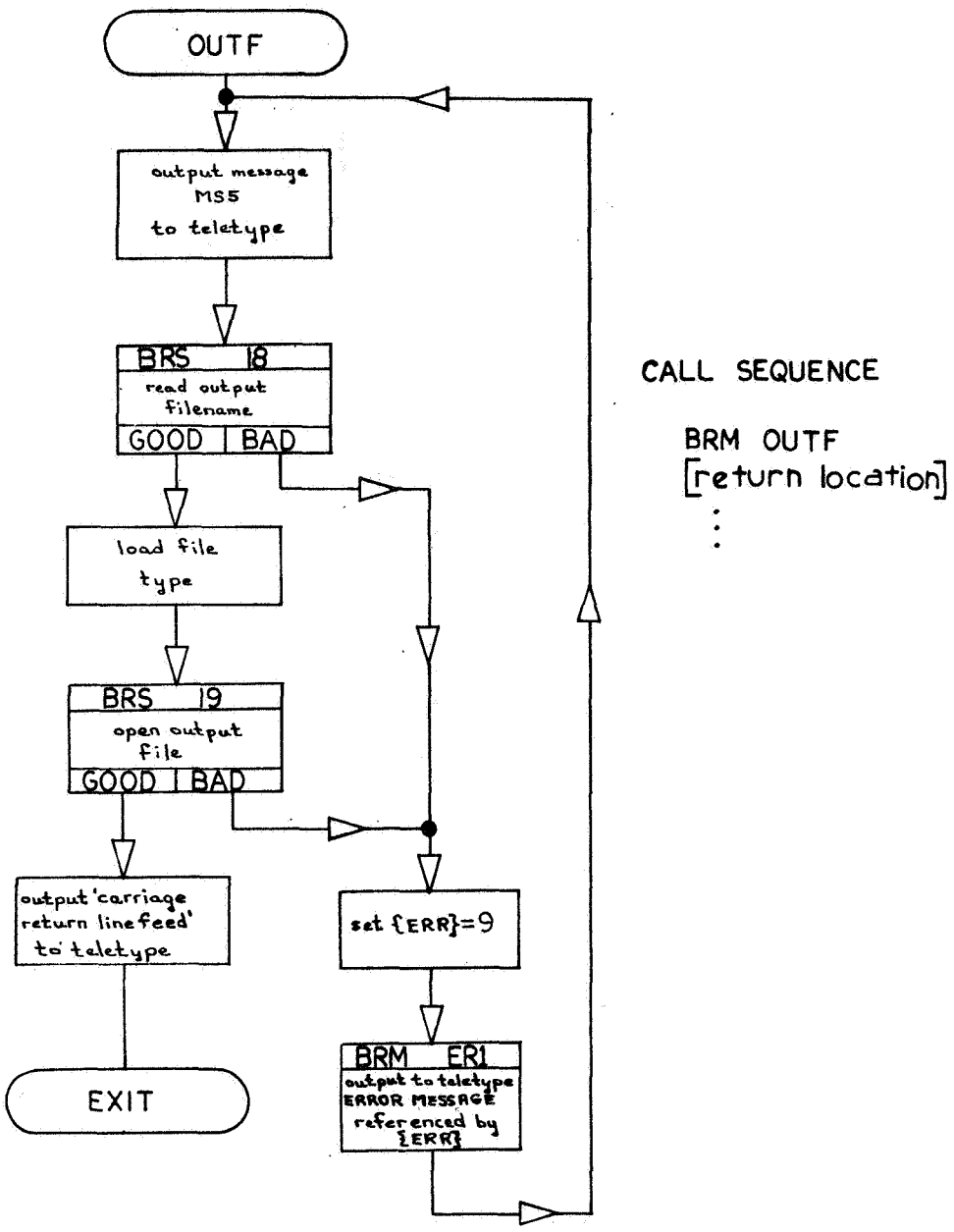
Flow Diagram 16. Flowchart of INSert, BRAnch, DELete, and COMment operations.



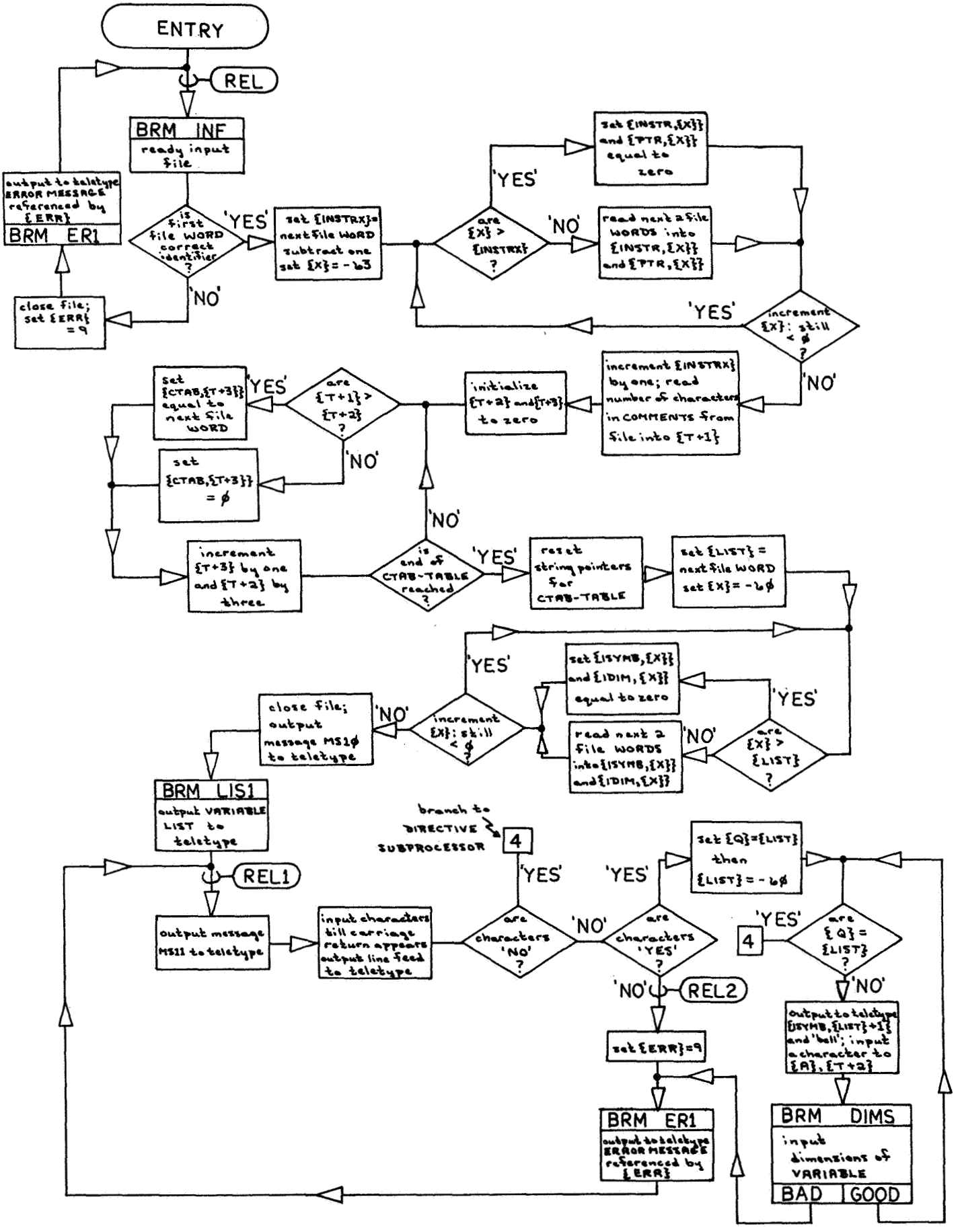
Flow Diagram 17. Flowchart of PROgram operation printing on the teletype a PROGRAM of INDIRECT STATEMENTS



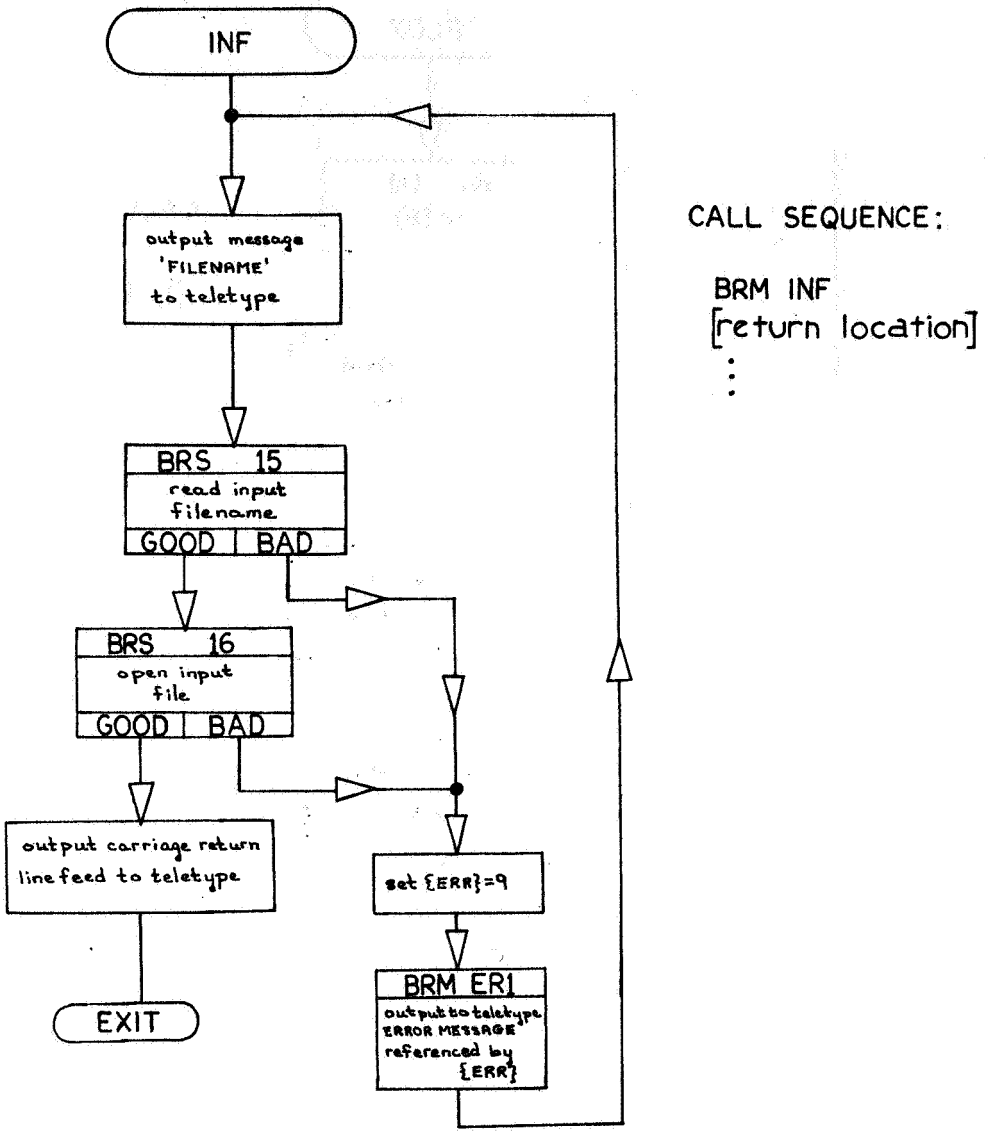
Flow Diagram 18. Flowchart of SAVe operation for storing a PROGRAM and its associated COMMENTS and VARIABLE LIST on a disc file.



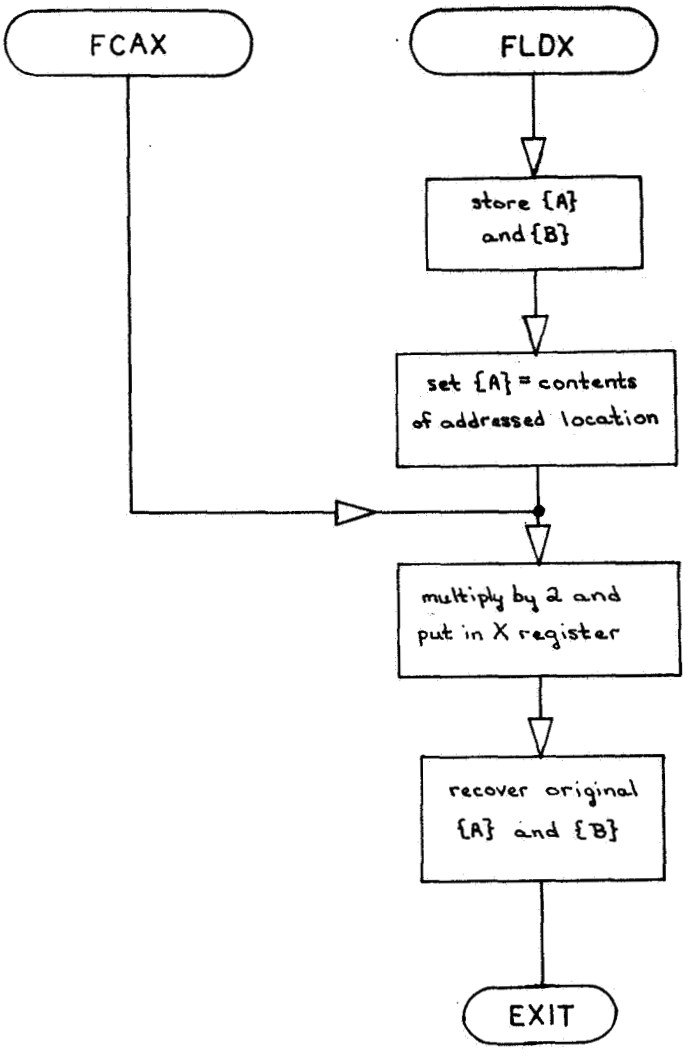
Flow Diagram 19. Flowchart of BRM OUTF readying a disc file for output



Flow Diagram 20. Flowchart of RESTORE operation for restoring a PROGRAM to core from a disc file.



Flow Diagram 21. Flowchart of BRM INF readying a disc file for input.

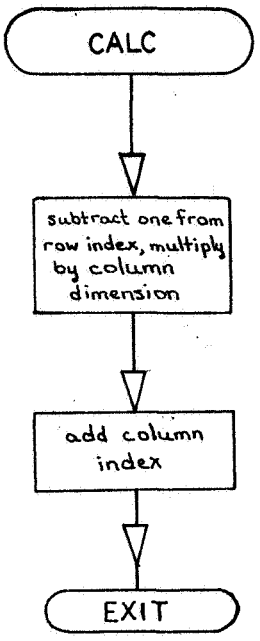


CALL SEQUENCES:

(i) FLDX [addr. of location containing INDEX]

(ii) FCAX

Flow Diagram 22. Flowchart of POP's FLDX and FCAX for getting INDICES in TYPE III TABLES.



CALL SEQUENCE:

CALC [address of location containing column size]

AT ENTRY:

{A} row index number

{B} column index number

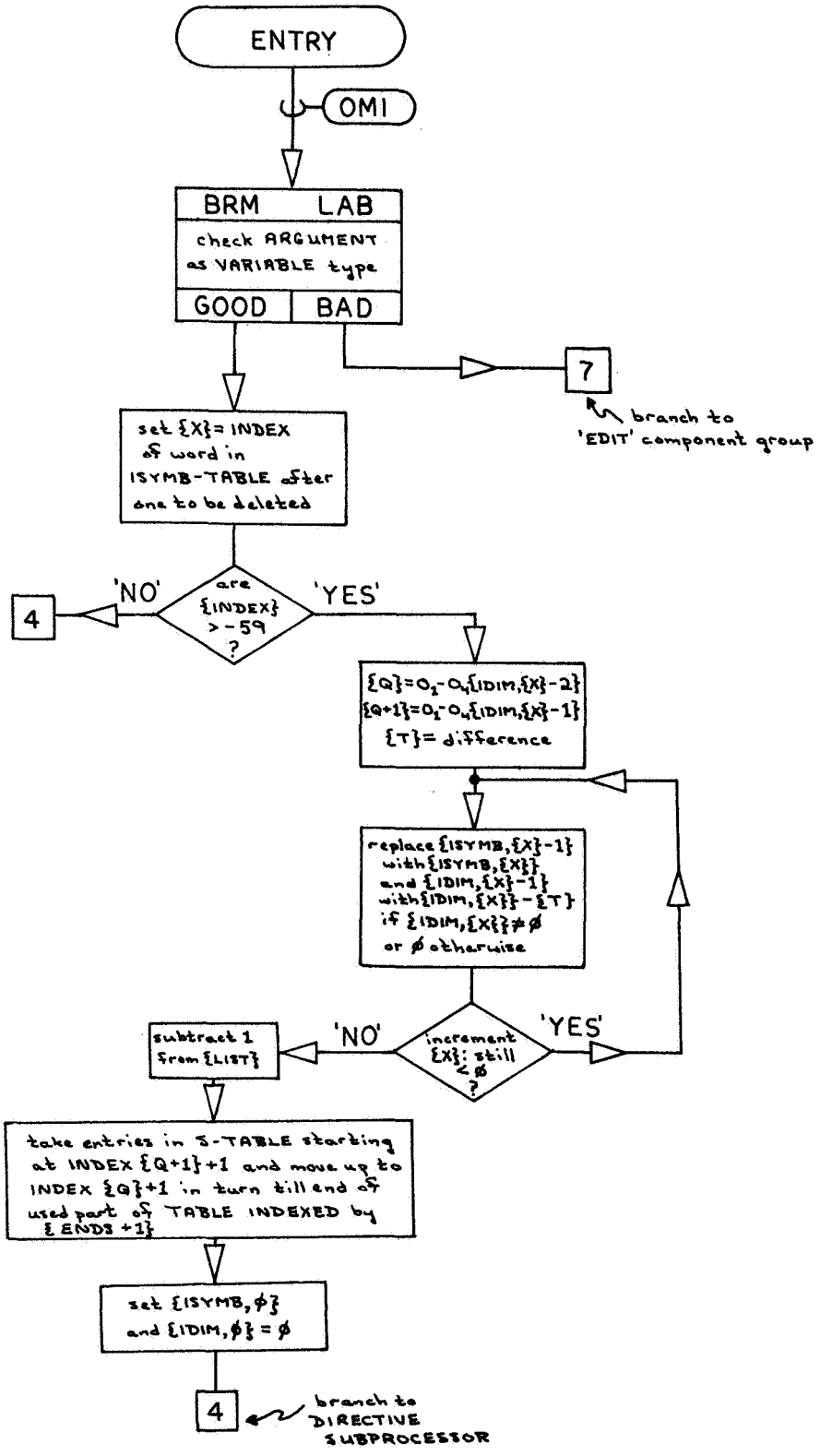
AT EXIT:

{A} INDEX in S-TABLE

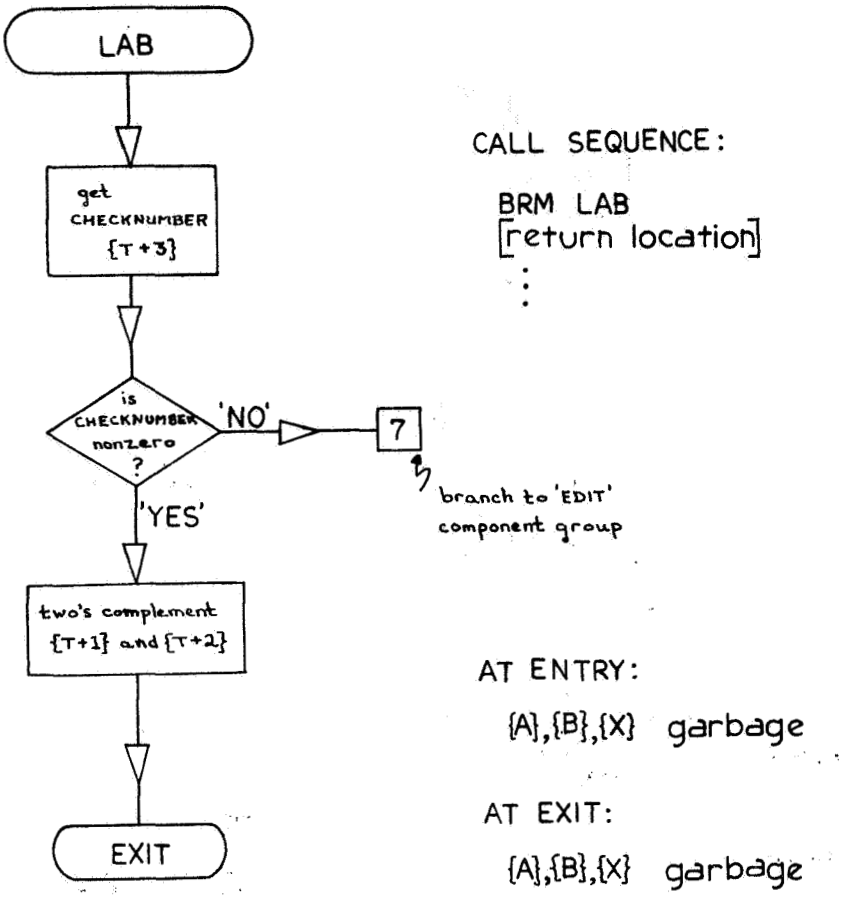
{B} garbage

{X} not used

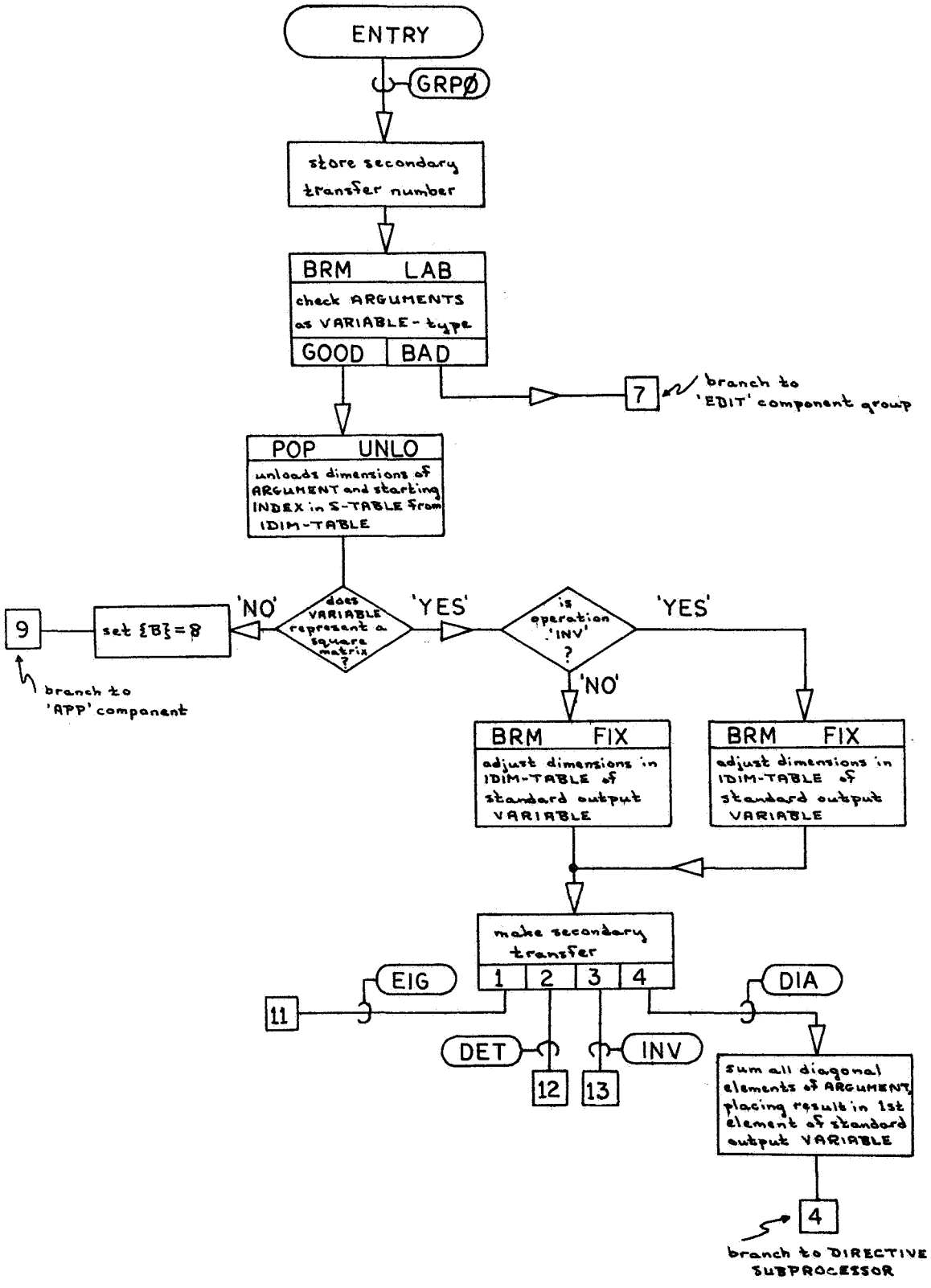
Flow Diagram 23. Flowchart of POP CALC for computing the relative INDEX in a BLOCK from the row and column indices of the matrix.



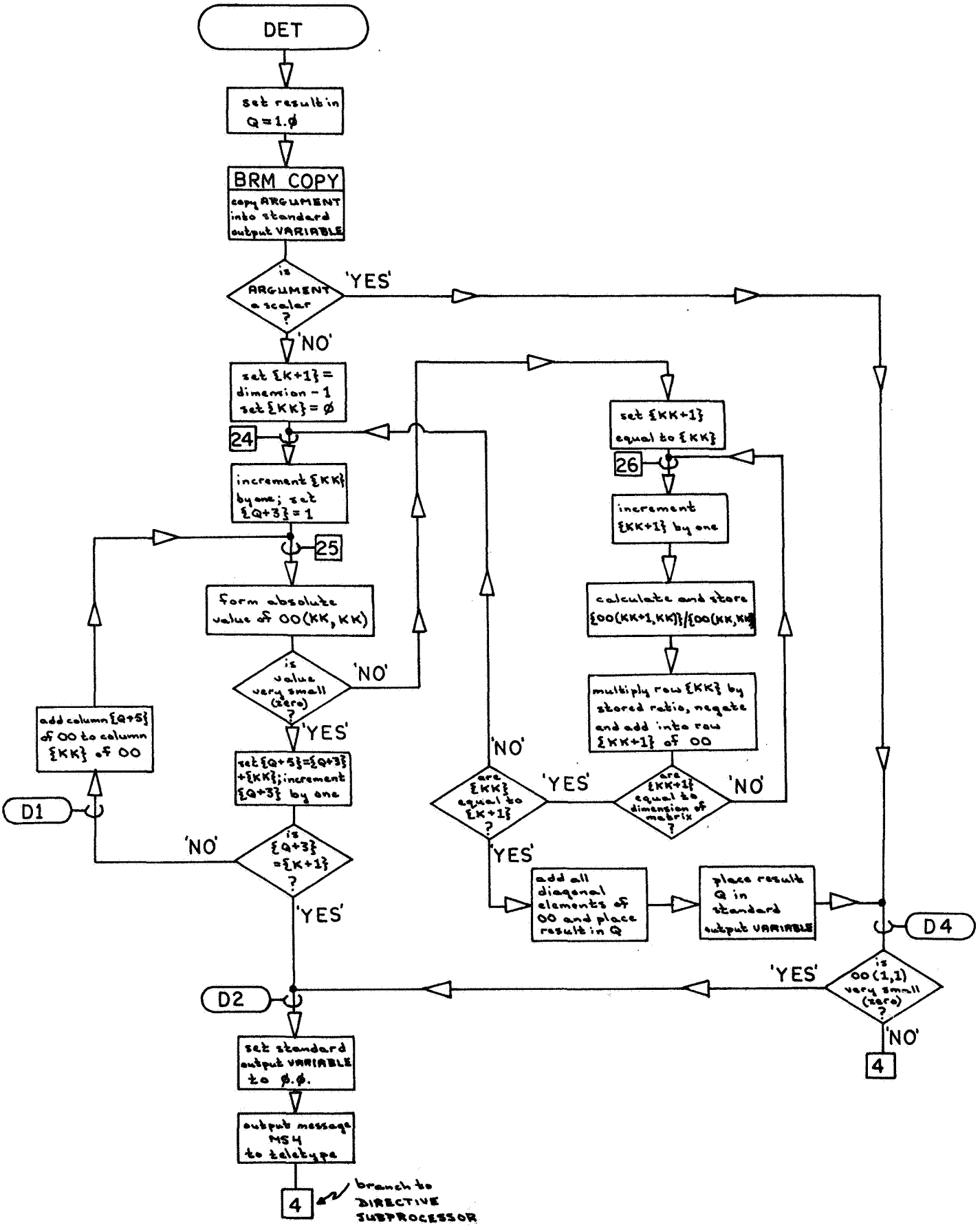
Flow Diagram 24. Flowchart of OMIt operation deleting a VARIABLE from the VARIABLE LIST.



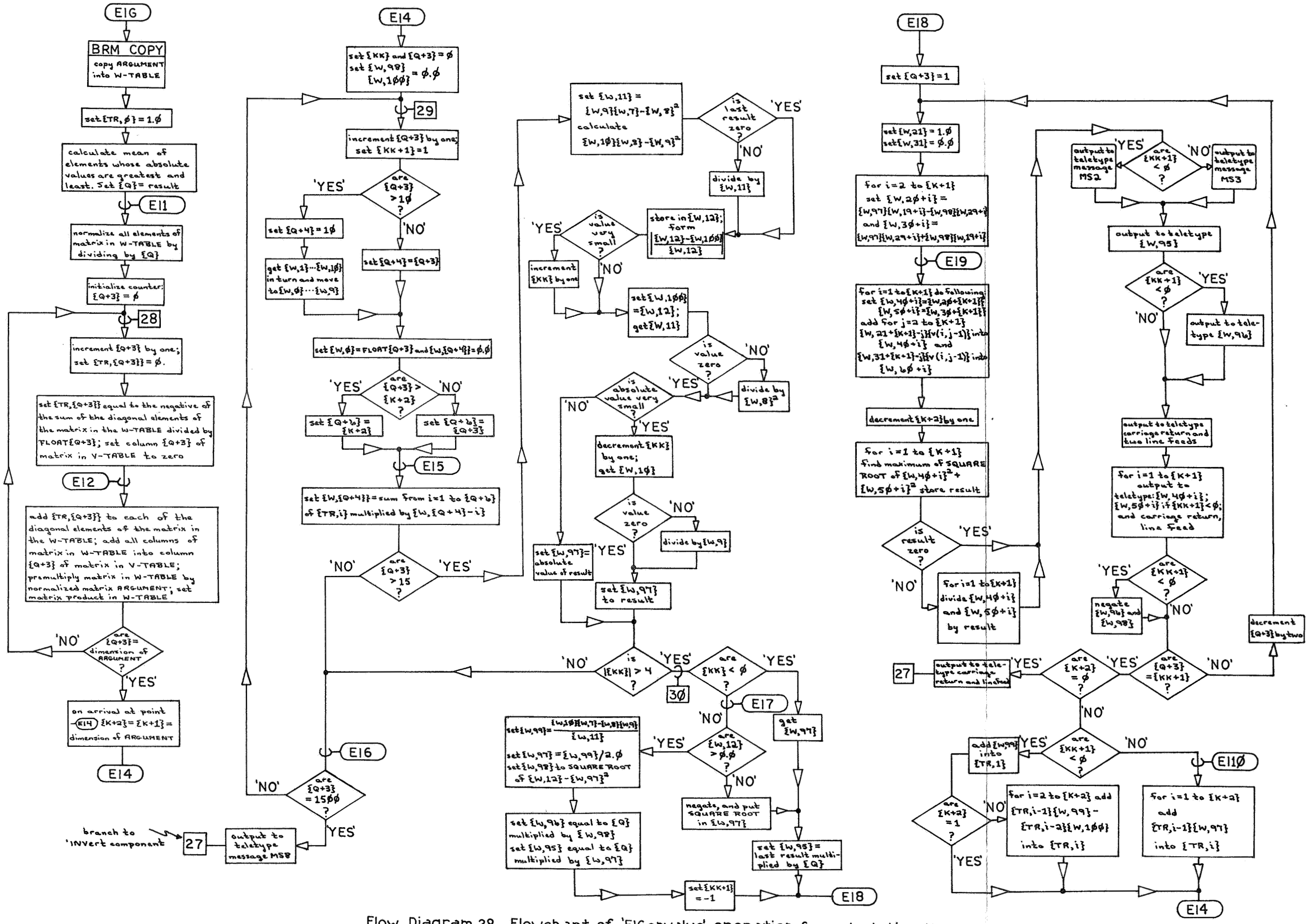
Flow Diagram 25. Flowchart for BRM LAB checking the type of ARGUMENTS of a STATEMENT



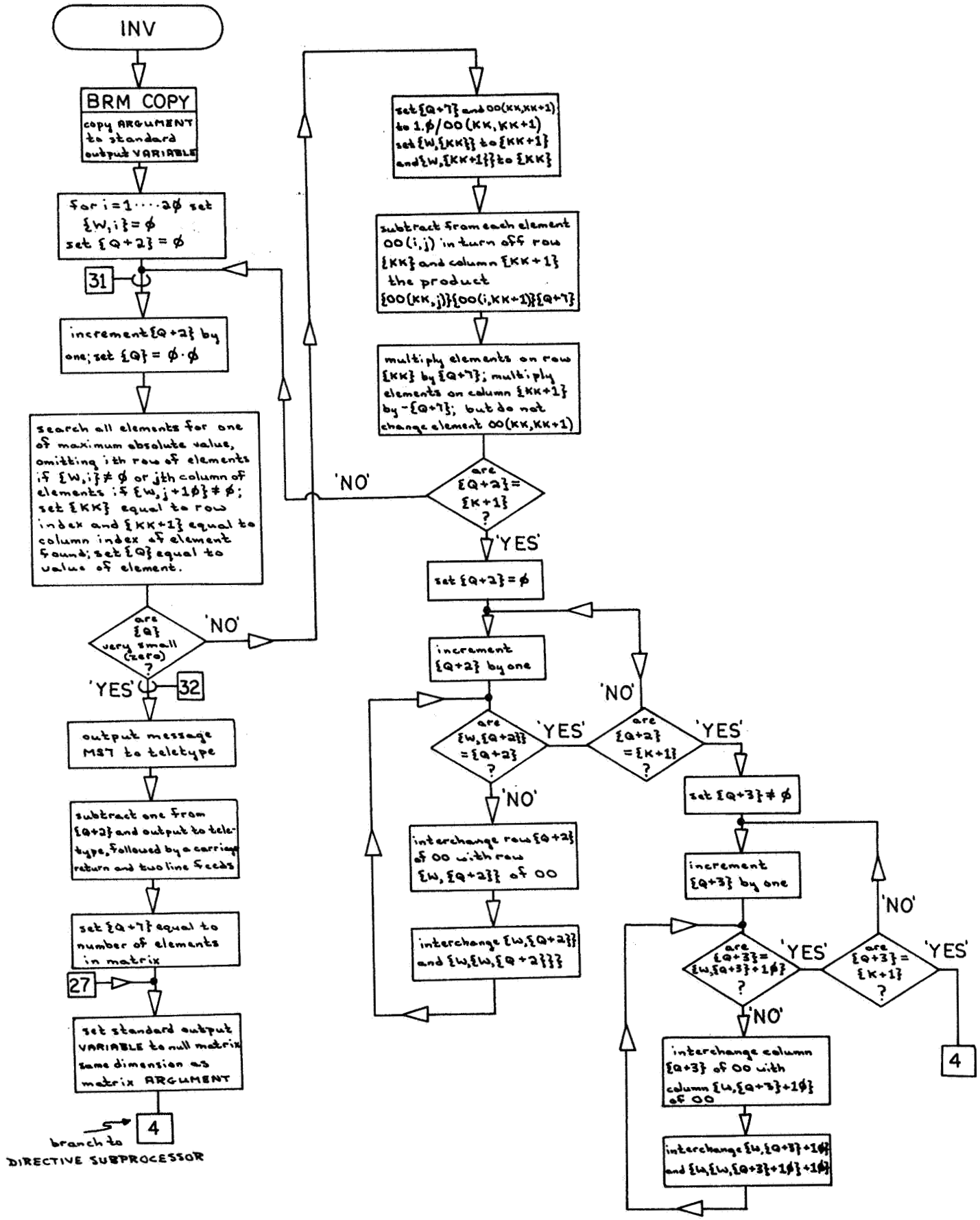
Flow Diagram 26. Flowchart of GROUPØ operations EIGenvalue, DETerminant, INVert, and DIAgonal sum.



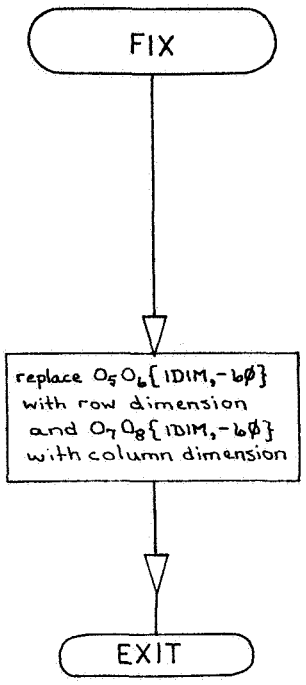
Flow Diagram 27. Flowchart of 'DETerminant' component for calculating the determinant of a VARIABLE.



Flow Diagram 28. Flowchart of 'EIGenvalue' operation for calculating the eigenvalues and eigenvectors of a square matrix.



Flow Diagram 29. Flowchart of 'INVert' operation for inverting a square matrix.



CALL SEQUENCE:

BRM FIX
[return location]
ZRO [addr. of location containing row size]
ZRO [addr. of location containing column size]
⋮

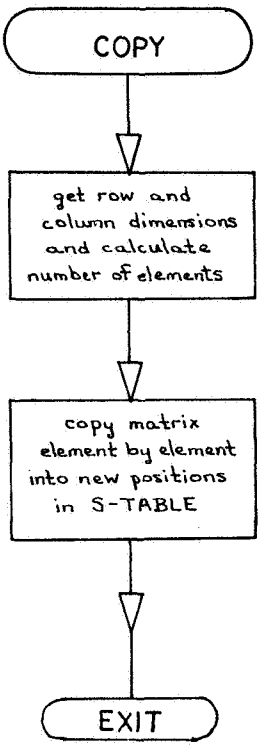
AT ENTRY:

{A}, {B}, {X} garbage

AT EXIT:

{A},{B},{X} garbage

Flow Diagram 30. Flowchart of BRM FIX adjusting the dimensions of the standard output VARIABLE.



CALL SEQUENCE:

```
BRM  COPY
[return location]
ZRO  [addr. of location containing row size]
ZRO  [addr. of location containing column size]
ZRO  [addr. of location containing old starting INDEX]
ZRO  [addr. of location containing new starting INDEX]
 .
 .
 .
```

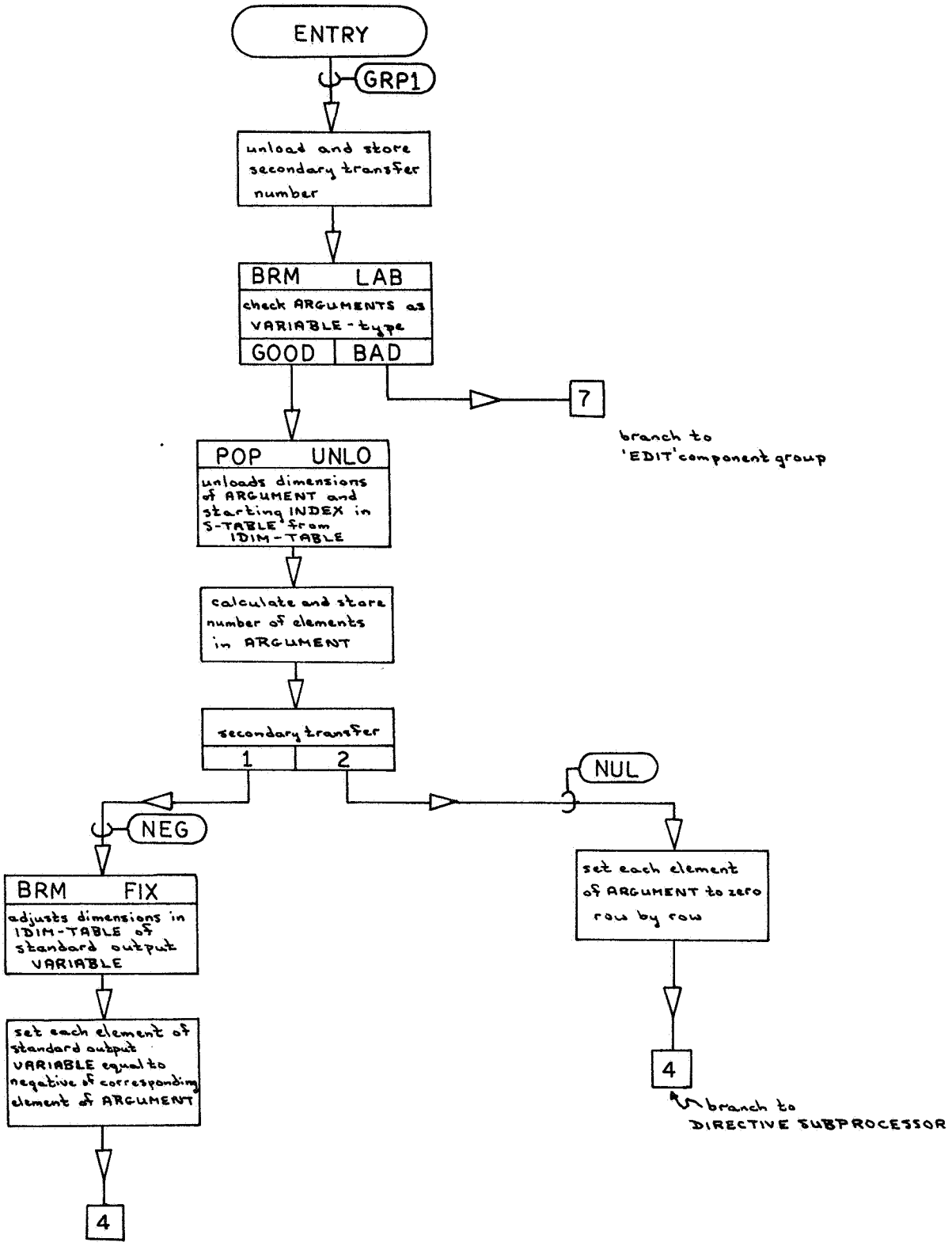
AT ENTRY:

{A},{B},{X}, garbage

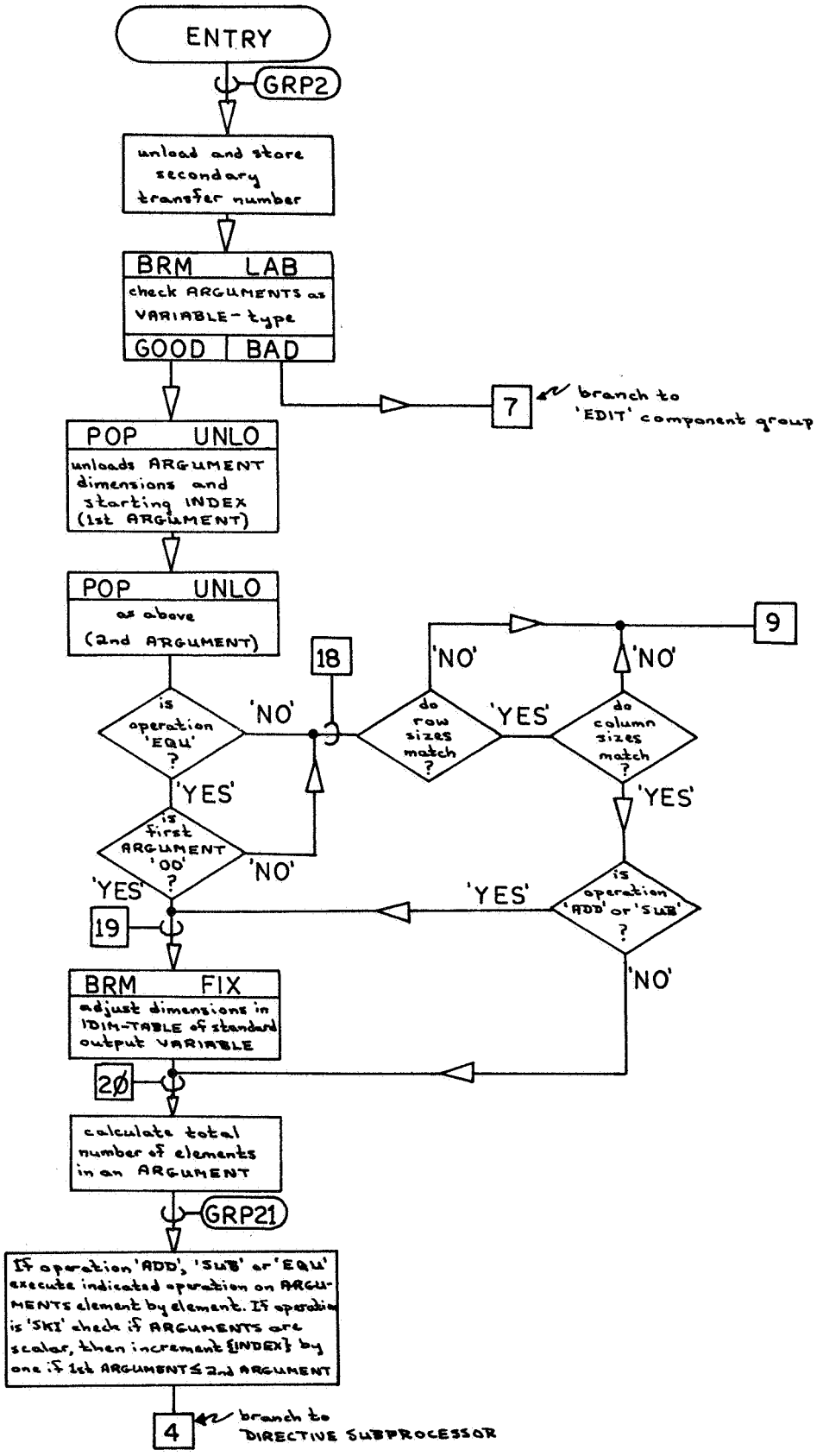
AT EXIT:

{A},{B},{X}, garbage

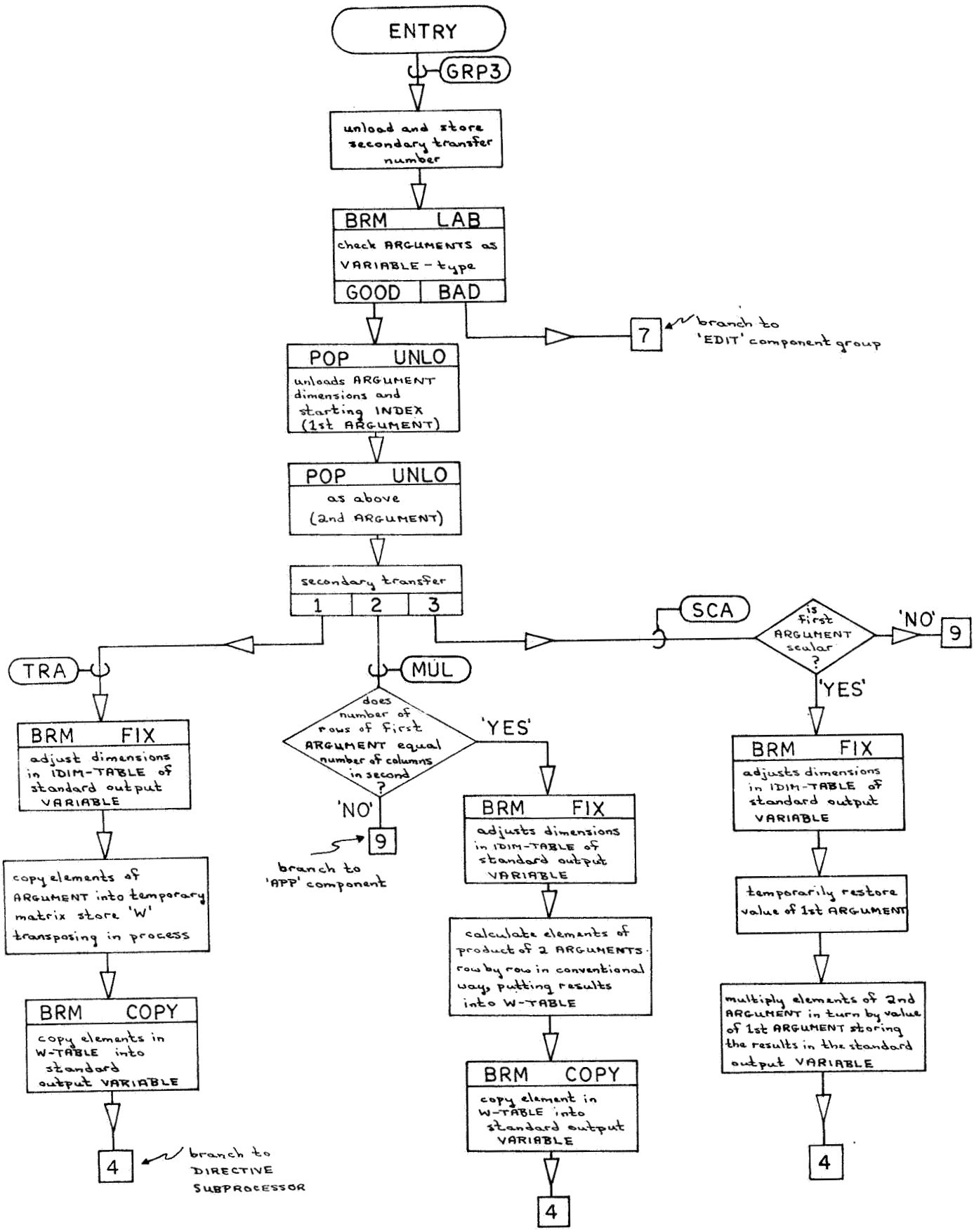
Flow Diagram 31. Flowchart of BRM COPY copying a matrix into a new position in the S-TABLE.



Flow Diagram 32. Flowchart of GROUP1 operations NULl, NEGate.

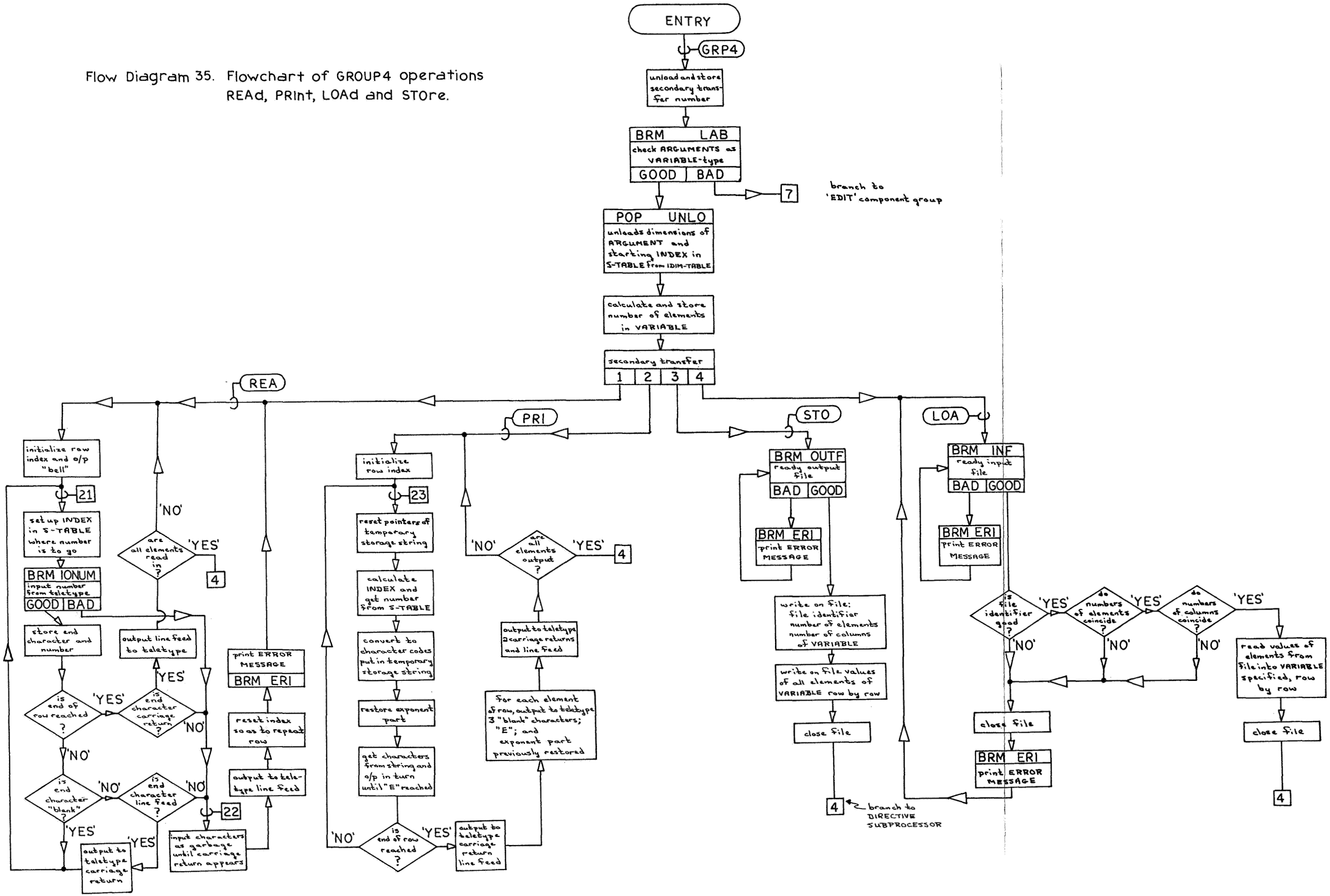


Flow Diagram 33. Flowchart of GROUP2 operations ADD, SUBtract, SKIp and EQUate.



Flow Diagram 34. Flowchart of GROUP3 operations MULtiply, SCAlar multiply and TRAnspose.

Flow Diagram 35. Flowchart of GROUP4 operations REAd, PRInt, LOAd and STOre.



7. COMPATIBILITY

AS IT STANDS, MAP language is incompatible with any other SDS 940 facility. Areas of compatibility can only arise if either modifications to the structure of the MAP processor are made, or if special interlinking routines are constructed.

As an example, it is possible by means of such a routine to create compatible disk files to transfer numerical values of matrices to MAP. An assembly language routine is given below, which may be called by a Fortran program to create a disk file usable by MAP. The routine is merely an adaptation of the 'STOre' component of MAP's EXECUTIVE SUBPROCESSOR. The subroutine call is

CALL STORE(A, M, N)

where

A is the matrix to be stored
M is the row dimension
N is the column dimension

The coding for the routine is as follows:[19]

```
$STORE  ZRO  RTN
        STX  TX
        LDA* 4ø4B        calculate number of elements
        STA  T+1              in matrix
        MUL* 4ø3B
        RSH  1
        STB  T
        LDA  =MG1
        LDB  =-1         output to teletype FILE NAME
        LDX  =1
        BRS  34
        CLA
        BRS  18          input filename
```

[19] The operation of this subroutine is not guaranteed.

cont'd

```
        BRU  *+8
        LDX  =3
        BRS  19                 open file
        BRU  *+5
        STA  T+2
        LDA  =66552B
        WIO  =1
        BRU  S1
        LDA  =MG2               come here if file is bad
        LDB  =-1                output to teletype WHAT?
        LDX  =1
        BRS  34
        BRU  STORE+7            go get a new file
MG1     ASC  '$FILE NAME/'
MG2     ASC  '$$WHAT?$$/'
S1      LDA  =31463146B         come here if file good,
        WIO  T+2                put on file: identifier,
        LDP  T                  number of elements, number
        WIO  T+2                of columns
        CBA
        WIO  T+2
        LDA  LNK
        ADD  4Ø2B               insert element loading in-
        STA  *+8                   struction
        CLA                     beginning of loop
        STA  T+3
        MIN  T+3
        LDA  T+3
        CLB
        LSH  1
        CAX                     calculate index of element
        ZRO
        WIO  T+2                put element of matrix on file
        CBA
        WIO  T+2
        LDA  T+3                end of loop
        SKE  T
        BRU  S1+11
        LDA  T+2                close file
        BRS  2Ø
        LDX  TX                 return
        BRR  RTN
LNK     LDP  Ø,2
T       BSS  4
TX      ZRO                     storage
RTN     ZRO
```

If the filename is to be provided by the calling sequence instead of input from the teletype, the routine would be slightly more complicated.

It can be seen that the provision of linking routines need not necessarily be too difficult a task. More complex forms of iteration between MAP and other facilities, such as the automatic entry of MAP for the execution of certain operations, during the course of execution of some other program, are not feasible.

8. CHANGING THE LIMITS ON STORAGE SPACE

DUE TO THE MANNER in which INDICES are stored in the WORDS of TABLES, it is in general not possible to enlarge the TABLES of MAP. As the processor stands at the time of publication, only the CTAB-TABLE and the ICOM- and PLACE-TABLES can be extended. Thus only the number of COMMENTS and the number of COMMANDS can be increased. If other machines in which less storage space is available are used, then either or both the CTAB- and S-TABLE would be reduced in size.

It is also possible to increase the maximum permissible dimensions of MAP matrices. This is more difficult, however, and not recommended on the grounds that the processing of large matrices on the SDS 940 is uneconomical.

COMMENTS

To change the number of COMMENTS it is only necessary to redefine the size of the CTAB-TABLE, and to set {ENDS+3} equal to three times the new number of WORDS in the TABLE minus 73.

VARIABLE STORAGE

To decrease the VARIABLE STORAGE space it is necessary to redefine the length of the S-TABLE. Let the new number of locations be n: n - 2 must then be divisible by 2Ø. {ENDS} are set equal to (n - 2)/2 and {ENDS+2} to (n - 2)/2 + 8.

NUMBER OF COMMANDS

If it is desired to make available new COMMANDS the ICOM- and PLACE-TABLES must be extended, placing the new COMMAND character codes in the ICOM-TABLE and their transfer locations in the PLACE-TABLE. In addition in the BRM 'INPUT', the BRM 'LOOK' used to search the ICOM-TABLE for the COMMAND input by the user, must have its call sequence changed. If n COMMANDS are added then the call sequence should become

```
BRM  LOOK
ZRO  =77777777B
ZRO  =[insert value of -(3Ø+n)]
ZRO  ICOM
 .
 .
 .
```

The placing of the sections of coding dealing with the new COMMANDS is not important, except that if they make use of user-defined POP's, the coding must be inserted after the POP definitions.

APPENDIX A1

THE FOLLOWING IS a list of the BRS's and SYSPOPS used in the MAP processor. A full list of all available system routines with descriptions of their operation may be found in [3].

BRS's

15	read input filename
16	open input file
17	close all files
18	read output filename
19	open output file
2ø	close file
21	floating-point negate
34	output string
36	output number
38	input number
51	fixed to floating conversion
53	output floating-point number
78	arm interrupt

SYSPOPS

FAD	floating-point add
FDV	floating-point divide
FMP	floating-point multiply
FSB	floating-point subtract
GCI	get character and increment
ISC	internal to string conversion (floating-point output[20])

[20] In the symbolic listings of the processor program the SYSPOP 'ISC' appears as the SYSPOP 'SIC'. Due to an error in the Harvard University SDS 940 ARPAS assembler 'ISC' assembles as 'SIC' and vice-versa.

LDP	load pointers (or floating-point number)
STP	store pointers (or floating-point number)
TCI	teletype character input
TCO	teletype character output
WCI	write character and increment
WIO	word input and output

APPENDIX A2

THE FOLLOWING IS a list of all messages output by the MAP processor to the teletype. It includes a list of all ERROR MESSAGES with their reference numbers.

Error Messages

Reference number	message
1	®®NO MORE STATEMENTS ®®
2	®®VARIABLE UNDEFINED ®®
3	®®TOO MANY VARIABLES ®®
4	®®VARIABLE STORE FULL ®®
5	®®ILLEGAL VARIABLE ®®
6	®®MATRIX IS OVERSIZE ®®
7	®®LABEL UNDEFINED ®®
8	®®INCOMPATIBLE MATRIX ®®
9	®®WHAT? ®®
1ø	®®COMMENT STORE FULL ®®

Other Messages

MS1	®®MATRIX MANIPULATOR (11ø-3) MAY 1968®®
MS2	®COMPLEX EIGENVALUE
MS3	®REAL EIGENVALUE
MS4	®DETERMINANT ZERO®®
MS5	FILE NAME
MS6	®®STOP IN STATEMENT
MS7	®MATRIX SINGULAR - RANK =
MS8	®COMPUTATION FAILURE®®
MS1ø	®VARIABLES USED®®
MS11	®REDIMENSION VARIABLES?®®

In the above messages ® denotes a carriage return and line feed.

INDEX OF COMMANDS

REFERENCES

[1] P. M. Newbold. "M.A.P. - A Conversational Language for Numerical Matrix Operations. Part I: User's Manual." Harvard University Technical Report TR561, May 1968.

[2] B. W. Lampson, L. P. Deutsch, L. L. Barnes. "Floating-point System Manual." Contract SD-185, Document No. 30 10 40, University of California, February 1966.

[3] "SDS 940 Timesharing System Technical Manual." Scientific Data Systems Publication 98 11 16A, November 1967.

[4] "SDS 940 Computer Reference Manual." Scientific Data Systems Publication 90 06 40B, September 1967.

[5] "TAP Reference Manual for SDS 940 Time-sharing Computer System." Scientific Data Systems Publication 90 11 17A, October 1967.

[6] B. W. Lampson, L. P. Deutsch, L. L. Barnes. "S.P.S. Timesharing String Process System Reference Manual." Contract SD-185, Document No. 30 10 20, March 1966.

[7] W. Jennings. 'First Course in Numerical Methods.' Sections 5.3, 5.4 and 22.5. Macmillan, N. Y. 1964.

Joint Services Electronics Program

N00014-67-A-0298-0006, 0005, and 0008

Academy Library (DFSLB)
U. S. Air Force Academy
Colorado Springs, Colorado 80912

AEDC (ARO, INC)
Attn: Library/Documents
Arnold AFB, Tenn. 37389

Aeronautics Library
Graduate Aeronautical Laboratories
California Institute of Technology
1201 E. California Blvd.
Pasadena, California 91109

Aerospace Corporation
P. O. Box 95085
Los Angeles, Calif. 90045
Attn: Library Acquisitions Group

Airborne Instruments Laboratory
Deerpark, New York 11729

AFAL (AVTE/R. D. Larson)
Wright-Patterson AFB
Ohio 45433

AFCRL (CRMXLR)
ARCRL Research Library, Stop 29
L. G. Hanscom Field
Bedford, Mass. 01731

AFETR (ETLIG - 1)
STINFO Officer (for library)
Patrick AFB, Florida 32925

AFETR Technical Library
(ETV, MU-135)
Patrick AFB, Florida 32925

AFFTC (FRBPP-2)
Technical Library
Edwards AFB, Calif. 93523

AFGC (PHRPS-12)
Eglin AFB
Florida 32542

ARL (ARIY)
Wright-Patterson AFB
Ohio 45433

AUL3T-9663
Maxwell AFB
Alabama 36112

Mr. Henry L. Bachman
Assistant Chief Engineer
Wheeler Laboratories
122 Cuttermill Road
Great Neck, N. Y. 11021

Bendix Pacific Division
11600 Sherman Way
North Hollywood, Calif. 91605

Colonel A. D. Blue
RTD (RTTL)
Bolling AFB
Washington, D. C. 20332

California Institute of Technology
Pasadena, California 91109
Attn: Documents Library

Carnegie Institute of Technology
Electrical Engineering Dept.
Pittsburg, Pa. 15213

Central Intelligence Agency
Attn: OCR/DD Publications
Washington, D. C. 20505

Chief of Naval Operations
OP-07
Washington, D. C. 20350 [2]

Chief of Naval Research
Department of the Navy
Washington, D. C. 20360
Attn: Code 427 [3]

Commandant
U. S. Army and General Staff College
Attn: Secretary
Fort Leavenworth, Kansas 66370

Commander
Naval Air Development and
Material Center
Johnsville, Pennsylvania 18974

Commanding General
Frankford Arsenal
Attn: SMUFA-L6000 (Dr. Sidney Ross)
Philadelphia, Pa. 19137

Commandant
U. S. Army Air Defense School
Attn: Missile Sciences Div. C and S Dept.
P. O. Box 9390
Fort Bliss, Texas 79916

Commander
U. S. Naval Air Missile Test Center
Point Magu, California 93041

Commanding General
Attn: STEWS-WS-VT
White Sands Missile Range
New Mexico 88002 [2]

Commanding General
U. S. Army Electronics Command
Fort Monmouth, N. J. 07703
Attn: AMSEL-SC
RD-D
RD-G
RD-GF
RD-MAT
XL-D
XL-E
XL-C
XL-S
HL-D
HL-CT-R
HL-CT-P
HL-CT-L
HL-CT-O
HL-CT-I
HL-CT-A
NL-D
NL-A
NL-P
NL-R
NL-S
KL-D
KL-E
KL-S
KL-T
VL-D
WL-D

Commanding General
U. S. Army Material Command
Attn: AMCRD-RS-DE-E
Washington, D. C. 20315

Commanding General
U. S. Army Missile Command
Attn: Technical Library
Redstone Arsenal, Alabama 35809

Commanding Officer
Naval Avionics Facility
Indianapolis, Indiana 46241

Commanding Officer
U. S. Army Limited War Laboratory
Attn: Technical Director
Aberdeen Proving Ground
Aberdeen, Maryland 21005

Commanding Officer
U. S. Army Materials Research Agency
Watertown Arsenal
Watertown, Massachusetts 02172

Commanding Officer
U. S. Army Security Agency
Arlington Hall
Arlington, Virginia 22212

Commanding Officer and Director
U. S. Naval Underwater Sound Lab.
Fort Trumbull
New London, Conn. 06840

Defense Documentation Center
Attn: TISIA
Cameron Station, Bldg. 5
Alexandria, Virginia 22314 [20]

Det No. 6, OAR (LODAR)
Air Force Unit Post Office
Los Angeles, Calif. 90045

Director
Advanced Research Projects Agency
Department of Defense
Washington, D. C. 20301

Director for Materials Sciences
Advanced Research Projects Agency
Department of Defense
Washington, D. C. 20301

Director
Columbia Radiation Laboratory
Columbia University
538 West 120th Street
New York, New York 10027

Director
Coordinated Science Laboratory
University of Illinois
Urbana, Illinois 61803

Director
Electronics Research Laboratory
University of California
Berkeley, California 94720

Director
Electronic Sciences Laboratory
University of Southern California
Los Angeles, California 90007

Director
Microwave Laboratory
Stanford University
Stanford, California 94305

Director - Inst. for Exploratory
Research
U. S. Army Electronics Command
Attn: Mr. Robert O. Parker
Executive Secretary, JSTAC
(AMSEL-XL-D)
Fort Monmouth, N. J. 07703

Director
National Security Agency
Fort George G. Meade
Maryland 20755
Attn: James T. Tippett

Director, Naval Research Laboratory
Technical Information Officer
Washington, D. C.
Attn: Code 2000 [6]

Director
Research Laboratory of Electronics
Massachusetts Institute of Technology
Cambridge, Mass. 02139

Director
Stanford Electronics Laboratories
Stanford University
Stanford, California 94305

Commanding Officer
Naval Ordnance Laboratory
Corona, California 91720

Commanding Officer
Naval Ordnance Laboratory
White Oak, Maryland 21502 [2]

Commanding Officer
Naval Ordnance Test Station
China Lake, Calif. 93555

Commanding Officer
Naval Training Device Center
Orlando, Florida 32811

Commanding Officer
Office of Naval Research Branch Office
1030 East Green Street
Pasadena, California

Commanding Officer
Office of Naval Research Branch Office
219 South Dearborn Street
Chicago, Illinois 60604

Commanding Officer
Office of Naval Research Branch Office
495 Summer Street
Boston, Massachusetts 02210

Commanding Officer
Office of Naval Research Branch Office
207 West 24th Street
New York, New York 10011

Commanding Officer
Office of Naval Research Branch Office
Box 39, Fleet Post Office
New York 09510 [2]

Commanding Officer
U. S. Army Electronics R & D Activity
White Sands Missile Range
New Mexico 88002

Commanding Officer
U. S. Army Engineer R & D Laboratory
Attn: STINFO Branch
Fort Belvoir, Virginia 22060

Commanding Officer
U. S. Army Research Office (Durham)
Attn: CRD-AA-IP (Richard O. Ulsh)
Box CM, Duke Station
Durham, North Carolina 27706

Commanding General
USASTRATCOM
Technical Information Center
Fort Huachuca, Arizona 85613

Commanding Officer
Harry Diamond
Attn: Dr. Berthold Altman (AMXDO-TI)
Connecticut Ave. & Van Ness St. NW
Washington, D. C. 20438

Commanding Officer
Human Engineering Laboratories
Aberdeen Proving Ground
Maryland 21005

Commanding Officer
U. S. Army Ballistics Research Lab.
Attn: V. W. Richards
Aberdeen Proving Ground
Maryland 21005

Director, USAF Project RAND
Via: Air Force Liaison Office
The RAND Corporation
1700 Main Street
Santa Monica, Calif. 90406
Attn: Library

Director
U. S. Army Engineer Geodesy,
Intelligence and Mapping
Research and Development Agency
Fort Belvoir, Virginia 22060

Director
U. S. Naval Observatory
Washington, D. C. 20390

Director, U. S. Naval Security Group
Attn: G43
3801 Nebraska Avenue
Washington, D. C. 20390

Division of Engineering and Applied
Physics
130 Pierce Hall
Harvard University
Cambridge, Massachusetts 02138

Professor A. A. Dougal, Director
Laboratories for Electronics and
Related Sciences Research
University of Texas
Austin, Texas 78712

ESD (ESTI)
L. G. Hanscom Field
Bedford, Mass. 01731 [2]

European Office of Aerospace Research
Shell Building
47 Rue Cantersteen
Brussels, Belgium [2]

Colonel Robert E. Fontana
Dept. of Electrical Engineering
Air Force Institute of Technology
Wright-Patterson AFB, Ohio 45433

General Electric Company
Research Laboratories
Schenectady, New York 12301

Professor Nicholas George
California Institute of Technology
Pasadena, California 91109

Goddard Space Flight Center
National Aeronautics and Space Admin.
Attn: Library, Documents Section
Code 252
Green Belt, Maryland 20771

Dr. John C. Hancock, Director
Electronic Systems Research Laboratory
Purdue University
Lafayette, Indiana 47907

Dr. H. Harrison, Code RRE
Chief, Electrophysics Branch
National Aeronautics and Space Admin.
Washington, D. C. 20546

Head, Technical Division
U. S. Naval Counter Intelligence
Support Center
Fairmont Building
4420 North Fairfax Drive
Arlington, Virginia 22203

Headquarters
Defense Communications Agency
The Pentagon
Washington, D. C. 20305

Dr. L. M. Hollingsworth
ARCRL (CRN)
L. G. Hanscom Field
Bedford, Massachusetts 01731

Hunt Library
Carnegie Institute of Technology
Schenely Park
Pittsburgh, Pa. 15213

The Johns Hopkins University
Applied Physics Laboratory
8621 Georgia Avenue
Silver Spring, Maryland 20910
Attn: Boris W. Kuvshinoff
Document Librarian

Lt. Col. Robert B. Kalisch
Chief, Electronics Division
Directorate of Engineering Sciences
Air Force Office of Scientific Research
Arlington, Virginia 22209 [5]

Colonel Kee
AFRSTE
Hqs. USAF
Room ID-429, The Pentagon
Washington, D. C. 20330

Dr. S. Benedict Levin, Director
Institute for Exploratory Research
U. S. Army Electronics Command
Fort Monmouth, New Jersey 07703

Los Alamos Scientific Laboratory
Attn: Reports Library
P. O. Box 1663
Los Alamos, New Mexico 87544

Librarian
U. S. Naval Electronics Laboratory
San Diego, California 95152 [2]

Lockheed Aircraft Corp.
P. O. Box 504
Sunnyvale, California 94088

Dr. I. R. Mirman
AFSC (SCT)
Andrews Air Force Base, Maryland

Lt. Col. Bernard S. Morgan
Frank J. Seiler Research Laboratory
U. S. Air Force Academy
Colorado Springs, Colorado 80912

Dr. G. J. Murphy
The Technological Institute
Northwestern University
Evanston, Illinois 60201

Mr. Peter Murray
Air Force Avionics Laboratory
Wright-Patterson AFB, Ohio 45433

NASA Lewis Research Center
Attn: Library
21000 Brookpark Road
Cleveland, Ohio 44135

NASA Scientific & Technical
Information Facility
Attn: Acquisitions Branch (S/AK/DL)
P. O. Box 33
College Park, Maryland 20740 [2]

National Science Foundation
Attn: Dr. John R. Lehmann
Division of Engineering
1800 G Street, NW
Washington, D. C. 20550

National Security Agency
Attn: R4 - James Tippet
Office of Research
Fort George C. Meade, Maryland 20755

Naval Air Systems Command
AIR 03
Washington, D. C. 20360 [2]

Naval Electronics Systems Command
ELEX 03
Falls Church, Virginia 22046 [2]

Naval Ordnance Systems Command
ORD 32
Washington, D. C. 20360 [2]

Naval Ordnance Systems Command
SHIP 035
Washington, D. C. 20360

Naval Ship Systems Command
SHIP 031
Washington, D. C. 20360

New York University
College of Engineering
New York, New York 10019

Dr. H. V. Noble
Air Force Avionics Laboratory
Wright-Patterson AFB, Ohio 45433

Office of Deputy Director
(Research and Information Rm. 3D1037)
Department of Defense
The Pentagon
Washington, D. C. 20301

Polytechnic Institute of Brooklyn
55 Johnson Street
Brooklyn, New York 11201
Attn: Mr. Jerome Fox
Research Coordination

RAD (EMLAL-1)
Griffiss AFB, New York 13442
Attn: Documents Library

Raytheon Company
Bedford, Mass. 01730
Attn: Librarian

Lt. Col. J. L. Reeves
AFSC (SCBB)
Andrews Air Force Base, Md. 20331

Dr. A. A. Dougal
Asst. Director of Research
Office of Defense Res. and Eng.
Department of Defense
Washington, D. C. 20301

Research Plans Office
U. S. Army Research Office
3045 Columbia Pike
Arlington, Virginia 22204

Dr. H. Robl, Deputy Chief Scientist
U. S. Army Research Office (Durham)
Durham, North Carolina 27706

Emil Schafer, Head
Electronics Properties Info. Center
Hughes Aircraft Company
Culver City, California 90230

School of Engineering Sciences
Arizona State University
Tempe, Arizona 85281

SAMSO (SMSDI-STINFO)
AF Unit Post Office
Los Angeles, California 90045

SSD (SSTRT/Lt. Starbuck)
AFUPO
Los Angeles, California 90045

Superintendent
U. S. Army Military Academy
West Point, New York 10996

Colonel A. Swan
Aerospace Medical Division
AMD (AMRXI)
Brooks AFB, Texas 78235

Syracuse University
Dept. of Electrical Engineering
Syracuse, New York 13210

University of California
Santa Barbara, California 93106
Attn: Library

University of Calif. at Los Angeles
Dept. of Engineering
Los Angeles, California 90024

University of Michigan
Electrical Engineering Dept.
Ann Arbor, Michigan 48104

U. S. Army Munitions Command
Attn: Technical Information Branch
Picatinney Arsenal
Dover, New Jersey 07801

U. S. Army Research Office
Attn: Physical Sciences Division
3045 Columbia Pike
Arlington, Virginia 22204

U. S. Atomic Energy Commission
Division of Technical Information Ext.
P. O. Box 62
Oak Ridge, Tenn. 37831

Dept. of Electrical Engineering
Texas Technological College
Lubbock, Texas 79409

U. S. Naval Weapons Laboratory
Dahlgren, Virginia 22448

Major Charles Waespy
Technical Division
Deputy for Technology
Space Systems Division, AFSC
Los Angeles, California 90045

The Walter Reed Institute of Research
Walter Reed Medical Center
Washington, D. C. 20012

AFSC (SCTR)
Andrews Air Force Base
Maryland 20331

Weapons Systems Test Division
Naval Air Test Center
Patuxtent River, Maryland 20670
Attn: Library

Weapons Systems Evaluation Group
Attn: Col. Daniel W. McElwee
Department of Defense
Washington, D. C. 20305

Yale University
Engineering Department
New Haven, Connecticut 06720

Mr. Charles F. Yost
Special Asst. to the Director of Research
NASA
Washington, D. C. 20546

Dr. Leo Young
Stanford Research Institute
Menlo Park, California 94025

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Division of Engineering and Applied Physics Harvard University Cambridge, Massachusetts	Unclassified
	2b. GROUP

3. REPORT TITLE

THE MATRIX ALGEBRA PROGRAM - A CONVERSATIONAL LANGUAGE FOR NUMERICAL MATRIX OPERATIONS - PART II: REFERENCE MANUAL

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Interim Technical Report

5. AUTHOR(S) (First name, middle initial, last name)

P. M. Newbold

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1968	150	7

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
N00014-67-A-0298-0006 and b. PROJECT NO. NASA NGR 22-007-068	Technical Report No. 562
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Office of Naval Research

13. ABSTRACT

This report is Part II of a two-part description of a new programming language MAP. The language is in a conversational mode, created expressly for direct-access time-sharing computer systems. It is designed to execute numerical matrix operations with the same ease and flexibility as scalar operations. No knowledge of any other language is required.

Part II is the Reference Manual for the language.

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Direct-access computer language Conversational programming language Numerical matrix operations						