1-1-1975

The Lehigh University IBM 360 assembler.

Henry P. Seager

THE LEHIGH UNIVERSITY IBM 360 ASSEMBLER

by

Henry P. Seager

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Electrical Engineering

Lehigh University

1975

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

May 9, 1975
(date)

Professor in Charge

Chairman of Department

Acknowledgments

The author would like to express his gratitude to his advisor, Professor Peggy A. Ota. Her comments and suggestions were a valuable aid in preparing this thesis.

The author also wishes to thank Mr. W. R. Scheffley, Manager, System Operating Department, Pennsylvania Power & Light Co., and Mr. J. R. Evans, Supervisor, System Operating Computer Systems, Pennsylvania Power & Light Co., for their encouragement and understanding during the preparation of this thesis.

Table of Contents

List of Figures

Contents of the Flowchart

Abstract

The Lehigh University IBM 360 Assembler (LUIAS) is a batch mode program which enables a user to assemble an IBM 360 assembly language program on a Control Data 6400 Computer System. LUIAS assembles all but seven of the instructions in the IBM System/360 standard instruction set, 18 extended mnemonics, and 10 pseudo-instructions.

The assembler will generate the appropriate machine object code (up to 25,000 bytes), a symbol table (up to 100 symbols), a literal table (up to 50 literals), and a base table.

The assembler will attempt to assemble any source card input but will abort the assembly under 4 conditions, (core overflow, symbol table overflow, literal table overflow, or no end card encountered). Under such a condition the program will print an appropriate error and exit. If an abort condition does not occur, the assembler will print the source card images, appropriate machine code, and syntax errors encountered. Following the assembly, the assembler will print the meaning of errors encountered, the symbol table, the base table and the literal table. The generated object code is passed to a random access file for later use.

I Introduction

1.1 Background

This thesis describes an IBM System/360 assembler written for a Control Data 6400 Computer System. Using this program it is possible to obtain IBM System/360 machine code output outside of an IBM computing environment.

At the present time, Lehigh University offers a senior level elective in systems programming, E.E.315. The text used for this course is Systems Programming by John J. Donovan. The examples in this book concerned with assembly language programming are written for an IBM System/360 system. It would be desirable for the students at Lehigh University to be able to verify these examples, and also write programs in IBM Systems/360 Basic Assembly Language and have them executed. A set of programs would accomplish this end. The first of these programs would assemble the students program on a CDC 6400 Computer, the only large scale computer system presently available at Lehigh University. A second program would take the machine language assembled output and execute it. This second program The Lehigh University IBM Simulator [1] (LUIS) is available at the Lehigh University Computer Center.

This thesis describes the first program, the Lehigh University IBM 360 Assembler (LUIAS). LUIAS is a batch mode program which assembles all but seven of the instructions in the IBM System/360 standard instruction set, [2] 18 mnemonics, and 10 pseudo-instructions. The program is available through the Lehigh University Computing Center.

LUIAS can assemble up to ten thousand bytes of machine code which is stored in the common block "INFO" (Fig. 1.1). LUIAS also creates a symbol table of up to 100 symbols which is stored in the common block "SYMINFO" (Fig. 1.2). In addition LUIAS creates a literal table of maximum length fifty stored in the common block "LITINFO" (Fig. 1.3) and a base table stored in the common block "BASINFO" (Fig. 1.4).

LUIAS was written to be compatible with a second program which will execute the assembled machine code. This second program must execute the assembled code within the constraints of a CDC 6400 Computer System. Thus, the normal methods of machine control including I/O and interrupts cannot be used. For this reason, the seven control instructions (Diagnose, Set System Mask, Load PSW, Halt I/O, Supervisor Call, Test Channel, and Test I/O) has been omitted from the standard instruction set.* A modified form of the SIO instruction has been

included which is compatible with the program which will execute the machine code.

LUIAS is a basic two pass assembler. The main purpose of pass 1 is to create a symbol table and a literal table for use in symbolic referencing. This pass must constantly calculate the instruction location counter (ILC) so that the proper values for the symbols and literals can be generated. Pass 2 is concerned with generating the actual machine code using the symbol and literal tables extensively to generate the instruction fields. Details of the decisions made in the two passes are contained in Chapter IV.

LUIAS is powerful enough to generate machine code to perform any logical function. However, to duplicate every feature available on a real IBM System/360 Assembler would have been a tremendous task. Certain simplifying assumptions were made to make the machine code output (byte oriented) compatible with the word length of the CDC 6400 computer (60 bits). These assumptions restrict the programmer in the use of evaluating arithmetic expressions and in declaring arbitrary character lengths. Still, within these constraints, the programmer has lost no basic tool necessary to write an assembly language program. The constraints imposed on the programmer are explained in Section 2.3. Suggestions for improving the constraints

are contained in Chapter VI.

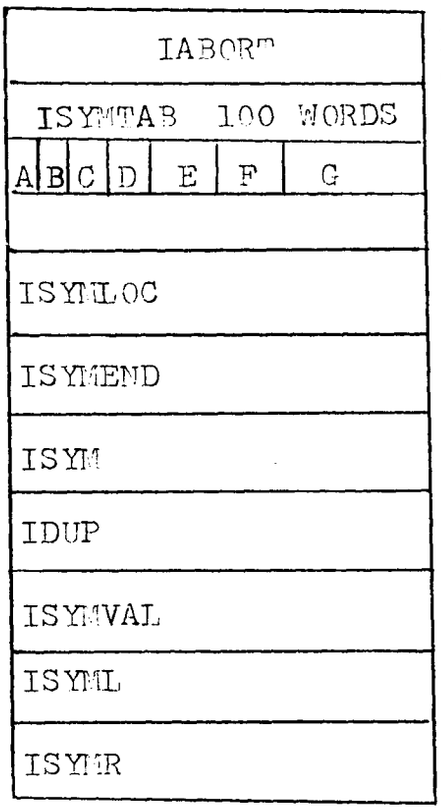
1.2 Contents of the Thesis

The remaining portion of this thesis is divided into five chapters and three appendices. Chapter II is a user's guide for using the Lehigh University IBM 360 Assembler. Chapter III gives a short description of each of the five routines which comprise the assembler. Chapter IV gives a more detailed description of the subroutine ASSEMBL which actually assembles the machine code. The last chapter, Chapter VI, describes how additional capabilities can be added to LUIAS.

Appendix A contains material (reproduced from the manual, IBM System/360 Principles of Operation) which should be of value to those who are unfamiliar with the instructions of the IBM System/360 Appendix B contains a typical output from LUIAS which contains no assembly errors. Appendix C contains an output from LUIAS which contains all possible (16) assembly errors.

ICONT 2500 WORDS	The assembled machine code
ICARD 8 WORDS	The current source card being analyzed
IERR	The syntax error number on the card (1-16)
ILOC	The current location counter
IOLOC	The previous location counter
ICODE 2 WORDS	The assembled code for the current output line
IEND	If equal to a 1, signifies an end card has been encountered
IPROC	If not equal to a zero, signifies there is more processing on the current card
IPRINT	Determines type of listing printing card (1-6)
IPASS	If equal to a 1, signifies pass 2
IDEL	Spare
ILABL	If equal to a 1, signifies there is a label on the card
ICHAR	Pointer to current character being analyzed (0-71)
IERSEV	Spare
IWORD	Pointer to active word within ICONT (0-2499)
IOUT	Pointer to active word within ICODE (0-1)
IWSHIFT	Pointer to active bit within ICONT active word (54-12)
IOSHIFT	Pointer to active bit within active word of ICODE (54-0)
ISTART	Contains program starting byte address

Fig. 1.1 The "INFO" Common Block



If not equal to zero, an abort condition was found in pass 1

The symbol table

Current pointer into ISYMTAB to extract an entry for printing (0-99)

If equal to a 1, the end of the table has been encountered during printing

Current symbol for printing

If equal 1, the current symbol is duplicated

Current symbol value, right justified

Contains symbol length in bytes (1-6)

If equal 1, current symbol is relocatable

A. If equal 1, entry is in use (1 bit)

B. If equal 1, entry is duplicated (1 bit)

C. If equal 1, entry is absolute (1 bit)

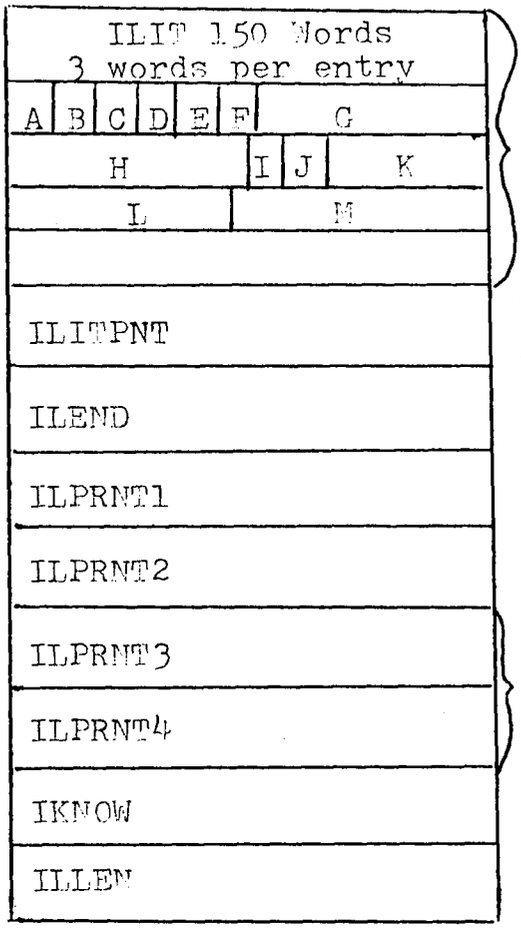
D. Spare (4 bits)

E. Symbol length in bytes (3 bits)

F. Symbol value in binary (20 bits)

G. The symbol in display code (30 bits)

Fig. 1.2 The "SYMINFO" Common Block



The literal table

Pointer to entry in ILIT to be removed for printing

If equal to a 1, end of table found

Current literal in display code

Current literal core address

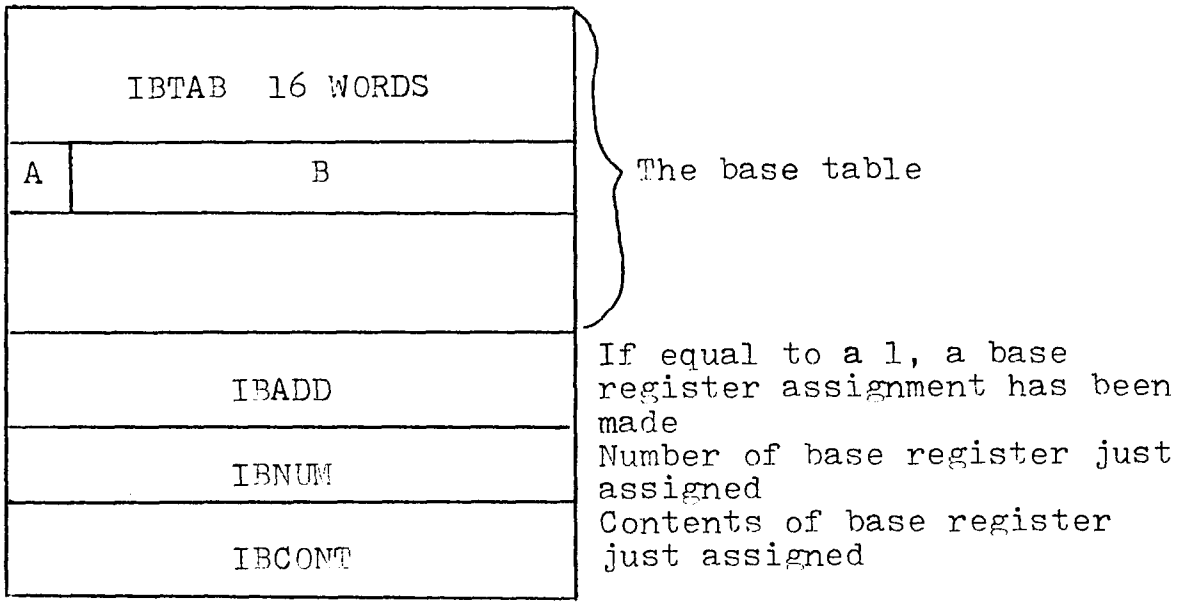
Contents of literal core address in binary

If equal 0, contents of literal core address are not known

Length of literal in bytes

A. If equal 1, entry is in use (1 bit)
B. If equal 1, entry in previous litorg block (1 bit)
C. If equal 1, end of this litorg block (1 bit)
D. If equal 1, this entry has been printed and cannot be used for backward references (1 bit)
E. The core contents of the literal are known if equal to 1 (1 bit)
F. Spare (1 bit)
G. The literal in display code (54 bits)
H. Spare (36 bits)
I. Literal boundary 0=byte, 1=halfword, 2=fullword, 3=doubleword (2 bits)
J. The length of the literal in bytes (4 bits)
K. The core address of the literal (20 bits)
L. Spare (12 bits)
M. Core contents of the literal location (48 bits)

Fig. 1.3 The "LITINFO" Common Block



A. If equal to a 1, this base register is available (1 bit)

B. Contents of base register (59 bits)

Relative position in table (0-15) determines base register number.

Fig. 1.4 The "BASINFO" Common Block

II User's Guide to LUIAS

2.1 Introduction

LUIAS takes a program written in IBM System/360 assembly language and converts it into IBM System/360 machine language with appropriate listings. LUIAS assembles all but seven of the instructions in IBM System/360 standard instruction set (Diagnose, Set System Mask, Load PSW, Halt I/O, Supervisor Call, Test Channel and Test I/O are omitted), 18 extended mnemonics, and 10 pseudo-ops. LUIAS outputs the assembled machine language to the file PGM. LUIAS will also output to the line printer an assembly listing which includes the card source image, current location counter, machine code, an error code, and the source card number. At the end of the listing the total number of errors is printed followed by the meaning of each error. Next is printed the program first word address, next available address and program length. Following are printed the actual object code passed to the file. PGM, the symbol table, the literal table, and the base table, Appendices B and C contain typical outputs of LUIAS.

This chapter describes the features and use of LUIAS. Section 2 describes the abort conditions. Section 3

describes the general format for source card input to LUIAS. Section 4 discusses the instruction set, Section 5, the extended mnemonics and Section 6, the pseudo-instructions. Section 7 states the job control language necessary to execute LUIAS on the CDC 6400.

2.2 Abort Conditions

LUIAS will attempt to assemble whatever source cards are presented. However, 4 conditions will cause LUIAS to abort during pass 1 and produce no output:

1. No END card is encountered.
2. Greater than 10,000 bytes of object code are to be generated.
3. More than 100 symbols are defined.
4. More than 50 literals are defined.

In all cases, an abort message will be printed.

2.3 General Format

1. LUIAS accepts source statements in a free format.
2. A comment card must have an asterick in column 1.
3. A label must begin in column 1 and must begin with an alpha character. The rest of the label can be alpha numeric and can be up to 5 characters long. If it is more than 5 characters long, only the first five are used.

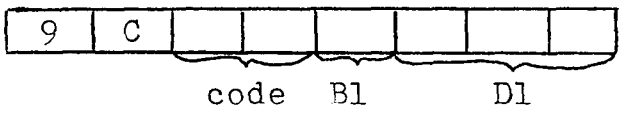
4. A comment can appear on any card providing it is preceded by at least one blank following the operand field.
5. Card columns 73-80 are ignored and used for sequencing or identification only.
6. A constant, whether defined as a literal or in a DC statement can be at maximum 6 characters long. e.g. X(0004) is valid, X(0145764) is invalid.
7. The length parameter in a DC statement is not implemented.
8. A label multiplier in any statement is not permitted; e.g., DC ALPHA P(1).
9. Left and right parentheses replace the quotes when defining a constant, e.g. F(1) instead of F'1'. The quote does not exist in the CDC 6400 character set.
10. The maximum length of any constant or literal is 6. Therefore, in a storage to storage instruction, explicit declarations must be used if the implied length is greater than 6.
11. In any operand field, an expression to be evaluated can contain fixed point numbers only. e.g. LABEL+3 is valid while LABEL+X(3) is invalid.
12. Imbedded blanks in any operand are invalid.

2.4 The Instruction Set

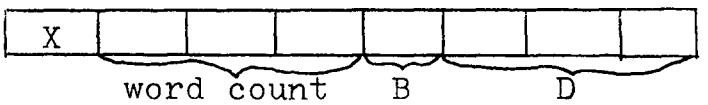
Eighty (80) instructions are implemented and all follow the standard IBM assembly language format (except for the SIO instruction) with the exceptions as noted in Section 2.3. The SIO instruction has the format:

```
     LABEL1     SIO     R,LABEL2
     LABEL1     SIO     R,D1(B1)  for a read operation
and  LABEL1     SIO     W,LABEL2
     LABEL1     SIO     W,D1(B1)  for a write operation.
```

The SIO instruction will be assembled into the following format:



Where the code is 01 for writing and 00 for reading. The address specified by B1 and D1 is a fullword. This fullword contains the count of the number of words to be read into memory or output from memory and the starting address of the user's buffer area according to the following format:



Where B and D specify the starting address (See Section 2.6). This form of the SIO instruction is necessary for the companion program, LUIS which will execute the assembled code.

2.5 The Extended Mnemonics

All 18 extended mnemonics are implemented and all follow the standard IBM assembly language format with the exceptions as noted in Section 2.3.

2.6 The Pseudo Instructions

The assembler will process 10 pseudo-ops. Nine (9) are standard IBM pseudo-ops. The 10th is a new pseudo-op created for ease in doing I/O.

1. START — The operand field may be blank or a fixed point number.
2. ORG — The operand field may be blank or a fixed point number.
3. USING — Any number of registers can be defined in one statement.
4. DROP — Only one register can be dropped per statement.
5. LTORG
6. END
7. EQU
8. DC — Only one type of constant can be defined per statement. Multiple different constants of the same type can be defined by one statement. If a multiplier is used, it must be a fixed point number and then only one constant can be defined by that statement.

```
e.g.  DC F(1,2,56)
      DC 23F(126)
```

9. DC — Only one type of constant can be defined per statement. A fixed point multiplier is optional.

```
e.g.  DS F
      DS 100F
```

10. IO — This is a new pseudo-op created for ease in doing I/O. It has the format:

```
    LABEL2  IO  N,LABEL3
 or LABEL2  IO  N,D1(B1)
```

where N is a fixed point number, the word count to be used in the I/O opera-

tion, and LABEL3 points to the user's buffer address to be used in the I/O operation. The IO pseudo-op aligns itself on a full word boundary (a requirement of the SIO instruction). However, it is the users' responsibility to insure that LABEL3 points to a full word boundary (another requirement of the SIO Instruction).

2.7 Instructions Necessary to Execute LUIAS

The following minimum job control language is necessary to successfully execute LUIAS:

```
JOB CARD
ATTACH(LUIAS,ID=HPS)
LUIAS
CATALOG(PGM,XXXX,ID=YYY)*
7-8-9 card
Source Statements
EOF
```

*XXXX and YYY are any alpha characters as determined by the user which identify the file which contains the object code output for later use by the program LUIS.

III The Routines of LUIAS

3.1 Introduction

The Lehigh University IBM 360 Assembler is written in Compass[3,4] assembly language and Fortran. The program consists of a main routine written in Fortran and four subroutines written in Compass. The Compass subroutines assume that the main routine is compiled using the RUN compiler. Approximately 27.1 CP seconds are needed to compile and assemble the program. 32462 octal words of central memory are required by the program and the various system routines which it calls. Additional central memory is needed for the loader and the loader tables. Section 2 gives a general description of each of the routines of LUIAS.

3.2 Description of the Routines

3.2.1 The Main Routine ASSMB

The main routine is written in Fortran. All I/O communications are handled by this routine. During pass 1, this routine is responsible for reading cards from the card reader, passing the source code to the subroutine ASSMBL for analysis, saving the source on mass storage for pass 2, and printing any abort conditions. During pass 2, this routine will again pass the source code to the subroutine ASSMBL for further analysis, print the assembled machine code, and pass the final output code to mass storage for later use. At the conclusion of the assembly, this routine is also responsible for printing the symbol table, the base table and the literal table. (See flowchart pp. 27-31).

3.2.2 Subroutine ASSMBL

Subroutine ASSMBL is written in COMPASS. This subroutine is the routine which actually assembles the source statements. This routine is responsible for generating symbols and literals during pass 1, and for updating the location counter. When an END statement is encountered this routine will pass a flag to the routine ASSMB to signify the end of pass 1. Abort conditions are also determined by this routine during pass 1. During pass 2, ASSMBL is responsible for forming

the base table and generating the actual machine code. As each line of code is generated, it is passed along with any error conditions encountered to the routine ASSMBL for printing. When the END statement is encountered during pass 2, ASSEMBL will generate any literals necessary and pass them to ASSEMB for printing. This signifies the end of the assembly. (See flowchart pp. 32-94).

3.2.3 Subroutine CONVERT

Subroutine CONVERT is written in COMPASS. This subroutine will convert one word passed to it to the equivalent display code representation. The passed word contains ten right justified hex digits. The converted word contains ten display code digits, the CDC 6400 representation of the hex digits pass (See flowchart pp. 94).

3.2.4 Subroutine GETSYM

Subroutine GETSYM is written in COMPASS. This subroutine is responsible for removing an entry from the symbol table and formating it for printing. All parameters are passed via the common block "SYMINFO" (Fig. 1.2). Everytime this routine is called it will remove and format the next entry. A flag is set to signify the end of table has been encountered. (See flowchart pp. 95).

3.2.5 Subroutine LITPRIN

Subroutine LITPRIN is written in COMPASS. This subroutine is responsible for removing an entry from the literal table and formating it for printing. All parameters are passed via the common block "LITINFO" (Fig. 1.3). Everytime this routine is called it will remove and format the next entry. A flag is set to signify the end of table has been encountered. (See flowchart pp. 96).

IV ASSEMBL

4.1 Introduction

Subroutine ASSEMBL is the heart of the Lehigh University IBM 360 Assembler. ASSEMBL is the routine which actually assembles the source statements, generates machine code, determines abort and syntax errors, and creates all tables necessary for the assembly. Section 2 describes the action taken by ASSEMBL during pass 1 of the assembly. Section 3 describes how the actual machine code is generated during pass 2.

4.2 Pass 1

During pass 1, the main task of ASSEMBL is to create the symbol and literal tables. A number of secondary functions are performed while accomplishing this end.

As each source card is passed to ASSEMBL, a check is made for an asterick in card column 1. If it is found, no further action is taken since the card is a comment card. Otherwise, ASSEMBL looks for a valid label. If one is found, it is saved for possible later insertion into the symbol table.

Next, a check is made for a valid pseudo-op. If one is found, a branch is taken to the proper processor. Some pseudo-ops require no processing at this time (DROP, USING), while others require updating the location counter only (START,ORG). Other types (DC,DS,IO) require evaluating the operand expression, while another group (LTORG,END) require resolving addresses for the literal table. The EQU pseudo-op requires evaluation of the operand and making an entry in the symbol table. At this point, if a label had been found, an entry is made into the symbol table.

It is during this pass that a check is made for a table overflow (machine code output, literal, or symbol table). If an overflow occurs, a flag is set and a return is made to the main program (ASSEMB) to abort

the assembly.

If a pseudo-op had not been found, a valid machine operation or extended mnemonic is searched for. If one is found, it is only necessary to update the instruction location counter (ILC) at this time.

If no valid operation had been found on the card, no action is taken since the card will be flagged to be in error during pass 2.

When the end card is encountered, a flag is set for the main program to signify the end of pass 1.

4.3 Pass 2

As each card image is received, a check is made for an asterick in card column 1. If one is found, return is made to the main program to print the card. If no asterick is found, the label field is analyzed. If an invalid is found, an error flag is set. However, if a valid label is found, a search is made of the symbol table to see if the label is duplicated. If it is duplicated an error flag is set.

Next, the operation field is checked. If a valid pseudo-op is found, a branch is made to the proper processor. The DROP and USING statements require making entries in the base table, while the START and ORG cards require updating the location counter. The pseudo-ops DC,DS,IO require evaluating the operand field and generating the appropriate machine code. LTORG and END cards require generating code for any existing literals, while the EQU statement requires no action. If any error had been found while evaluating the operand field of the pseudo-op card, an error flag is set accordingly.

If no valid pseudo-op has been found a search is made for a valid instruction type or extended mnemonic. If one is found, a branch is made to the appropriate instruction processor. This processor will evaluate the

operand field and insert the proper fields into the instruction machine code. If an error is found during evaluation, an error flag is set accordingly.

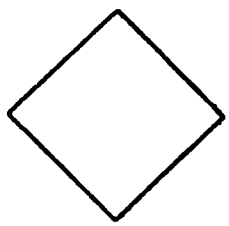
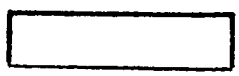
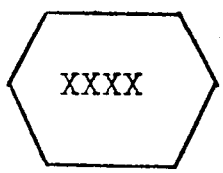
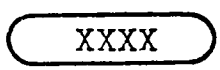
If no valid operation field is found, an error flag is set and return is made to the main program.

For each call made to ASSEMBL, return is made to the main program with 5 variables set in the "INFO" common block (Fig. 1.1). IPRINT contains a number (1-6) which tells the main program how to print the assembled line (e.g. a START card does not require printing any assembled machine code). IERR contains a number (0-16) which defines the error type encountered. ICODE contains the assembled machine code. If the variable IPROC is not equal to a zero, then ASSEMBL has more processing on the same card image. (e.g. a DC statement could generate more than one line to be printed). IEND is set to a 1 to signify the end of the assembly.

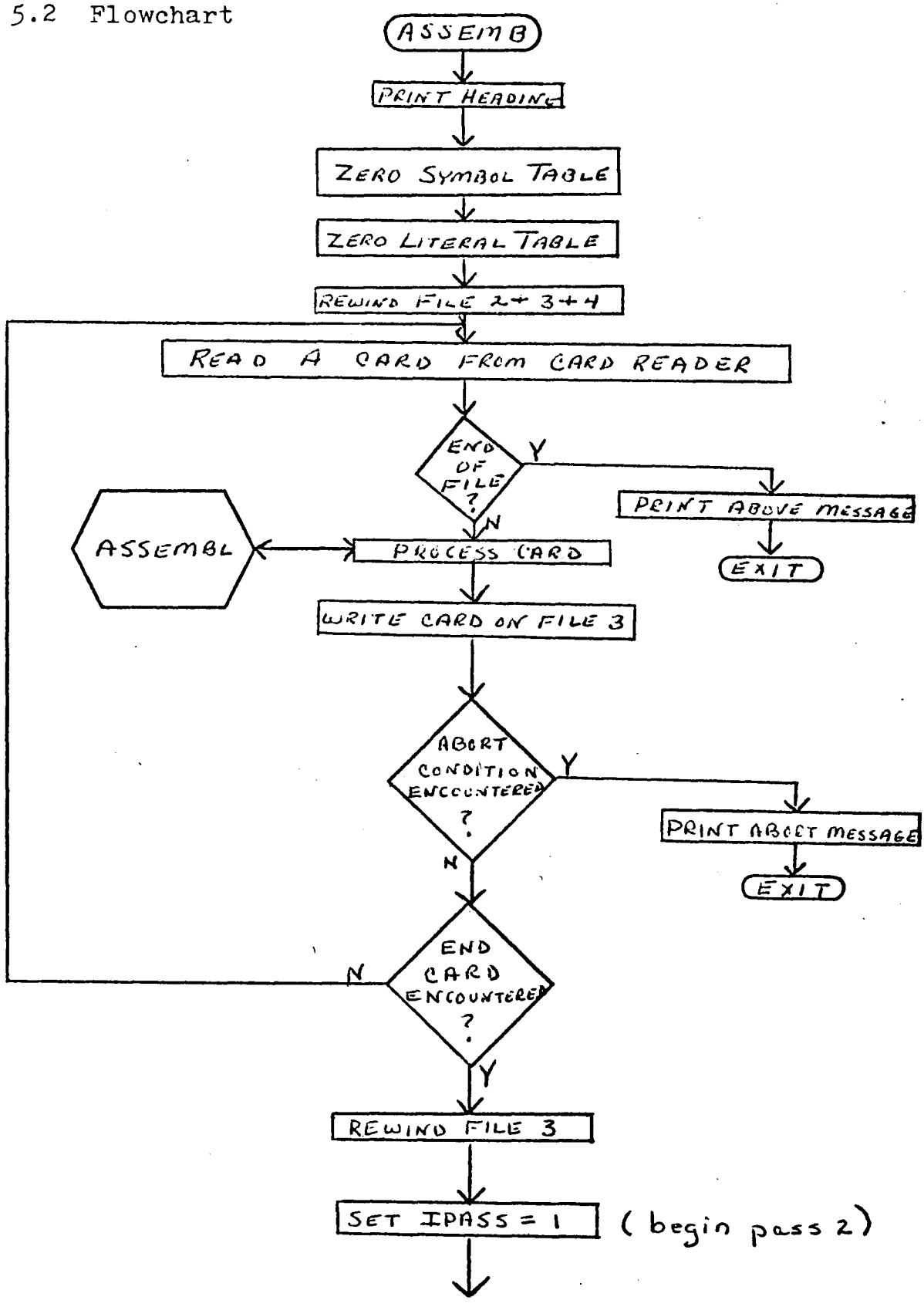
V. The Flowchart of LUIAS

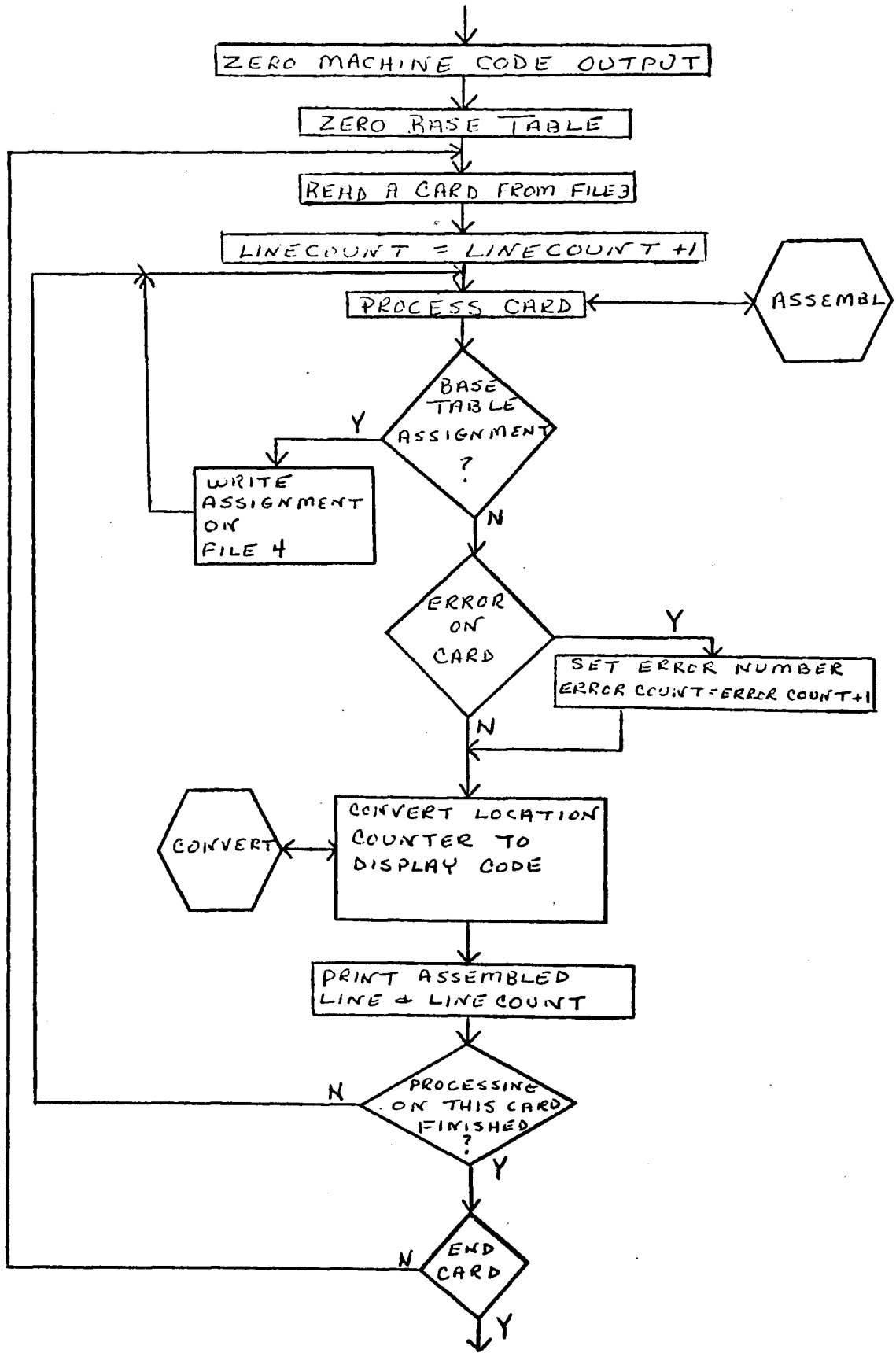
5.1 Notation

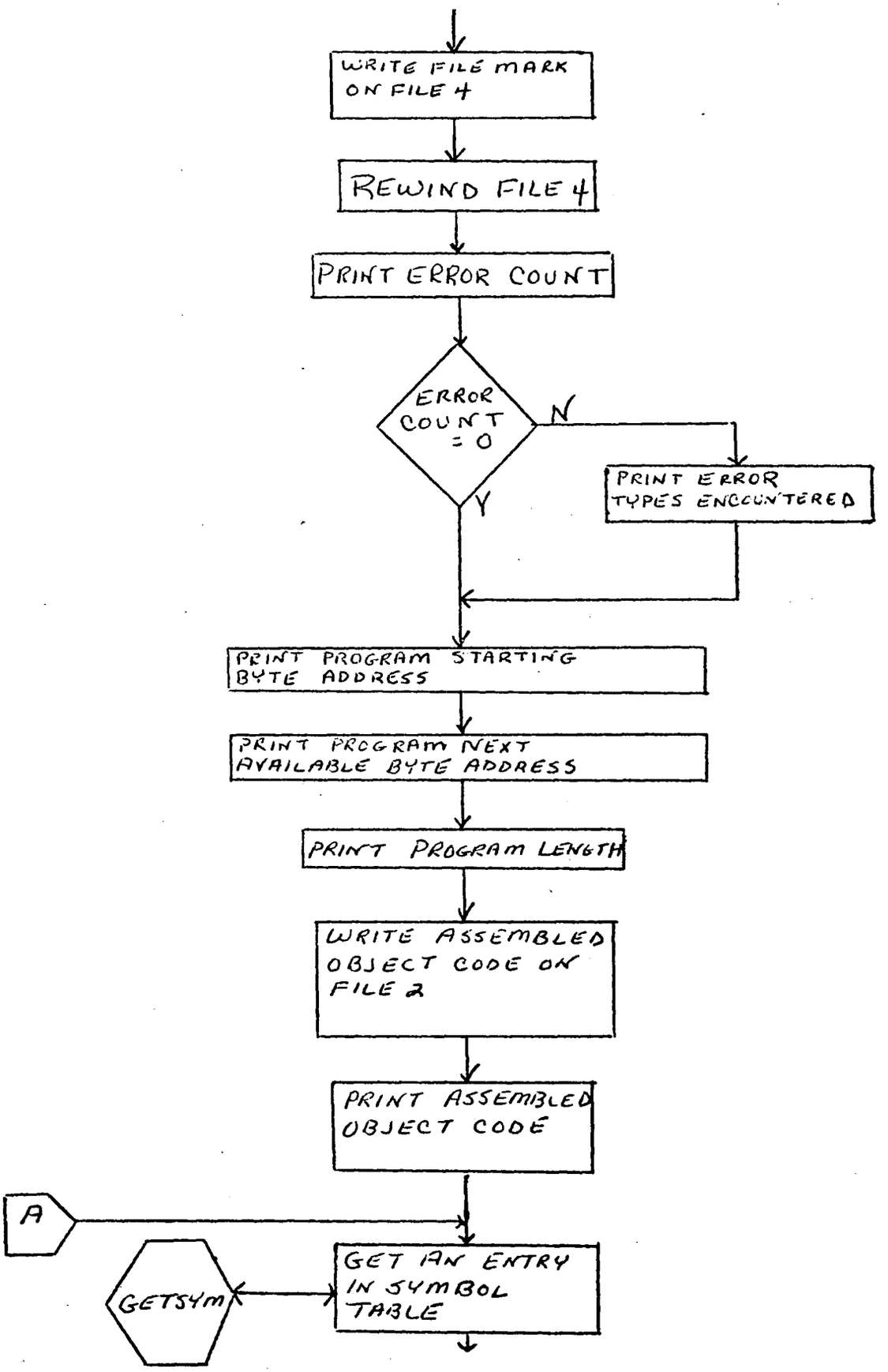
The following notations will be used in the flowchart:

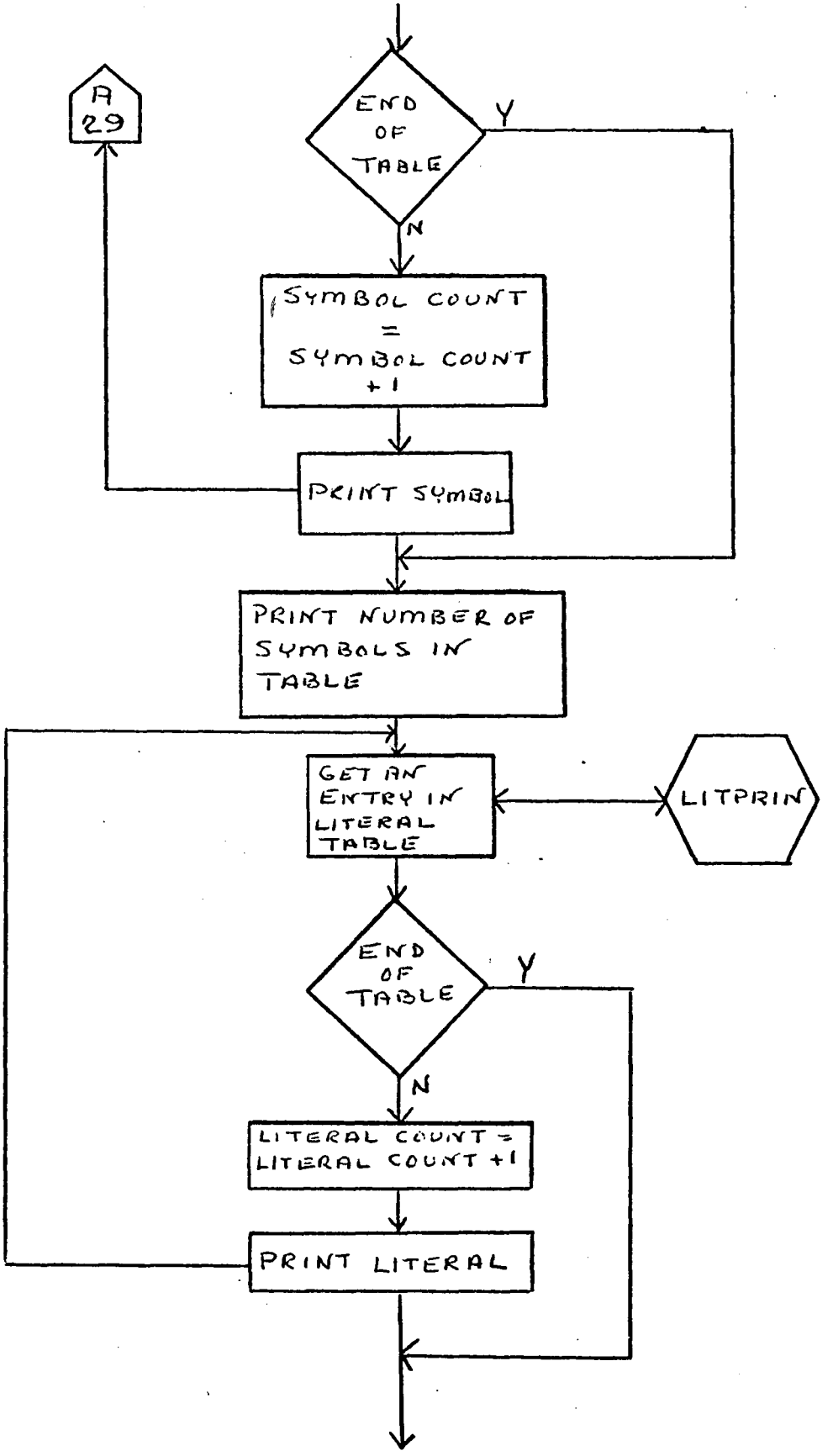
Notation	Meaning
	Decision block
	Processing block
	Closed subroutine where XXXX is the name of the closed subroutine
	Off page connector, which signifies the page where the branch is made to
	On page connector
	Arithmetic sum
	Arithmetic difference
	Entry point of subroutine XXXX
	Ending point of subroutine

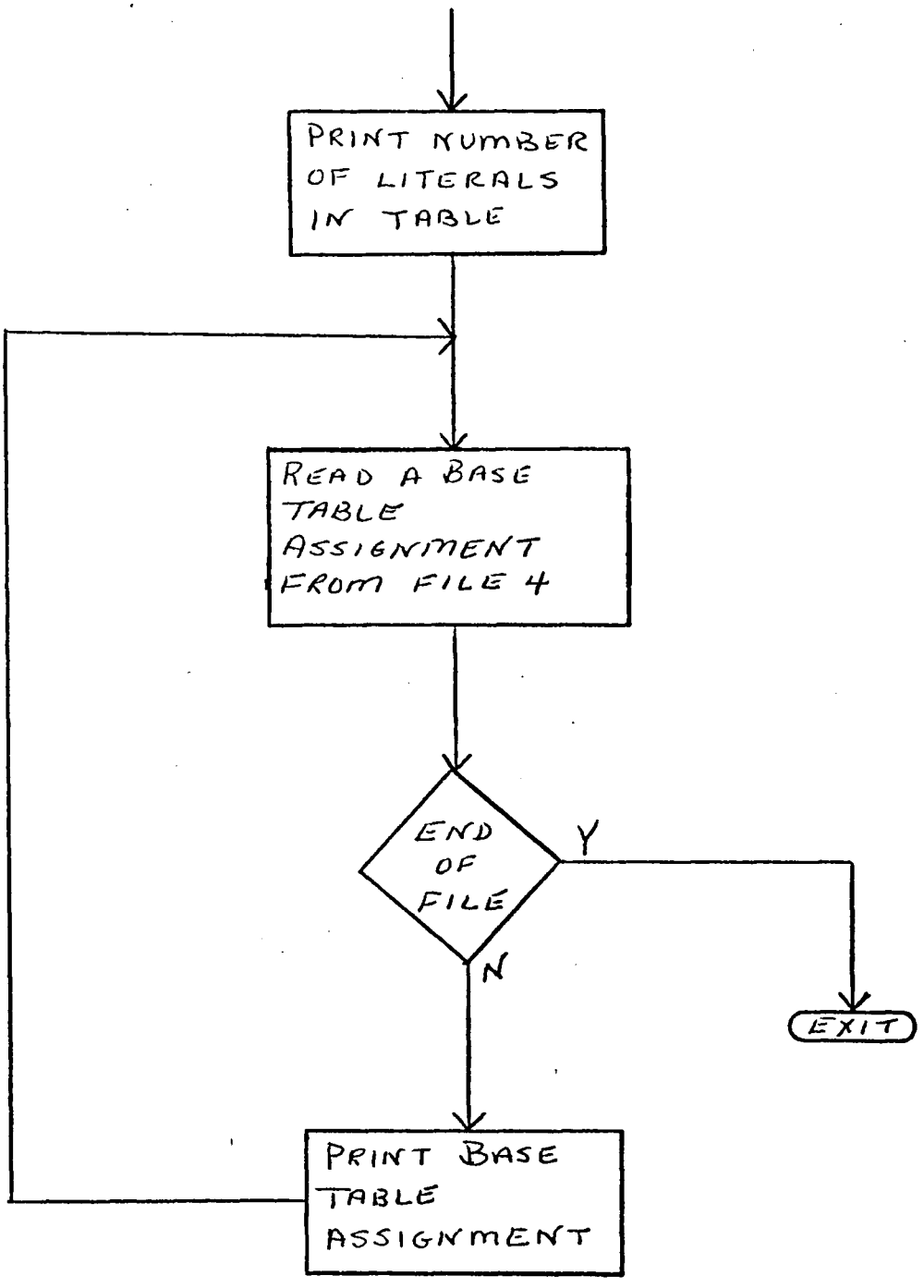
5.2 Flowchart

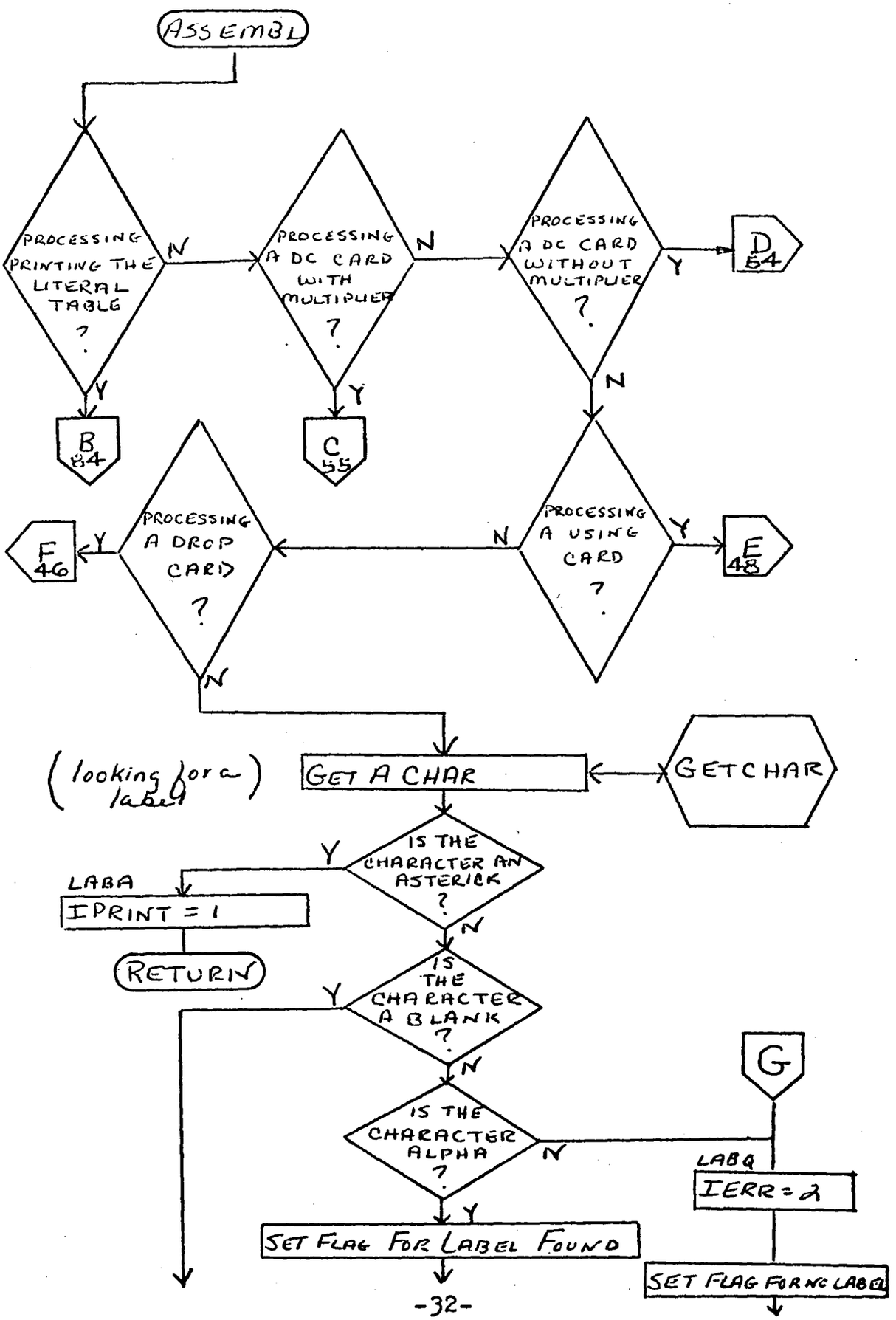


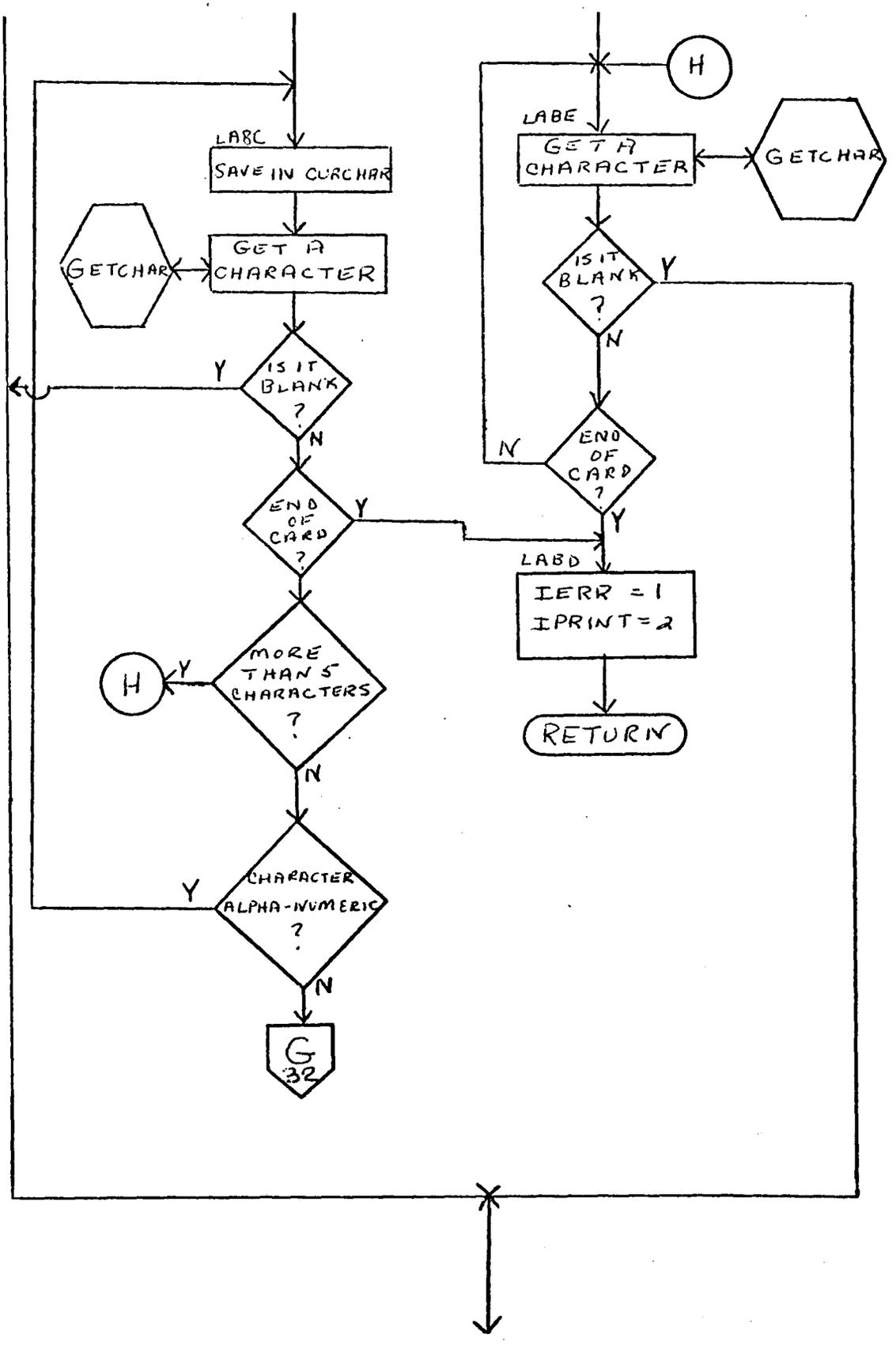

ZERO MACHINE CODE OUTPUT
ZERO BASE TABLE
READ A CARD FROM FILE3
LINECOUNT = LINECOUNT +1
PROCESS CARD
ASSEMBL
BASE TABLE ASSIGNMENT ?
Y
N
WRITE ASSIGNMENT ON FILE 4
ERROR ON CARD
Y
SET ERROR NUMBER
ERROR COUNT=ERROR COUNT+1
N
CONVERT
CONVERT LOCATION COUNTER TO DISPLAY CODE
PRINT ASSEMBLED LINE & LINE COUNT
N
PROCESSING ON THIS CARD FINISHED ?
Y
N
END CARD
Y

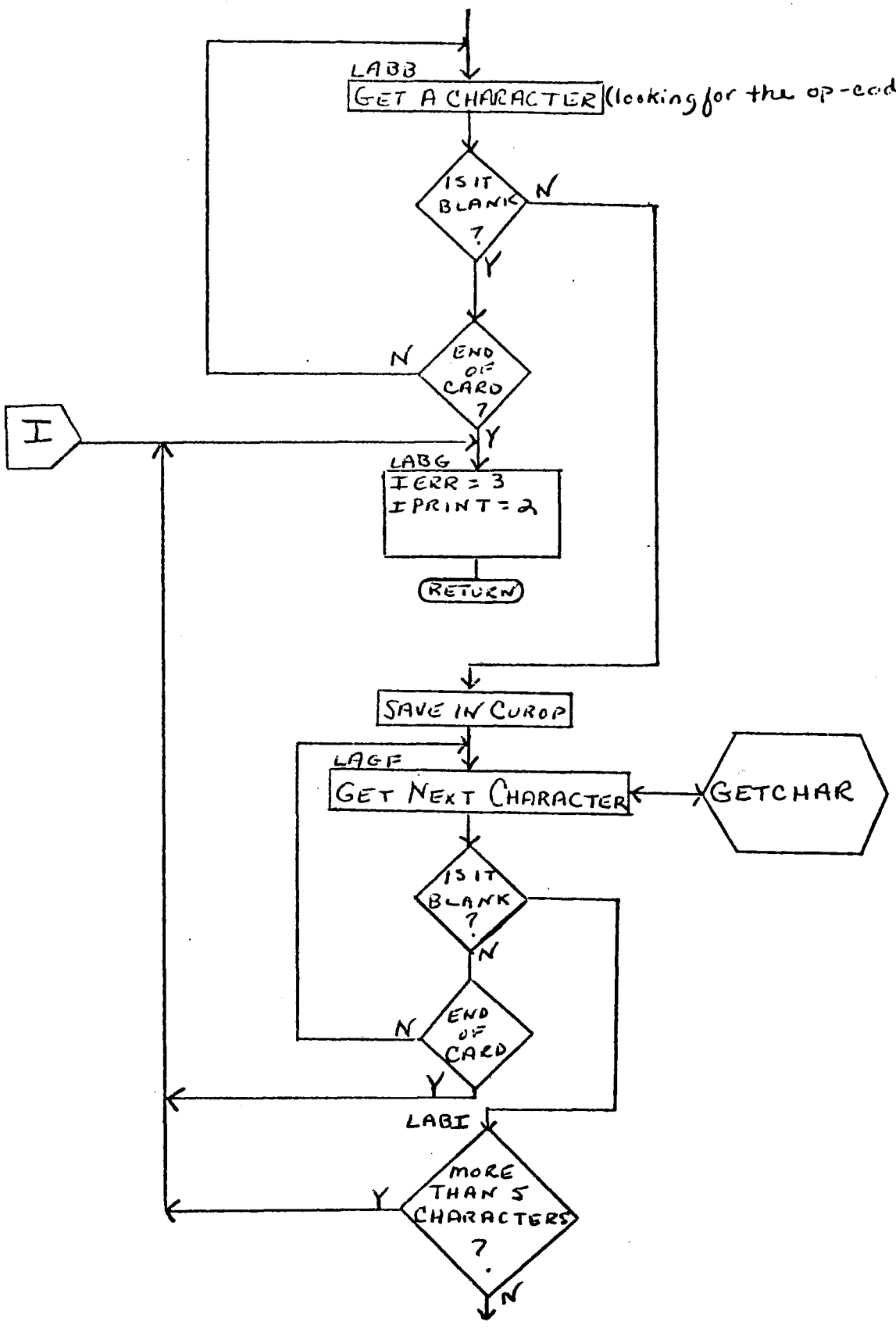

WRITE FILE MARK ON FILE 4
REWIND FILE 4
PRINT ERROR COUNT
ERROR COUNT = 0
N
Y
PRINT ERROR TYPES ENCOUNTERED
PRINT PROGRAM STARTING BYTE ADDRESS
PRINT PROGRAM NEXT AVAILABLE BYTE ADDRESS
PRINT PROGRAM LENGTH
WRITE ASSEMBLED OBJECT CODE ON FILE 2
PRINT ASSEMBLED OBJECT CODE
A
GETSYM
GET AN ENTRY IN SYMBOL TABLE

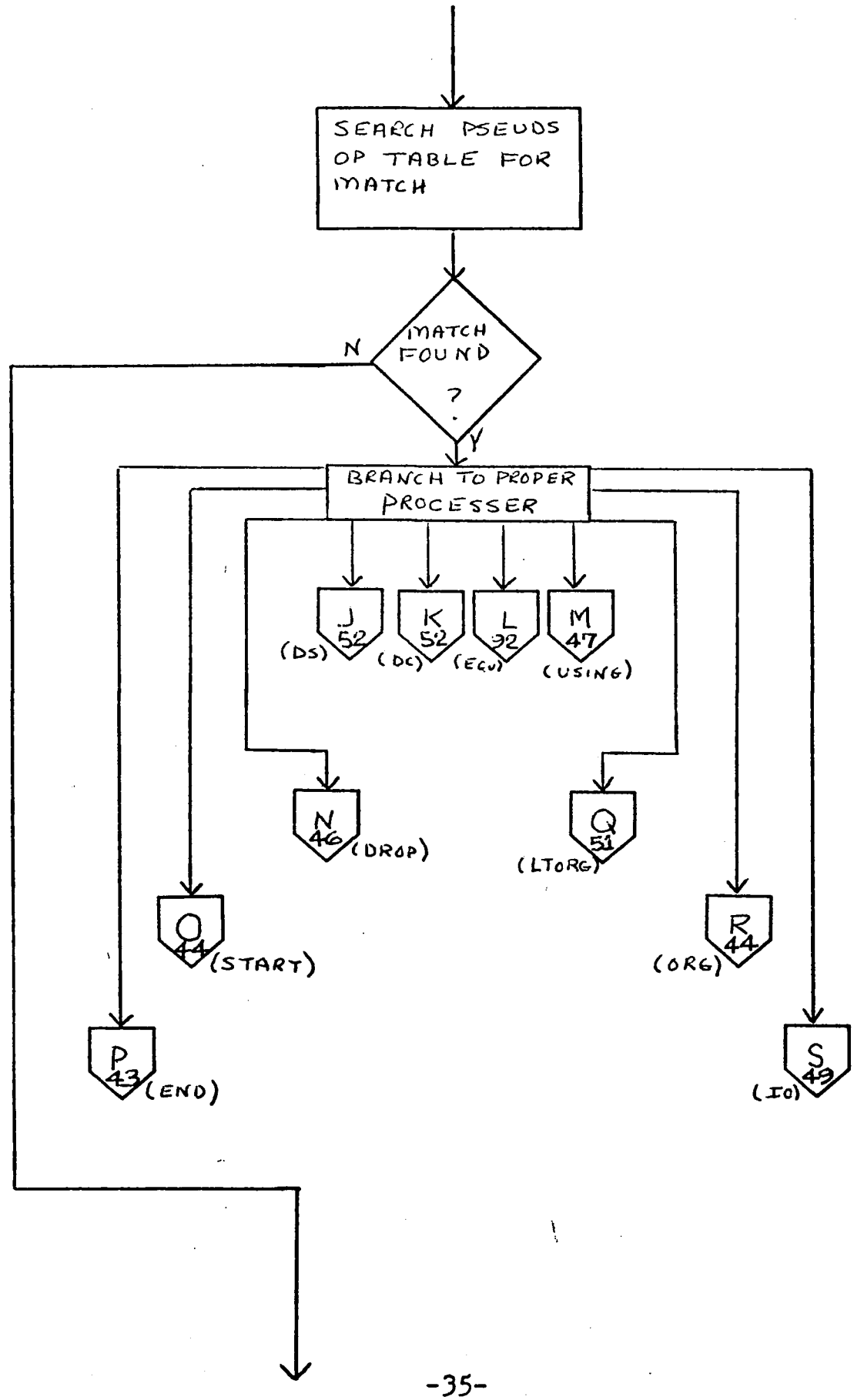

A
29
END
OF
TABLE
Y
N
SYMBOL COUNT
=
SYMBOL COUNT
+1
PRINT SYMBOL
PRINT NUMBER OF
SYMBOLS IN
TABLE
GET AN
ENTRY IN
LITERAL
TABLE
LITPRIN
END
OF
TABLE
Y
N
LITERAL COUNT =
LITERAL COUNT +1
PRINT LITERAL

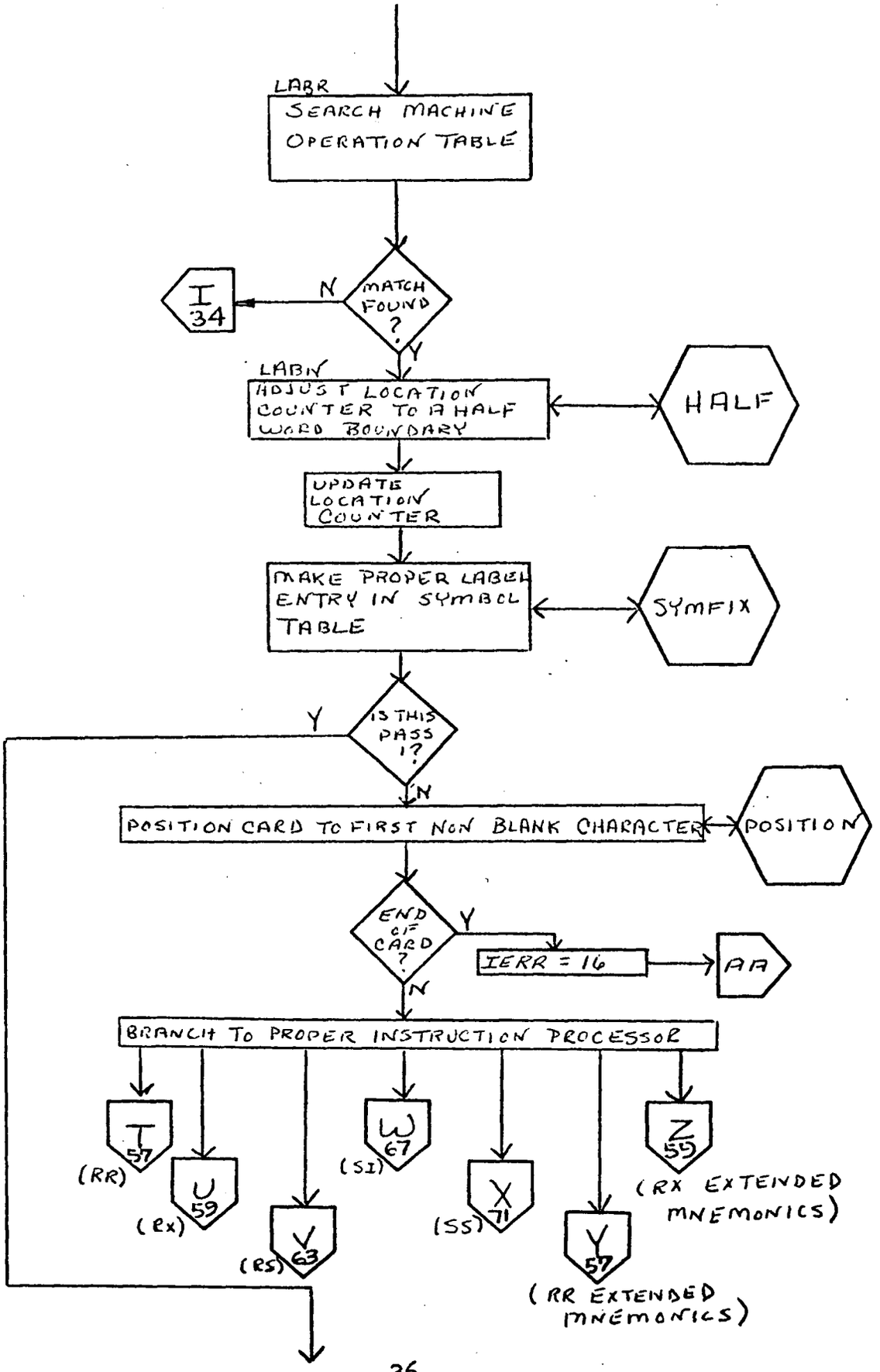

PRINT NUMBER OF LITERALS IN TABLE
READ A BASE TABLE ASSIGNMENT FROM FILE 4
END OF FILE
Y
N
EXIT
PRINT BASE TABLE ASSIGNMENT

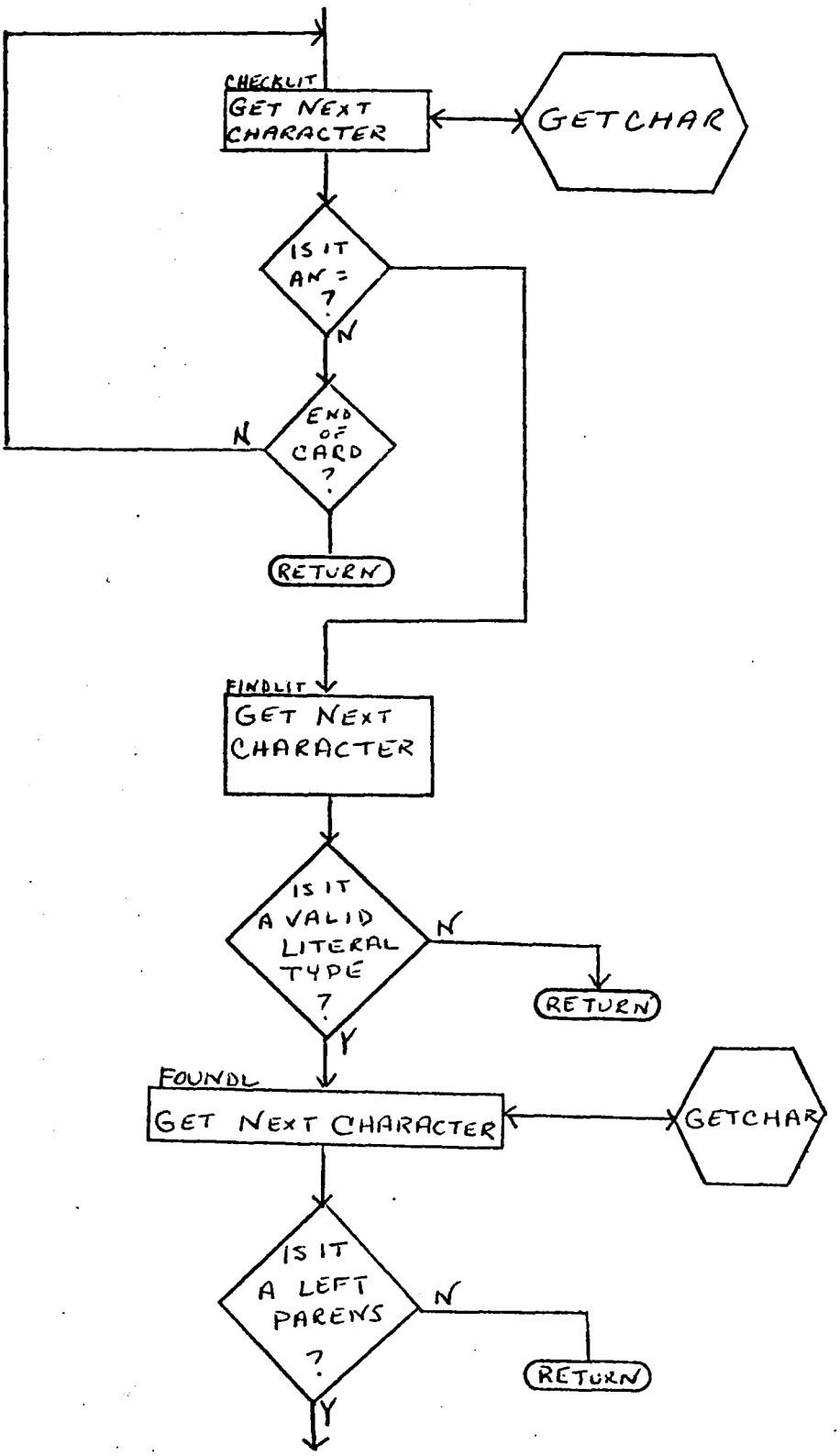

ASSEMBL
PROCESSING PRINTING THE LITERAL TABLE ?
N
PROCESSING A DC CARD WITH MULTIPUER ?
N
PROCESSING A DC CARD WITHOUT MULTIPUER ?
Y
D 54
Y
B 84
Y
C 55
N
PROCESSING A USING CARD ?
Y
E 48
N
PROCESSING A DROP CARD ?
Y
F 46
N
(looking for a label)
GET A CHAR
GETCHAR
IS THE CHARACTER AN ASTERICK ?
Y
LABA
IPRINT = 1
RETURN
N
IS THE CHARACTER A BLANK ?
Y
N
IS THE CHARACTER ALPHA ?
N
G
LABG
IERR = 2
Y
SET FLAG FOR LABEL FOUND
SET FLAG FOR NO LABEL

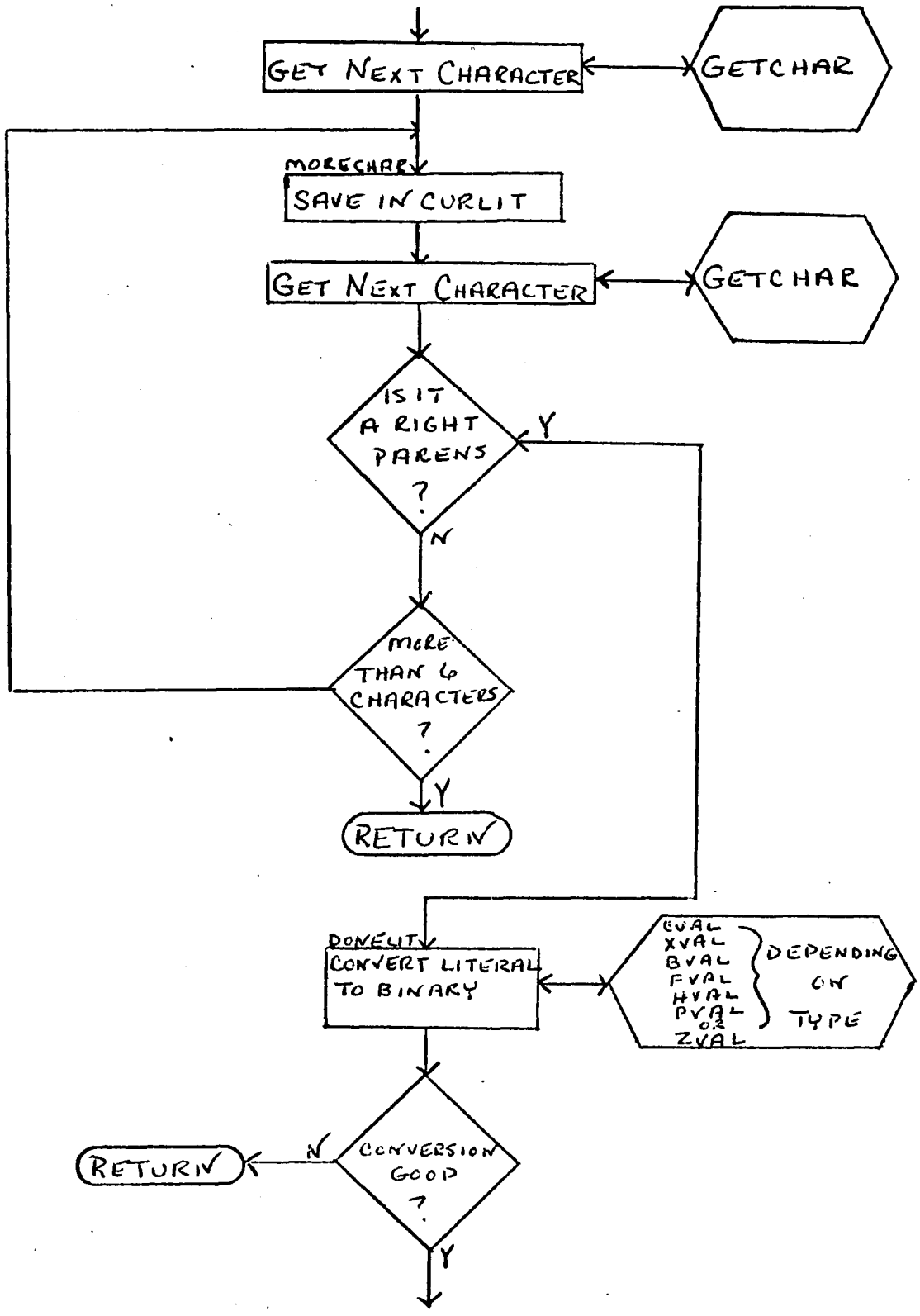

H
LABE
GET A CHARACTER
GETCHAR
IS IT BLANK ?
Y
N
END OF CARD ?
N
Y
LABD
IERR = 1
IPRINT = 2
RETURN
LABC
SAVE IN CURCHAR
GETCHAR
GET A CHARACTER
IS IT BLANK ?
Y
N
END OF CARD ?
Y
H
MORE THAN 5 CHARACTERS ?
Y
N
CHARACTER ALPHA-NUMERIC ?
Y
N
G 32

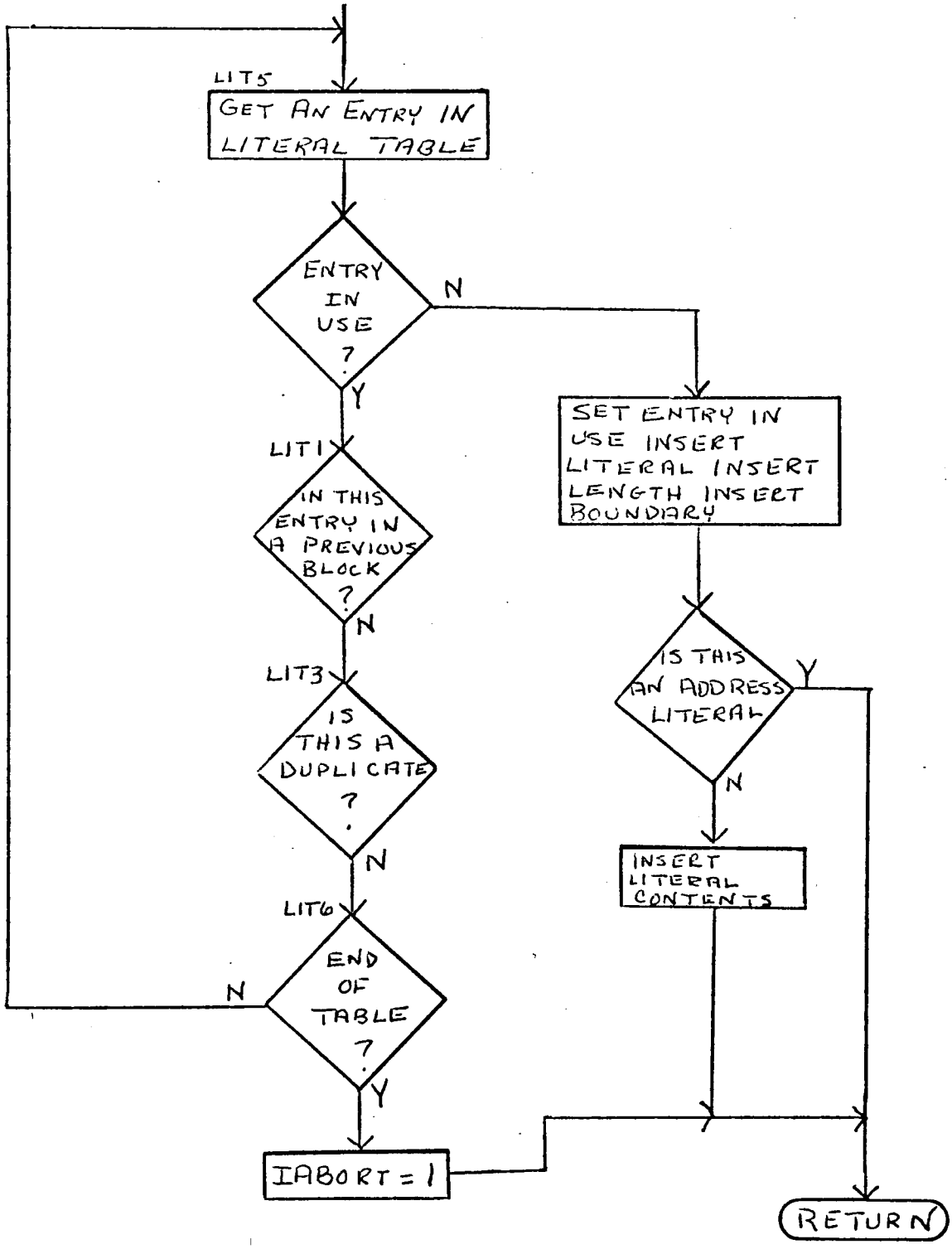

LABB
GET A CHARACTER (looking for the op-code)
IS IT BLANK ?
N
Y
END OF CARD ?
N
Y
I
LABG
IERR = 3
IPRINT = 2
RETURN
SAVE IN CUROP
LAGF
GET NEXT CHARACTER
GETCHAR
IS IT BLANK ?
N
END OF CARD
N
Y
LABI
MORE THAN 5 CHARACTERS ?
Y
N

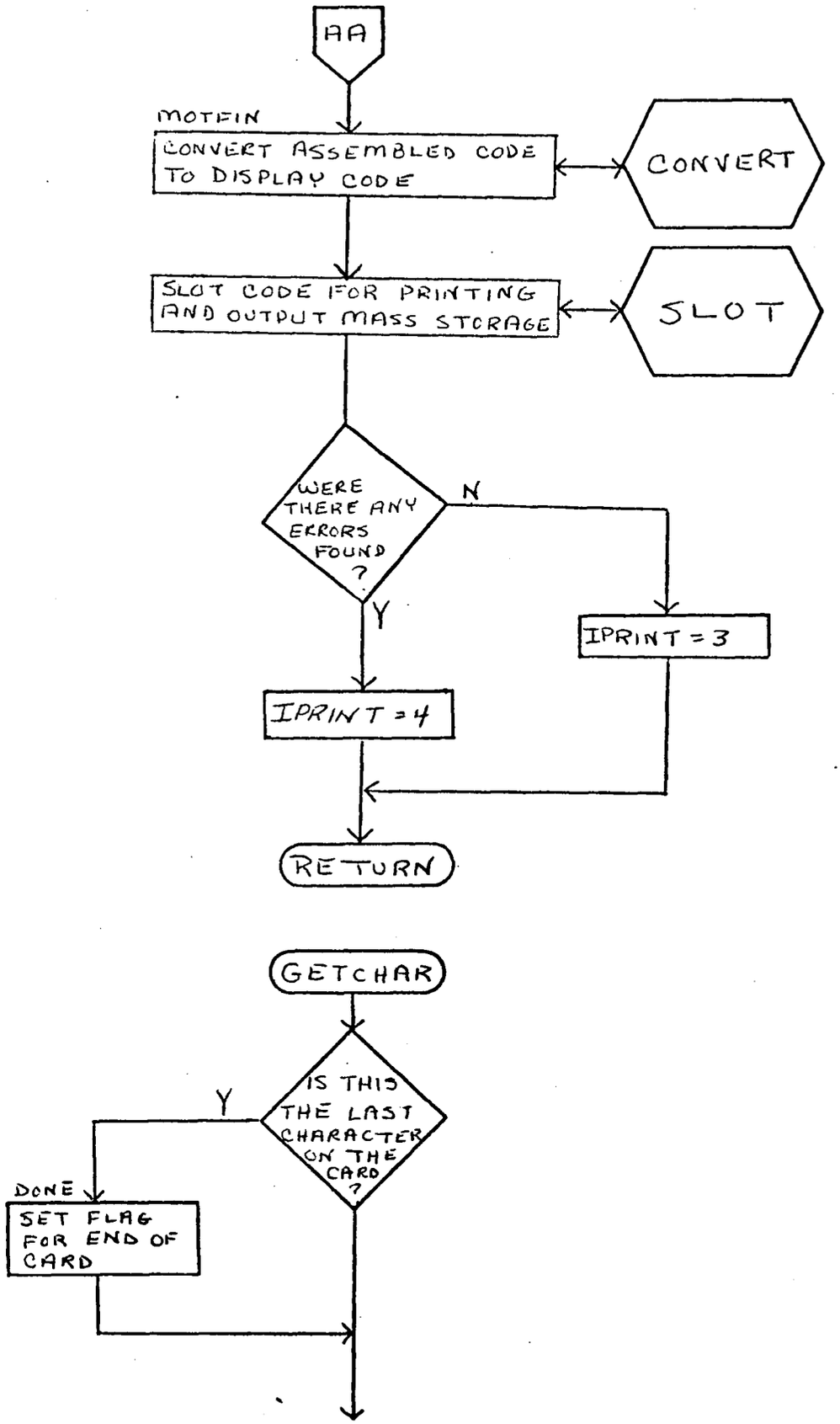

SEARCH PSEUDS OP TABLE FOR MATCH
MATCH FOUND ?
N
Y
BRANCH TO PROPER PROCESSER
J 52 (DS)
K 52 (DC)
L 52 (EQU)
M 47 (USING)
N 46 (DROP)
Q 51 (LTORG)
O 44 (START)
R 44 (ORG)
P 43 (END)
S 49 (IO)

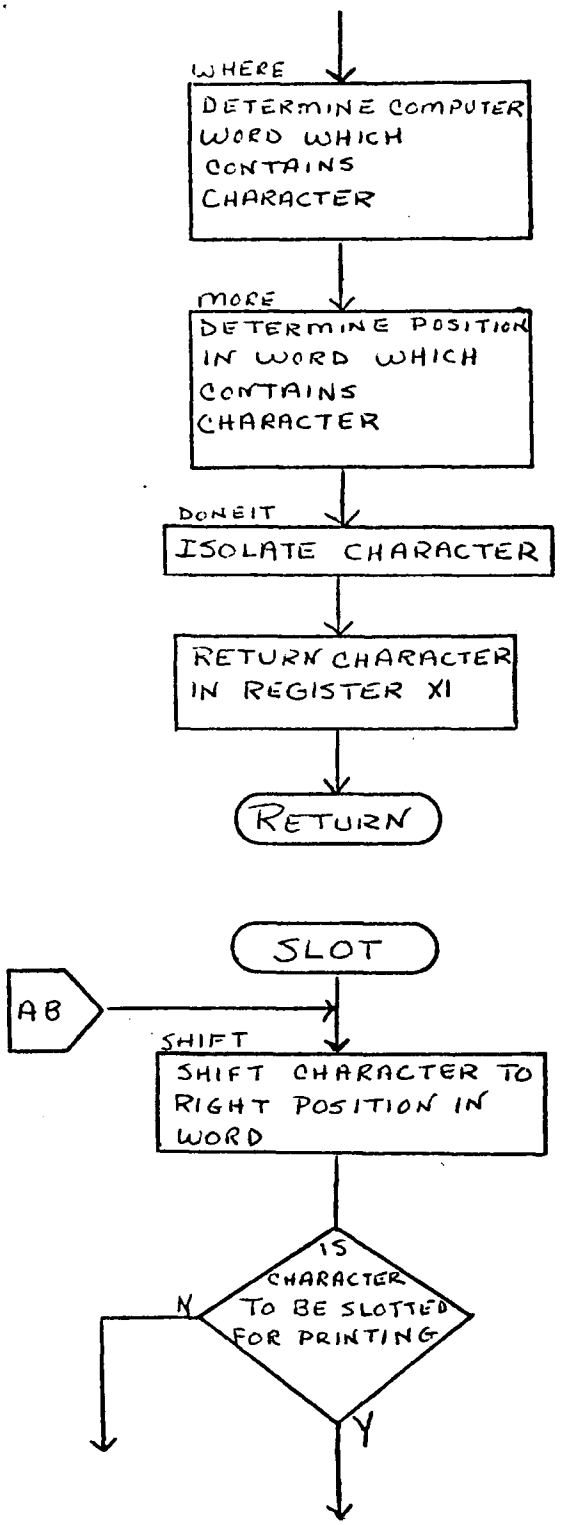

LABR
SEARCH MACHINE OPERATION TABLE
MATCH FOUND ?
N
I 34
Y
LABIN
ADJUST LOCATION COUNTER TO A HALF WORD BOUNDARY
HALF
UPDATE LOCATION COUNTER
MAKE PROPER LABEL ENTRY IN SYMBOL TABLE
SYMFIX
IS THIS PASS 1?
Y
N
POSITION CARD TO FIRST NON BLANK CHARACTER
POSITION
END OF CARD ?
Y
IERR = 16
AA
N
BRANCH TO PROPER INSTRUCTION PROCESSOR
T 57 (RR)
U 59 (RX)
V 63 (RS)
W 67 (SI)
X 71 (SS)
Y 57 (RR EXTENDED MNEMONICS)
Z 55 (RX EXTENDED MNEMONICS)

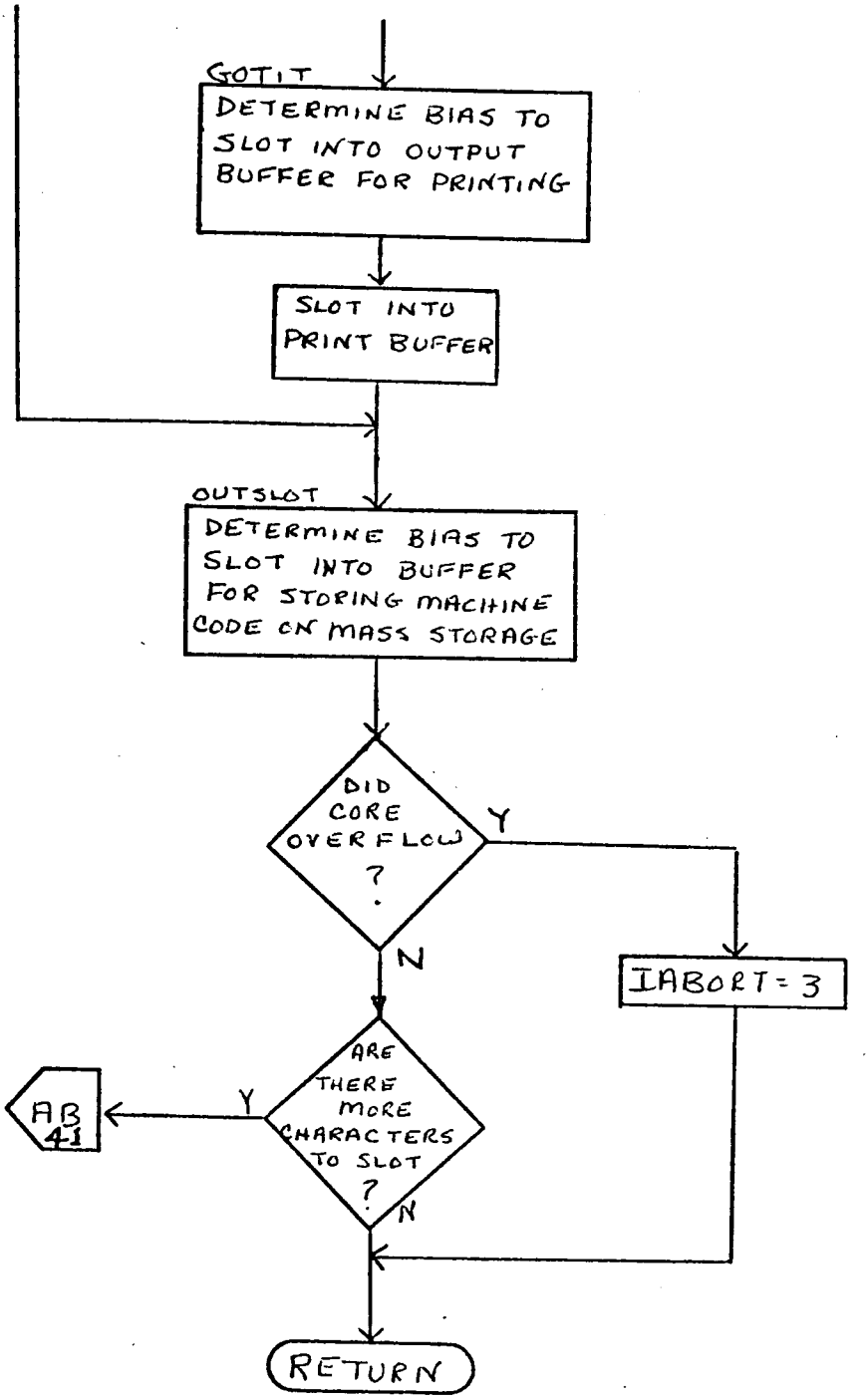

CHECKLIT
GET NEXT CHARACTER
GETCHAR
IS IT AN = ?
N
END OF CARD ?
N
RETURN
FINDLIT
GET NEXT CHARACTER
IS IT A VALID LITERAL TYPE ?
N
RETURN
Y
FOUNDL
GET NEXT CHARACTER
GETCHAR
IS IT A LEFT PARENS ?
N
RETURN
Y

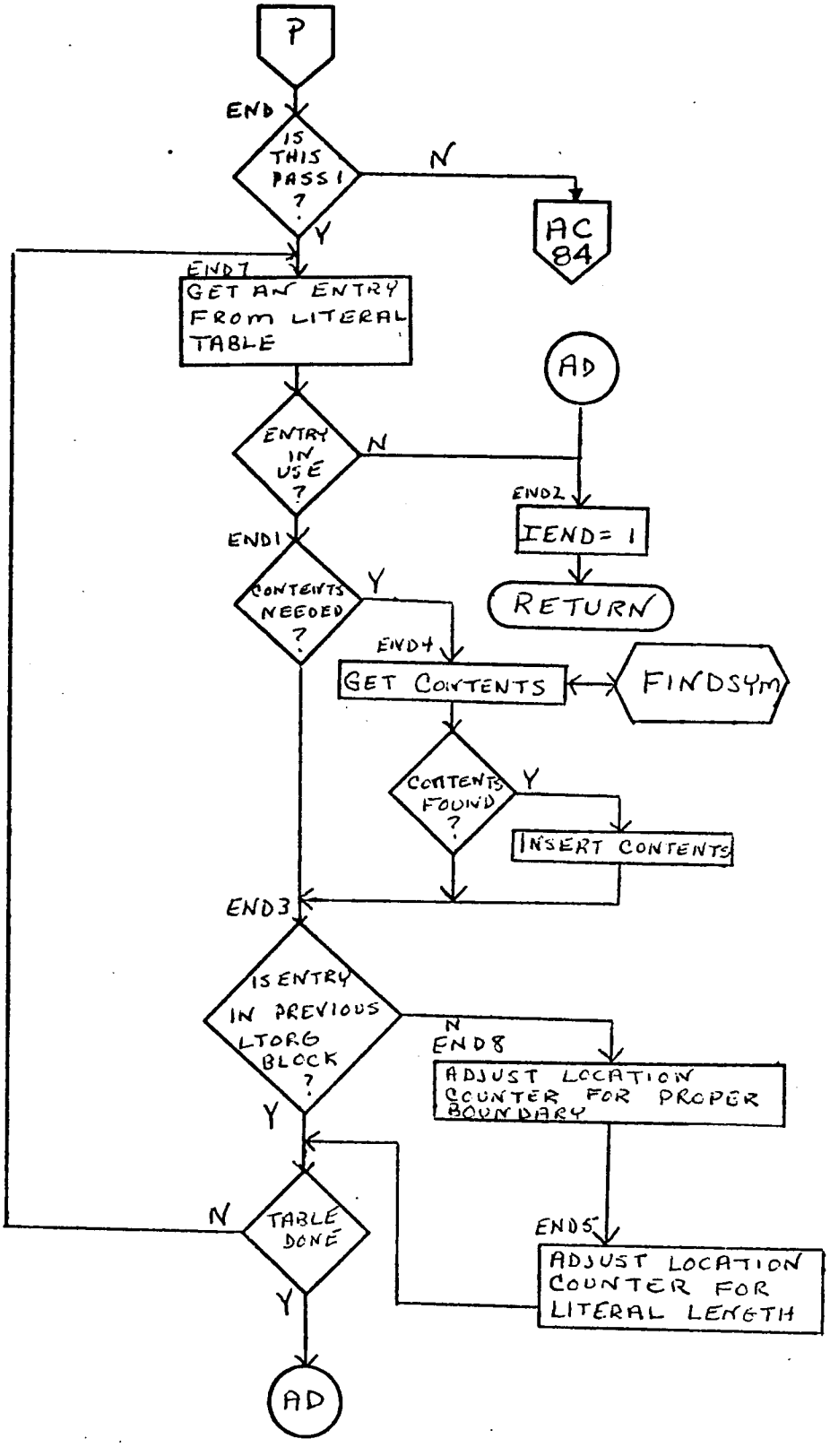

GET NEXT CHARACTER
GETCHAR
MORECHAR
SAVE IN CURLIT
GET NEXT CHARACTER
GETCHAR
IS IT A RIGHT PARENS ?
Y
N
MORE THAN 6 CHARACTERS ?
Y
RETURN
DONELIT
CONVERT LITERAL TO BINARY
EVAL
XVAL
BVAL
FVAL
HVAL
PVAL
OR
ZVAL
DEPENDING ON TYPE
CONVERSION GOOD ?
N
RETURN
Y

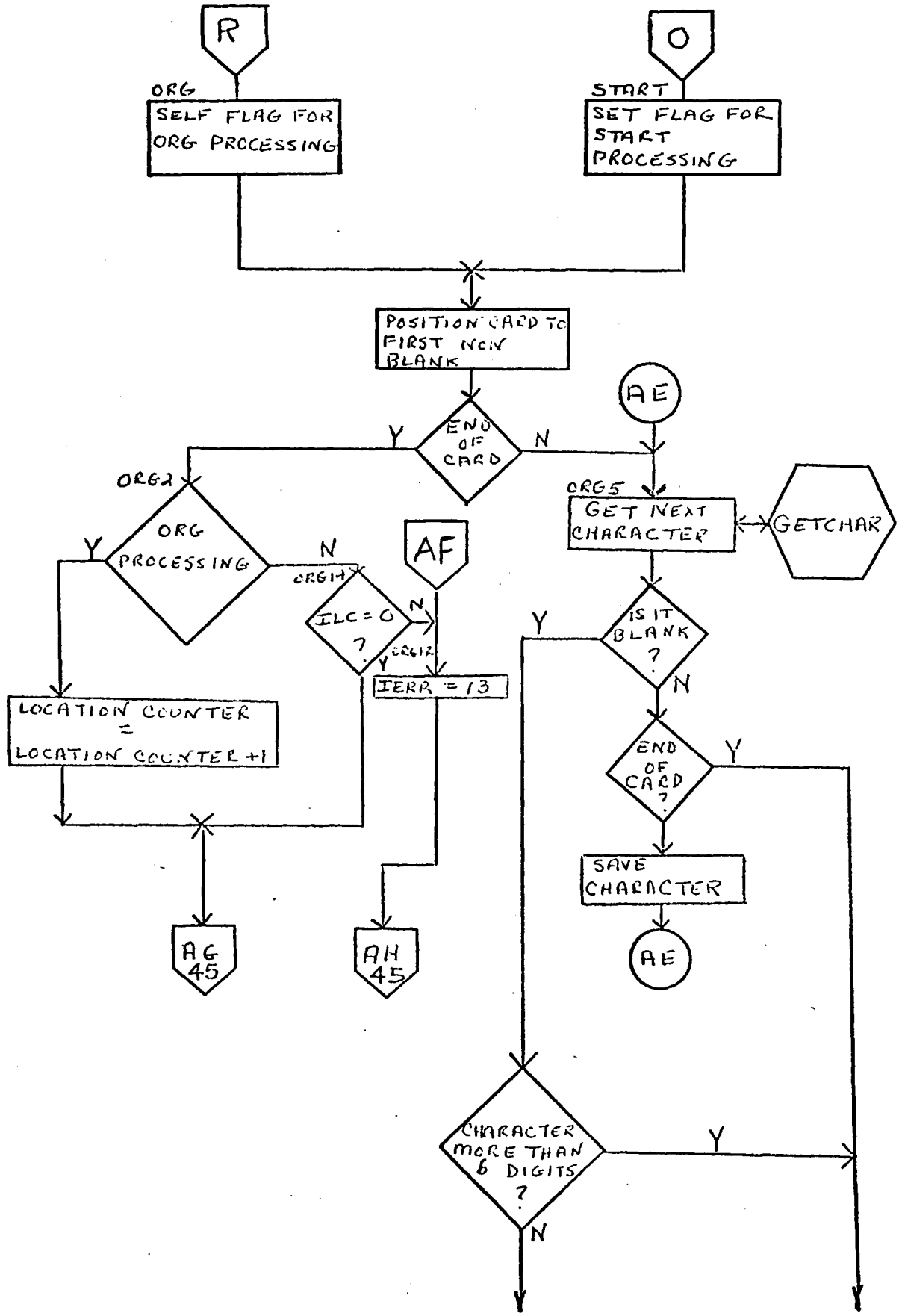

LIT5
GET AN ENTRY IN LITERAL TABLE
ENTRY IN USE ?
N
Y
SET ENTRY IN USE INSERT LITERAL INSERT LENGTH INSERT BOUNDARY
LIT1
IN THIS ENTRY IN A PREVIOUS BLOCK ?
N
LIT3
IS THIS A DUPLICATE ?
N
LIT6
END OF TABLE ?
N
Y
IABORT = 1
IS THIS AN ADDRESS LITERAL
Y
N
INSERT LITERAL CONTENTS
RETURN

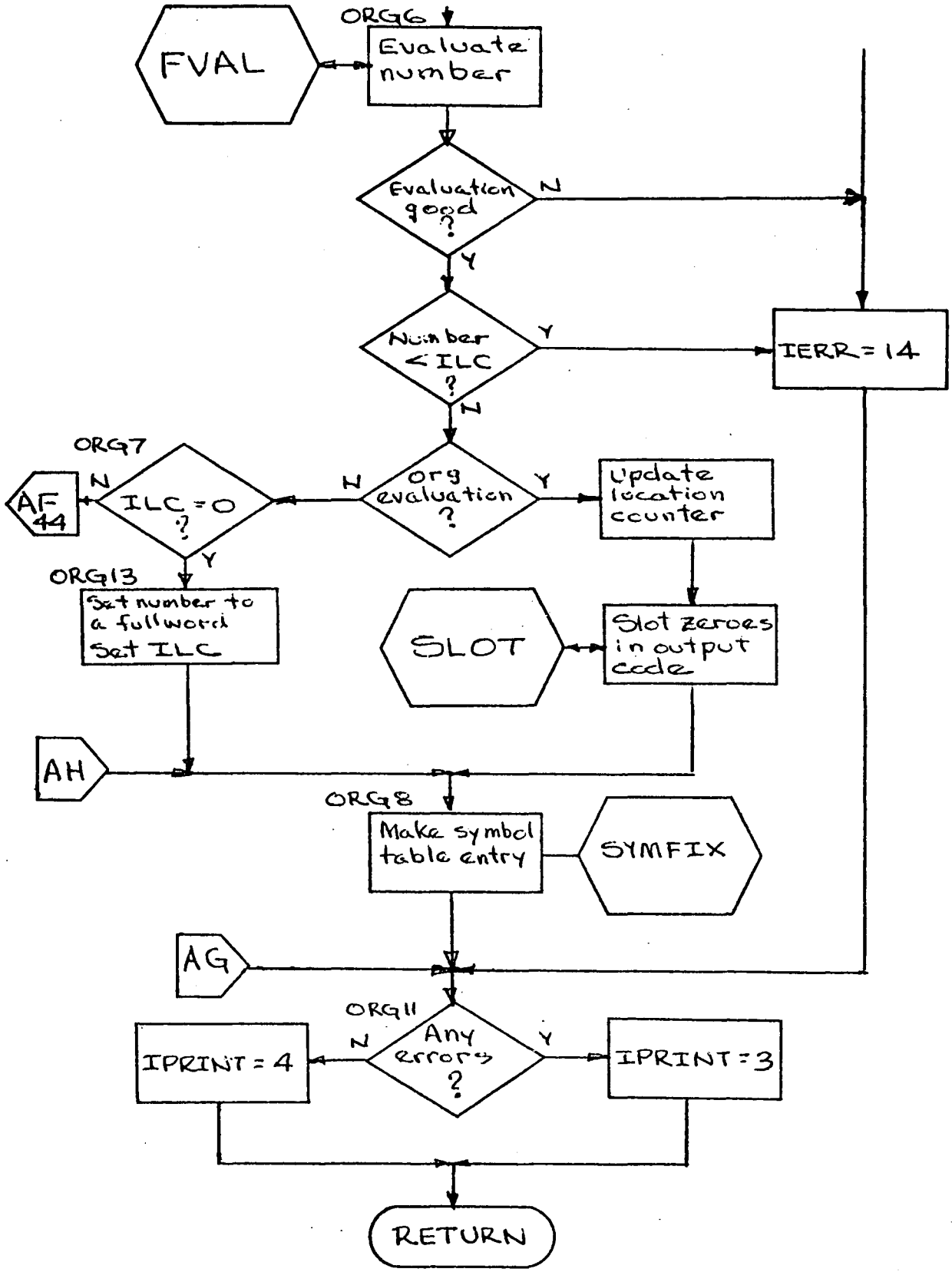

AA
MOTFIN
CONVERT ASSEMBLED CODE TO DISPLAY CODE
CONVERT
SLOT CODE FOR PRINTING AND OUTPUT MASS STORAGE
SLOT
WERE THERE ANY ERRORS FOUND ?
N
Y
IPRINT = 3
IPRINT = 4
RETURN
GETCHAR
IS THIS THE LAST CHARACTER ON THE CARD ?
Y
DONE
SET FLAG FOR END OF CARD

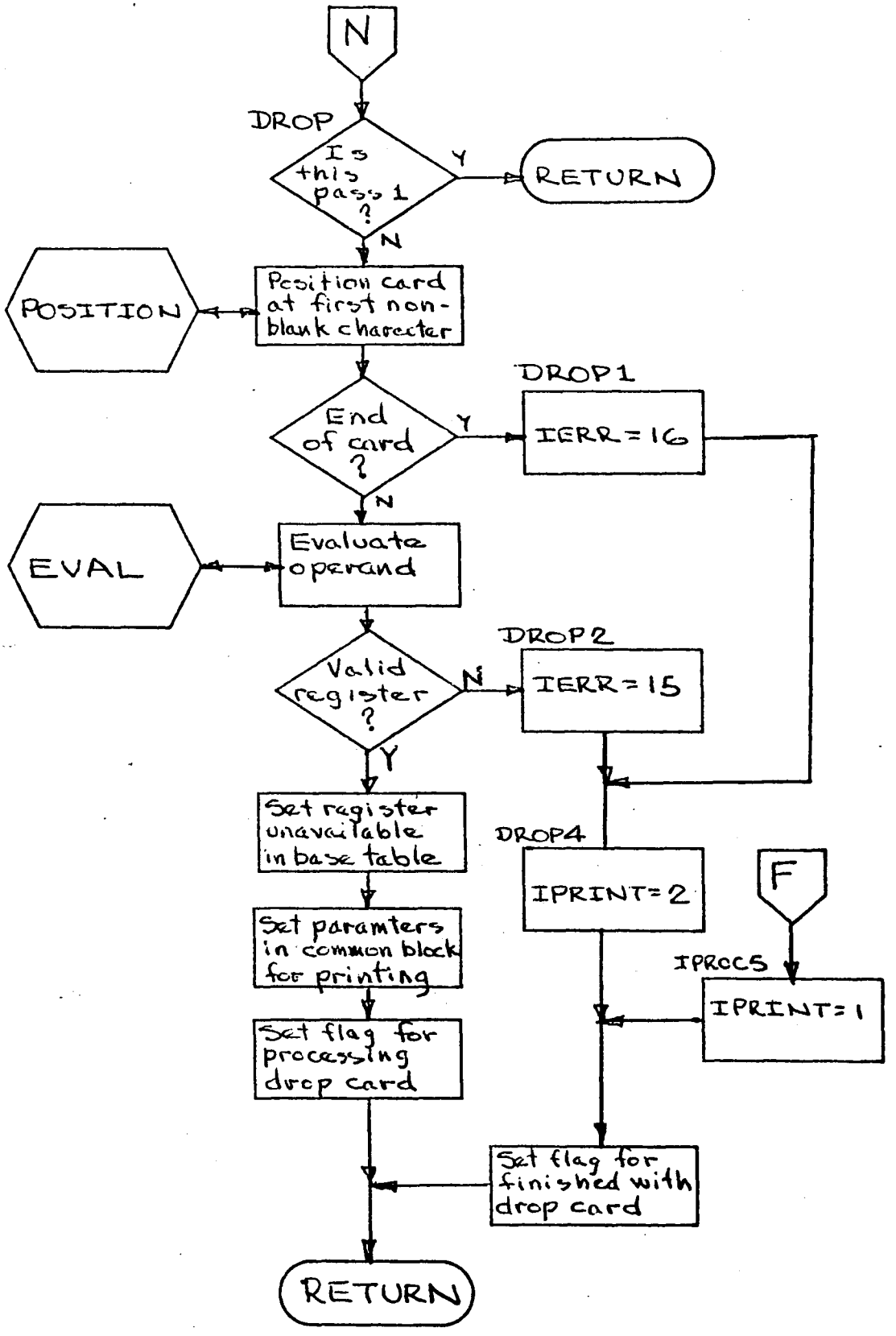

WHERE
DETERMINE COMPUTER WORD WHICH CONTAINS CHARACTER
MORE
DETERMINE POSITION IN WORD WHICH CONTAINS CHARACTER
DONEIT
ISOLATE CHARACTER
RETURN CHARACTER IN REGISTER X1
RETURN
SLOT
AB
SHIFT
SHIFT CHARACTER TO RIGHT POSITION IN WORD
IS CHARACTER TO BE SLOTTED FOR PRINTING
N
Y

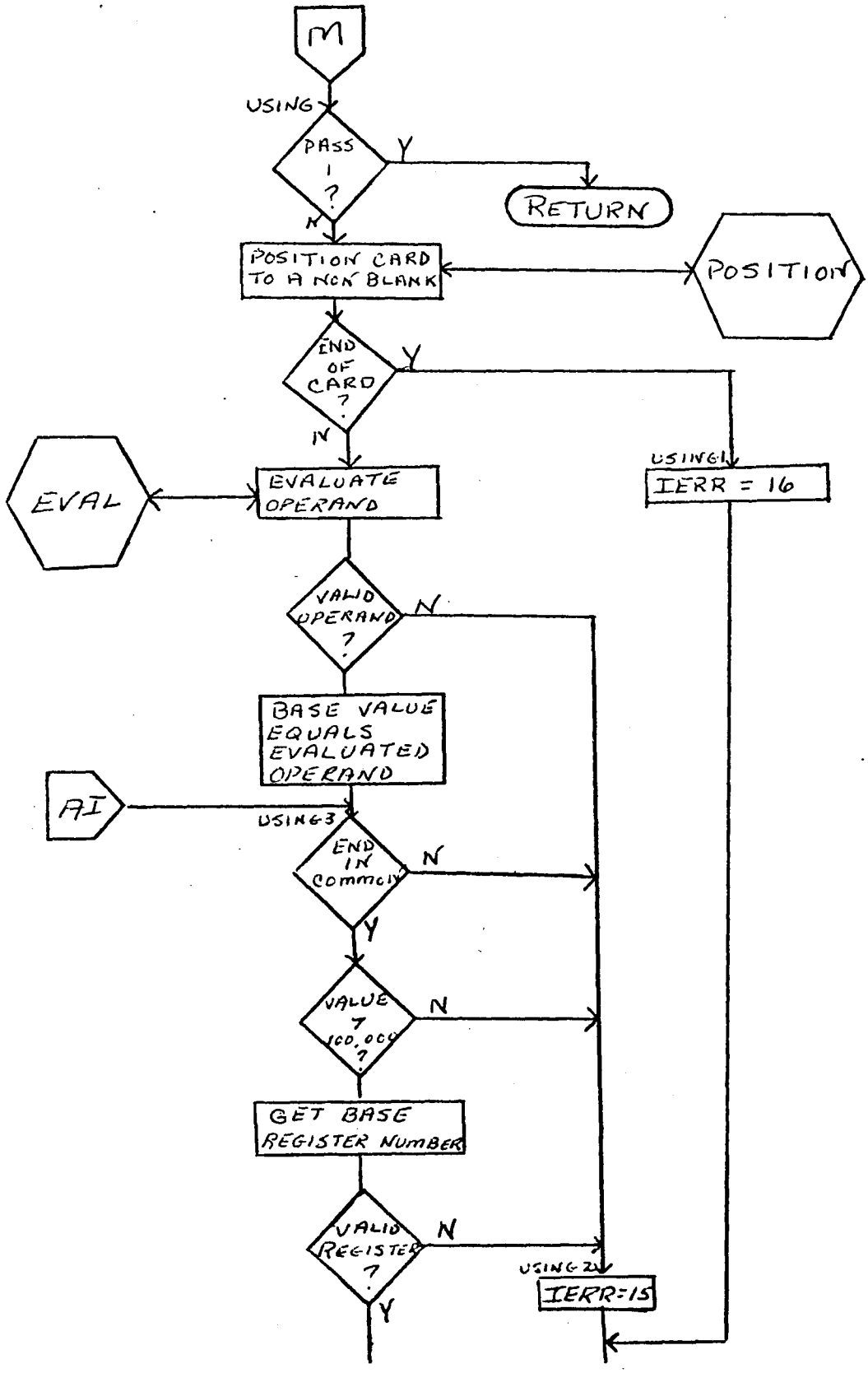

GOTIT
DETERMINE BIAS TO SLOT INTO OUTPUT BUFFER FOR PRINTING
SLOT INTO PRINT BUFFER
OUTSLOT
DETERMINE BIAS TO SLOT INTO BUFFER FOR STORING MACHINE CODE ON MASS STORAGE
DID CORE OVERFLOW ?
Y
N
IABORT = 3
ARE THERE MORE CHARACTERS TO SLOT ?
Y
N
AB 4-1
RETURN

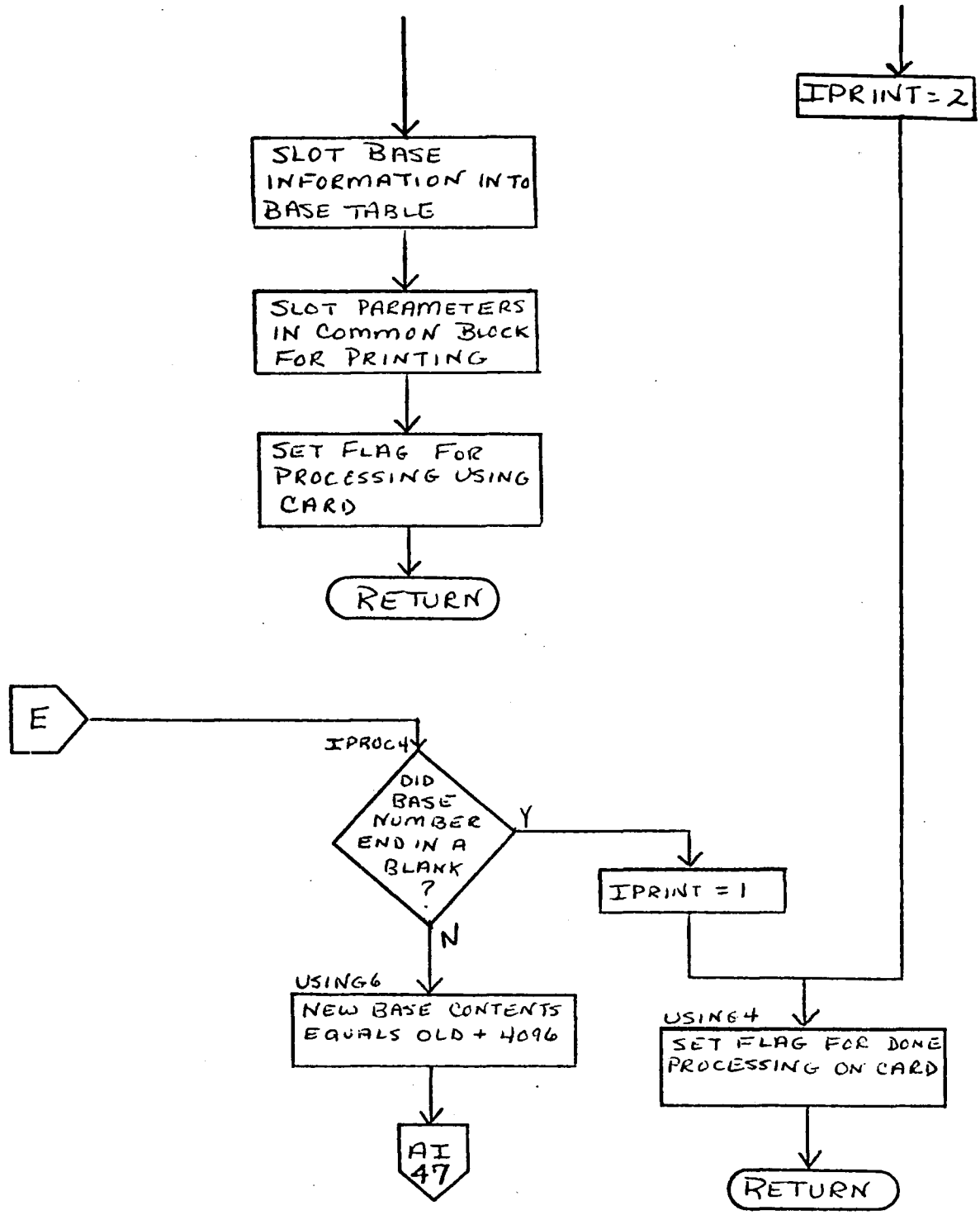

P
END
IS THIS PASS 1 ?
N
AC 84
Y
END7
GET AN ENTRY FROM LITERAL TABLE
AD
ENTRY IN USE ?
N
END2
IEND = 1
RETURN
END1
CONTENTS NEEDED ?
Y
END4
GET CONTENTS
FINDSYM
CONTENTS FOUND ?
Y
INSERT CONTENTS
END3
IS ENTRY IN PREVIOUS LTORG BLOCK ?
N
END8
ADJUST LOCATION COUNTER FOR PROPER BOUNDARY
Y
N
TABLE DONE
END5
ADJUST LOCATION COUNTER FOR LITERAL LENGTH
Y
AD

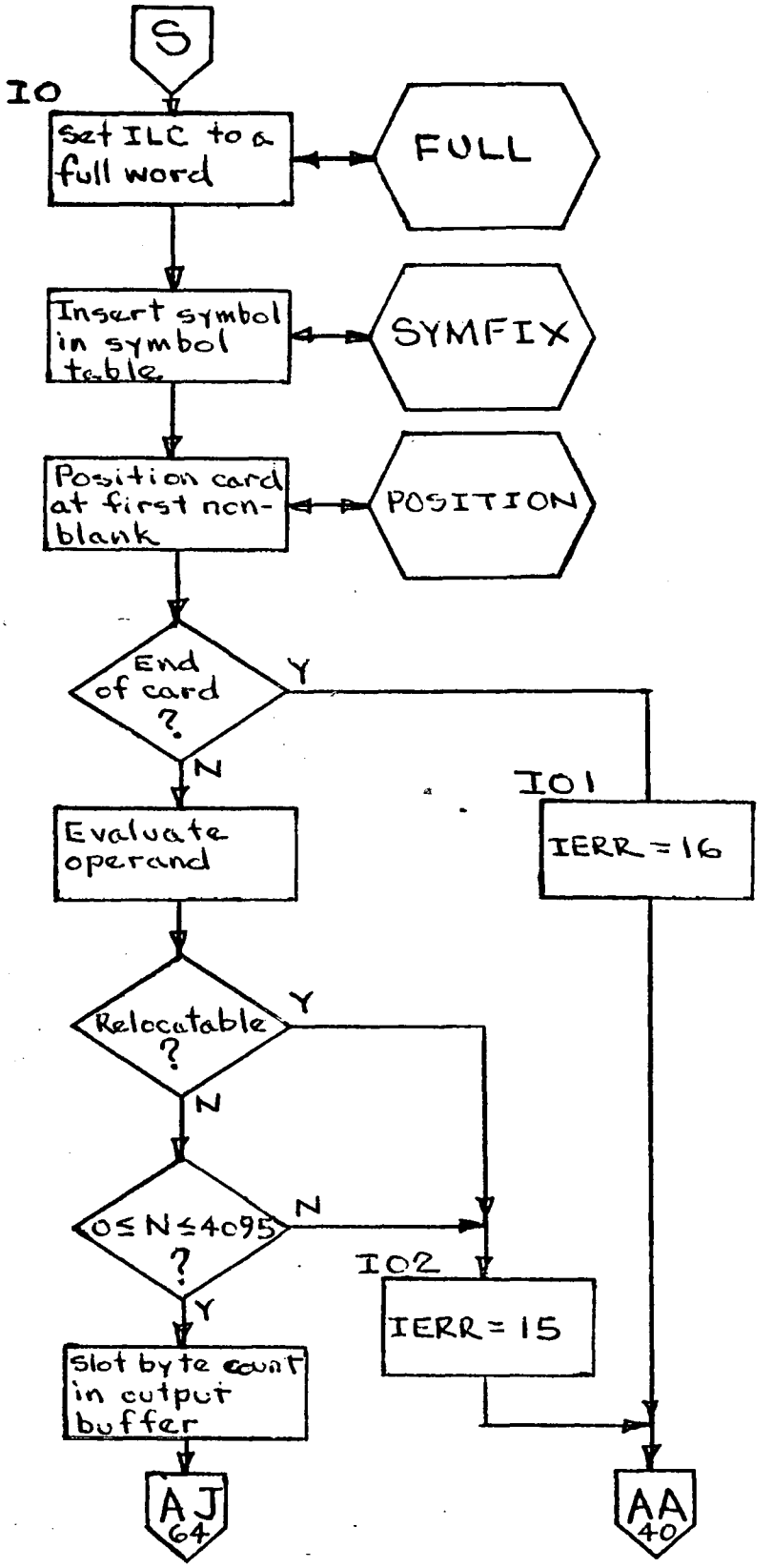

R
ORG
SELF FLAG FOR ORG PROCESSING
O
START
SET FLAG FOR START PROCESSING
POSITION CARD TO FIRST NON BLANK
AE
END OF CARD
Y
N
ORG2
ORG PROCESSING
Y
N
ORG1
ILC = 0 ?
N
Y
AF
ERR13
IERR = 13
LOCATION COUNTER = LOCATION COUNTER +1
AG 45
AH 45
ORG5
GET NEXT CHARACTER
GETCHAR
IS IT BLANK ?
Y
N
END OF CARD ?
Y
SAVE CHARACTER
AE
CHARACTER MORE THAN 6 DIGITS ?
Y
N

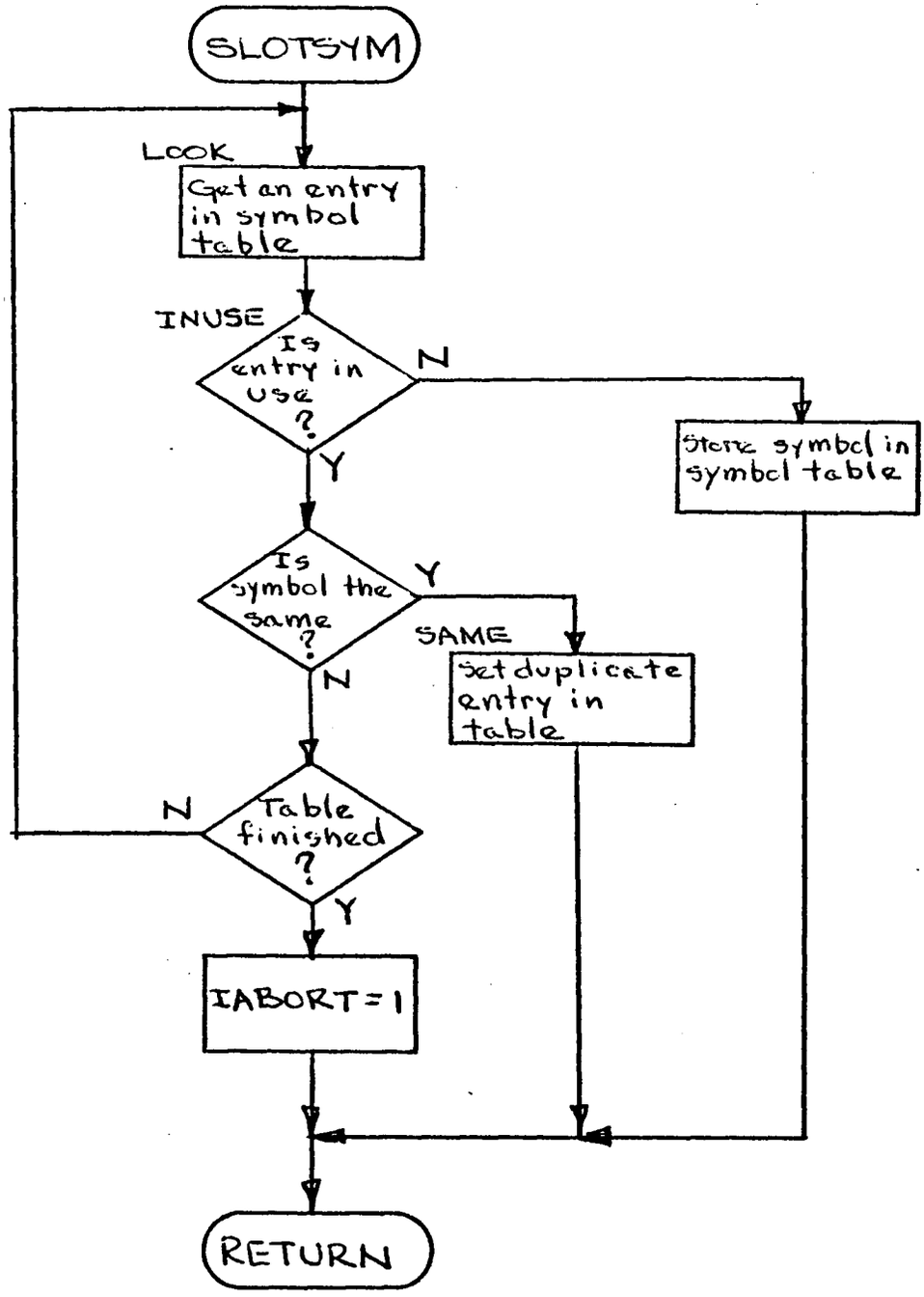

ORG6
FVAL
Evaluate number
Evaluation good ?
N
Y
Number < ILC ?
Y
IERR = 14
N
ORG7
AF 44
N
ILC = 0 ?
N
Org evaluation ?
Y
Update location counter
Y
ORG13
Set number to a fullword
Set ILC
SLOT
Slot zeroes in output code
AH
ORG8
Make symbol table entry
SYMFIX
AG
ORG11
IPRINT = 4
N
Any errors ?
Y
IPRINT = 3
RETURN

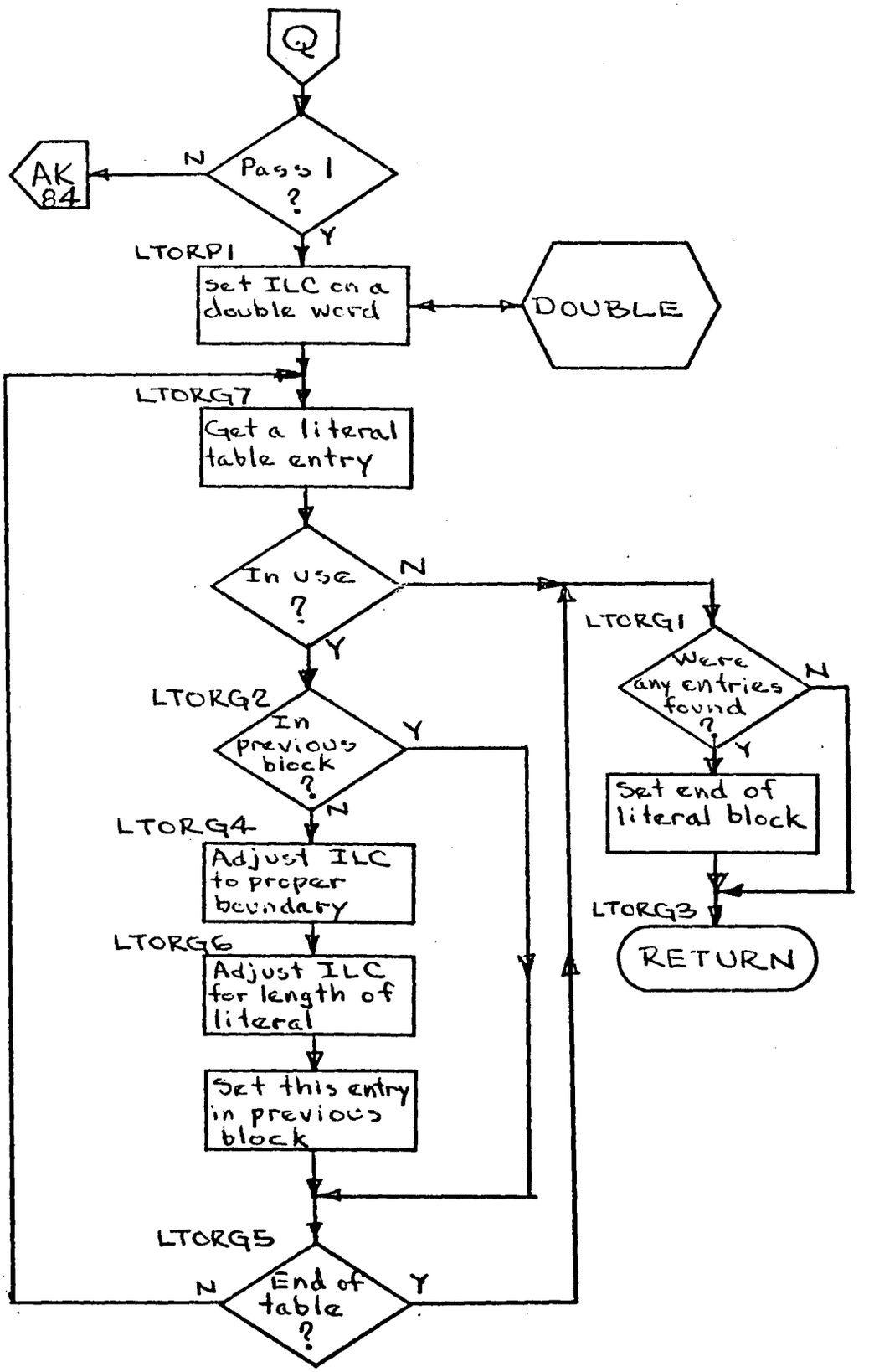

N
DROP
Is this pass 1 ?
Y
RETURN
N
POSITION
Position card at first non-blank character
DROP1
End of card ?
Y
IERR=16
N
EVAL
Evaluate operand
DROP2
Valid register ?
N
IERR=15
Y
Set register unavailable in base table
DROP4
IPRINT=2
F
IPROC5
IPRINT=1
Set paramters in common block for printing
Set flag for processing drop card
Set flag for finished with drop card
RETURN

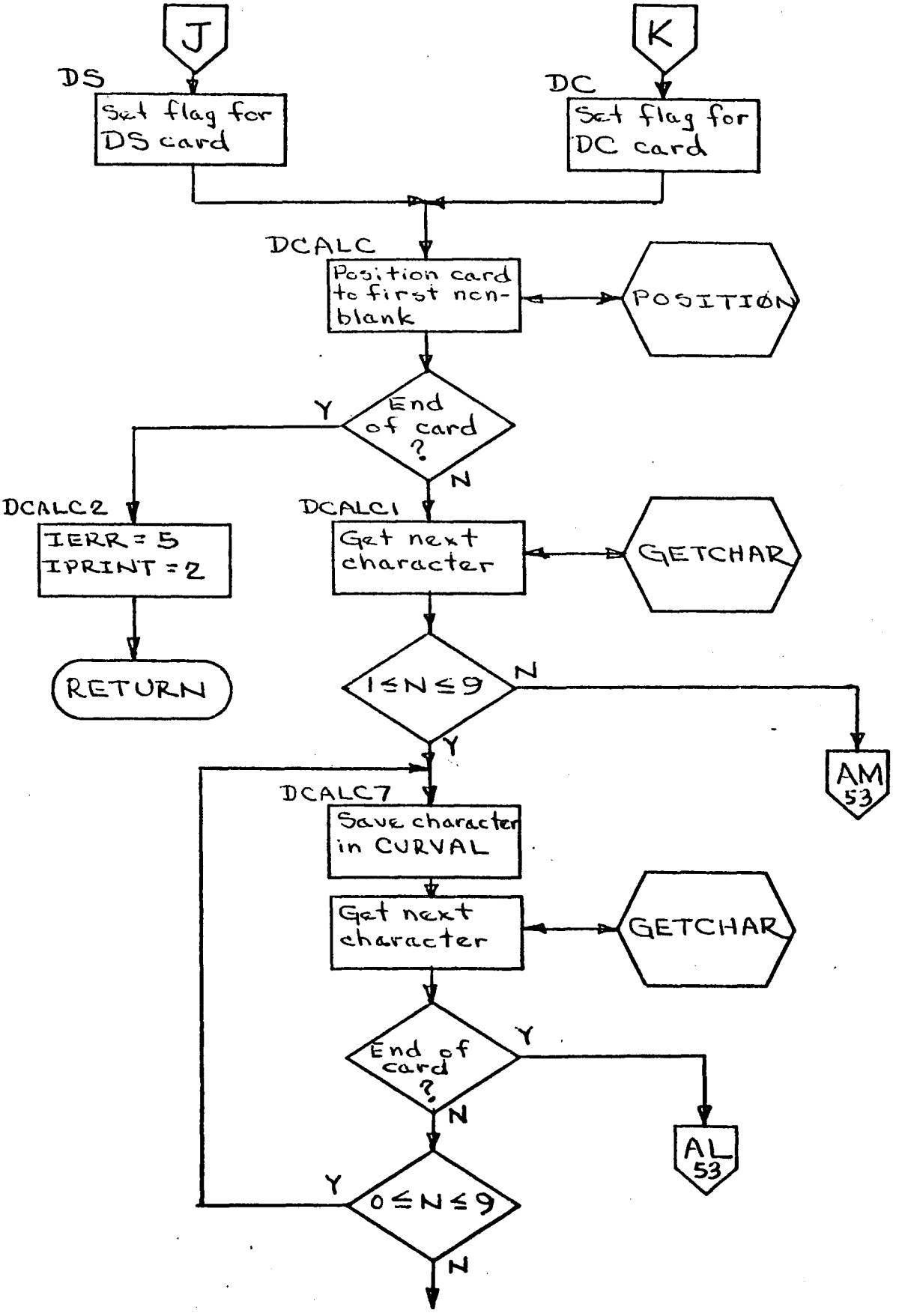

M
USING
PASS 1 ?
Y
N
RETURN
POSITION CARD TO A NON BLANK
POSITION
END OF CARD ?
Y
N
USING1
IERR = 16
EVAL
EVALUATE OPERAND
VALID OPERAND ?
N
BASE VALUE EQUALS EVALUATED OPERAND
AI
USING3
END IN COMMA ?
N
Y
VALUE ? 1CO, OCC ?
N
GET BASE REGISTER NUMBER
VALID REGISTER ?
N
Y
USING2
IERR=15

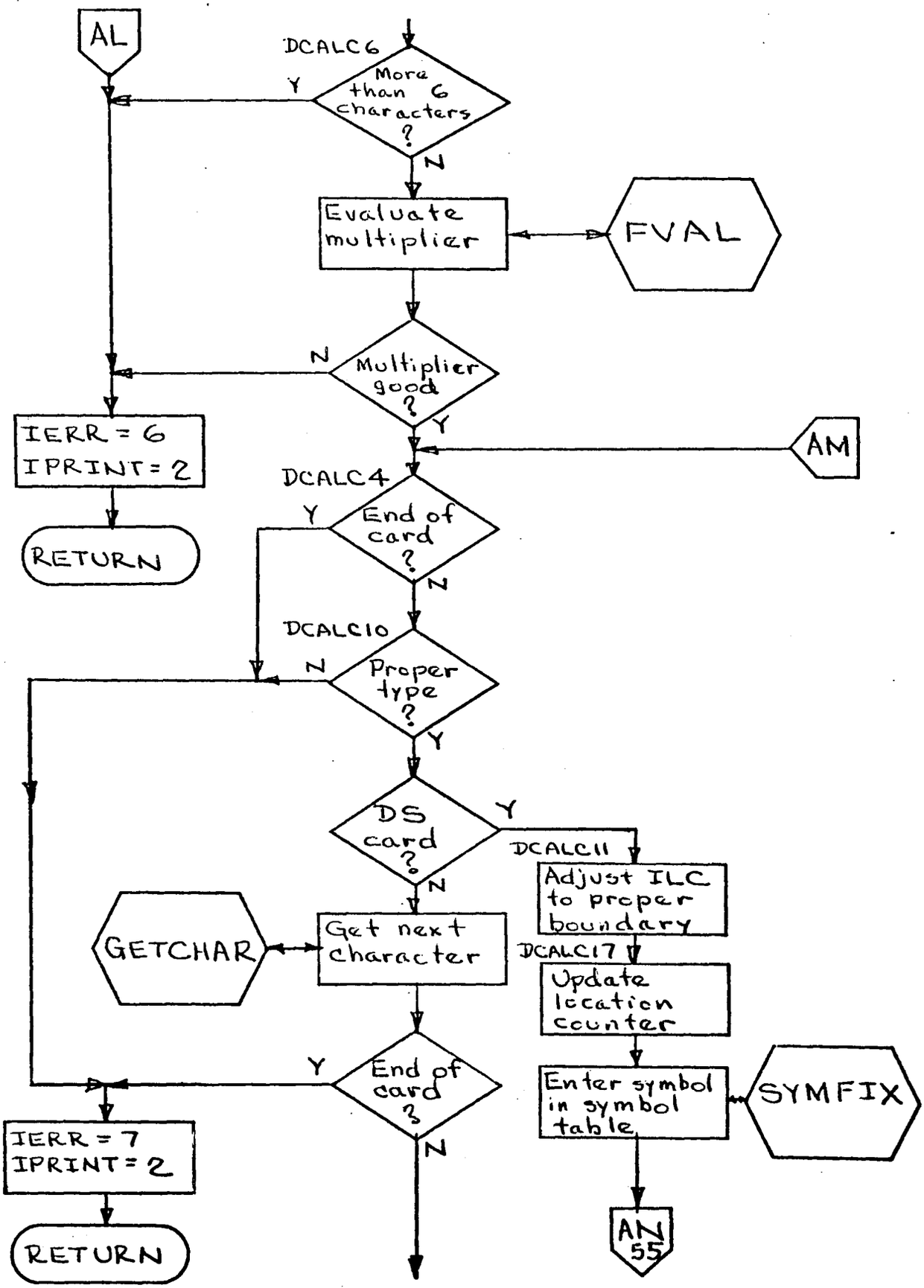

IPRINT=2
SLOT BASE INFORMATION INTO BASE TABLE
SLOT PARAMETERS IN COMMON BLOCK FOR PRINTING
SET FLAG FOR PROCESSING USING CARD
RETURN
E
IPROC4
DID BASE NUMBER END IN A BLANK ?
Y
N
IPRINT = 1
USING6
NEW BASE CONTENTS EQUALS OLD + 4096
AI 47
USING4
SET FLAG FOR DONE PROCESSING ON CARD
RETURN

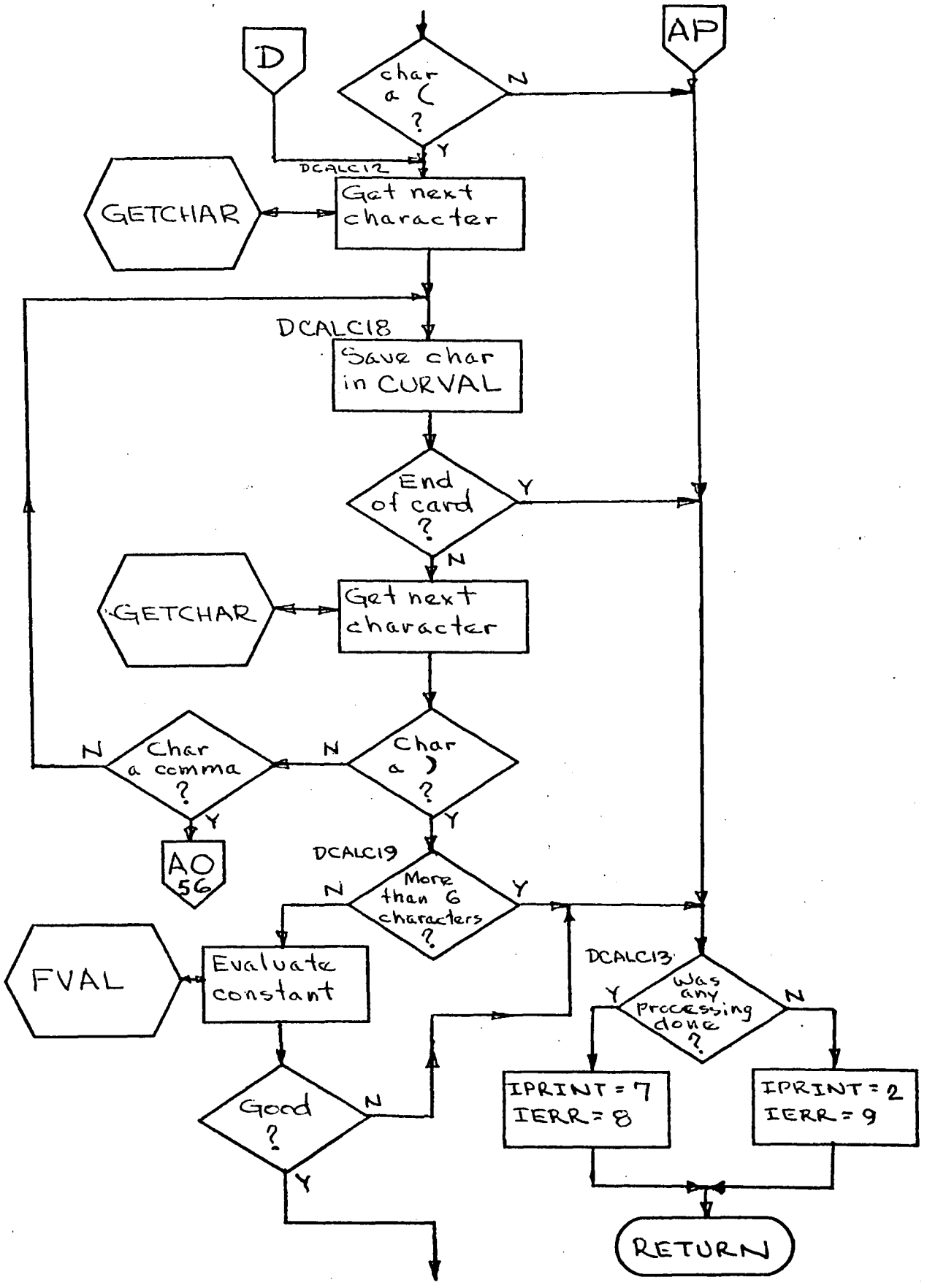

S
IO
Set ILC to a full word
FULL
Insert symbol in symbol table
SYMFIX
Position card at first non-blank
POSITION
End of card ?
Y
N
IO1
IERR = 16
Evaluate operand
Relocatable ?
Y
N
0 ≤ N ≤ 4095 ?
N
Y
IO2
IERR = 15
Slot byte count in output buffer
AJ 64
AA 40

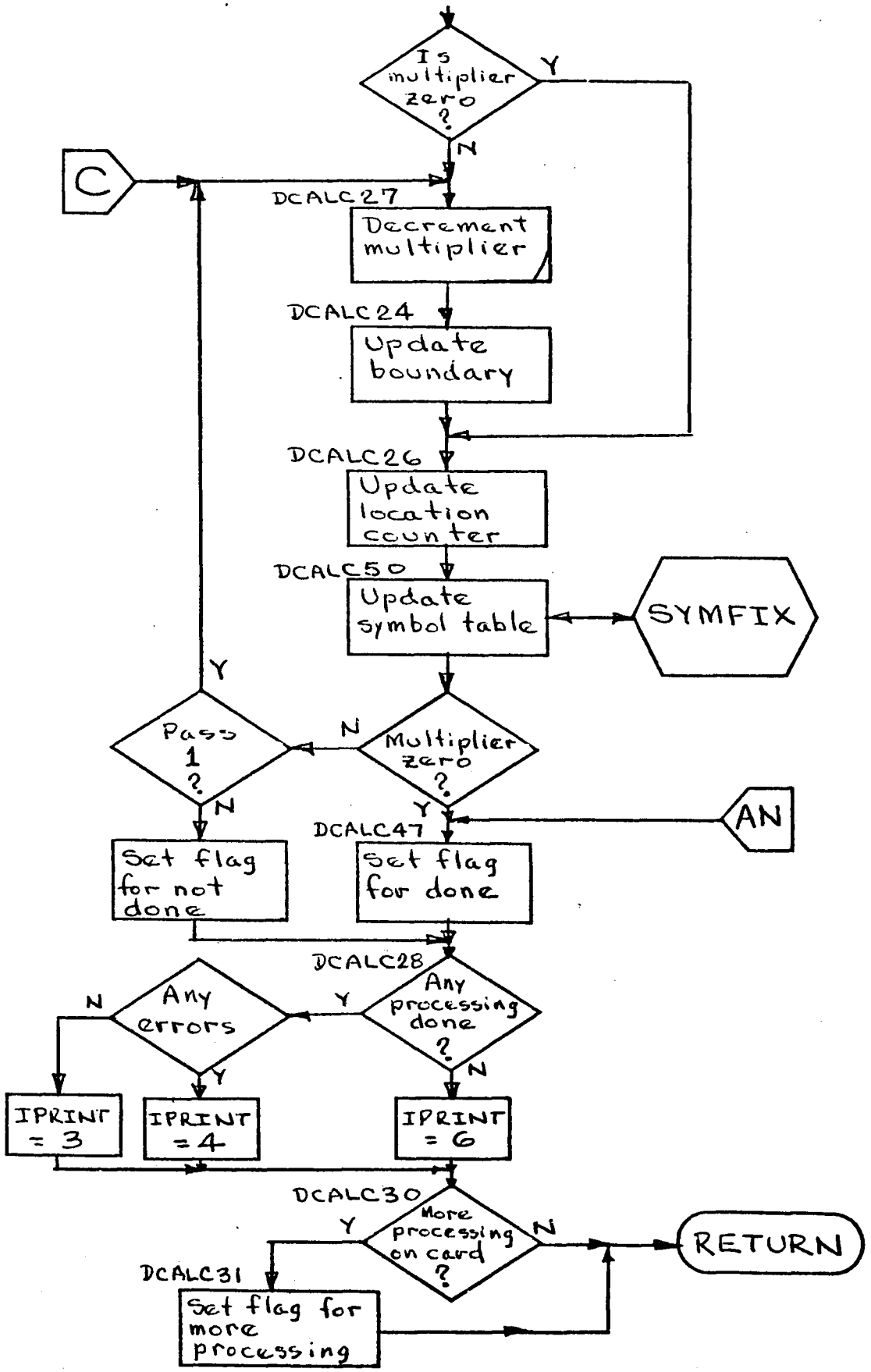

SLOTSYM
LOOK
Get an entry in symbol table
INUSE
Is entry in use ?
N
Y
Store symbol in symbol table
Is symbol the same ?
Y
SAME
N
Set duplicate entry in table
N
Table finished ?
Y
IABORT = 1
RETURN

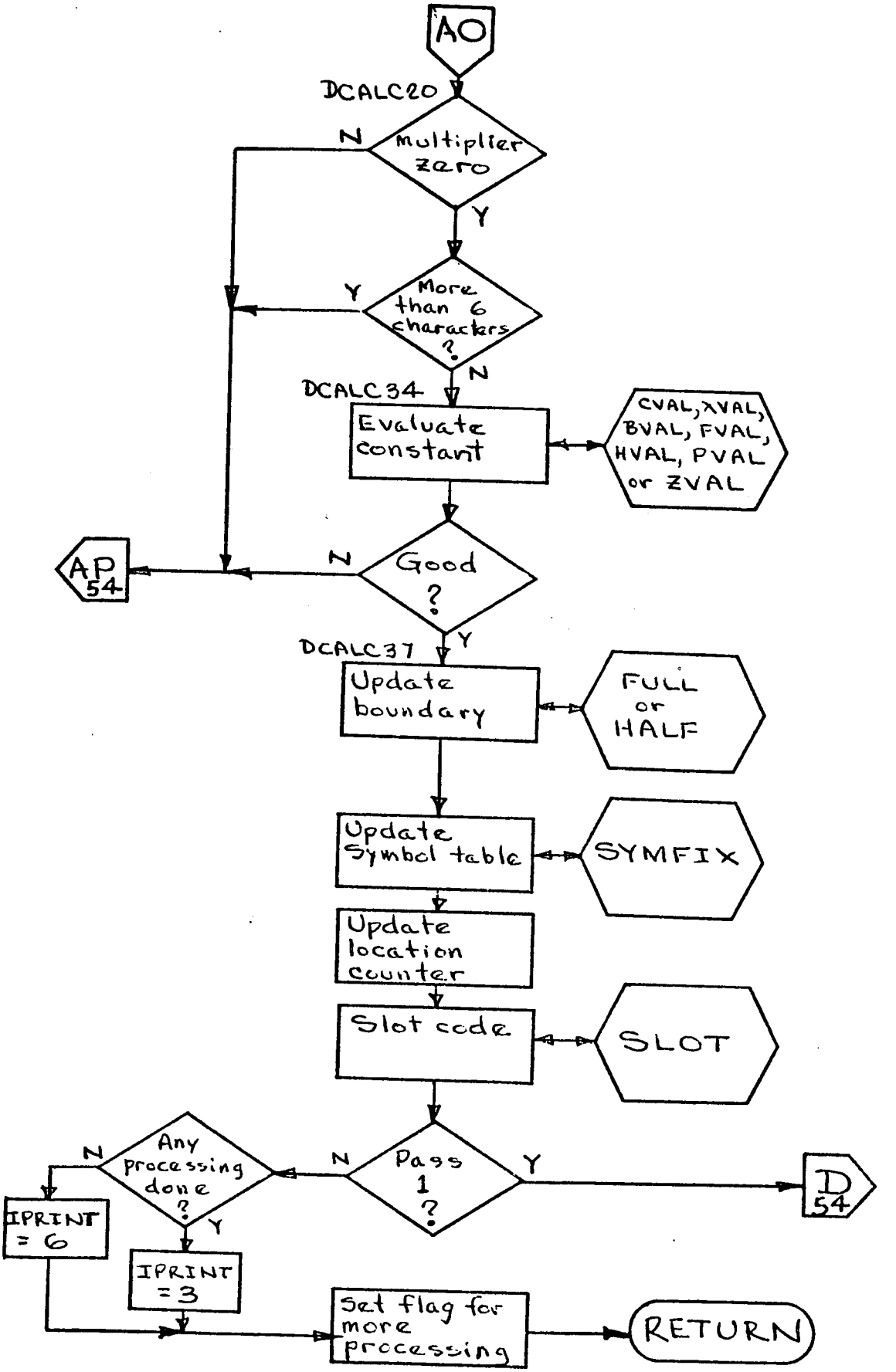

Q
AK 84
N
Pass 1 ?
Y
LTORP1
set ILC on a double word
DOUBLE
LTORG7
Get a literal table entry
In use ?
N
Y
LTORG1
Were any entries found ?
N
Y
Set end of literal block
LTORG3
RETURN
LTORG2
In previous block ?
Y
N
LTORG4
Adjust ILC to proper boundary
LTORG6
Adjust ILC for length of literal
Set this entry in previous block
LTORG5
End of table ?
N
Y

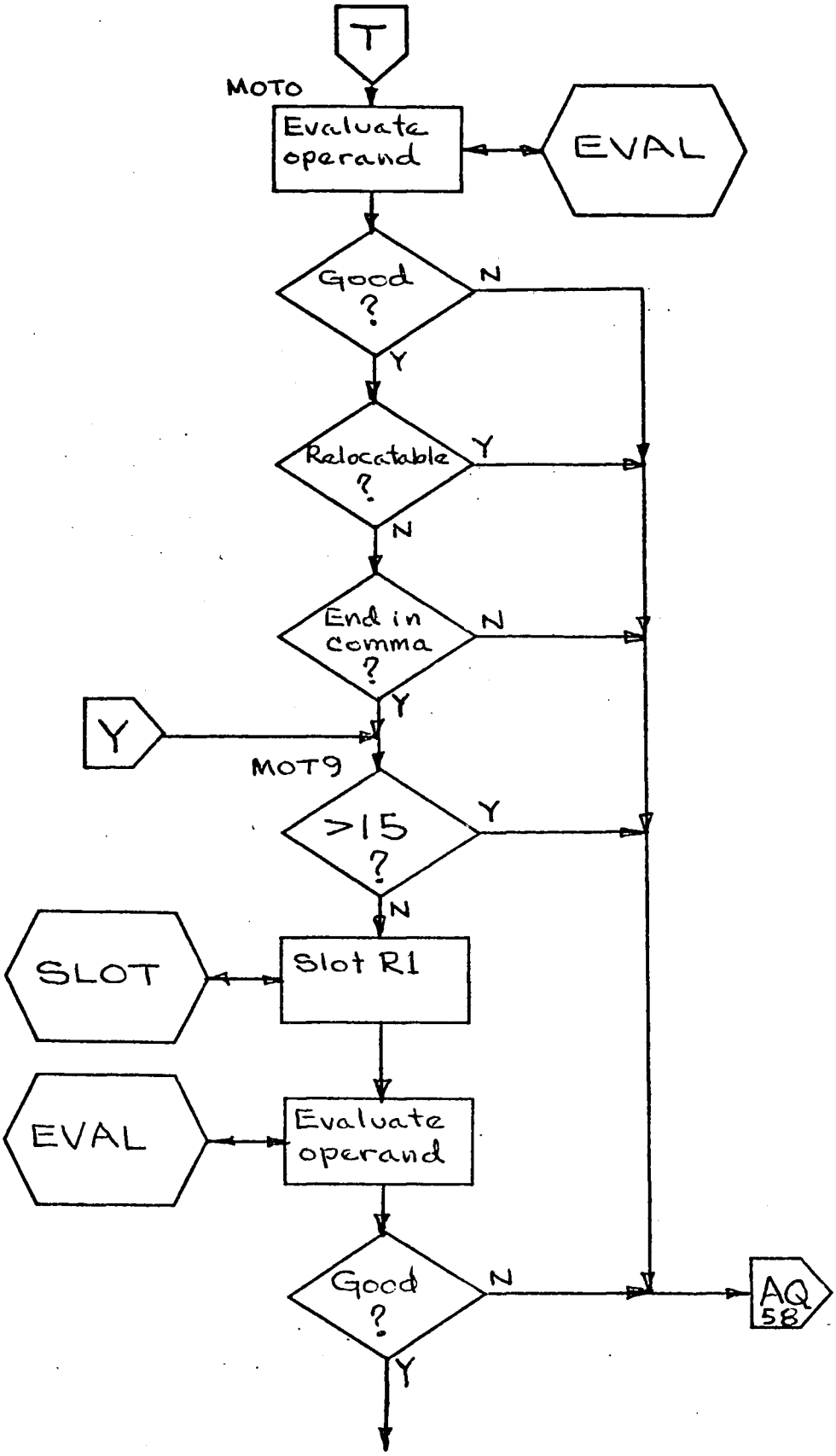

J
DS
Set flag for DS card
K
DC
Set flag for DC card
DCALC
Position card to first non-blank
POSITION
End of card ?
Y
N
DCALC2
IERR = 5
IPRINT = 2
RETURN
DCALC1
Get next character
GETCHAR
1≤N≤9
N
Y
AM 53
DCALC7
Save character in CURVAL
Get next character
GETCHAR
End of card ?
Y
N
AL 53
0≤N≤9
Y
N

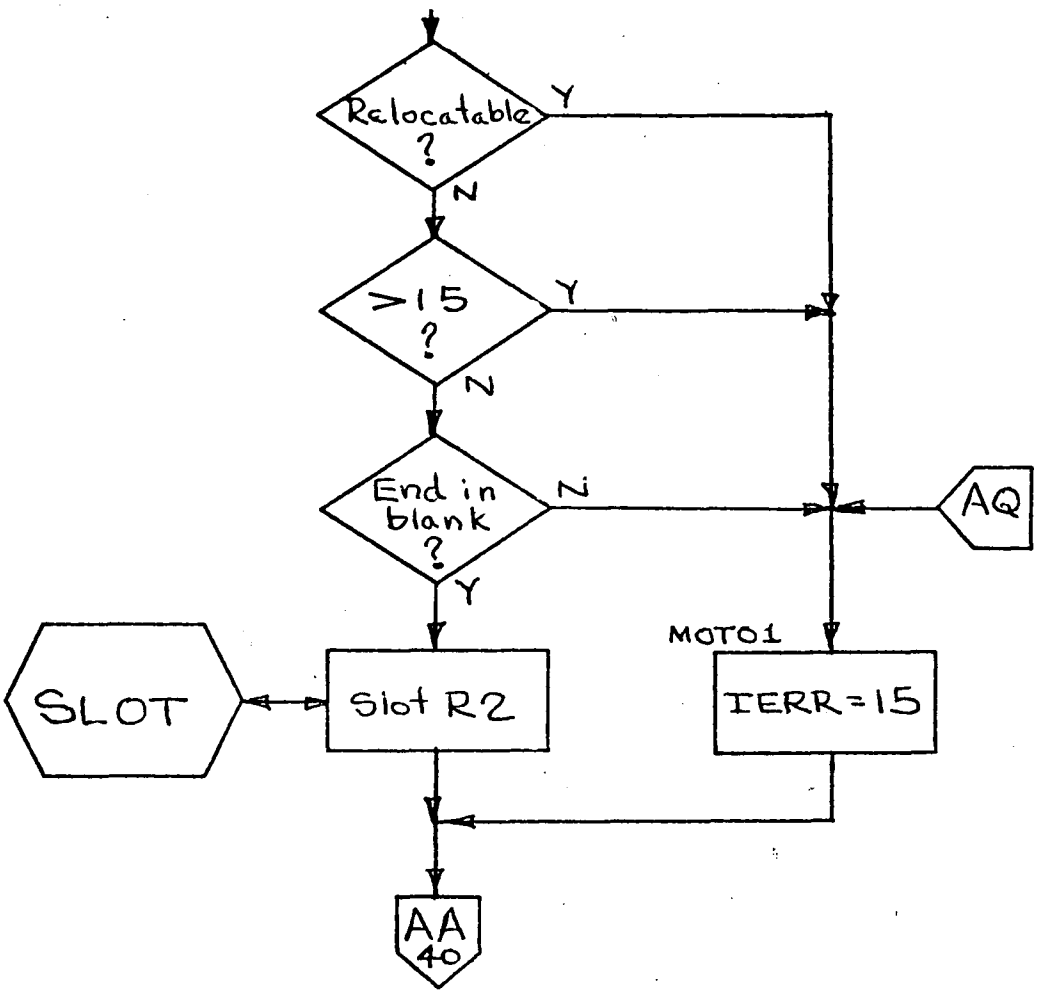

AL
DCALC6
More than 6 characters ?
Y
N
Evaluate multiplier
FVAL
Multiplier good ?
N
Y
IERR = 6
IPRINT = 2
RETURN
AM
DCALC4
End of card ?
Y
N
DCALC10
Proper type ?
N
Y
DS card ?
Y
N
DCALC11
Adjust ILC to proper boundary
DCALC17
Update location counter
GETCHAR
Get next character
End of card ?
Y
N
Enter symbol in symbol table
SYMFIX
IERR = 7
IPRINT = 2
RETURN
AN 55

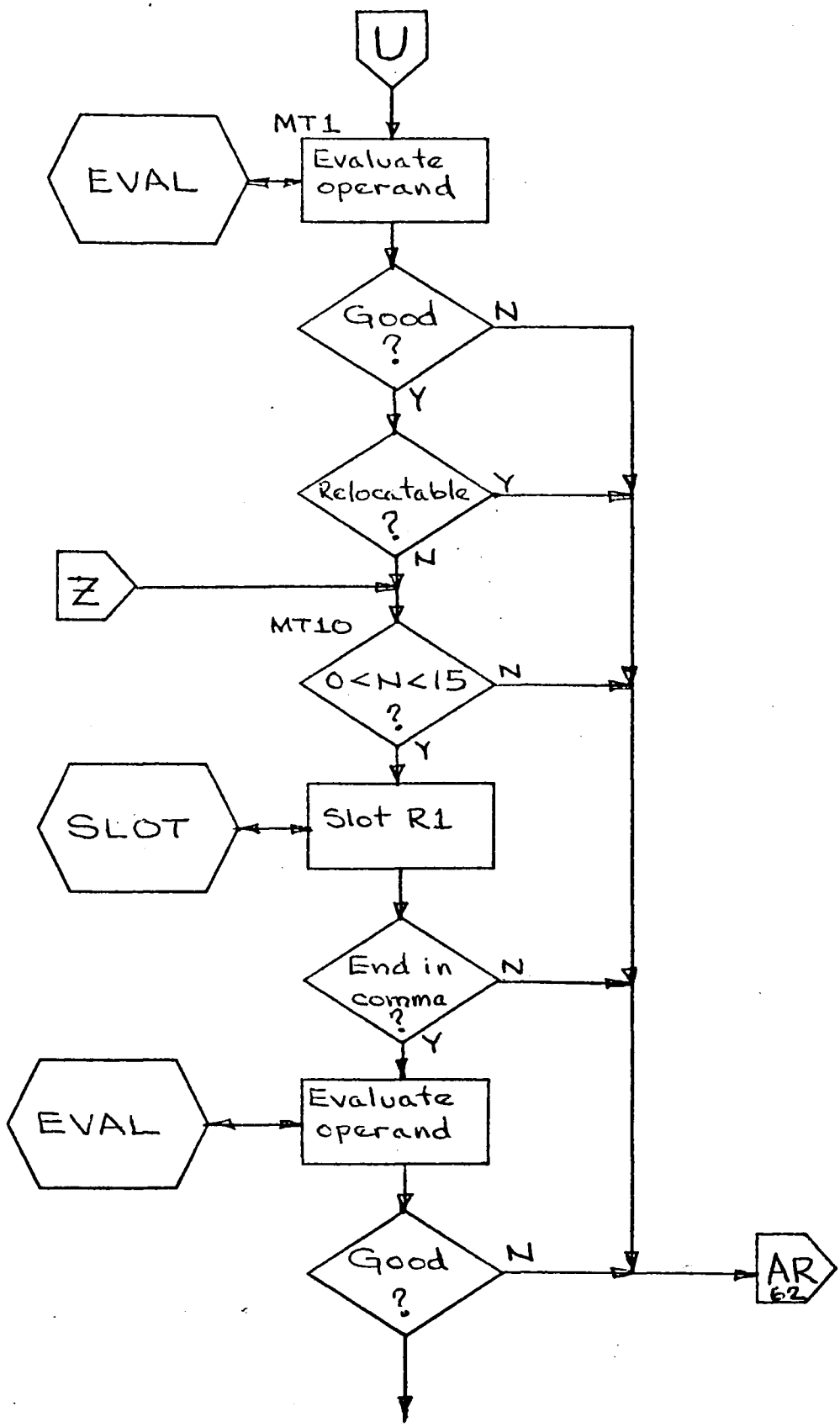

D
char a (?
N
AP
Y
DCALC12
GETCHAR
Get next character
DCALC18
Save char in CURVAL
End of card ?
Y
N
GETCHAR
Get next character
N
Char a comma ?
N
Char a) ?
Y
Y
AO 56
DCALC19
N
More than 6 characters ?
Y
FVAL
Evaluate constant
DCALC13
Y
Was any processing done ?
N
Good ?
N
Y
IPRINT = 7
IERR = 8
IPRINT = 2
IERR = 9
RETURN

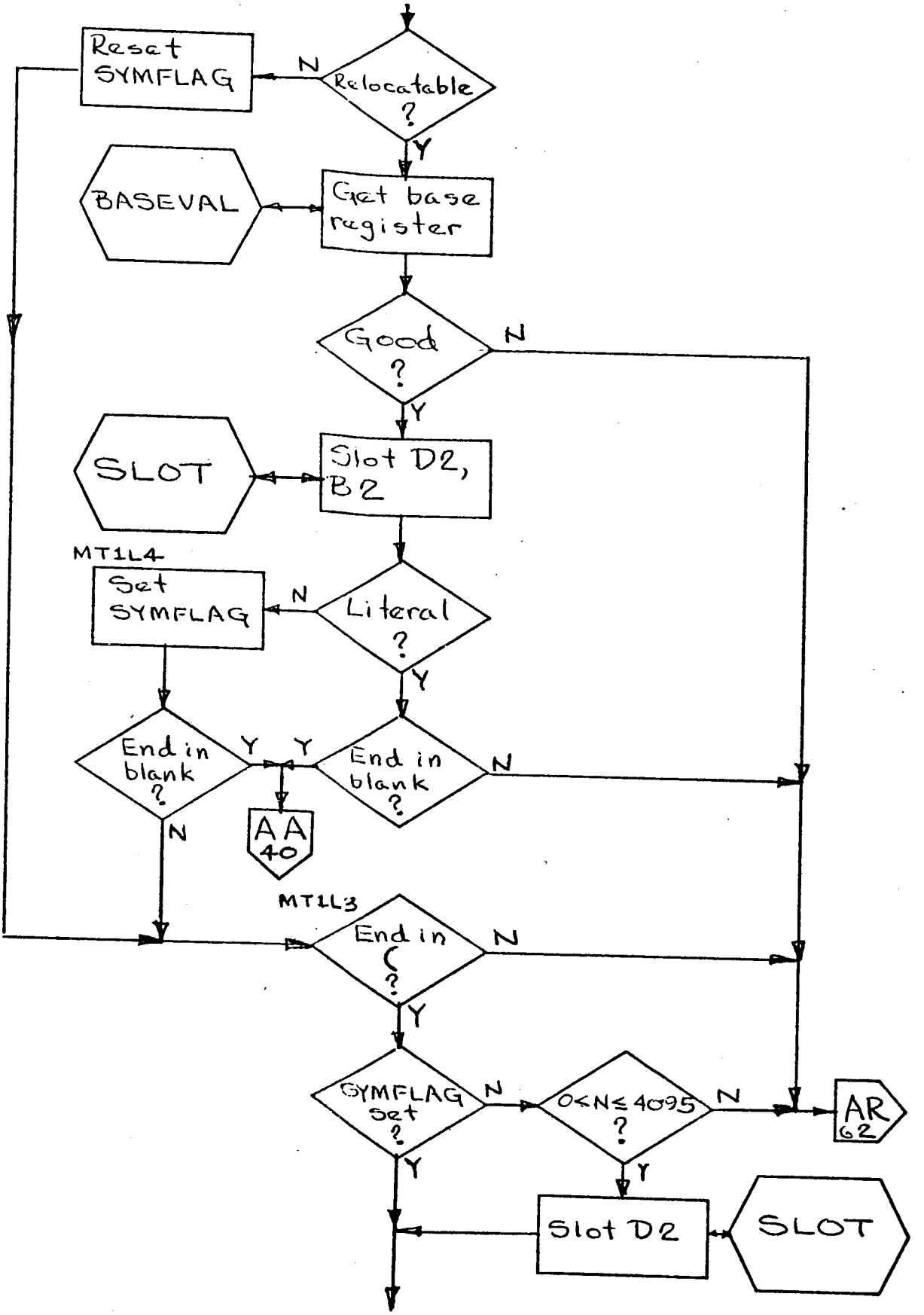

Is multiplier zero ?
Y
N
C
DCALC27
Decrement multiplier
DCALC24
Update boundary
DCALC26
Update location counter
DCALC50
Update symbol table
SYMFIX
Pass 1 ?
Multiplier zero ?
N
Y
N
Y
AN
DCALC47
Set flag for not done
Set flag for done
DCALC28
Any errors
Any processing done ?
N
Y
Y
N
IPRINT = 3
IPRINT = 4
IPRINT = 6
DCALC30
More processing on card ?
Y
N
RETURN
DCALC31
Set flag for more processing

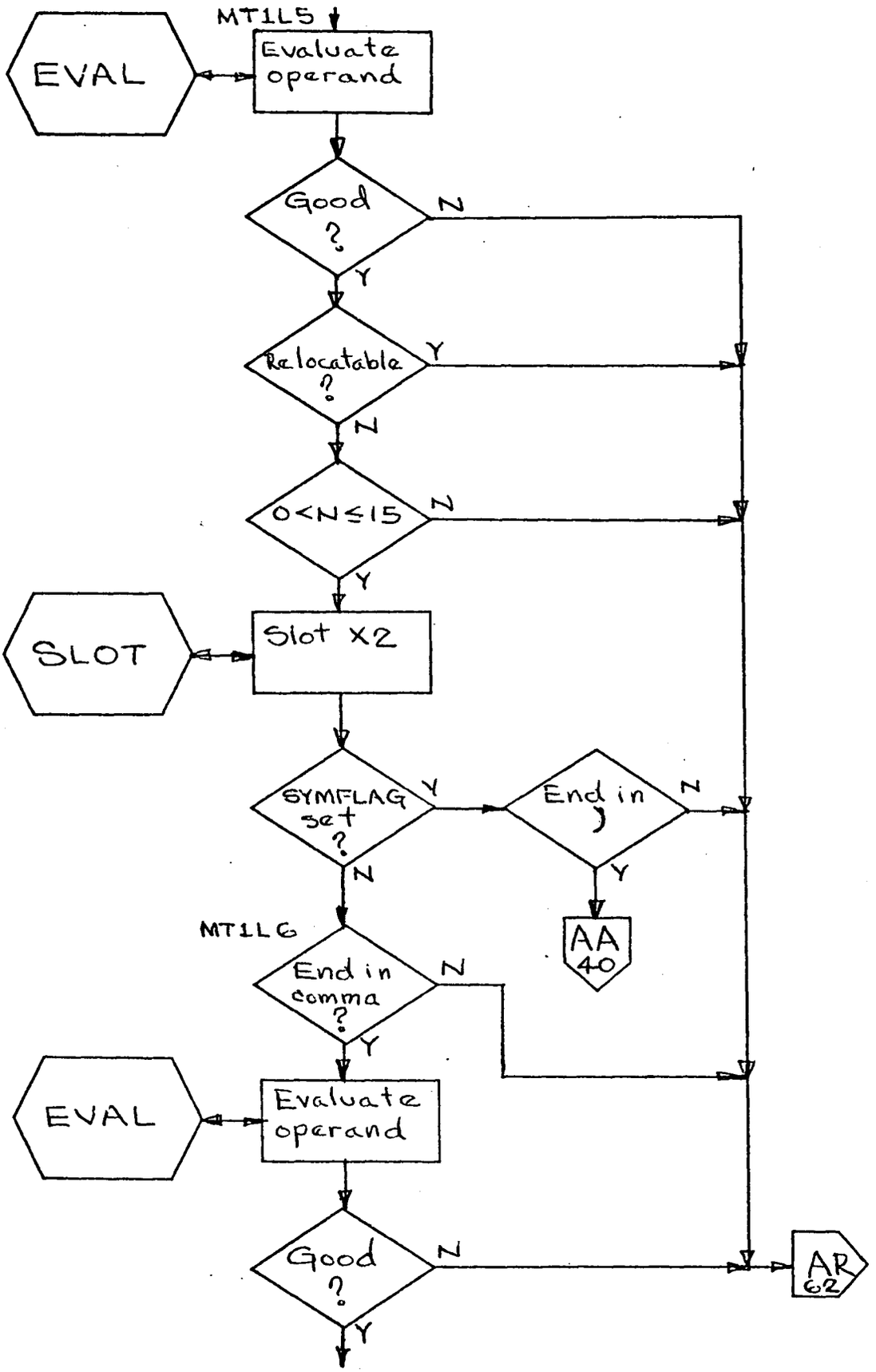

AO
DCALC20
multiplier zero
N
Y
More than 6 characters ?
Y
N
DCALC34
Evaluate constant
CVAL, XVAL, BVAL, FVAL, HVAL, PVAL or ZVAL
AP 54
Good ?
N
Y
DCALC37
Update boundary
FULL or HALF
Update Symbol table
SYMFIX
Update location counter
Slot code
SLOT
Any processing done ?
N
Y
Pass 1 ?
N
Y
D 54
IPRINT = 6
IPRINT = 3
Set flag for more processing
RETURN

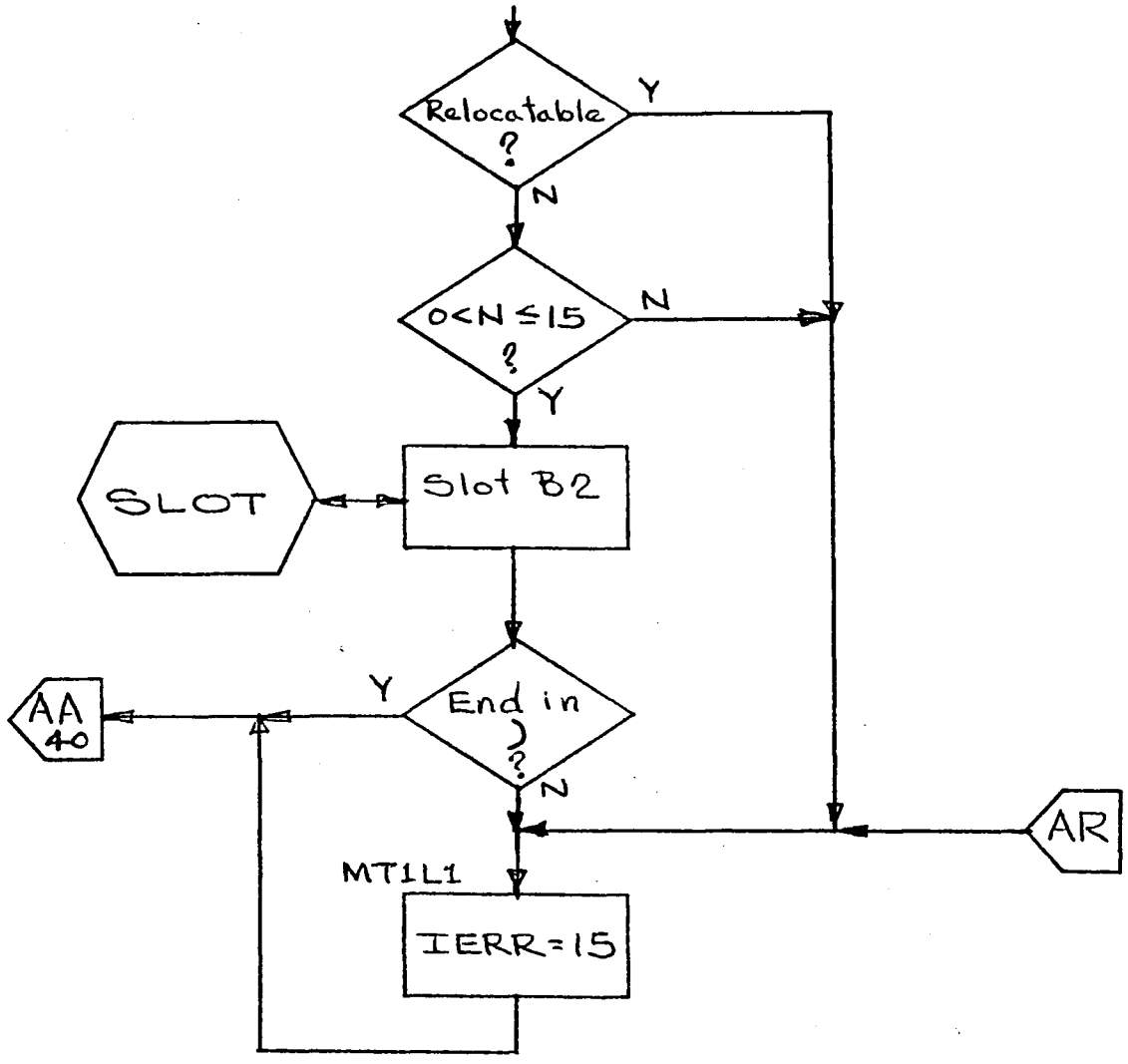

T
MOTO
Evaluate operand
EVAL
Good ?
N
Y
Relocatable ?
Y
N
End in comma ?
N
Y
Y
MOT9
>15 ?
Y
N
SLOT
Slot R1
EVAL
Evaluate operand
Good ?
N
AQ 58
Y

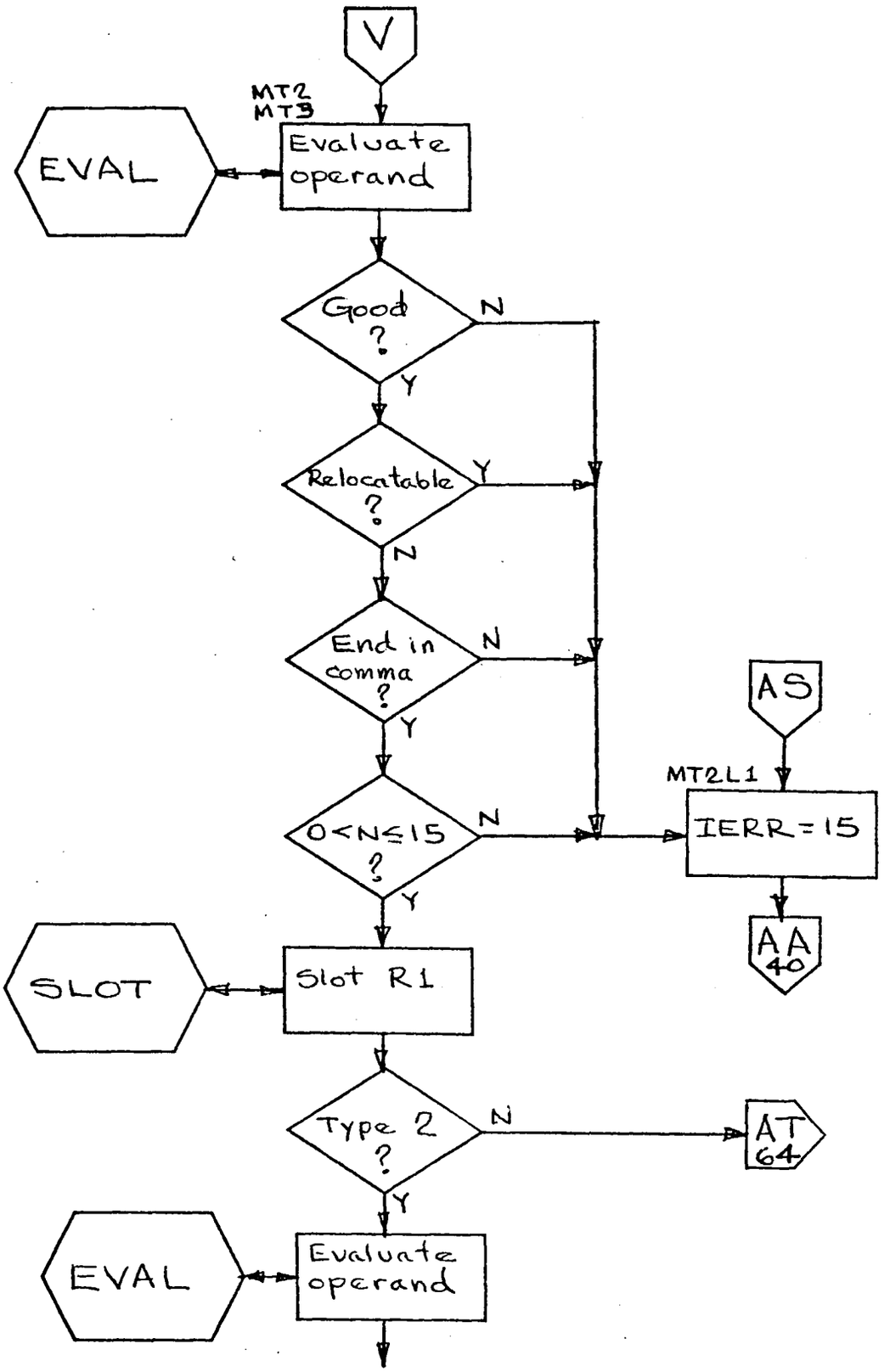

Relocatable ?
Y
N
>15 ?
Y
N
End in blank ?
N
Y
AQ
MOTO1
SLOT
Slot R2
IERR=15
AA
40

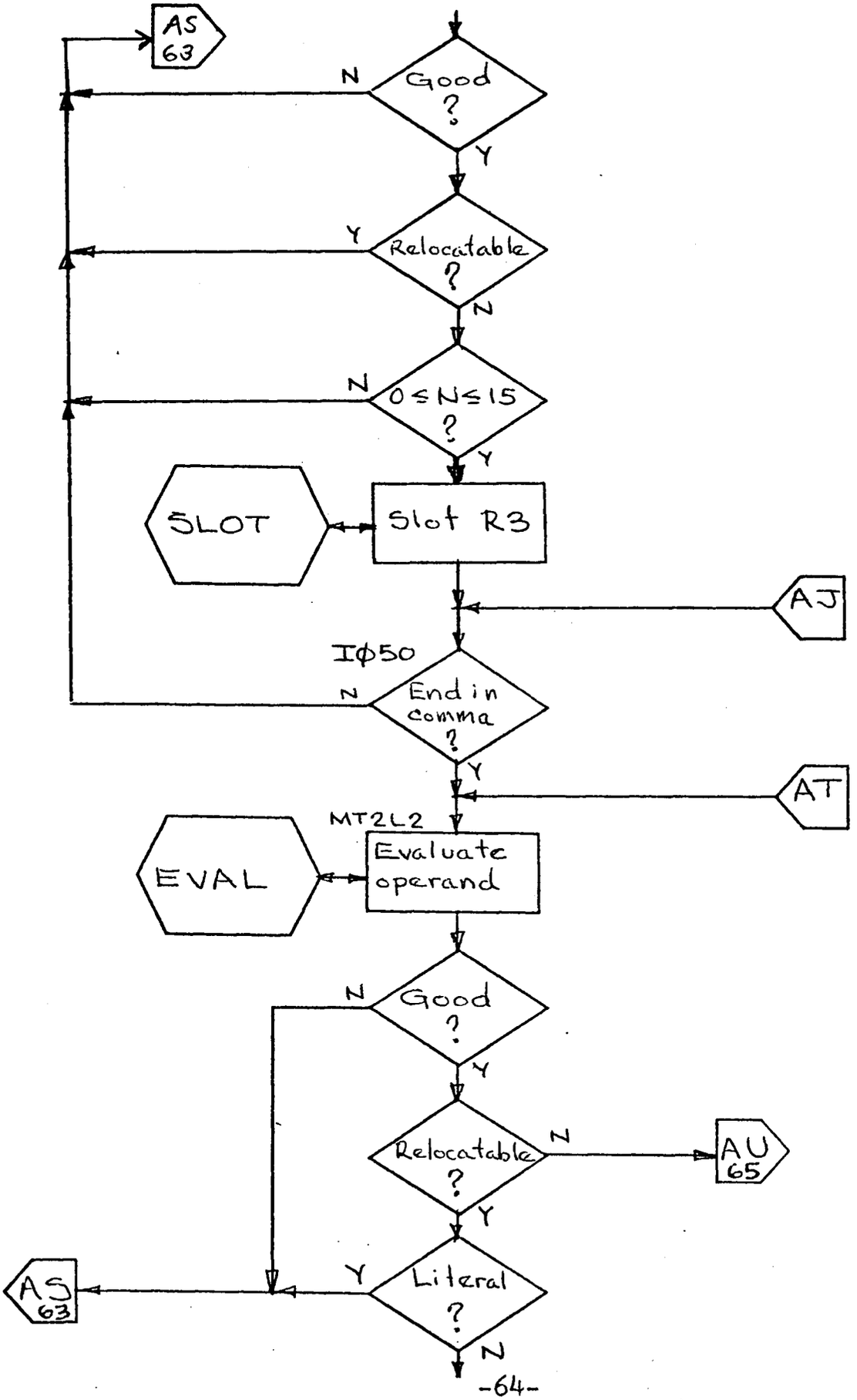

U
MT1
EVAL
Evaluate operand
Good ?
N
Y
Relocatable ?
Y
N
Z
MT10
0<N<15 ?
N
Y
SLOT
Slot R1
End in comma ?
N
Y
EVAL
Evaluate operand
Good ?
N
AR 62

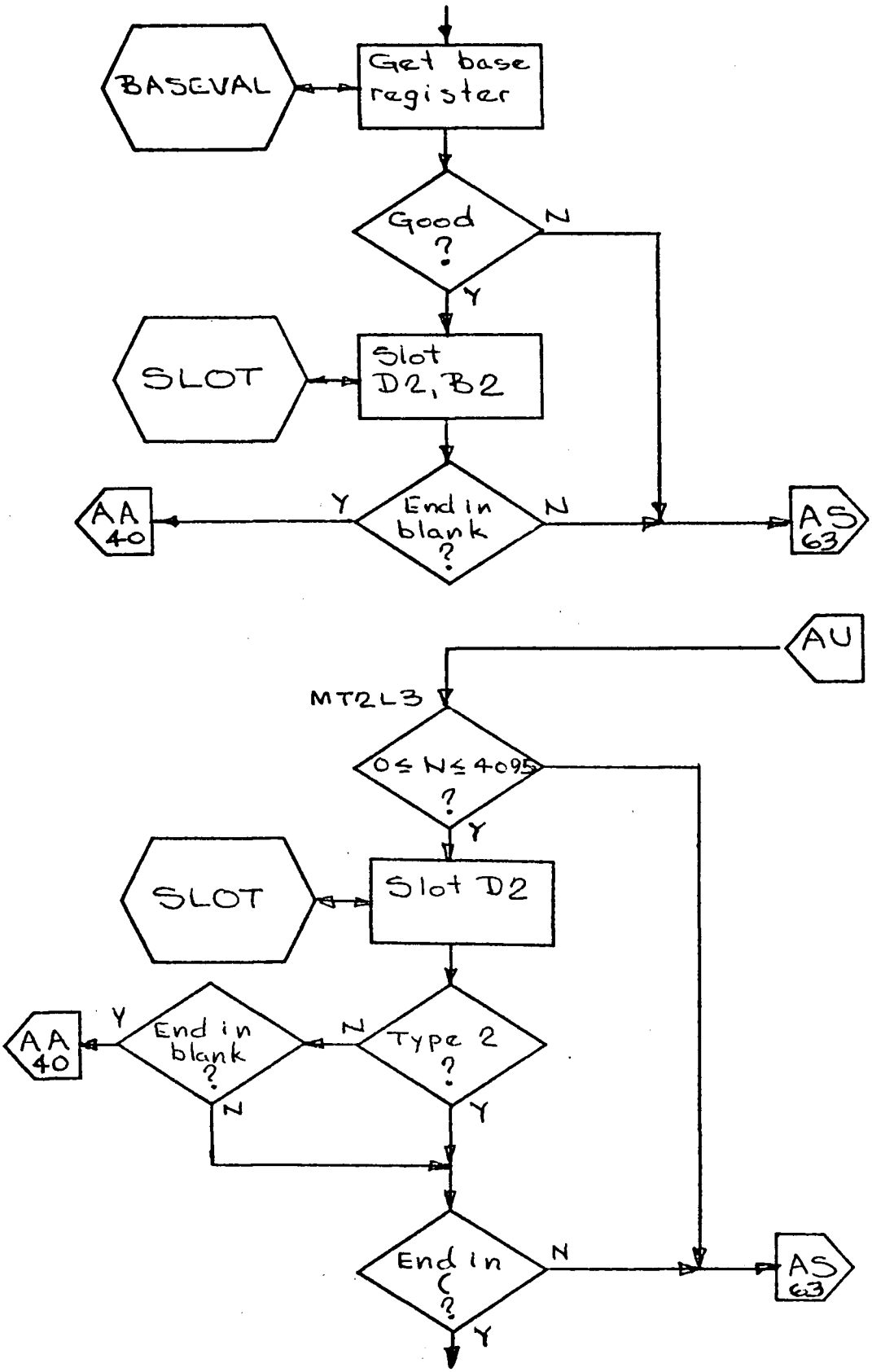

Resat SYMFLAG
N
Relocatable ?
Y
BASEVAL
Get base register
Good ?
N
Y
SLOT
Slot D2, B2
MT1L4
Set SYMFLAG
N
Literal ?
Y
End in blank ?
Y
Y
End in blank ?
N
N
AA 40
MT1L3
End in (?
N
Y
SYMFLAG Set ?
N
0<N≤4095 ?
N
AR 62
Y
Y
Slot D2
SLOT

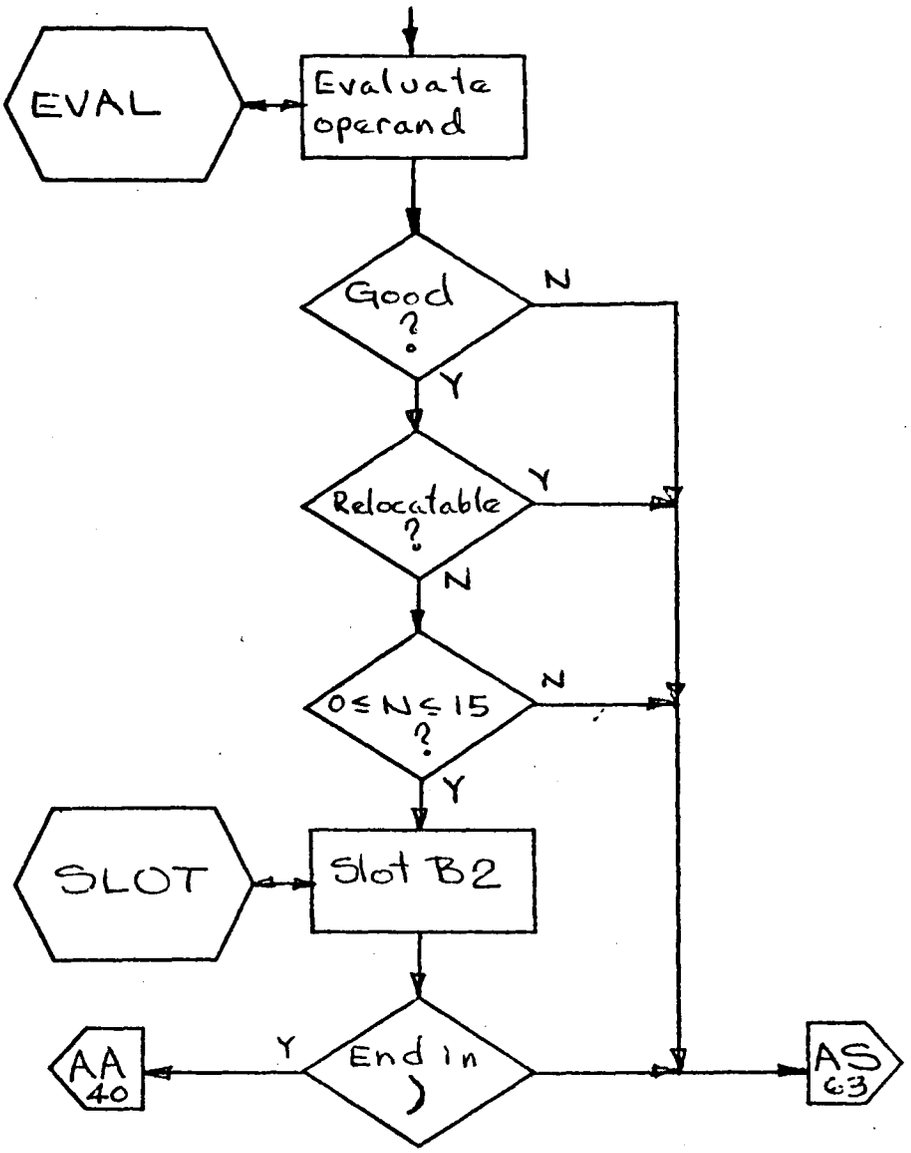

MT1L5
EVAL
Evaluate operand
Good ?
N
Y
Relocatable ?
Y
N
0<N≤15
N
Y
SLOT
Slot X2
SYMFLAG set ?
Y
End in)
N
Y
N
AA 40
MT1L6
End in comma ?
N
Y
EVAL
Evaluate operand
Good ?
N
Y
AR 62

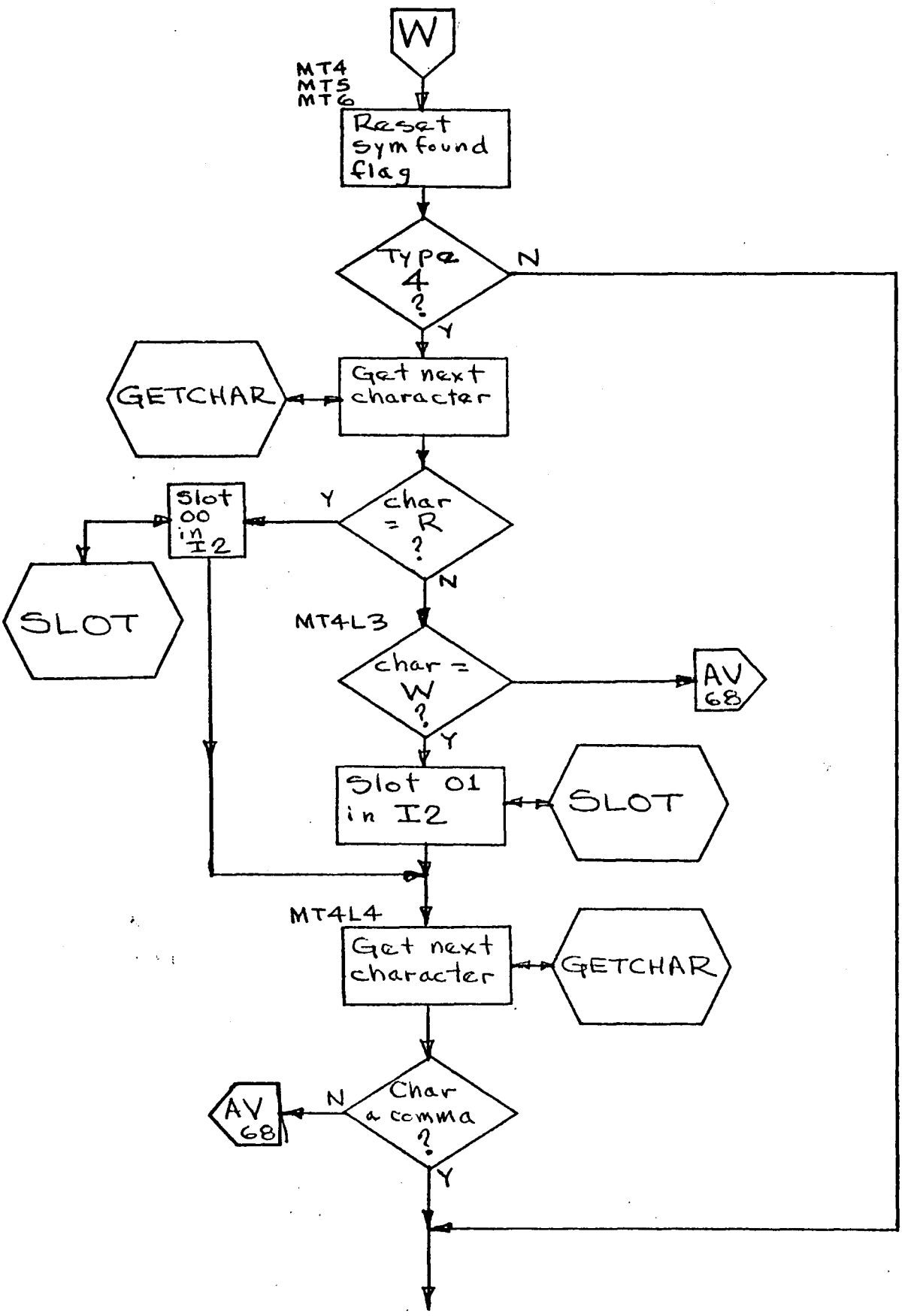

Relocatable ?
Y
N
0<N≤15 ?
N
Y
SLOT
Slot B2
End in ?
Y
AA 40
N
AR
MT1L1
IERR=15

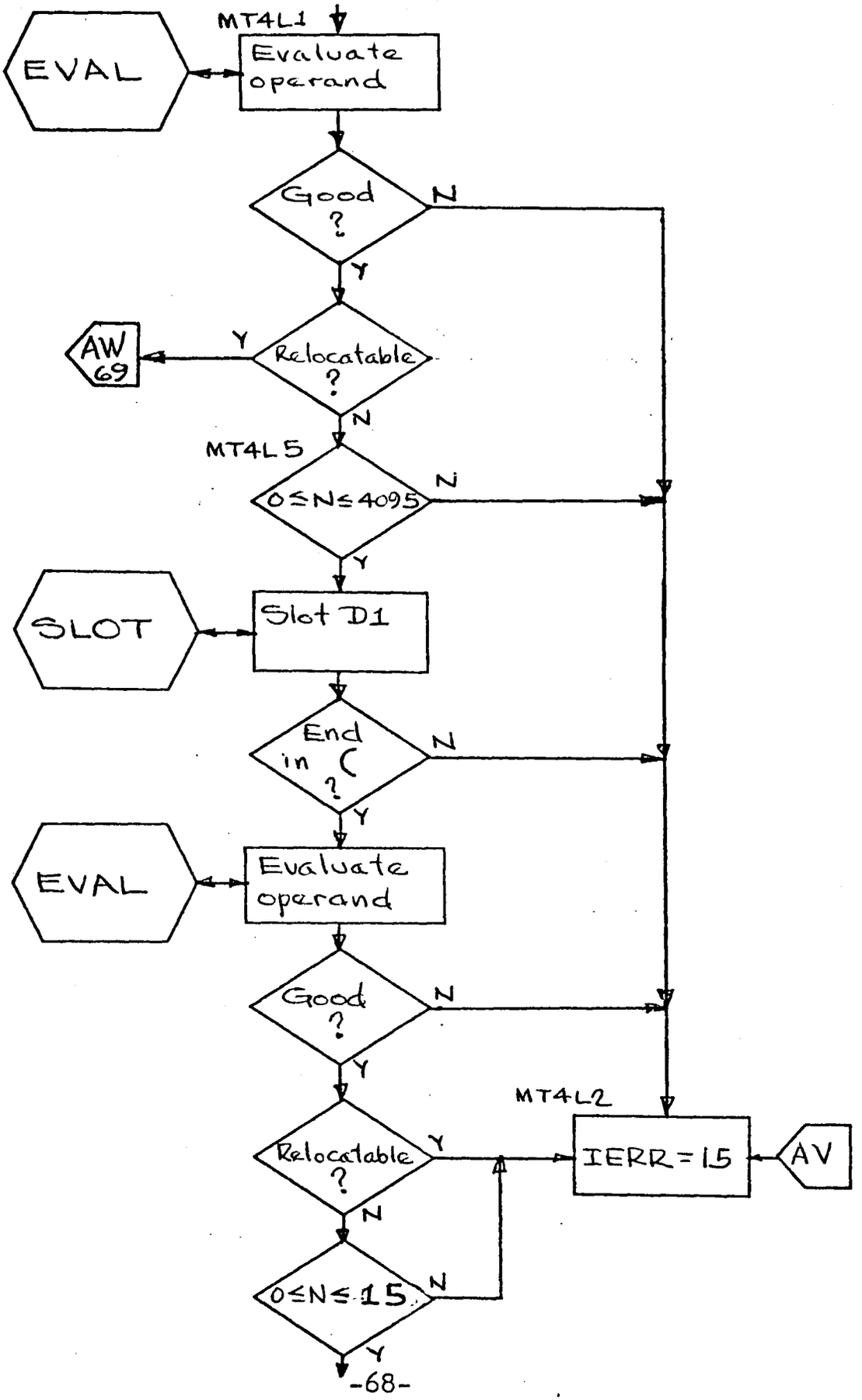

V
MT2
MT3
Evaluate operand
EVAL
Good ?
N
Y
Relocatable ?
Y
N
End in comma ?
N
Y
AS
MT2L1
0<N≤15 ?
N
IERR=15
Y
AA 40
SLOT
Slot R1
Type 2 ?
N
AT 64
Y
EVAL
Evaluate operand

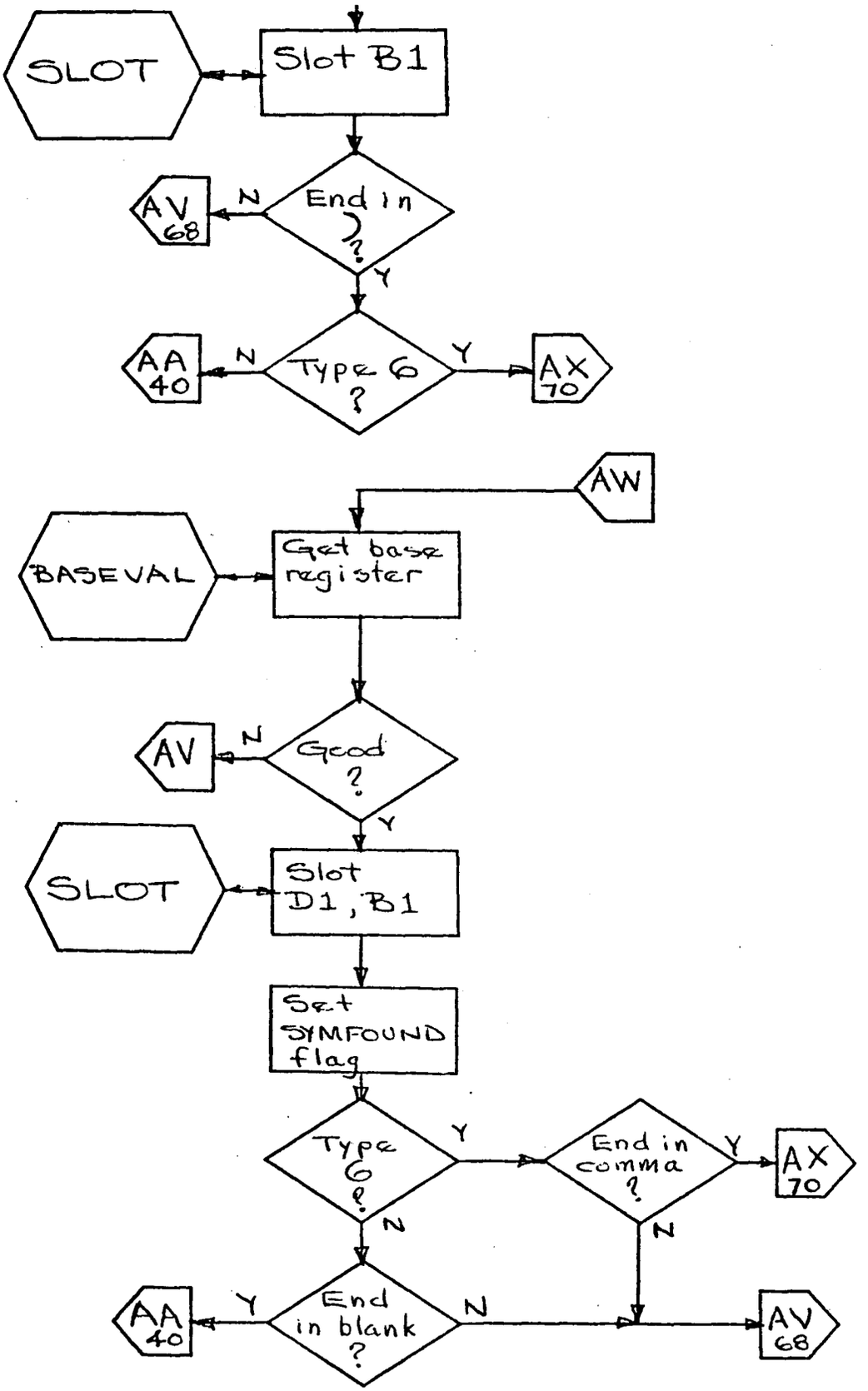

AS 63
Good ?
N
Y
Relocatable ?
Y
N
0≤N≤15 ?
N
Y
SLOT
Slot R3
AJ
IΦ50
End in comma ?
N
Y
AT
MT2L2
EVAL
Evaluate operand
Good ?
N
Y
Relocatable ?
N
AU 65
Y
Literal ?
Y
AS 63
N

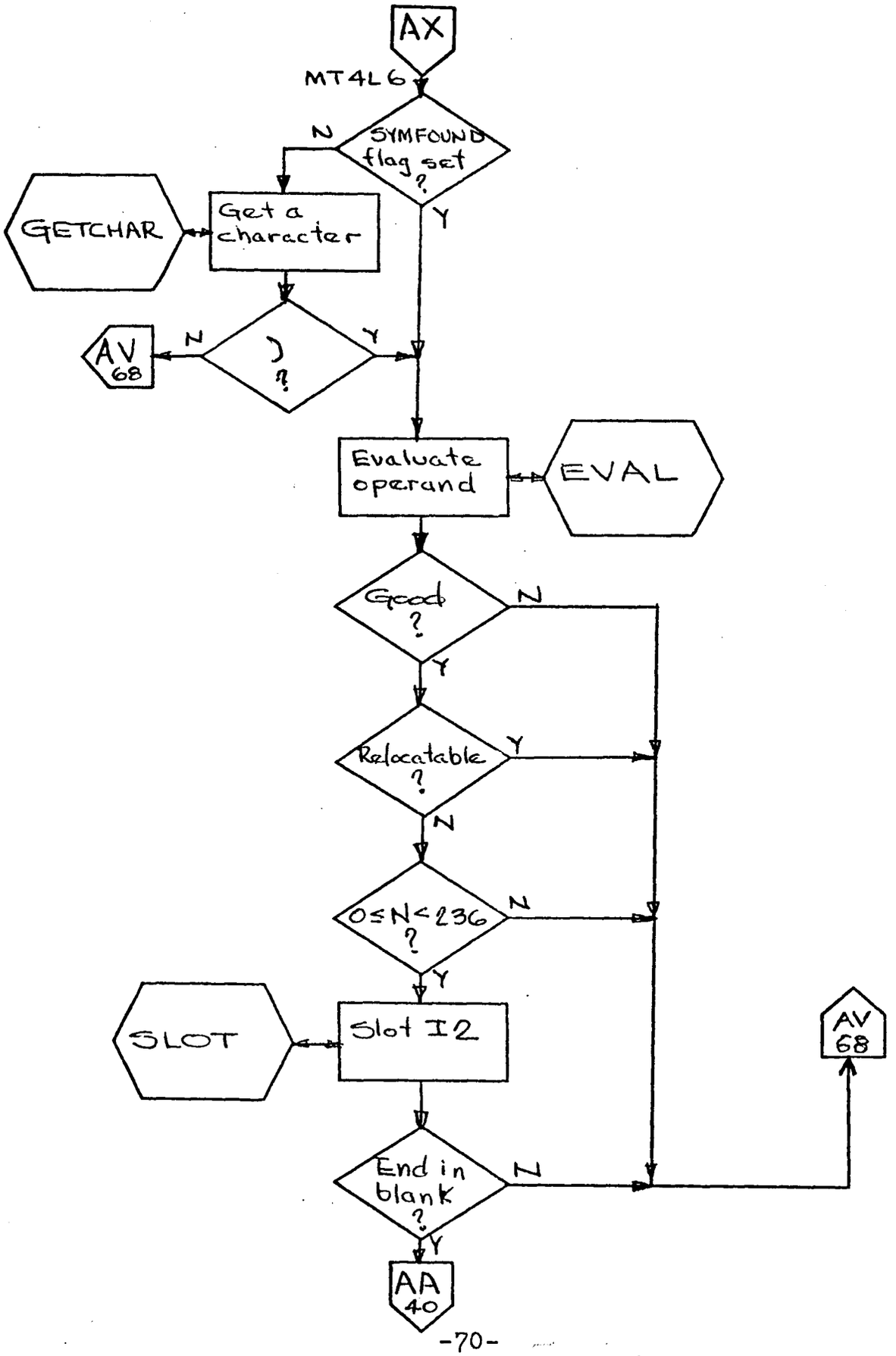

BASEVAL
Get base register
Good ?
N
Y
SLOT
Slot D2, B2
End in blank ?
Y
N
AA 40
AS 63
AU
MT2L3
0 ≤ N ≤ 4095 ?
Y
SLOT
Slot D2
End in blank ?
Y
N
Type 2 ?
N
Y
AA 40
End in C ?
N
Y
AS 63

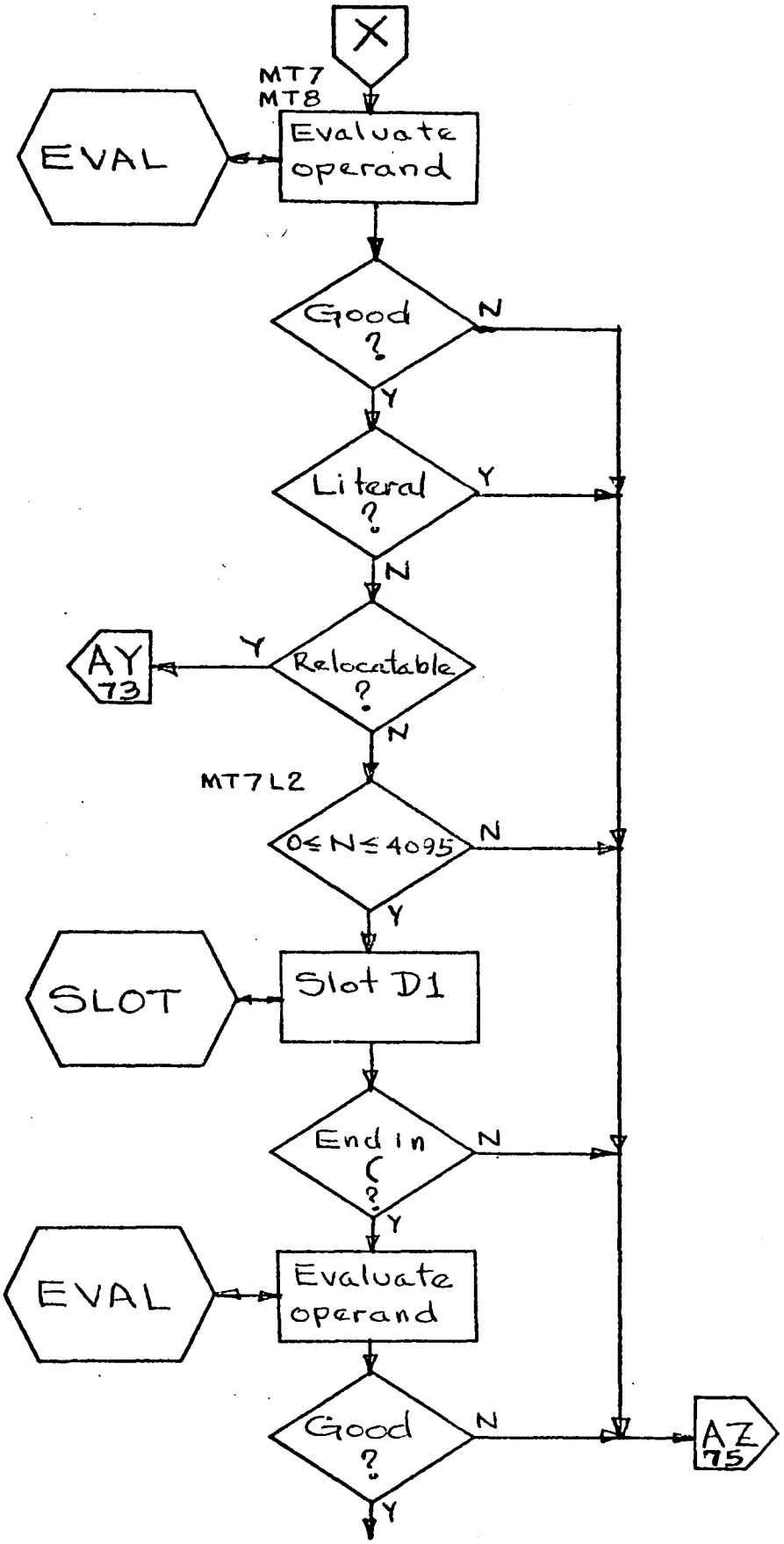

EVAL
Evaluate operand
Good ?
N
Y
Relocatable ?
Y
N
0≤N≤15 ?
N
Y
SLOT
Slot B2
End in)
Y
AA 40
AS 63

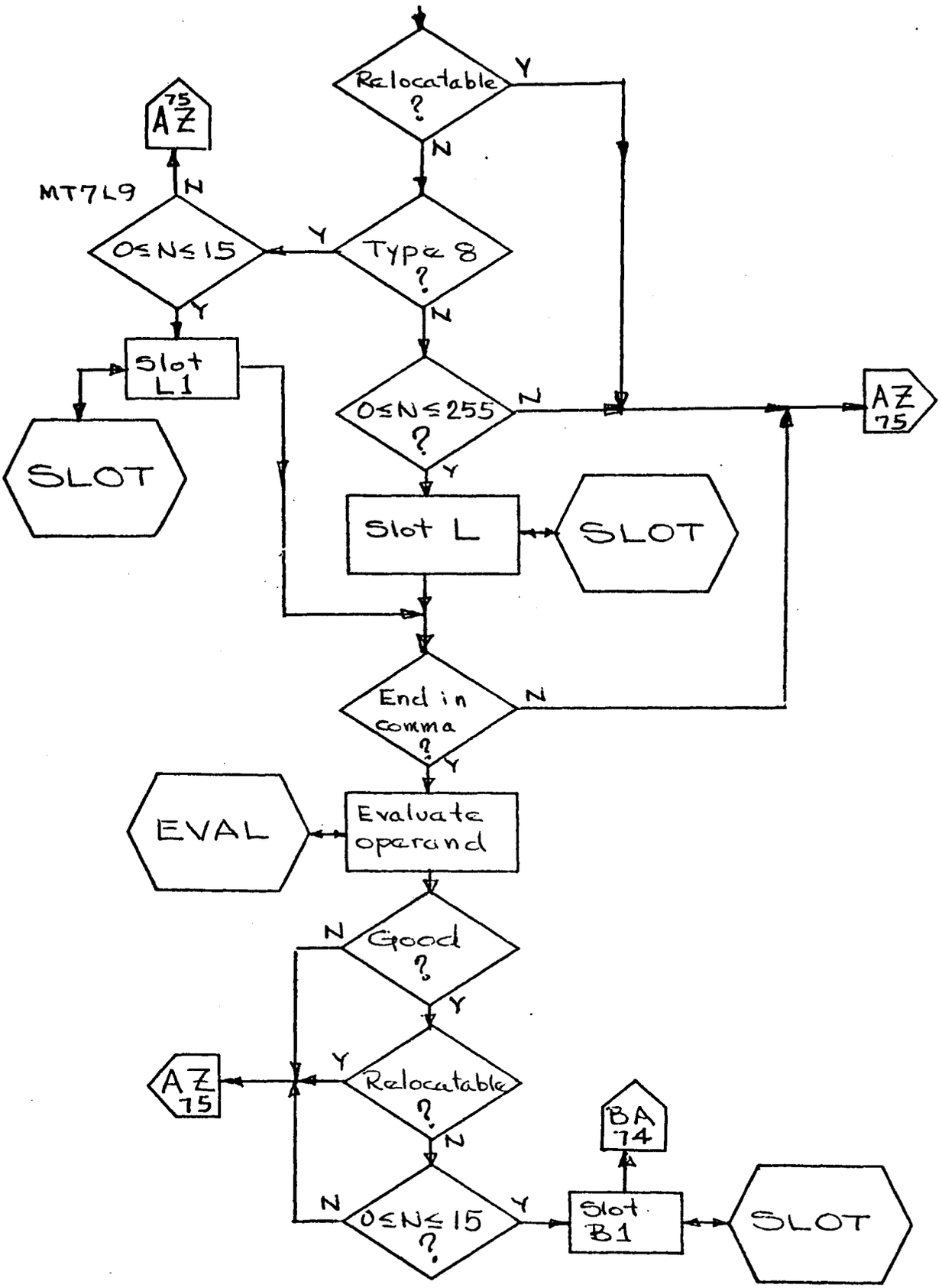

W
MT4
MT5
MT6
Reset sym found flag
Type 4 ?
N
Y
GETCHAR
Get next character
Slot 00 in I2
Y
char = R ?
N
SLOT
MT4L3
char = W ?
AV 68
Y
Slot 01 in I2
SLOT
MT4L4
Get next character
GETCHAR
Char a comma ?
AV 68
N
Y

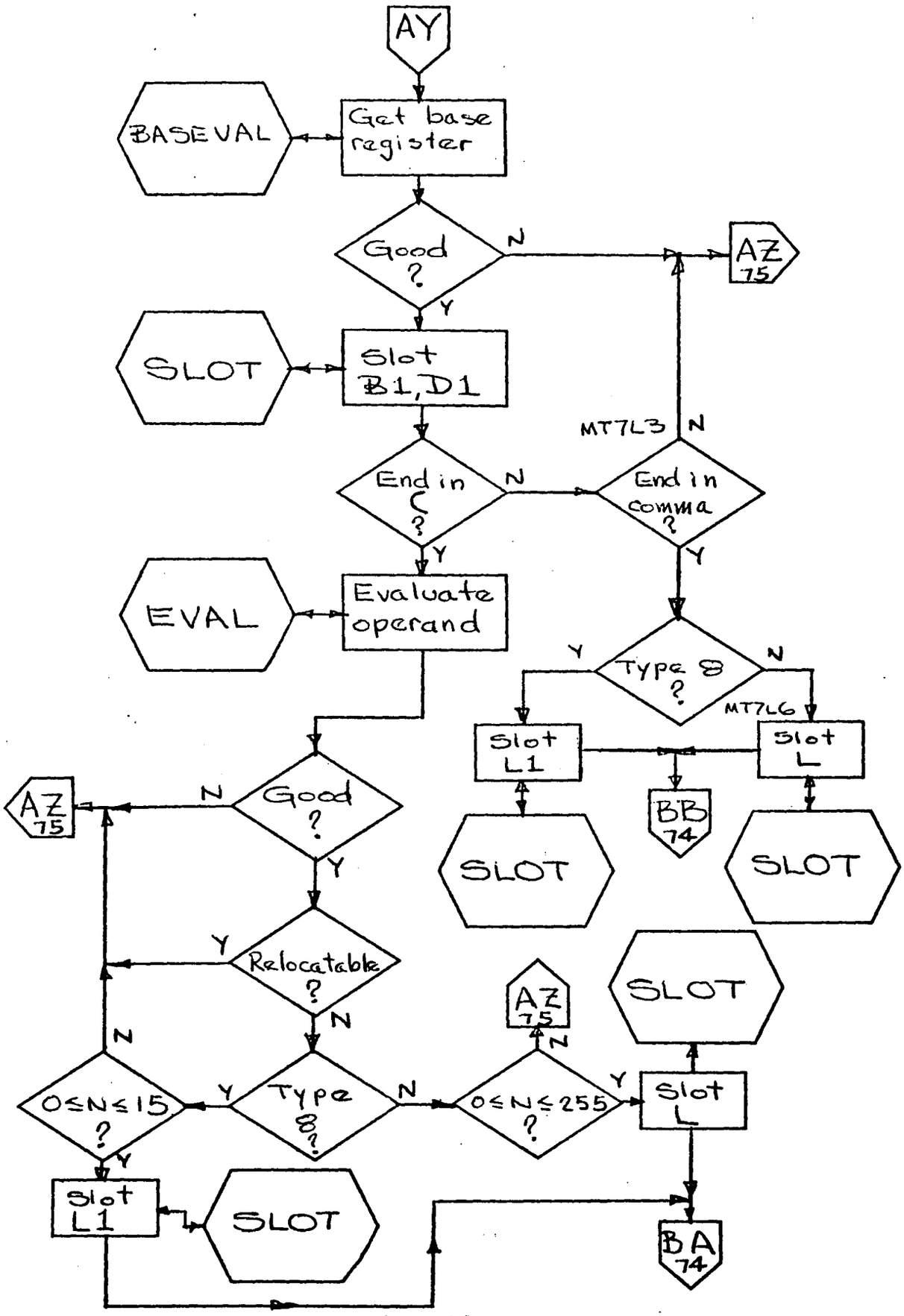

MT4L1
EVAL
Evaluate operand
Good ?
N
Y
AW 69
Y
Relocatable ?
N
MT4L5
0≤N≤4095
N
Y
SLOT
Slot D1
End in (?
N
Y
EVAL
Evaluate operand
Good ?
N
Y
MT4L2
Relocatable ?
Y
IERR=15
AV
N
0≤N≤15
N
Y

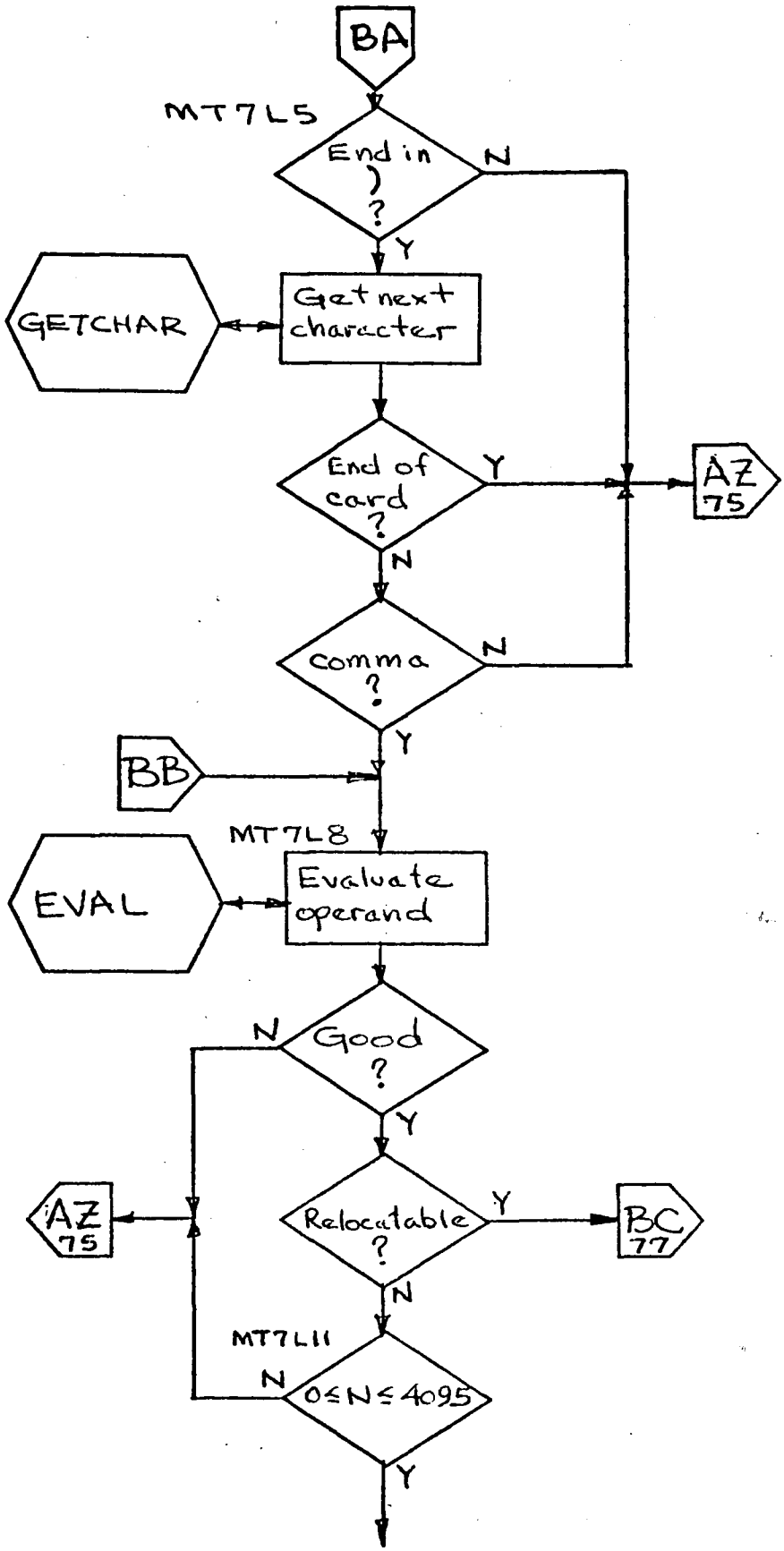

SLOT
Slot B1
End in)?
N
AV 68
Y
Type 6 ?
N
AA 40
Y
AX 70
AW
BASEVAL
Get base register
Good ?
N
AV
Y
SLOT
Slot D1, B1
Set SYMFOUND flag
Type 6 ?
Y
End in comma ?
Y
AX 70
N
N
End in blank ?
Y
AA 40
N
AV 68

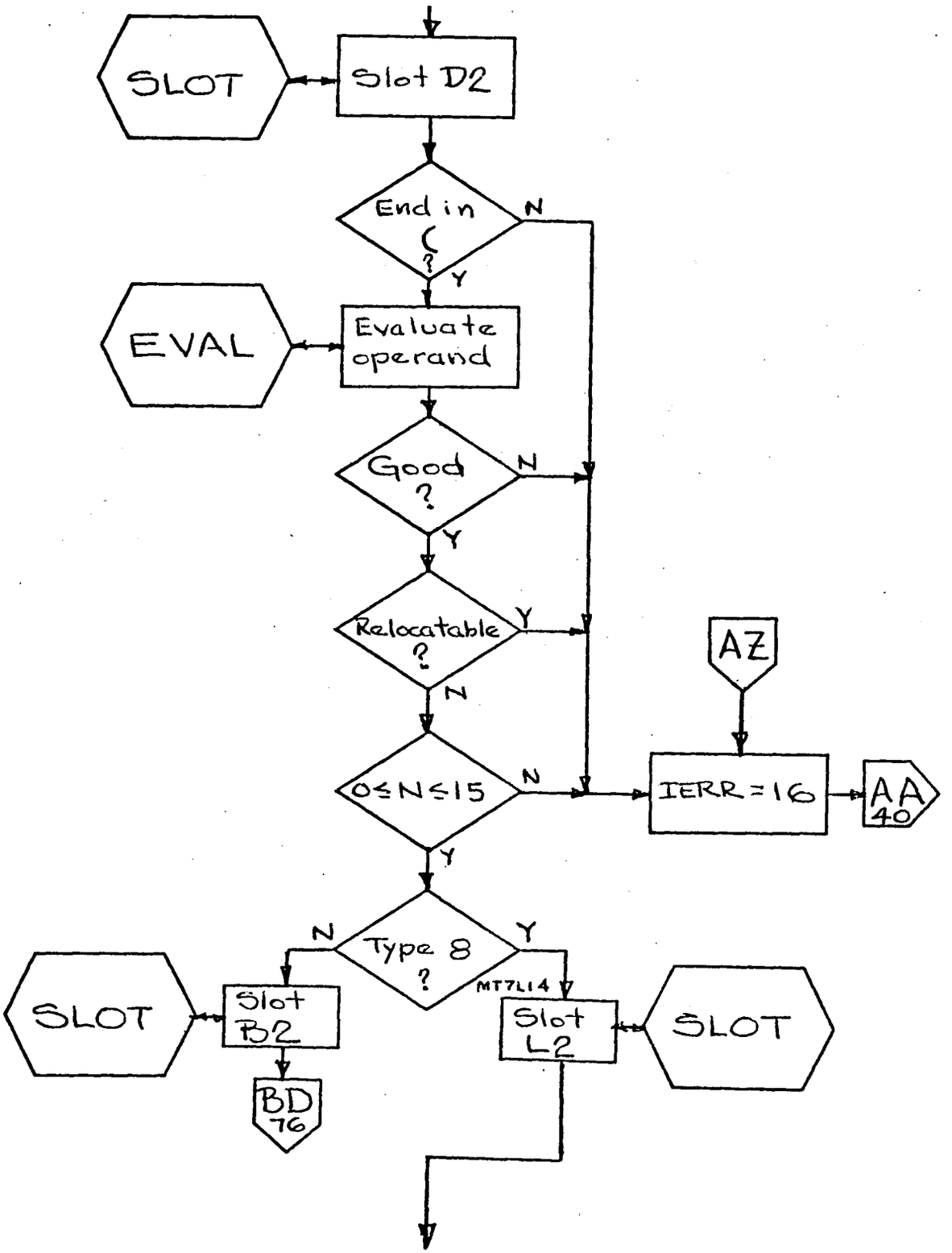

AX
MT4L6
SYMFOUND flag set ?
N
Y
GETCHAR
Get a character
AV 68
N
)
?
Y
Evaluate operand
EVAL
Good ?
N
Y
Relocatable ?
Y
N
0≤N<236 ?
N
Y
SLOT
Slot I2
AV 68
End in blank ?
N
Y
AA 40

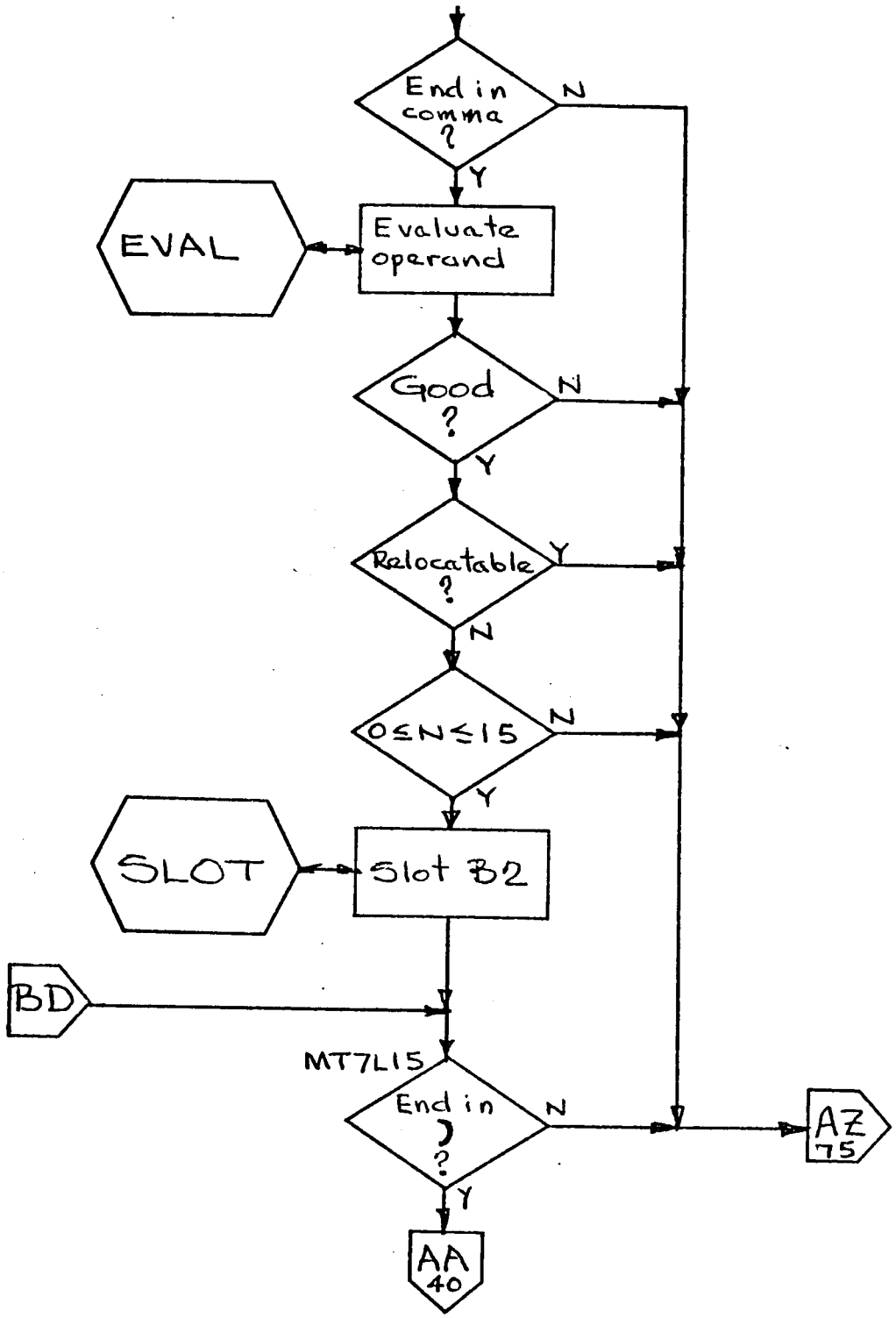

X
MT7
MT8
EVAL
Evaluate operand
Good ?
N
Y
Literal ?
Y
N
AY 73
Y
Relocatable ?
N
MT7L2
0≤N≤4095
N
Y
SLOT
Slot D1
End in C ?
N
Y
EVAL
Evaluate operand
Good ?
N
AZ 75
Y

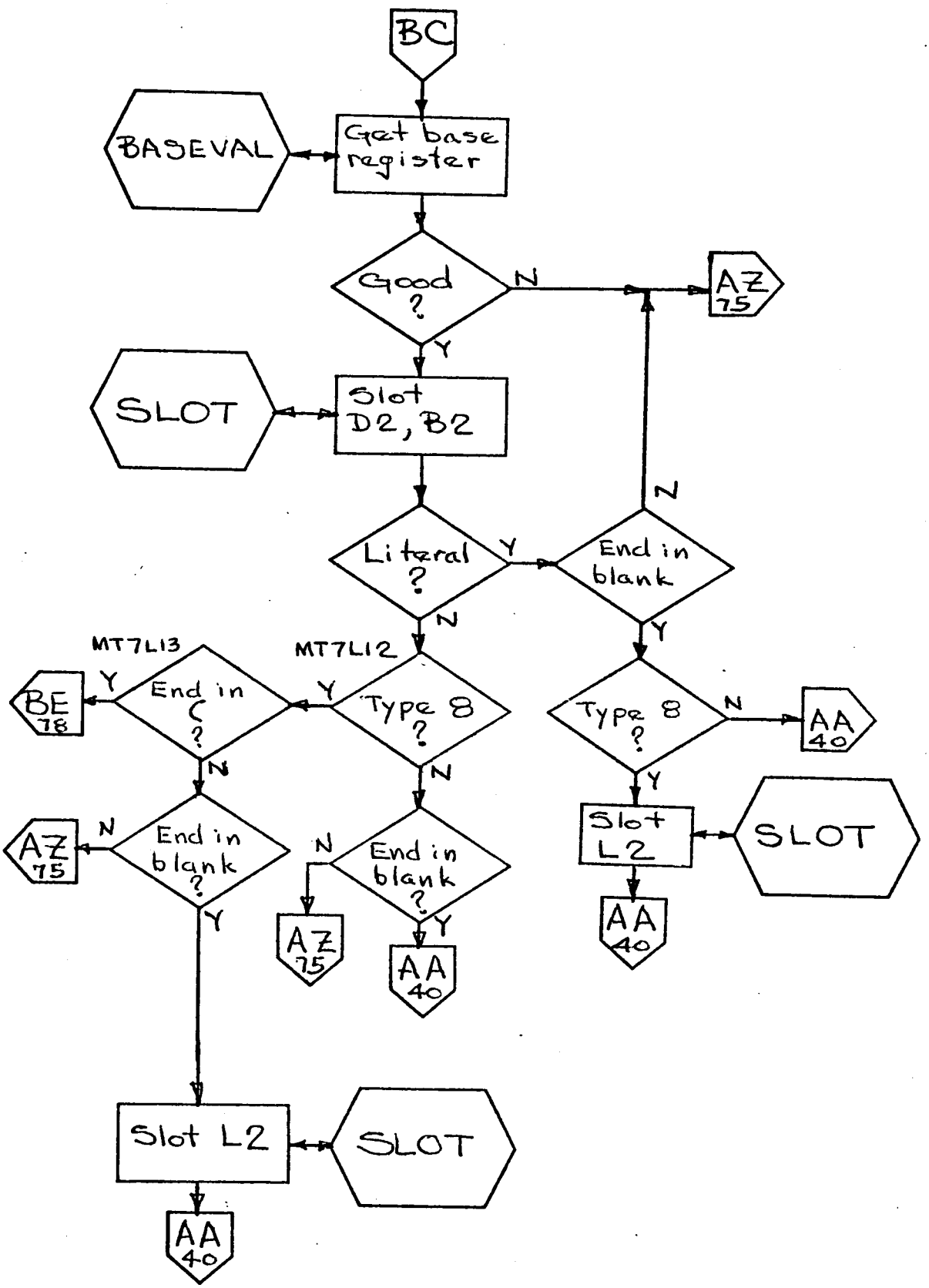

Relocatable ?
Y
N
AZ 75
MT7L9
N
0≤N≤15
Y
Type 8 ?
Y
N
Slot L1
SLOT
0≤N≤255 ?
N
AZ 75
Y
Slot L
SLOT
End in comma ?
N
Y
EVAL
Evaluate operand
Good ?
N
Y
AZ 75
Y
Relocatable ?
N
BA 74
N
0≤N≤15 ?
Y
Slot B1
SLOT

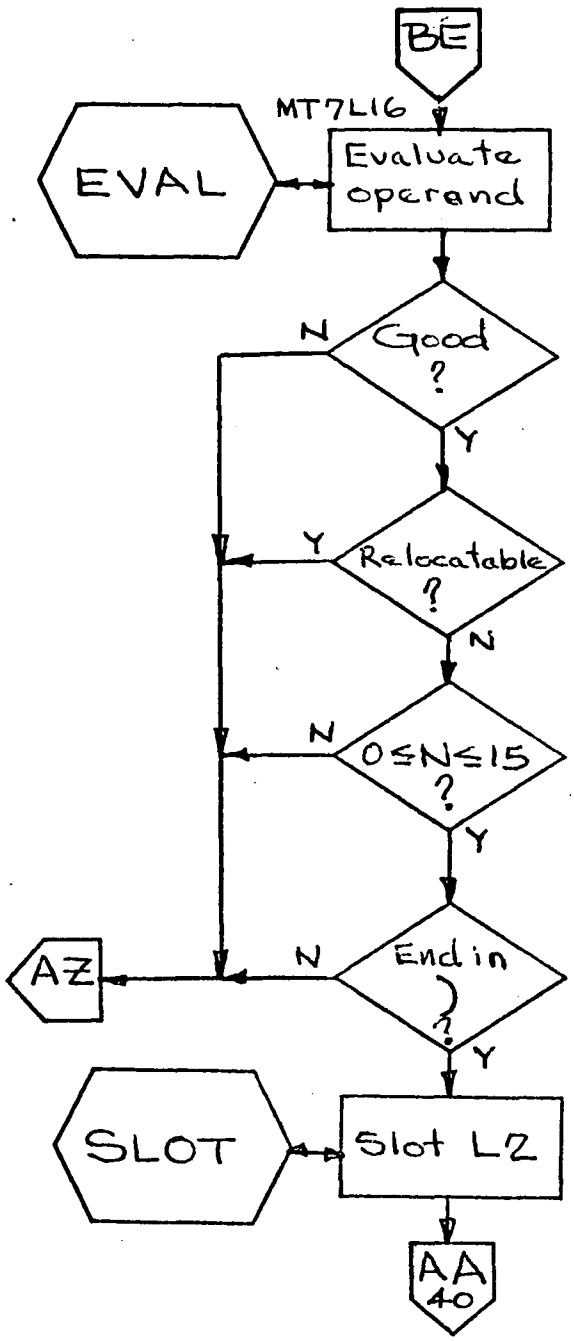

AY
BASEVAL
Get base register
Good ?
N
AZ 75
Y
SLOT
Slot B1,D1
MT7L3
N
End in (?
N
End in comma ?
Y
Y
EVAL
Evaluate operand
Y
Type 8 ?
N
MT7L6
Slot L1
Slot L
AZ 75
N
Good ?
BB 74
SLOT
SLOT
Y
Y
Relocatable ?
N
AZ 75
SLOT
N
N
0≤N≤15 ?
Y
Type 8 ?
N
0≤N≤255 ?
Y
Slot L
Y
Slot L1
SLOT
BA 74

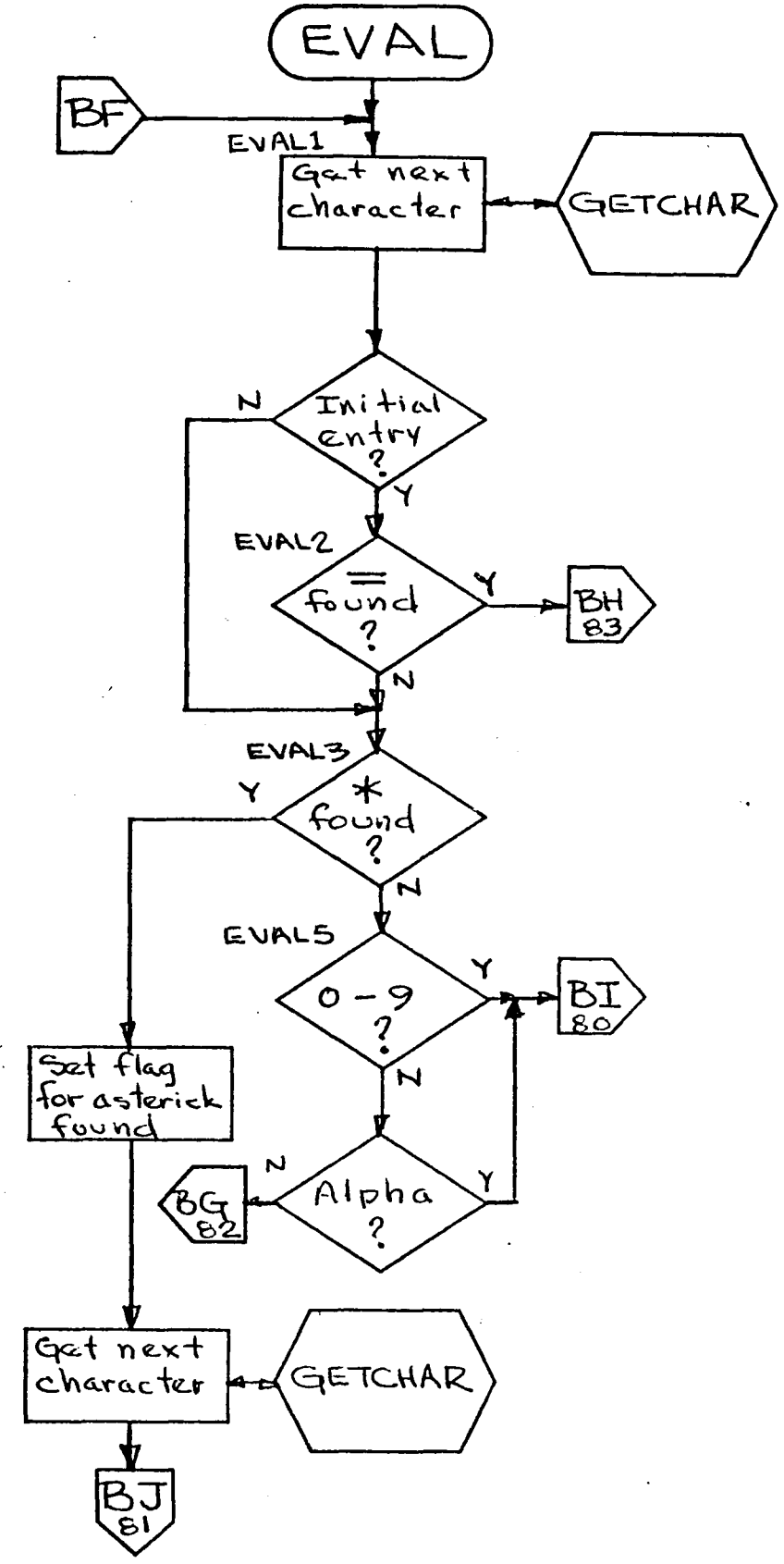

BA
MT7L5
End in)?
N
Y
GETCHAR
Get next character
End of card?
Y
AZ 75
N
Comma?
N
Y
BB
MT7L8
EVAL
Evaluate operand
N
Good?
Y
AZ 75
Relocatable?
Y
BC 77
N
MT7L11
N
0≤N≤4095
Y

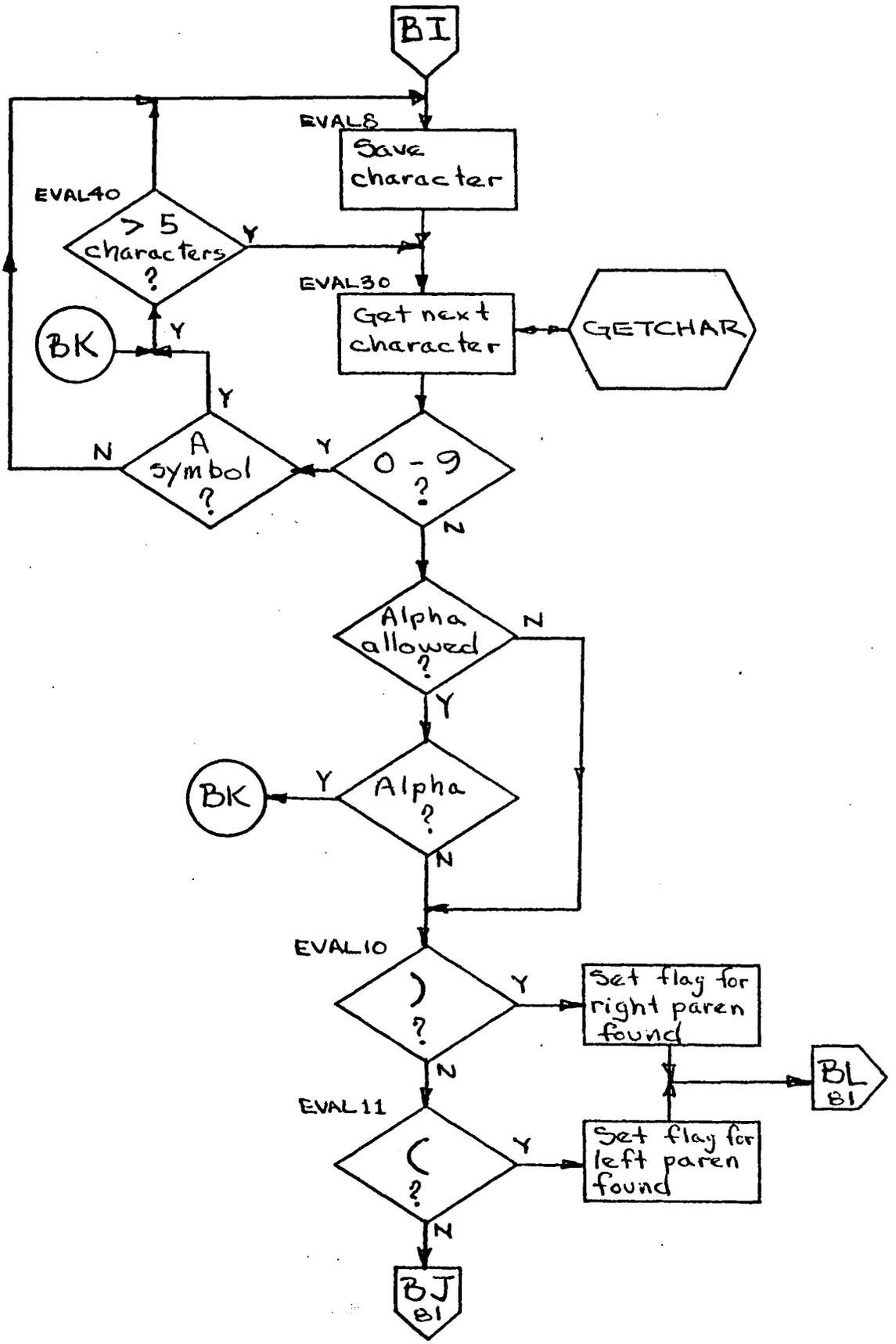

SLOT
Slot D2
End in (?
N
Y
EVAL
Evaluate operand
Good ?
N
Y
Relocatable ?
Y
N
AZ
0≤N≤15
N
IERR=16
AA 40
Y
Type 8 ?
N
Y
MT7L14
SLOT
Slot B2
BD 76
Slot L2
SLOT

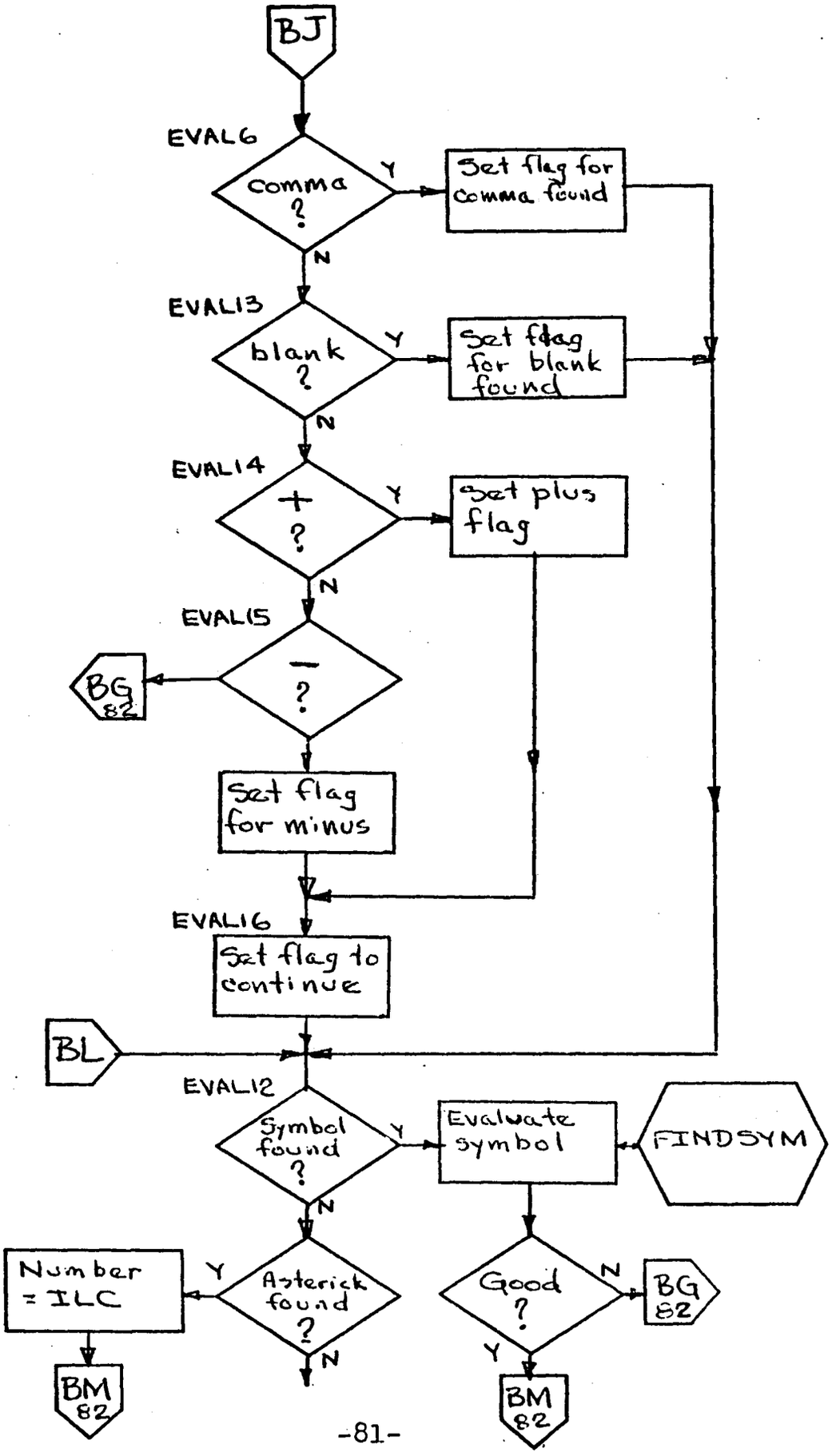

End in comma ?
N
Y
EVAL
Evaluate operand
Good ?
N
Y
Relocatable ?
Y
N
0≤N≤15
N
Y
SLOT
Slot B2
BD
MT7L15
End in) ?
N
Y
AZ 75
AA 40

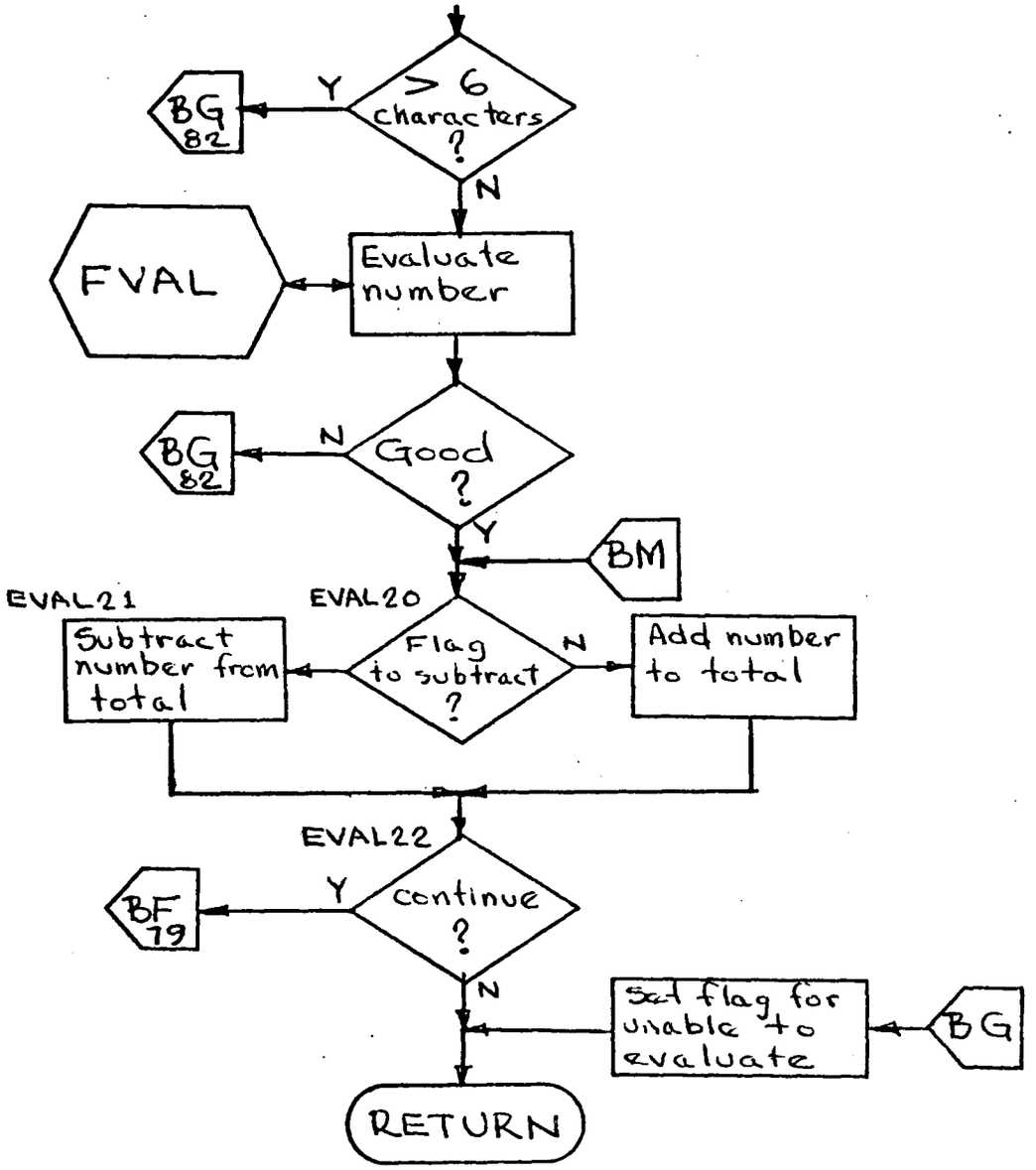

BC
BASEVAL
Get base register
Good ?
N
AZ 75
Y
SLOT
Slot D2, B2
Literal ?
Y
End in blank
N
N
Y
MT7L13
MT7L12
BE 78
Y
End in C ?
Y
Type 8 ?
Type 8 ?
N
AA 40
N
N
Y
AZ 75
N
End in blank ?
N
End in blank ?
Slot L2
SLOT
Y
Y
AZ 75
AA 40
AA 40
Slot L2
SLOT
AA 40

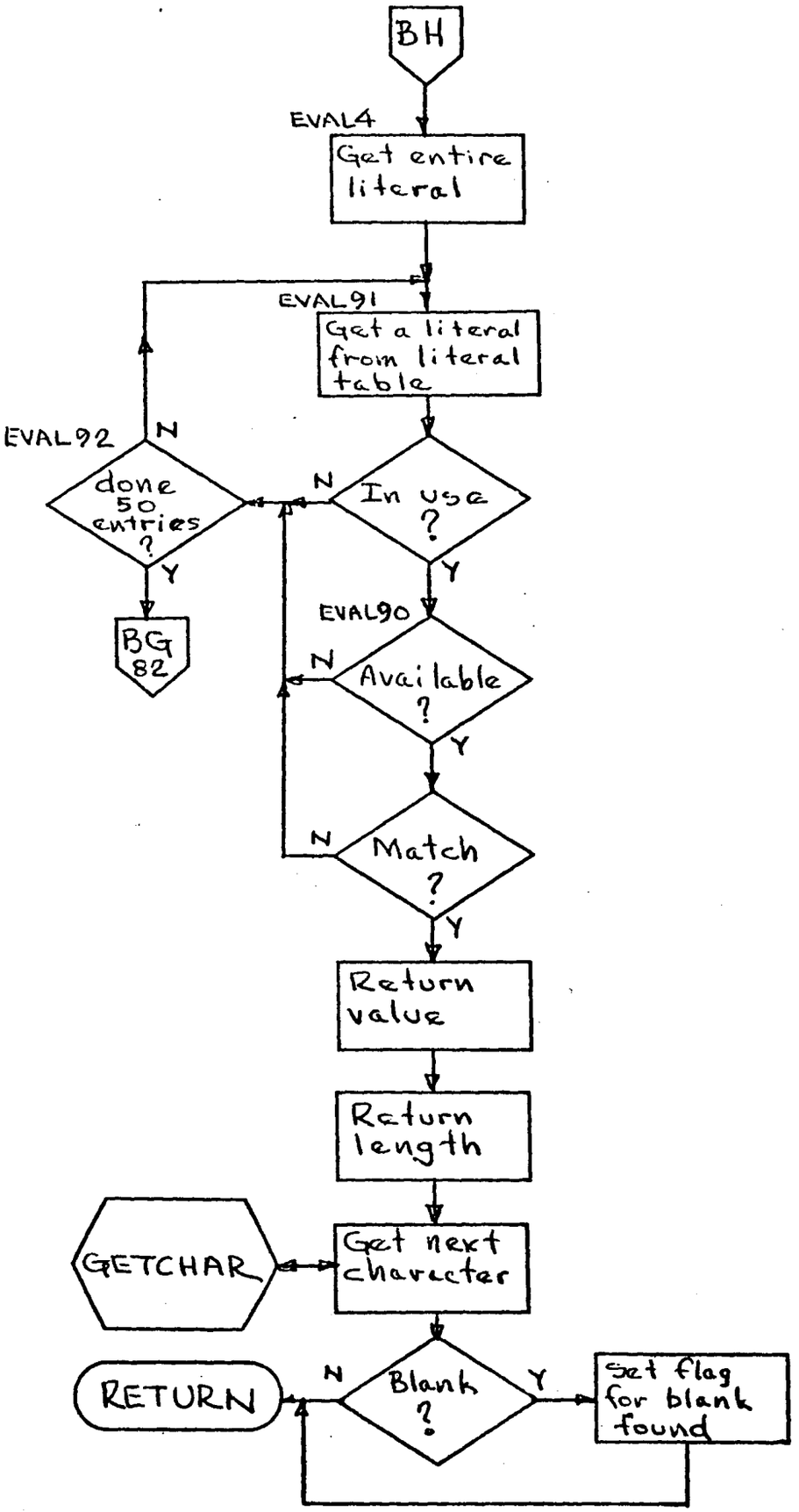

BE
MT7L16
EVAL
Evaluate operand
Good ?
N
Y
Relocatable ?
Y
N
0≤N≤15 ?
N
Y
End in)?
N
Y
AZ
SLOT
Slot LZ
AA 40

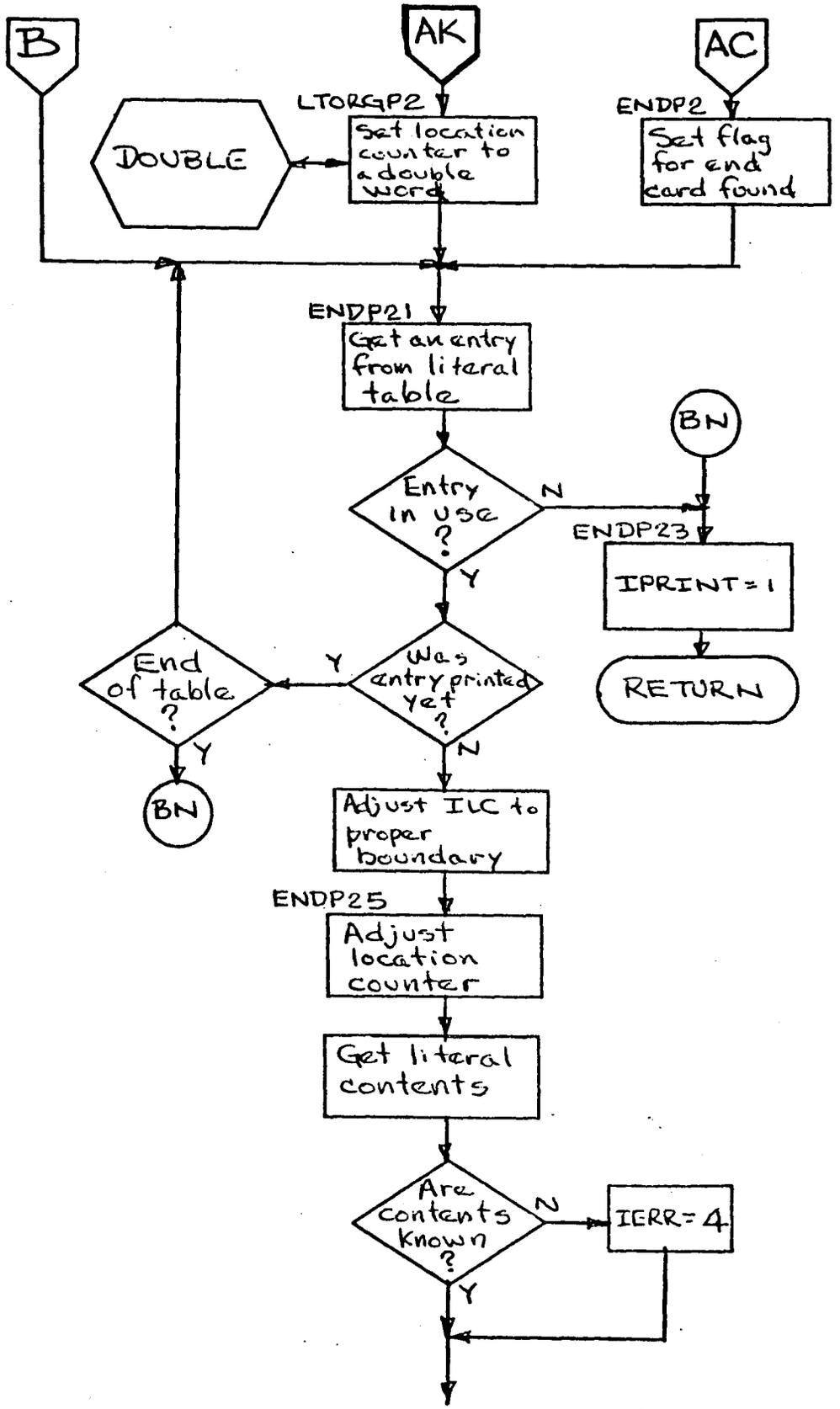

EVAL
BF
EVAL1
Get next character
GETCHAR
N
Initial entry ?
Y
EVAL2
= found ?
Y
BH 83
N
EVAL3
Y
* found ?
N
EVAL5
0-9 ?
Y
BI 80
N
Set flag for asterick found
N
Alpha ?
Y
BG 82
Get next character
GETCHAR
BJ 81

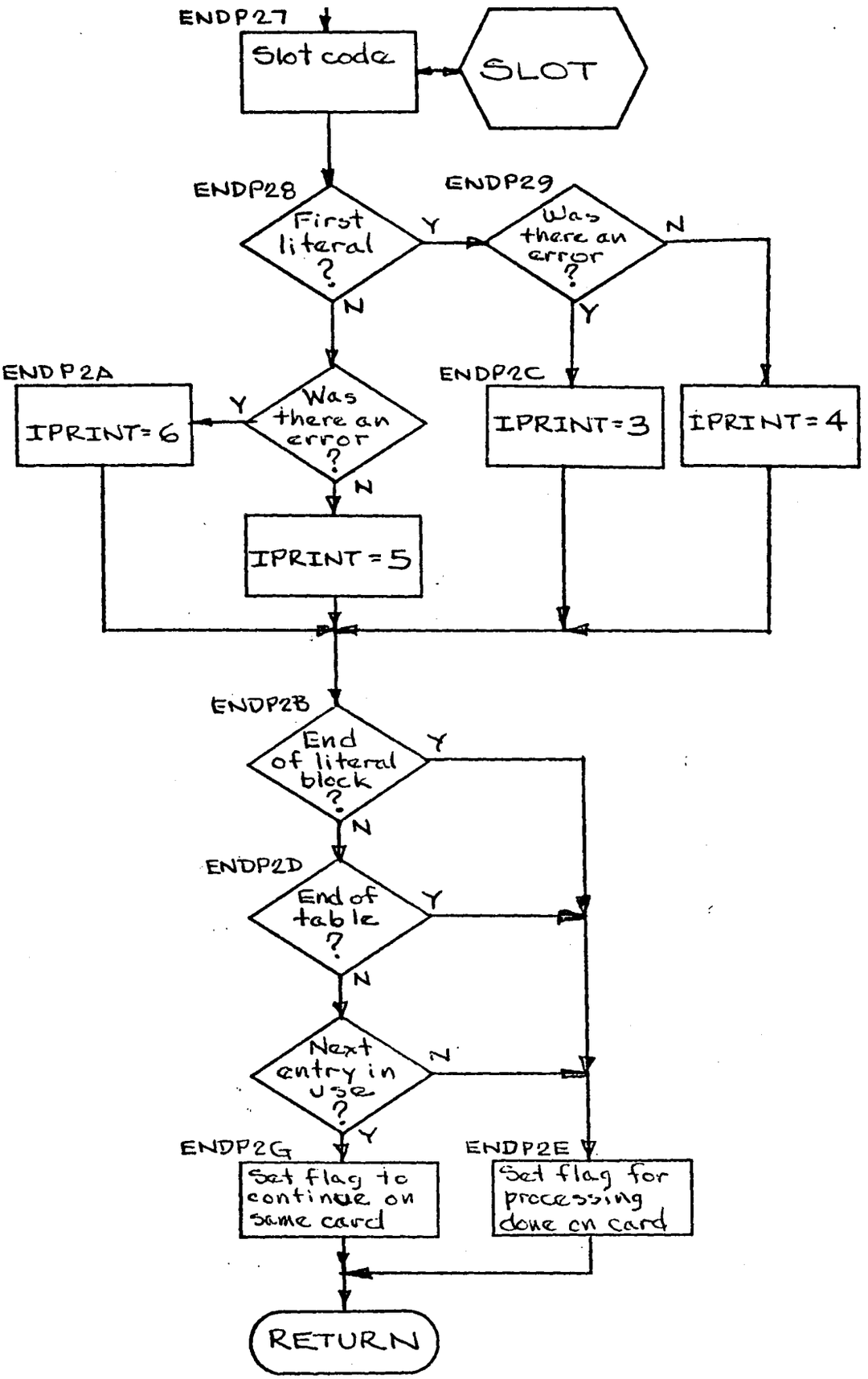

BI
EVAL8
Save character
EVAL40
> 5 characters ?
Y
EVAL30
Get next character
GETCHAR
BK
Y
Y
A symbol ?
N
Y
0 - 9 ?
N
Alpha allowed ?
N
Y
BK
Y
Alpha ?
N
EVAL10
) ?
Y
Set flag for right paren found
N
BL BI
EVAL11
(?
Y
Set flag for left paren found
N
BJ BI

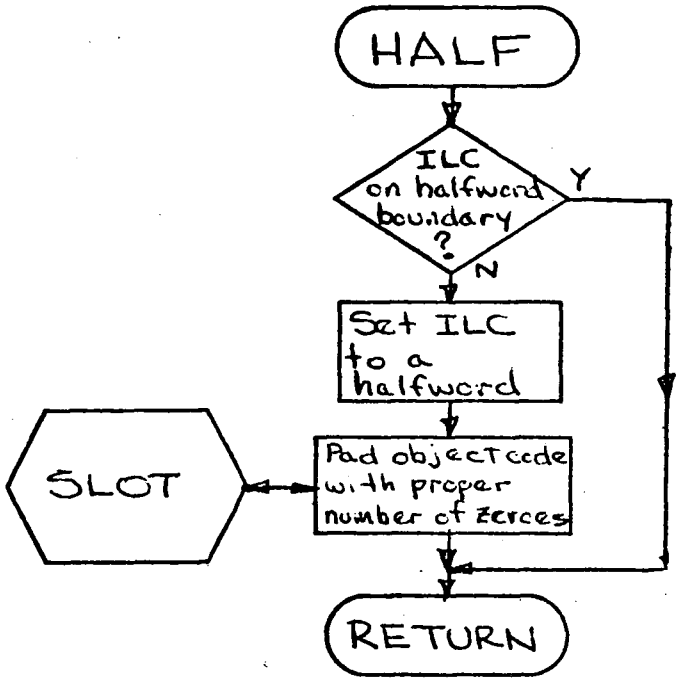

BJ
EVAL6
comma ?
Y
Set flag for comma found
N
EVAL13
blank ?
Y
Set flag for blank found
N
EVAL14
+ ?
Y
Set plus flag
N
EVAL15
- ?
BG 82
Set flag for minus
EVAL16
Set flag to continue
BL
EVAL12
Symbol found ?
Y
Evaluate symbol
FINDSYM
N
Asterick found ?
Y
Number = ILC
BM 82
N
Good ?
N
BG 82
Y
BM 82

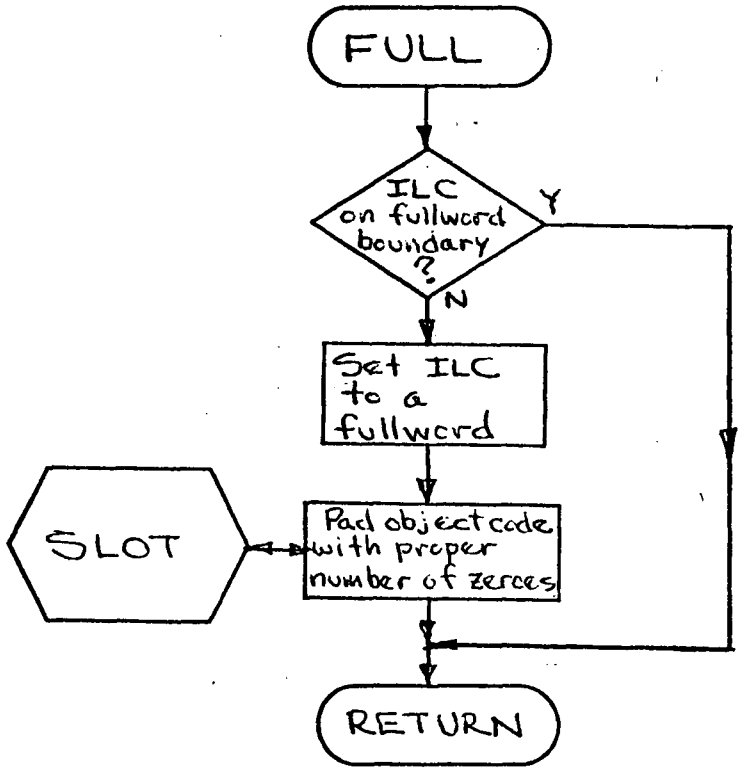

> 6 characters ?
Y
BG 82
N
FVAL
Evaluate number
Good ?
N
BG 82
Y
BM
EVAL21
EVAL20
Subtract number from total
Flag to subtract ?
N
Add number to total
EVAL22
Y
BF 79
continue ?
N
Set flag for unable to evaluate
BG
RETURN

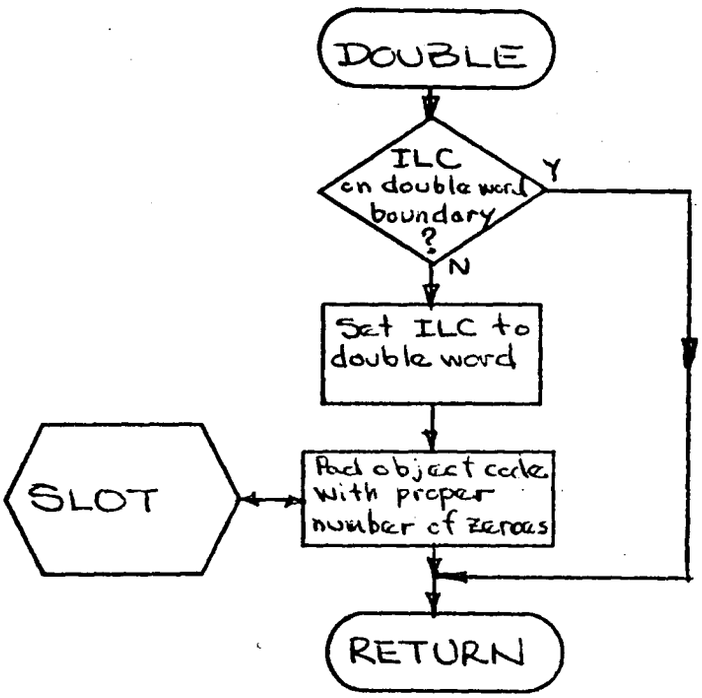

BH
EVAL4
Get entire literal
EVAL91
Get a literal from literal table
EVAL92
N
done 50 entries ?
Y
BG 82
N
In use ?
Y
EVAL90
N
Available ?
Y
N
Match ?
Y
Return value
Return length
GETCHAR
Get next character
RETURN
N
Blank ?
Y
Set flag for blank found

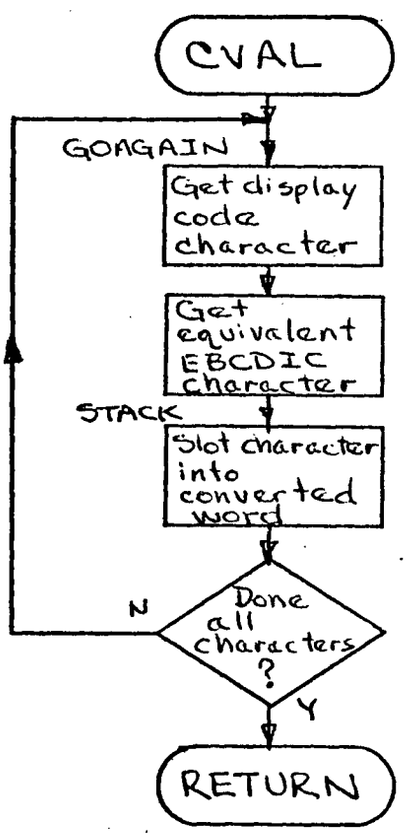

B
AK
AC
LTORGP2
ENDP2
DOUBLE
Set location counter to a double word
Set flag for end card found
ENDP21
Get an entry from literal table
BN
Entry in use ?
N
Y
ENDP23
IPRINT = 1
RETURN
End of table ?
Y
Was entry printed yet ?
Y
N
Y
BN
Adjust ILC to proper boundary
ENDP25
Adjust location counter
Get literal contents
Are contents known ?
N
IERR = 4
Y

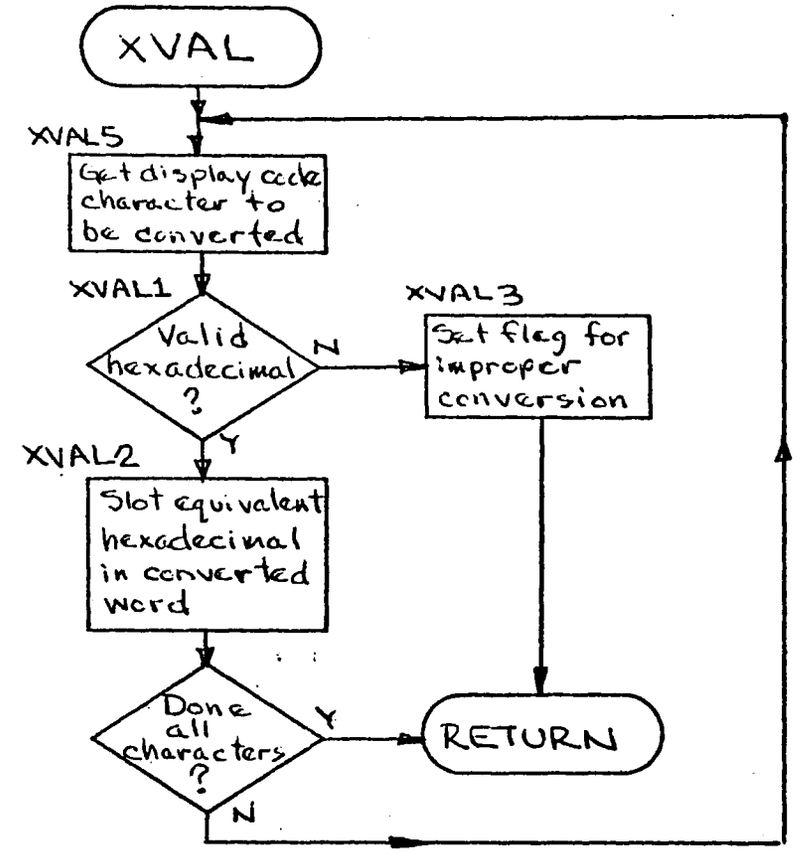

ENDP27
Slot code
SLOT
ENDP28
First literal ?
Y
N
ENDP29
Was there an error ?
N
Y
ENDP2A
IPRINT=6
Y
Was there an error ?
N
IPRINT=5
ENDP2C
IPRINT=3
IPRINT=4
ENDP2B
End of literal block ?
Y
N
ENDP2D
End of table ?
Y
N
Next entry in use ?
N
Y
ENDP2G
Set flag to continue on same card
ENDP2E
Set flag for processing done on card
RETURN

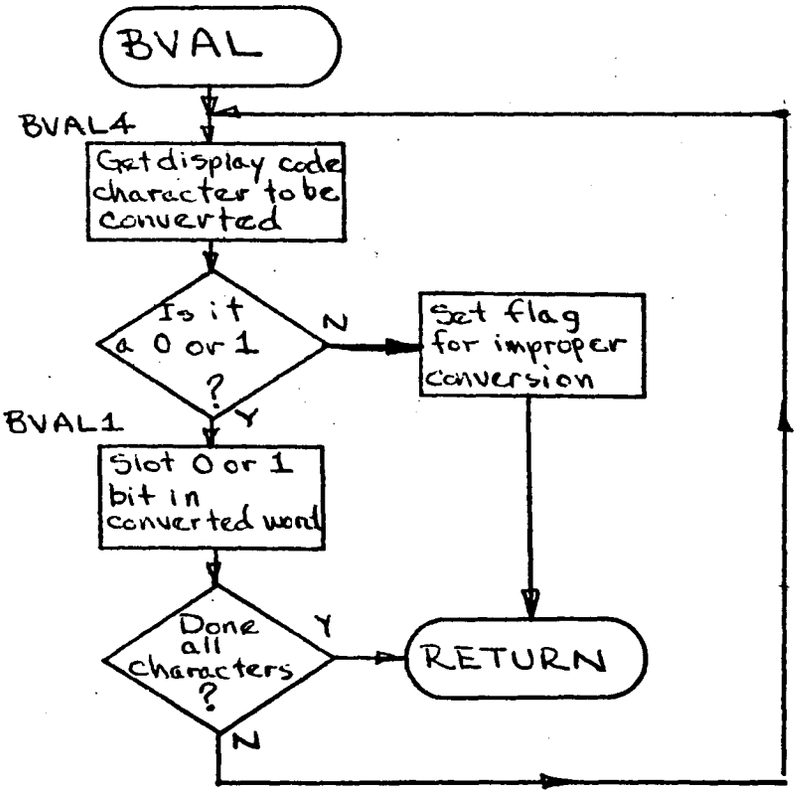

HALF
ILC on halfword boundary ?
Y
N
Set ILC to a halfword
SLOT
Pad object code with proper number of zeroes
RETURN

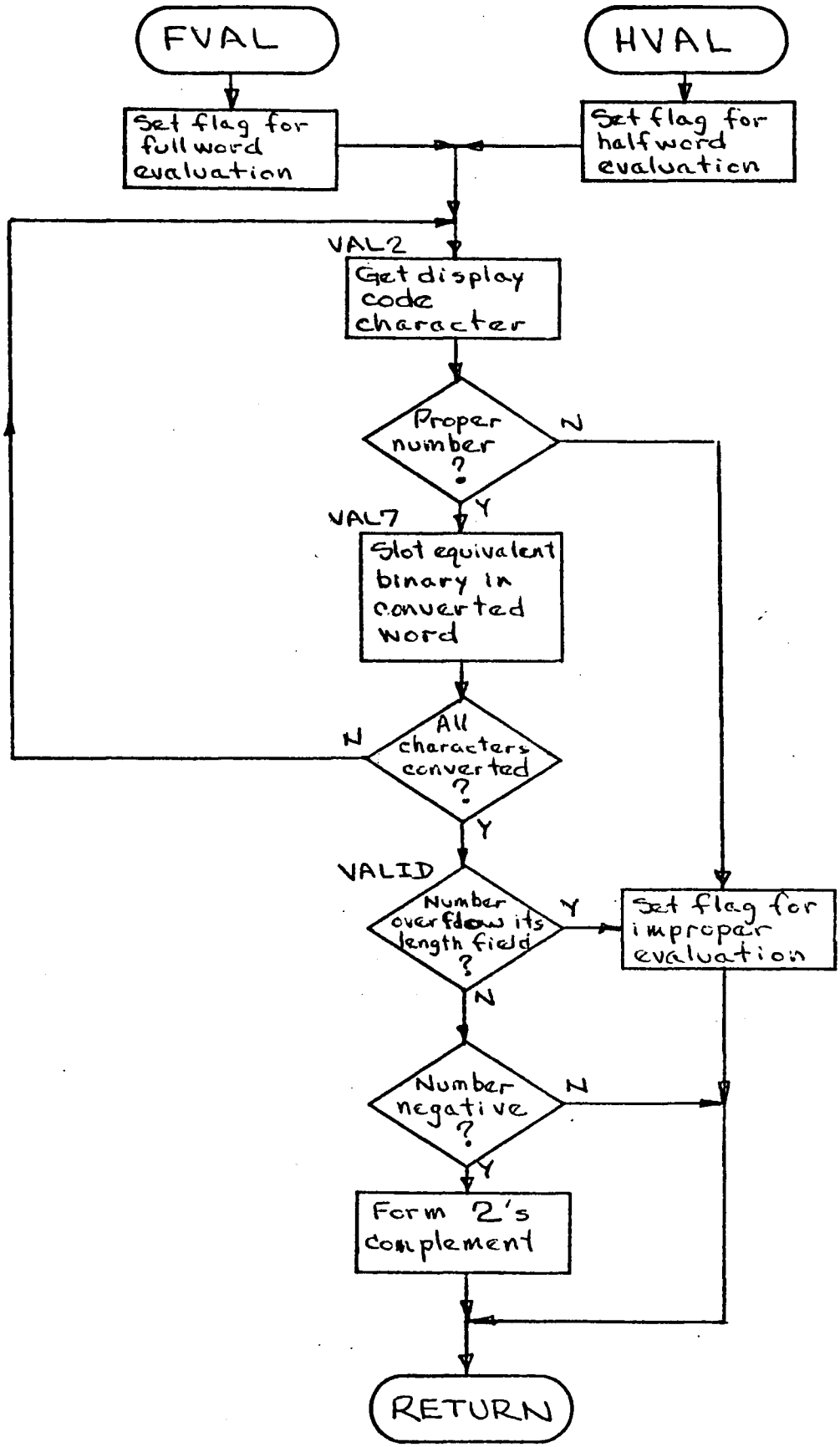

FULL
ILC on fullword boundary ?
Y
N
Set ILC to a fullword
SLOT
Pad object code with proper number of zeroes
RETURN

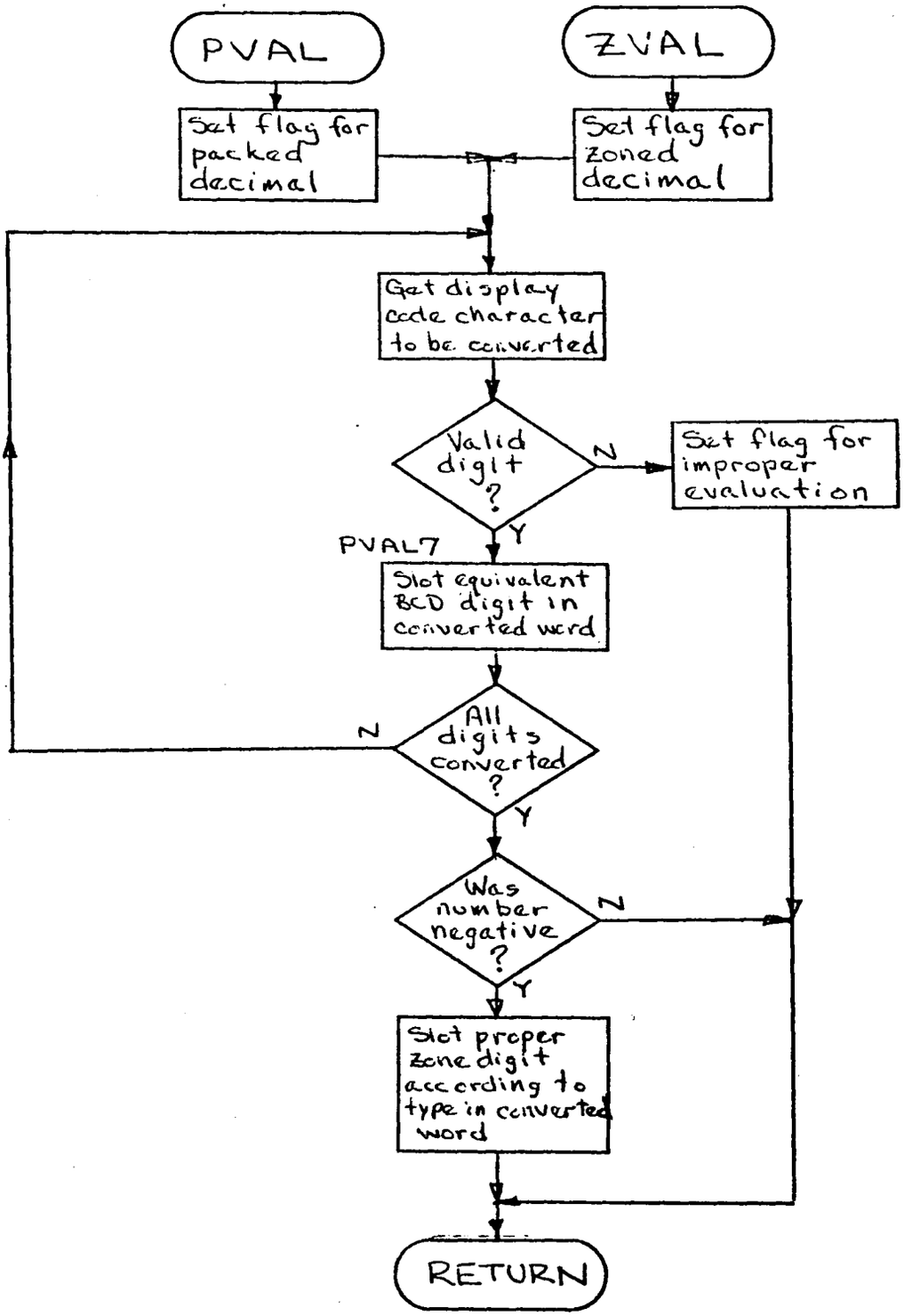

DOUBLE
ILC on double word boundary ?
Y
N
Set ILC to double word
SLOT
Add object code with proper number of zeroes
RETURN

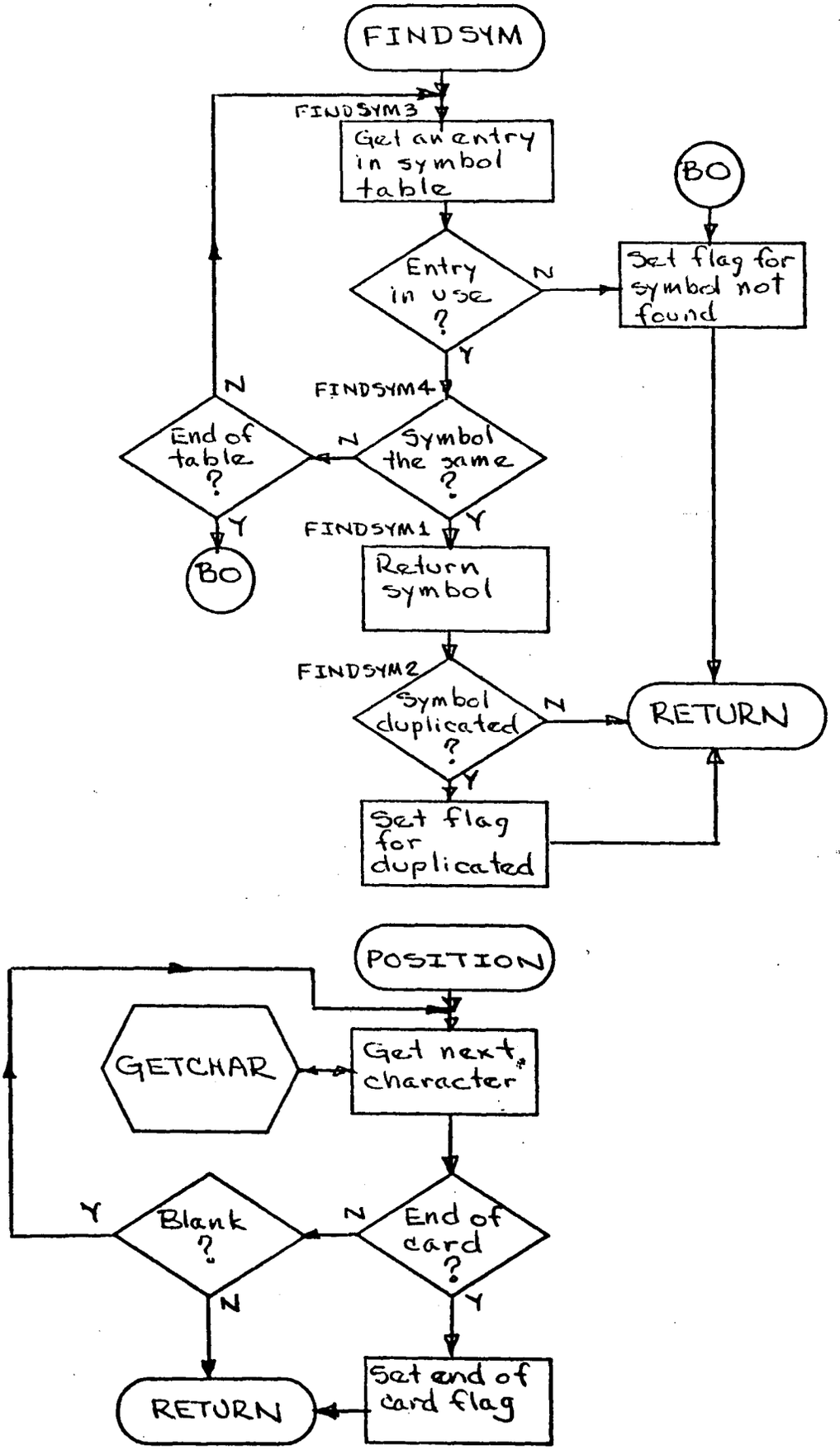

CVAL
GOAGAIN
Get display code character
Get equivalent EBCDIC character
STACK
Slot character into converted word
N
Done all characters ?
Y
RETURN

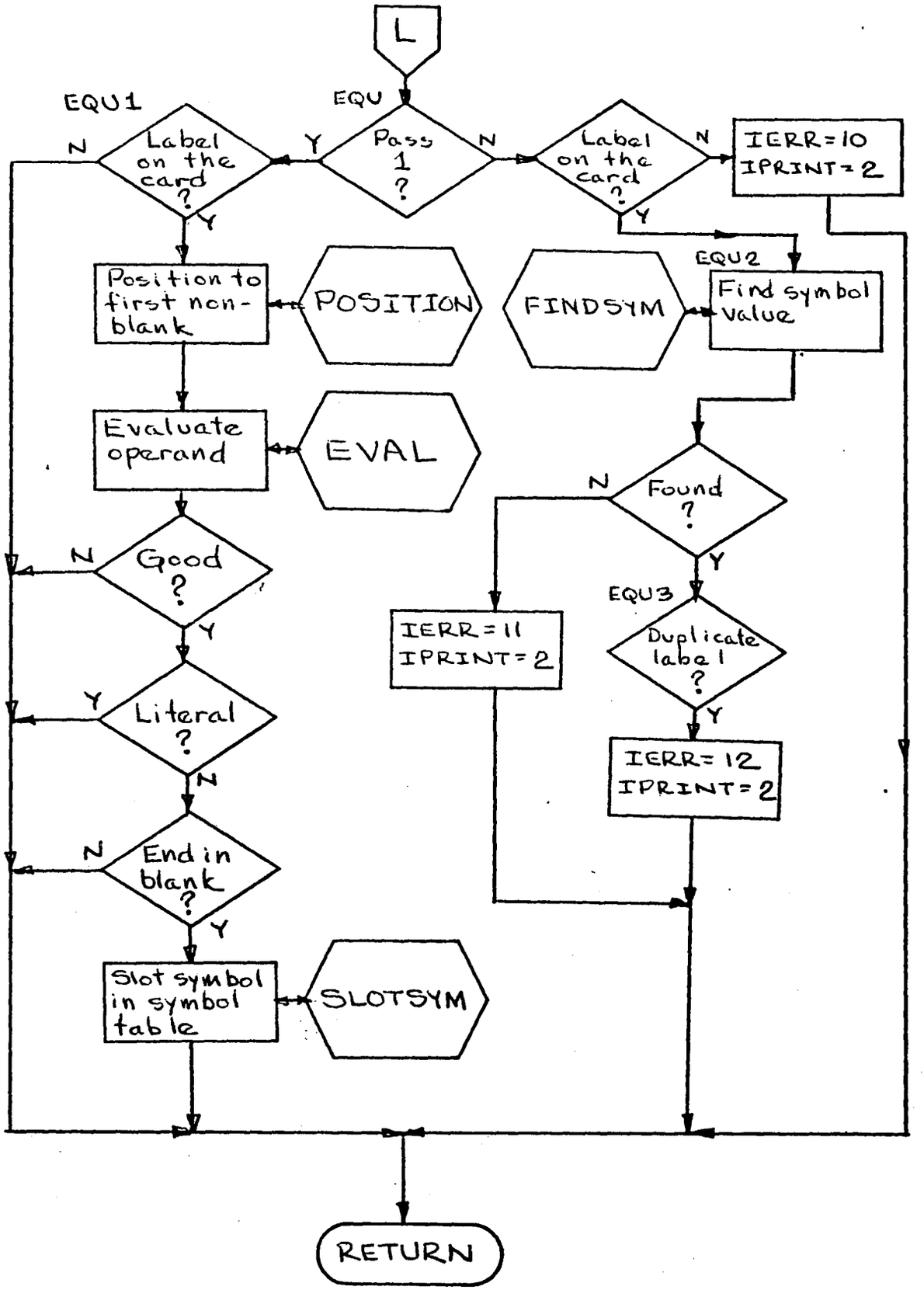

XVAL
XVAL5
Get display code character to be converted
XVAL1
Valid hexadecimal ?
N
XVAL3
Set flag for improper conversion
Y
XVAL2
Slot equivalent hexadecimal in converted word
Done all characters ?
Y
RETURN
N

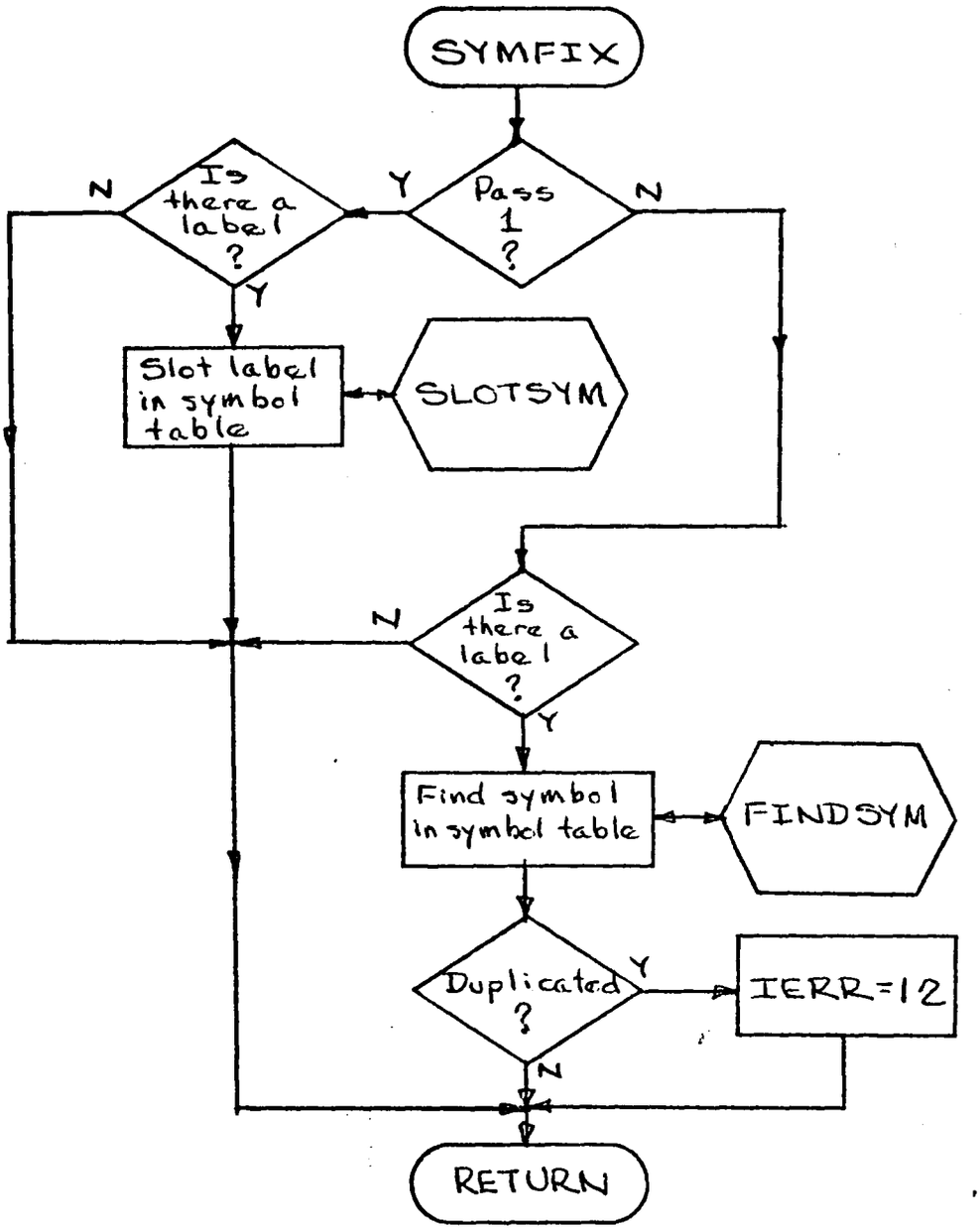

BVAL
BVAL4
Get display code character to be converted
Is it a 0 or 1 ?
N
Set flag for improper conversion
Y
BVAL1
Slot 0 or 1 bit in converted word
Done all characters ?
Y
RETURN
N

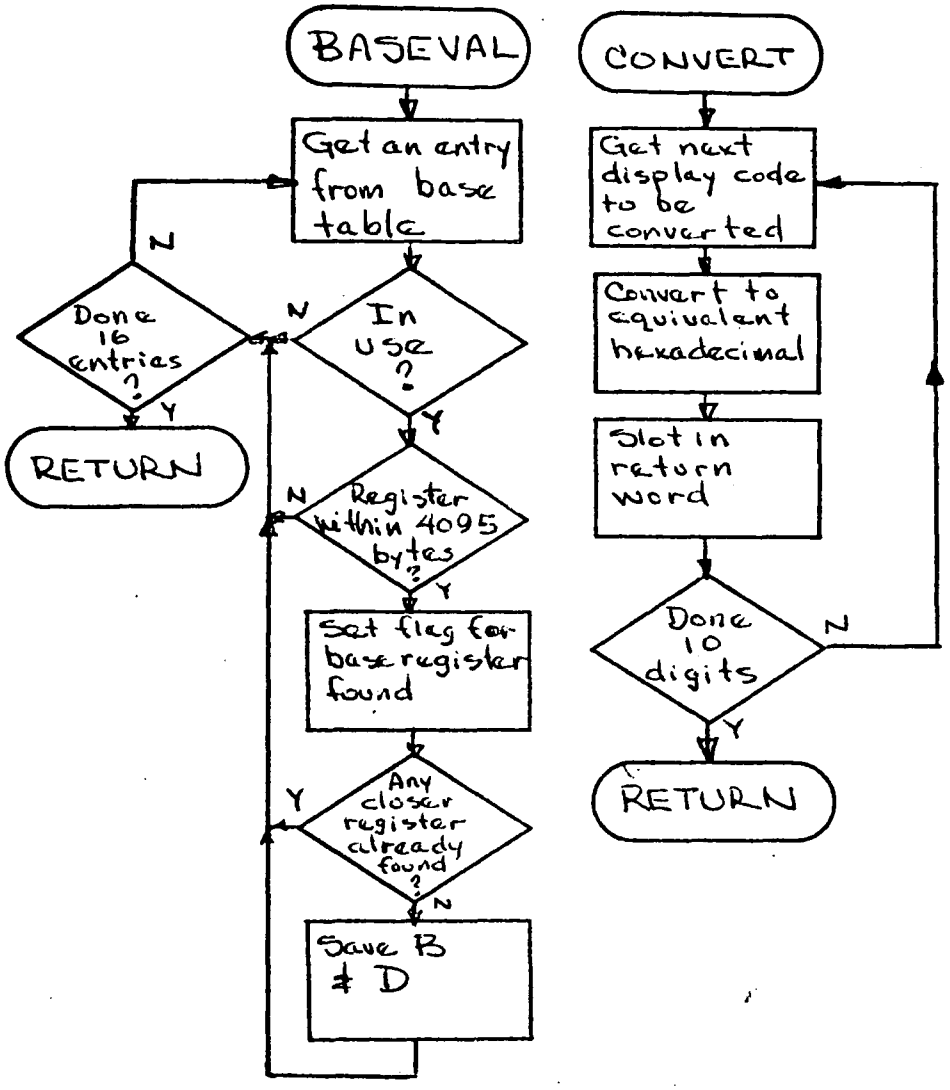

FVAL
HVAL
Set flag for full word evaluation
Set flag for half word evaluation
VAL2
Get display code character
Proper number ?
N
Y
VAL7
Slot equivalent binary in converted word
All characters converted ?
N
Y
VALID
Number overflow its length field ?
Y
Set flag for improper evaluation
N
Number negative ?
N
Y
Form 2's complement
RETURN

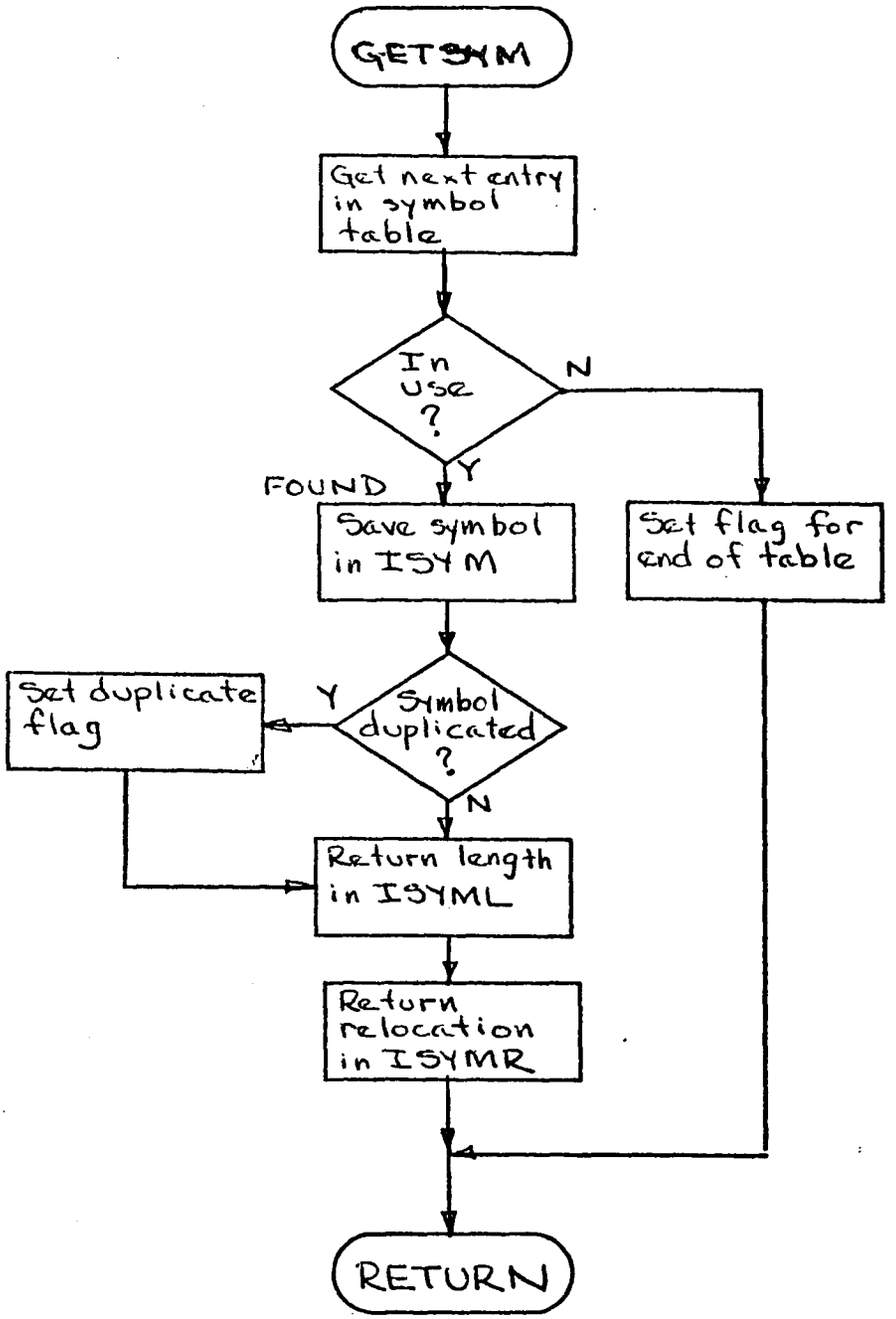

PVAL
ZVAL
Set flag for packed decimal
Set flag for zoned decimal
Get display code character to be converted
Valid digit ?
N
Set flag for improper evaluation
Y
PVAL7
Slot equivalent BCD digit in converted word
All digits converted ?
N
Y
Was number negative ?
N
Y
Slot proper zone digit according to type in converted word
RETURN

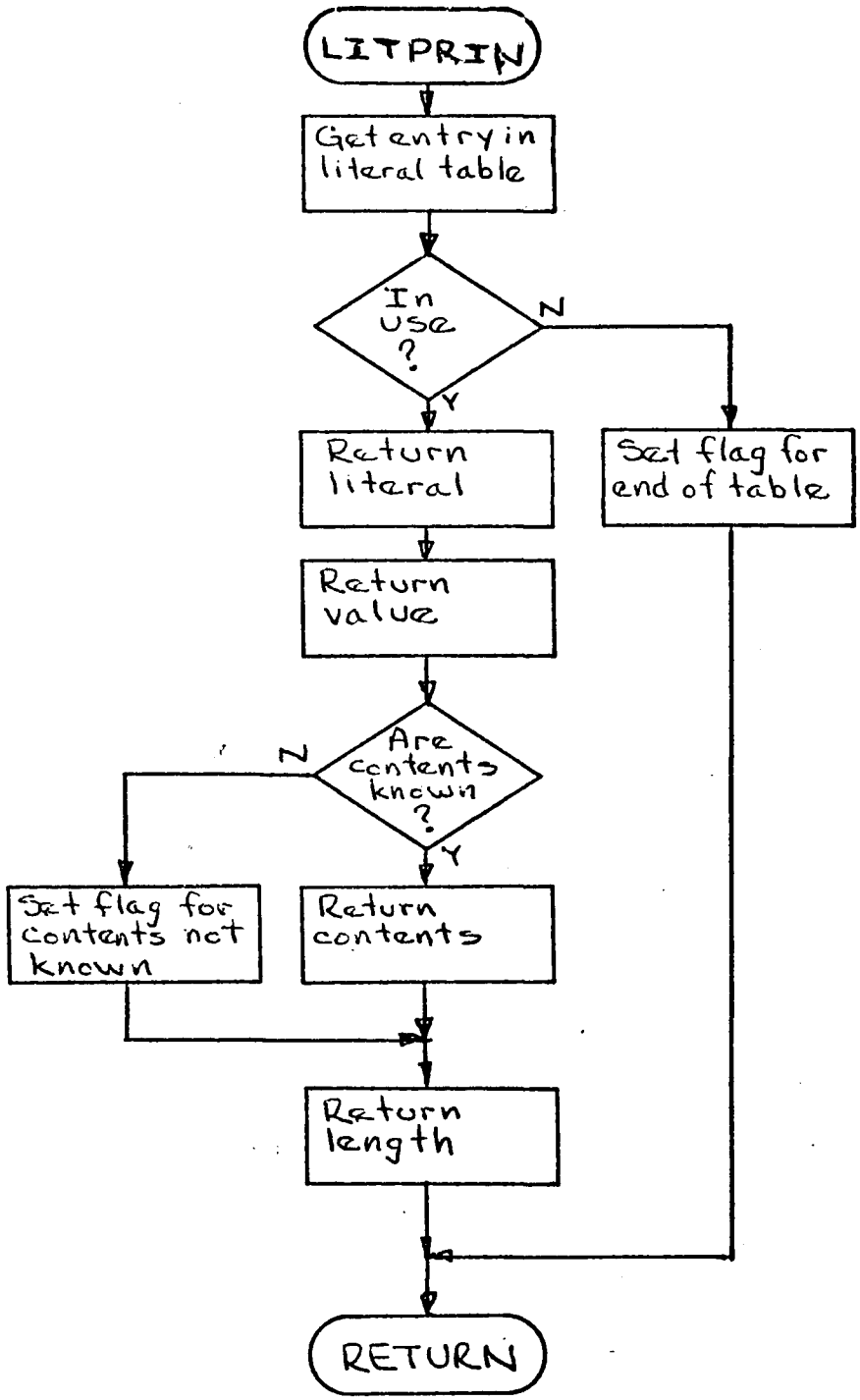

FINDSYM
FINDSYM3
Get an entry in symbol table
BO
Entry in use ?
N
Set flag for symbol not found
Y
FINDSYM4
N
End of table ?
N
Symbol the same ?
Y
FINDSYM1
Y
BO
Return symbol
FINDSYM2
Symbol duplicated ?
N
RETURN
Y
Set flag for duplicated
POSITION
GETCHAR
Get next character
Y
Blank ?
N
End of card ?
N
Y
RETURN
Set end of card flag


L
EQU
Pass 1 ?
Y
N
Label on the card ?
N
IERR=10
IPRINT=2
EQU1
Label on the card ?
N
Y
EQU2
Position to first non-blank
POSITION
FINDSYM
Find symbol value
Evaluate operand
EVAL
Found ?
N
Y
Good ?
N
Y
EQU3
IERR=11
IPRINT=2
Duplicate label ?
Y
IERR=12
IPRINT=2
Literal ?
Y
N
End in blank ?
N
Y
Slot symbol in symbol table
SLOTSYM
RETURN


SYMFIX
Pass 1 ?
Y
N
Is there a label ?
N
Y
Slot label in symbol table
SLOTSYM
Is there a label ?
N
Y
Find symbol in symbol table
FINDSYM
Duplicated ?
Y
IERR=12
N
RETURN


BASEVAL
Get an entry from base table
Done 16 entries ?
N
Y
RETURN
In use ?
N
Y
Register within 4095 bytes ?
N
Y
Set flag for base register found
Any closer register already found ?
Y
N
Save B ≠ D
CONVERT
Get next display code to be converted
Convert to equivalent hexadecimal
Slot in return word
Done 10 digits
N
Y
RETURN


GETSYM
Get next entry in symbol table
In use ?
N
Y
FOUND
Save symbol in ISYM
Set flag for end of table
Set duplicate flag
Y
Symbol duplicated ?
N
Return length in ISYML
Return relocation in ISYMR
RETURN


LITPRIN
Get entry in literal table
In use ?
N
Y
Return literal
Set flag for end of table
Return value
Are contents known ?
N
Y
Set flag for contents not known
Return contents
Return length
RETURN

VI. Improving LUIAS

6.1 Introduction

LUIAS contains certain restrictions which were imposed because of the differences between the byte oriented IBM System/360 and the word oriented CDC 6400. Other restrictions were imposed to decrease the length of the program LUIAS. This chapter discusses how to remove or lessen the impact of these restrictions.

Section 2,3 and 4 discuss how the increase the table lengths contained within LUIAS. Section 5 discusses increasing the length of a valid character string. Section 6 discusses how to increase LUIAS's ability to evaluate expressions. Section 7 describes how to add additional pseudo-instructions, while Section 8 describes how to add more machine instructions.

6.2 Increasing the Length of Assembled Machine Code

At the present time, LUIAS can assemble up to 25,000 bytes of machine code. This code is stored 10 bytes per word (display code) in the array ICONT. The maximum number of bytes of assembled machine code can be increased (10 bytes at a time) by increasing ICONT (1 word at a time), which is contained in the "INFO" common block (Fig. 1.1). This change must be made in the main program ASSMB and in the subroutine ASSMBL. A constant within ASSMBL which defines the length of the table must be changed accordingly.

6.3 Increasing the Length of the Symbol Table

LUIAS can presently store up to 100 symbols. These symbols are stored in the array ISYMTAB, one word per entry, within the common block "SYMINFO" (Fig. 1.2). For every additional symbol desired, one word must be added to ISYMTAB. This change must be made in the main routine ASSMB, the subroutine ASSMBL, and the subroutine GETSYM. A constant within ASSMBL and a constant within GETSYM which define the length of the table must be changed accordingly.

6.4 Increasing the Length of the Literal Table

LUIAS can generate up to 50 literals and store them in the literal table. These literals are stored in the array ILIT, 3 words per entry, within the common block "LITINFO" (Fig. 1.3). To increase the length of the literal table, the array ILIT must be increased 3 words for each additional entry. This change must be made within the main routine ASSMB, the subroutine ASSMBL, and the subroutine LITPRIN. A constant within ASSMBL and a constant within LITPRIN which define the length of the table must be increased by 3 for each additional entry.

6.5 Increasing the Length of a Character String

LUIAS will accept a character string of maximum length 6. This restriction was imposed so that the entire string could be stored in 1 CDC 6400 computer word. If this restriction had not been imposed, a literal character string could not be stored in one word of the literal table. An elaborate scheme would then have to be developed to keep track of the length of literals and their position within the literal table. This restriction does not exist within the IBM System/360 since every character occupies one byte and every byte in an IBM/360 can be addressed.

The logic necessary to keep track of the position of a character within a CDC 6400 word seemed not be justify the limited improvement to be gained.

A programmer using LUIAS must merely use explicit declarations in a storage to storage instruction if he wishes to use data that is longer than 6 characters.

6.6 Increasing the Ability to Evaluate Expressions

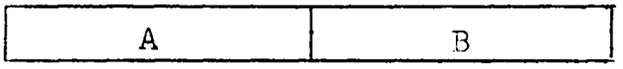
At the present time, LUIAS will evaluate any expression that contains fixed point numbers only. This restriction is imposed because the evaluation routines cannot distinguish between a symbol and a constant other than fixed point.

The CDC 6400 computer system does not support the single quote character ('). To enclose a declared constant, the left and right parenthesis were chosen to replace the single quote. The general purpose evaluation routines can now no longer distinguish between a symbol followed by a left parenthesis and a character declaration.

This problem can be overcome by using symbols other than parenthesis for character declarations, or by writing more specific evaluation routines. The closed subroutine EVAL within the subroutine ASSEMBL could then be easily expanded to accept constant declarations that are not fixed point numbers.

6.7 Adding More Pseudo-Instructions

To add any additional pseudo-instructions, it is necessary to add an entry into the pseudo-operation table and to change a constant within ASSEMBL which defines the length of the table. This entry has the following format:



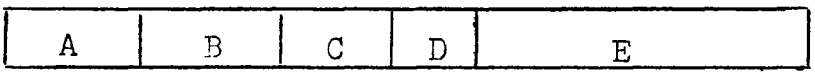
A. Unused (30 bits)

B. Pseudo-op mnemonic (30 bits display code)

It is also necessary to make an entry into a table which parallels the pseudo-op table. This table entry would be a branch to the proper pseudo-op processor. The logic to process the new pseudo-op can be inserted at the end of subroutine ASSEMBL.

6.8 Adding More Machine Instructions

To add any additional instruction it is necessary to add an entry into the machine operation table and to change a constant within ASSEMBL which defines the length of the table. This entry has the following format:



A. Hexidecimal op-code (12 bits display code)

B. Unused (12 bits)

C. Instruction type (9 bits)

0 RR Format
1 RX Format
2 RS Format (no shift instructions)
3 RS Format (all shift instructions)
4 SI Format (SI0 instruction only)
5 SI Format (without an I2 field)
6 SI Format (with an I2 field)
7 SS Format (L field only)
8 SS Format (L1 and L2 fields)

D. Instruction length in bytes (6 bits)

E. Instruction mnemonic (24 bits display code)

Appendix A lists all IBM System/360 instructions.

References

1. Horey, Leonard I: The Lehigh University IBM/360 Simulator. A Masters Thesis, Lehigh University, 1975.

2. A22-6821-6, IBM System/360 Principles of Operation, Copyright 1967, International Business Machines Corporation.

3. COMPASS Version 3 Reference Manual, Copyright 1974, Control Data Corporation.

4. Grishman, Ralph: "Assembly Language Programming for the Control Data 6000 Series," Algorithmic Press, New York, Copyright 1971.

5. SCOPE Reference Manual Version 3.4.1, Copyright 1974, Control Data Corporation.

6. Donovan, John J: "Systems Programming," McGraw-Hill Book Company, Copyright 1972.

Appendix A

STANDARD INSTRUCTION SET

NAME	MNEMONIC	TYPE	CODE	OPERAND
Add	AR	RR	1A	R1, R2
Add	A	RX	5A	R1, D2 (X2, B2)
Add Halfword	AH	RX	4A	R1, D2 (X2, B2)
Add Logical	ALR	RR	1E	R1, R2
Add Logical	AL	RX	5E	R1, D2 (X2, B2)
AND	NR	RR	14	R1, R2
AND	N	RX	54	R1, D2 (X2, B2)
AND	NI	SI	94	D1 (B1), I2
AND	NC	SS	D4	D1 (L, B1), D2 (B2)
Branch and Link	BALR	RR	05	R1, R2
Branch and Link	BAL	RX	45	R1, D2 (X2, B2)
Branch on Condition	BCR	RR	07	M1, R2
Branch on Condition	BC	RX	47	M1, D2 (X2, B2)
Branch on Count	BCTR	RR	06	R1, R2
Branch on Count	BCT	RX	46	R1, D2 (X2, B2)
Branch on Index High	BXH	RS	86	R1, R3, D2 (B2)
Branch on Index Low or Equal	BXLE	RS	87	R1, R3, D2 (B2)
Compare	CR	RR	19	R1, R2
Compare	C	RX	59	R1, D2 (X2, B2)
Compare Halfword	CH	RX	49	R1, D2 (X2, B2)
Compare Logical	CLR	RR	15	R1, R2
Compare Logical	CL	RX	55	R1, D2 (X2, B2)
Compare Logical	CLC	SS	D5	D1 (L, B1), D2 (B2)
Compare Logical	CLI	SI	95	D1 (B1), I2
Convert to Binary	CVB	RX	4F	R1, D2 (X2, B2)
Convert to Decimal	CVD	RX	4E	R1, D2 (X2, B2)
Diagnose		SI	83	
Divide	DR	RR	1D	R1, R2
Divide	D	RX	5D	R1, D2 (X2, B2)
Exclusive OR	XR	RR	17	R1, R2
Exclusive OR	X	RX	57	R1, D2 (X2, B2)
Exclusive OR	XI	SI	97	D1 (B1), I2
Exclusive OR	XC	SS	D7	D1 (L, B1), D2 (B2)
Execute	EX	RX	44	R1, D2 (X2, B2)
Halt I/O	HIO	SI	9E	D1 (B1)
Insert Character	IC	RX	43	R1, D2 (X2, B2)
Load	LR	RR	18	R1, R2
Load	L	RX	58	R1, D2 (X2, B2)
Load Address	LA	RX	41	R1, D2 (X2, B2)
Load and Test	LTR	RR	12	R1, R2
Load Complement	LCR	RR	13	R1, R2
Load Halfword	LH	RX	48	R1, D2 (X2, B2)
Load Multiple	LM	RS	98	R1, R3, D2 (B2)
Load Negative	LNR	RR	11	R1, R2
Load Positive	LPR	RR	10	R1, R2
Load PSW	LPSW	SI	82	D1 (B1)
Move	MVI	SI	92	D1 (B1), I2
Move	MVC	SS	D2	D1 (L, B1), D2 (B2)
Move Numerics	MVN	SS	D1	D1 (L, B1), D2 (B2)
Move with Offset	MVO	SS	F1	D1 (L1, B1), D2 (L2, B2)
Move Zones	MVZ	SS	D3	D1 (L, B1), D2 (B2)
Multiply	MR	RR	1C	R1, R2
Multiply	M	RX	5C	R1, D2 (X2, B2)
Multiply Halfword	MH	RX	4C	R1, D2 (X2, B2)
OR	OR	RR	16	R1, R2
OR	O	RX	56	R1, D2 (X2, B2)
OR	OI	SI	96	D1 (B1), I2
OR	OC	SS	D6	D1 (L, B1), D2 (B2)
Pack	PACK	SS	F2	D1 (L1, B1), D2 (L2, B2)
Set Program Mask	SPM	RR	04	R1
Set System Mask	SSM	SI	80	D1 (B1)
Shift Left Double	SLDA	RS	8F	R1, D2 (B2)
Shift Left Single	SLA	RS	8B	R1, D2 (B2)
Shift Left Double Logical	SLDL	RS	8D	R1, D2 (B2)
Shift Left Single Logical	SLL	RS	89	R1, D2 (B2)
Shift Right Double	SRDA	RS	8E	R1, D2 (B2)

STANDARD INSTRUCTION SET (Continued)

Shift Right Single	SRA	RS	8A	R1, D2 (B2)
Shift Right Double Logical	SRDL	RS	8C	R1, D2 (B2)
Shift Right Single Logical	SRL	RS	88	R1, D2 (B2)
Start I/O	SIO	SI	9C	D1 (B1)
Store	ST	RX	50	R1, D2 (X2, B2)
Store Character	STC	RX	42	R1, D2 (X2, B2)
Store Halfword	STH	RX	40	R1, D2 (X2, B2)
Store Multiple	STM	RS	90	R1, R3, D2 (B2)
Subtract	SR	RR	1B	R1, R2
Subtract	S	RX	5B	R1, D2 (X2, B2)
Subtract Halfword	SH	RX	4B	R1, D2 (X2, B2)
Subtract Logical	SLR	RR	1F	R1, R2
Subtract Logical	SL	RX	5F	R1, D2 (X2, B2)
Supervisor Call	SVC	RR	0A	I
Test and Set	TS	SI	93	D1 (B1)
Test Channel	TCH	SI	9F	D1 (B1)
Test I/O	TIO	SI	9D	D1 (B1)
Test Under Mask	TM	SI	91	D1 (B1), I2
Translate	TR	SS	DC	D1 (L, B1), D2 (B2)
Translate and Test	TRT	SS	DD	D1 (L, B1), D2 (B2)
Unpack	UNPK	SS	F3	D1 (L1, B1), D2 (L2, B2)

DECIMAL FEATURE INSTRUCTIONS

Add Decimal	AP	SS	FA	D1 (L1, B1), D2 (L2, B2)
Compare Decimal	CP	SS	F9	D1 (L1, B1), D2 (L2, B2)
Divide Decimal	DP	SS	FD	D1 (L1, B1), D2 (L2, B2)
Edit	ED	SS	DE	D1 (L, B1), D2 (B2)
Edit and Mark	EDMK	SS	DF	D1 (L, B1), D2 (B2)
Multiply Decimal	MP	SS	FC	D1 (L1, B1), D2 (L2, B2)
Subtract Decimal	SP	SS	FB	D1 (L1, B1), D2 (L2, B2)
Zero and Add	ZAP	SS	F8	D1 (L1, B1), D2 (L2, B2)

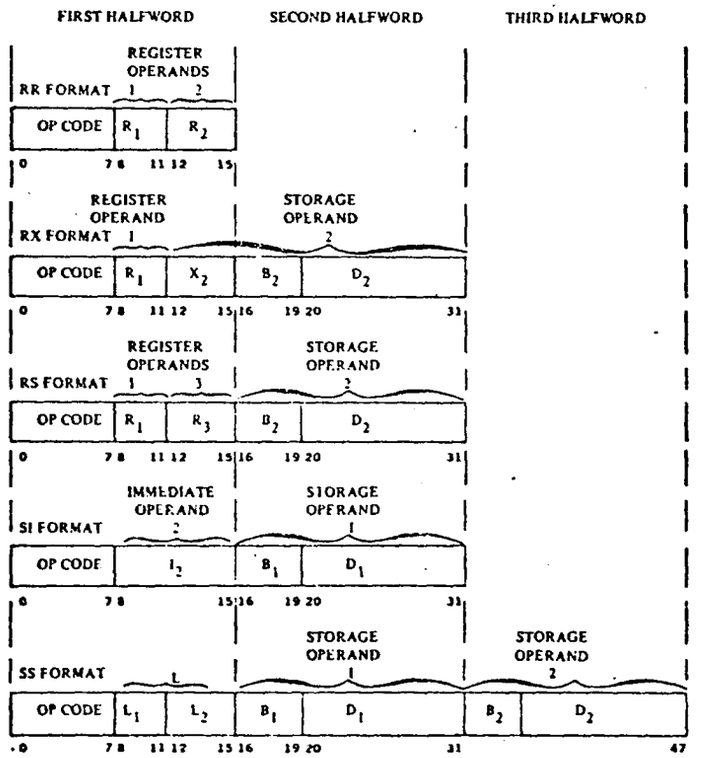
DIRECT CONTROL FEATURE INSTRUCTIONS

Read Direct	RDD	SI	85	D1 (B1), I2
Write Direct	WRD	SI	84	D1 (B1), I2

PROTECTION FEATURE INSTRUCTIONS

Insert Storage Key	ISK	RR	09	R1, R2
Set Storage Key	SSK	RR	08	R1, R2

BASIC INSTRUCTION FORMATS



EXTENDED MNEMONIC INSTRUCTION CODES ④

GENERAL

Extended Code		Machine Instruction		Meaning
B	D2(X2,B2)	BC	15, D2(X2,B2)	Branch Unconditionally
BR	R2	BCR	15, R2	Branch Unconditionally
NOP	D2(X2,B2)	BC	0, D2(X2,B2)	No Operation
NOPR	R2	BCR	0, R2	No Operation (RR)

AFTER COMPARE INSTRUCTIONS (A:B)

BH	D2(X2,B2)	BC 2,	D2(X2,B2)	Branch on A High
BL	D2(X2,B2)	BC 4,	D2(X2,B2)	Branch on A Low
BE	D2(X2,B2)	BC 8,	D2(X2,B2)	Branch on A Equal B
BNH	D2(X2,B2)	BC 13,	D2(X2,B2)	Branch on A Not High
BNL	D2(X2,B2)	BC 11,	D2(X2,B2)	Branch on A Not Low
BNE	D2(X2,B2)	BC 7,	D2(X2,B2)	Branch on A Not Equal B

AFTER ARITHMETIC INSTRUCTIONS

BO	D2(X2,B2)	BC 1,	D2(X2,B2)	Branch on Overflow
BP	D2(X2,B2)	BC 2,	D2(X2,B2)	Branch on Plus
BM	D2(X2,B2)	BC 4,	D2(X2,B2)	Branch on Minus
BZ	D2(X2,B2)	BC 8,	D2(X2,B2)	Branch on Zero
BNP	D2(X2,B2)	BC 13,	D2(X2,B2)	Branch on Not Plus
BNM	D2(X2,B2)	BC 11,	D2(X2,B2)	Branch on Not Minus
BNZ	D2(X2,B2)	BC 7,	D2(X2,B2)	Branch on Not Zero

AFTER TEST UNDER MASK INSTRUCTIONS

BO	D2(X2,B2)	BC 1,	D2(X2,B2)	Branch if Ones
BM	D2(X2,B2)	BC 4,	D2(X2,B2)	Branch if Mixed
BZ	D2(X2,B2)	BC 8,	D2(X2,B2)	Branch if Zeros
BNO	D2(X2,B2)	BC 14,	D2(X2,B2)	Branch if Not Ones

RR FORMAT INSTRUCTIONS

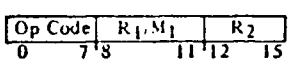
(7)

Decimal	Hexadecimal	Mnemonic	Graphic & Control Symbols BCDIC	Graphic & Control Symbols EBCDIC	(2) 7-Track Tape BCDIC	Punched Card Code	System/360 8-Bit Code
0	00			NUL		12-0-1-8-9	0000 0000
1	01			SOH		12-1-9	0000 0001
2	02			STX		12-2-9	0000 0010
3	03			ETX		12-3-9	0000 0011
4	04	SPM		PF		12-4-9	0000 0100
5	05	BALR		HT		12-5-9	0000 0101
6	06	BCTR		LC		12-6-9	0000 0110
7	07	BCR		DEL		12-7-9	0000 0111
8	08	SSK				12-8-9	0000 1000
9	09	ISK				12-1-8-9	0000 1001
10	0A	SVC		SMM		12-2-8-9	0000 1010
11	0B			VT		12-3-8-9	0000 1011
12	0C	(EBCDIC +)		FF		12-4-8-9	0000 1100
13	0D	(EBCDIC -)		CR		12-5-8-9	0000 1101
14	0E			SO		12-6-8-9	0000 1110
15	0F			SI		12-7-8-9	0000 1111
16	10	LPR		DLE		12-11-1-8-9	0001 0000
17	11	LNR		DC1		11-1-9	0001 0001
18	12	LTR		DC2		11-2-9	0001 0010
19	13	LCR		TM		11-3-9	0001 0011
20	14	NR		RES		11-4-9	0001 0100
21	15	CLR		NL		11-5-9	0001 0101
22	16	OR		BS		11-6-9	0001 0110
23	17	XR		IL		11-7-9	0001 0111
24	18	LR		CAN		11-8-9	0001 1000
25	19	CR		EM		11-1-8-9	0001 1001
26	1A	AR		CC		11-2-8-9	0001 1010
27	1B	SR		CU1		11-3-8-9	0001 1011
28	1C	MR		IFS		11-4-8-9	0001 1100
29	1D	DR		IGS		11-5-8-9	0001 1101
30	1E	ALR		IRS		11-6-8-9	0001 1110
31	1F	SLR		IUS		11-7-8-9	0001 1111
32	20	LPDR		DS		11-0-1-8-9	0010 0000
33	21	LNDR		SOS		0-1-9	0010 0001
34	22	LTDR		FS		0-2-9	0010 0010
35	23	LCDR				0-3-9	0010 0011
36	24	HDR		BYP		0-4-9	0010 0100
37	25	LRDR		LF		0-5-9	0010 0101
38	26	MXR		ETB		0-6-9	0010 0110
39	27	MXDR		ESC		0-7-9	0010 0111
40	28	LDR				0-8-9	0010 1000
41	29	CDR				0-1-8-9	0010 1001
42	2A	ADR		SM		0-2-8-9	0010 1010
43	2B	SDR		CU2		0-3-8-9	0010 1011
44	2C	MDR				0-4-8-9	0010 1100
45	2D	DDR		ENQ		0-5-8-9	0010 1101
46	2E	AWR		ACK		0-6-8-9	0010 1110
47	2F	SWR		BEL		0-7-8-9	0010 1111
48	30	LPER				12-11-0-1-8-9	0011 0000
49	31	LNER				1-9	0011 0001
50	32	LTER		SYN		2-9	0011 0010
51	33	LCER				3-9	0011 0011
52	34	HER		PN		4-9	0011 0100
53	35	LRER		RS		5-9	0011 0101
54	36	AXR		UC		6-9	0011 0110
55	37	SXR		EOT		7-9	0011 0111
56	38	LER				8-9	0011 1000
57	39	CER				1-8-9	0011 1001
58	3A	AER				2-8-9	0011 1010
59	3B	SER		CU3		3-8-9	0011 1011
60	3C	MER		DC4		4-8-9	0011 1100
61	3D	DER		NAK		5-8-9	0011 1101
62	3E	AUR				6-8-9	0011 1110
63	3F	SUR		SUB		7-8-9	0011 1111

(2) Add C (check bit) for odd or even parity as needed, except for even parity, decimal 64 is CA, the same as decimal 122.

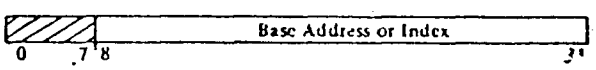
(3) Decimal Feature instructions. (4) System 360 assembler programs require these codes.

RR FORMAT



R1, R2 – meaningful for all RR instructions except SPM, SVC

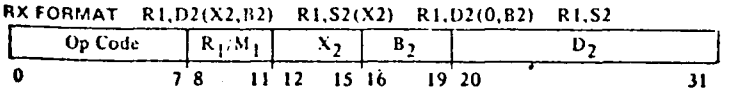
BASE AND INDEX REGISTERS



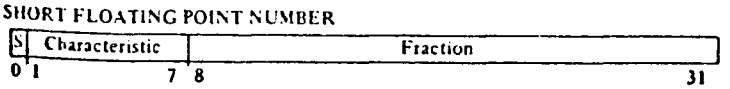
RX FORMAT INSTRUCTIONS

Decimal	Hexadecimal	Mnemonic	Graphic & Control Symbols BCDIC	Graphic & Control Symbols EBCDIC	(2) 7-Track Tape BCDIC	Punched Card Code	System/360 8 Bit Code	(4)
64	40	STH		SP	(2)	no punches	0100 0000	
65	41	LA				12-0-1-9	0100 0001	
66	42	STC				12-0-2-9	0100 0010	
67	43	IC				12-0-3-9	0100 0011	
68	44	EX				12-0-4-9	0100 0100	
69	45	BAL				12-0-5-9	0100 0101	
70	46	BCT				12-0-6-9	0100 0110	
71	47	BC				12-0-7-9	0100 0111	
72	48	LH				12-0-8-9	0100 1000	
73	49	CH				12-1-8	0100 1001	
74	4A	AH	¢	¢		12-2-8	0100 1010	
75	4B	SH	.	.	B A 8 2 1	12-3-8	0100 1011	
76	4C	MH	¤)	<	B A 8 4	12-4-8	0100 1100	
77	4D		(	(	B A 8 4 1	12-5-8	0100 1101	(
78	4E	CVD	<	+	B A 8 4 2	12-6-8	0100 1110	+
79	4F	CVB	&	!	B A 8 4 2 1	12-7-8	0100 1111	
80	50	ST	& +	&	B A	12	0101 0000	
81	51					12-11-1-9	0101 0001	
82	52					12-11-2-9	0101 0010	
83	53					12-11-3-9	0101 0011	
84	54	N				12-11-4-9	0101 0100	
85	55	CL				12-11-5-9	0101 0101	
86	56	O				12-11-6-9	0101 0110	
87	57	X				12-11-7-9	0101 0111	
88	58	L				12-11-8-9	0101 1000	
89	59	C				11-1-8	0101 1001	
90	5A	A		!		11-2-8	0101 1010	
91	5B	S	$	$	B 8 2 1	11-3-8	0101 1011	
92	5C	M	*	*	B 8 4	11-4-8	0101 1100	
93	5D	D	)	)	B 8 4 1	11-5-8	0101 1101	)
94	5E	AL	;	;	B 8 4 2	11-6-8	0101 1110	
95	5F	SL	Δ	¬	B 8 4 2 1	11-7-8	0101 1111	
96	60	STD	-	-	B	11	0110 0000	
97	61		/	/	A 1	0-1	0110 0001	
98	62					11-0-2-9	0110 0010	
99	63					11-0-3-9	0110 0011	
100	64					11-0-4-9	0110 0100	
101	65					11-0-5-9	0110 0101	
102	66					11-0-6-9	0110 0110	
103	67	MXD				11-0-7-9	0110 0111	
104	68	LD				11-0-8-9	0110 1000	
105	69	CD				0-1-8	0110 1001	
106	6A	AD				12-11	0110 1010	
107	6B	SD	,	,	A 8 2 1	0-3-8	0110 1011	
108	6C	MD	% (	%	A 8 4	0-4-8	0110 1100	
109	6D	DD	Y		A 8 4 1	0-5-8	0110 1101	
110	6E	AW	\	>	A 8 4 2	0-6-8	0110 1110	
111	6F	SW	¤	?	A 8 4 2 1	0-7-8	0110 1111	
112	70	STE				12-11-0	0111 0000	
113	71					12-11-0-1-9	0111 0001	
114	72					12-11-0-2-9	0111 0010	
115	73					12-11-0-3-9	0111 0011	
116	74					12-11-0-4-9	0111 0100	
117	75					12-11-0-5-9	0111 0101	
118	76					12-11-0-6-9	0111 0110	
119	77					12-11-0-7-9	0111 0111	
120	78	LE				12-11-0-8-9	0111 1000	
121	79	CE				1-8	0111 1001	
122	7A	AE	b	:	A	2-8	0111 1010	
123	7B	SE	# ¤	#	8 2 1	3-8	0111 1011	
124	7C	ME	@	@	8 4	4-8	0111 1100	
125	7D	DE	'	'	8 4 1	5-8	0111 1101	'
126	7E	AU	>	=	8 4 2	6-8	0111 1110	=
127	7F	SU	√	"	8 4 2 1	7-8	0111 1111	

RX FORMAT R1,D2(X2,B2) R1,S2(X2) R1,D2(0,B2) R1,S2

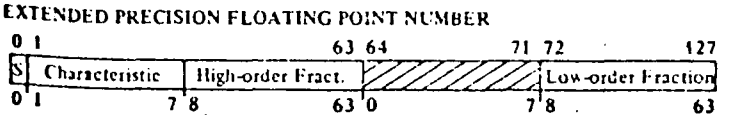


SHORT FLOATING POINT NUMBER



LONG FLOATING POINT NUMBER – same as short floating point number except fraction is longer – bits 8 through 63

EXTENDED PRECISION FLOATING POINT NUMBER



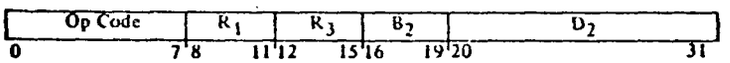
RS, SI FORMAT INSTRUCTIONS

Decimal	Hexadecimal	Mnemonic	Graphic & Control Symbols BCDIC	Graphic & Control Symbols EBCDIC	(2) 7-Track Tape BCDIC	Punched Card Code	System/360 8-Bit Code
128	80	SSM				12-0-1-8	1000 0000
129	81			a		12-0-1	1000 0001
130	82	LPSW		b		12-0-2	1000 0010
131	83	(Diagnose)		c		12-0-3	1000 0011
132	84	WRD		d		12-0-4	1000 0100
133	85	RDD		e		12-0-5	1000 0101
134	86	BXH		f		12-0-6	1000 0110
135	87	BXLE		g		12-0-7	1000 0111
136	88	SRL		h		12-0-8	1000 1000
137	89	SLL		i		12-0-9	1000 1001
138	8A	SRA				12-0-2-8	1000 1010
139	8B	SLA				12-0-3-8	1000 1011
140	8C	SRDL				12-0-4-8	1000 1100
141	8D	SLDL				12-0-5-8	1000 1101
142	8E	SRDA				12-0-6-8	1000 1110
143	8F	SLDA				12-0-7-8	1000 1111
144	90	STM				12-11-1-8	1001 0000
145	91	TM		j		12-11-1	1001 0001
146	92	MVI		k		12-11-2	1001 0010
147	93	TS		l		12-11-3	1001 0011
148	94	NI		m		12-11-4	1001 0100
149	95	CLI		n		12-11-5	1001 0101
150	96	OI		o		12-11-6	1001 0110
151	97	XI		p		12-11-7	1001 0111
152	98	LM		q		12-11-8	1001 1000
153	99			r		12-11-9	1001 1001
154	9A					12-11-2-8	1001 1010
155	9B					12-11-3-8	1001 1011
156	9C	SIO				12-11-4-8	1001 1100
157	9D	TIO				12-11-5-8	1001 1101
158	9E	HIO				12-11-6-8	1001 1110
159	9F	TCH				12-11-7-8	1001 1111
160	A0					11-0-1-8	1010 0000
161	A1					11-0-1	1010 0001
162	A2			s		11-0-2	1010 0010
163	A3			t		11-0-3	1010 0011
164	A4			u		11-0-4	1010 0100
165	A5			v		11-0-5	1010 0101
166	A6			w		11-0-6	1010 0110
167	A7			x		11-0-7	1010 0111
168	A8			y		11-0-8	1010 1000
169	A9			z		11-0-9	1010 1001
170	AA					11-0-2-8	1010 1010
171	AB					11-0-3-8	1010 1011
172	AC					11-0-4-8	1010 1100
173	AD					11-0-5-8	1010 1101
174	AE					11-0-6-8	1010 1110
175	AF					11-0-7-8	1010 1111
176	B0					12-11-0-1-8	1011 0000
177	B1					12-11-0-1	1011 0001
178	B2					12-11-0-2	1011 0010
179	B3					12-11-0-3	1011 0011
180	B4					12-11-0-4	1011 0100
181	B5					12-11-0-5	1011 0101
182	B6					12-11-0-6	1011 0110
183	B7					12-11-0-7	1011 0111
184	B8					12-11-0-8	1011 1000
185	B9					12-11-0-9	1011 1001
186	BA					12-11-0-2-8	1011 1010
187	BB					12-11-0-3-8	1011 1011
188	BC					12-11-0-4-8	1011 1100
189	BD					12-11-0-5-8	1011 1101
190	BE					12-11-0-6-8	1011 1110
191	BF					12-11-0-7-8	1011 1111

RS FORMAT

R1,R3,D2(B2) } BXH, BXLE
R1,R3,S2 } LM, STM

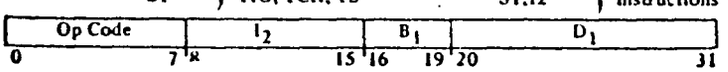
R1,D2(B2) } Shift instructions
R1,S2 }



SI FORMAT

D1(B1) } LPSW, SSM, HIO, SIO
SI } TIO, TCH, TS

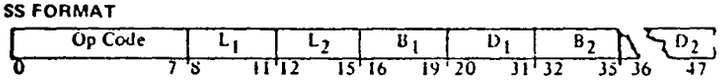
D1(B1),I2 } All other SI instructions
SI,I2 }



SS FORMAT INSTRUCTIONS

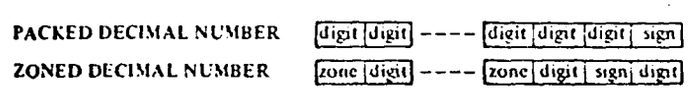
Decimal	Hexadecimal	Mnemonic	Graphic & Control Symbols BCDIC	Graphic & Control Symbols EBCDIC	(2) 7-Track Tape BCDIC	Punched Card Code	System/360 8-Bit Code
192	C0				B A 8 2	12-0	1100 0000
193	C1		A	A	B A 1	12-1	1100 0001
194	C2		B	B	B A 2	12-2	1100 0010
195	C3		C	C	B A 2 1	12-3	1100 0011
196	C4		D	D	B A 4	12-4	1100 0100
197	C5		E	E	B A 4 1	12-5	1100 0101
198	C6		F	F	B A 4 2	12-6	1100 0110
199	C7		G	G	B A 4 2 1	12-7	1100 0111
200	C8		H	H	B A 8	12-8	1100 1000
201	C9		I	I	B A 8 1	12-9	1100 1001
202	CA					12-0-2-8-9	1100 1010
203	CB					12-0-3-8-9	1100 1011
204	CC					12-0-4-8-9	1100 1100
205	CD					12-0-5-8-9	1100 1101
206	CE					12-0-6-8-9	1100 1110
207	CF					12-0-7-8-9	1100 1111
208	D0		!		B 8 2	11-0	1101 0000
209	D1	MVN	J	J	B 1	11-1	1101 0001
210	D2	MVC	K	K	B 2	11-2	1101 0010
211	D3	MVZ	L	L	B 2 1	11-3	1101 0011
212	D4	NC	M	M	B 4	11-4	1101 0100
213	D5	CLC	N	N	B 4 1	11-5	1101 0101
214	D6	OC	O	O	B 4 2	11-6	1101 0110
215	D7	XC	P	P	B 4 2 1	11-7	1101 0111
216	D8		Q	Q	B 8	11-8	1101 1000
217	D9		R	R	B 8 1	11-9	1101 1001
218	DA					12-11-2-8-9	1101 1010
219	DB					12-11-3-8-9	1101 1011
220	DC	TR				12-11-4-8-9	1101 1100
221	DD	TRT				12-11-5-8-9	1101 1101
222	DE	ED (3)				12-11-6-8-9	1101 1110
223	DF	EDMK (3)				12-11-7-8-9	1101 1111
224	E0		‡		A 8 2	0-2-8	1110 0000
225	E1					11-0-1-9	1110 0001
226	E2		S	S	A 2	0-2	1110 0010
227	E3		T	T	A 2 1	0-3	1110 0011
228	E4		U	U	A 4	0-4	1110 0100
229	E5		V	V	A 4 1	0-5	1110 0101
230	E6		W	W	A 4 2	0-6	1110 0110
231	E7		X	X	A 4 2 1	0-7	1110 0111
232	E8		Y	Y	A 8	0-8	1110 1000
233	E9		Z	Z	A 8 1	0-9	1110 1001
234	EA					11-0-2-8-9	1110 1010
235	EB					11-0-3-8-9	1110 1011
236	EC					11-0-4-8-9	1110 1100
237	ED					11-0-5-8-9	1110 1101
238	EE					11-0-6-8-9	1110 1110
239	EF					11-0-7-8-9	1110 1111
240	F0		0	0	8 2	0	1111 0000
241	F1	MVO	1	1	1	1	1111 0001
242	F2	PACK	2	2	2	2	1111 0010
243	F3	UNPK	3	3	2 1	3	1111 0011
244	F4		4	4	4	4	1111 0100
245	F5		5	5	4 1	5	1111 0101
246	F6		6	6	4 2	6	1111 0110
247	F7		7	7	4 2 1	7	1111 0111
248	F8	ZAP (3)	8	8	8	8	1111 1000
249	F9	CP (3)	9	9	8 1	9	1111 1001
250	FA	AP (3)				12-11-0-2-8-9	1111 1010
251	FB	SP (3)				12-11-0-3-8-9	1111 1011
252	FC	MP (3)				12-11-0-4-8-9	1111 1100
253	FD	DP (3)				12-11-0-5-8-9	1111 1101
254	FE					12-11-0-6-8-9	1111 1110
255	FF					12-11-0-7-8-9	1111 1111

SS FORMAT



D1(L,B1),D2(B2) } NC, OC, XC, CLC / MVC, MVN, MVZ / TR, TRT, ED, EDMK
S1(L),S2

D1(L1,B1),D2(L2,B2) } PACK, UNPK / MVO, AP, CP / DP, MP, SP, ZAP
S1(L1),S2(L2)



LEHIGH UNIVERSITY IBM 360 ASSEMBLER

```
CARD NO.   ERROR   LOCATION OBJECT CODE    ---------- SOURCE STATEMENTS --------

     1             00000                  TEST    START
     2             00000   05F0           BEGIN   BALR    15,0          SET REG 15 AS BASE REGISTER
     3                                             USING   BEGIN+2,15
     4             00002   90000056       READC   STO     R,RDLIST      READ IN DATA 3 WORDS
     5             00006   5810F046               L       1,A           LOAD FIRST DATA INPUT
     6             0000A   5510F04A               CL      1,B           COMPARE FIRST TO SECOND
     7             0000E   4720F034               BC      2,AHIGH       BRANCH IF A IS GREATER THAN B
     8             00012   4780F034               BC      8,AHIGH       BRANCH IF A=B
     9             00016   5810F04A               L       1,B           B IS GREARER THAN A
    10             0001A   5510F04E               CL      1,C           COMPARE B TO C
    11             0001E   4720F028       T       BC      2,STORE       STORE B IT IS LARGEST
    12             00022   4780F028               BC      8,STORE       STORE B, B=C
    13             00026   5810F04E       CHIGH   L       1,C           C IS LARGEST
    14             0002A   5010F052       STORE   ST      1,D           STORE LARGEST IN D
    15             0002E   9001F056               SIO     W,WRTLIST     PRINT LARGEST VALUE
    16             00032   47F0F000               BC      15,READC      START OVER AGAIN
    17             00036   5510F04E       AHIGH   CL      1,C           COMPARE A TO C
    18             0003A   4720F028               BC      2,STORE       STORE A, A IS GREATER THAN C
    19             0003E   4780F028               BC      8,STORE       STORE A, A=C
    20             00042   47F0F024               BC      15,CHIGH      C IS GREATER THAN A
    21             00046                  A       DS      1F
    22             0004A                  B       DS      1F
    23             0004E                  C       DS      1F
    24             00052                  D       DS      1F
    25             00056   0007F046       RDLIST  IO      3,A           READ 3 WORDS AND STORE HERE
    26             0005A   0001F052       WRTLIST IO      1,D           WRITE 1 WORD STARTING HERE
    27                                             END

        0 ERRORS

        PROGRAM FIRST BYTE ADDRESS (HEXIDECIMAL)   00000
        PROGRAM NEXT AVAILABLE BYTE ADDRESS (HEXIDECIMAL)   00060
        PROGRAM LENGTH IN BYTES (DECIMAL)          96

        OBJECT CODE PASSED TO FILE PGM

05F090C00F05658 10F0465510F04A4720F0344780F0345810F04A5510F04E4720F0284780F0285910
F04E5010F0529C01F05A47F0F0005510F04E4720F0284780F02847F0F0240000000000000000000000
0000000000000000003F0460001F052
```

SYMBOL TABLE

SYMBOL	VALUE	LENGTH	RELOCATION
TEST	00000	1	R
BEGIN	00000	2	R
READC	00002	4	R
T	0001E	4	R
CHIGH	00026	4	R
STORE	0002A	4	R
AHIGH	00036	4	R
A	00048	4	R
B	0004C	4	R
C	00050	4	R
D	00054	4	R
RDLIS	00058	4	R
WRTLI	0005C	4	R

13 SYMBOLS IN TABLE

END OF SYMBOL TABLE

LITERAL TABLE

LITERAL	VALUE	CONTENTS	LENGTH	RELOCATION

0 LITERALS IN TABLE

END OF LITERAL TABLE

BASE TABLE ASSIGNMENTS

LINE NUMBER	AVAILABILITY	REGISTER	CONTENTS
3	Y	15	00002

END OF BASE TABLE ASSIGNMENTS

LEHIGH UNIVERSITY IBM 360 ASSEMBLER

```
CARD NO.   ERROR   LOCATION OBJECT CODE   ---------------------------- SOURCE STATEMENTS ----------------------------

   1         1                             AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
   2        15     00000    58100000       A12:        L       1,=A(SAM)
   3         3                             LABLE
   4         5                                         DC
   5         6                             LABL1       DC      12346678F(1)
   6         7                                         DC      V(1)
   7                00004    00000001                  DC      F(1,X)
             8
   8        13                                         START   10
   9         9                                         CC      X(H)
  10        10                                         EQU     1
  11        11                             NOW         EQU     1A
  12        12     00008    1B11           LABL1       SR      1,1
  13        12     0000A    1B11           LABL1       SR      1,1
  14         2     0000C    1B11           A12^        SR      1,1
  15        14                                         ORG     A
  16        15     0000E    1B10                       SR      1,NOW
  17        16     00010    1800                       SR
  18         4     00014    00000000                   END

        17 ERRORS
```

ERROR TYPES ENCOUNTERED

```
TYPE 1  NO POSSIBLE WAY TO ANALYZE THIS CARD. THE CARD HAS BEEN DELETED. .....HAVE YOU READ THE REFERENCE MANUAL+
TYPE 2  A LABEL MUST START WITH AN ALPHA CHARACTER. THE REST OF THE LABEL MUST BE ALPHA-NUMERIC. .....LABEL DELETED
TYPE 3  NO VALID OPERATION OR PSEUDO-OPERATION FOUND. .....THE CARD HAS BEEN DELETED
TYPE 4  UNABLE TO ASSIGN VALUE TO ADDRESS LITERAL
TYPE 5  NO EXPRESSION FOUND IN DC OR DS STATEMENT
TYPE 6  IMPROPER MULTIPLIER IN DC OR DS STATEMENT
TYPE 7  IMPROPER DATA TYPE IN DC OR DS STATEMENT
TYPE 8  UNABLE TO EVALUATE ENTIRE PRECEEDING DC STATEMENT
TYPE 9  UNABLE TO EVALUATE CONSTANT FOLLOWING VALID DATA TYPE IN DC STATEMENT
TYPE 10 UNABLE TO EVALUATE OPERAND
TYPE 11 NO LABEL FOUND IN EQUATE STATEMENT
TYPE 12 DUPLICATE LABEL
TYPE 13 A START CARD MUST APPEAR BEFORE ANY CODE IS GENERATED
TYPE 14 INVALID OPERAND IN ORG OR START STATEMENT
TYPE 15 UNABLE TO EVALUATE ALL OR PART OF OPERAND
TYPE 16 NO OPERAND FOUND
```

```
PROGRAM FIRST BYTE ADDRESS (HEXIDECIMAL)  00000
PROGRAM NEXT AVAILABLE BYTE ADDRESS (HEXIDECIMAL)  00018
PROGRAM LENGTH IN BYTES (DECIMAL)          24

OBJECT CODE PASSED TO FILE PGM

581000000000000118111811181118101800000000000000

                    SYMBOL TABLE
                    SYMBOL          VALUE       LENGTH     RELOCATION

          DUPLICATE LABL1           00008         2            R

                      1  SYMBOLS IN TABLE

                    END OF SYMBOL TABLE

                    LITERAL TABLE
                    LITERAL         VALUE         CONTENTS      LENGTH  RELOCATION

                    A(SAM)          00014                         4         R

                      1  LITERALS IN TABLE

                    END OF LITERAL TABLE

                    BASE TABLE ASSIGNMENTS

                    LINE NUMBER  AVAILABILITY  REGISTER  CONTENTS


                    END OF BASE TABLE ASSIGNMENTS
```

Vita

Henry P. Seager, son of Henry B. and Gertrude R. Seager was born on April 24, 1944 in Mahanoy City, PA. He did his undergraduate work at Villanova University and received the Degree of Bachelor of Electrical Engineering, Summa Cum Laude, in 1966. He is a member of TAU BETA PI and ETA KAPPA NU.

The author is currently a Senior Project Engineer, System Operating Computer Systems, employed by the Pennsylvania Power and Light Company, Allentown, PA.