

DECEMBER 1968

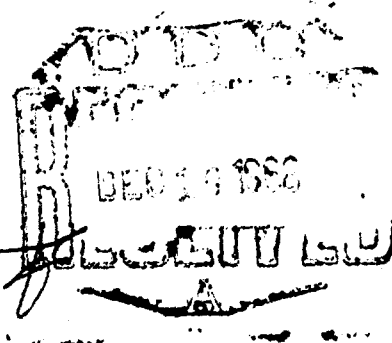
AD 679272

SYSTEM AND SOFTWARE SIMULATOR
VOLUME IV

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

UNITED STATES ARMY
COMPUTER SYSTEMS SUPPORT
AND EVALUATION COMMAND
WASHINGTON, D.C. 20310

Reproduced by the
CLEARINGHOUSE
for Federal Scientific & Technical
Information Springfield Va. 22151



3200.8 (Att 1 to Encl 1)
Mar 7, 66

Security Classification		
DOCUMENT CONTROL DATA - R & D		
(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)		
1. ORIGINATING ACTIVITY (Corporate author) Leo J. Cohen Associates 334 West State Street Trenton, New Jersey		2a. REPORT SECURITY CLASSIFICATION Unclassified
3. REPORT TITLE SYSTEM AND SOFTWARE SIMULATOR VOLUME IV		2b. GROUP
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) Working Paper		
5. AUTHOR(S) (First name, middle initial, last name) Leo J. Cohen		
6. REPORT DATE December 1968	7a. TOTAL NO. OF PAGES 424	7b. NO. OF REFS
6a. CONTRACT OR GRANT NO. DAAB09-68-C-0118	9a. ORIGINATOR'S REPORT NUMBER(S)	
b. PROJECT NO.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
c.		
d.		
10. DISTRIBUTION STATEMENT Distribution of this document is unlimited.		
11. SUPPLEMENTARY NOTES This documentation has been assembled and released by the sponsoring activity		12. SPONSORING MILITARY ACTIVITY USA Computer Systems Support and Evaluation Command Washington, D. C.
13. ABSTRACT The System and Software Simulator (S3) is a digital event simulator written in FORTRAN IV and designed to perform simulations of computer systems hardware and software and of the workload being applied to the system. This and the other three volumes constitute the complete documentation available on S3. Volume I describes the inputs, outputs, methods and capabilities of S3. Volume II contains the flowcharts, narrative description of the flowcharts, layouts and descriptions of the tables utilized by S3. Volume III contains descriptions of the assembly language used for preparation of input to S3, of the macro capability of the assembler, and of the modifications made to S3 to provide additional output data. Volume IV is the program documentation on the internal workings of the assembler. It consists of table descriptions, flow charts and narratives, and file descriptions. These volumes are a collection of documentation delivered under two separate contracts. They have not been edited and as such are considered working papers.		

DD FORM 1473
1 NOV 65

Security Classification

3200.8 (Att 1 to Encl 1)
Mar 7, 66

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Simulation Computer Systems Operating Systems Evaluation Timing Dynamic Event Simulation						

WORKING PAPER

The documentation on the System and Software Simulator (S3) contained in this and the other three volumes is considered a working paper and no claims are made as to its accuracy. There has been no attempt to edit the information. Discrepancies and inconsistencies are known to exist.

This information is being released as a service to interested parties and to satisfy the numerous requests for information on S3.

The documentation of S3 is contained in four volumes. Volumes I and II are contract end items delivered under contract number DA-49-083 OSA-3306 and contain the technical descriptions of S3. Volume I describes the inputs, outputs, methods and capabilities of S3. Volume II contains the flowcharts, narrative description of the flowcharts, layouts and descriptions of the tables utilized by S3.

Volumes III and IV contain the documentation delivered as contract end items under contract number DAAB09-68-C-0118. Volume III contains descriptions of the assembly language used for preparation of input to S3, of the macro capability of the assembler, and of the modifications made to S3 to provide additional output data. Volume IV is the program documentation on the internal workings of the assembler. It consists of table descriptions, flow charts and narratives, and file descriptions.

WORKING PAPER

WORKING PAPER

USACSSEC S3
Assembler

Programmer's Manual

Presented by

Leo J. Cohen Associates, Inc.
334 West State Street
Trenton, New Jersey
(609) 695-1488

November 8, 1968

WORKING PAPER

TABLE OF CONTENTS

	<u>Page</u>
Introduction	1
Section I - Table Descriptions	2
Section II - Flow Charts and Narrative	86
Section III - File Descriptions	402

Section I - Table Descriptions

	<u>Page</u>
Table Descriptions	3
Table 1 - General Assembler Table	6
Table 2 - Primary Input Area	17
Table 3 - Fixed Fields Output Area	18
Table 4 - Global Dictionary	21
Table 5 - Local Dictionary	25
Table 6 - Global Hash Table	28
Table 7 - Local Hash Table	30
Table 8 - Macro Prototype Area	32
Table 9 - Current Output Area	34
Table 10 - Second Pass Output Area	35
Table 11 - Second Pass Table	37
Table 12 - Title Area	40
Table 13 - CPU Definition Table	41
Table 14 - MEM-DEF Table	43
Table 15 - CHAN-DEF Table	45
Table 16 - DECIVE-DEF Table	47
Table 17 - CPU Configuration Table	49
Table 18 - Memory Configuration Table	51
Table 19 - Channel Configuration Table	53
Table 20 - Control Configuration Table	55
Table 21 - Device Configuration Table	57
Table 22 - To-From Table	59
Table 23 - Queue Table	61
Table 24 - Real File Table	63
Table 25 - Input-Left Justified by Character	65
Table 26 - Load Class Table	66
Table 27 - Run Class Table	68
Table 28 - Table Dump Control Table	70
Table 29 - Statistics Control Table	72
Table 30 - Program Distribution Table	74
Table 31 - Interrupt Vector Table	76
Table 32 - Operation Code Table	78
Table 33 - Job Name Table	80
Table 34X - Ordinal File Name Table	82
Table 35 - O/S Memory Allocation Table	84

Section II - Flow Charts and Narrative

	<u>Page</u>
Assembler Main Program	87
Error Message Formatting Main Program	89
Op-Code Sequencing Program	95
Subroutine ASML	98
Subroutine ASP	123
Subroutine CONV	164
Subroutine ERPASS	167
Subroutine EVAL	174
Subroutine GGSYM	177
Subroutine GLSYM	179
Subroutine HASH	181
Subroutine NERR	183
Subroutine PAGE	185
Subroutine PGSYM	187
Subroutine PLSYM	190
Subroutine RBLK	193
Subroutine RZRO	195
Subroutine SA	197
Subroutine SA1	199
Subroutine SA2	206
Subroutine SA3	208
Subroutine SA4(SA5)	210
Subroutine SA6	231
Subroutine SA7	233
Subroutine SA8	235
Subroutine SA9	237
Subroutine SERR	240
Subroutine SMACRO	242
Subroutine SOPCD	253
Subroutine SX1	255
Subroutine SX2	265
Subroutine SX3	273
Subroutine SX4	284
Subroutine SX5	290
Subroutine SX51	290
Subroutine SX52	290
Subroutine SX53	301

	<u>Page</u>
Subroutine SX54	304
Subroutine SX55	308
Subroutine SX56	313
Subroutine SX6	16
Subroutine SX7	319
Subroutine SX8	322
Subroutine SX9	324
Subroutine SX10	326
Subroutine SX11	330
Subroutine SX12	332
Subroutine SX13	334
Subroutine SX14	337
Subroutine SX15	341
Subroutine SX16	343
Subroutine SX17	345
Subroutine SX18	349
Subroutine SX19	351
Subroutine SX20	353
Subroutine SX21	357
Subroutine SX22	359
Subroutine SX23	363
Subroutine SX24	366
Subroutine SX25	368
Subroutine SX26	372
Subroutine SX27	374
Subroutine SX28	376
Subroutine SX29	378
Subroutine SX30	381
Subroutine SX31	383
Subroutine SX32	386
Subroutine SX33	389
Subroutine SX35	392
Subroutine SX36	394
Subroutine SX37	396
Subroutine S1	400

Section III - File Descriptions

	<u>Page</u>
FORTTRAN I/O Unit Assignments	403
Intermediate Assembly File	404
Local Symbol Table File	406
Copy Input File	408
Macro Files A and B	410
First Pass Error Output	412
Second Pass Error Output	414
Macro Definition File	416
Diagnostics File	420

INTRODUCTION

This programmer's manual contains the documentation for the internal workings of the S3 assembler.

The documentation consists of table descriptions, flow charts and narratives, and file descriptions which will be found in Section I, Section II, and Section III respectively.

WORKING PAPER

2

SECTION I

TABLE DESCRIPTIONS

WORKING PAPER

TABLE DESCRIPTIONS

<u>FORTTRAN TABLE</u>	<u>DESCRIPTION</u>
IT1 (100)	GENERAL ASSEMBLER TABLE
AT2 (1)-(80)	PRIMARY INPUT AREA (CHAR)
AT2 (81)-(94)	PRIMARY INPUT AREA (WORD)
IT3 (175)	FIX FIELDS OUTPUT AREA
IT4 (800,5)	GLOBAL DICTIONARY
IT5 (800,5)	LOCAL DICTIONARY
IT6 (90)	GLOBAL HASH TABLE
IT7 (90)	LOCAL HASH TABLE
IT8 (175)	MACRO PROTOTYPE AREA
IT9 (14)	CUR OUTPUT AREA
IT10 (9)	SECOND PASS OUTPUT AREA
IT11 (100,9)	SECOND PASS TABLE
IT12 (14)	TITLE AREA
IT13 (44,5)	CPU-DEF TABLE
IT14 (9,30)	MEM-DEF TABLE
IT15 (9,50)	CHAN-DEF TABLE
IT16 (13,50)	DEVICE-DEF TABLE

FORTRAN TABLE

IT17 (3,5)
IT18 (8,30)
IT19 (8,50)
IT20 (12,50)
IT21 (13,100)
IT22 (16)
IT23 (5,30)
IT24 (8,150)
IT25 (80)
IT26 (5,16)
IT27 (5,6)
IT28 (25)
IT29 (10)
IT30 (3,100)
IT31 (22,5)
IT32 (3,150)
IT33 (3,100)
IT34X (4,200)
IT35 (10)

DESCRIPTION

CPU CONFIGURATION TABLE
MEMORY CONFIGURATION TABLE
CHANNEL CONFIGURATION TABLE
CONTROL CONFIGURATION TABLE
DEVICE CONFIGURATION TABLE
TO-FROM TABLE
QTABLE
REAL FILE TABLE
INPUT-LEFT JUSTIFIED BY CHARACTER
LOAD CLASS TABLE
RUN CLASS TABLE
TABLE DUMP CONTROL TABLE
STATISTICS CONTROL TABLE
PROGRAM DISTRIBUTION TABLE
INTERRUPT VECTOR TABLE
OP-CODE TABLE
JOB NAME TABLE
ORDINAL FILE NAME TABLE
O/S MEMORY ALLOCATION TABLE

ERROR CODE RANGES

100	ASM1 - PASS 1
200	PLSYM, GLSYM, PGSYM, GGSYM
300	ASM2 - PASS 2
400	HARDWARE DEFINITIONS
500	FIX FIELDS
600	SYSTEM PARAMETERS

TABLE 1 - GENERAL ASSEMBLER TABLE

Table 1 contains all one word variables which are referenced in two or more subroutines.

A description of each field in table 1 can be found starting on the next page.

GENERAL ASSEMBLER TABLEIT1(1) ISW1

SUBROUTINE SA4 (FIX FIELDS ROUTINE)

RETURN CODE

1 = COMMENTS CARD

2 = NORMAL STATEMENT

3 = CONTINUATION REQUIRED

IT1(2) ISW2

MACRO DEFINITION SWITCH

0 = NO MACRO DEFINITION IN PROGRESS

1 = MACRO DEFINITION IN PROGRESS

IT1(3) MACNBR

CURRENT MACRO SUFFIX VALUE

IT1(4) N

CURRENT INPUT CHARACTER BEING EXAMINED

IT1(5) CPSW

OPERAND SWITCH

0 = NO MORE OPERANDS

1 = ADDITIONAL OPERANDS

IT1(6) STNBR
CURRENT STATEMENT NUMBER

IT1(7) MIN
MACRO INPUT SW
0 = NOTHING ON MACRO INPUT FILE
1 = DATA ON MACRO INPUT FILE

IT1(8) NGF INITIAL VALUE = 1
NUMBER OF NEXT FREE ENTRY IN LOCAL
SYMBOL TABLE

IT1(9) NLF INITIAL VALUE = 1
NUMBER OF NEXT FREE ENTRY IN LOCAL
SYMBOL TABLE

IT1(10) NG INITIAL VALUE = 800
MAXIMUM NUMBER OF ENTRIES IN GLOBAL
SYMBOL TABLE

IT1(11) NL INITIAL VALUE = 800
MAXIMUM NUMBER OF ENTRIES IN LOCAL
SYMBOL TABLE

IT1(12) CIN

COPY INPUT SWITCH

0 = NOTHING ON COPY INPUT FILE

1 = DATA ON COPY INPUT FILE

IT1(13) PEOF

PRIMARY INPUT (CARD READER)

END OF FILE SWITCH

0 = NO END OF FILE

1 = END OF FILE

IT1(14) CEOF

COPY END OF FILE SWITCH

0 = NO END OF FILE

1 = END OF FILE

IT1(15) MEOF

MACRO END OF FILE SWITCH

0 = NO END OF FILE

1 = END OF FILE

IT1(16) ISW3

RESOLVE OP-CD OUTPUT SWITCH

0 = NORMAL OP-CD

1 = SPECIAL OP-CD

2 = UNRESOLVED OR MACRO OP-CD

IT1(17) IASM

ASSEMBLY STATEMENT SWITCH

0 = NOT RECEIVED

1 = RECEIVED

IT1(18) OPCD

CURRENT OPERATION CODE

IT1(19)-IT1(23)

NEXT EXPECTED OP-CD TABLE

IT1(24) BLANKS INITIAL VALUE = 6 BLANKS

IT1(25) CURUNT INITIAL VALUE = 15

UNIT FOR CUR OUTPUT TAPE

IT1(26) SEQCHK

SEQUENCE CHECKING COUNTER FOR SECOND PASS

IT1(27) NWR

NUMBER OF WORKER ROUTINE CURRENTLY BEING
PROCESSED

IT1(28) ERRSW

ERROR SWITCH

0 = CURRENT STATEMENT HAS NO ERROR

1 = CURRENT STATEMENT HAS AN ERROR

IT1(29) ILNCT
NUMBER OF LINES LEFT ON CURRENT PAGE

IT1(30) IPGCT
CURRENT PAGE COUNT

IT1(31) MACINP INITIAL VALUE = 10
CURRENT MACRO INPUT FILE

IT1(32) MACOPT INITIAL VALUE = 11
CURRENT MACRO OUTPUT FILE

IT1(33) ICPUD
CPU DEFINITION COUNTER

IT1(34) ICFACT
ADJUSTMENT FACTOR FOR COMPUTE STATEMENTS

IT1(35) IMEND
MEMORY DEFINITION COUNTER

IT1(36) ICHAND
CHANNEL DEFINITION COUNTER

IT1(37) ICHCT
CHANNEL TABLE ENTRY COUNTER

IT1(38) PRTSW
0 = DON'T PRINT SECOND PASS OUTPUT
1 = PRINT SECOND PASS OUTPUT

IT1(39)	<u>JSTNBR</u>	JOB STATEMENT NUMBER
IT1(40)	<u>ICON1</u>	CPU CONFIGURATION COUNTER
IT1(41)	<u>ICON2</u>	MEM CONFIG CTR
IT1(42)	<u>ICON3</u>	CHAN CONFIG CTR
IT1(43)	<u>ICON4</u>	CTL CONFIG CTR
IT1(44)	<u>ICON5</u>	DEV CONFIG CTR
IT1(45)	<u>ITFP</u>	TO-FROM TABLE IN PROGRESS SW
IT1(46)	<u>IROW</u>	CURRENT TO-FROM ROW
IT1(47)	<u>INFL</u>	NUMBER TO-FROM FILES
IT1(48)	<u>IDEV</u>	DEVICE # FOR CURRENT TO-FROM TABLE

IT1(49) IMER

IT1(50) IQR

QUEUE DEFINITION SWITCH

0 = Q-DEF NOT RECEIVED

1 = Q-DEF RECEIVED

2 = Q-END RECEIVED

IT1(51) IQCTR

QUEUE COUNTER

IT1(52) IQENT

QUEUE ENTRY COUNTER

IT1(53) MULT

COMPUTE MULTIPLICATION FACTOR

IT1(54) NSUM

HASH VALUE OF SYMBOL

IT1(55) IFLR

FILE DEFINITION CONTROL SW

0 = FILES NOT RECEIVED

1 = FILES BEING RECEIVED

2 = FILES RECEIVED

IT1(56) IFLCTR

REAL FILE COUNTER

IT1(57) ILCR

LOAD CLASS CONTROL SW

0 = LOAD CLASS LOT RECEIVED

1 = LOAD CLASS BEING RECEIVED

2 = LOAD CLASS RECEIVED

IT1(58) IRCR

RUN CLASS CONTROL SW

0 = RUN CLASS NOT RECEIVED

1 = RUN CLASS BEING RECEIVED

2 = RUN CLASS RECEIVED

IT1(59) ITAB

TABLE DUMP CONTROL SW

0 = TABLE DUMP CTL NOT RECEIVED

1 = TABLE DUMP CTL RECEIVED

IT1(60) ISTAT

STATISTICS CONTROL SW

0 = STATISTICS CONTROL NOT RECEIVED

1 = STATISTICS CONTROL RECEIVED

IT1(61) KINT

STATISTICS INTERVAL RECEIVED SW

0 = NOT RECEIVED

1 = RECEIVED

IT1(62) JOBSW

JOB SWITCH

0 = NO JOB IN PROGRESS

1 = JOB IN PROGRESS

IT1(63) JOBCTR

JOB COUNTER

IT1(64) OFCTR

ORDINAL FILE COUNTER

IT1(65) MINP

MORE INPUT SWITCH USED IN SMACRO

0 = OFF

1 = ON

IT1(66) ANPS

ANOTHER PASS SWITCH USED IN SMACRO

0 = OFF

1 = ON

IT1(67) HIMAC

HIGH MACRO NBR COUNTER USED IN SMACRO

IT1(68) ERFIL

CURRENT FORTRAN UNIT NUMBER FOR ERROR OUTPUT

IT1(69) OFCTR

ORDINAL FILE COUNTER

IT1(70) KCOUNT

NUMBER OF STATISTICAL INTERVALS FOR SIMULATION

IT1(71) STSW

STORE SWITCH

0 = NO STORE IN PROGRESS

1 = STORE IN PROGRESS

TABLE 2 - PRIMARY INPUT AREA

Table 2 is divided into two sections. The first 80 words, AT2(1) to AT2(80), contain 80 characters as read from the current input source. Each word contains one character left justified and right filled with blanks. The second section of table 2 consists of 14 words, AT2(81) to AT2(94). The first 13 words of this section contain six characters each and the 14th word contains two characters left justified and right filled with blanks.

TABLE 3 - FIXED FIELDS OUTPUT AREA

Table 3 is the area used to store the output from the fix fields routine (SA4, SA5). Word 1 is not normally used. Word 2 contains the total number of operands in this statement in the first 12 bits. Word 2 also contains the two character statement variable, if any, in the last 12 bits. Words 3 and 4 contain up to 12 characters of the statement label. Word 5 contains the length of the statement label in the first 6 bits. The next 12 bits are used to hold a code which indicates if the label is fixed or variable. Labels may be variable only when a macro definition is in progress. The next 12 bits contain a code which indicates if a macro number is to be appended to the label. This code may be set only if a macro definition is currently in progress. Words 6 and 7 contain up to 12 characters of operation code. Word 8 contains the length of the operation code in the first 6 bits. The next 12 bits contain a code

which indicates if the operation code is fixed or variable. Variable operation codes are permitted only in macro definitions. Words 9 and 10 contain up to 12 characters of the first operand for this statement. Word 11 contains the length of the first operand in the first 6 bits. The next 12 bits contain a code which indicates if the first operand is fixed or variable. The next 12 bits indicate whether a macro number should be appended to this operand or not. The remainder of table 3 consists of three word units each of which is used to describe a single operand. A maximum of 55 operands may be accommodated by table 3.

TABLE 3

FIXED FIELDS OUTPUT AREA

WORD	6 bits	6 bits	6 bits	6 bits	6 bits	6 bits
1	NOT USED					
2	NUMBER OF OPERANDS			STATEMENT VARIABLE		
3	LABEL			PART I		
4	LABEL			PART II		
5	LABEL LENGTH	0 = FIXED LABEL N = VARIABLE LABEL	0 = NO MACRO # 99 = ADD MACRO #		0	
6	OP-CD			PART I		
7	OP-CD			PART II		
8	OP-CD LENGTH	0 = FIXED OP-CD 99 = VARIABLE OP-CD			0	
9	OPERAND-1			PART I		
10	OPERAND-1			PART II		
11	OPERAND-1 LENGTH	0 = FIXED OPERAND 99 = VARIABLE OPERAND	0 = NO MACRO # 99 = ADD MACRO #		0	
171	OPERAND 55			PART I		
172	OPERAND 55			PART II		
173	OPERAND-55 LENGTH	0 = FIXED OPERAND 99 = VARIABLE OPERAND	0 = NO MACRO # 99 = ADD MACRO #		0	

TABLE 4 - GLOBAL DICTIONARY

The Global Dictionary, table 4, can contain up to 800 five word entries. Each entry is used to describe a single global symbol. The following page describes the various types of symbols which may be found in the global symbol table.

The first two words of a Global Dictionary entry contain up to 12 characters of the symbol. The 3rd word of an entry contains the type of symbol as described on the following page. The 4th word of an entry contains the value which has been assigned to this symbol. The 5th word of an entry may contain a pointer to subsequent entries on this chain in the Global Dictionary. For a brief description of the use of table 4 see the PGSYM, GGSYM, and the HASH subroutine descriptions.

GLOBAL SYMBOL TABLE TYPES

<u>TYPE</u>	<u>DESCRIPTION</u>
1	AT = 1 IOT = 2
2	LIB = 1 NCAT = 2 CAT = 3
3	cpu-name
4	mem-names
5	chan-name
6	IN = 1 OUT = 2 I/O = 3
7	ctl-names
8	SEIZE = 1 NOSEIZE = 2
9	dev-names
10	queue-names
11	NOT USED
12	PRI = 1 FIFO = 2 LIFO = 3

<u>TYPE</u>	<u>DESCRIPTION</u>
13	real-file-names
14	function-names
15	global-equates
16	job-name
17	interrupt-names

TABLE 4GLOBAL DICTIONARYWORD

1

SYMBOL PART I

2

SYMBOL PART II

3

SYMBOL TYPE

4

SYMBOL VALUE

5

CHAIN INDEX

TABLE 5 - LOCAL DICTIONARY

The Local Dictionary can contain up to 800 symbols as defined by the worker routine currently in process. Each time an END statement is encountered the current Local Dictionary is written out to FORTRAN unit number 8, and the Local Dictionary table is initialized to contain zeroes. The Local Dictionary File as contained on FORTRAN unit number 8 is then used by the second pass to resolve symbols from the intermediate assembly file.

Words 1 and 2 for an entry in the Local Dictionary can contain up to 12 characters from a local symbol. Word 3 contains the symbol type as described on the following page. Word 4 contains the value assigned to this symbol. Word 5 may contain a pointer to subsequent entries in the current Local Dictionary table. For a complete description of the use of the Local Dictionary see the PLSYM, GLSYM, and HASH subroutine descriptions.

LOCAL SYMBOL TABLE TYPES

<u>TYPE</u>	<u>DESCRIPTION</u>
1	statement-label
2	ordinal-file-name
3	local-equate

TABLE 5
LOCAL DICTIONARY

WORD

1

SYMBOL PART I

2

SYMBOL PART II

3

SYMBOL TYPE

4

SYMBOL VALUE

5

CHAIN INDEX

28

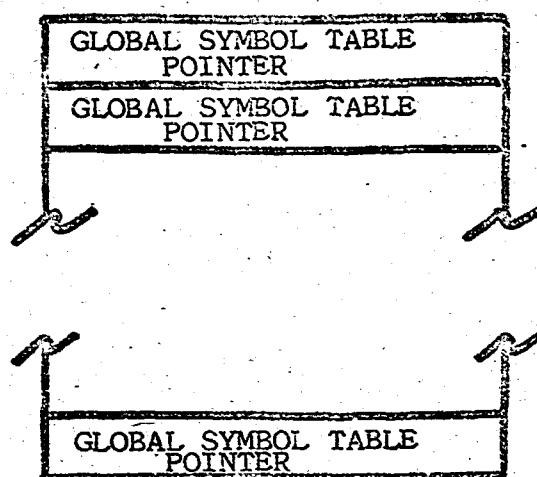
TABLE 6 - GLOBAL HASH TABLE

The Global Hash Table consists of 90 words, all of which are initialized to zero when the assembler is loaded. When a symbol is to be entered into the global symbol table a value from 1 to 90 is calculated by the HASH subroutine from the characters in the symbol to be placed into the table. That address is then used to place a pointer to an address in the global symbol table into the global hash table. The global hash table then contains pointers to entries in the global symbol table.

TABLE 6GLOBAL HASH TABLEWORD

1

2



90

TABLE 7 - LOCAL HASH TABLE

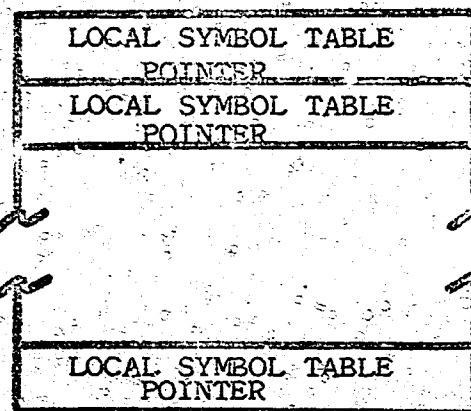
The Local Hash Table consists of 90 words, all of which are initialized to zero when the assembler is loaded. When a symbol is to be entered into the local symbol table a value from 1 to 90 is calculated by the HASH subroutine from the characters in the symbol to be placed into the table. That address is then used to place a pointer to an address in the local symbol table into the local hash table. The hash table then contains pointers to entries in the local symbol table.

TABLE 7LOCAL HASH TABLEWORD

1

2

90



20

TABLE 8 - MACRO PROTOTYPE AREA

Table 8 is used to store the label, operation code, and operands for macro calls during processing by the SMACRO subroutine. Table 8 is filled by copying the current entry from table 3. Therefore, table 8 is an exact duplicate of table 3.

TABLE 8
MACRO PROTOTYPE AREA

WORD	6 bits	6 bits	6 bits	6 bits	6 bits	6 bits
1	NOT USED					
2	NUMBER OF OPERANDS			STATEMENT VARIABLE		
3	LABEL			PART I		
4	LABEL			PART II		
5	LABEL LENGTH	NOT USED				
6	MACRO NAME					
7	NOT USED					
8	MACRO NAME LENGTH	NOT USED				
9	OPERAND-1			PART I		
10	OPERAND-1			PART II		
11	OPERAND-1 LENGTH	NOT USED				
171	OPERAND 55			PART I		
172	OPERAND 55			PART II		
173	OPERAND-55 LENGTH					

TABLE 9 - CURRENT OUTPUT AREA

Table 9 consists of 14 words used to build records for entry into the PCF library. These entries are then placed on the CUR output tape by means of the CROUT subroutine. Table 9 is also used to store 14 word records retrieved from the library by means of the SRCHNV and NXTMT subroutines.

35

O

TABLE 10 - SECOND PASS OUTPUT AREA

Table 10 consists of 9 words which are used to build records by the second pass for output to the simulator. The first word of this table contains the current worker routine number. The second word contains the current worker routine statement number. The third word contains the numeric operation code for this statement. Words 4 through 9 contain the numeric values for operands 1 through 6.

TABLE 10SECOND PASS OUTPUT AREAWORD

1

W/R NUMBER

2

W/R STATEMENT NUMBER

3

OP-CODE

4

OPERAND -1

5

OPERAND -2

6

OPERAND -3

7

OPERAND -4

8

OPERAND -5

9

OPERAND -6

TABLE 11 - SECOND PASS TABLE

Table 11 consists of a nine word entry for each operation code which is capable of being used by the simulator. The operation code value is used as an index to table 11. Therefore, operation code 1 would cause the second pass to use entry 1 in table 11. The nine words in each entry in table 11 are used to control the processing performed by the second pass on each statement before it is written out for use by the simulator. Word 1 may contain either a zero or a number indicating a pre-process is required for the current statement. An example of a pre-process would be sequence checking a JO statement, CF statement, MEM-1 statement, and GEN statement. Word 2 contains a zero or a number indicating that post-processing is necessary for this statement. An example of post-processing would be checking to insure that a SELECT ATQUEUEL statement referenced a queue which contained ATs.

Word 3 contains the instruction length for the current statement as used by the simulator. This value is used to maintain a current location counter which is printed with each statement if the PRINT option was specified in the ASSEMBLY statement.

Words 4 through 9 may contain a zero, a positive value, or a negative value. These words control the processing for operands 1 through 6 of the current statement. If the operand process number is zero, the operand must be omitted and any operand present with that number will be flagged as an error. If the process number for an operand is positive the operand must be included and the number indicates the process which will be used to convert the operand into an output value. If a process number for an operand is negative, the operand is optional. If the operand is present, the number indicates the process which is to be used to convert the operand to an output value. However, if the operand is missing no error is indicated.

TABLE 11SECOND PASS TABLEWORD

1

PRE-PROCESS NUMBER

2

POST-PROCESS NUMBER

3

S3 INSTRUCTION LENGTH

4

OPERAND-1 PROCESS NUMBER

5

OPERAND-2 PROCESS NUMBER

6

OPERAND-3 PROCESS NUMBER

7

OPERAND-4 PROCESS NUMBER

8

OPERAND-5 PROCESS NUMBER

9

OPERAND-5 PROCESS NUMBER

43

TABLE 12 - TITLE AREA

Table 12 consists of 14 words which are used to store the page title as specified by the user. Whenever a TITLE statement is encountered the following card is read into table 12. The contents of this table are printed at the top of each page by the PAGE subroutine.

TABLE 13 - CPU DEFINITION TABLE

Table 13 contains a maximum of five 44 word entries, each of which completely defines a CPU. This table is filled as specified by the CPU-DEF statement. If either CAT or NCAT are specified, this table is filled by reading from the current input stream. If LIB is specified this table is filled by obtaining a CPU definition from the library. Each entry in this table is completely described by the following table layout.

TABLE 13

CPU-DEF TABLE

WORD

1	CPU-ID	PART I
2	CPU-ID	PART II
3	CARD CODE	
4	CPU-ID (CARD)	
5	LOGICAL DATA UNIT	
6	DEC-DIG	
7	DEC. ADD TIME	
8	DEC. MULT TIME	
9	DEC. DIV TIME	
10	FIXED ADD TIME	
11	FIXED MULT TIME	
12	FIXED DIV TIME	
13	FLOAT ADD TIME	
14	FLOAT MULT TIME	
15	FLOAT DIV TIME	
16	CARD CODE	
17	CPU-ID	
18	INST LENGTH	
19	DEC-DIG/LDU	
20	CHAR/LDU	
21	MOVE TIME	
22	MOVE-E TIME	

WORD

23	E1-A FIXED
24	E1-B POINT
25	E2-A TABLE
26	E2-B
27	E3-A
28	E3-B
29	E4-A
30	E4-B
31	E1-A FLOATING
32	E1-B POINT
33	E2-A TABLE
34	E2-B
35	E3-A
36	E3-B
37	NOT USED
38	NOT USED
39	NOT USED
40	NOT USED
41	NOT USED
42	NOT USED
43	NOT USED
44	NOT USED

TABLE 14 - MEM-DEF TABLE

The Memory Definition Table may contain up to a maximum of 30 nine word memory definitions. Entries in the memory definition table are filled as specified by the MEM-DEF statement. If CAT or NCAT is specified the memory definition table is filled from the current input stream. If LIB is specified the memory definition table entry is filled on the library. Each word of an entry in the memory definition table is described in the following table layout.

TABLE 14

MEM-DEF TABLE

WORD

1
2
3
4
5
6
7
8
9

MEM-ID	PART I
MEM-ID	PART II
CARD CODE	
MEM-ID (CARD)	
MEM ACCESS UNIT	
MEM CYCLE TIME	
MEM ACCESS TIME	
MEM SIZE	
PAGE SIZE	

TABLE 15 - CHAN-DEF TABLE

The Channel Definition Table can contain up to a maximum of fifty 9 word channel definitions. Each entry in this table is filled as specified by a CHAN-DEF statement. If CAT or NCAT is specified the current entry is filled from the current input source. If LIB is specified, the current entry is filled from the library. Each word of a Channel Definition entry is described in the following table layout.

TABLE 15

CHAN-DEF TABLE

WORD

1

CHAN-ID PART I

2

CHAN-ID PART II

3

CARD CODE

4

CHAN-ID (CARD)

5

TYPE

6

SEL OR BURST RATE

7

% CPU INTERFERENCE

8

MPX RATE

9

% CPU INTERFERENCE

TABLE 16 - DEVICE-DEF TABLE

The Device Definition Table is capable of containing up to fifty 13 word device definitions. Each entry in the Device Definition Table is filled as specified by the DEV-DEF statement. If CAT or NCAT is specified, the current entry in the table is filled from the input source. If LIB is specified the current entry in the Device Definition Table is filled from the library. Each word in a device definition table entry is described by the following table layout.

TABLE 16DEVICE-DEF TABLEWORD

1	DEV-ID	PART I
2	DEV-ID	PART II
3	CARD CODE	
4	DEV-ID (CARD)	
5	TYPE	
6	DEV-RATE	
7	TRANSFER WIDTH	
8	START TIME	
9	STOP TIME	
10	REWIND TIME	
11	PRE-PENALTY TIME	
12	PENALTY TIME	
13	FORM TIME	

TABLE 17 - CPU CONFIGURATION TABLE

Table 17 can contain up to five entries, each of which describes a CPU for the current simulation. Each entry in table 17 is filled as specified by the CPU statement. Words 1 and 2 contain up to 12 characters of the CPU-NAME. Word 3 contains a pointer to the CPU definition which describes this CPU.

TABLE 17CPU CONFIGURATION TABLEWORD

1

CPU-NAME PART I

2

CPU-NAME PART II

3

CPU-DEF-INDEX

TABLE 18 - MEMORY CONFIGURATION TABLE

Table 18 can contain the configuration data for up to 30 memories. Each entry in this table is filled from the data supplied with a MEM statement. The first two words contain up to 12 characters of the MEM-NAME. Words 3 through 7 can contain CPU numbers which are used to define the CPUs to which this memory is attached. Word 8 contains the index of the entry in the memory definition table which describes the physical characteristics of this memory. The organization of a single entry in the memory configuration table is described by the following table layout.

TABLE 18MEMORY CONFIGURATION TABLEWORD

1

MEM-NAME PART I

2

MEM-NAME PART II

3

CPU-# INDEX

4

"

5

"

6

"

7

"

8

MEM-DEF INDEX

TABLE 19 - CHANNEL CONFIGURATION TABLE

The Channel Configuration Table can contain the configuration data for up to 50 channels. The information for each entry in this table is extracted from the CHANNEL statement. The first two words of an entry contain up to 12 characters of the CHAN-NAME. Words 3 through 7 can contain CPU numbers which are used to indicate the CPUs to which this channel is attached. Word 8 contains the index of an entry in the channel definition table which describes the physical characteristics of this channel. A complete description of a channel configuration table entry is given by the following table layout.

TABLE 19CHANNEL CONFIGURATION TABLEWORD

1

CHAN-NAME PART I

2

CHAN-NAME PART II

3

CPU-# INDEX

4

"

5

"

6

"

7

"

8

CHAN-DEF INDEX

TABLE 20 - CONTROL CONFIGURATION TABLE

The Control Configuration Table is capable of holding the configuration data for up to 50 control units. Each entry in this table is filled from the information obtained from a CONTROL statement. The first two words of an entry in this table contain up to 12 characters of a CTL-NAME. The third word contains a code which indicates whether this control unit is capable of handling INPUT, OUTPUT, or INPUT and OUTPUT operations. Words 4 through 12 may contain channel numbers indicating those channels to which this control unit is attached. A detailed description of each table entry for the control configuration table is provided by the following table layout.

TABLE 21 - DEVICE CONFIGURATION TABLE

The Device Configuration Table is capable of containing configuration data for up to 100 devices. The configuration data for each entry in the device configuration table comes from a DEVICE statement. The first two words contain up to 12 characters of the DEV-NAME. Word 3 contains a code which indicates if this device is seizable or not. Words 4 through 12 may contain a control number, indicating those control units to which this device is attached. Word 13 contains the number of the entry in the device definition table which describes the physical characteristics of this device. A complete description of an entry in the device configuration table is supplied in the following table layout.

DEVICE CONFIGURATION TABLE

1
2
3
4
5
6
7
8
9
10
11
12
13

[illegible]

TABLE 22 - TO-FROM TABLE

The To-From Table contains enough room to build a single entry in the to-from table. As each entry is built, it is written out to the simulator input file. Word 1 contains the device number for which this to-from table is being built. Word 2 contains the number of the row currently being described. Word 3 contains the dimensions of this to-from array. Words 4 through 16 contain the 13 possible entries in a to-from table. A complete description of a single entry in the to-from table is given by the following table layout.

TABLE 22TO-FROM TABLEWORD

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

DEV-#
TO ROW
ARRAY DIMENSION
ENTRY 1
ENTRY 2
ENTRY 3
ENTRY 4
ENTRY 5
ENTRY 6
ENTRY 7
ENTRY 8
ENTRY 9
ENTRY 10
ENTRY 11
ENTRY 12
ENTRY 13

TABLE 23 - QUEUE TABLE

The Queue Table can contain up to 30 queue definitions. The data for each entry in the queue table is obtained from the QUEUE statement. The first two words contain up to 12 characters of the QUEUE-NAME. The third word contains the maximum number of entries for the queue. The fourth word contains a code which indicates whether this queue can contain ATs or IOTs. Word 5 contains a code to indicate whether this queue should be processed as a FIFO, LIFO, or PRI type queue. A complete description of a single entry in the queue table is provided by the following table layout.

TABLE 23
QUEUE TABLE

WORD

1
2
3
4
5

Q-NAME	PART I
Q-NAME	PART II
MAX-NO-ENTRIES	
Q-CONTROL	
Q-METHOD	

TABLE 24 - REAL FILE TABLE

Table 24 can contain up to 150 real file definitions. The data for an entry in the real file table is obtained from the RF or RFC statements. Words 1 and 2 contain up to 12 characters of the real file name. Word 3 contains the number of this real file. Word 4 contains the number of the device on which this real file resides. Word 5 contains the relative location of this file on the device. Word 6 contains the number of characters in a single physical record in this file. Word 7 contains the number of logical records in a single physical record of this file. Word 8 contains the total number of physical records in this file. A complete description of a single entry in the real file table is supplied in the following table layout.

TABLE 24
REAL FILE TABLE

WORD

1
2
3
4
5
6
7
8

RF-NAME	PART I
RF-NAME	PART II
RF-#	
DEVICE #	
RELATIVE LOCATION	
BUFFER LENGTH (CHAR)	
RECORDS/BUFFER	
BUFFERS/FILE	

TABLE 25 - INPUT LEFT-JUSTIFIED BY CHARACTER

Table 25 contains 80 characters from the current input record, with one character per word, left justified, and right filled with blanks.

TABLE 26 - LOAD CLASS TABLE

The Load Class Table is capable of containing up to 15 load class entries. The 16th entry is used to temporarily store information for a load class statement in which an error has been found. Each entry in the load class table may contain up to 5 CPU numbers. A complete description of a load class table entry may be found in the following table layout.

TABLE 26LOAD CLASS TABLEWORD

1

CPU-#

2

CPU-#

3

CPU-#

4

CPU-#

5

CPU-#

TABLE 27 - RUN CLASS TABLE

The Run Class Table is capable of containing up to five run class entries. A sixth entry is provided to temporarily store data when a run class description has been found to contain an error. Each entry in the run class table may contain up to 5 CPU numbers. A complete description of a run class table entry may be found in the following table layout.

TABLE 27
RUN CLASS TABLE

WORD

1
2
3
4
5

CPU-#
CPU-#
CPU-#
CPU-#
CPU-#

TABLE 28 - TABLE DUMP CONTROL TABLE

Table 28 contains 25 words, each of which may be set to control the printing of tables contained within the simulator. If a word contains a zero, the corresponding table in the simulator will be printed at the end of each statistical interval. If a word contains a minus 1, the corresponding table in the simulator will not be printed at the end of a statistical interval. A detailed layout for table 28 may be found on the following page.

TABLE 28TABLE DUMP CONTROL TABLEWORD

1	TI 0=ON -1=OFF
2	T2
3	T3
4	T4
5	T5
6	T6
7	T7
8	T8
9	T9
10	T10
11	T11
12	T12
13	T13

WORD

14	T14
15	T15
16	T16
17	T17
18	T18
19	T19
20	T20
21	T21
22	T22
23	T23
24	T24
25	T25

TABLE 29 - STATISTICS CONTROL TABLE

The Statistics Control Table contains 10 words, each of which may be used to control the printing of statistics in the simulator at the end of simulation intervals. If a word contains a zero, the corresponding statistics will be printed at the end of each statistical interval. If a word contains a minus one, the corresponding statistics will not be printed at the end of intervals. A complete description of the Statistics Control Table may be found in the following table layout.

TABLE 29STATISTICS CONTROL TABLEWORD

1
2
3
4
5
6
7
8
9
10

ST1 0=ON -1=OFF
ST2
ST3
ST4
ST5
ST6
ST7
ST8
STAT
STAT1

TABLE 30 - PROGRAM DISTRIBUTION TABLE

The Program Distribution Table can contain program distribution information for up to 100 worker routines. This table is normally initialized so that each program is received by CPU 1 and has a load class number of 1. If a program is to be handled in some way other than normal, the RCV statement must be used in that program. The RCV statement may be used to specify the receiving CPU and the load class which is to be used for that program. Word 1 of each entry contains the program number. The second word contains the number of the receiving CPU for that program. Word 3 contains the number of the load class to be used by the current program. A description of a single entry in the program distribution table is supplied by the following table layout.

TABLE 30
PROGRAM DISTRIBUTION TABLE

WORD

1

PROG #

2

RECEIVING CPU-#

3

LOAD CLASS-#

TABLE 31 - INTERRUPT VECTOR TABLE

The Interrupt Vector Table can contain interrupt vector information for up to 5 CPUs. The first word in an interrupt vector table entry contains the number of the CPU for which this entry applies. The second word contains the number of the operating system to be used by this CPU. Words 3 through 22 contain the statement number for each of the possible 20 interrupts which have been defined.

TABLE 31
INTERRUPT VECTOR TABLE

WORD

1
2
3
4
5
6
7
8
9
10
11

CPU-#
O/S PROC-#
INT-1 STMT-#
INT-2 "
INT-3 "
INT-4 "
INT-5 "
INT-6 "
INT-7 "
INT-8 "
INT-9 "

WORD

12
13
14
15
16
17
18
19
20
21
22

INT-10 STMT-#
INT-11 "
INT-12 "
INT-13 "
INT-14 "
INT-15 "
INT-16 "
INT-17 "
INT-18 "
INT-19 "
INT-20 "

TABLE 32 - OPERATION CODE TABLE

The Operation Code Table is capable of containing up to 128 twelve character operation codes with their associated values. This table is loaded by the SOPCD subroutine before the actual reading of input data is begun. This table is loaded from a file catalogued in the library under the name OPCDS/VERSl. These entries in the library must have been processed by ASX001 to insure that they are in alphabetical order. This table is used by subroutine S1 which performs a binary search looking for the current operation code. Words 1 and 2 contain the operation symbol. Word 3 contains the value of the operation code.

TABLE 32
OPERATION CODE TABLE

WORD

1

OP-SYMBOL PART I

2

OP-SYMBOL PART II

3

VALUE

TABLE 33 - JOB NAME TABLE

The Job Name Table contains the name of each worker routine entered into the system in the sequence in which it appeared in the system. This table is then written out for use by the assembler statistical analysis program. Table 33 can contain up to 100 job names. Words 1 and 2 contain the 12 character job names and word 3 contains the starting ordinal file number for this job. A description of a single entry in the job name table is supplied in the following table layout.

TABLE 33JOB NAME TABLEWORD1
2
3

JOB-NAME	PART I
JOB-NAME	PART II
STARTING	OF-#

TABLE 34X - ORDINAL FILE NAME TABLE

The Ordinal File Name Table contains the names of all ordinal files utilized in the current run. Words 1 and 2 contain the ordinal file name, word 3 contains the number of the real file to which this ordinal file referred, and word 4 contains the number of buffers to be used in processing this ordinal file. A description of a single entry in the ordinal file name table can be seen in the following table layout.

TABLE 34X
ORDINAL FILE NAME TABLE

WORD.

1

OF-NAME	PART I

2

OF-NAME	PART II

3

RF-#

4

NUMBER OF BUFFERS

TABLE 35 - O/S MEMORY ALLOCATION TABLE

The operating system memory allocation table can contain 5 pairs of entries which assign up to 5 operating system programs to the memories in which they are to reside. The first word of a pair contains the operating system program number. The second word of a pair contains the number of the memory in which that operating system is to be placed. The data in the O/S memory allocation table comes from the OS statement. A complete description of the O/S memory allocation table may be seen in the following table layout.

TABLE 35O/S MEMORY ALLOCATION TABLEWORD

1
2
3
4
5
6
7
8
9
10

O/S PROG #
MEM #
O/S PROG #
MEM #
O/S PROG #
MEM #
O/S PROG #
MEM #
O/S PROG #
MEM #

WORKING PAPER

86

SECTION II

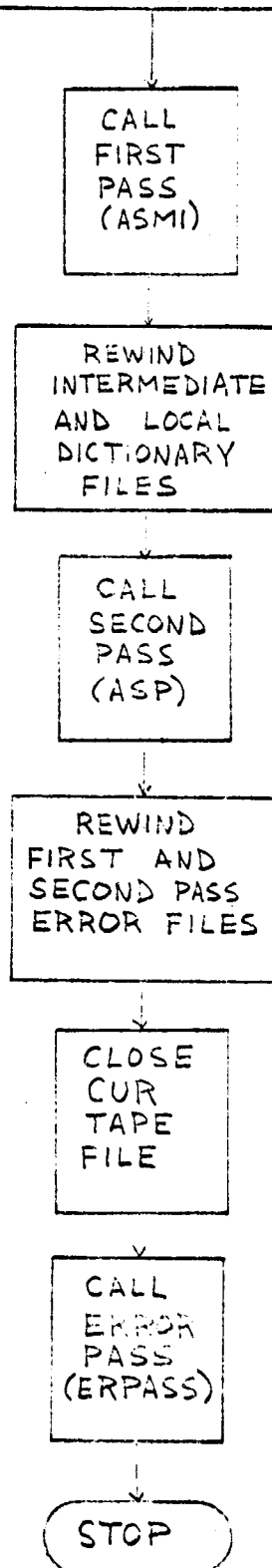
FLOW CHARTS AND NARRATIVE

WORKING PAPER

ASSEMBLER MAIN PROGRAM

The assembler main program calls the first pass subroutine, ASM1, first. It then rewinds the intermediate and local dictionary files. The second pass, ASP, is then called. The first and second pass error files are then rewound. The CUR tape file is then closed. The error pass is then called.

ASSEMBLER
MAIN PROGRAM



ERROR MESSAGE FORMATTING MAIN PROGRAM

This routine tests and prepares error messages for use by the error pass. A control card is read in containing either the word BUILD, PRINT, or PUNCH. If BUILD is present, then the error messages, after testing, are written out onto a tape in CUR format. If PUNCH is present, then the error messages, after testing, are punched out in form suitable for the CUR program. If PRINT is present, the input error message is only used for testing. Each input message is transformed into a CUR format and is also moved to a FORTRAN format area, from which it is used to write a sample error message. Duplicate error messages, by number, are so indicated.

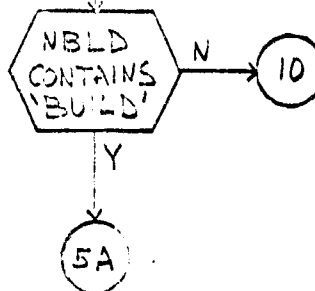
ERROR MESSAGE
FORMATTING PROGRAM

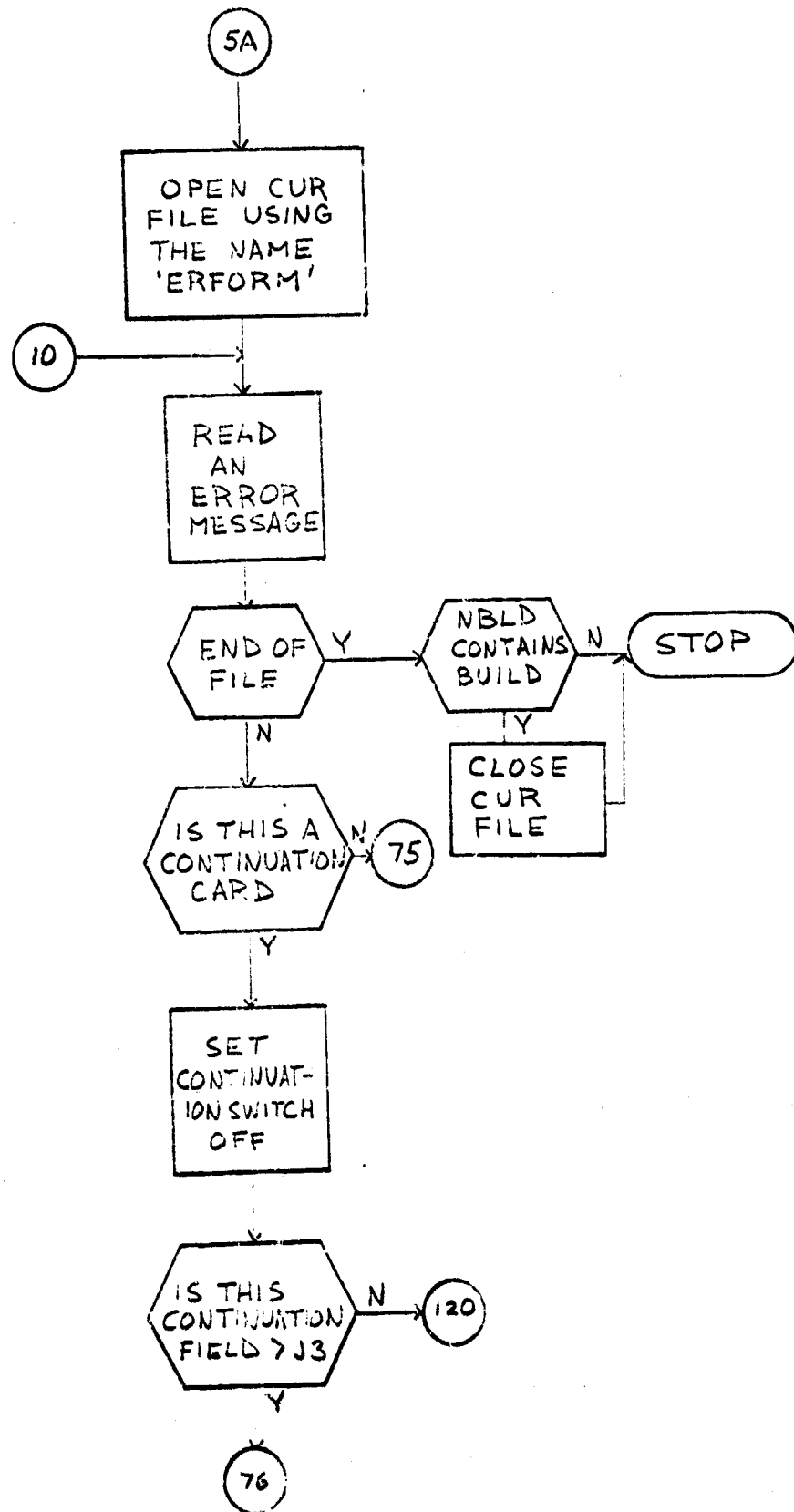
INITIALIZE
OUTPUT
VARIABLES

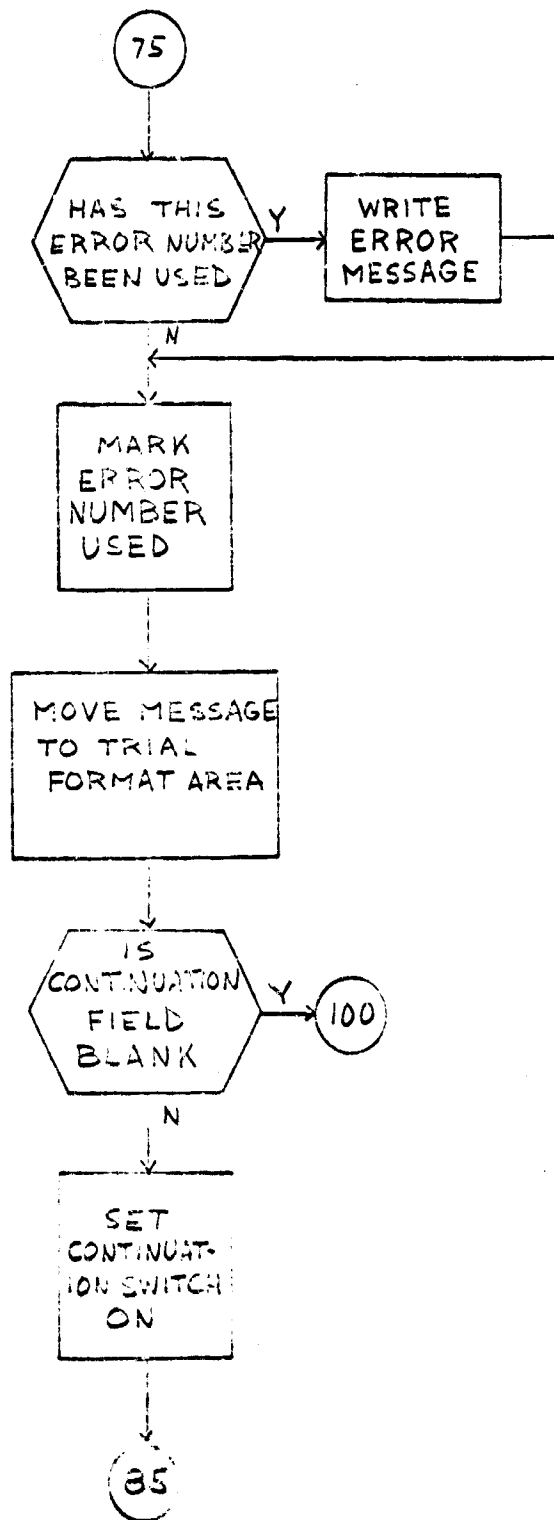
READ CONTROL
INFORMATION
INTO NBLD

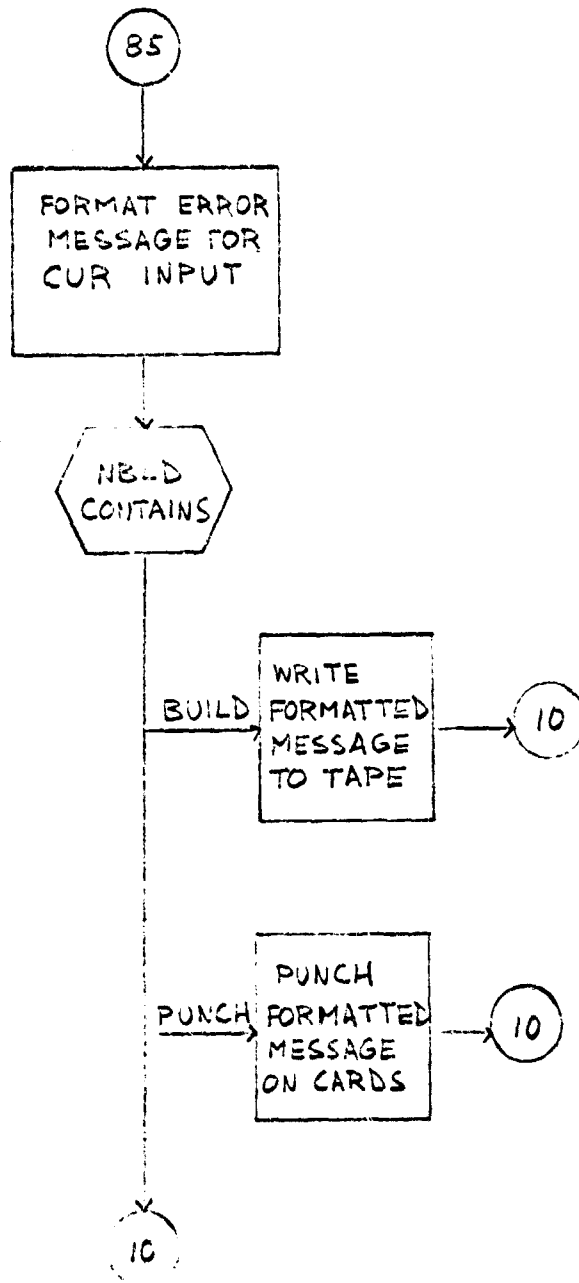
SET
CONTINUATION
SWITCH
OFF

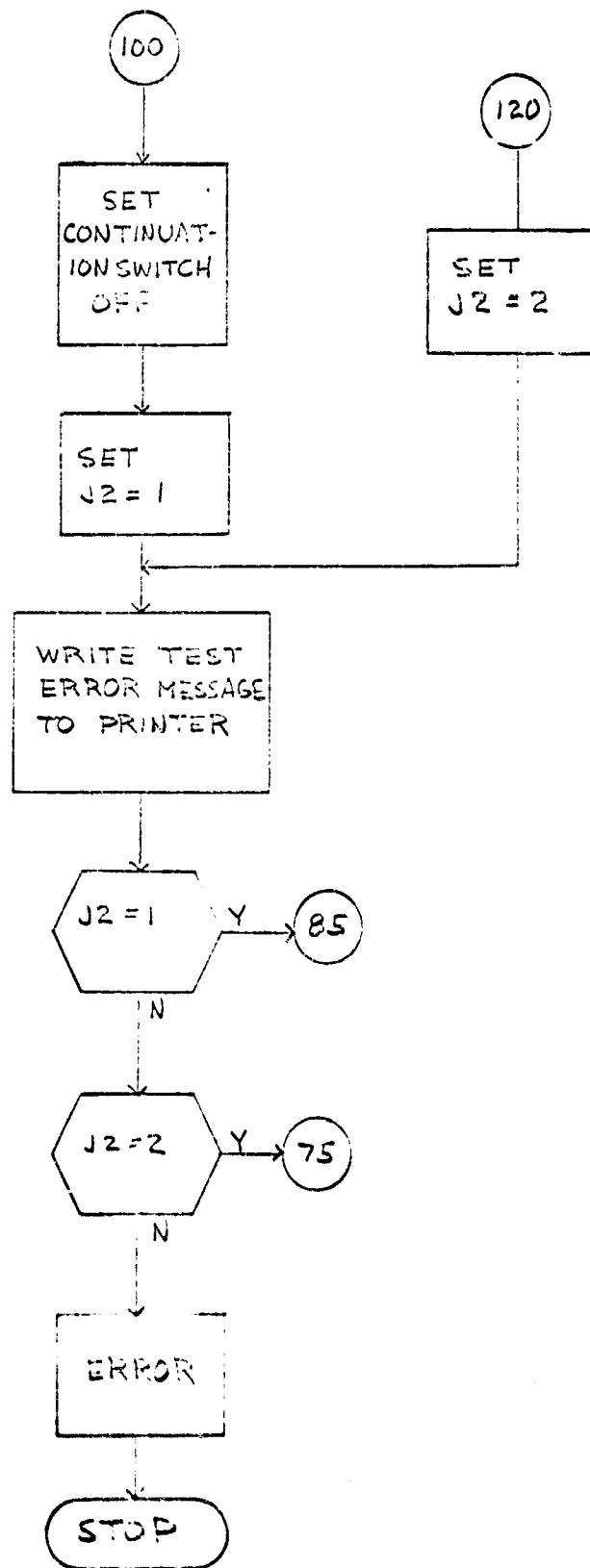
SET J3
TO BLANK





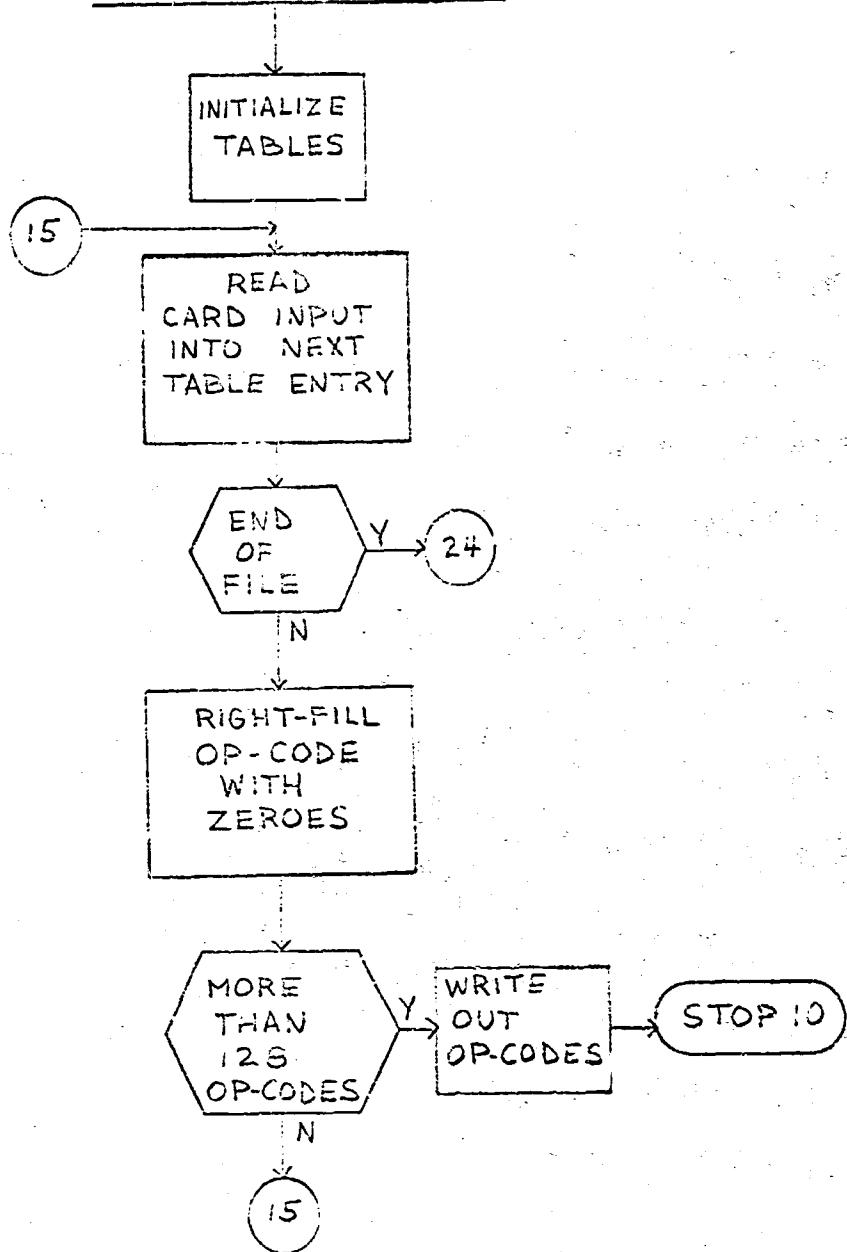


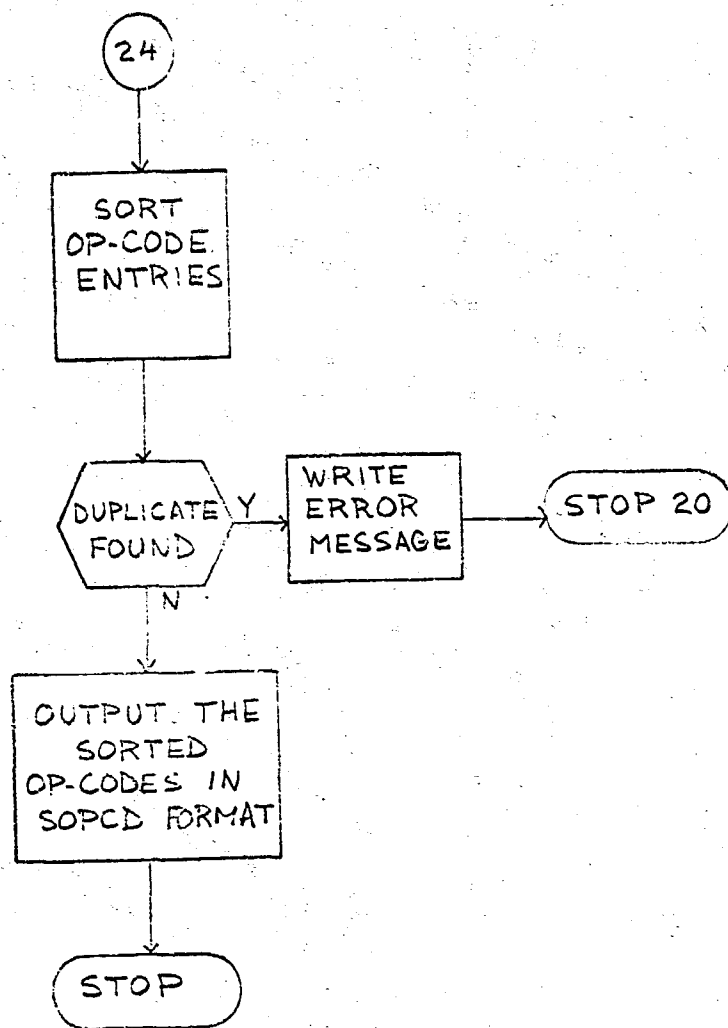




CP-CODE SEQUENCING PROGRAM

This program reads cards containing op-codes and their associated value. The op-codes can be presented in any order. This program sorts the op-codes and formats them for input to the SOPCD subroutine.

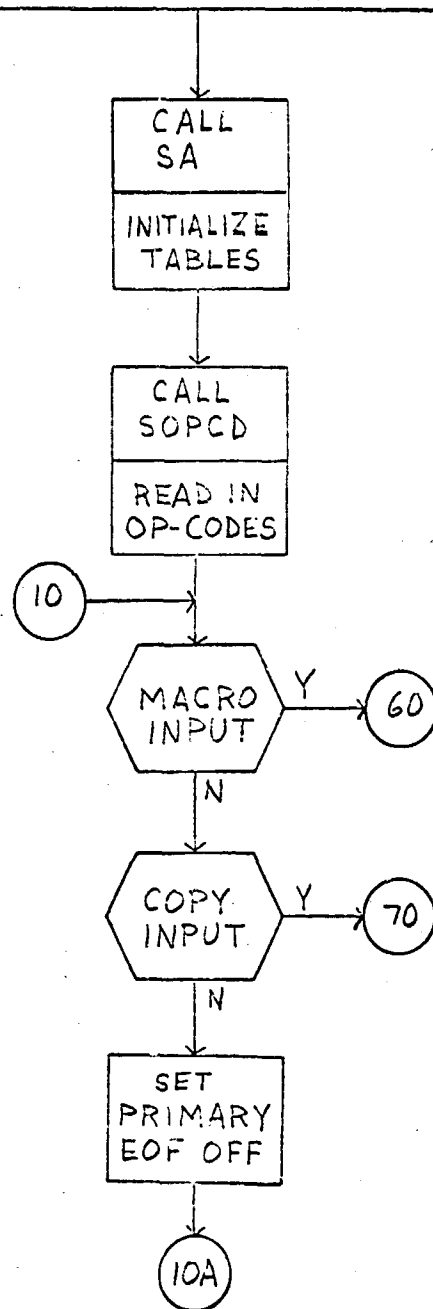
OP-CODE
SEQUENCING PROGRAM



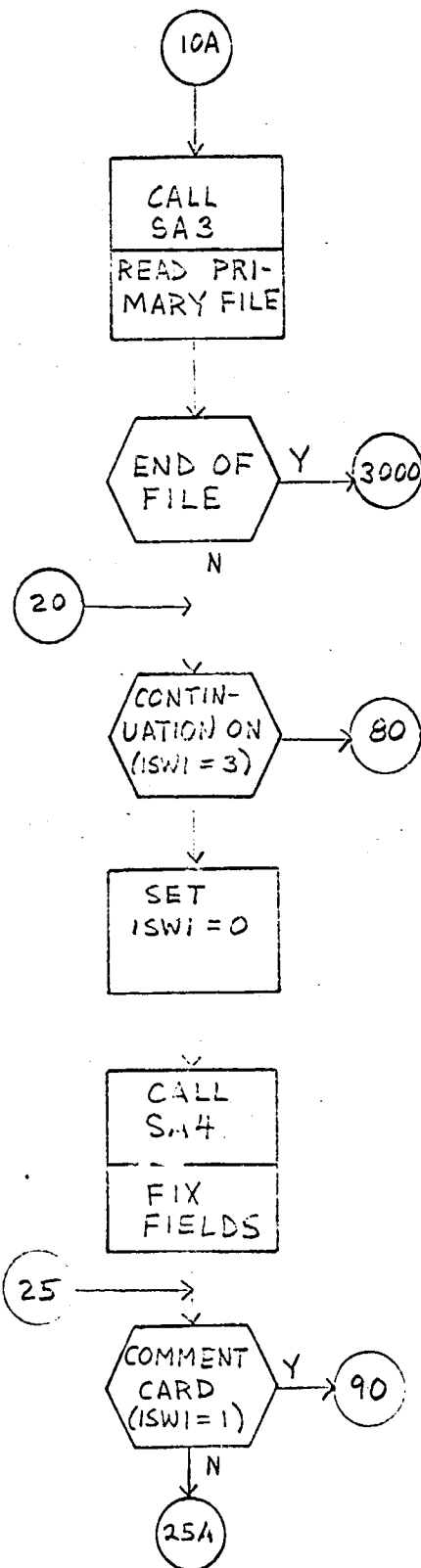
SUBROUTINE ASML

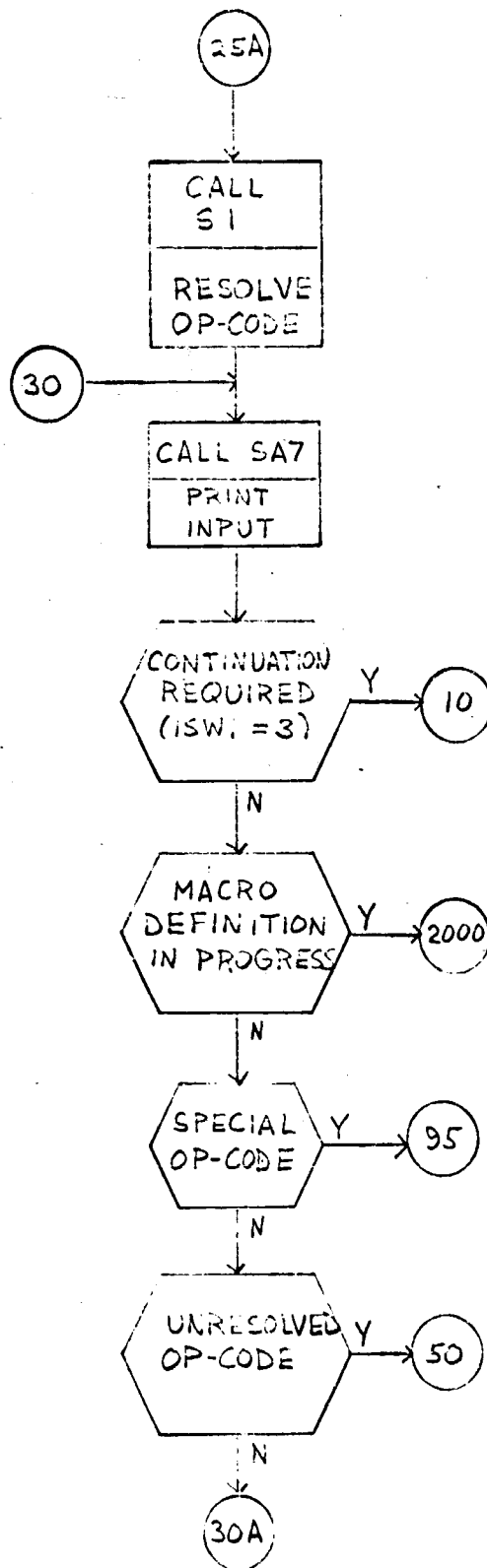
This subroutine makes the first pass through the source statements. It consists of two phases. In the first phase the hardware configuration description statements, and system parameters are read in, processed, and placed in appropriate tables. When all system parameters have been processed, the tables are written out to the simulator input file.

The second phase begins with the receipt of the ASSEMBLY statement. All operating system and worker program representations must follow the ASSEMBLY statement. Furthermore, all macro processing takes place in this phase. In this phase, statement labels and local equates are placed in the local symbol table. A local symbol table is maintained for each job being processed. As each job is ended, this symbol table is written out to a drum storage area. After examining each statement, expanding macro calls as required, the statements are written out to an intermediate file on the drum, from which they will be processed by the second pass.

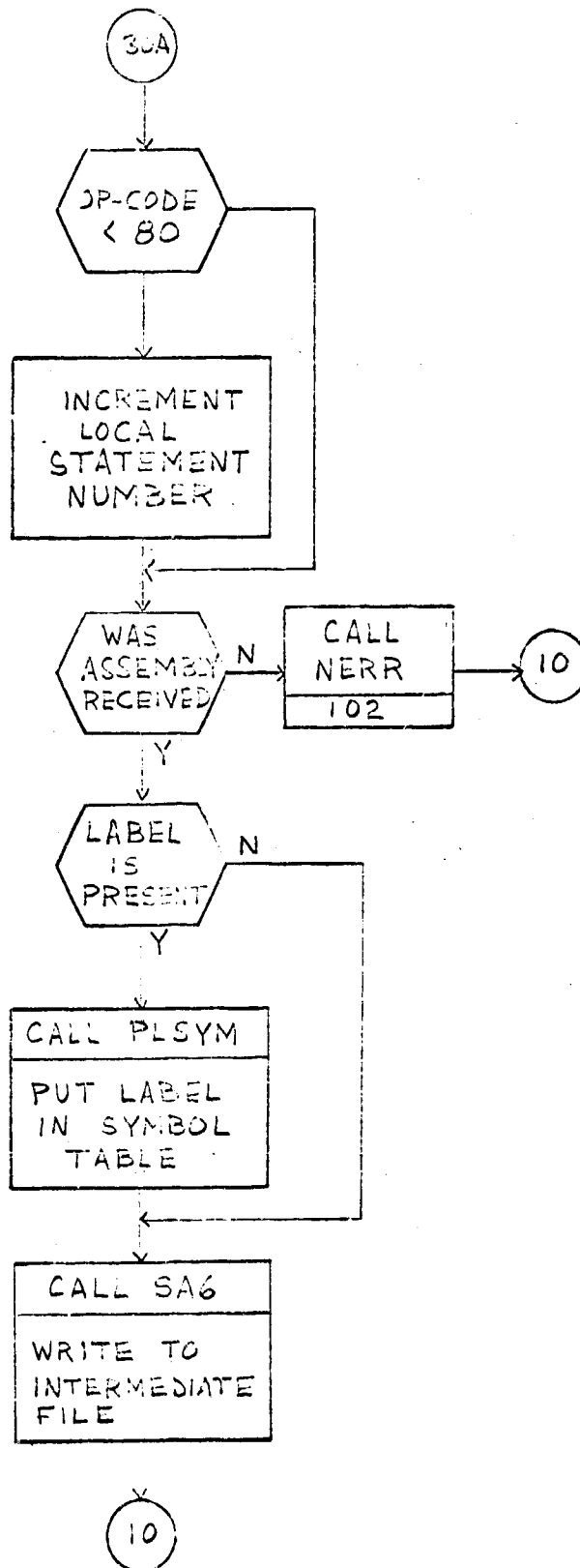
SUBROUTINE ASMI
(FIRST PASS)

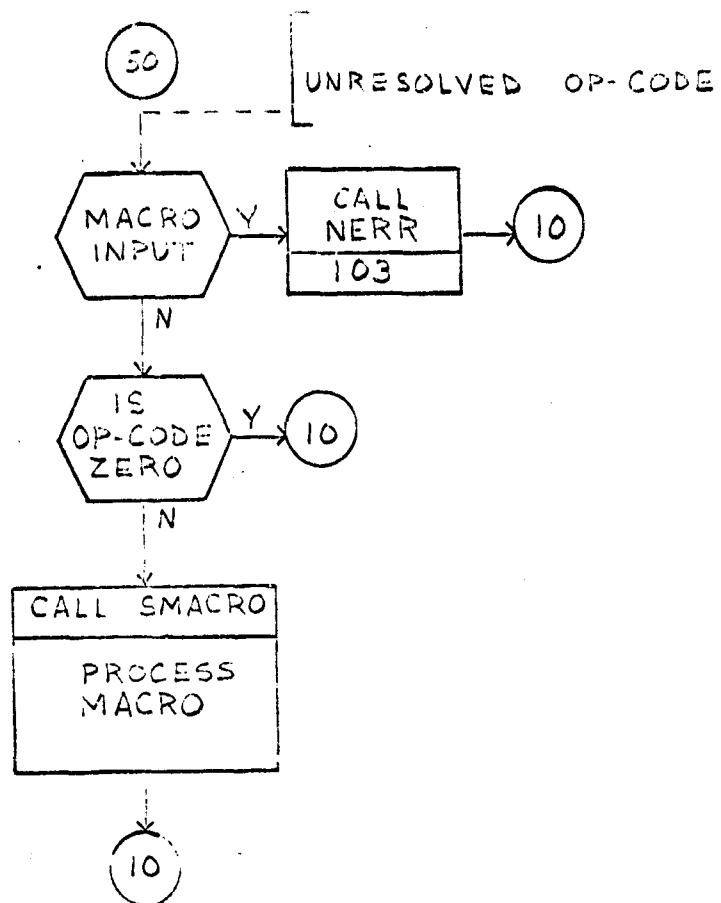
100



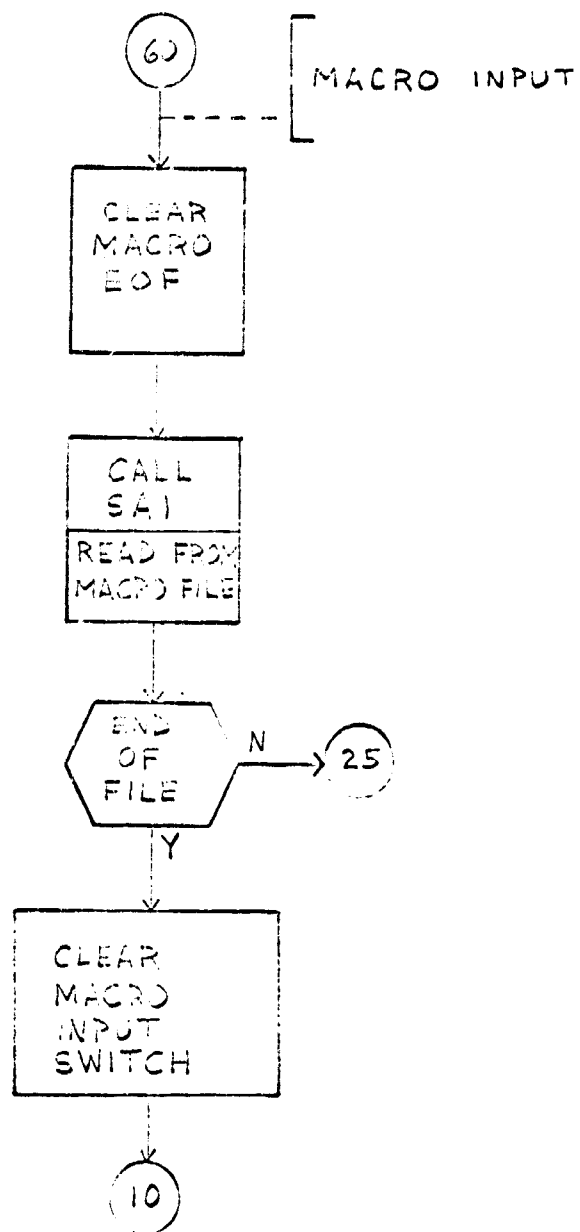


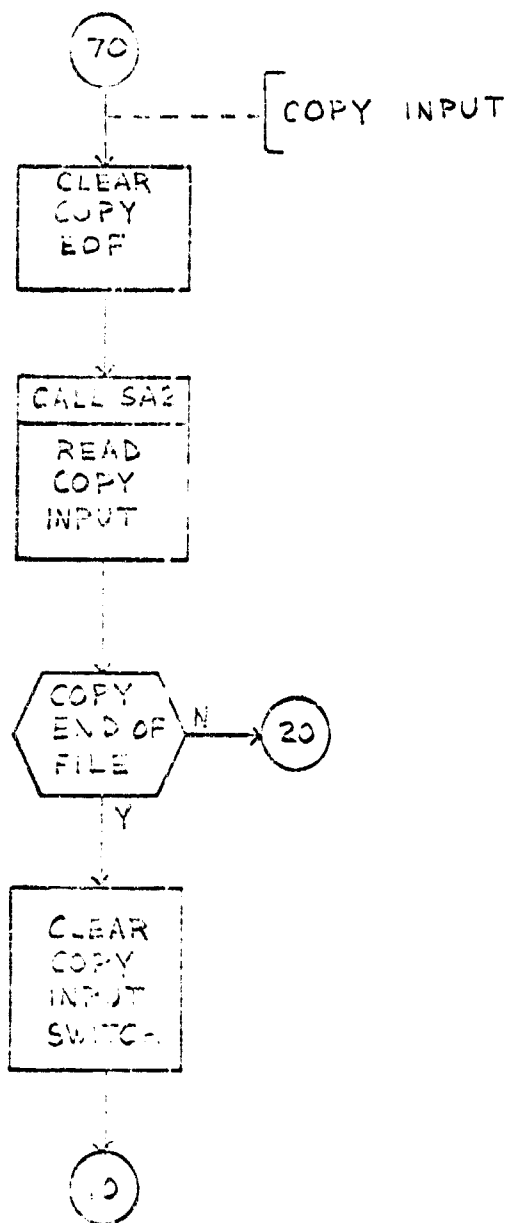
102



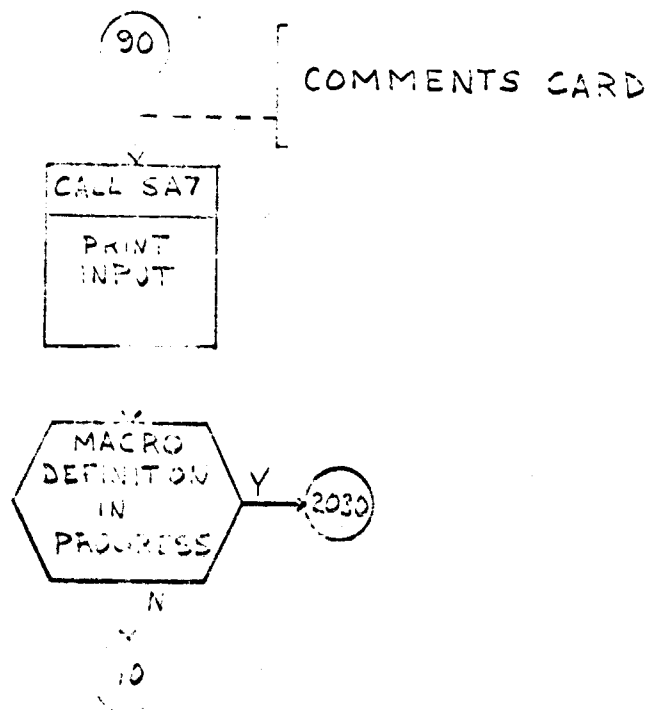
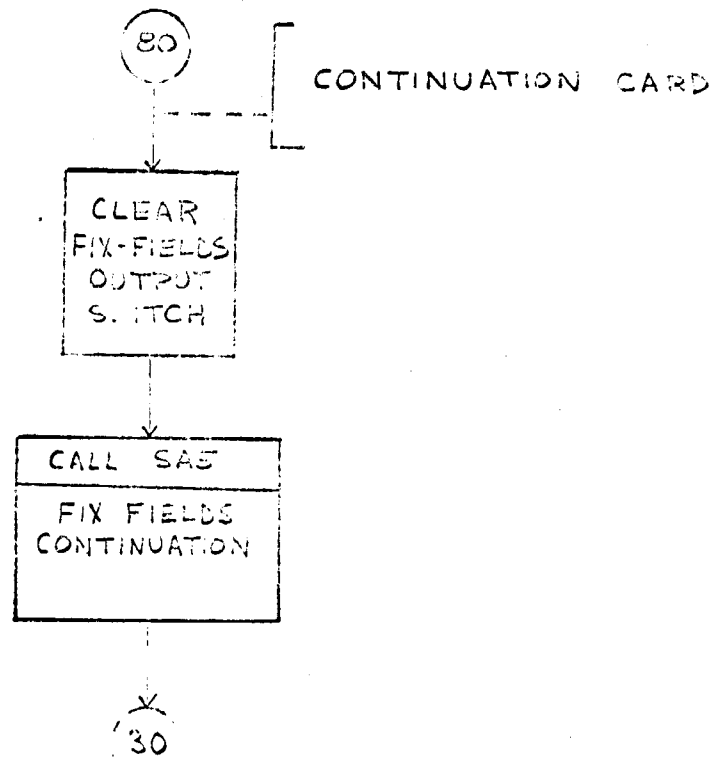


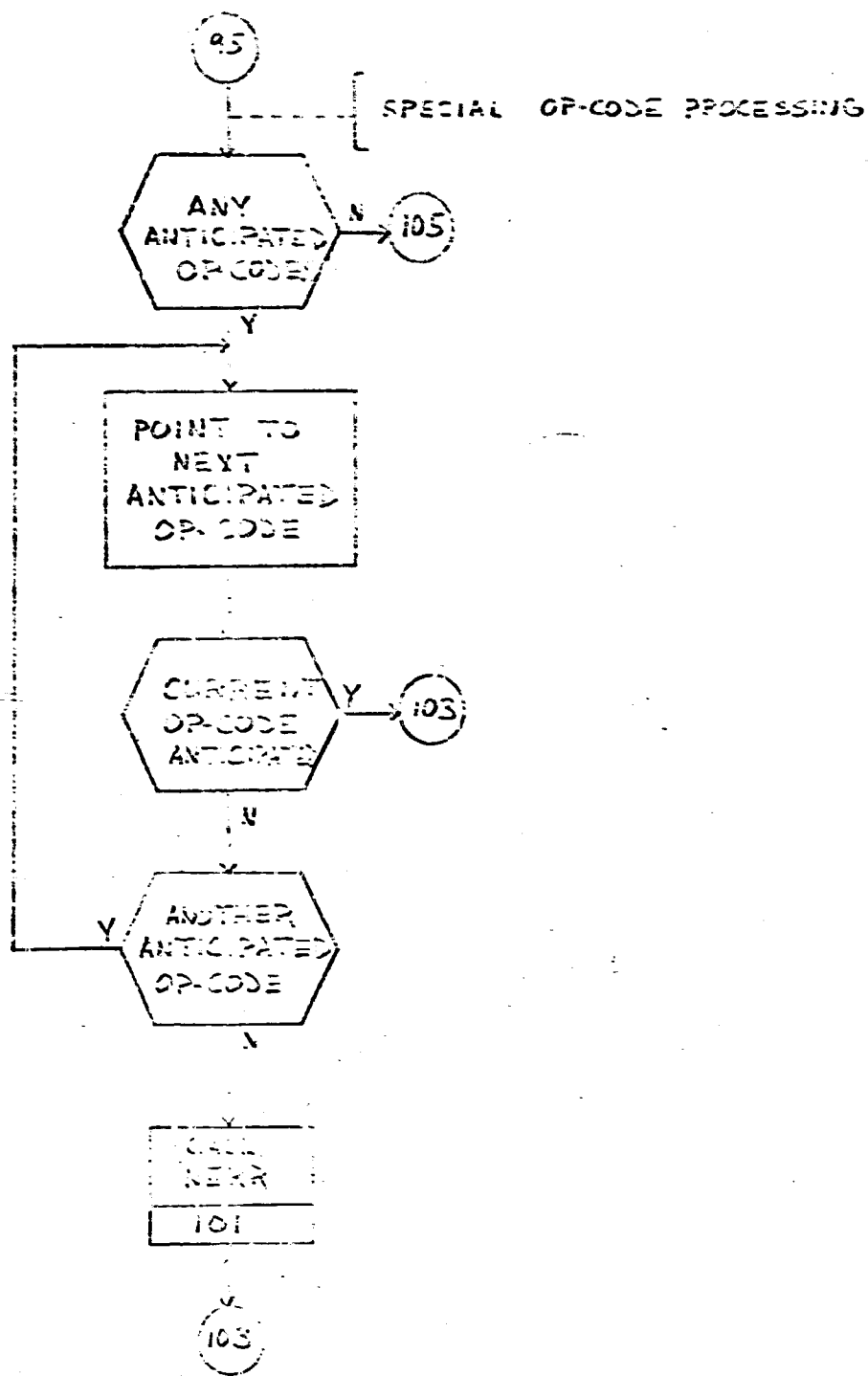
104



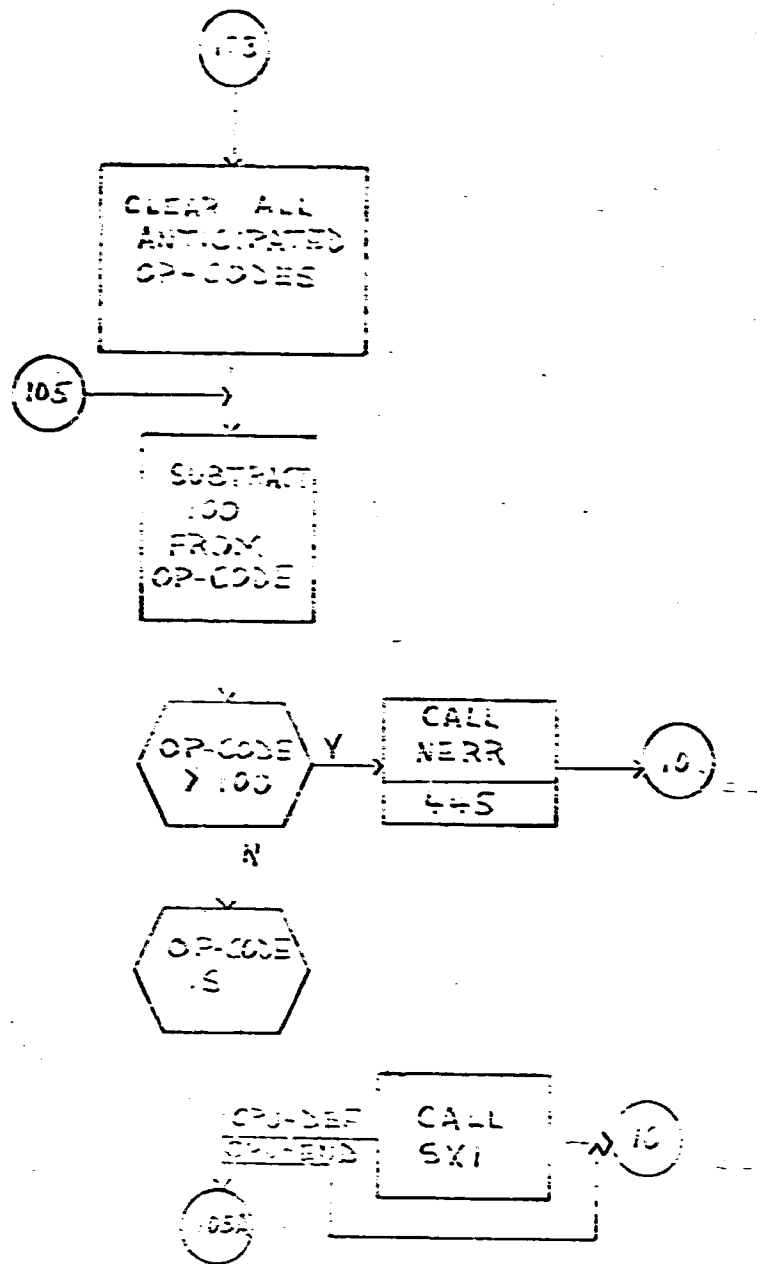


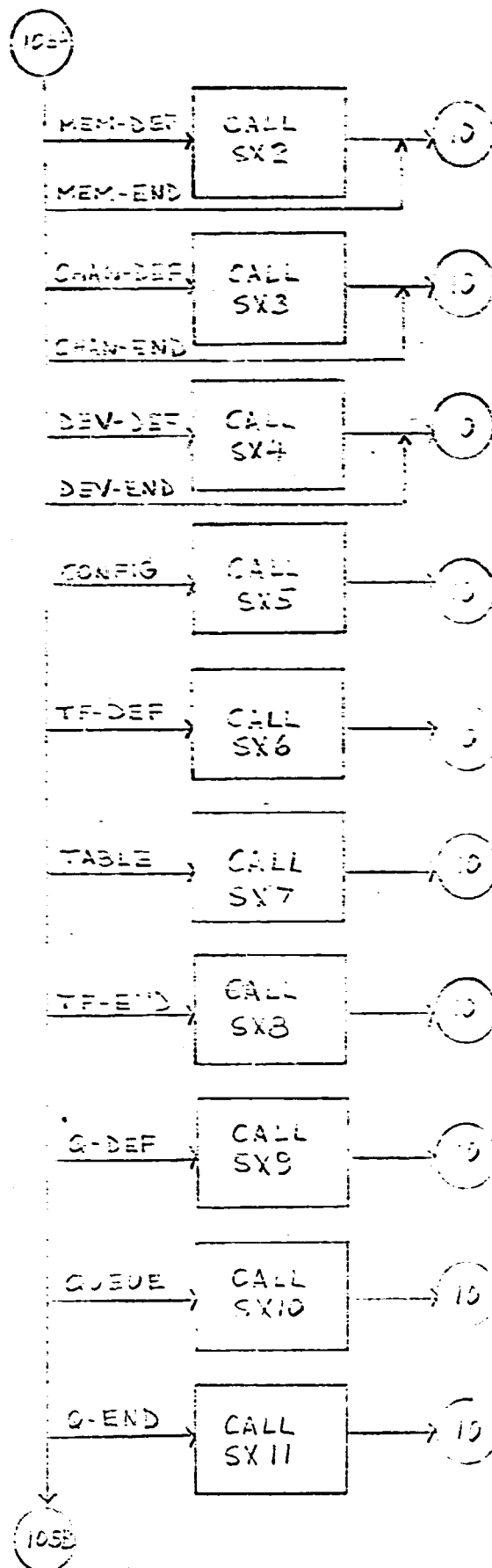
106

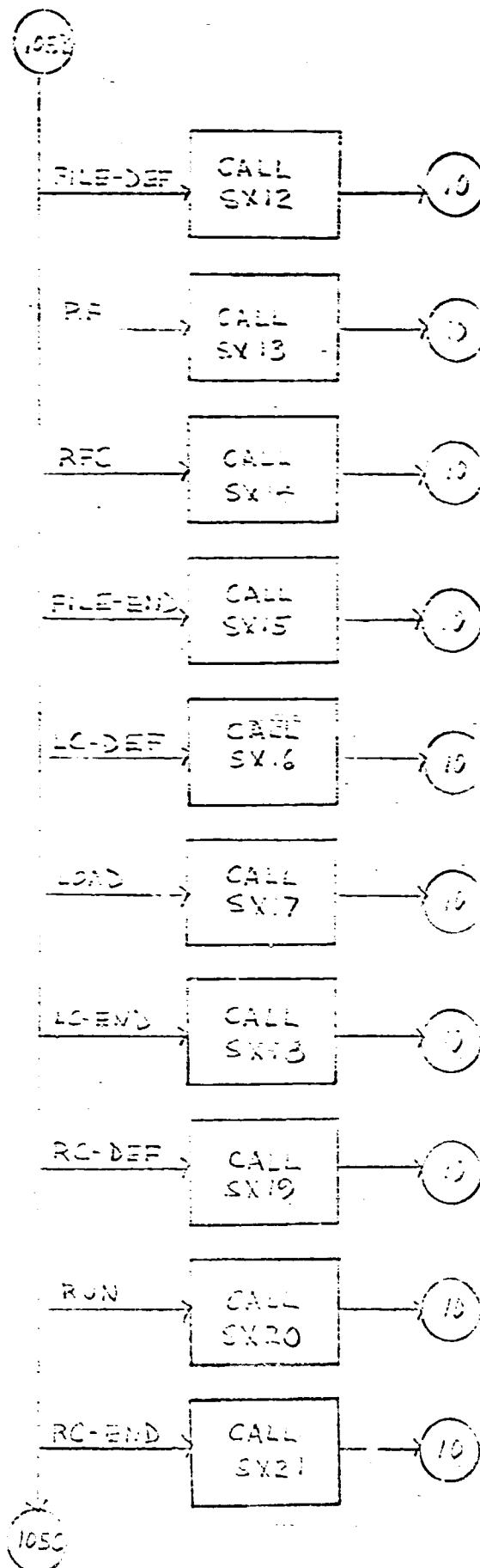


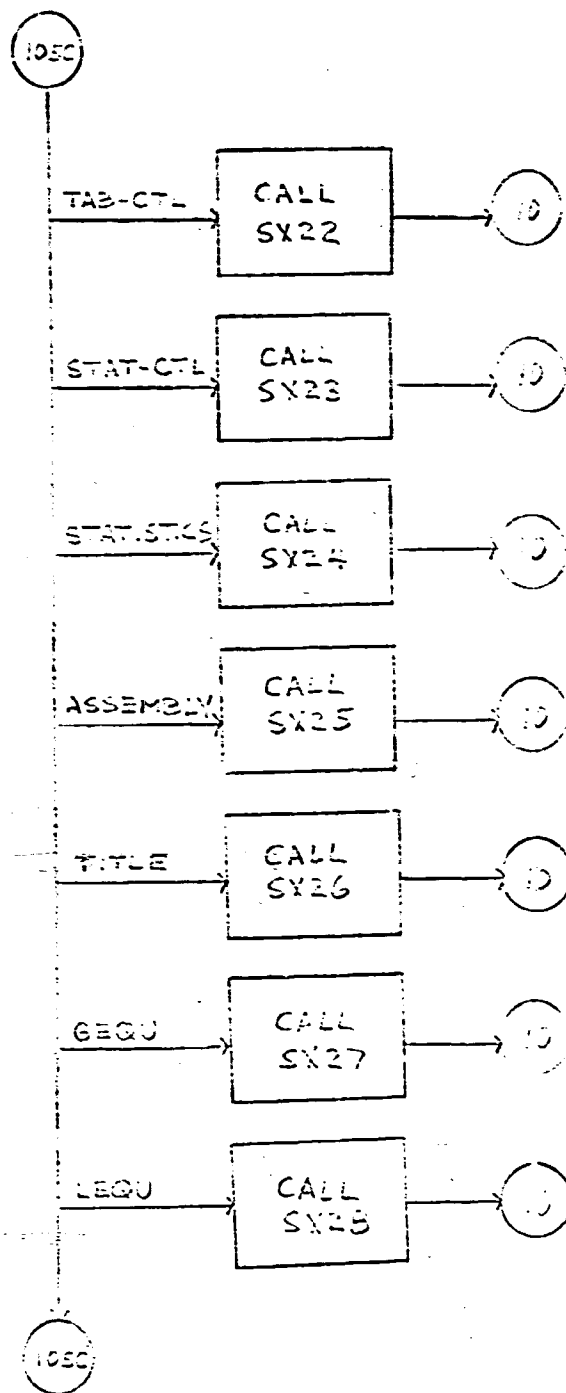


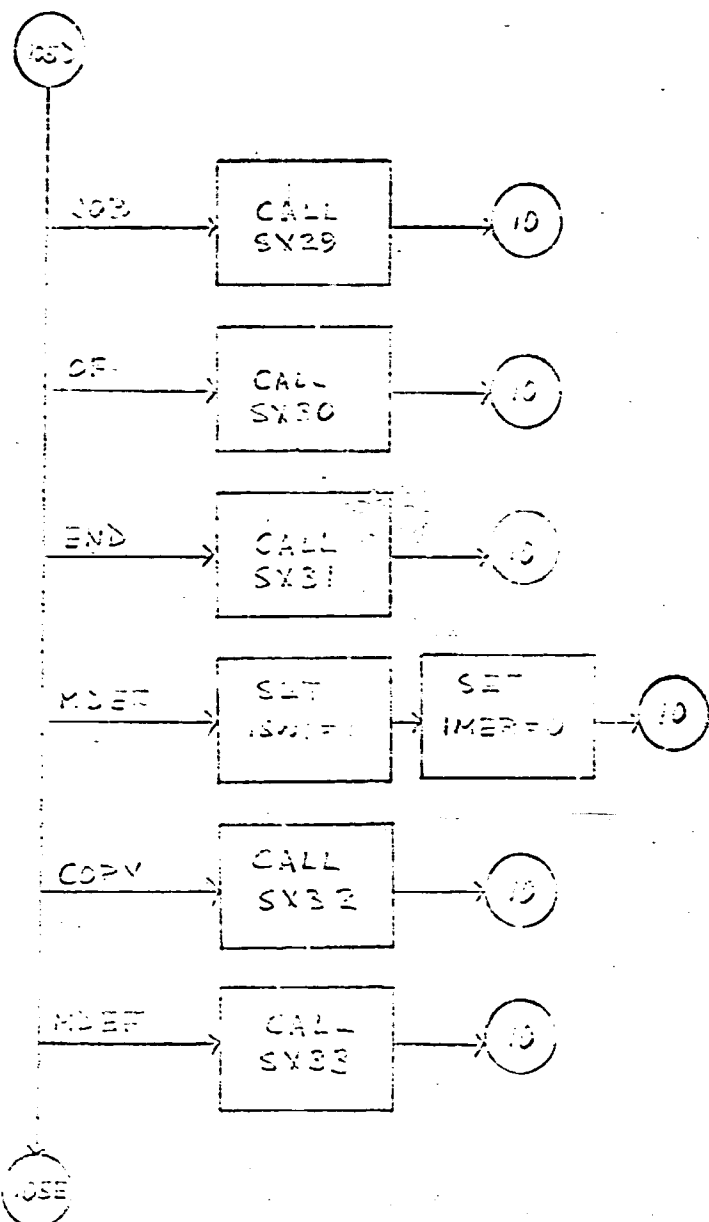
100

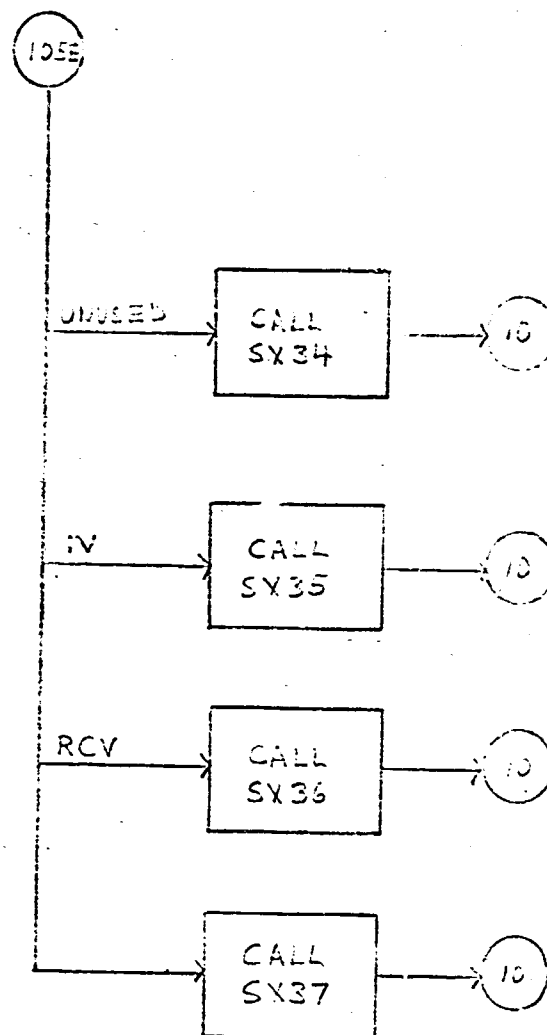


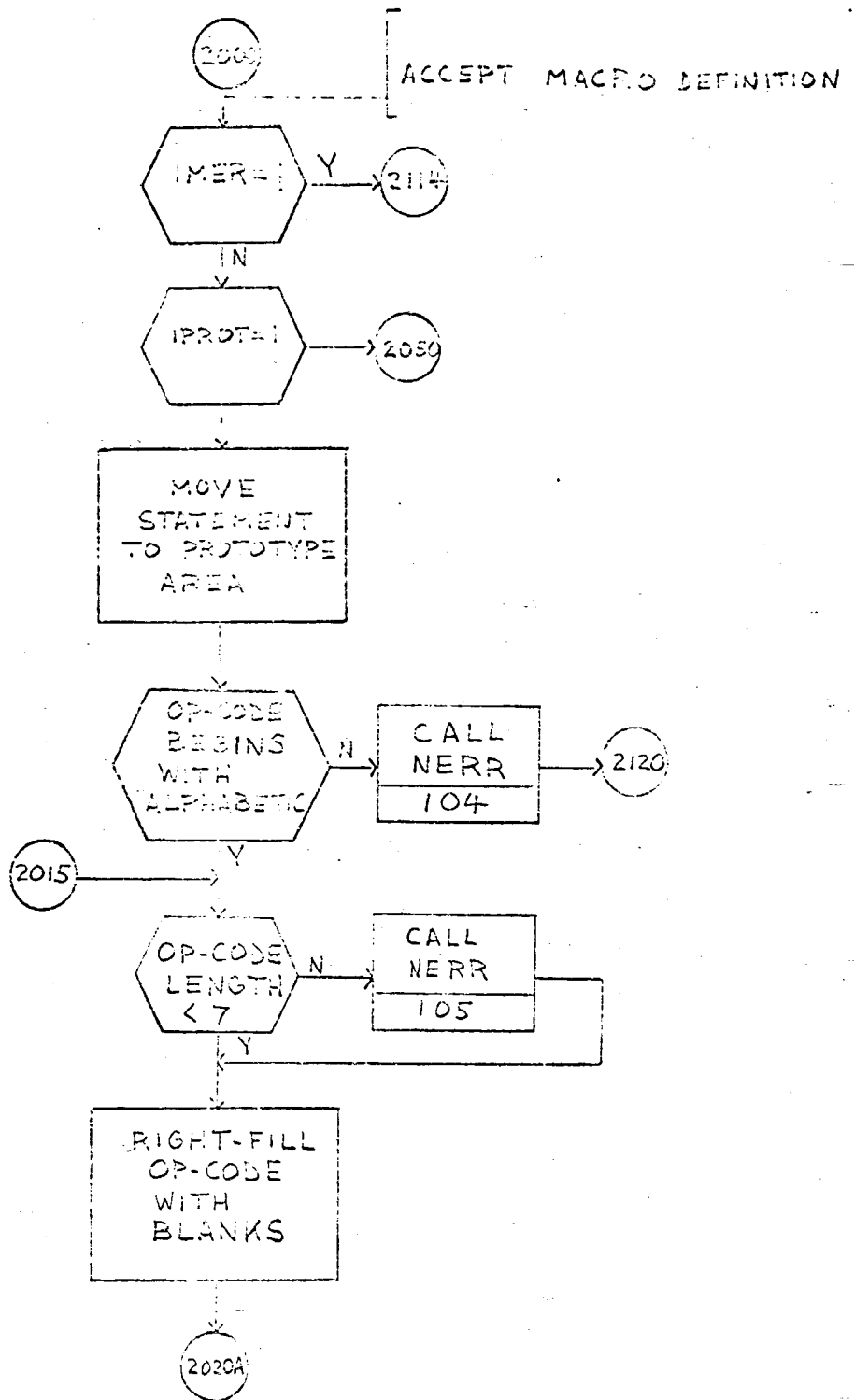


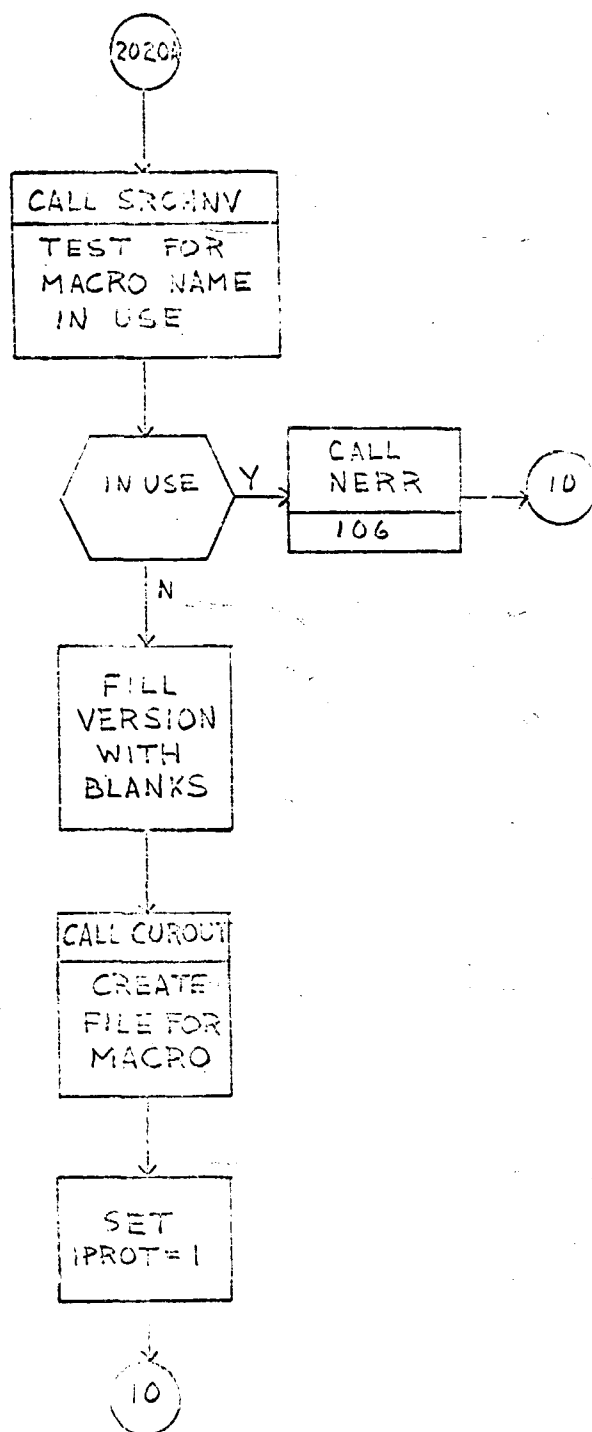




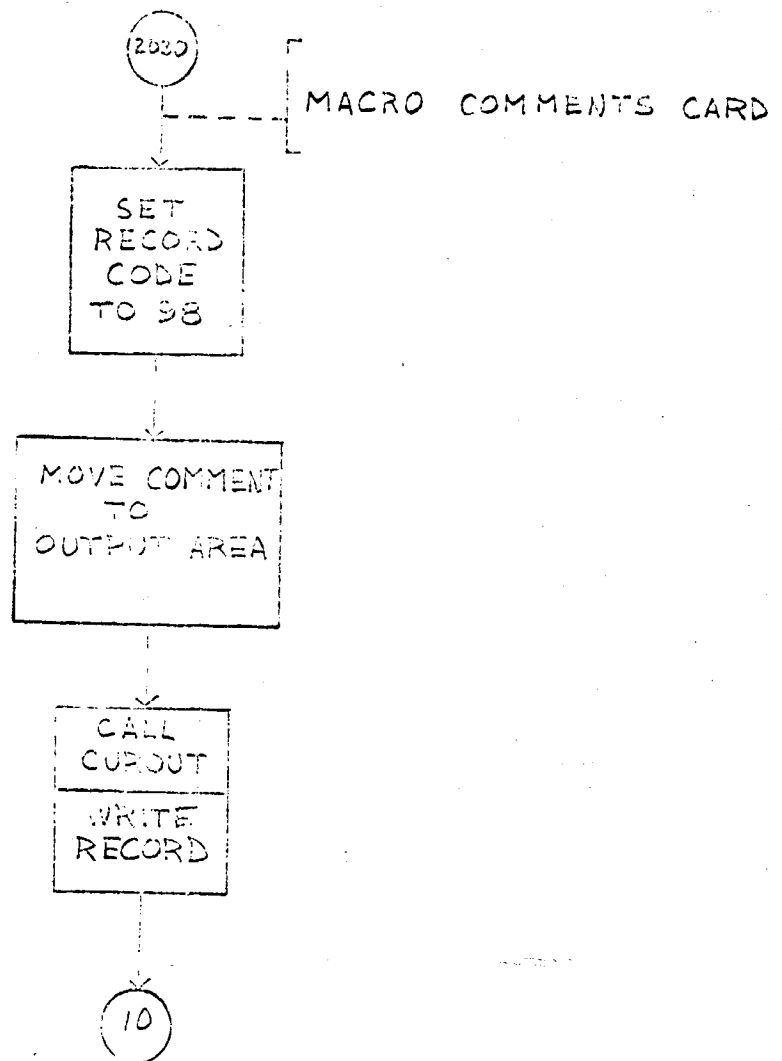


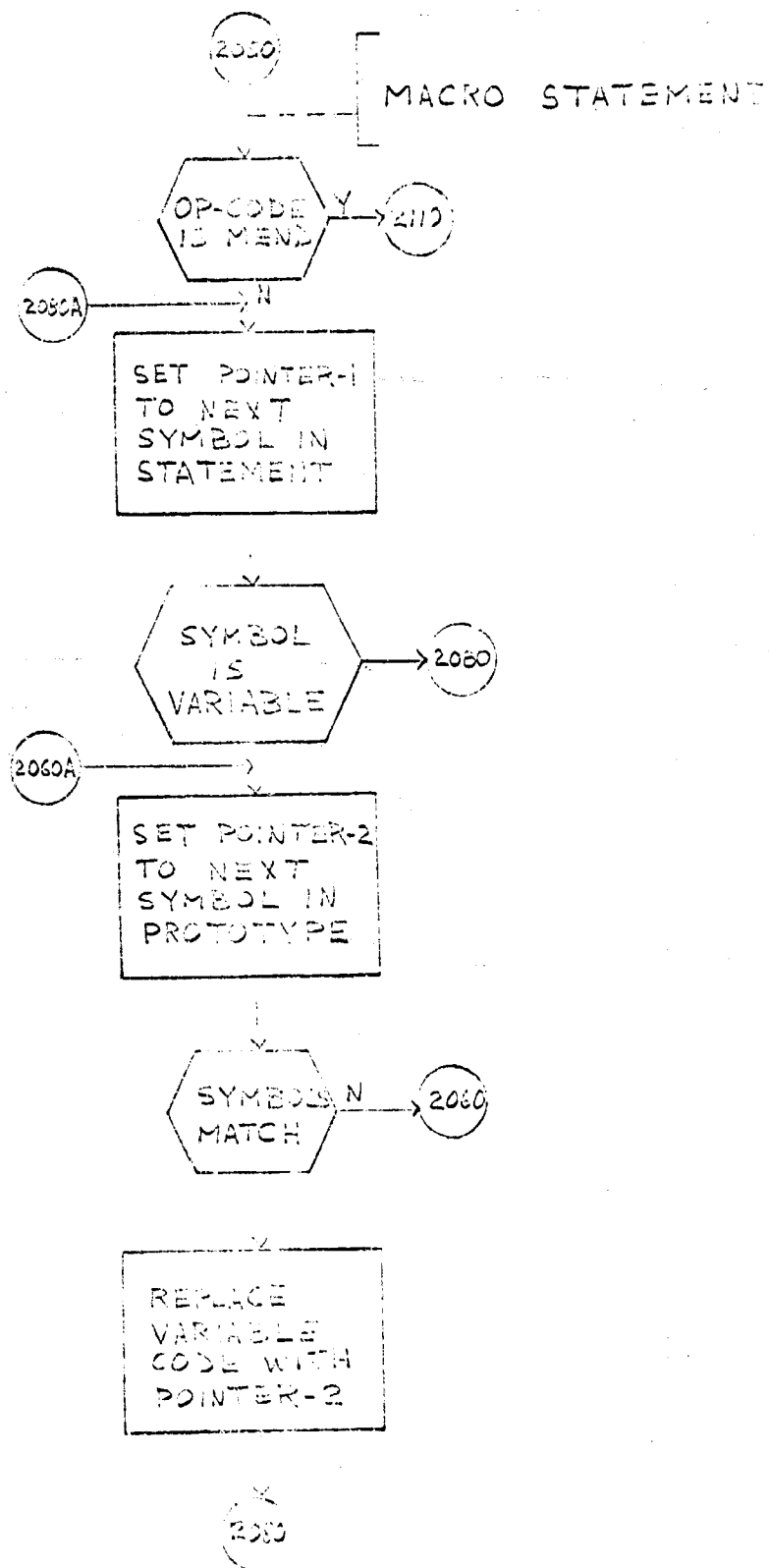




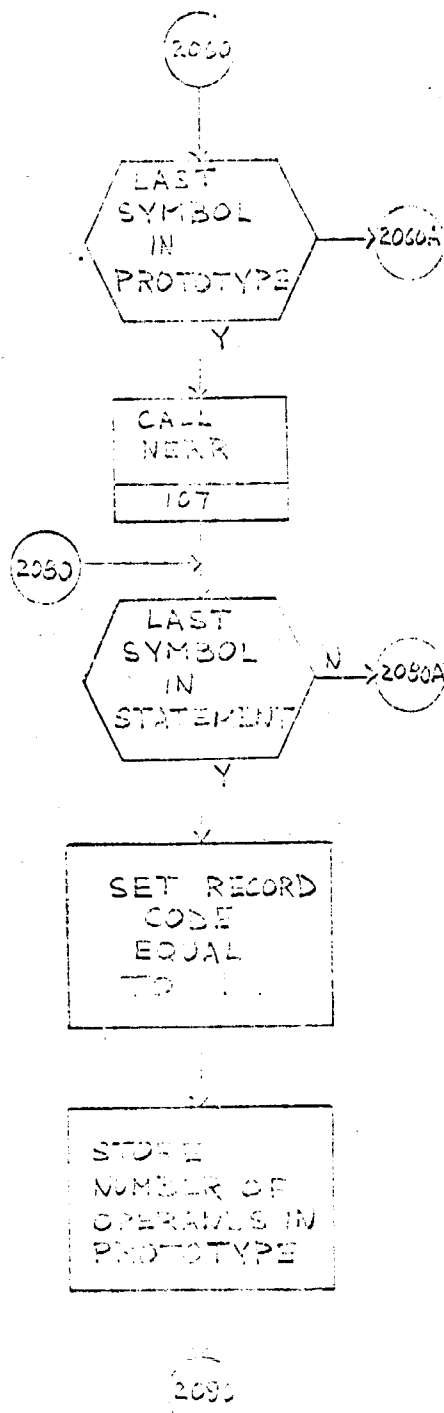


116





642



119

2090

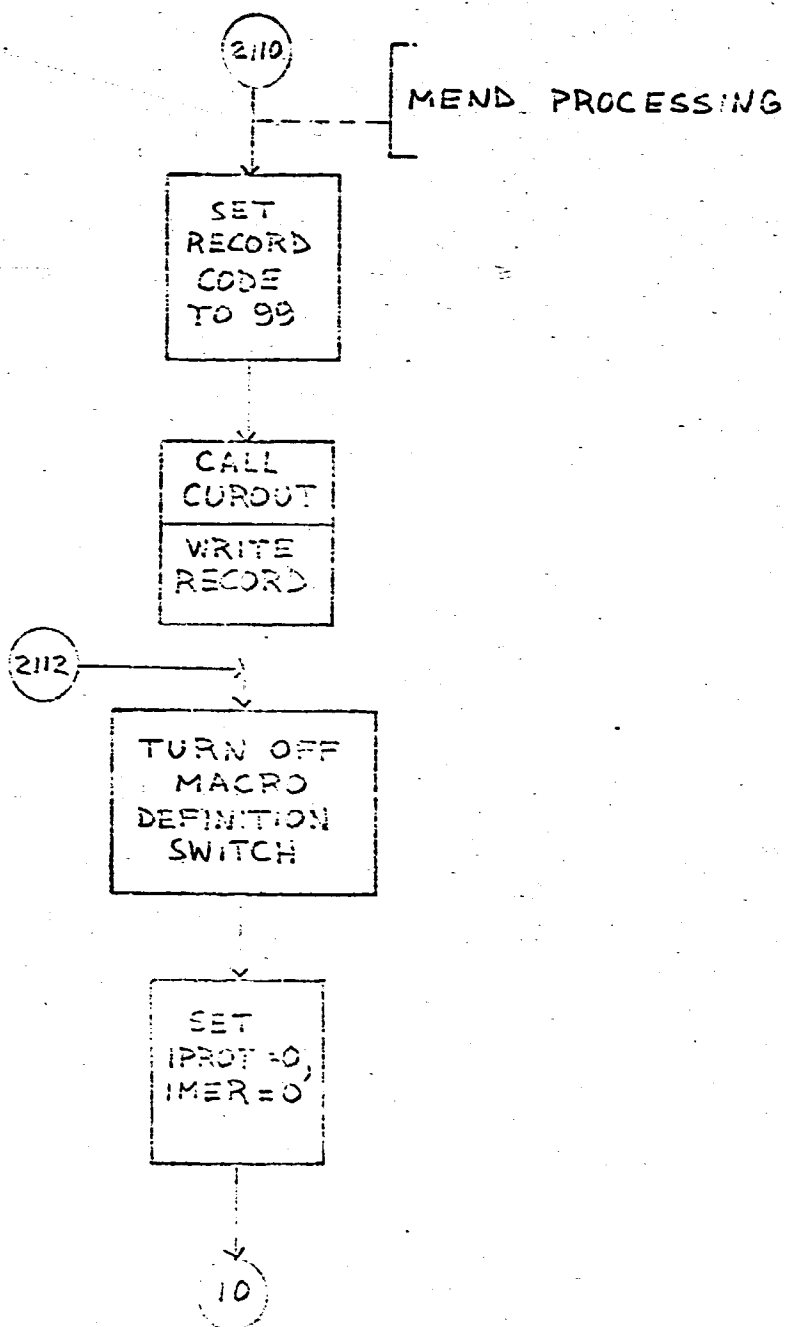
MOVE
STATEMENT
TO OUTPUT
AREA

CALL
CURCUT

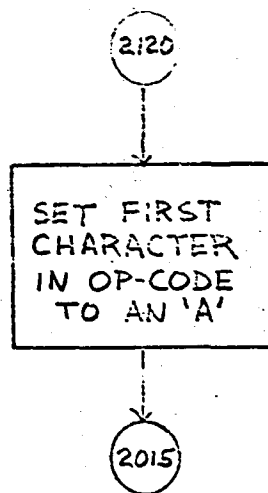
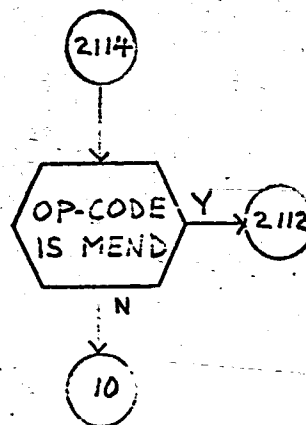
WRITE OUTPUT

10

120



121



122

3000

WRITE END OF
FILE RECORD
TO INTERMED-
IATE FILE

PRINT
'END
OF
ASSEMBLY'

WRITE END OF
FILE RECORD
ON LOCAL
DICTIONARY FILE

CALL
PAGE

CALL SAS
WRITE
TABLES

RETURN

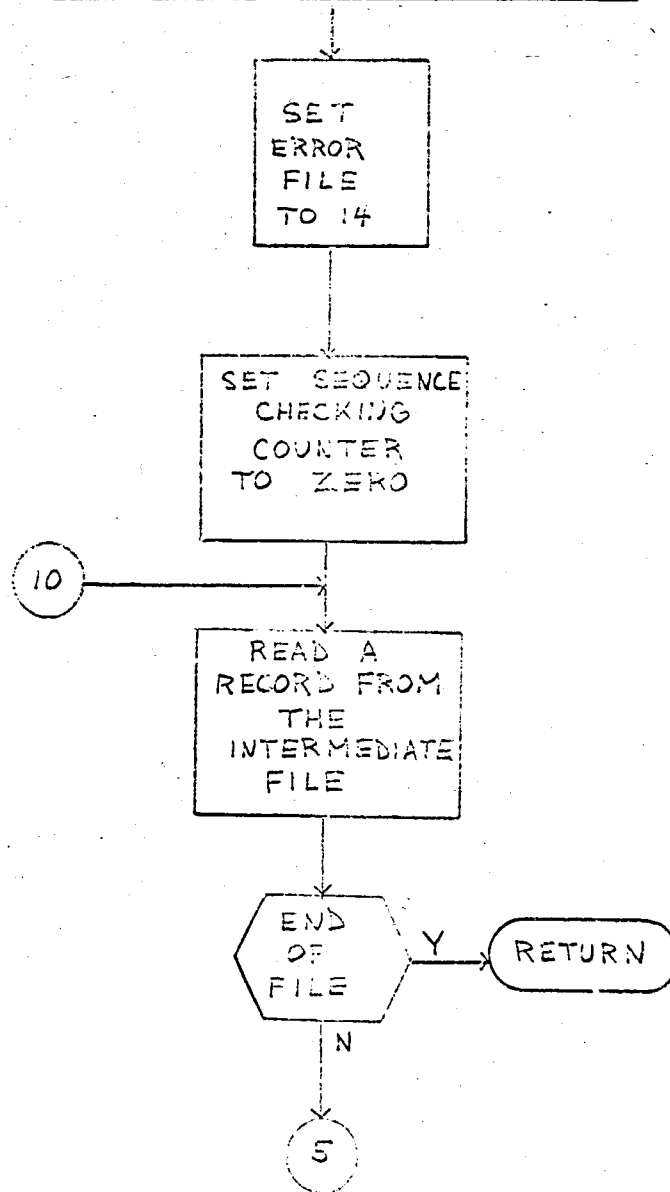
SUBROUTINE ASP

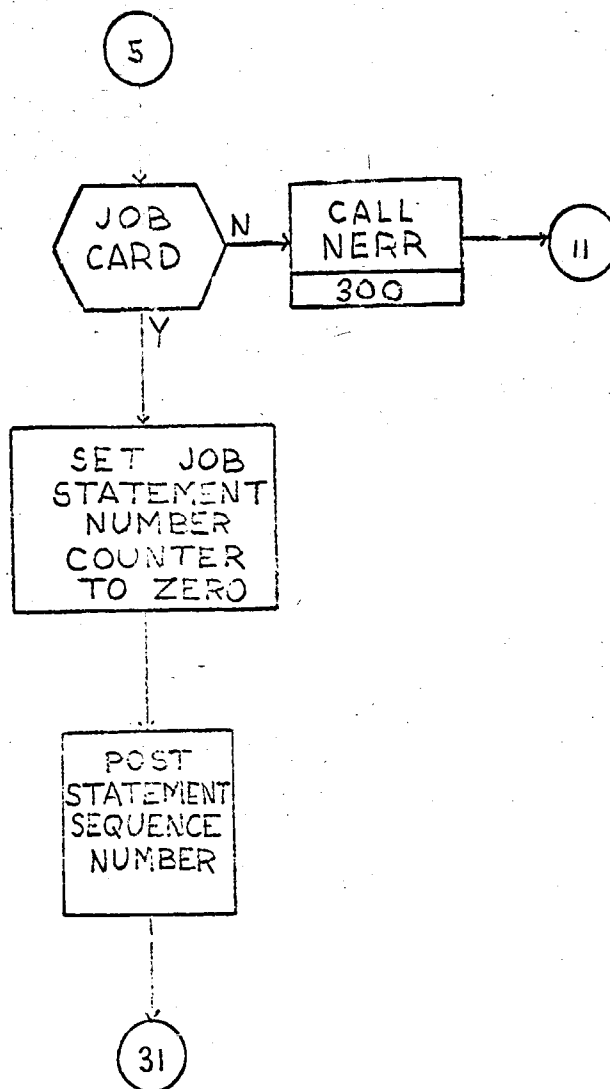
This is the assembler second pass routine.

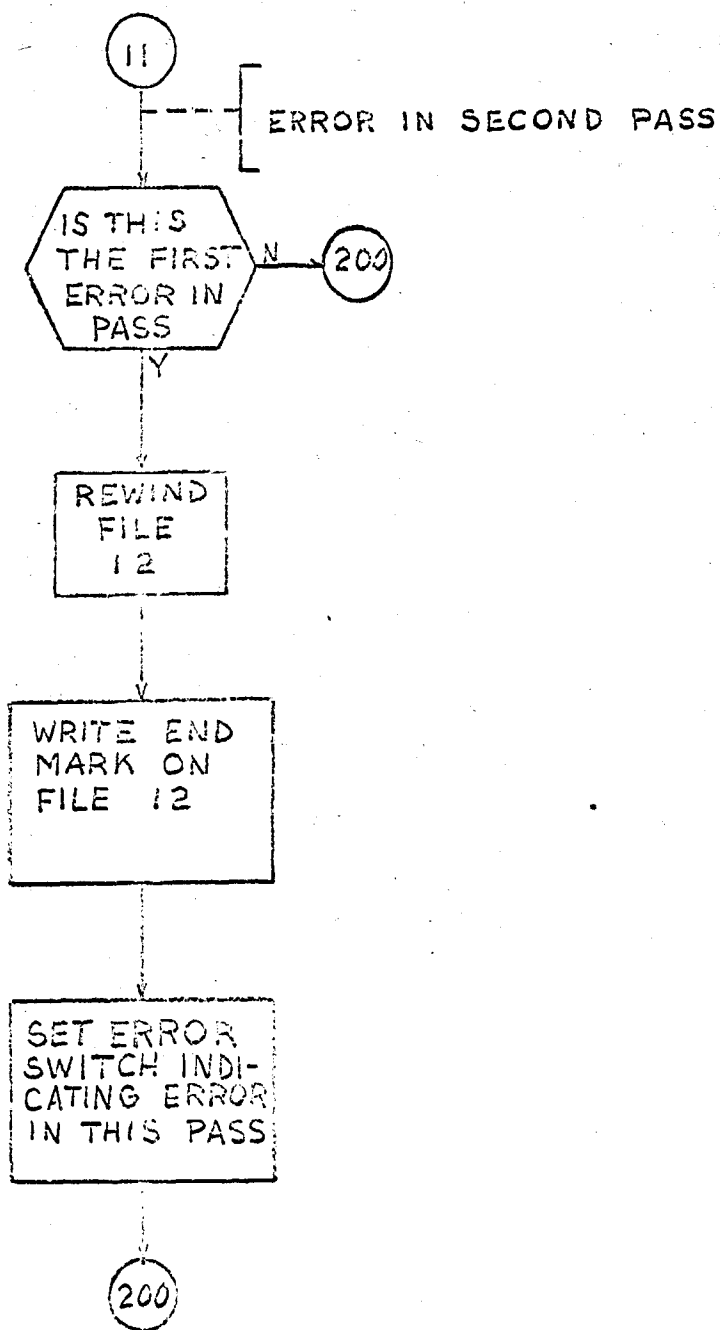
It processes the intermediate file records into assembler input records. It is a table driven translator. For each statement, there is a table entry. In the entry there is a field to indicate whether pre-operand processing is required, whether post operand processing is required, operand processing for each of the six possible operands for the given statement, and a field containing the length of the statement as it is used by the simulator. As each statement is read from the intermediate file the table entry is consulted and the appropriate pre, post, and operand processing routines are used.

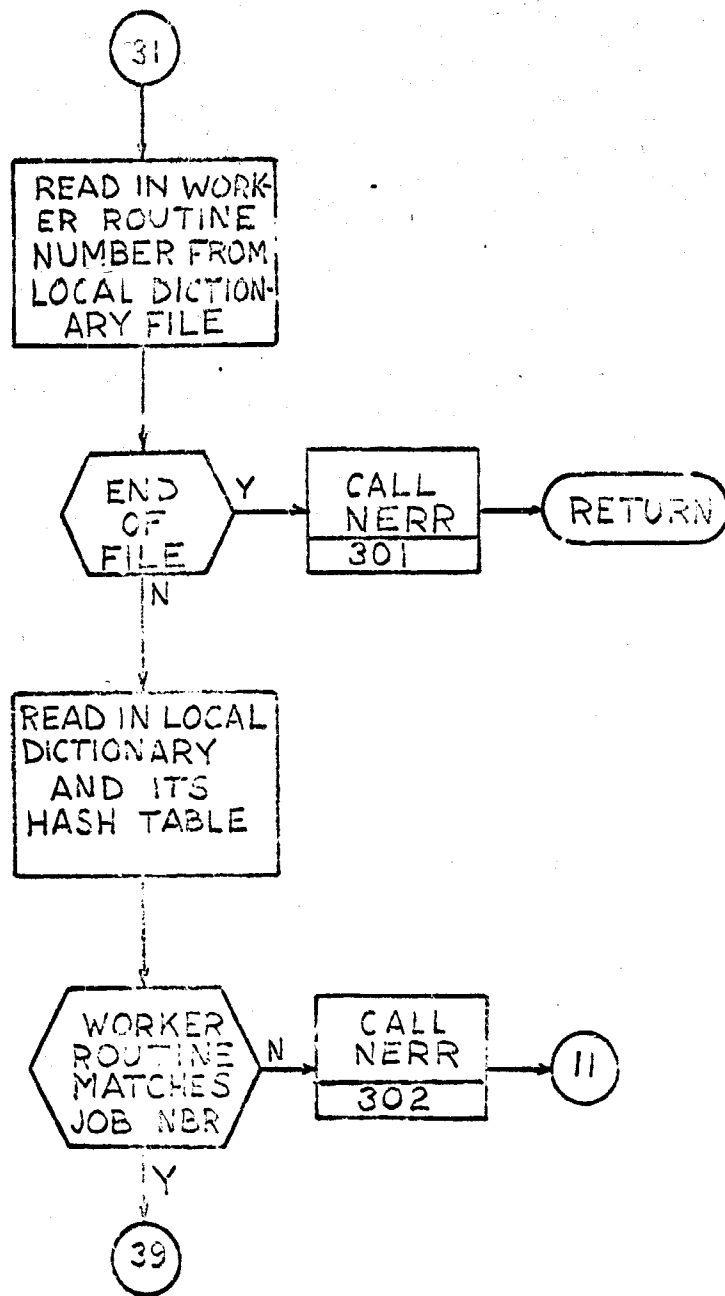
17A

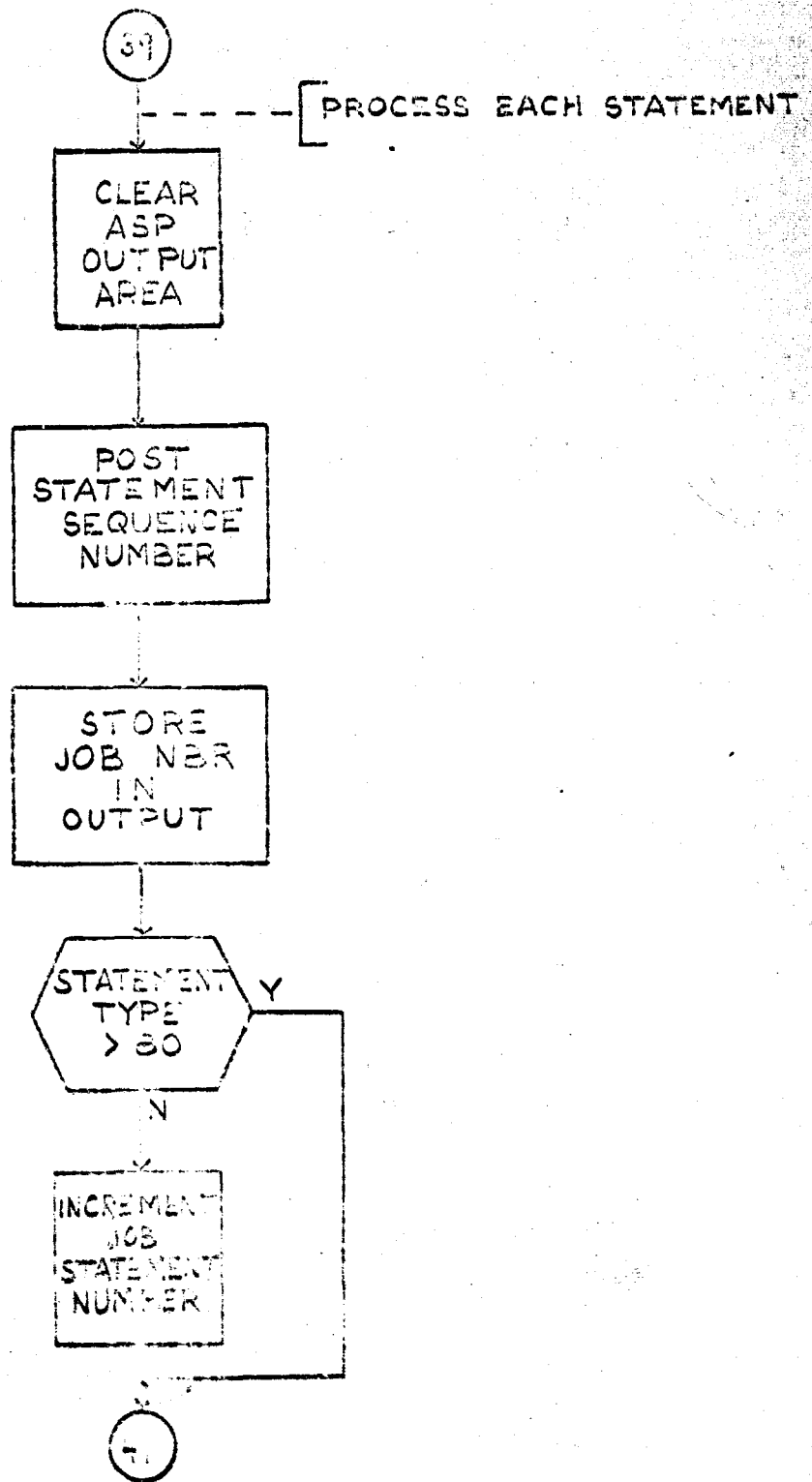
SUBROUTINE ASP
(ASSEMBLER SECOND PASS)











123

(41)

STORE JOB
STATEMENT
NUMBER IN
OUTPUT

STORE
OP-CODE
IN
OUTPUT

OP-CODE
IN ASP
TABLE

CALL
NERR
341

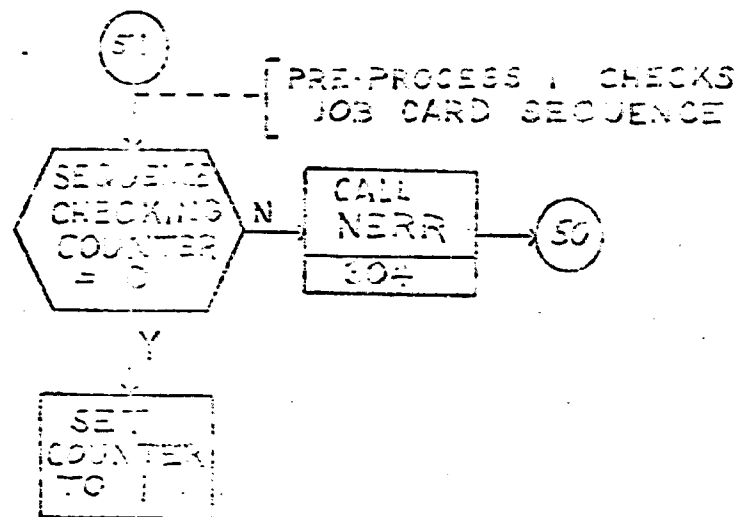
(11)

PRE-
PROCESSING
REQUIRED

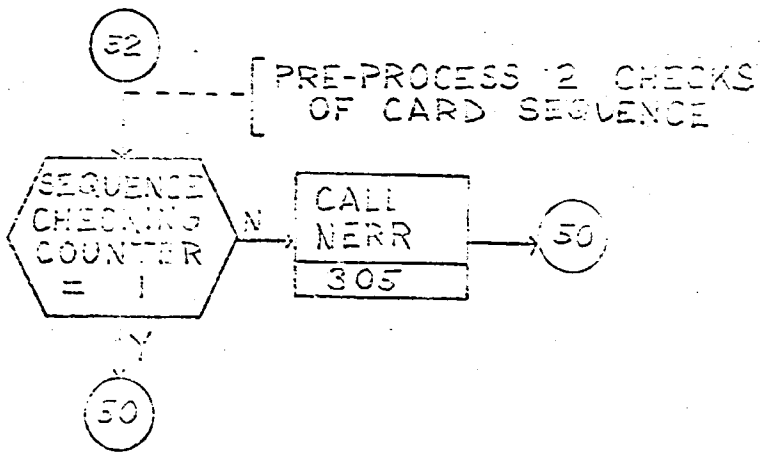
N

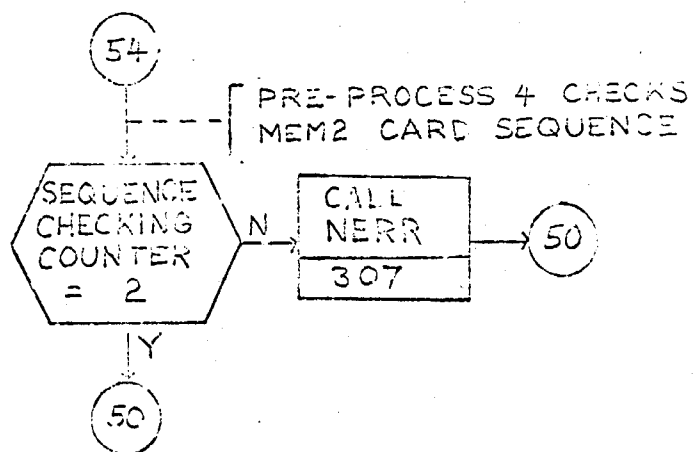
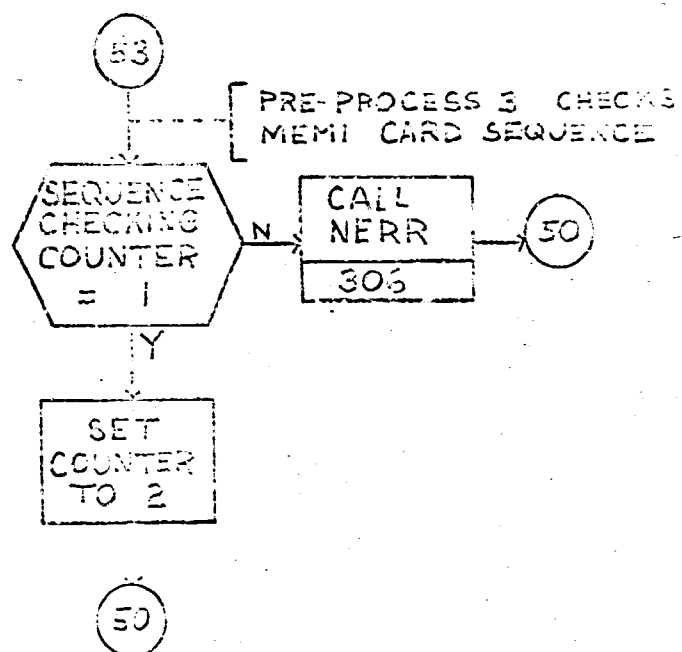
(50)

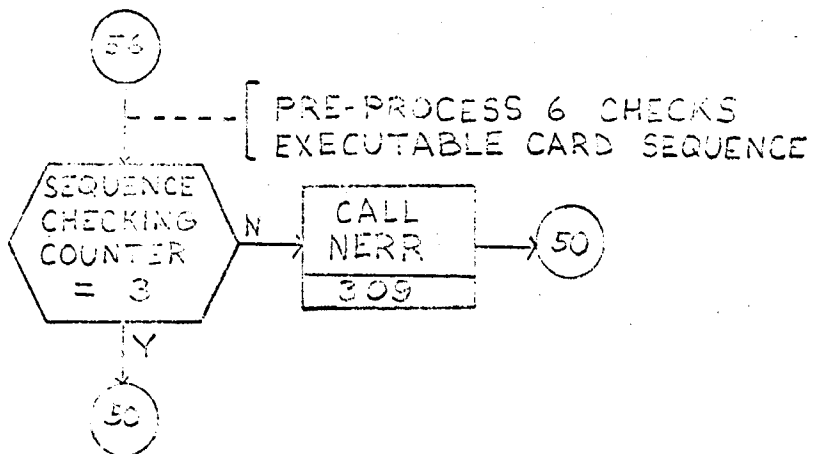
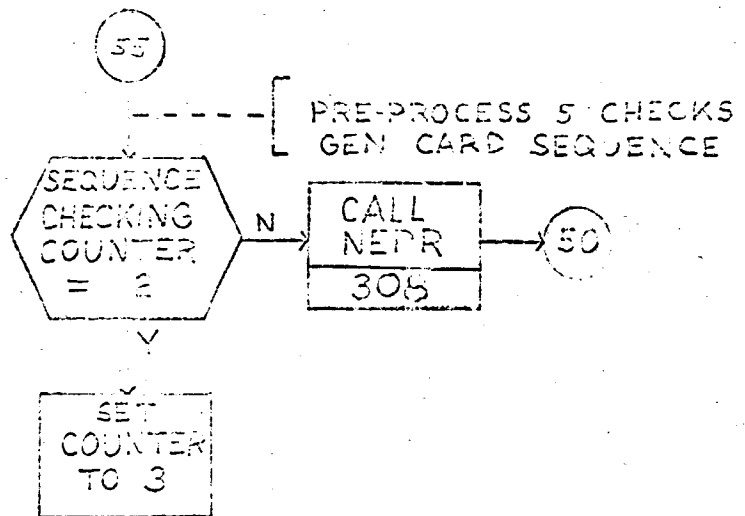
GO TO PRE-
PROCESS



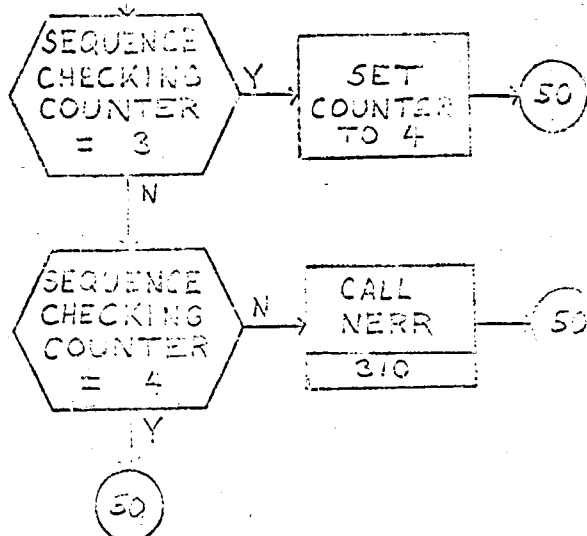
50



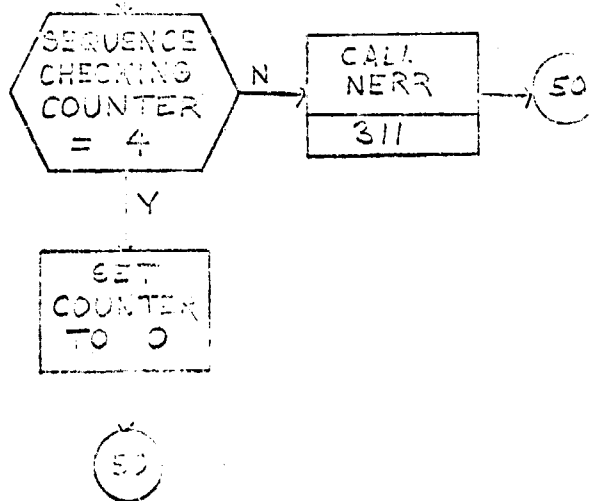


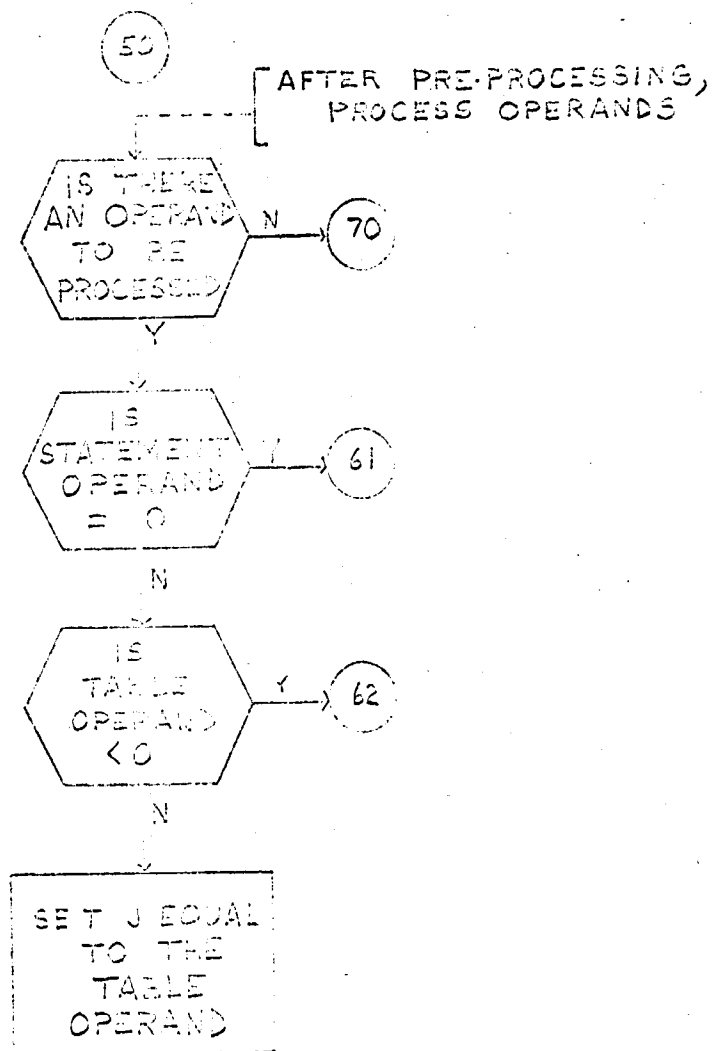


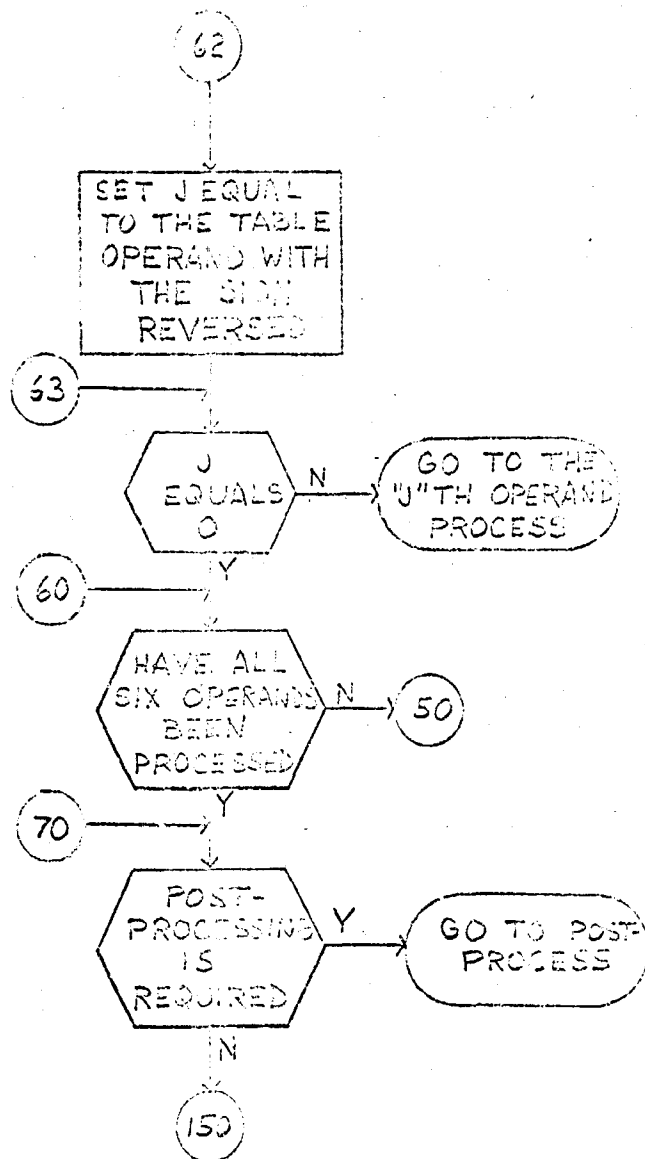
57 [PRE-PROCESS 7 CHECKS
TERMINATE CARD SEQUENCE

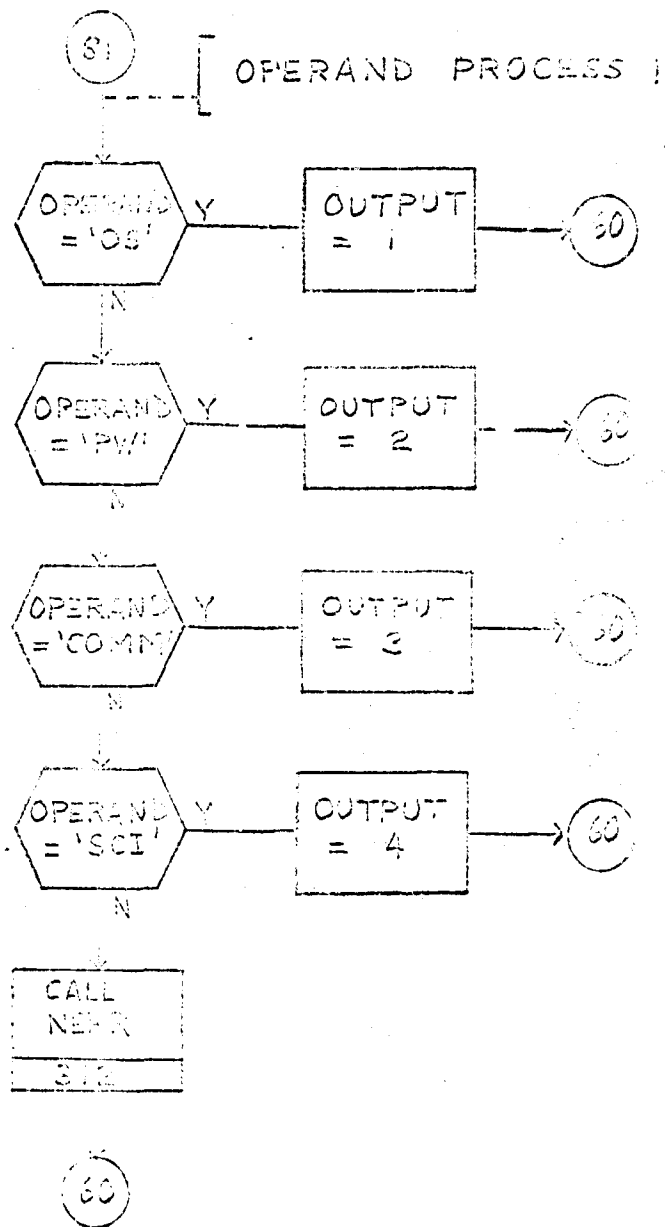


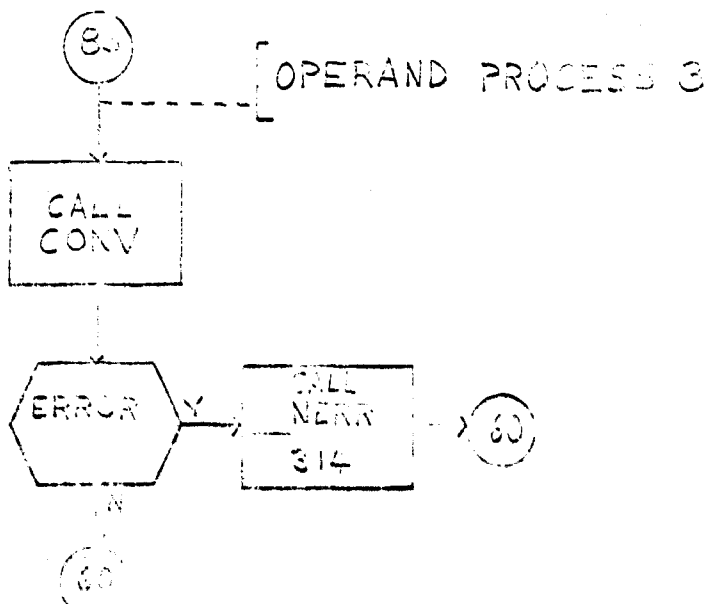
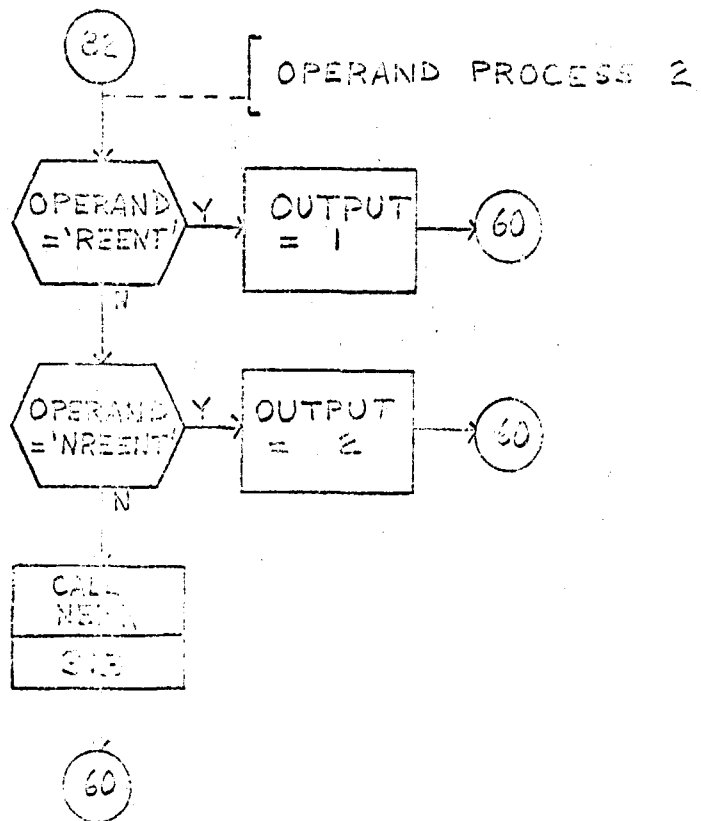
58 [PRE-PROCESS 8 CHECKS
END CARD SEQUENCE

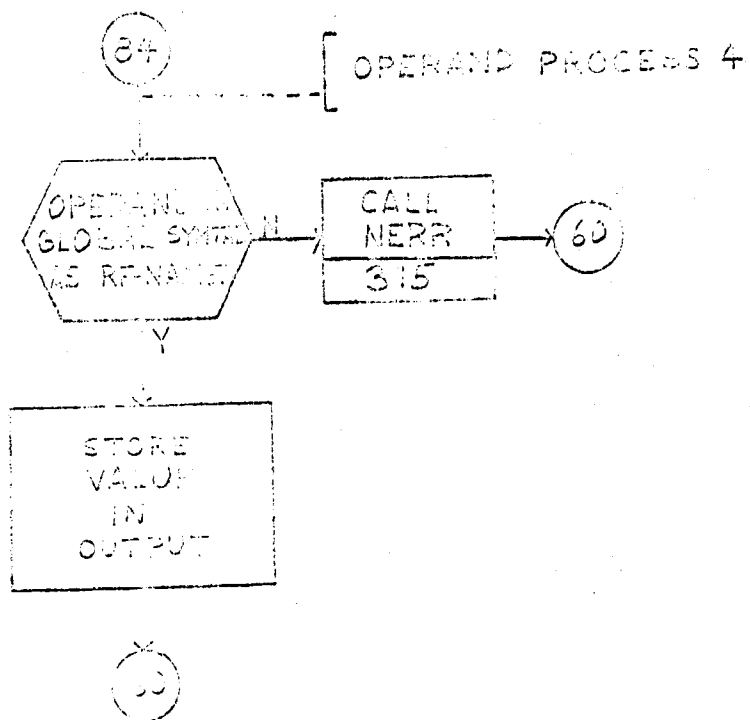












130

85

OPERAND PROCESS 5

CALL
EVAL

ERROR

Y

CALL
NERR
315

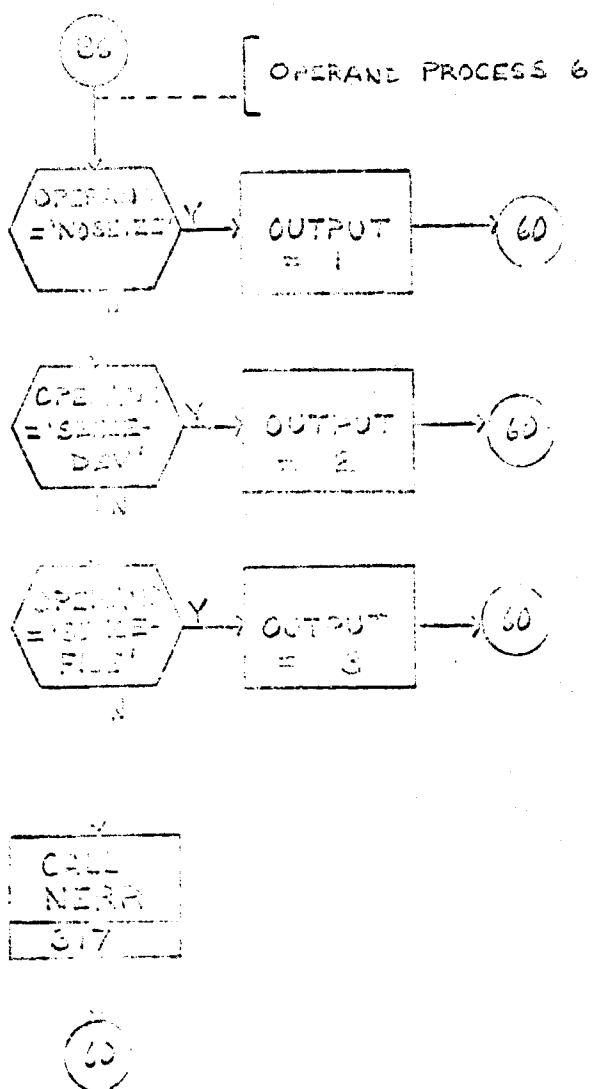
60

N

STORE
OUTPUT

60

146



280

67

OPERAND PROCESS 7

OPERAND
= 'FX'

Y

OUTPUT
= 1

60

N

OPERAND
= 'POISE'

Y

OUTPUT
= 2

60

N

CALL
NEBR
319

60

112

55

OPERAND PROCESS 3

CALL
CONV

ERROR

CALL
NERR
374

60

RANGE
IS
1-99

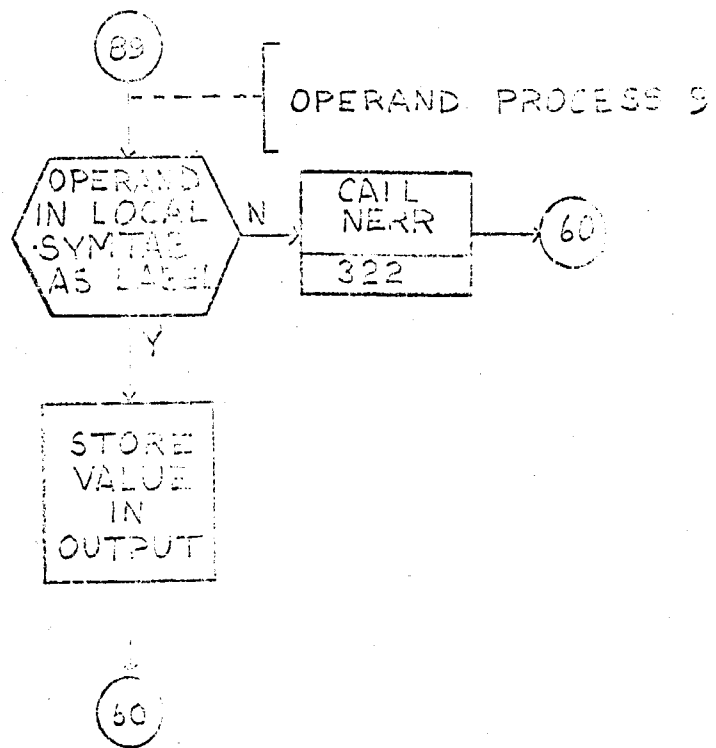
CALL
NERR
321

60

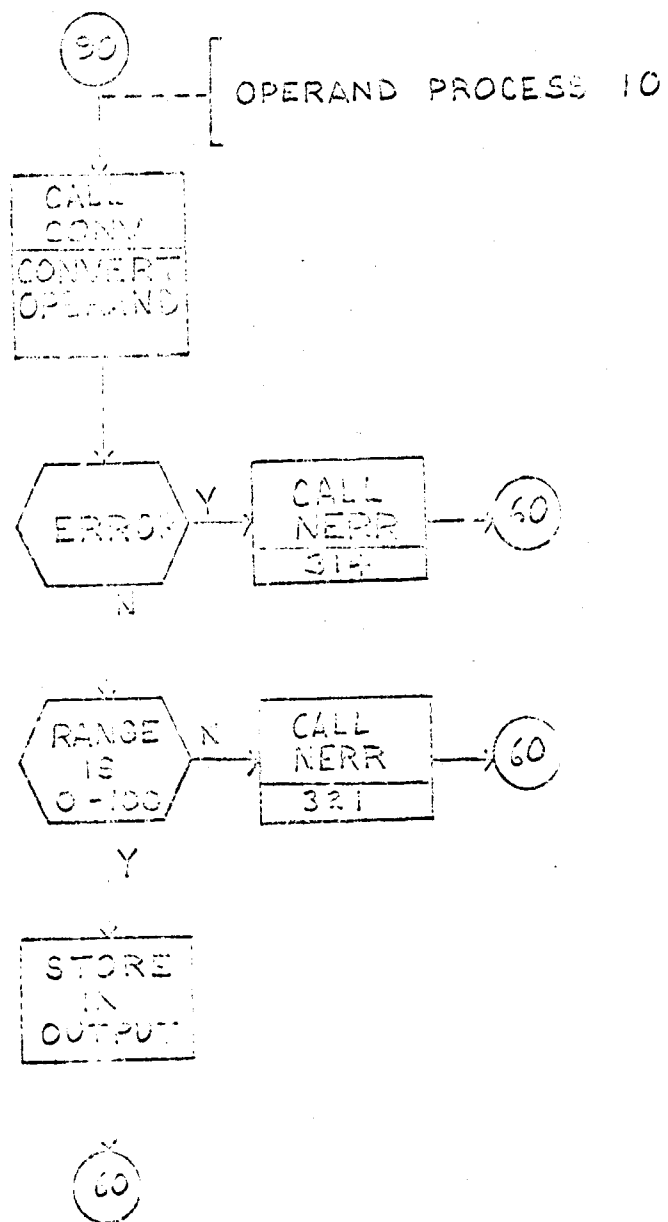
STORE
IN
OUTPUT

60

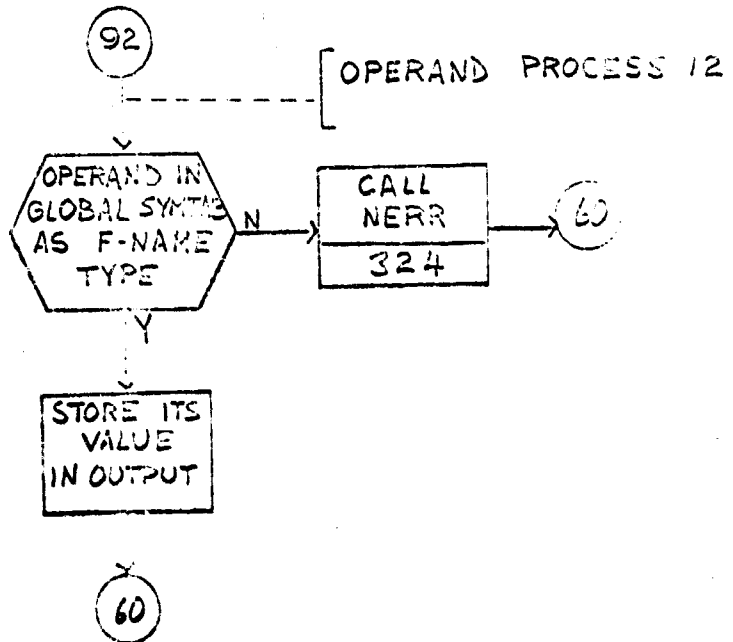
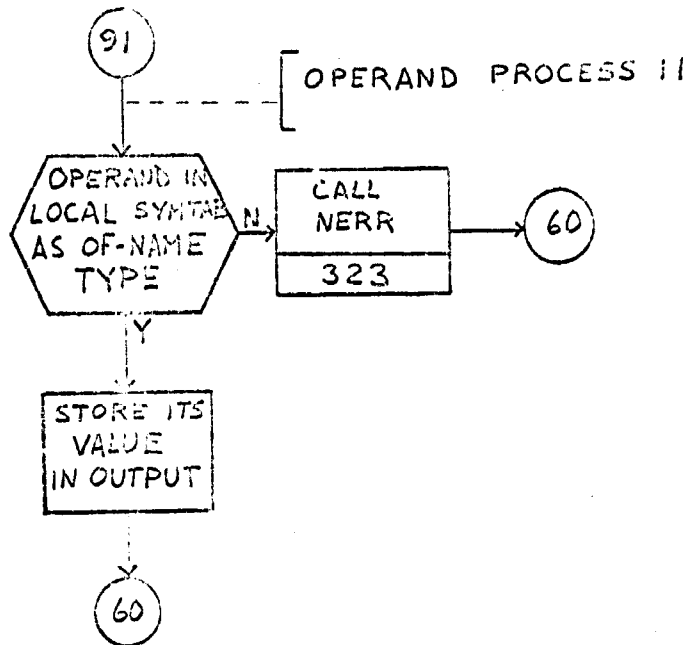
142



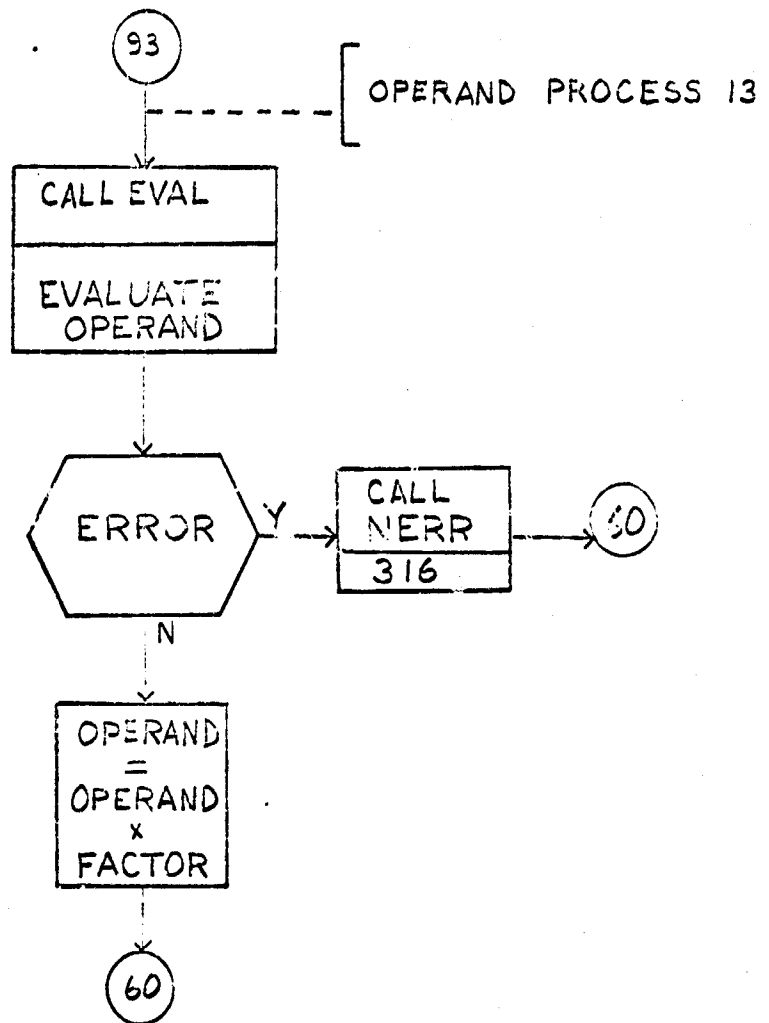
144

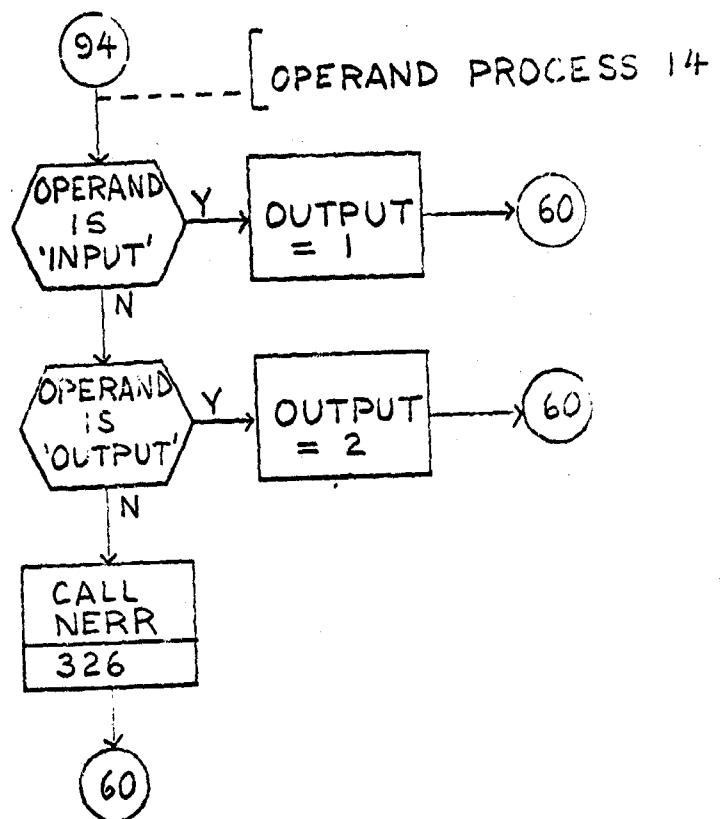


145

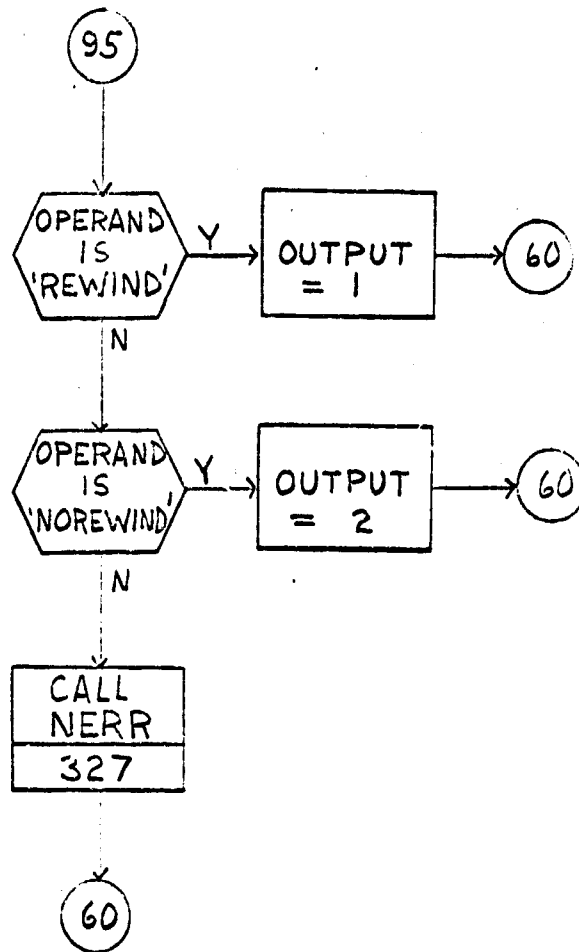


146

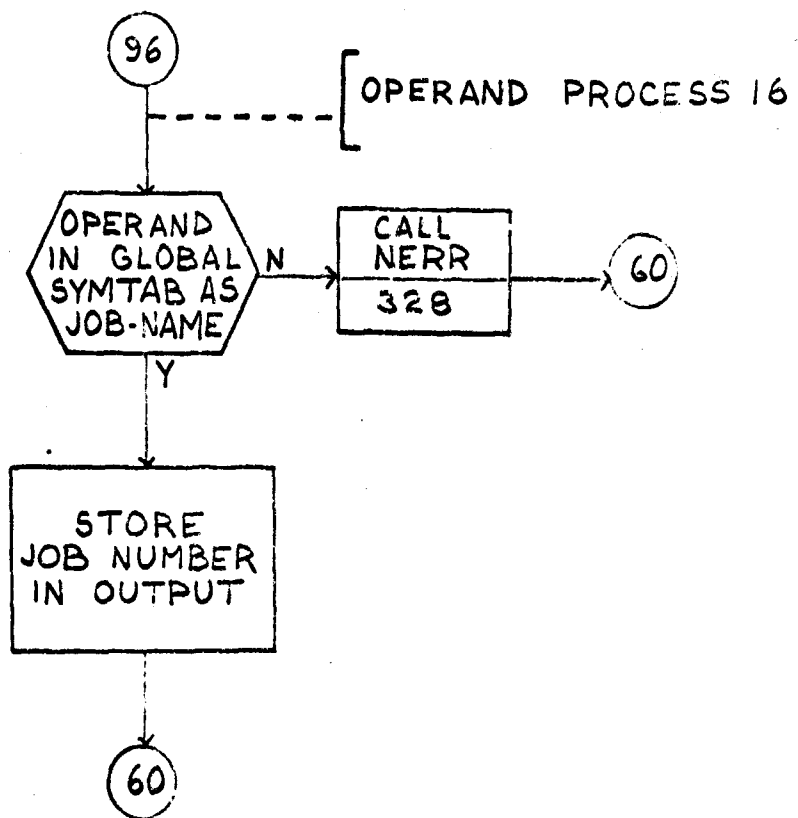


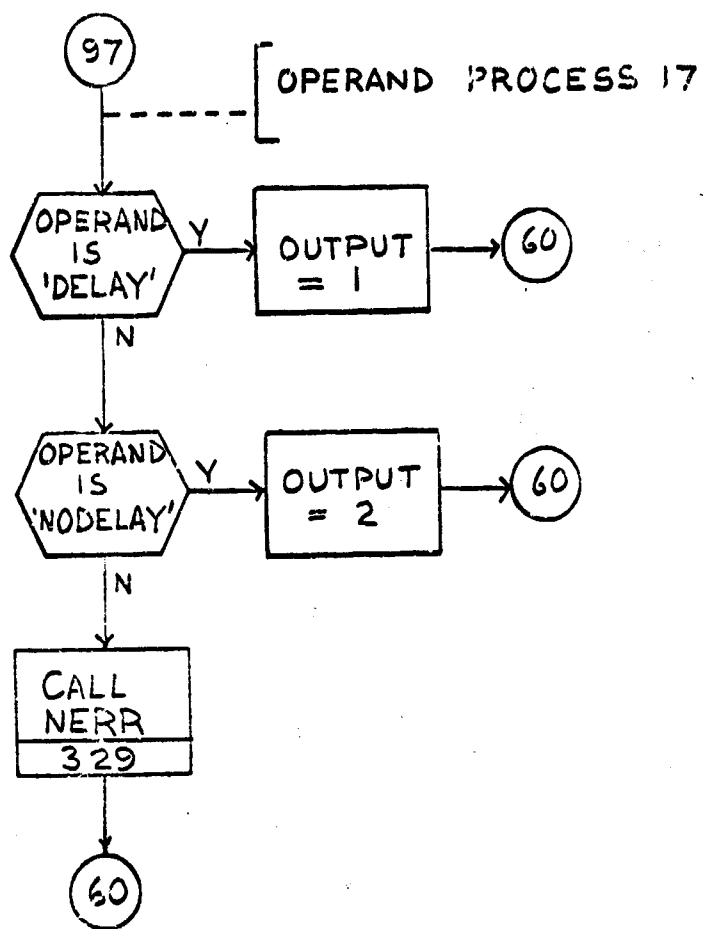


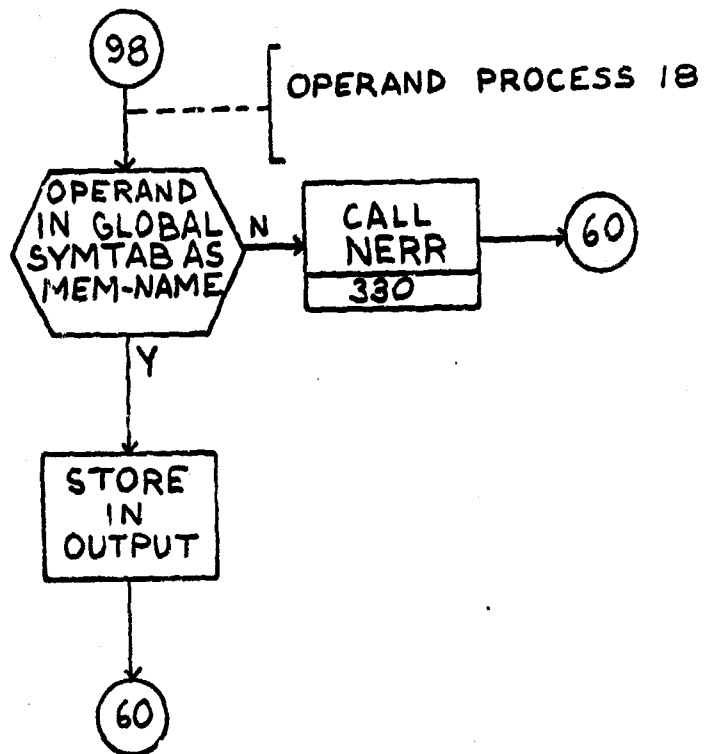
148

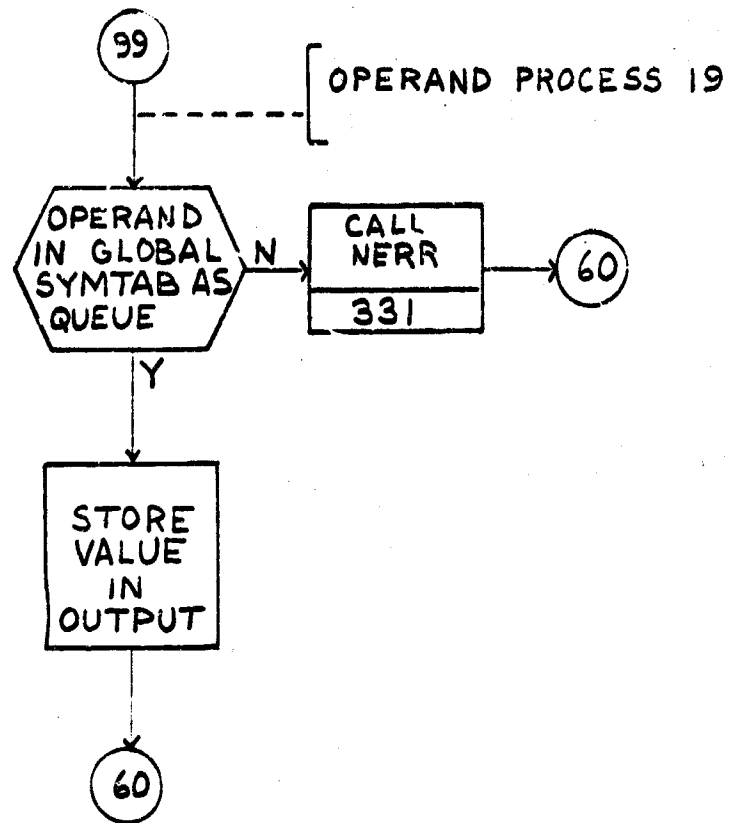


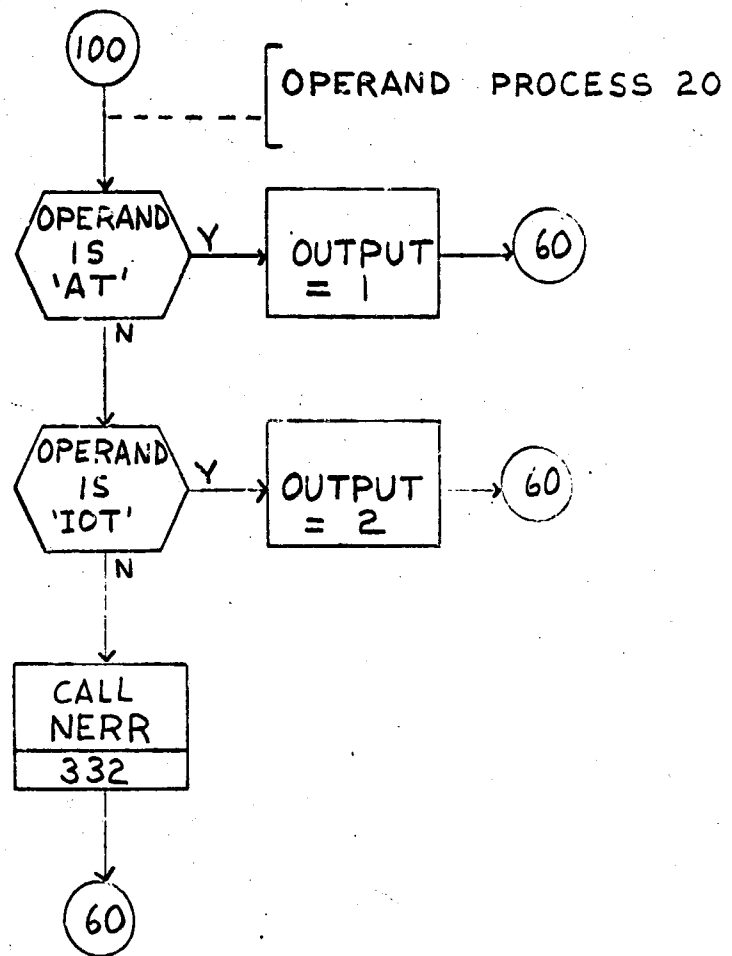
149

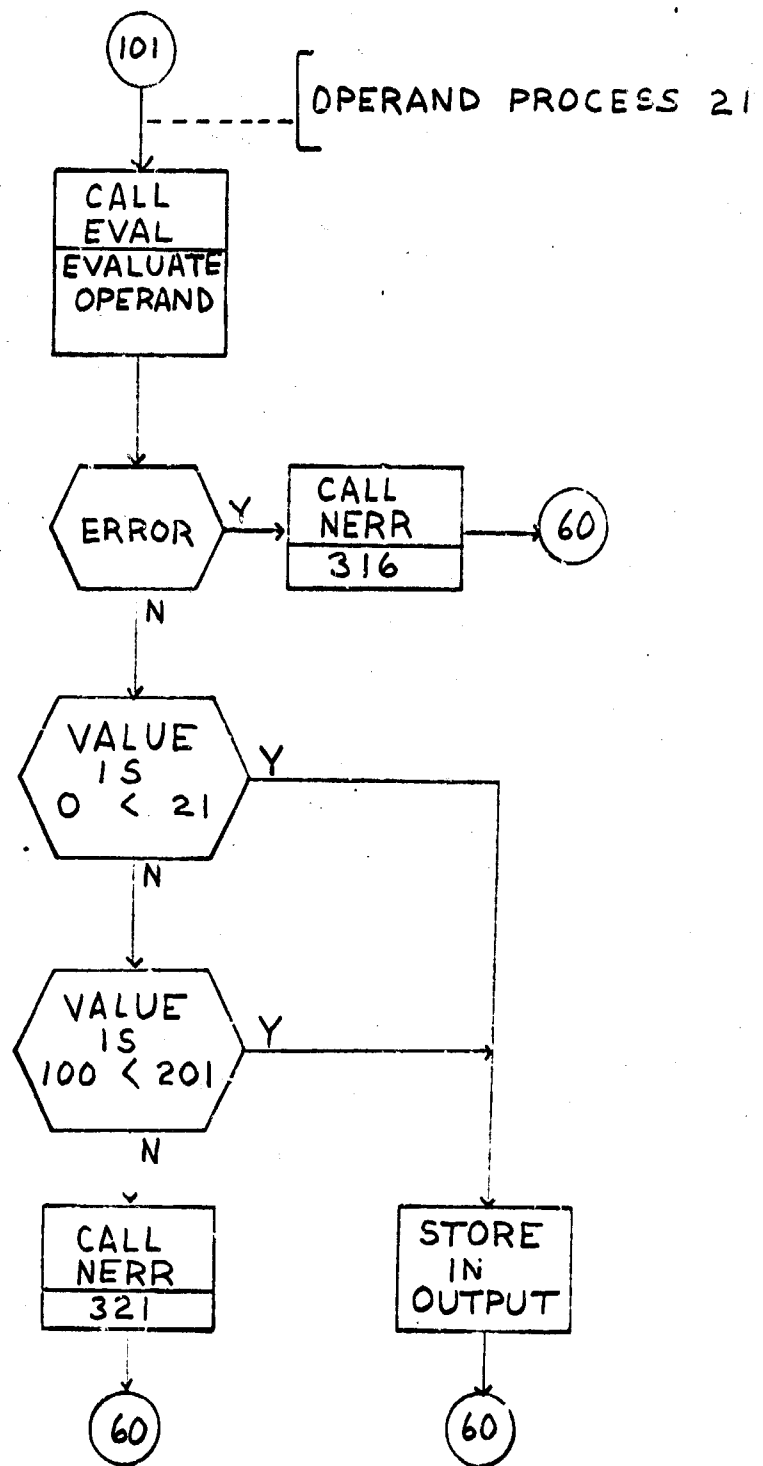


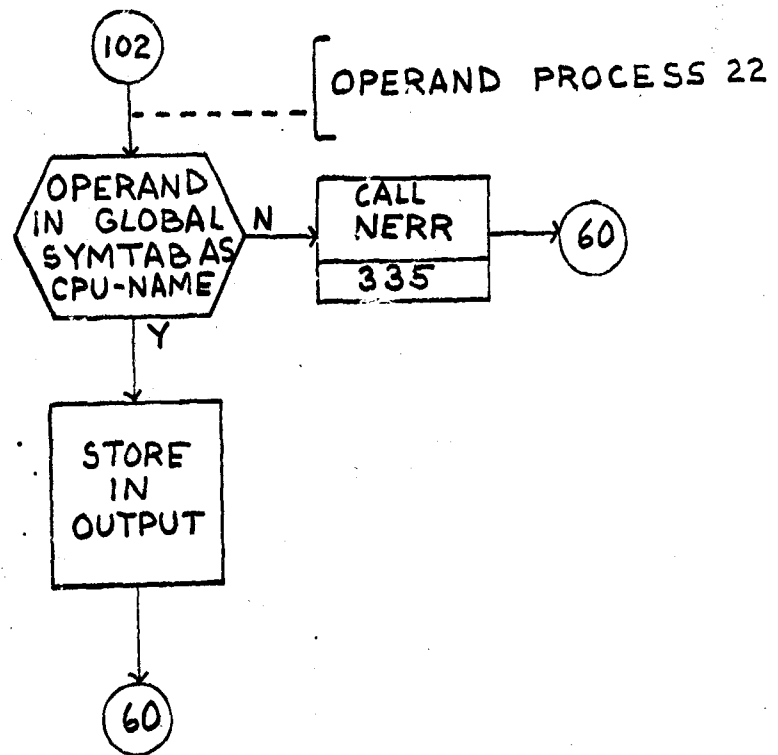




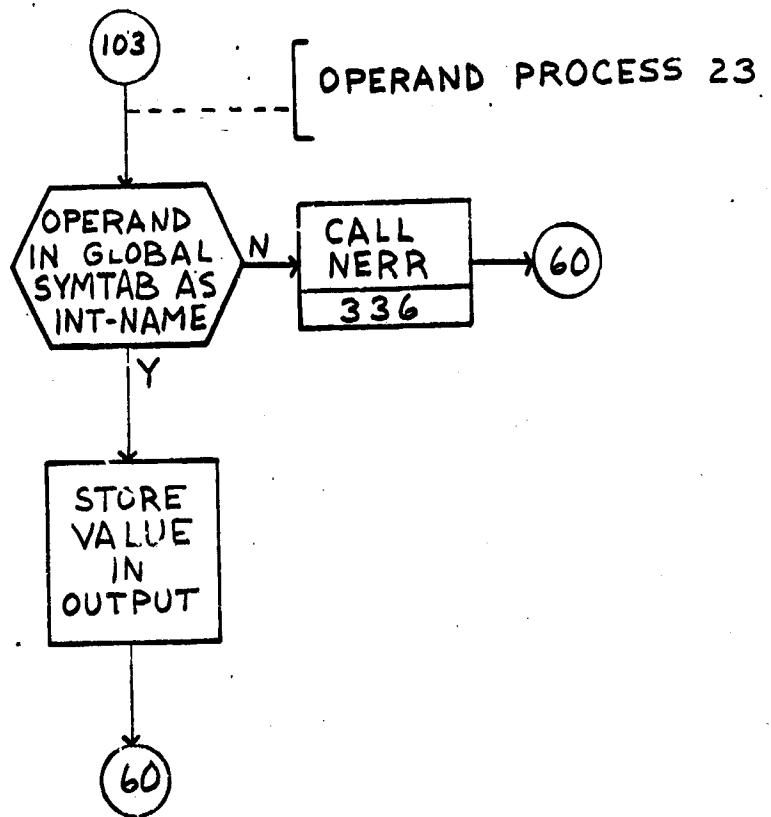






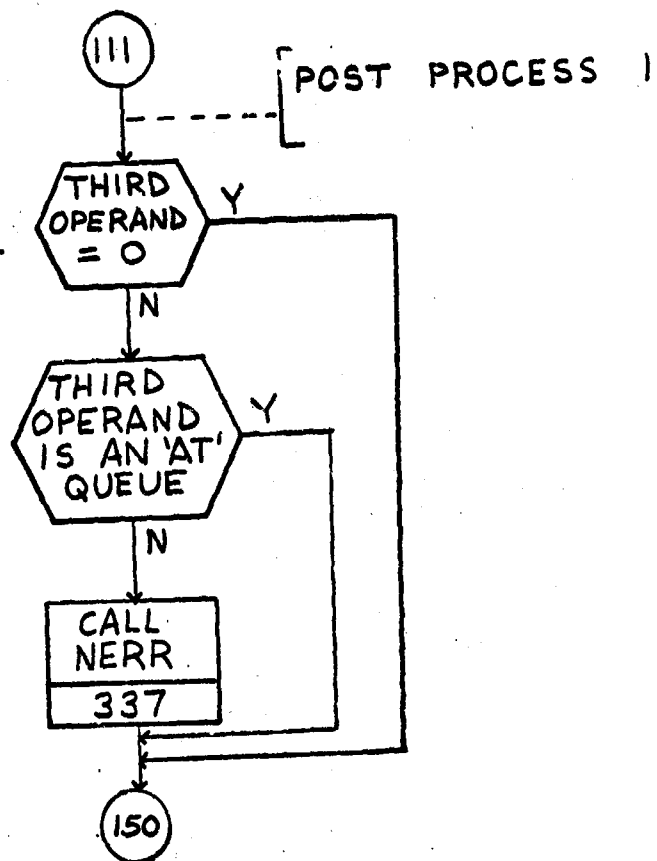


156

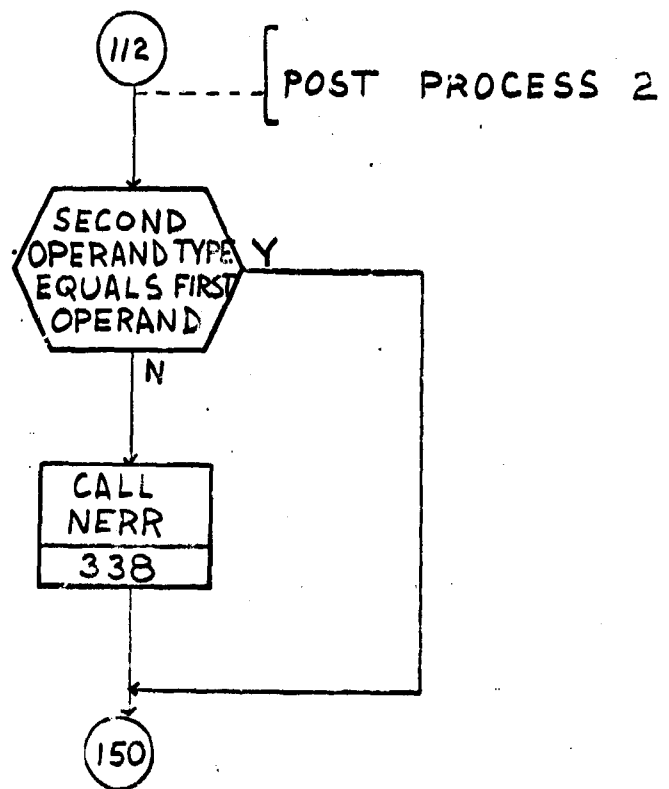


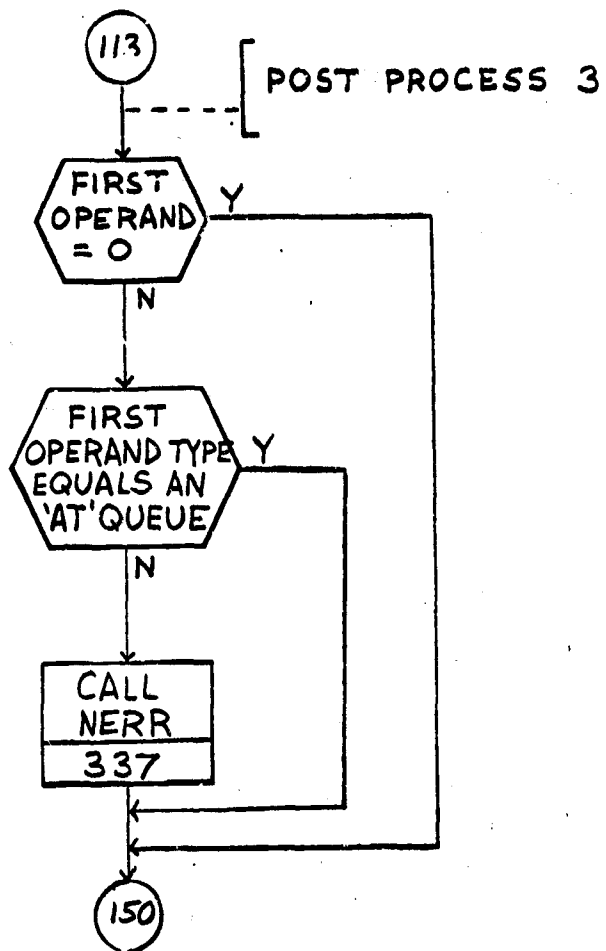
157

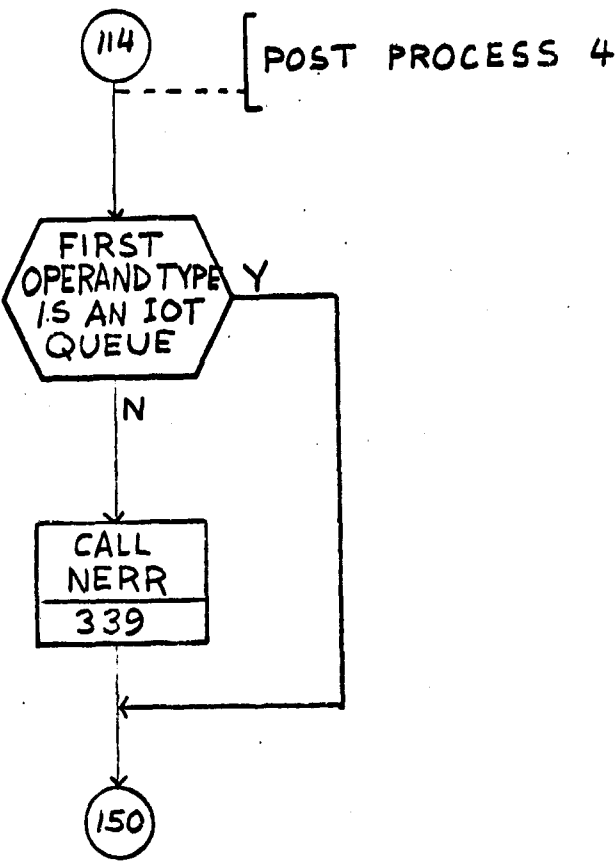
0

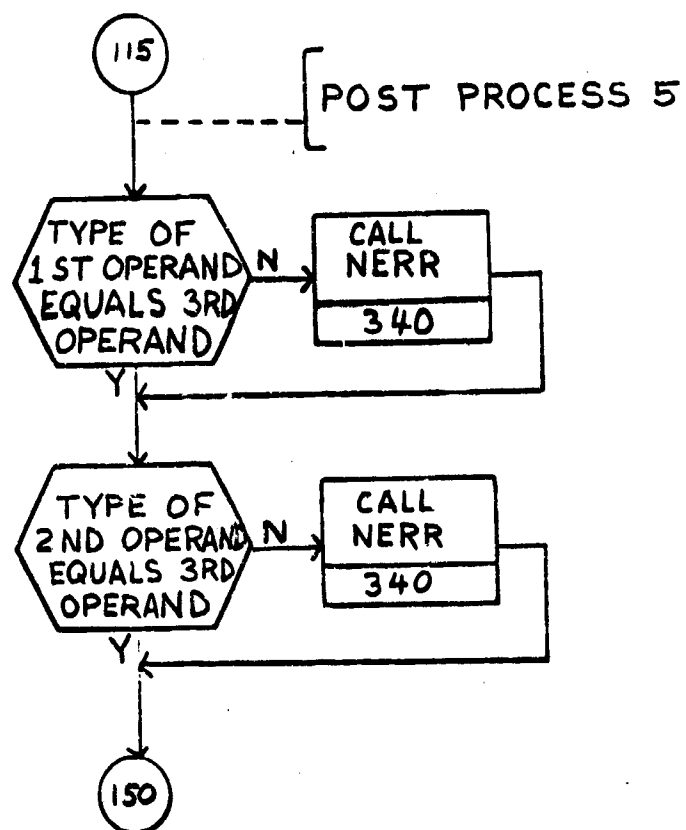


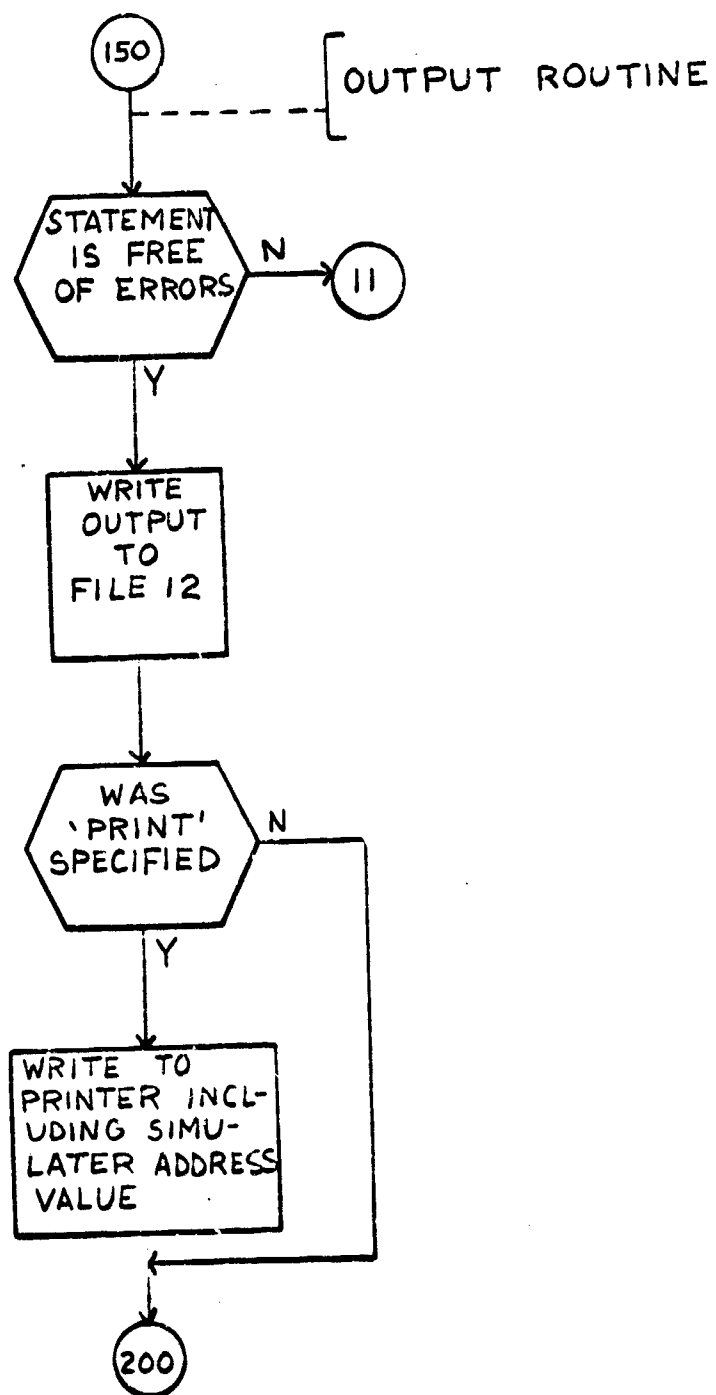
158

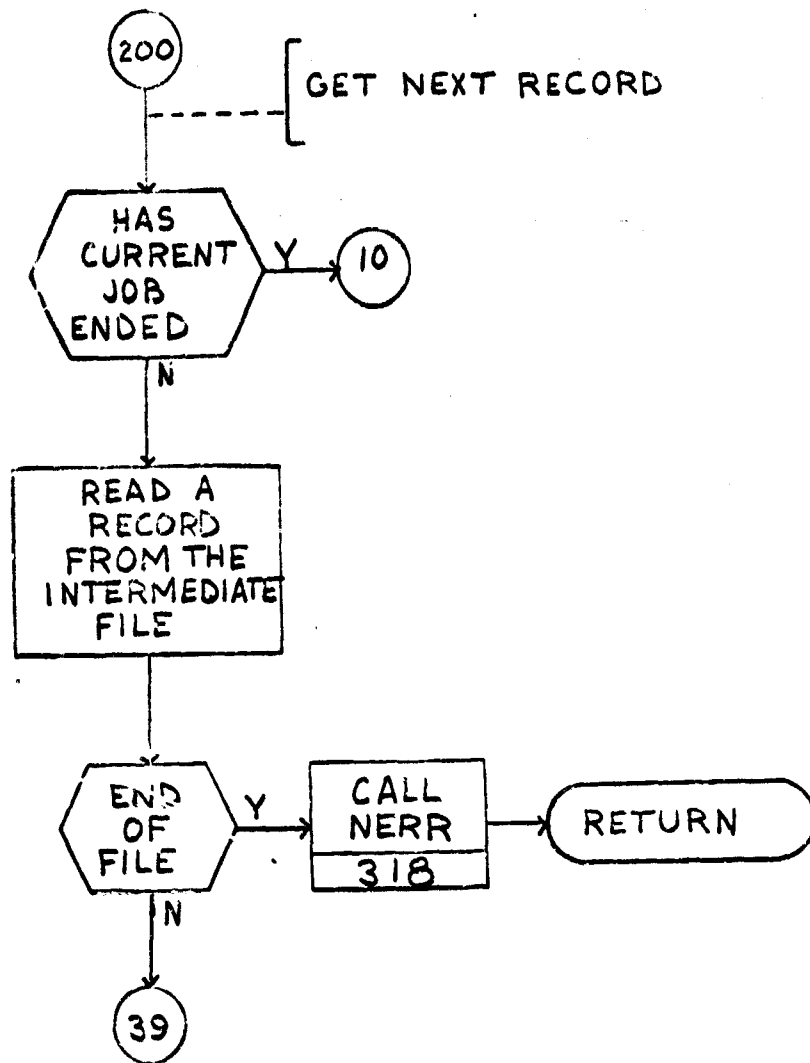






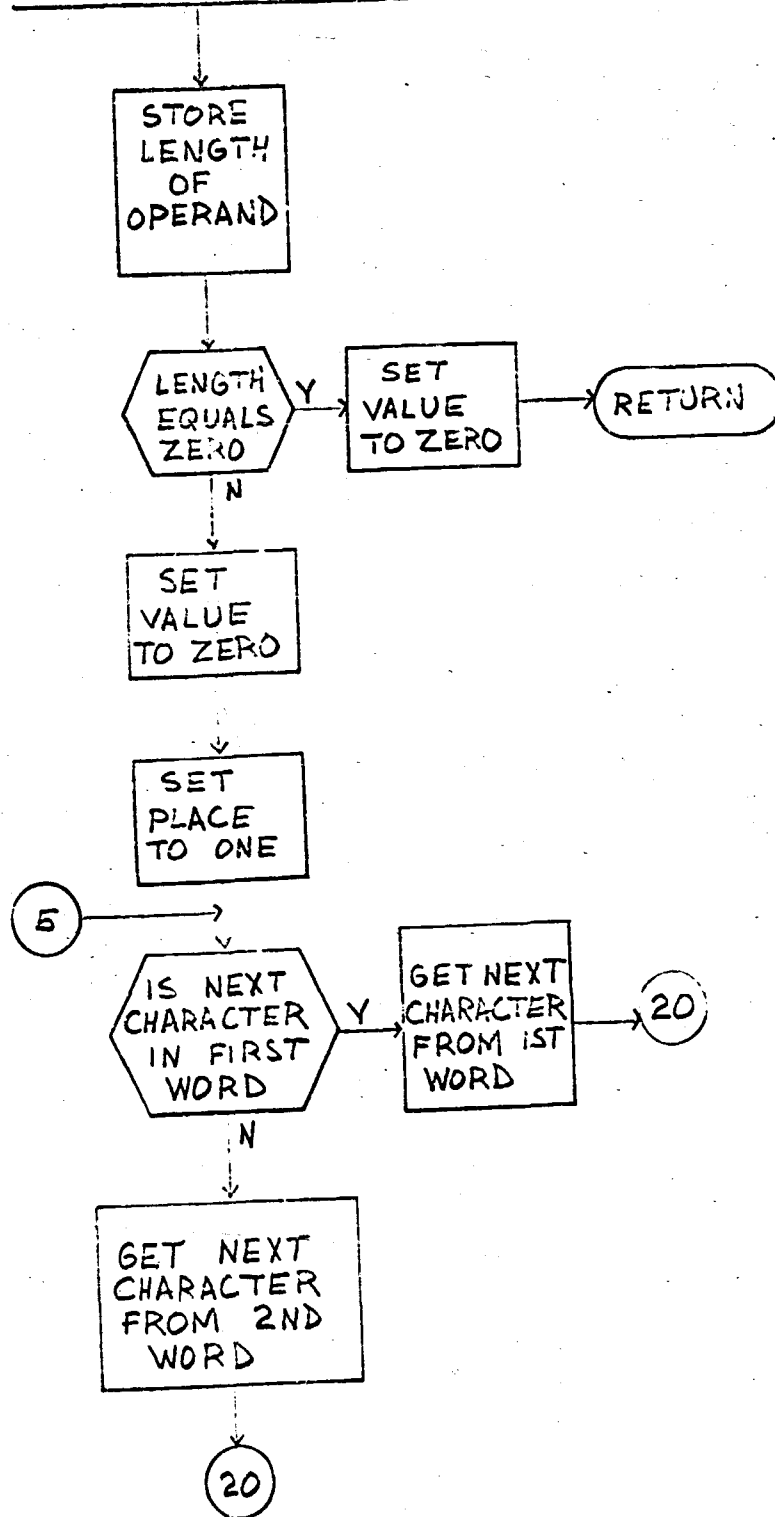


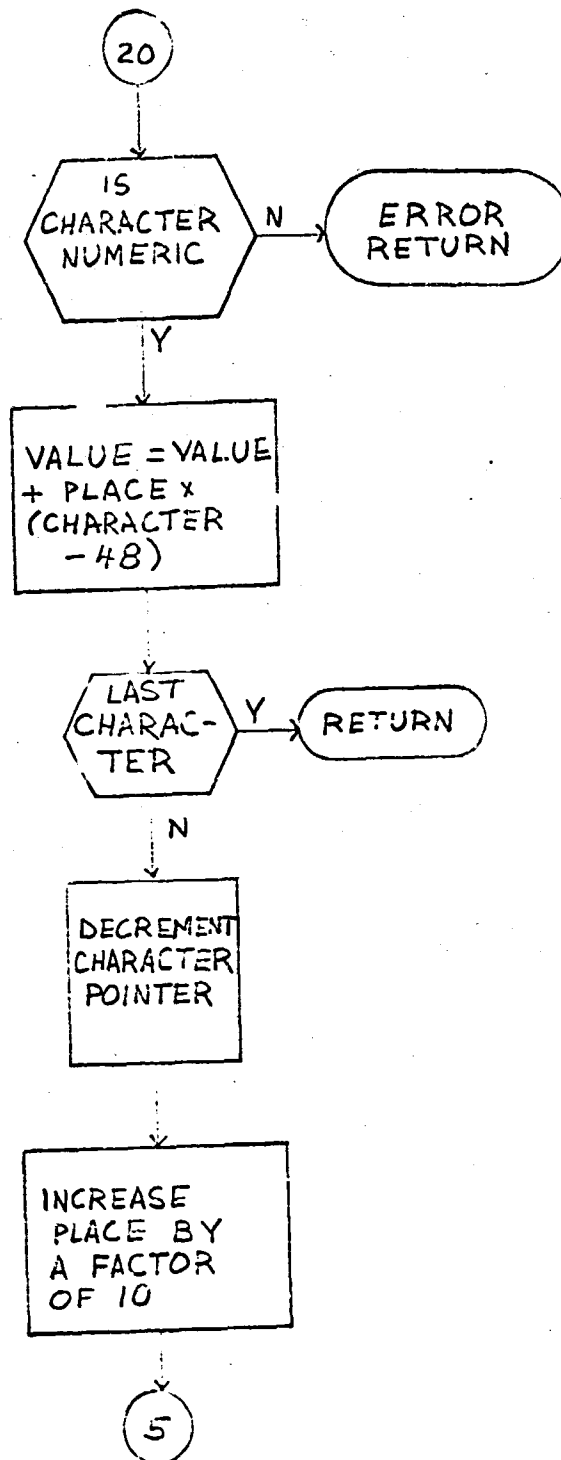




SUBROUTINE CONV

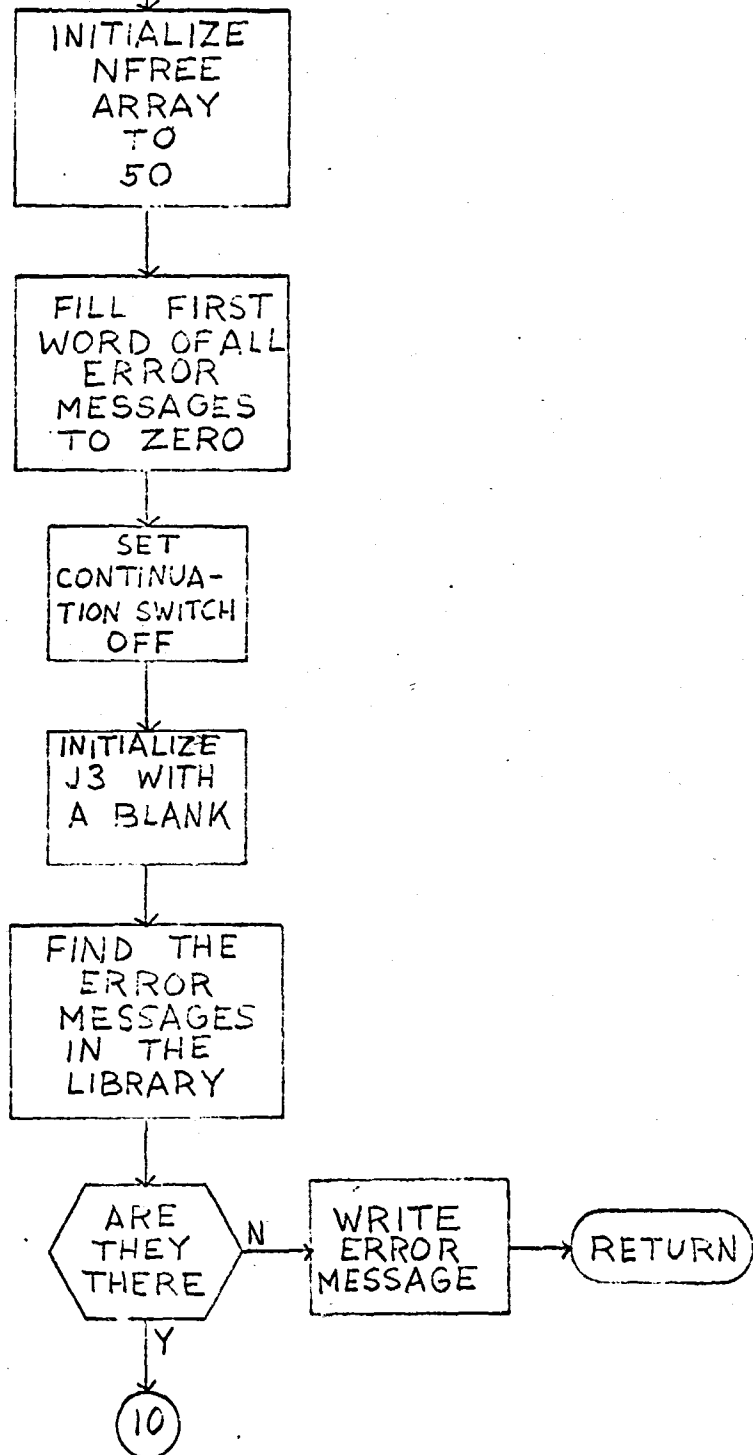
This is the routine which converts decimal representations of integers which are left justified in the fix field output format field into internal binary integer form. The routine is essentially a right to left scan of the decimal representation, in which each digit representation is converted to a binary value and multiplied by its positional value, and then added into a value. After all digits have been scanned, the net value, representing the binary value of the decimal representation, is returned.

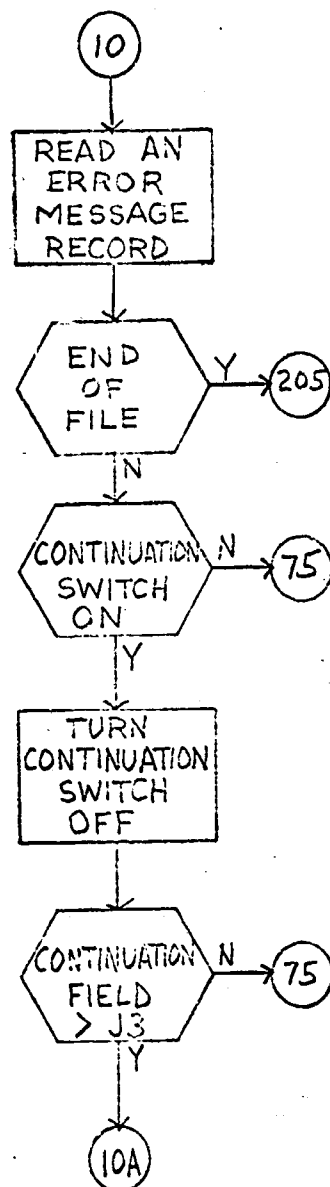
SUBROUTINE CONV
(DECIMAL TO BINARY)

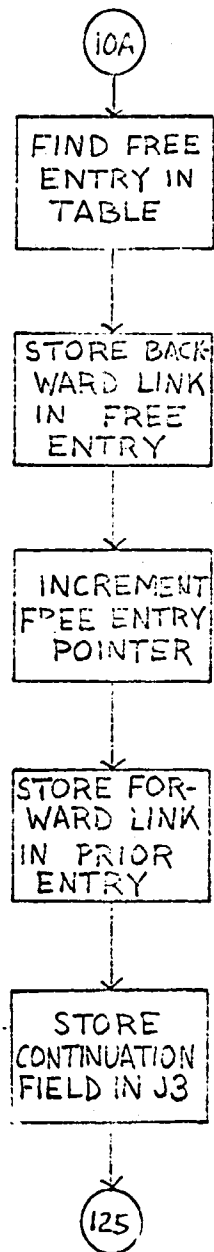


SUBROUTINE ERPASS

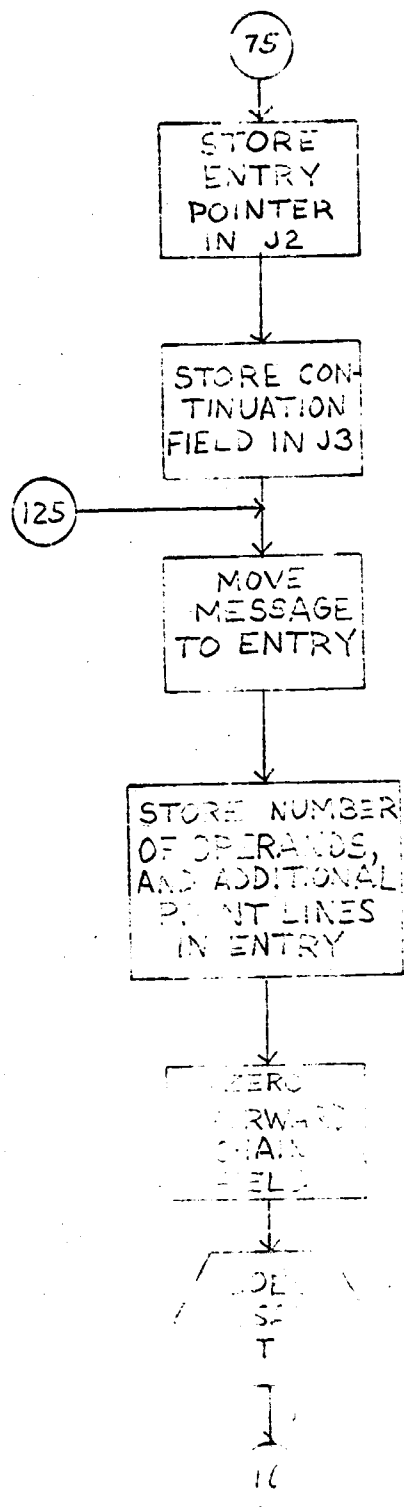
This routine merges the error files produced by the first and second passes and produces a listing of the errors in statement number sequence. For the sake of convenience, a prose diagnostic appears in addition to the statement number and error number.

SUBROUTINE ERPASS
(ERROR PASS)

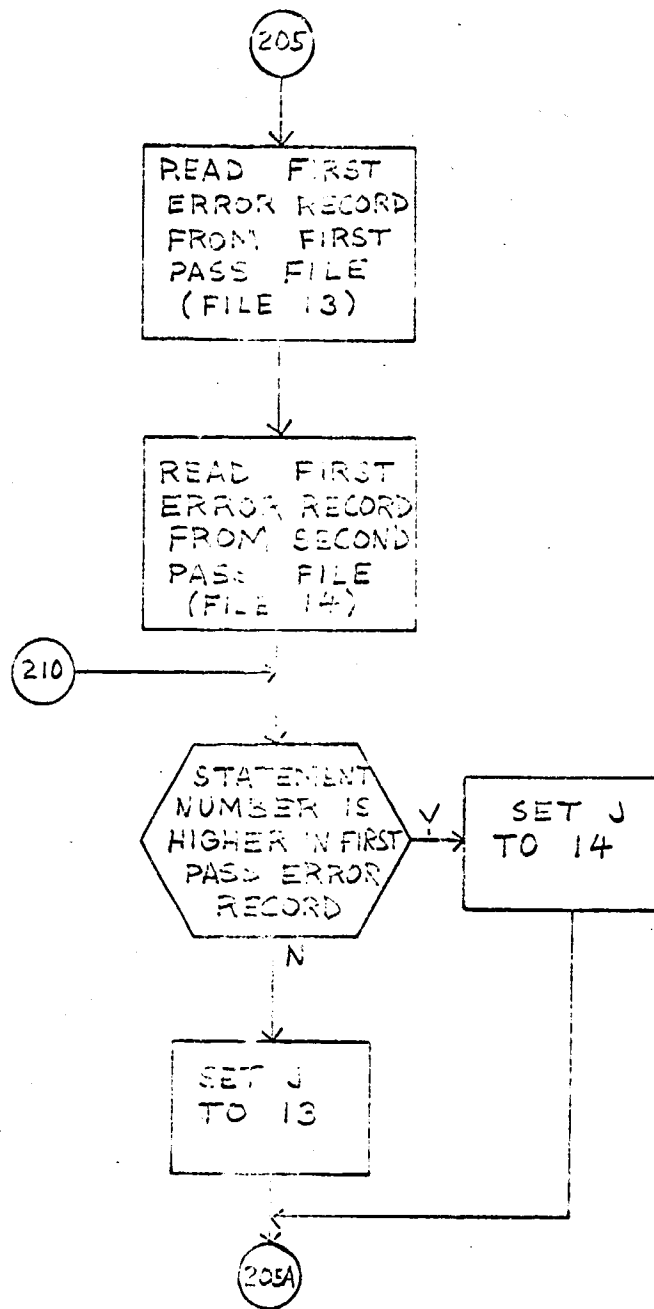


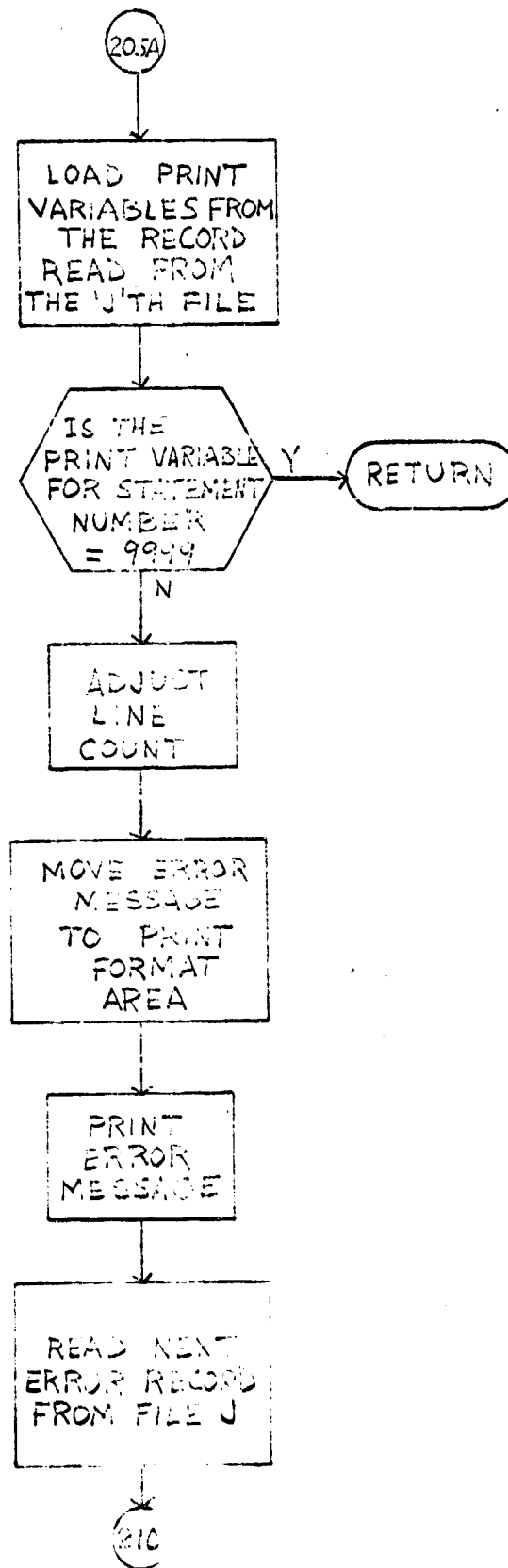


171



10



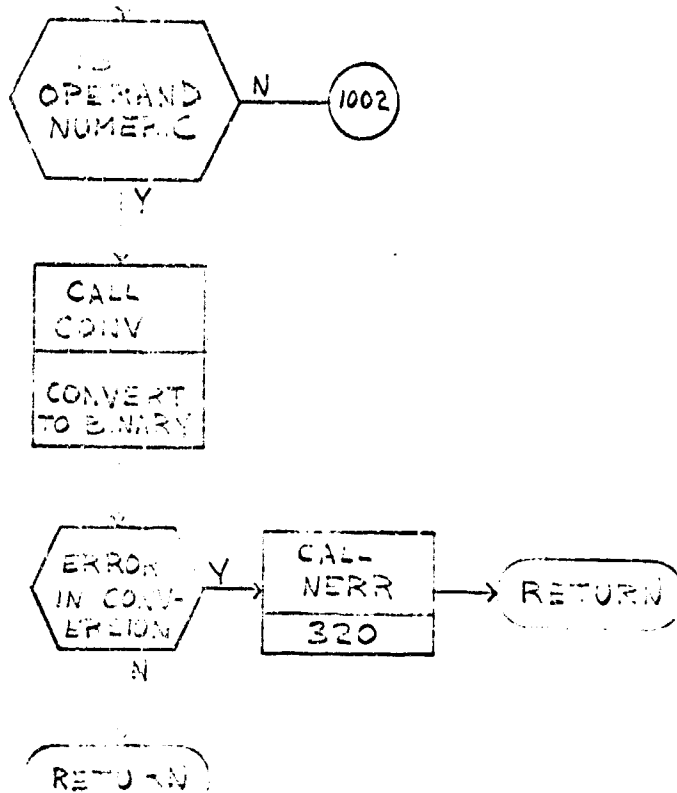


SUBROUTINE EVAL

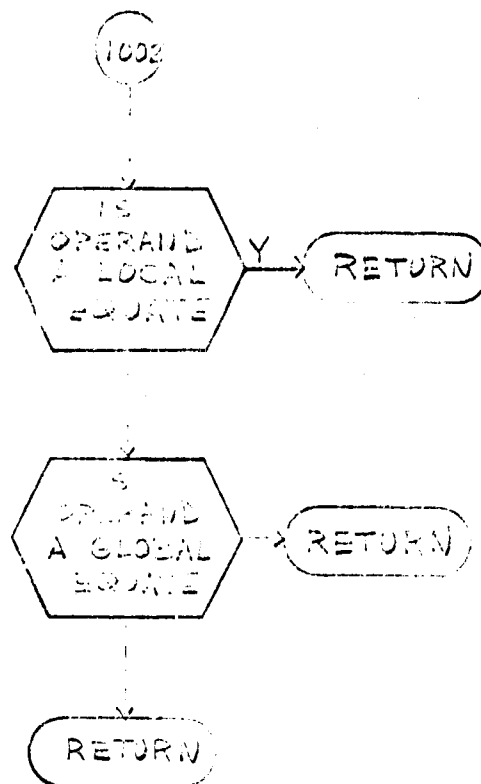
This routine evaluates operands which may be either a decimal representation of a constant, a local equate, or a global equate. A test on the first character is performed to see if it is numeric. If it is, then the CONV routine is called which converts it to internal binary form. If it is not a decimal representation of a number, then the local symbol table is searched to see if the operand is a local equate. If it is, the associated symbol table value is returned. If it is not a local equate, then the global symbol table is searched to see if it is a global equate. If it is, then the associated global symbol table value is returned.

3175

SUBROUTINE EVAL
(EVALUATE OPERAND)



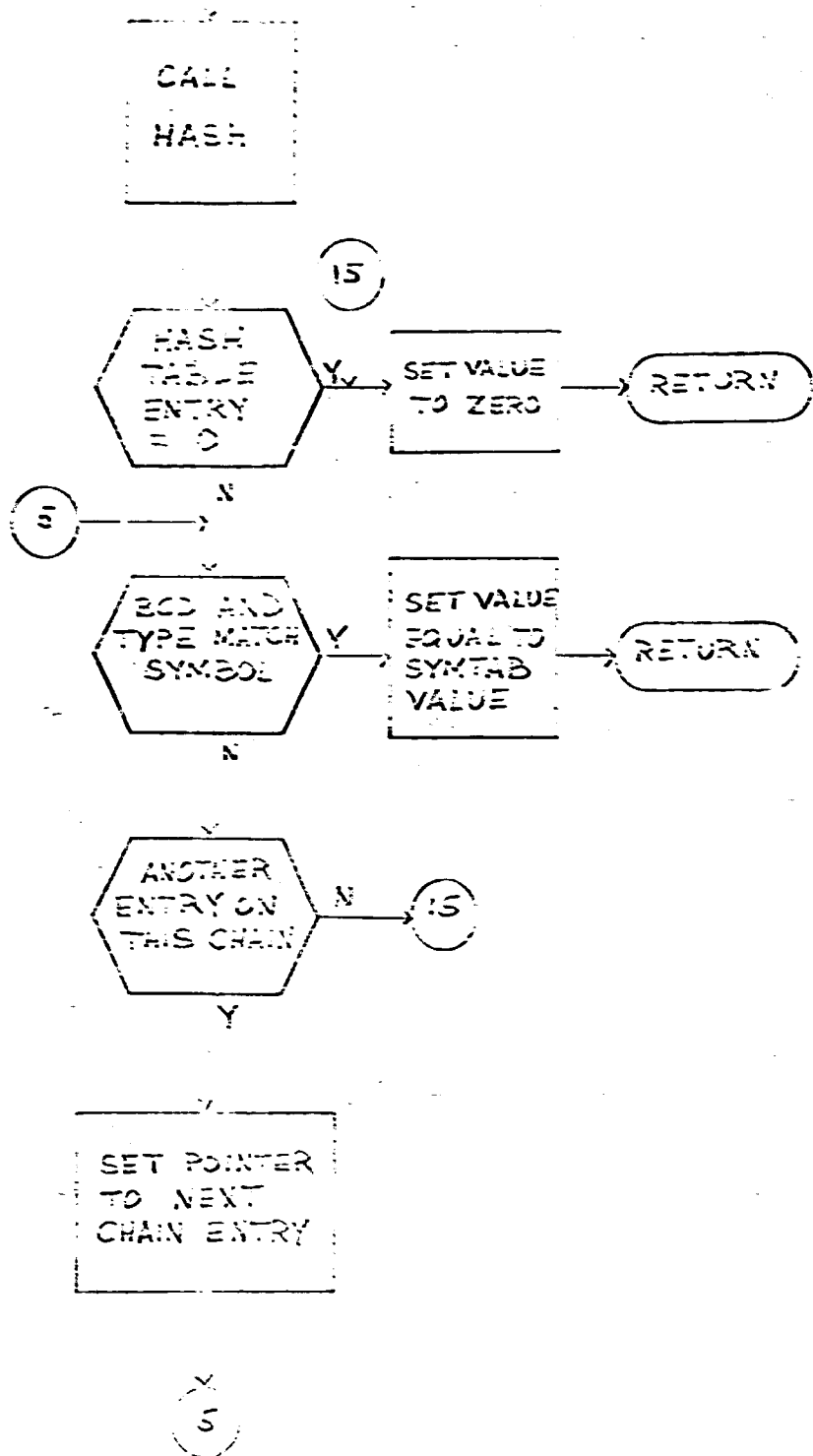
172



SUBROUTINE GGSYM

This routine is used to find the table value for a given global symbol in the global symbol dictionary. The global symbol is hashed into a value from 1 to 100 by calling the HASH routine. This table entry forms the head of a chain through the symbol table entries itself. This chain is searched for the symbol and type of the symbol being searched. When the symbol and type agree, the symbol value is then returned as the output of the subroutine.

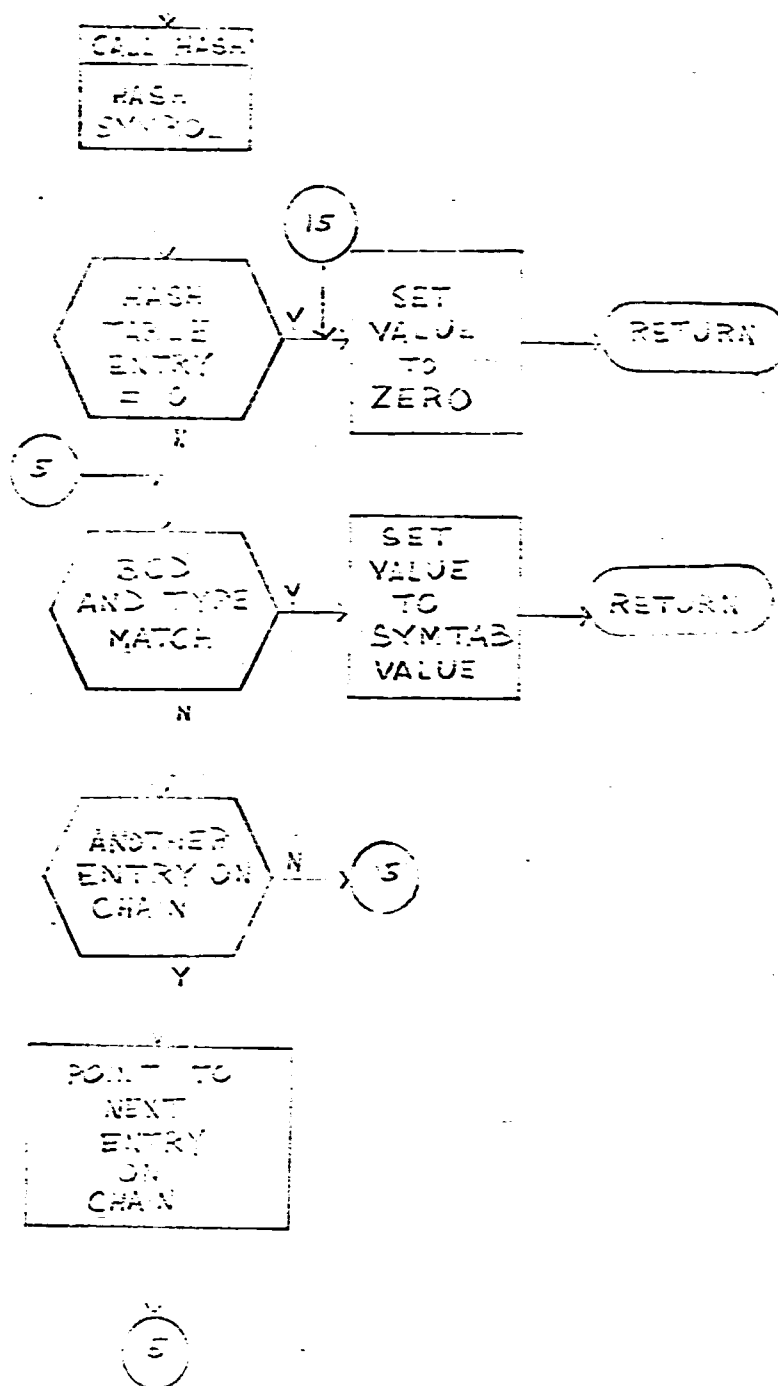
SUBROUTINE GGSYM
(GET FROM GLOBAL SYMTAB)



SUBROUTINE GLSYM

This routine is used to find the table value for a given local symbol in the local symbol dictionary. The local symbol is hashed into a value from 1 to 100 by calling the HASH routine. This table entry forms the head of a chain through the symbol table entries itself. This chain is searched for the symbol and type of the symbol being searched. When the symbol and type agree, the symbol value is then returned as the output of the subroutine.

SUBROUTINE GLSYM
(GET FROM LOCAL SYMTAB.)



SUBROUTINE HASH

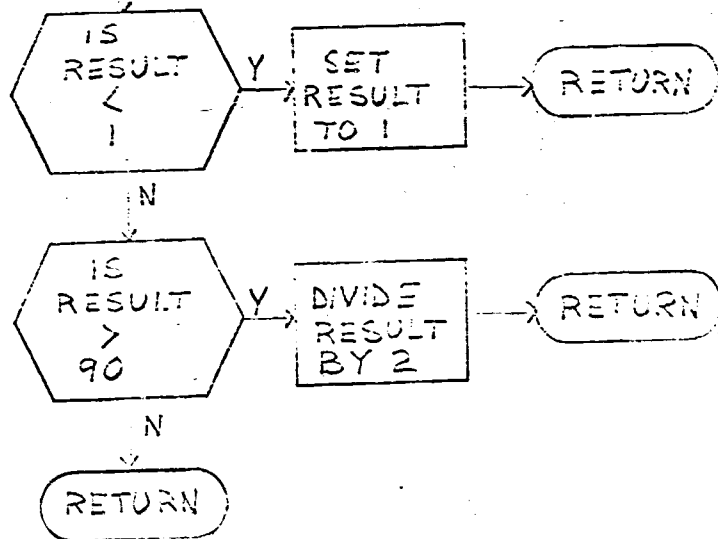
This routine hashes a name in the fix field output area into an integer in the range of 1 to 90. The number of pairs of characters in this symbol are computed. The pairs are then added together and their average value found. A shift and division are performed and a random number between the range of 1 and 90. This number is then returned as the hash value of the symbol.

SUBROUTINE HASH
(HASHEE SYMBOLS)

↓
COMPUTE
THE NUMBER
OF PAIRS OF
CHARACTERS IN
THE SYMBOL

↓
ADD THE
PAIRS
TOGETHER

↓
DIVIDE BY THE
NUMBER OF PAIRS,
SUBTRACT 370,
AND DIVIDE BY
17



SUBROUTINE NERR

This routine is the normal error handler. It accepts as input an error number, a statement label to which it will return, and two words which are to be included in the error message when it is finally printed out by the error pass. This routine writes the error to the current error file and turns on the master error switch.

184

SUBROUTINE NERR
(NORMAL ERROR HANDLER)

WRITE ERROR
NUMBER, STATE-
MENT NUMBER
AND TWO MES-
SAGE WORDS

TURN ON
ERROR
SWITCH

RETURN

SUBROUTINE PAGE

This routine is called whenever it is desired to move to a new page of printout. A title is printed at the top of each page. A count is maintained of the number of pages printed out. This page number also appears on the same line as the title. The routine resets the line counter to a maximum of 46 lines per page.

100

SUBROUTINE PAGE
(PRINT PAGE HEADING)

ADD 1
TO
IPGCT

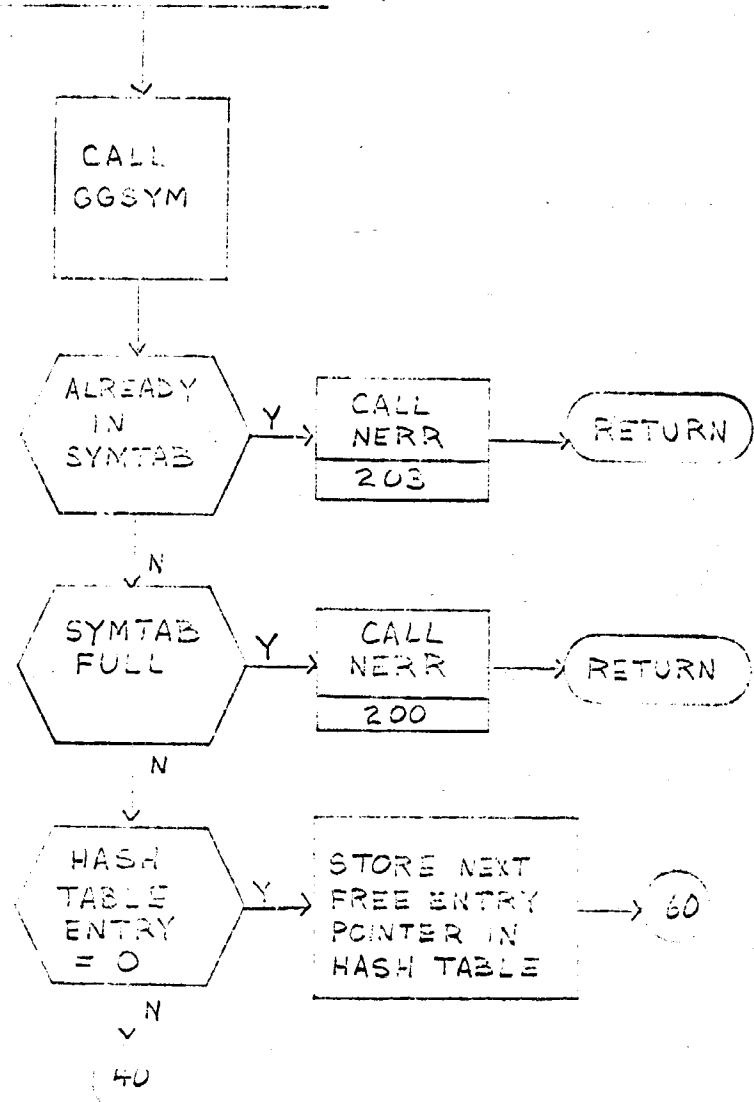
PRINT TITLE,
"PAGE" AND
IPGCT

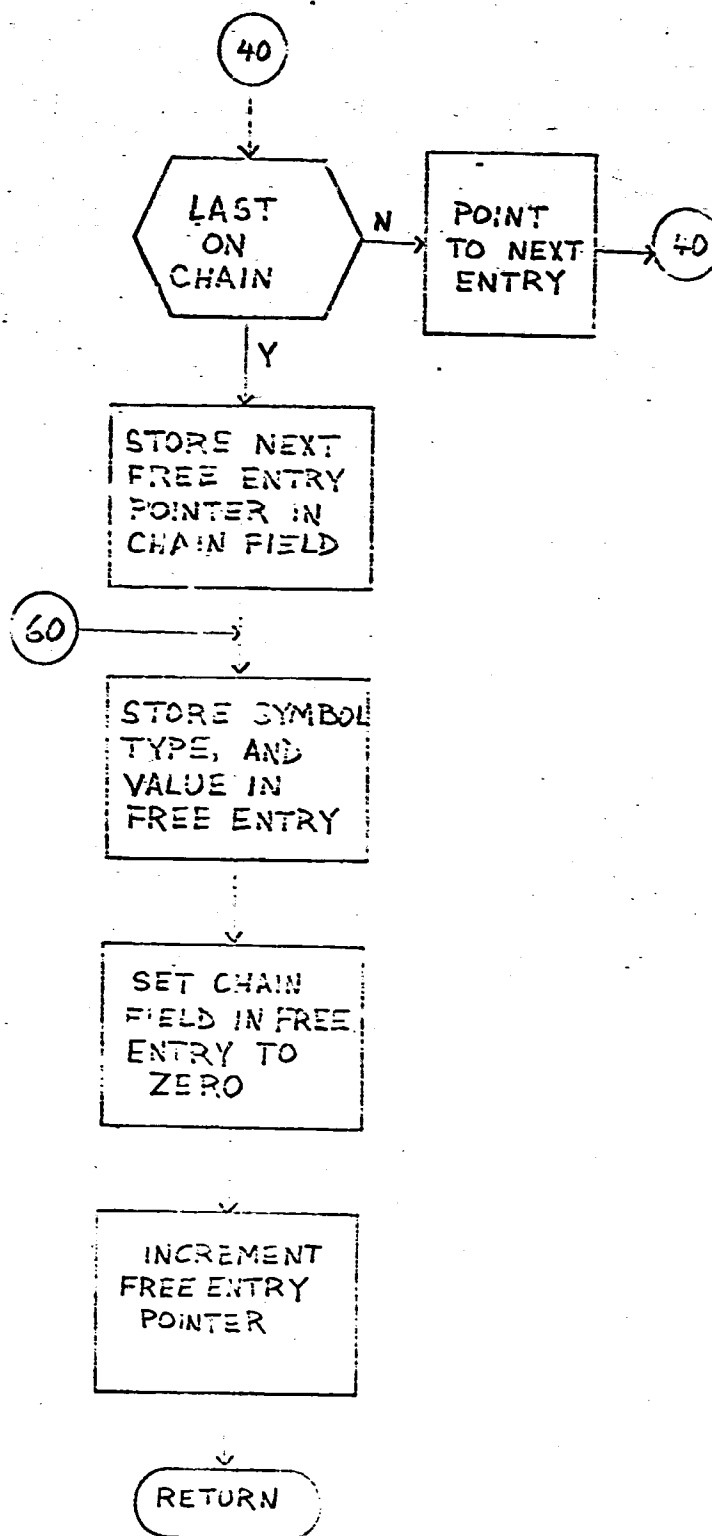
SET
ILNCT
TO 46

RETURN

SUBROUTINE PGSYM

This routine places symbols into the global symbol table. Before placing any symbol into the global symbol table, an attempt is made to see if that symbol is already there, avoiding duplicate entries. If the entry is not in the symbol table, then the hash table entry for that symbol is consulted and the chain followed looking for the last entry on the chain. When the last entry is found, the new symbol is appended to the chain.

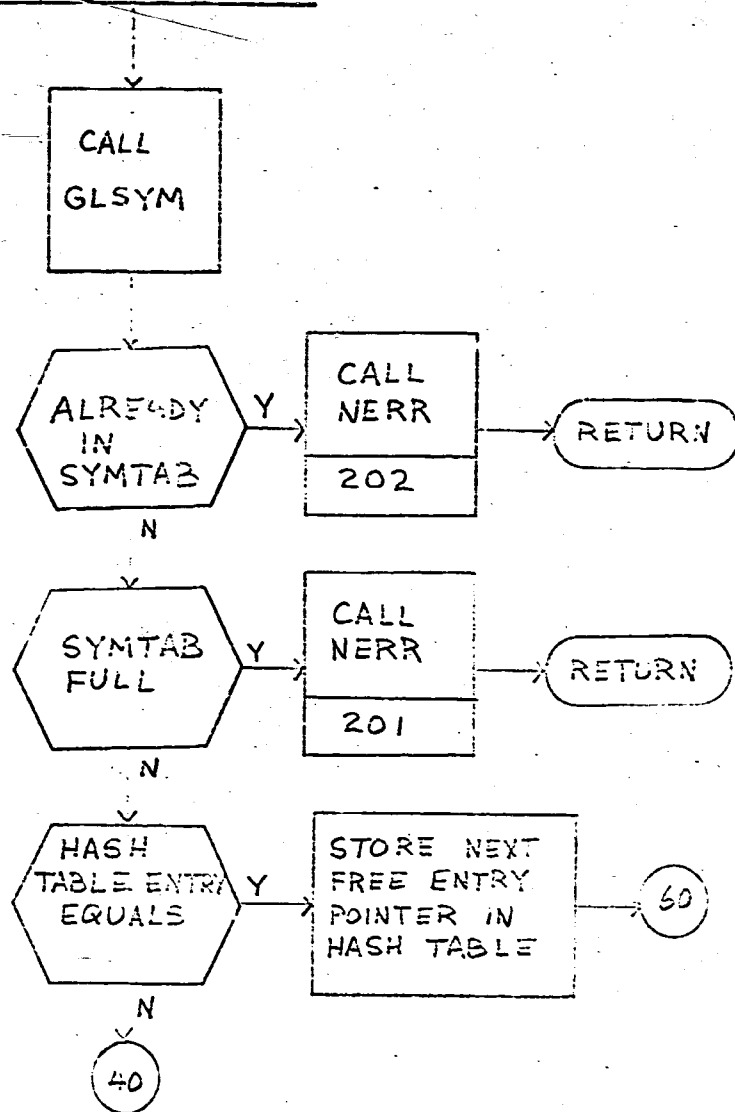
SUBROUTINE PGSYM
(PUT IN GLOBAL SYMTAB)

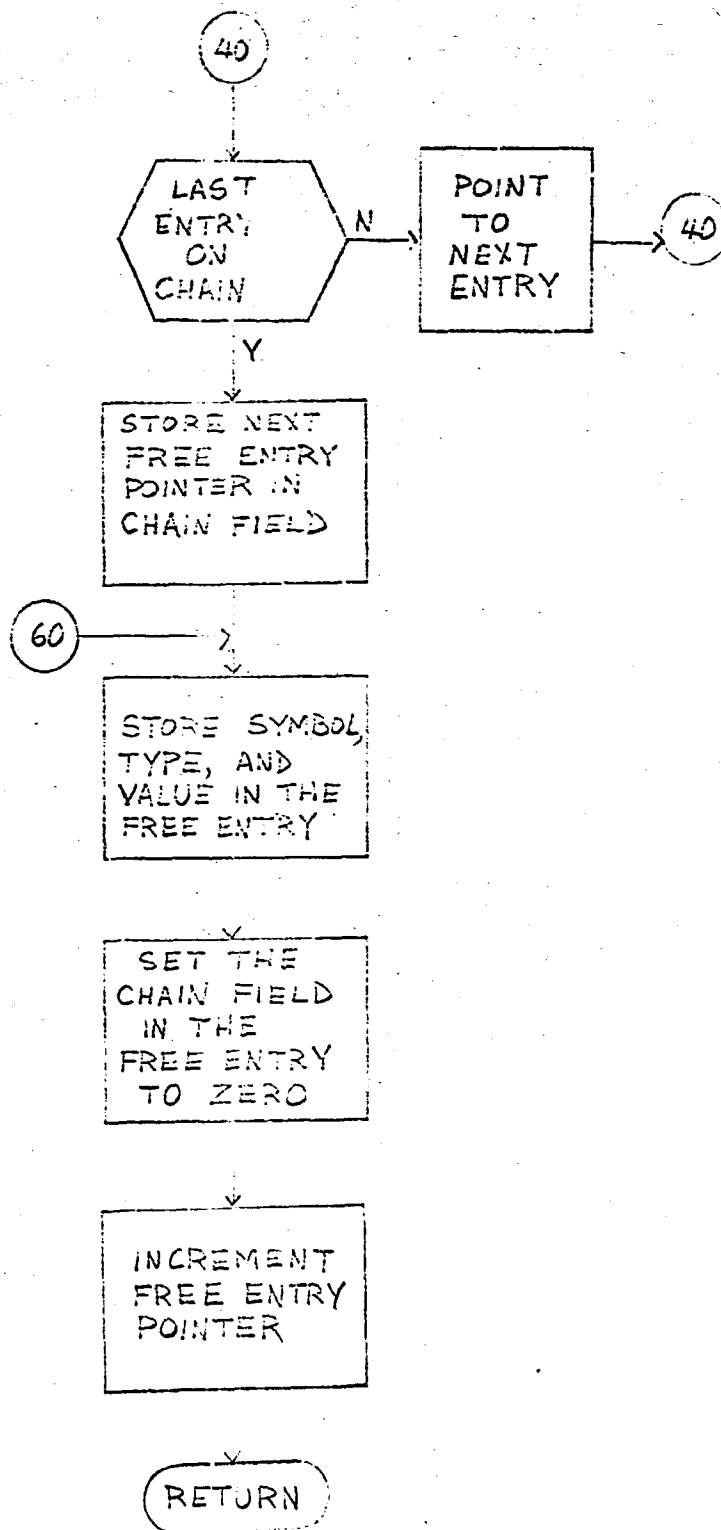


SUBROUTINE PLSYM

This routine places symbols into the local symbol table. Before placing any symbol into the local symbol table, an attempt is made to see if that symbol is already there, avoiding duplicate entries. If the entry is not in the symbol table, then the hash table entry for that symbol is consulted and the chain followed looking for the last entry on the chain. When the last entry is found, the new symbol is appended to the chain.

SUBROUTINE PLSYM
(PUT IN LOCAL SYMTAB)



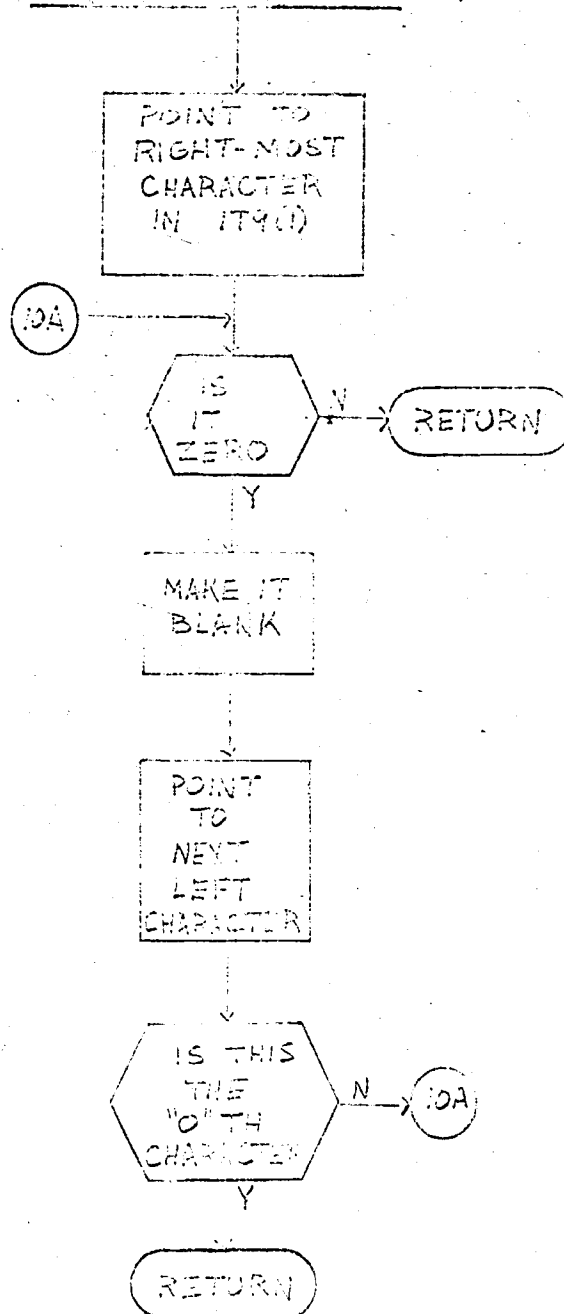


SUBROUTINE RBLK

This routine changes right hand zeroes to blanks in the first word of the input area used to read from the library. This padding with blanks is performed prior to searching the library for the given entry point.

194

SUBROUTINE RBLK
(RIGHT FILL WITH BLANKS)

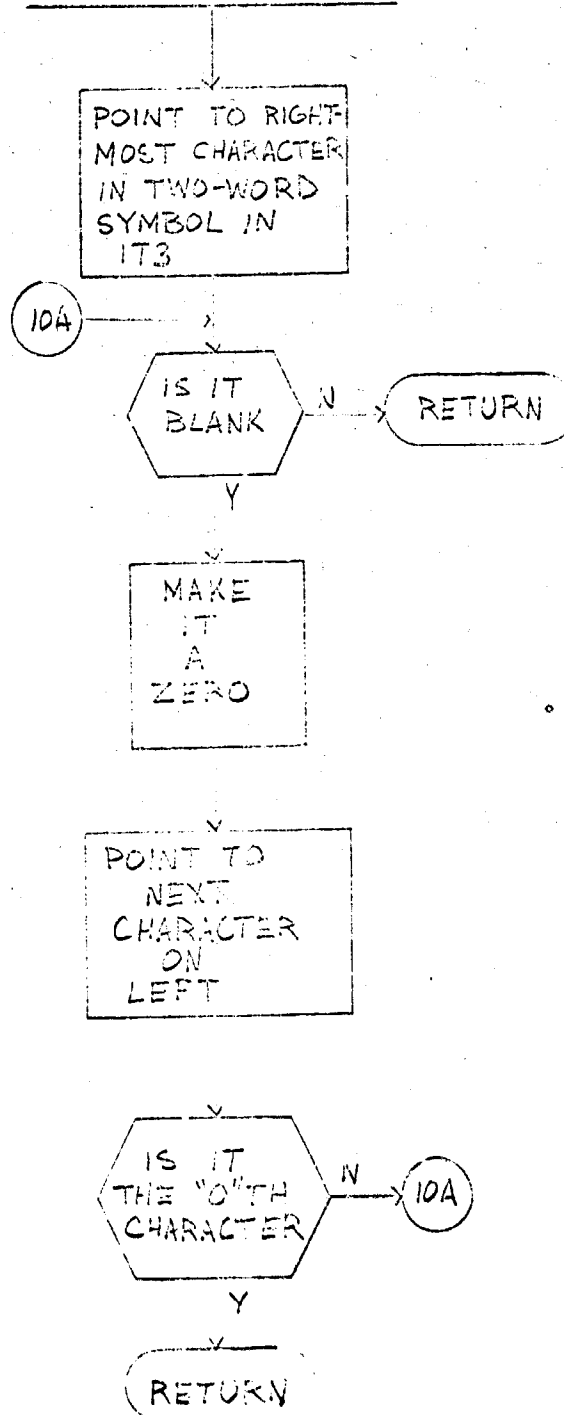


195

SUBROUTINE RZRO

This routine is used to right-fill a two word
symbol in the fix field output area with zeroes.

SUBROUTINE RZRO
(RIGHT FILL WITH ZEROS)



SUBROUTINE SA

This subroutine initializes phase 2 by filling common with zeroes, filling the op-code table with all 'Z's, filling the title with blanks, setting constants, and storing global symbols.

SUBROUTINE SA
(INITIALIZE PHASE 2)

FILL
COMMON
WITH
ZEROS

FILL
OP-CODE TABLE
WITH ALL
'ZZZZZZ'

FILL
TITLE
WITH
BLANKS

INITIALIZE
INDIVIDUAL
CONSTANTS
IN COMMON

STORE
GLOBAL
SYMBOLS

RETURN

SUBROUTINE SA1

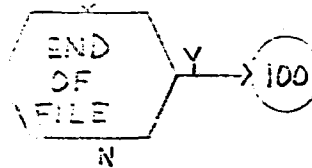
This routine reads input from the macro input file in fixed fields format. It also reconstructs the statement in preparation for printing out. Upon reaching the end of the macro input file it sets an end of file indicator.

200

SUBROUTINE SAI
(READ MACRO INPUT)

FILL FIX
FIELDS ROUTINE
OUTPUT AREA
WITH ZEROS

READ MACRO
INPUT INTO FIX
FIELDS ROUTINE
OUTPUT AREA



INCREMENT
STATEMENT
NUMBER

1A

201

1A

BLANK
NORMAL
INPUT
AREA



SET POINTER
TO 1ST SYMBOL
OF MACRO
INPUT

SET NEXT OUTPUT
POSITION TO 1ST
WORD OF NORMAL
INPUT AREA

30A

200

30A

GET
SYMBOL
LENGTH

LENGTH
EQUALS
ZERO

Y

60

N

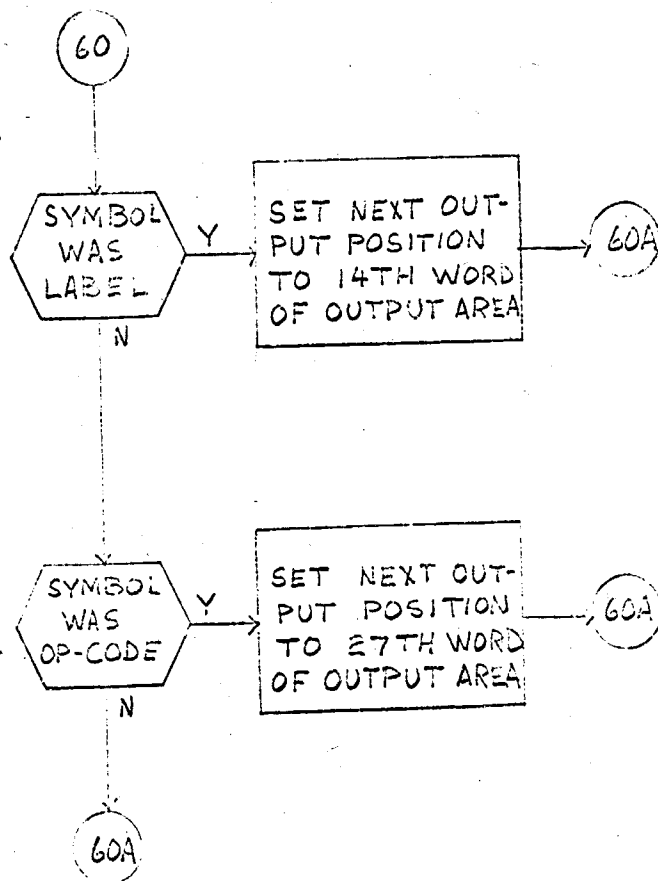
MOVE NEXT
CHARACTER OF
SYMBOL TO NEXT
OUTPUT WORD

ANOTHER
CHARACTER
IN
SYMBOL

Y

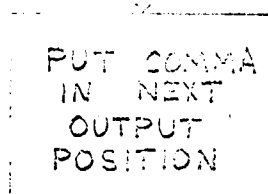
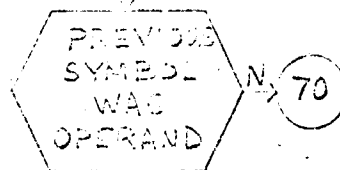
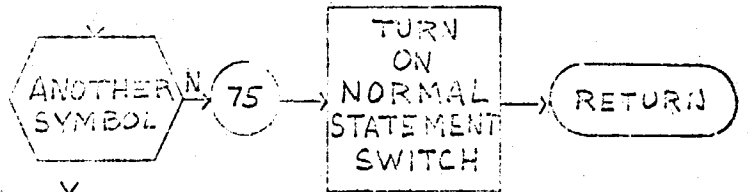
N

60

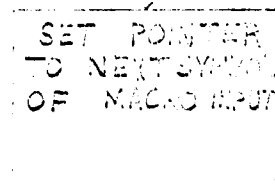


200

60A



70



30A

205

80

SET POINTER
TO FIRST
CHARACTER OF
COMMENT

SET POINTER
TO FIRST
WORD OF NORMAL
INPUT AREA

MOVE NEXT CHAR-
ACTER OF COM-
MENT TO NEXT
WORD OF NORMAL
INPUT AREA

ALL CHAR-
ACTERS OF
COMMENT
MOVED

N

Y

TURN
ON
COMMENT
SWITCH

RETURN

100

SET
MACRO
EOF
SWITCH

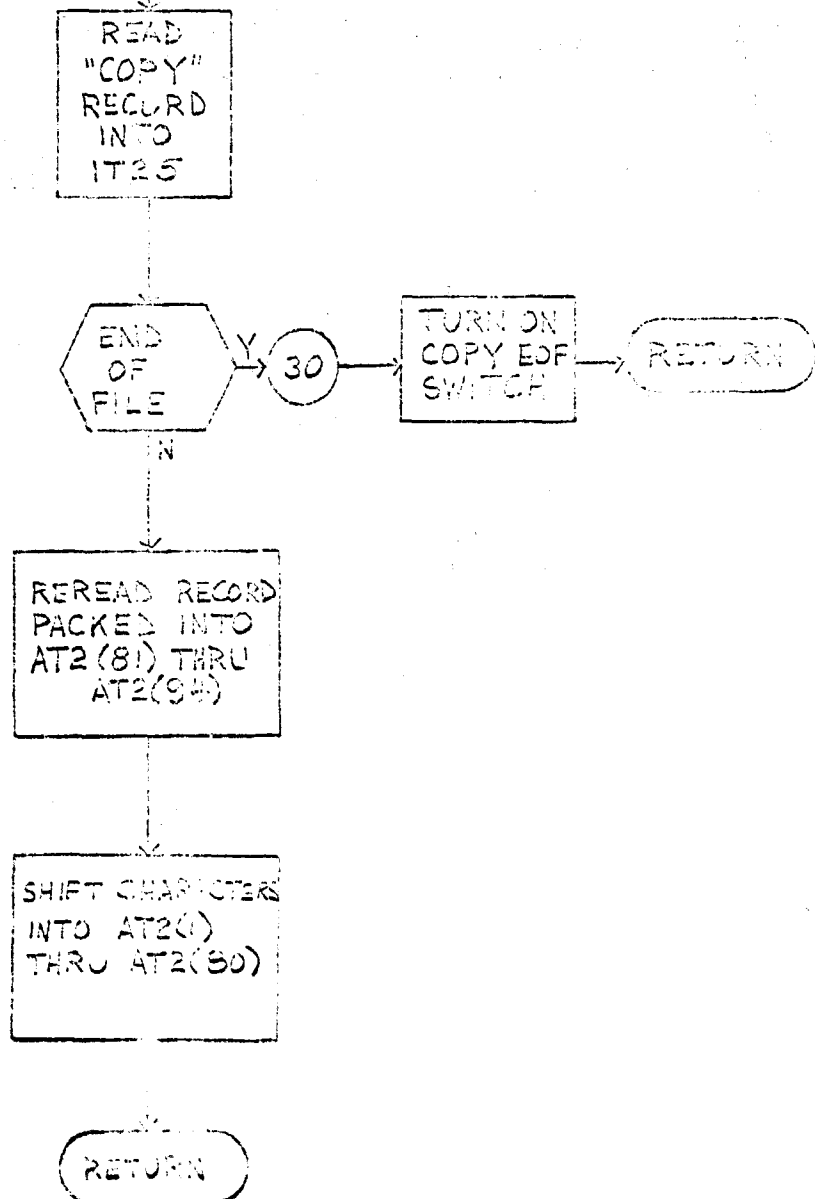
RETURN

SUBROUTINE SA2

This routine reads source statements that have been placed on the copy input file following execution of a copy statement. Each call on this subroutine causes one record to be read from the copy input file. At the end of the file, an end of file switch is set.

207

SUBROUTINE SA2
(READ COPY-INPUT)



SUBROUTINE SA3

This routine reads source statements from the standard input file. Each call on this routine causes one record to be read from the standard input file. The input is stored in both packed and unpacked forms for further processing. When the end of file is reached, an end of file switch is set.

200

SUBROUTINE SA3
(READ PRIMARY INPUT)

READ
PRIMARY
INPUT RECORD
INTO
IT23

END
OF
FILE N

TURN ON
PRIMARY EOF
SWITCH

RECORD RECORD
PACKED INTO
AT2(0), TERO
AT2(04)

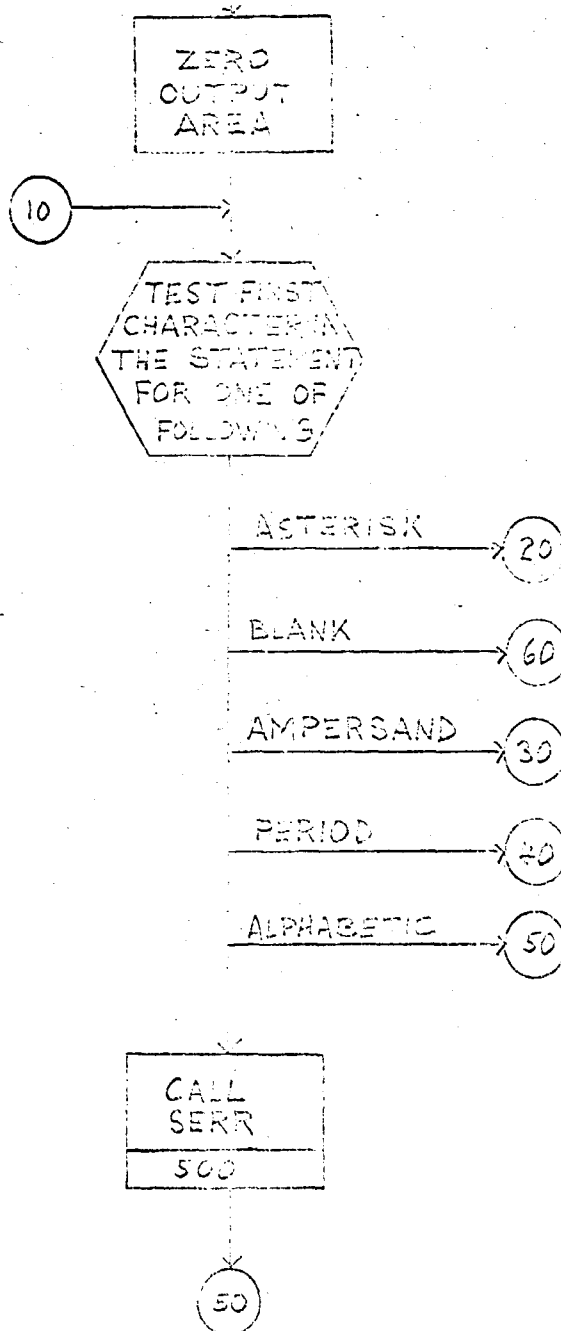
SHIFT
CHARACTERS
INTO
AT2(0), TERO
AT2(04)

RETURN

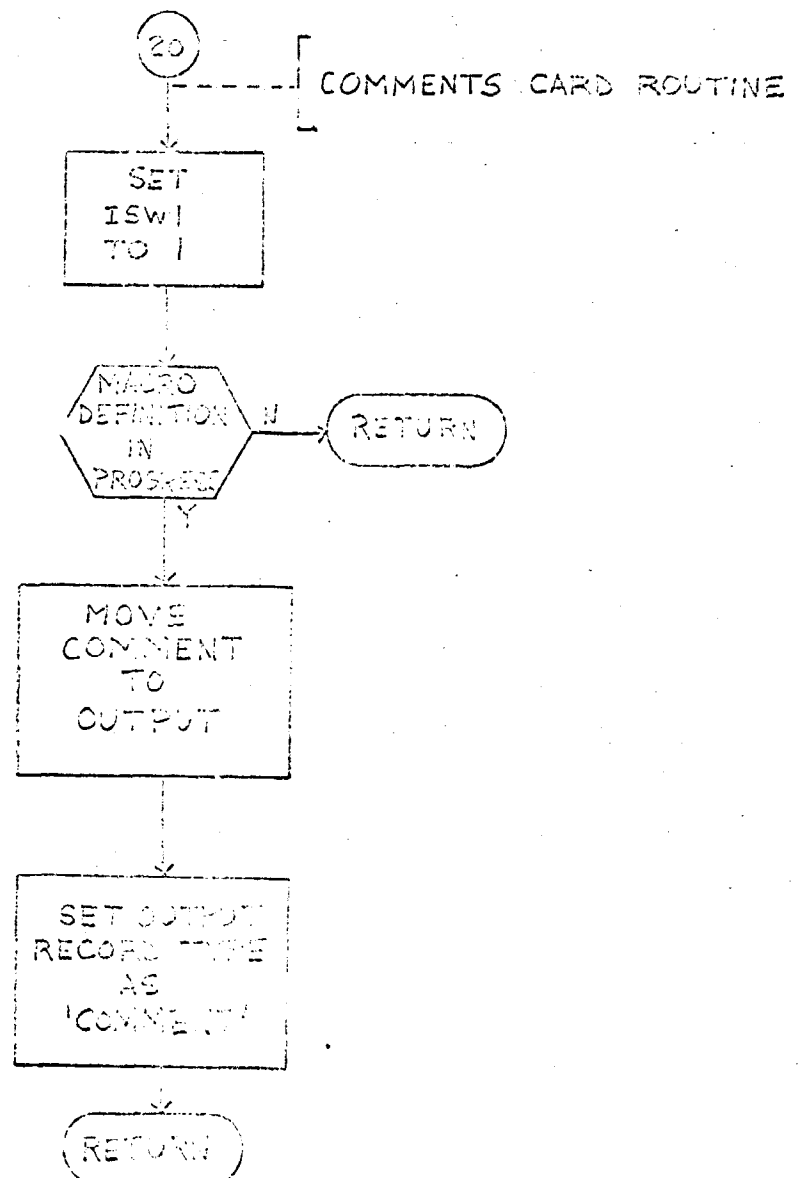
SUBROUTINE SA4

This routine transforms the free field format input into a fixed field form used for processing by all other routines in the first pass. Within this subroutine, there are separate routines for processing comment cards, macro definitions, special op-codes, normal op-codes, continuation cards, symbols followed by asterisks, symbols preceded by ampersands, and statement label variables. This routine has a secondary entry point, SA5, which is called whenever a continuation card is to be transformed into fixed field output form.

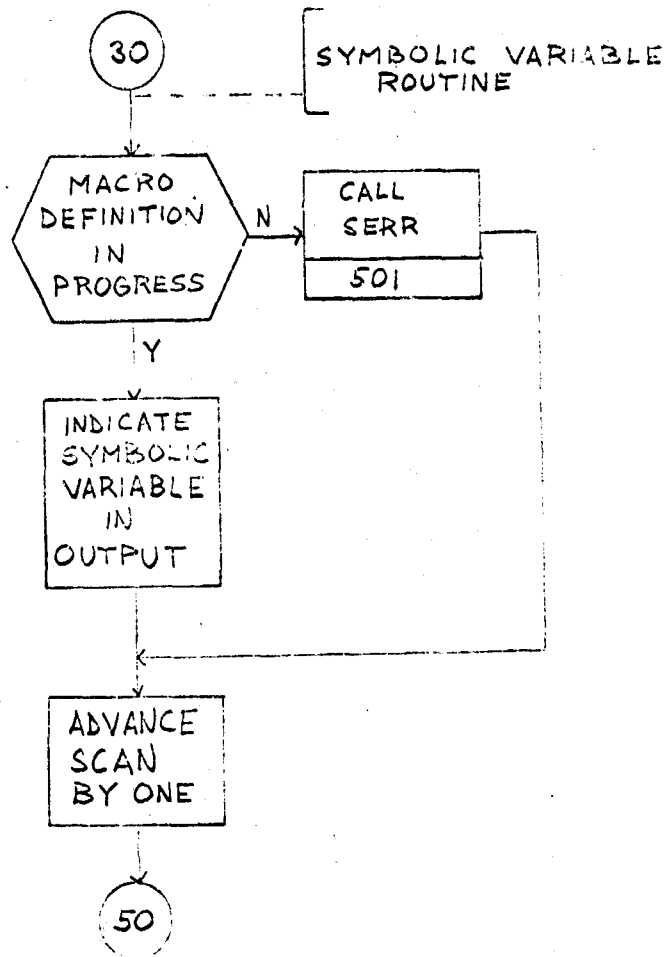
SUBROUTINE 3A4
(FIX FIELDS ROUTINE)

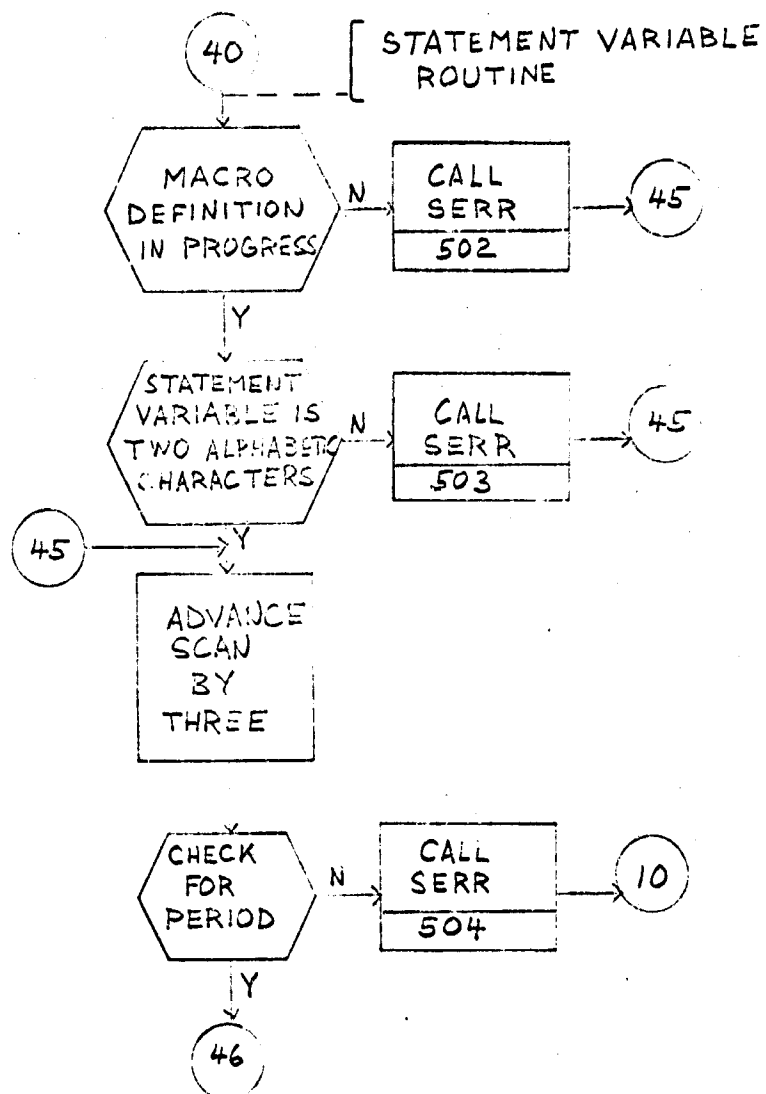


713.2

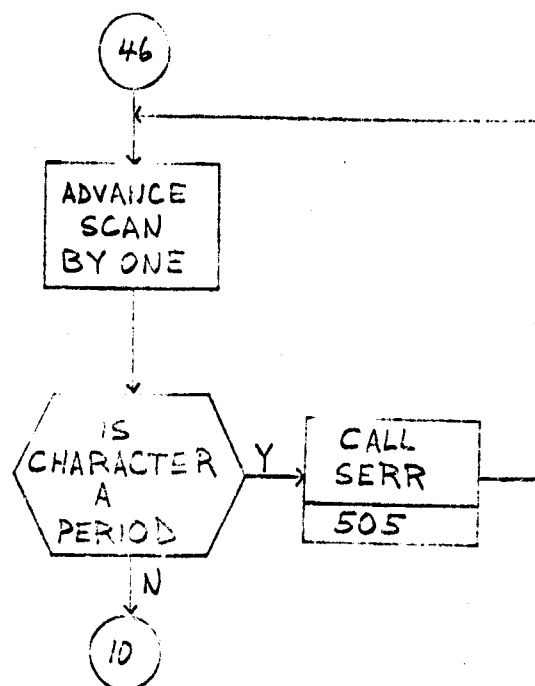


212

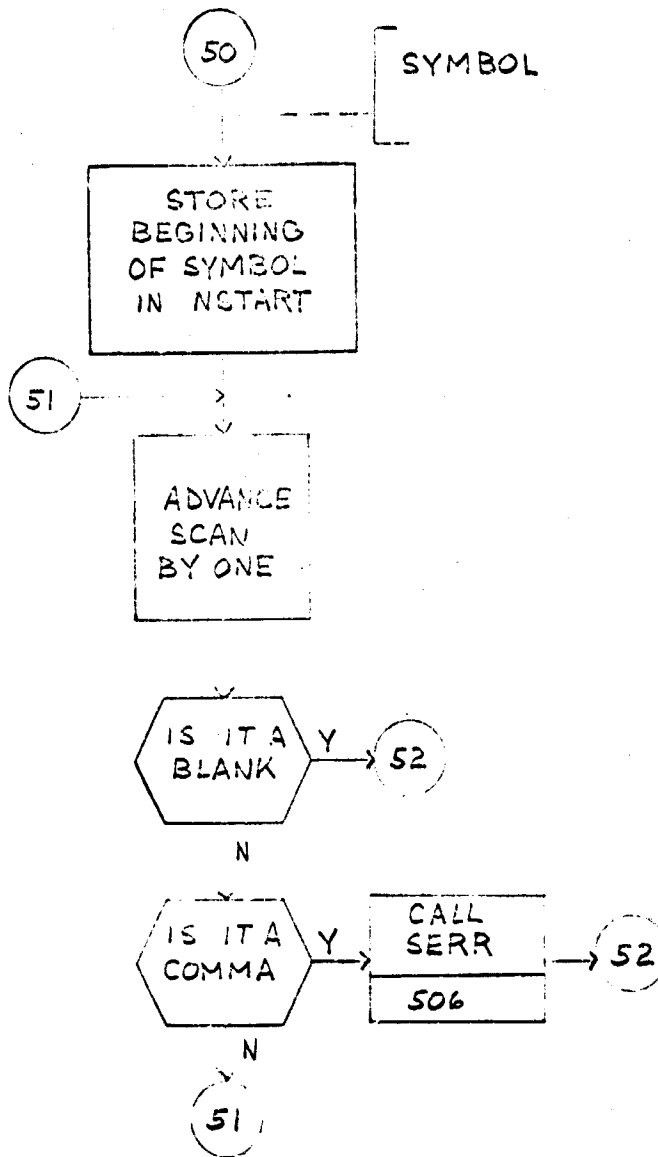




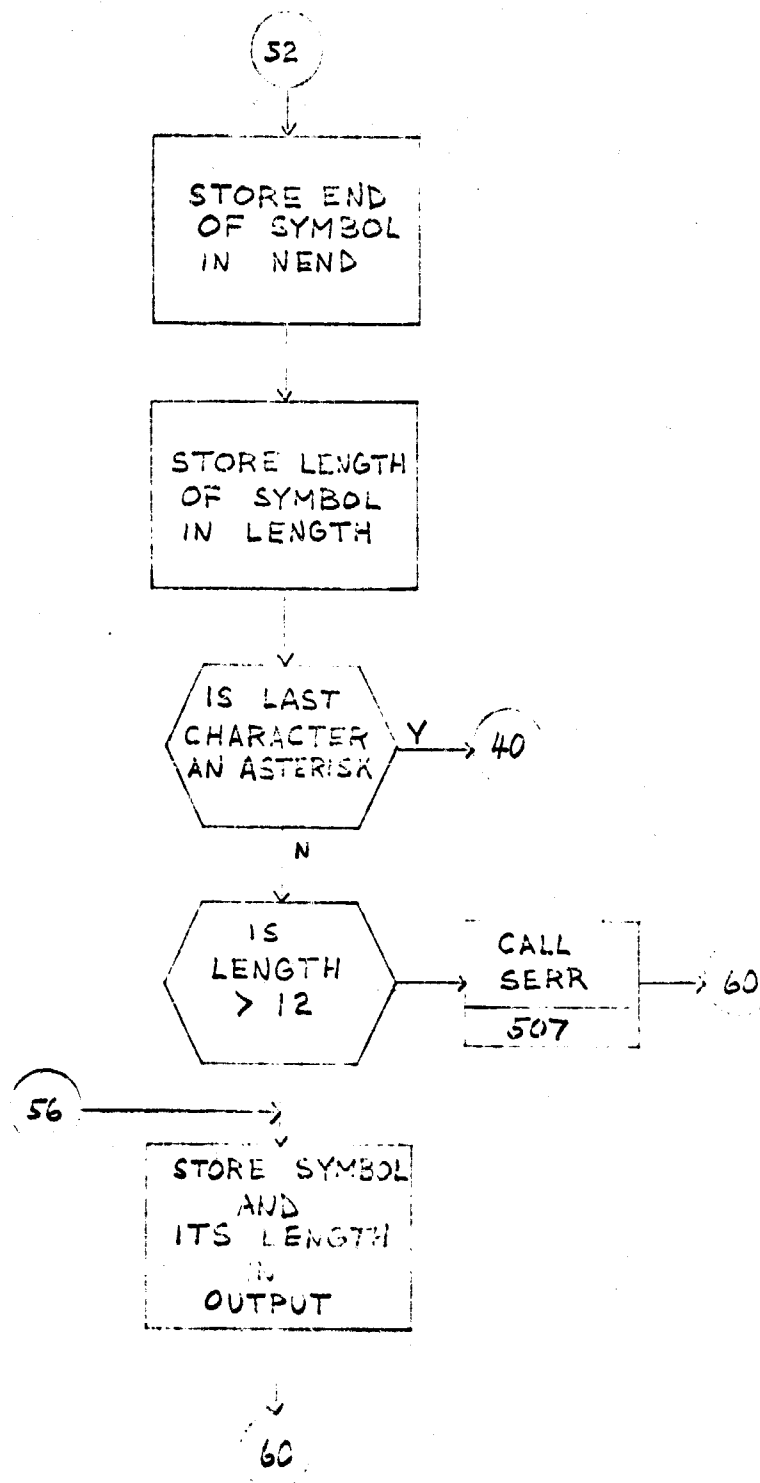
215



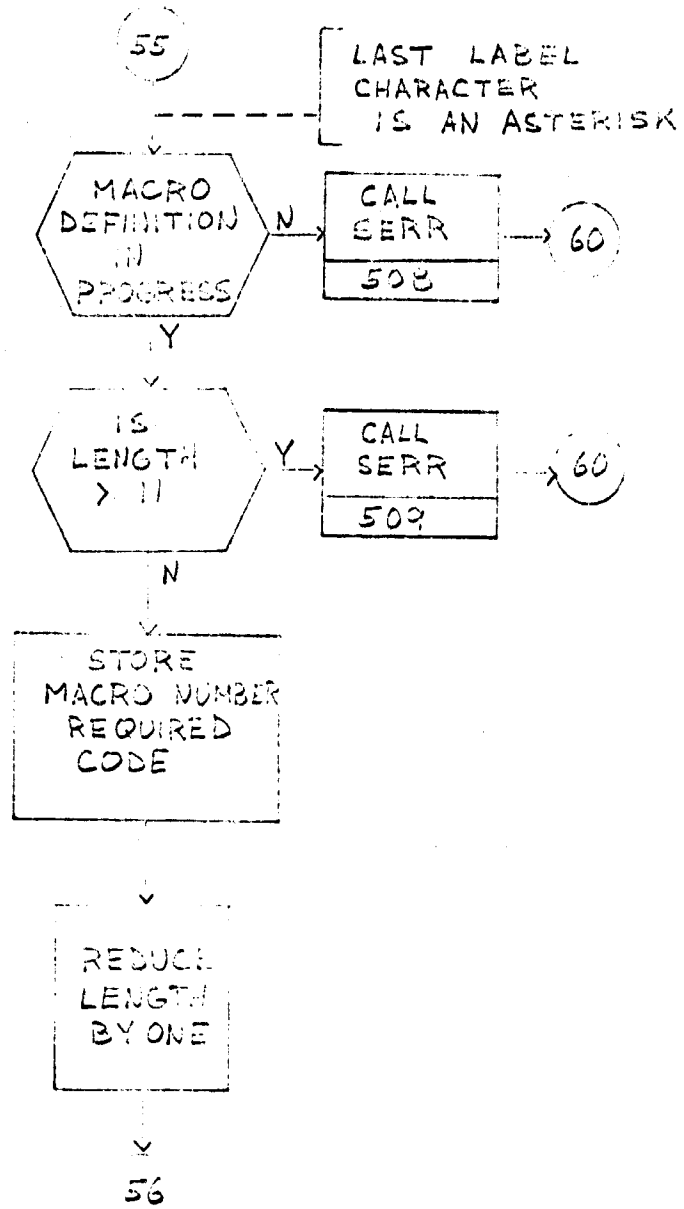
216

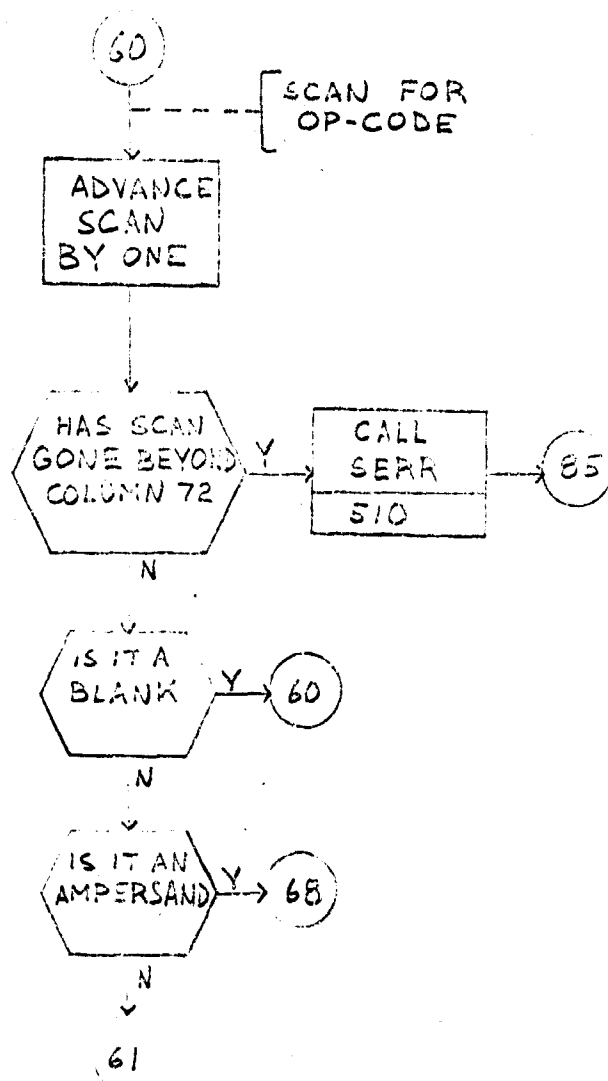


217

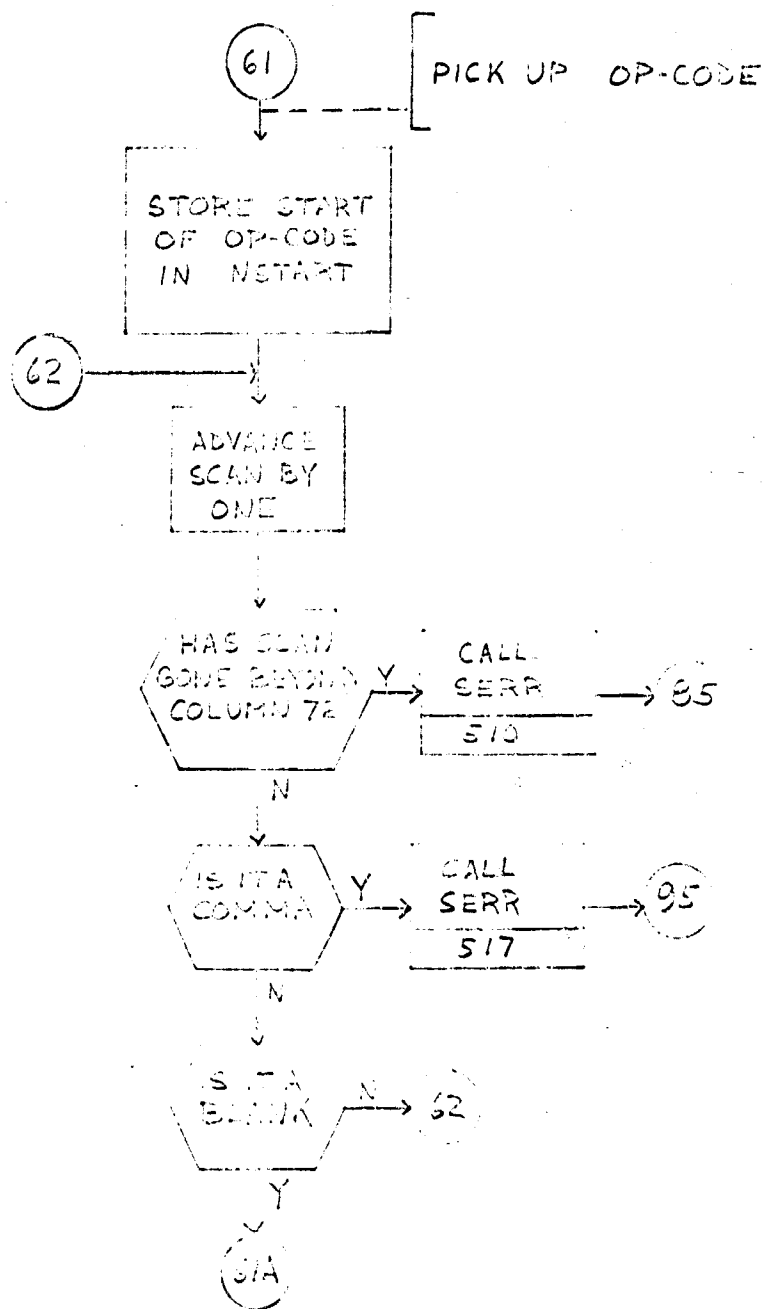


218

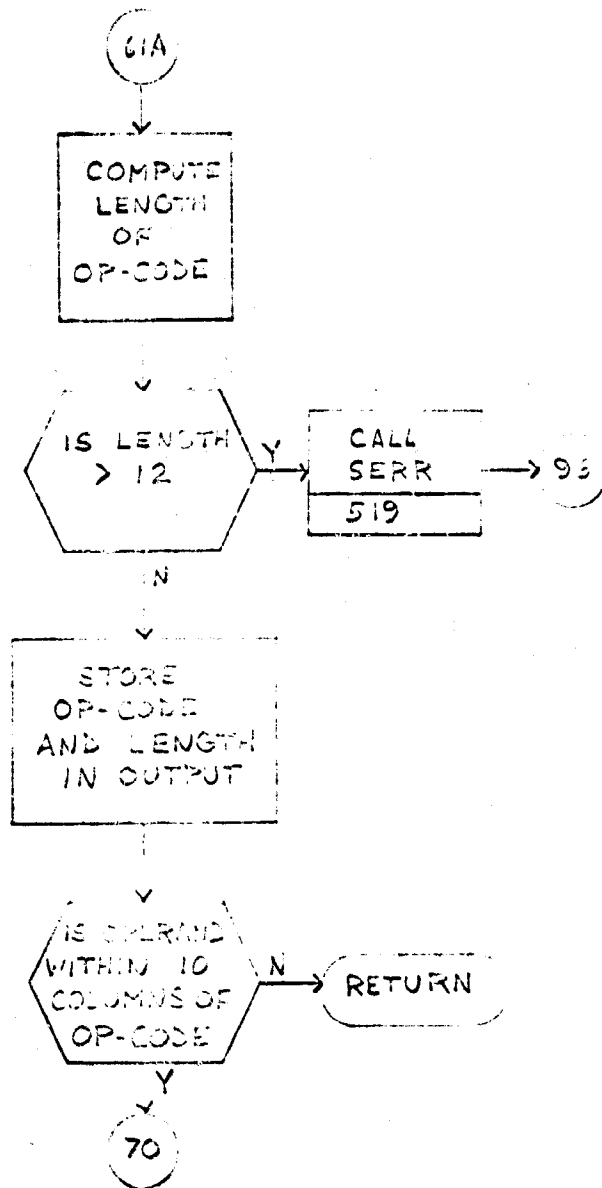




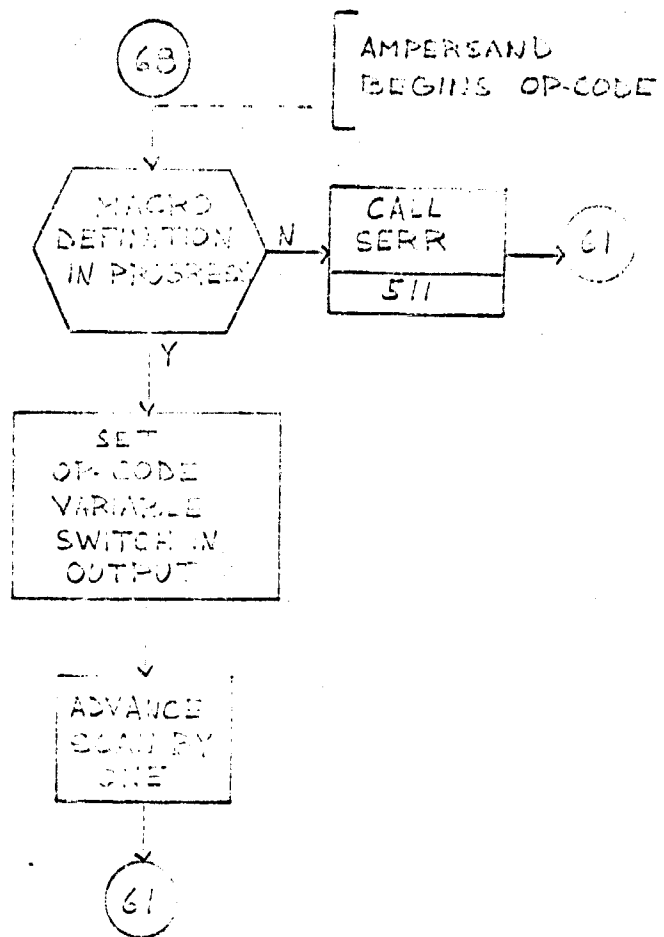
220



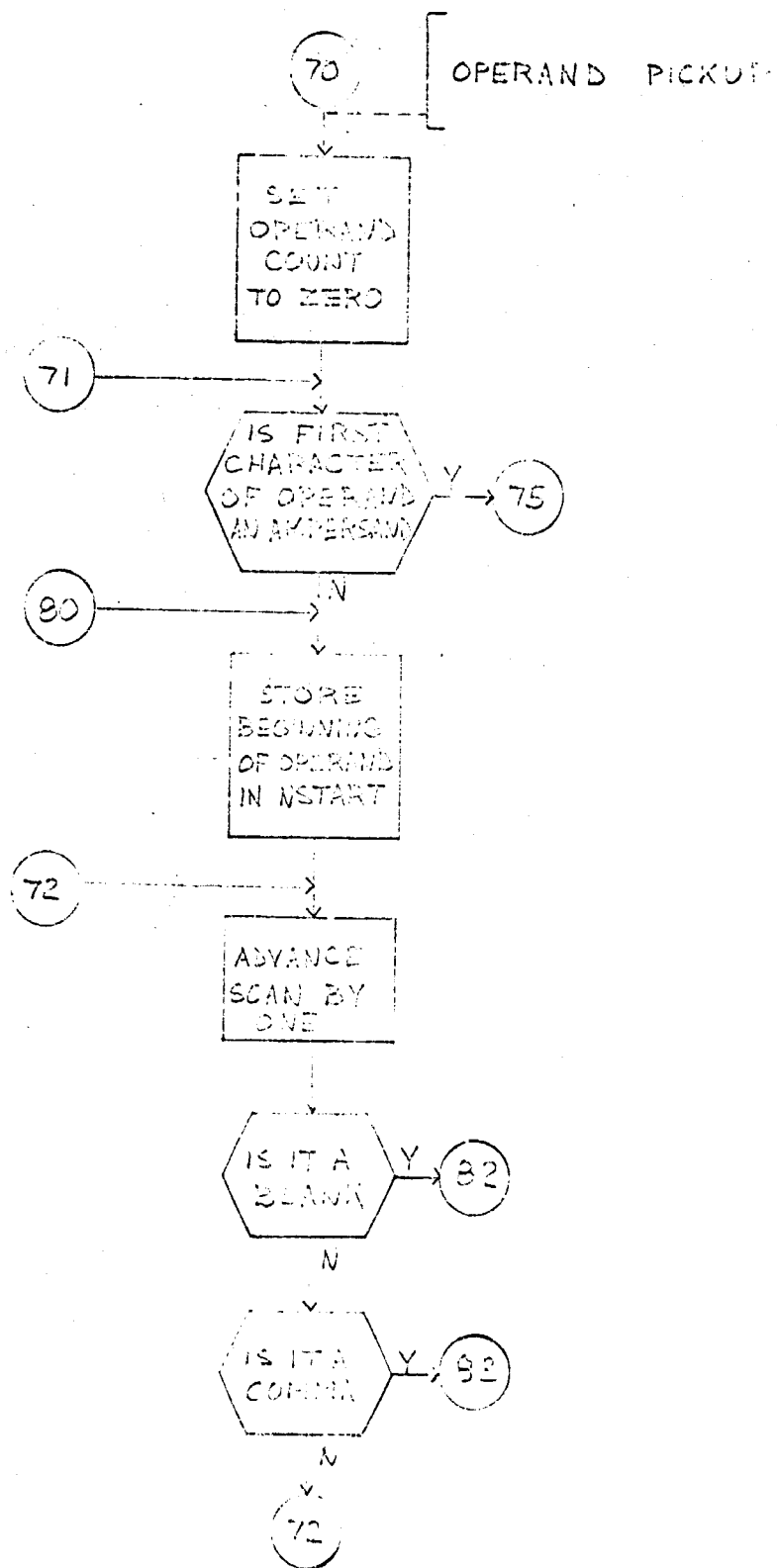
271



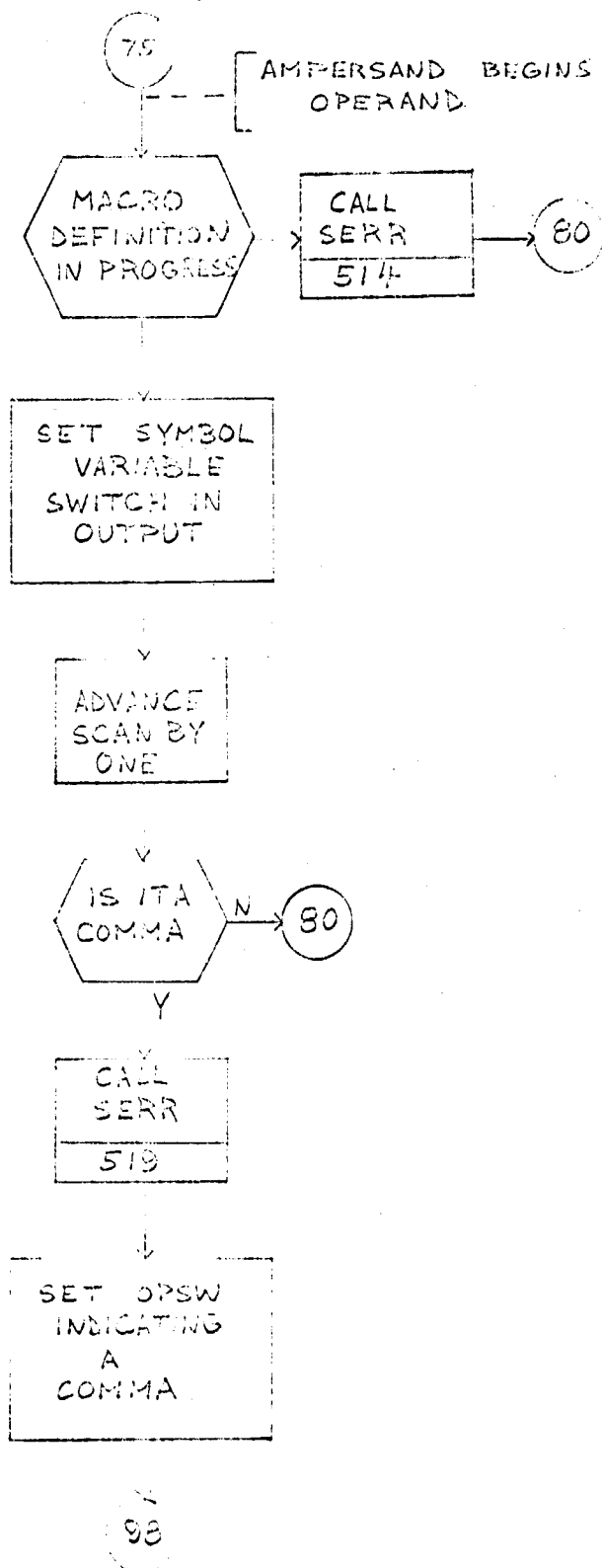
222

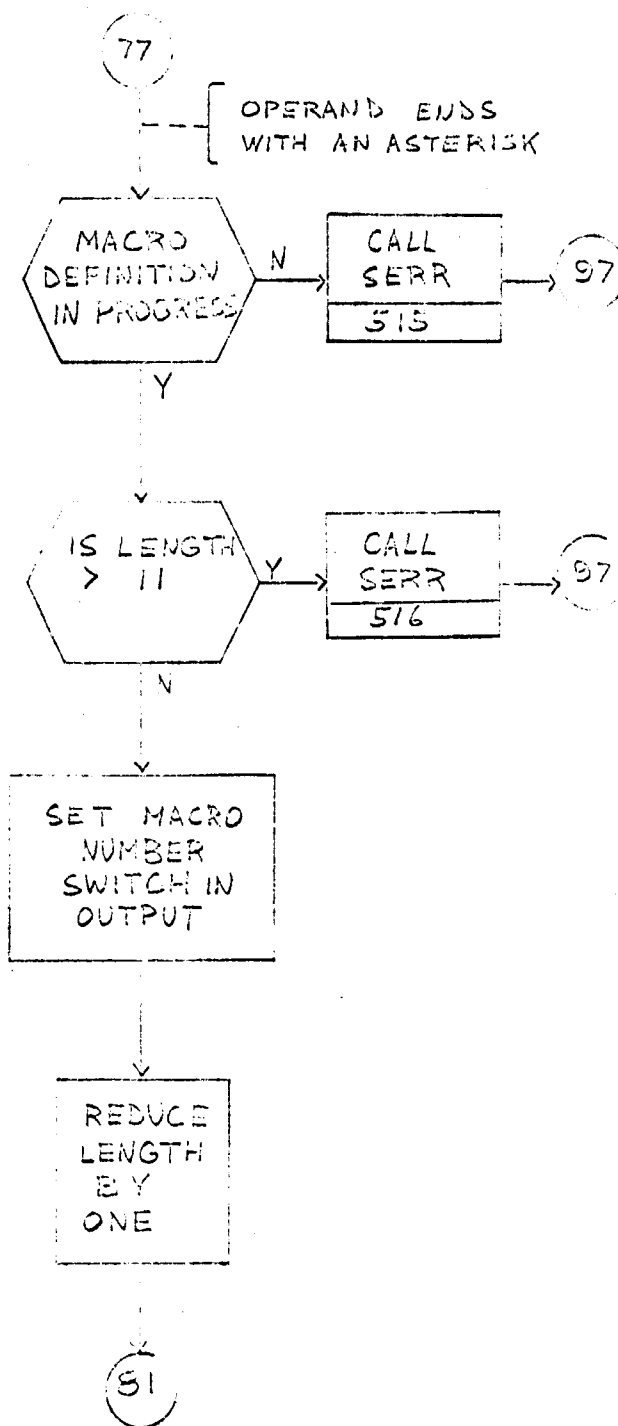


223

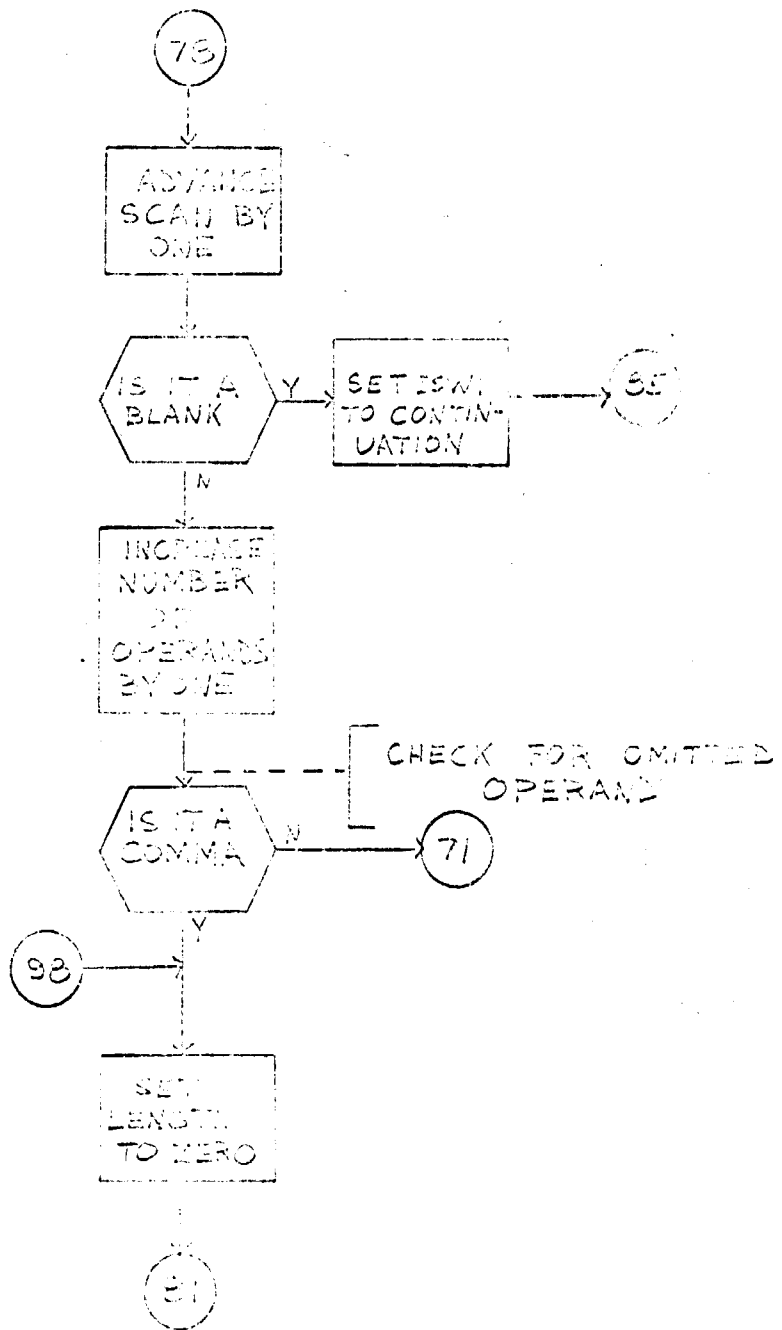


876

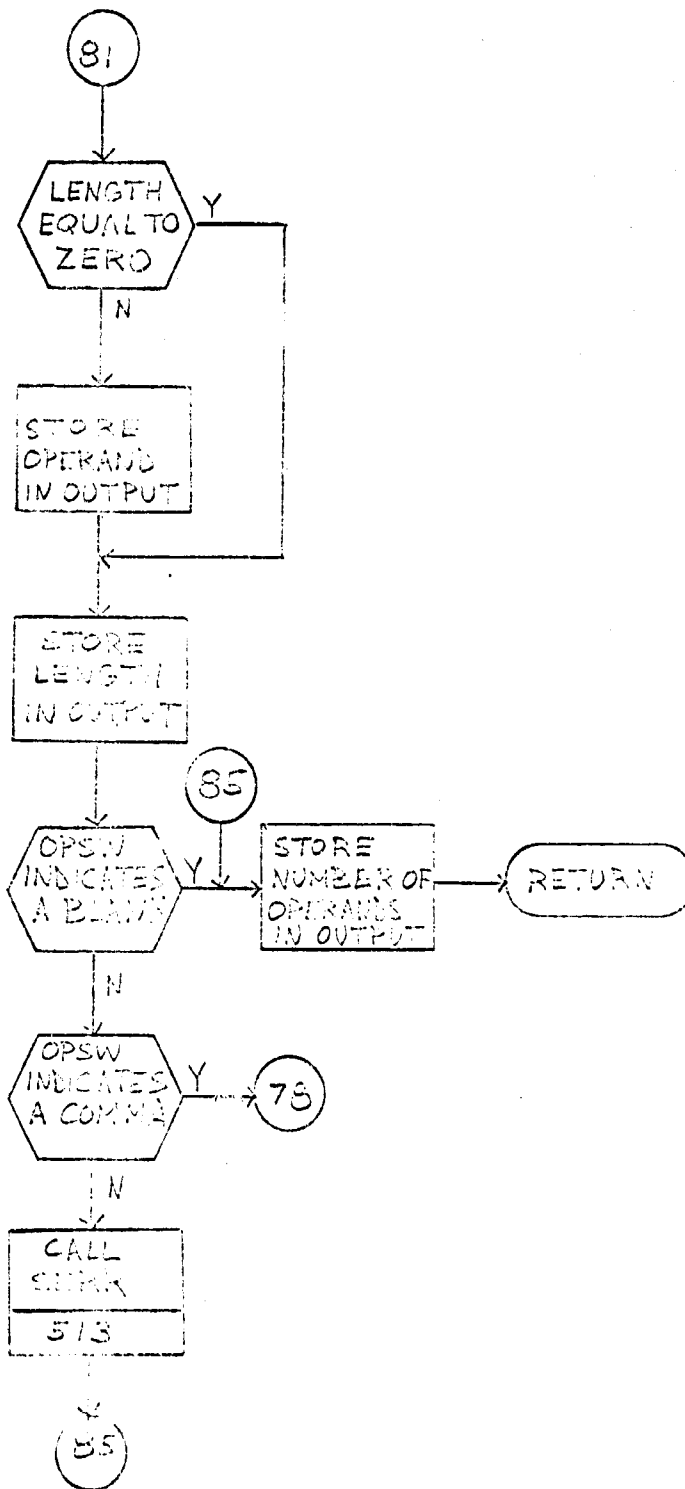




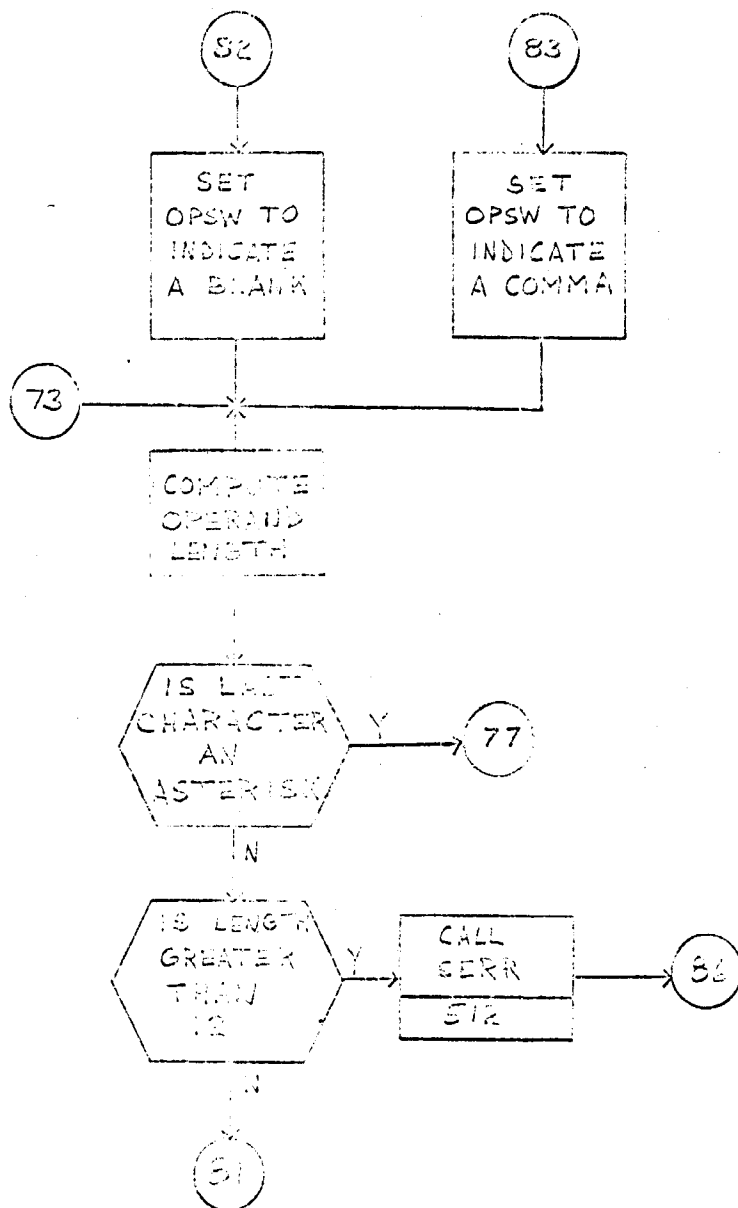
223



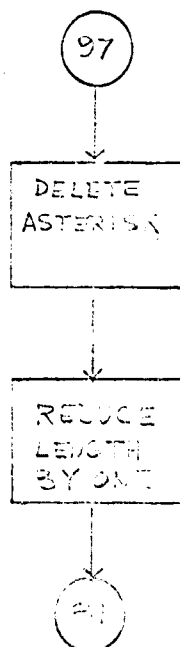
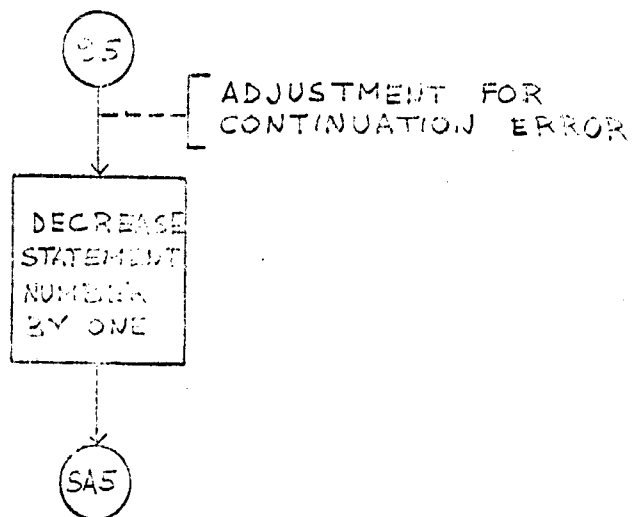
222



273



279



280

ENTRY SA5
(CONTINUATION ENTRY)

RESET
OUTPUT
SWITCH
TO
NORMAL

FIRST
COLUMN IS
BLANK

N

CALL
SERR
518

35

Y

INCREASE
THE NUMBER
OF OPERANDS
BY ONE

GO
TO
NEXT
OPERAND

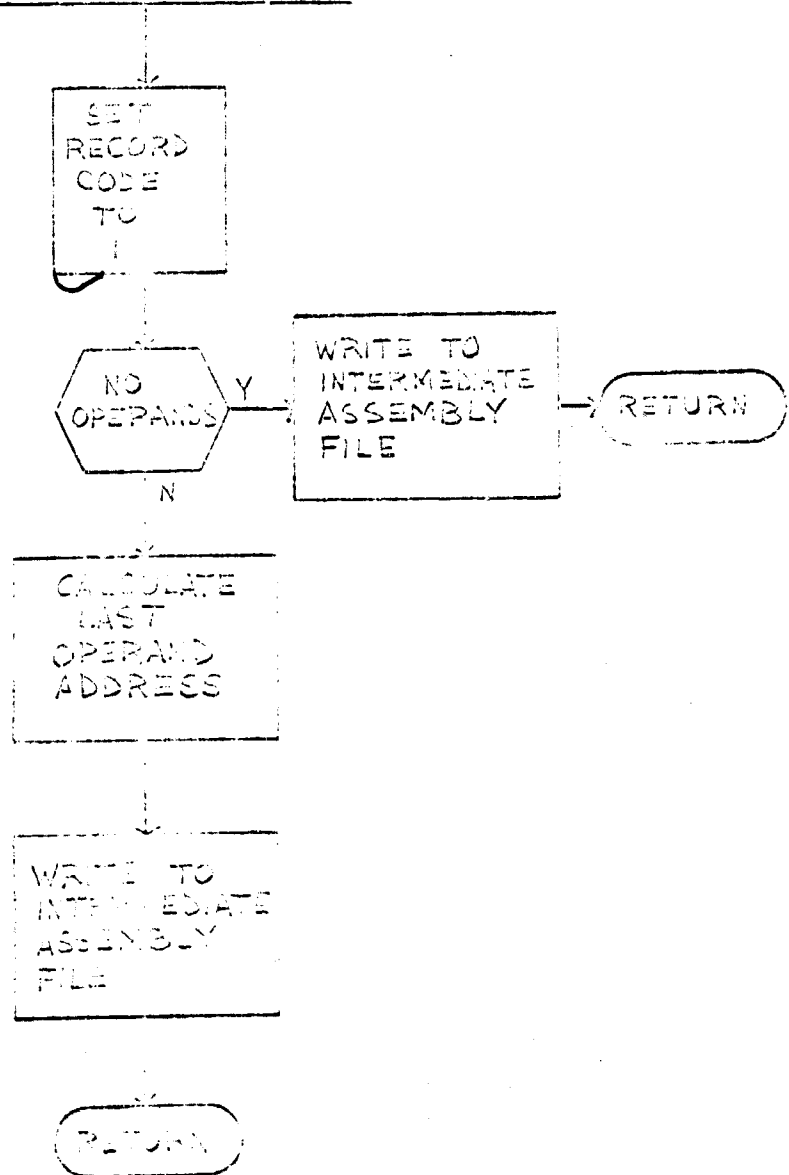
71

SUBROUTINE SA6

This routine is used to write records to the intermediate assembly file. It is used by the ASML routine. The first word of each record contains a count of the number of words in the record. The records are therefore of variable length.

722

SUBROUTINE SAG
(INTERMEDIATE OUTPUT)

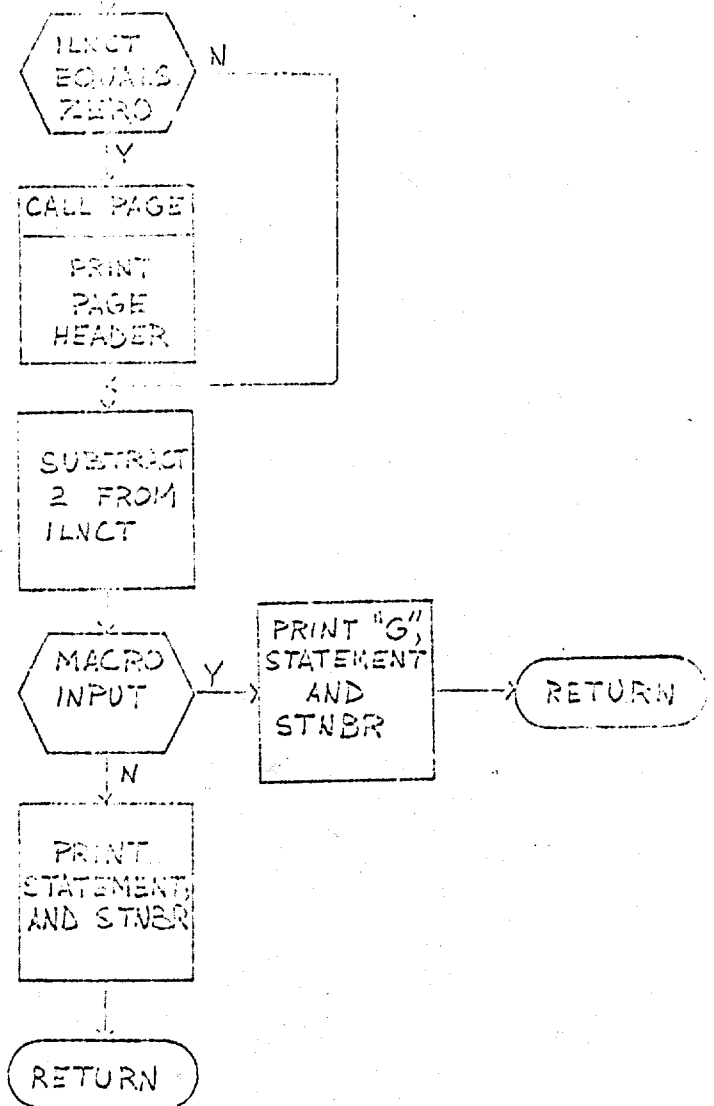


SUBROUTINE SA7

This routine is used by the ASM1 routine to print the input statements. The routine provides two auxiliary functions. The first of these is that a check is made of the number of lines written on the page. If the number of lines has been used up, the PAGE routine is called. The second auxiliary function consists of a check to see if the input is coming from the macro input file, corresponding to the expansion of a macro call, in which case a G precedes the print out of the statement.

234

SUBROUTINE SA7
(PRINT INPUT)



SUBROUTINE SA8

This routine prints out the hardware definitions at the end of the first phase of the first pass subroutine. The CPU, MEMORY, CHANNEL, and DEVICE definitions are printed out on the system printer.

236

SUBROUTINE SAB
(PRINT HARDWARE TABLES)

PRINT
CPU
DEFINITIONS

PRINT
MEMORY
DEFINITIONS

PRINT
CHANNEL
DEFINITIONS

PRINT
DEVICE
DEFINITIONS

RETURN

SUBROUTINE SA9

This routine is called by the first pass sub-routine to output hardware and program definition tables to the simulator input tape. The queue tables, load class tables, run class tables, real file tables, program distribution table, table dump control table, output statistics control, output interrupt vector table, output memory assignment table, are written to the simulator input tape. In addition, the job name and ordinal file name blocks are written to the statistics tape.

200

SUBROUTINE SAD
(OUTPUT TABLE)

OUTPUT
Q-TABLE

OUTPUT
P-Q
CLASS
TABLE

OUTPUT
RUN
CLASS
TABLE

OUTPUT
REAL
FILE
TABLE

OUTPUT
PROGRAM
DISTRIBUTION
TABLE

OUTPUT
TABLE DUMP
CONTROL
TABLE

OUTPUT
STEP DUMP
CONTROL
TABLE

V
ZSOK

200

200

OUTPUT
INTERROPT
VECTOR
TABLE

OUTPUT
MEMORY
ADDRESSMENT
TABLE

WRITE
JOE
NAME
BLOCKS

WRITE
ORDINAL
FILE NAME
BLOCKS

RETURN

SUBROUTINE SERR

This is the error routine for the fix fields routine. It does the same functions as the NERR routine, only it automatically computes the current location of the fix fields scan and includes the current two words to be written onto the error file. These words will then be used in the error message as it is printed out by the error pass. The master error switch is turned ON.

242
3. READ THE SERR
(FIX THE SERR ERROR MESSAGE)

COMPUTE
ADDRESS
OF CURRENT
PACKED
WORD

WRITE THE
NUMBER STATE-
MENT NUMBER
AND TWO MES-
SAGE NUMBERS

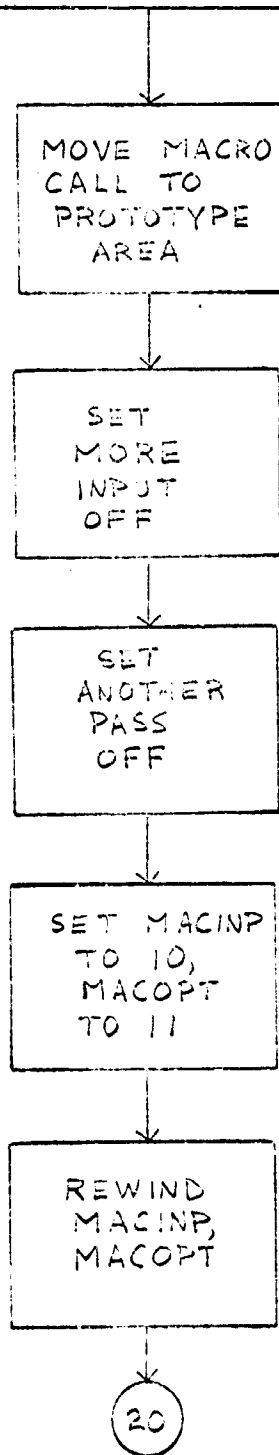
SET
ERROR
SWITCH

RETURN

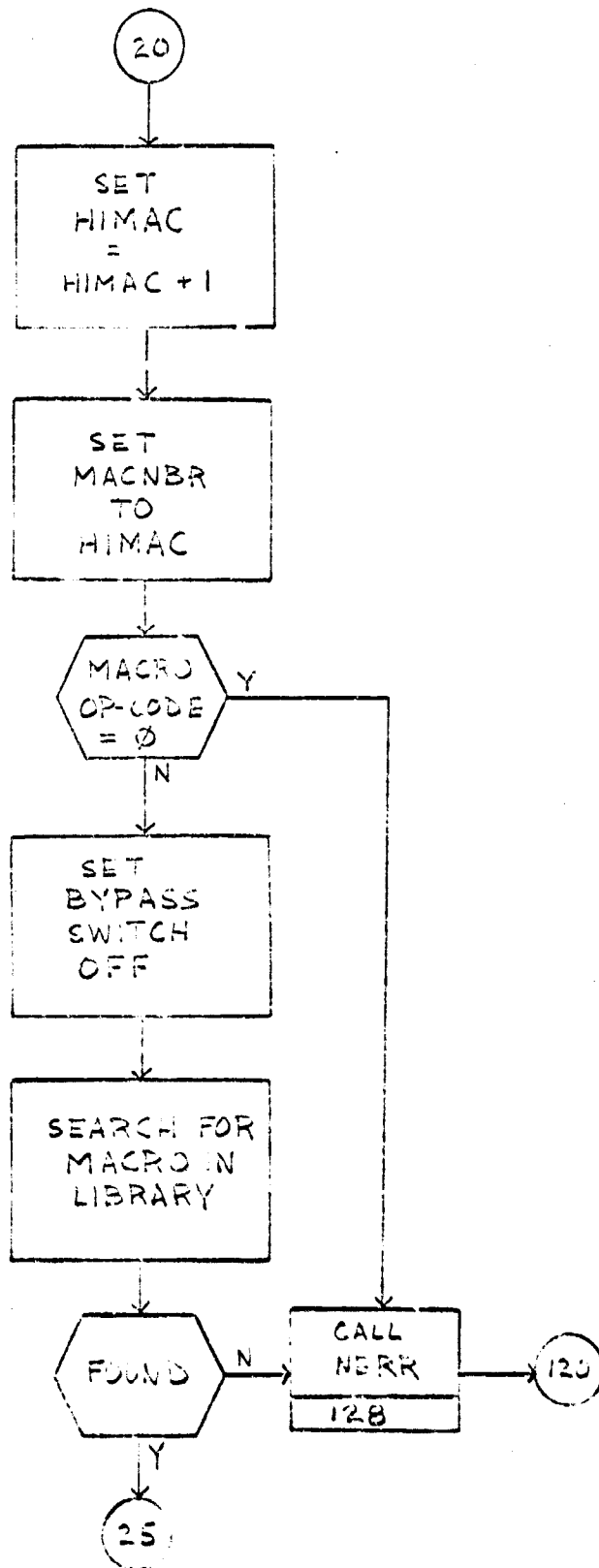
SUBROUTINE SMACRO

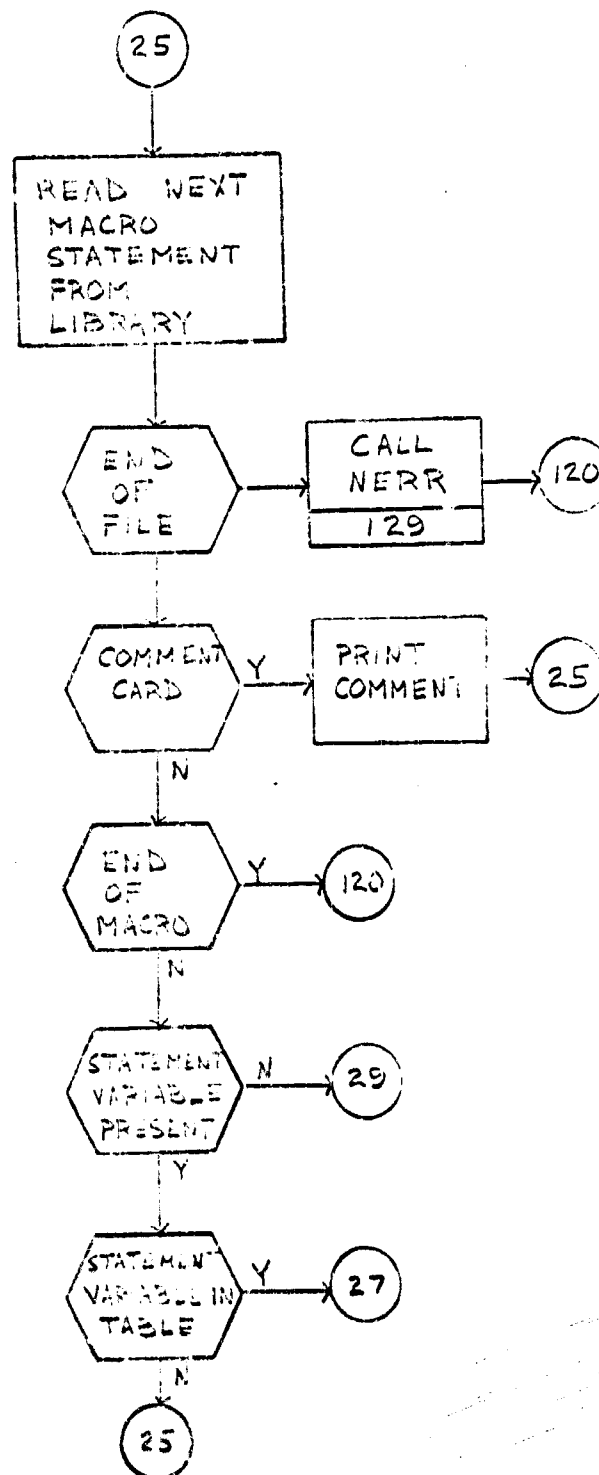
This routine is called to process a macro call appearing in job source input. This routine expands the macro call, processing nested macro calls as they occur, and places the generated statements into a macro input file. A switch is then set which signals the first pass subroutine that is now to take its input from the macro input file.

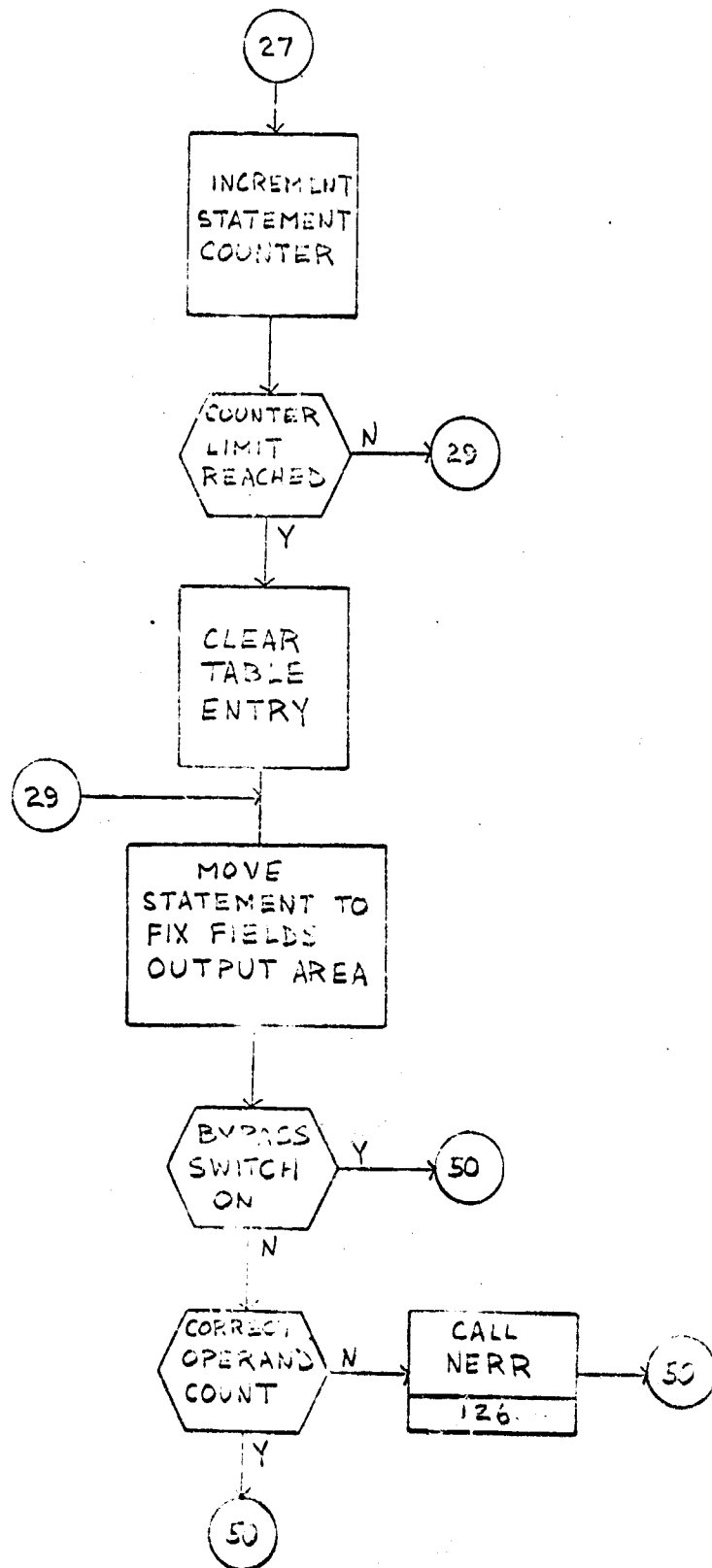
The SMACRO subroutine first moves the macro call to a prototype area. It then performs some house-keeping, initializing itself and setting values for macro numbers which must be appended to statement labels. The library is then searched for the macro call name, and the macro is read from the library and stored on the macro input file. Statement label variables, and statement labels requiring macro numbers to be appended are processed as required.

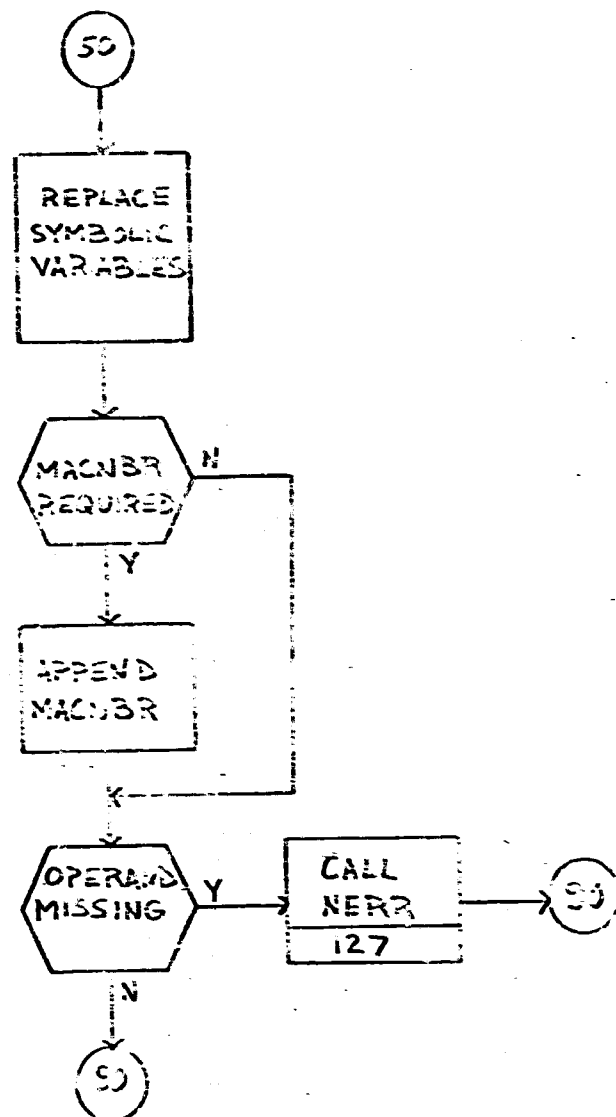
SUBROUTINE SMACRO
(PROCESS MACRO)

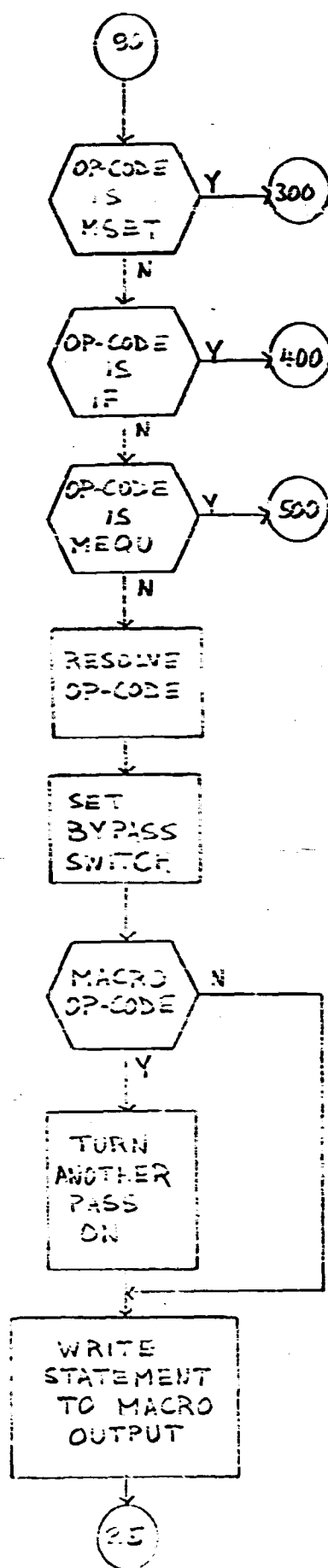
240

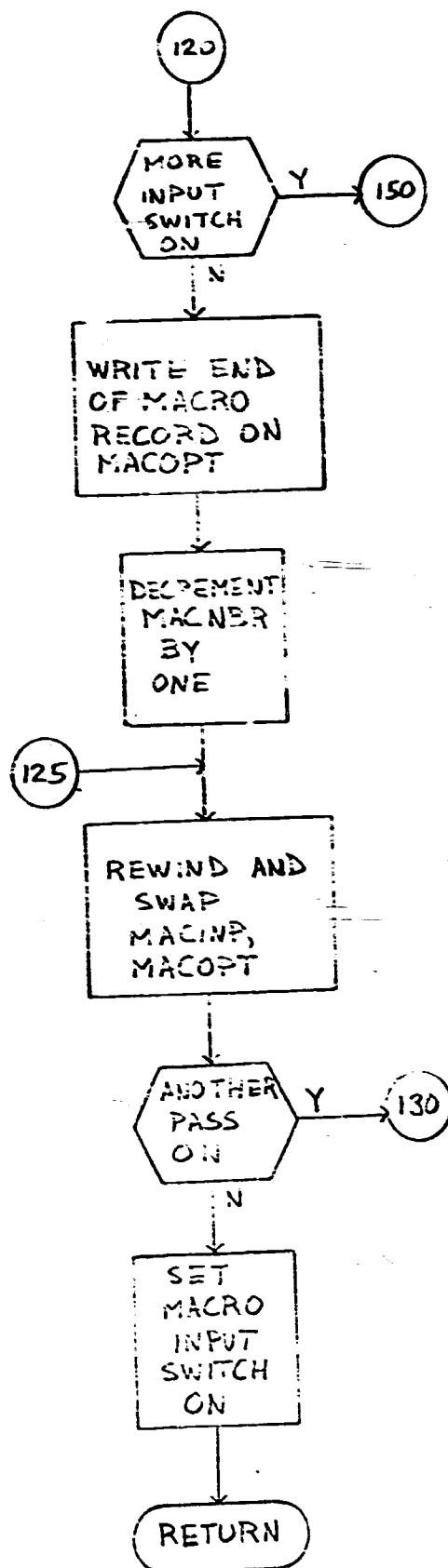


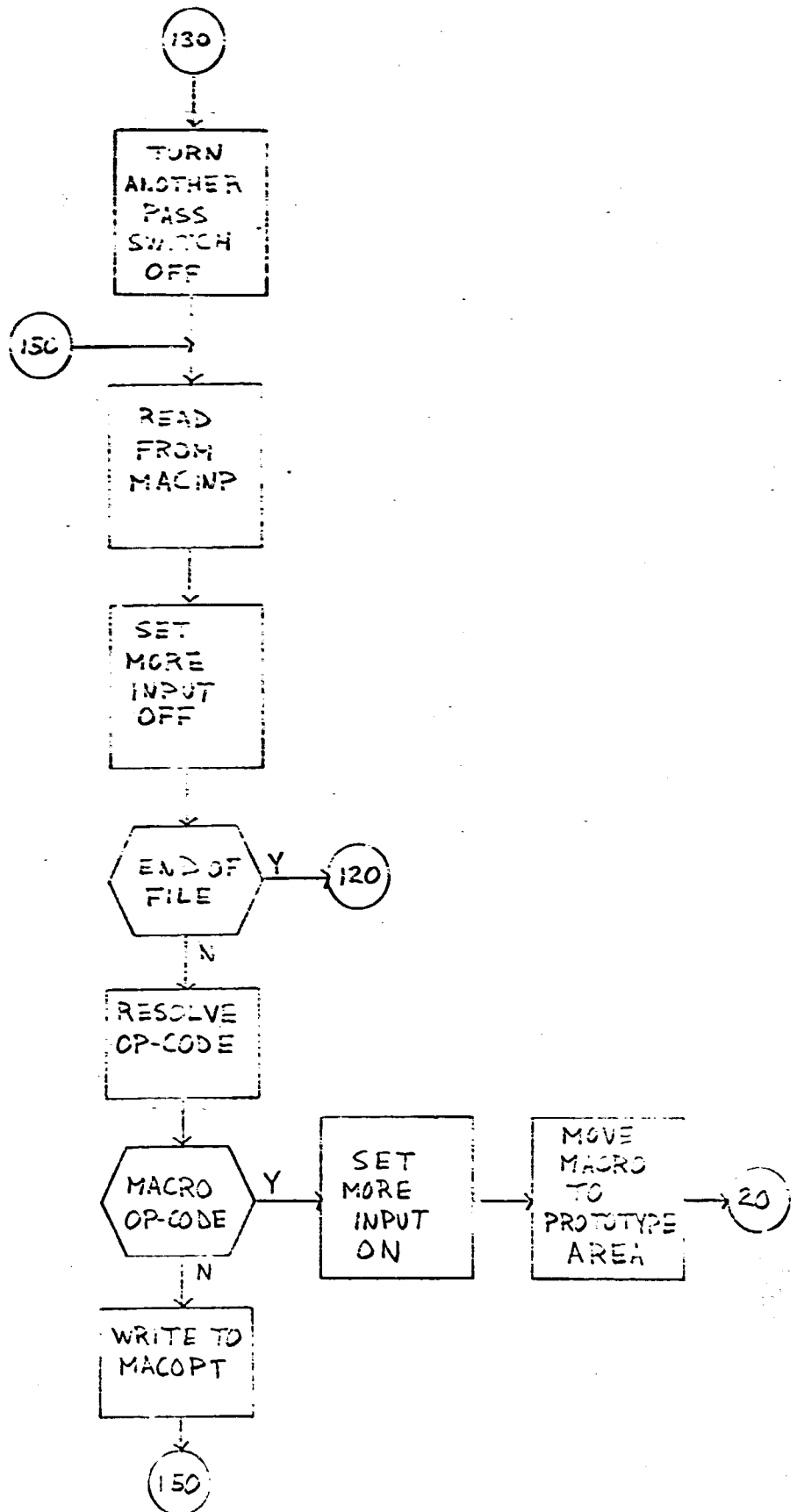




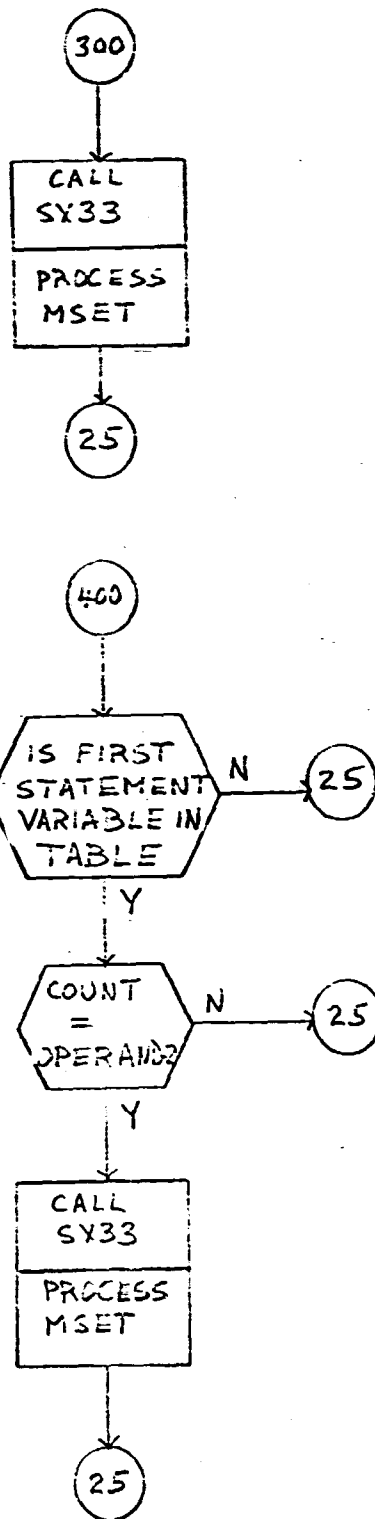




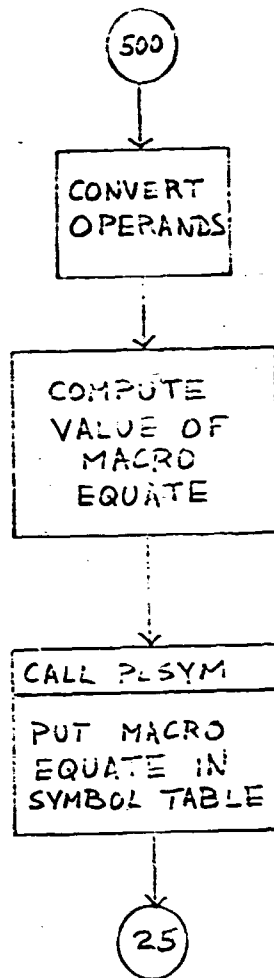




251

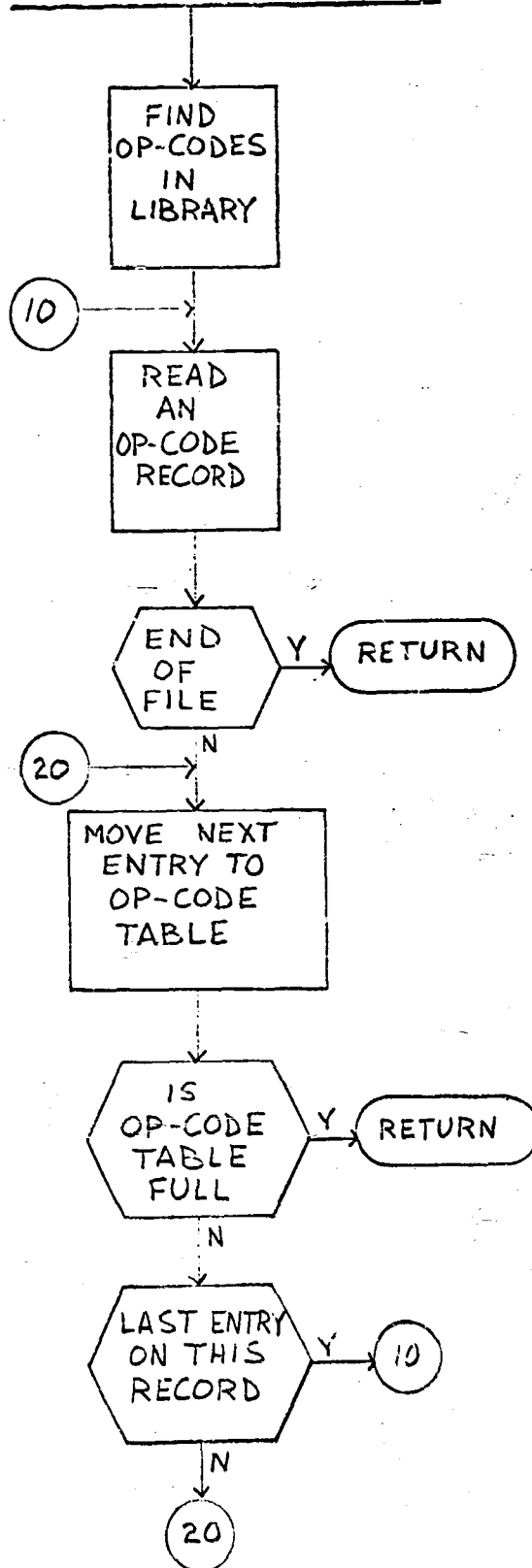


25?



SUBROUTINE SOPCD

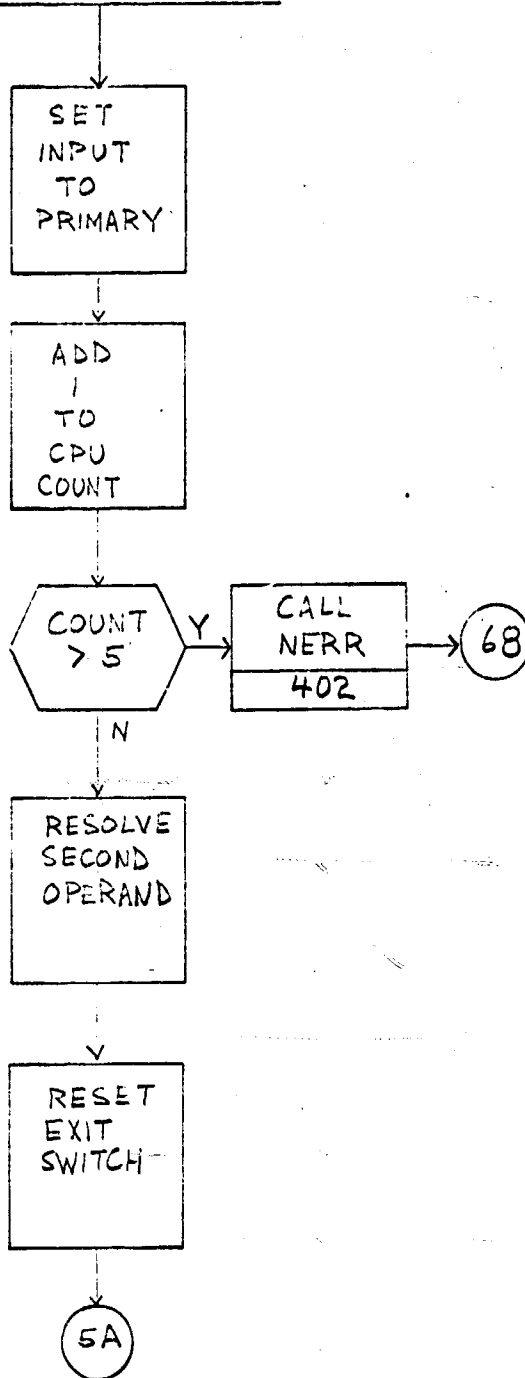
This routine searches the library for the op-codes. After finding them, it reads them into the op-code table in memory where it is used for processing by the first pass. After reading the formatted and packed op-codes and loading them unpacked into the table, the remainder of the table is filled with all nines to insure a correct binary search of the table for op-codes.

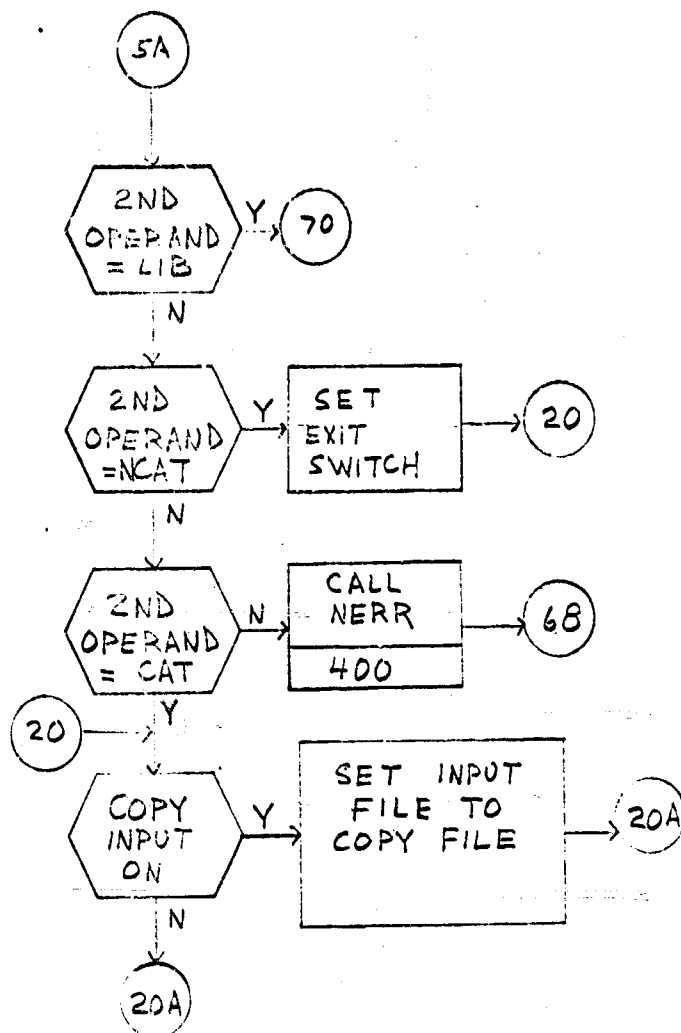
SUBROUTINE: SØPCD
(LOAD ØP-CØDE TABLE)

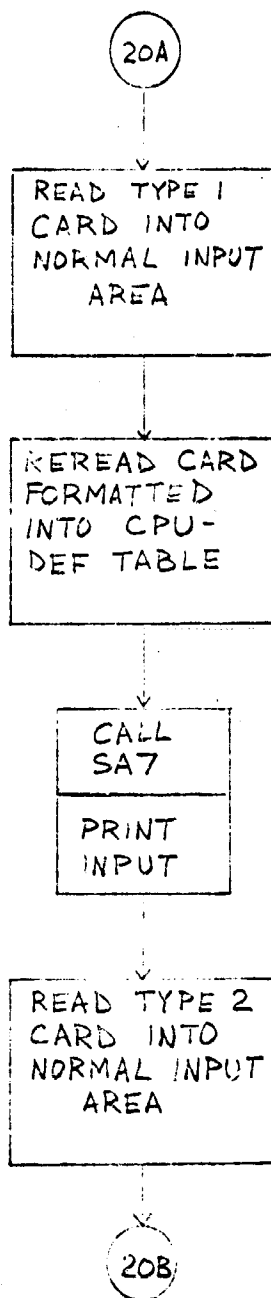
SUBROUTINE SX1

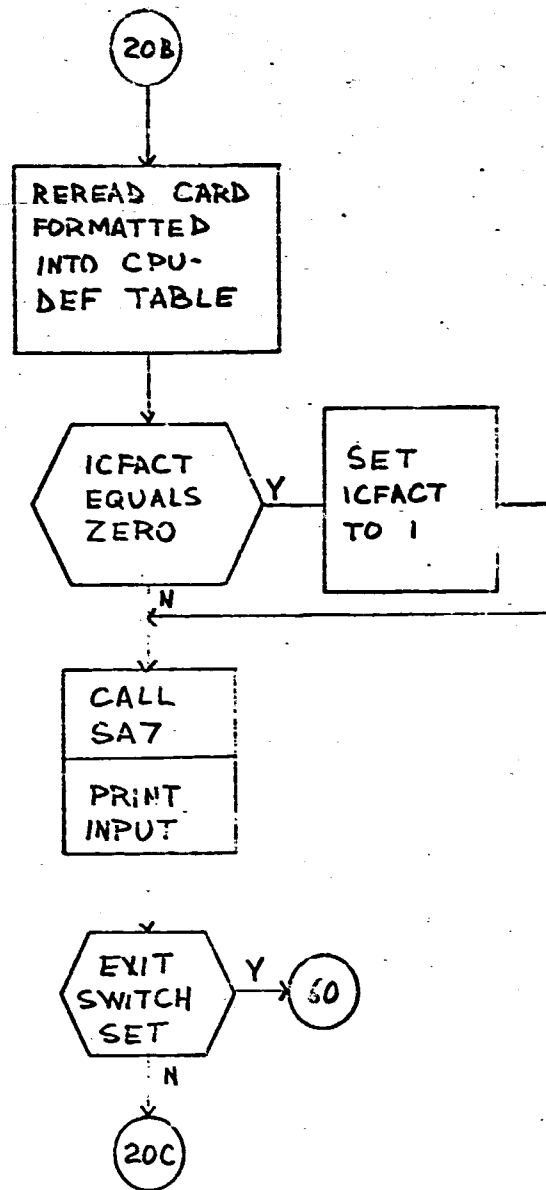
This routine processes CPU definitions.

Within this routine, a test is made of the second operand, which indicates whether the CPU definition is to be found in the library, or if it is in the input stream, whether it is to be catalogued or not. The CPU definition is then read from the library or from the input stream as required, and placed into the appropriate table in common. The CPU definition is printed out as it is received. A count is maintained of the number of such CPU definitions received. A check is made to determine if the maximum number has been exceeded.

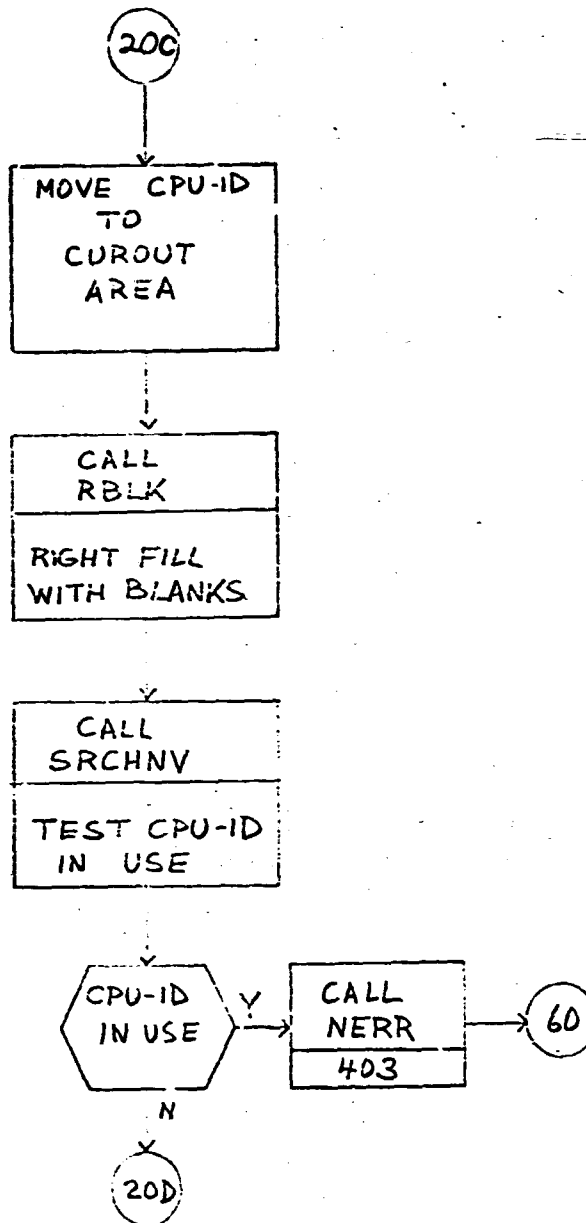
SUBROUTINE SX1
(CPU - DEF)







260



261

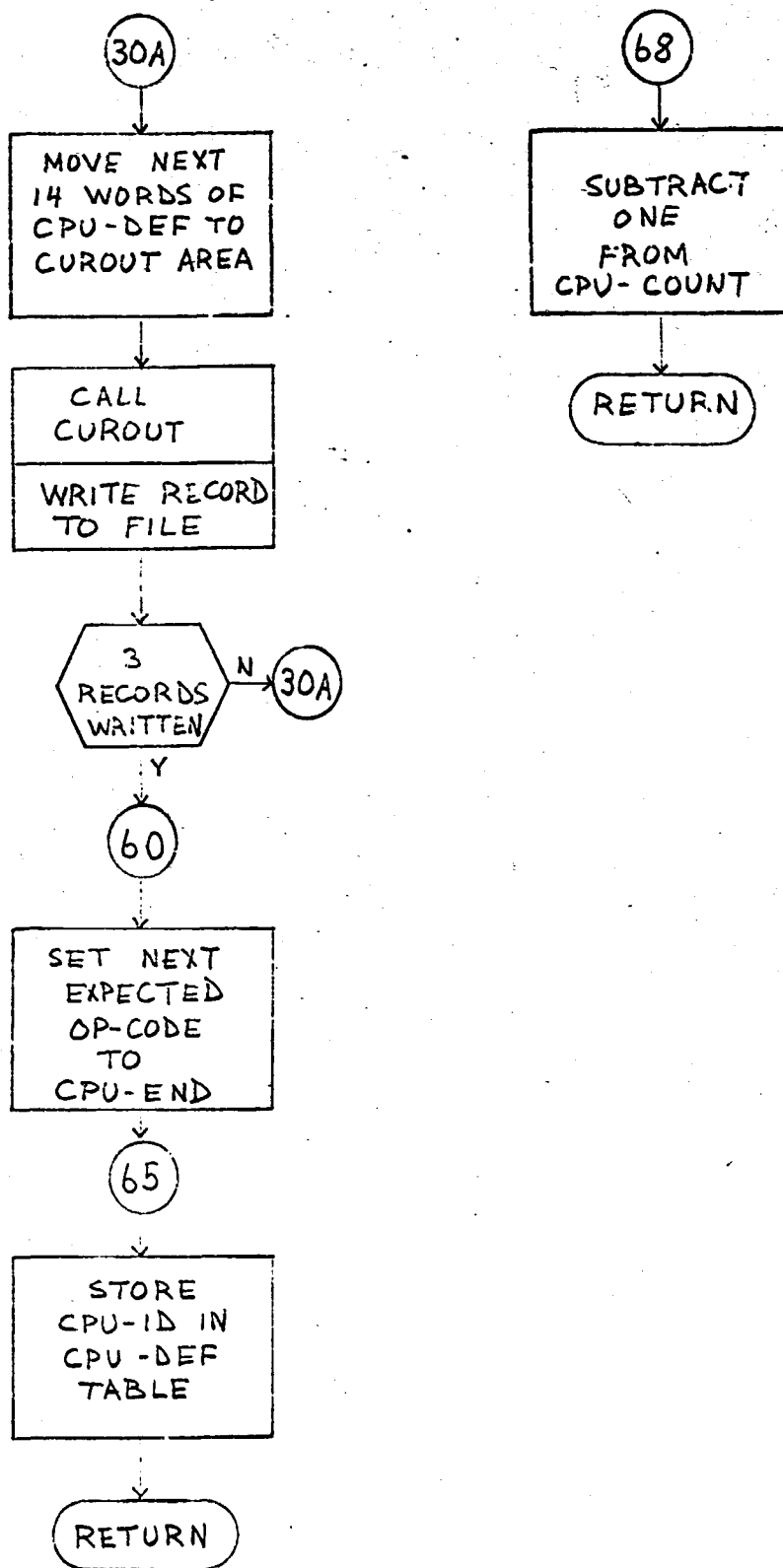
20D

SET
VERSION
FIELD
TO BLANKS

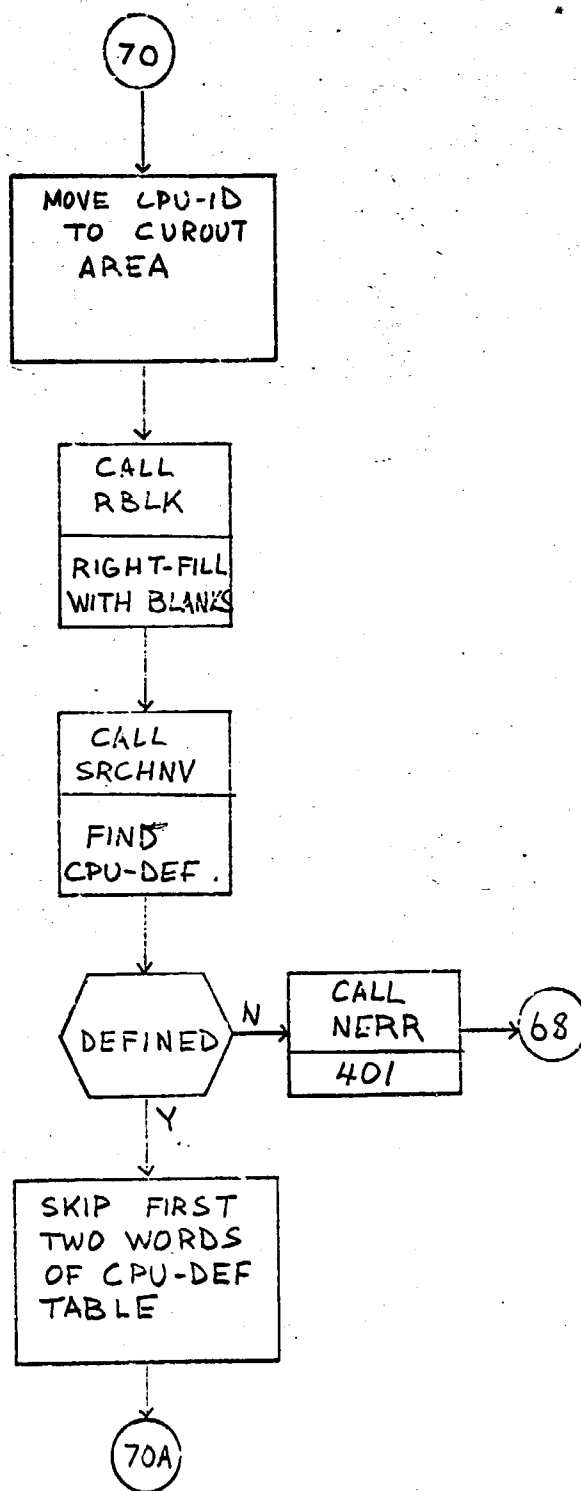
CALL
CURRNT
CREATE
FILE FOR
CPU-ID

SKIP
FIRST
TWO WORDS
OF CPU-DEF

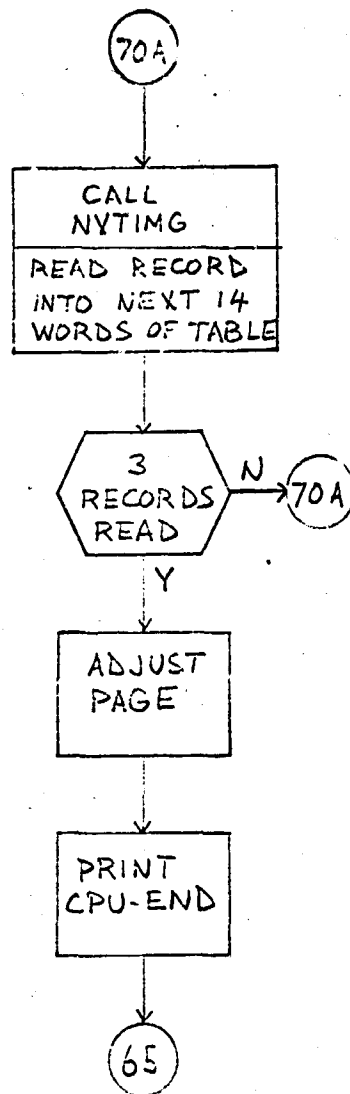
30A



26?



26A



SUBROUTINE SX2

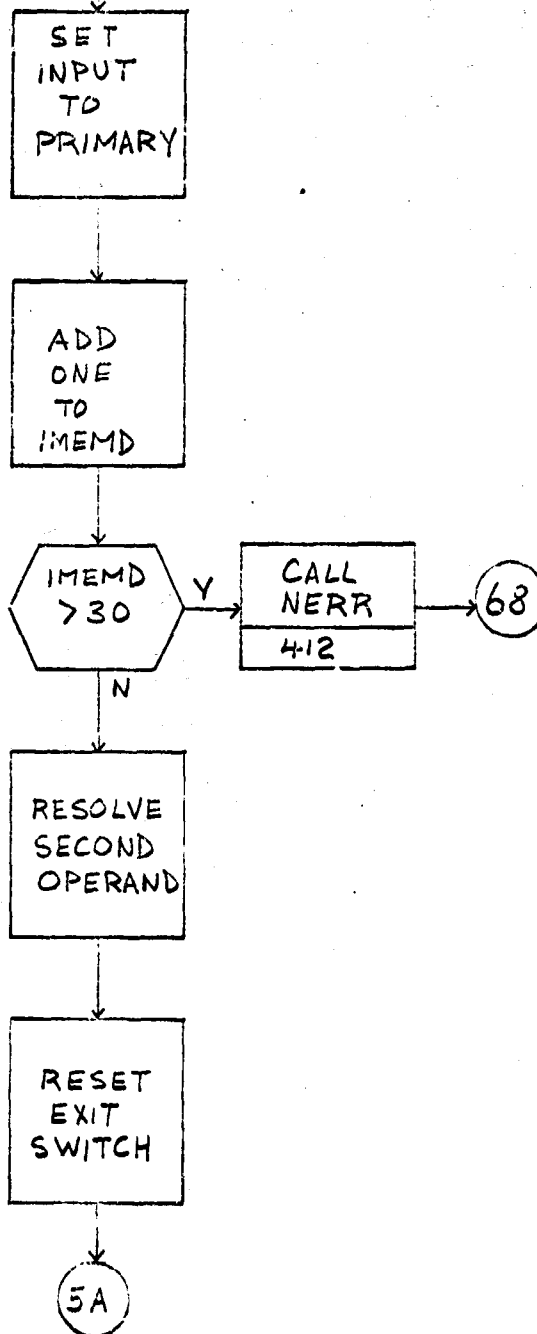
This routine processes memory definitions.

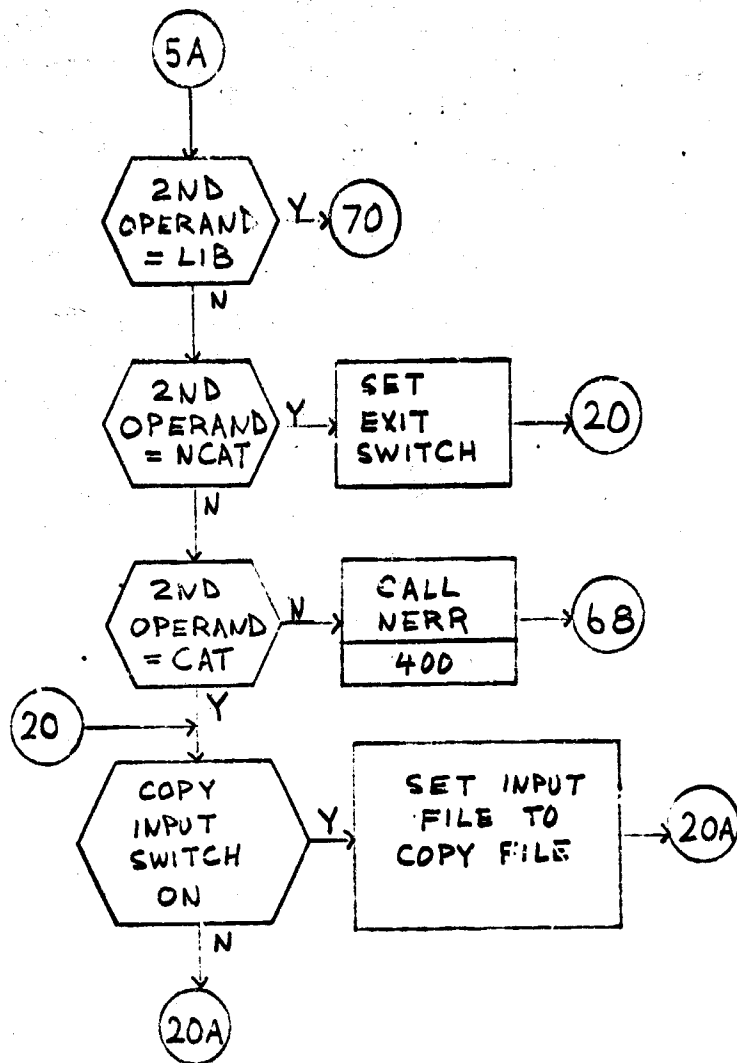
The second operand is checked to see whether the definition is in the library or, if it is in the input stream, whether it is to be catalogued or not.

The memory definition is then obtained from the appropriate input file, printed out, and stored in an appropriate table in common. A count is made of the number of memory definitions. A check is made to see if the number of memory definitions has exceeded the maximum permitted.

266

SUBROUTINE SX2
(MEM-DEF)





258

20A

READ TYPE 3
CARD INTO
NORMAL INPUT
AREA

REREAD CARD
FORMATTED
INTO MEM-
DEF TABLE

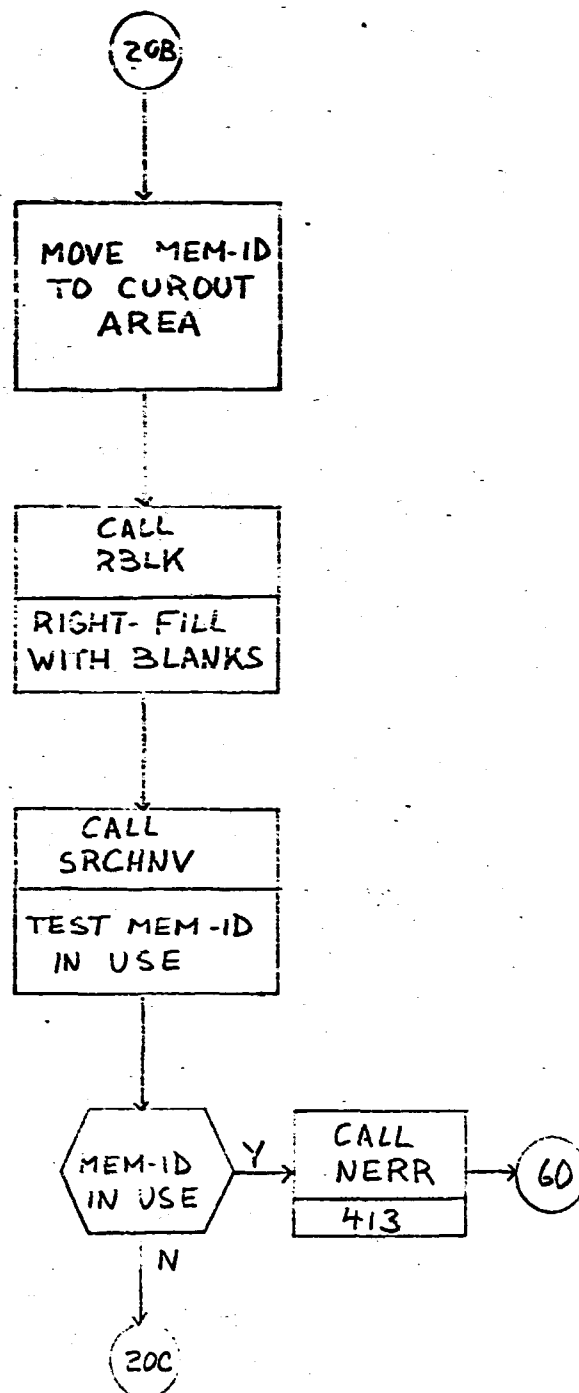
CALL
SA7
PRINT
INPUT

EXIT
SWITCH
SET

Y → 60

N → 20B

269



250

20C

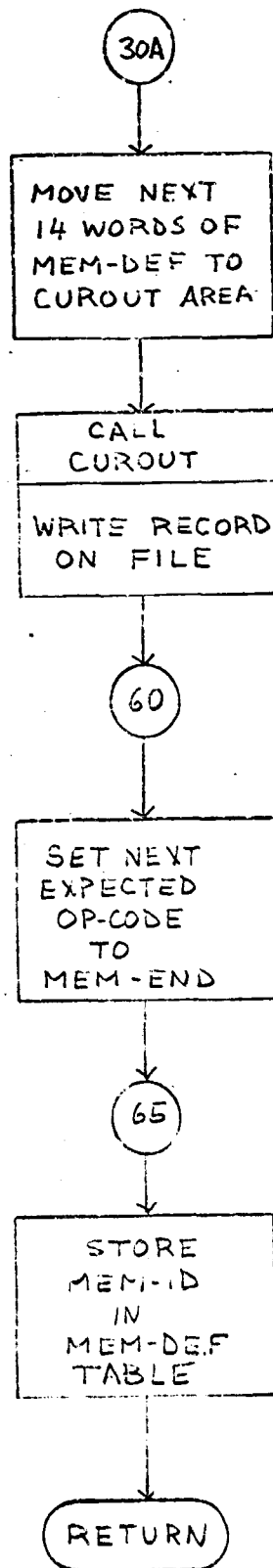
SET VERSION
FIELD
TO
BLANKS

CALL
CUROUT
CREATE FILE
FOR MEM-ID

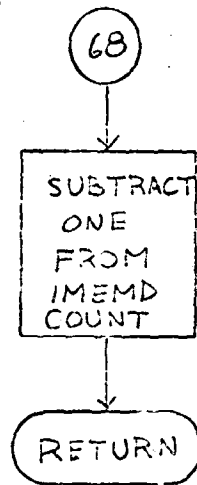
SKIP FIRST
TWO WORDS
OF MEM-DEF

30A

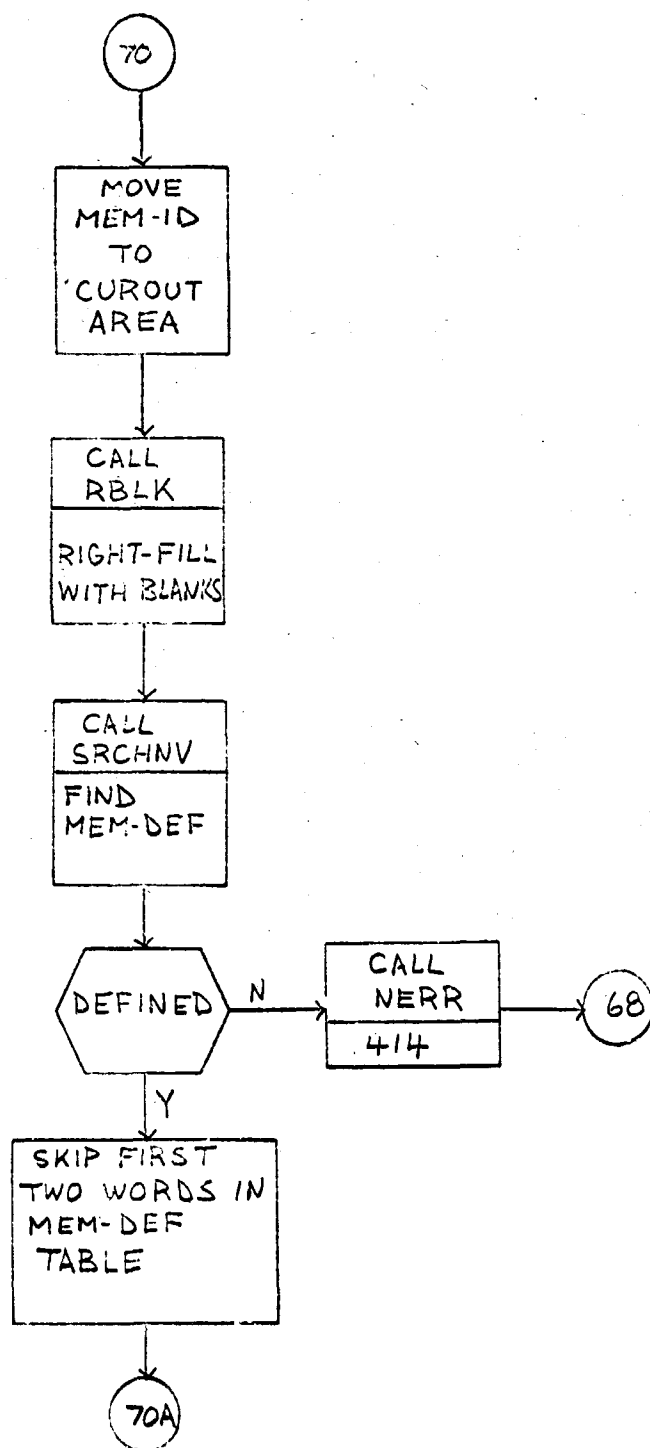
277



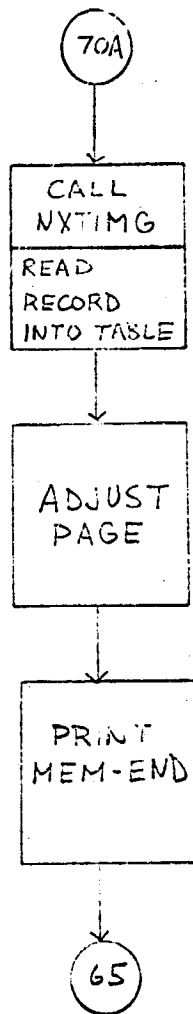
272



273



274



SUBROUTINE SX3

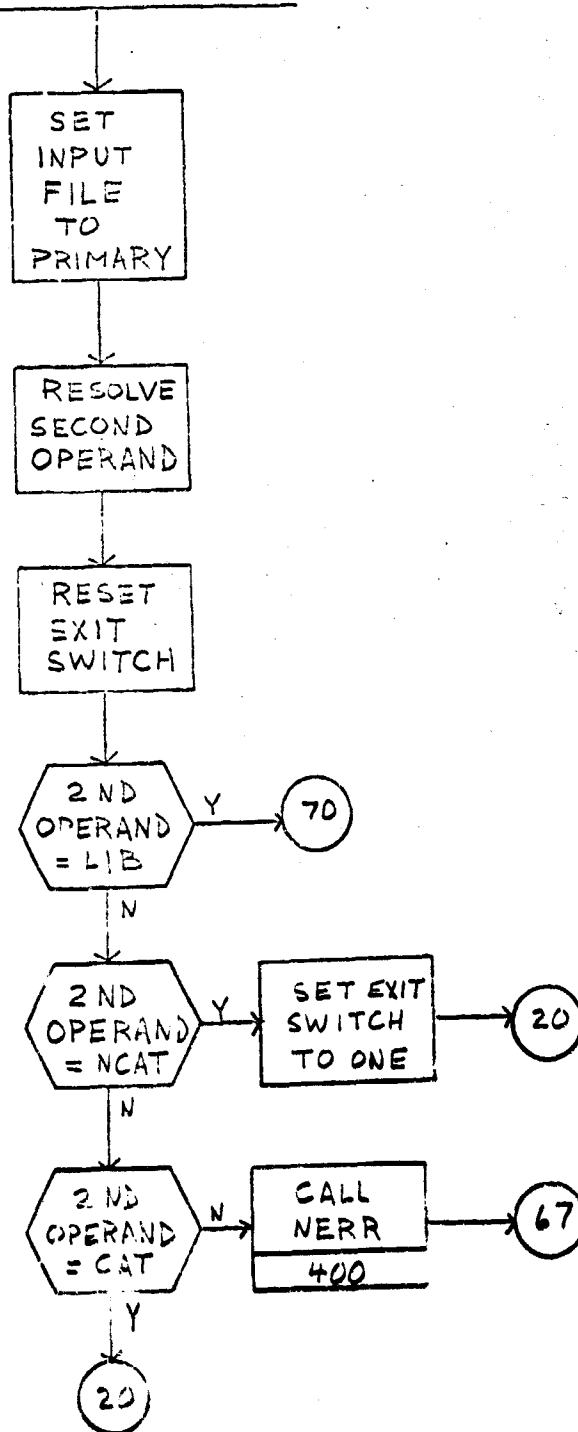
This routine processes channel definitions.

The second operand is checked to see whether the definition is in the library or, if it is in the input stream, whether it is to be catalogued or not.

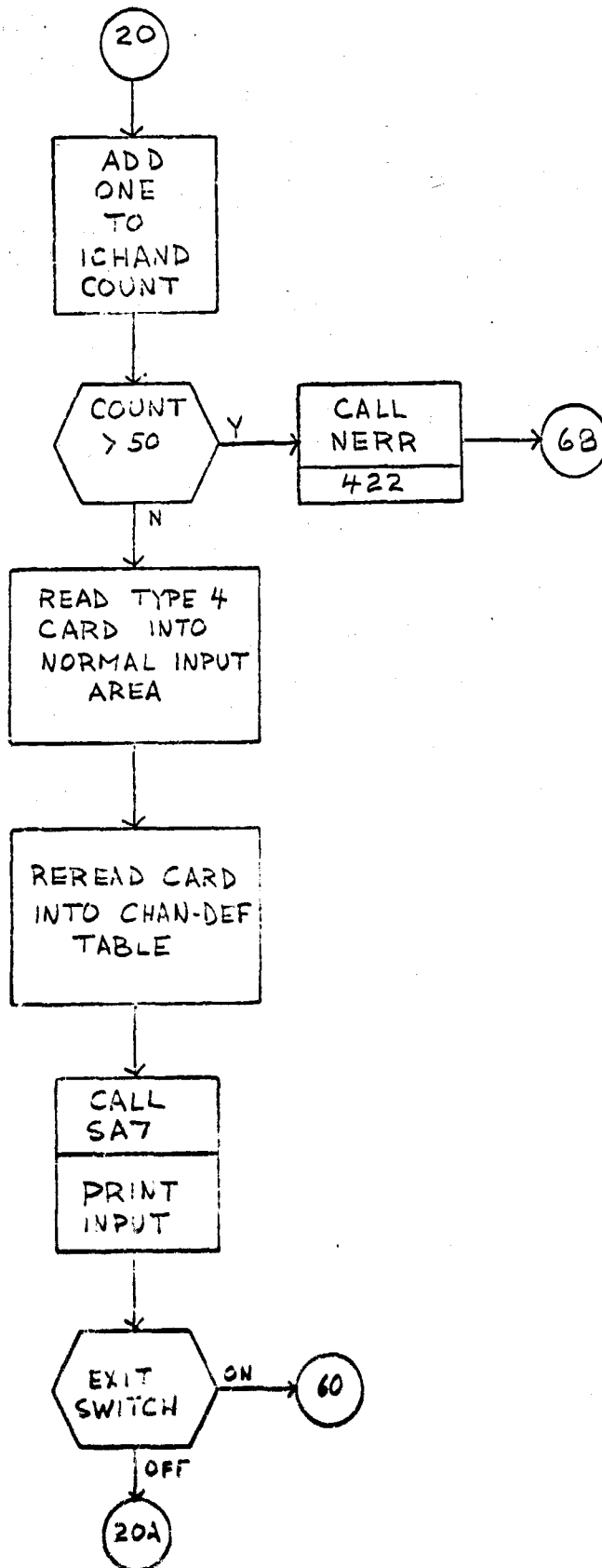
The channel definition is then obtained from the appropriate input file, printed out, and stored in an appropriate table in common. A count is made of the number of channel definitions. A check is made to see if the number of channel definitions has exceeded the maximum permitted.

276

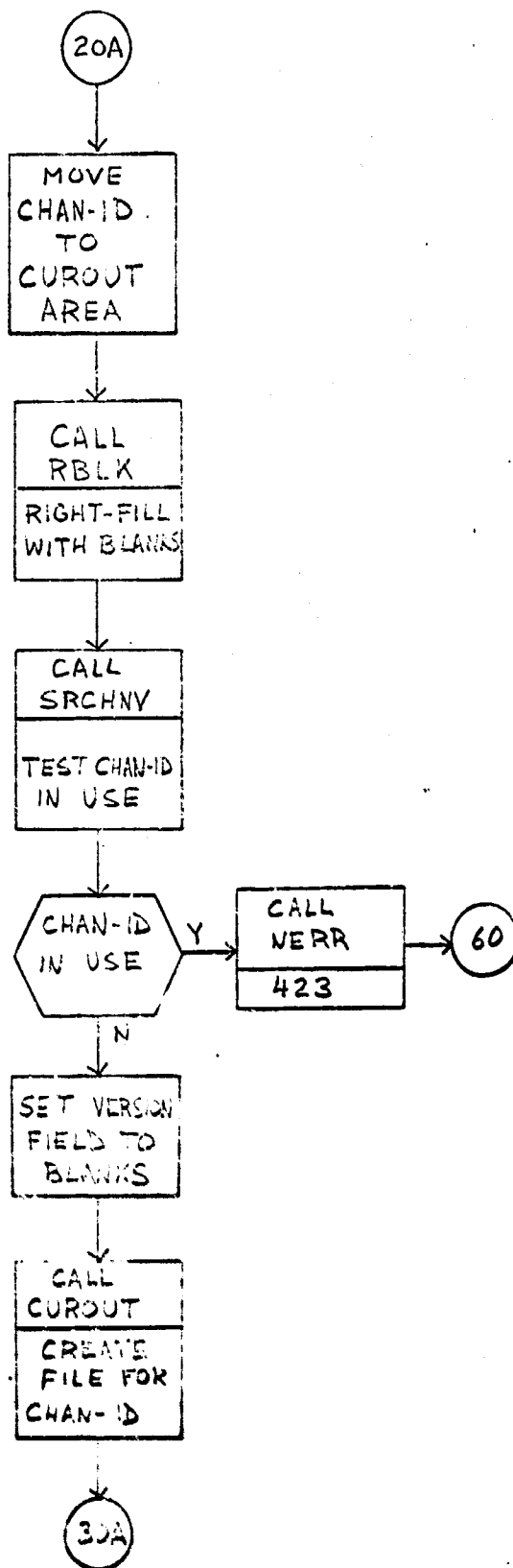
SUBROUTINE SX3
(CHAN-DEF)



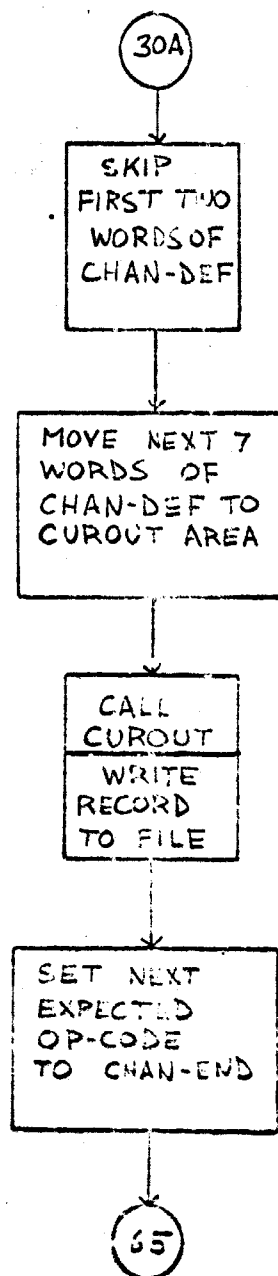
277

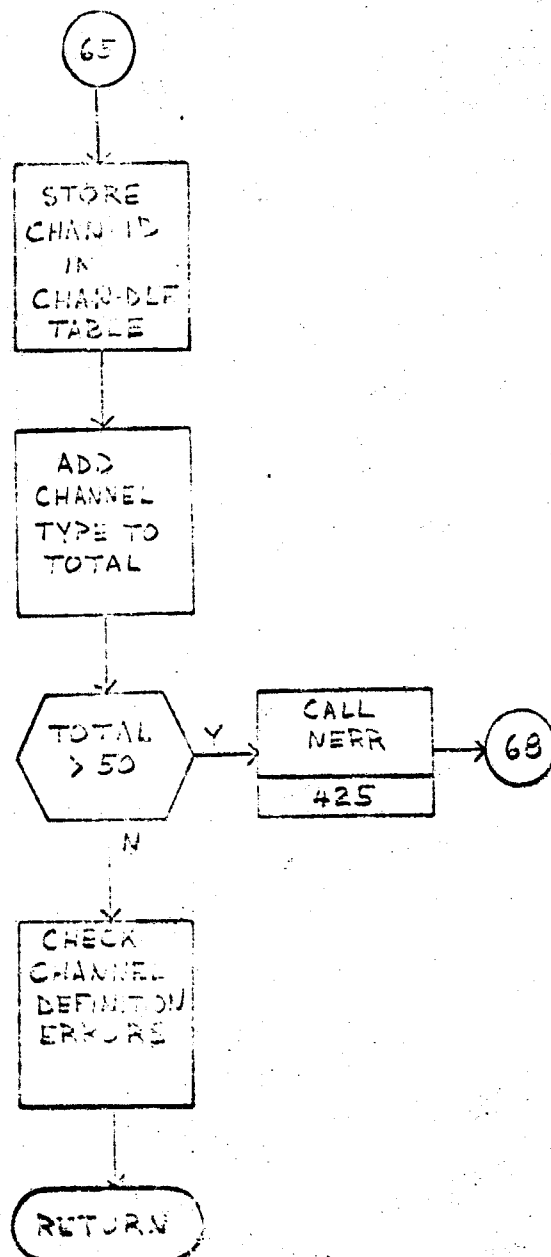


278

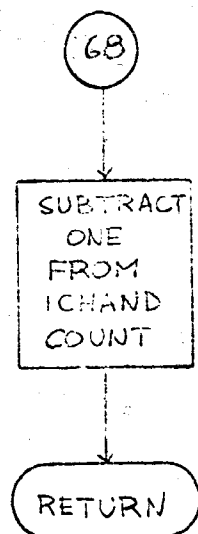


279

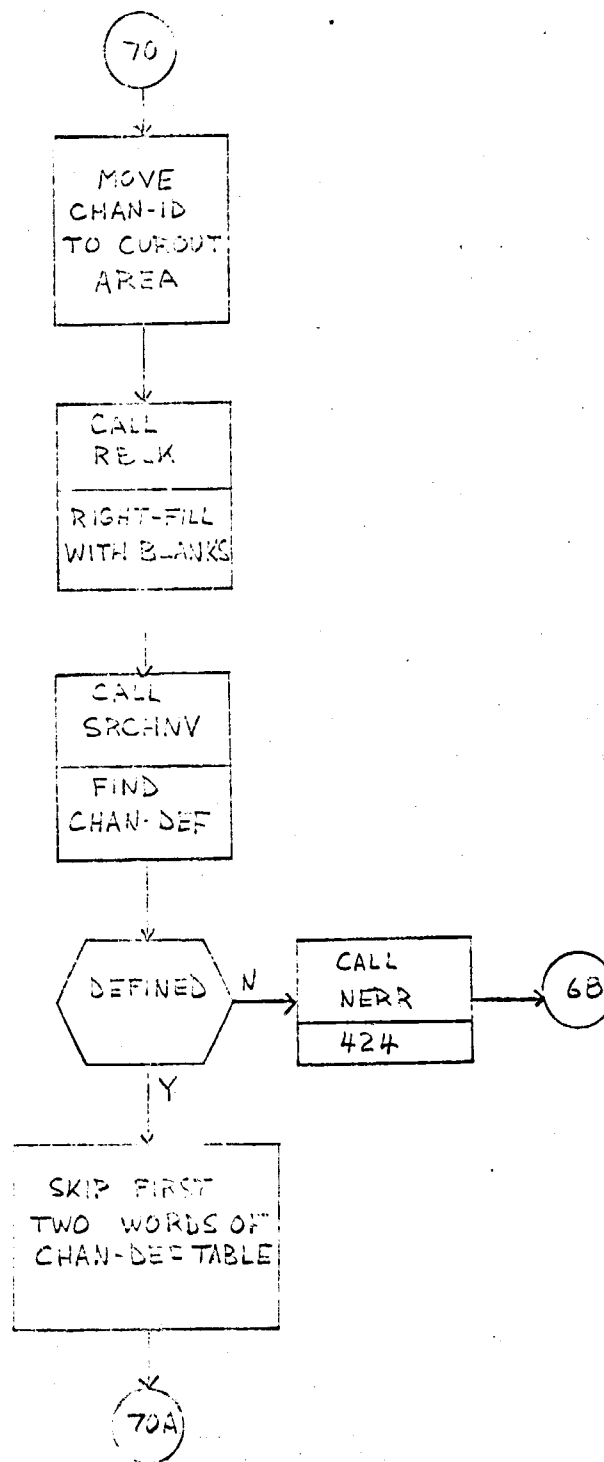




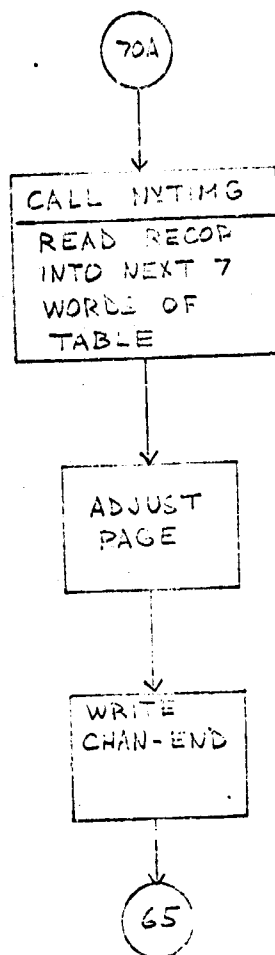
281



282



283



SUBROUTINE SX4

This routine processes a device definition statement. In this routine, the second operand is checked first to see if it indicates that the device definition is to be taken from the library, or if it is in the input stream, whether it is to be catalogued or not. The device definition is then read from the appropriate input file. It is then re-read into the device definition table. If the device definition is to be catalogued, a check is made to see if the device ID is already in use. If it is, an error is written. If not, a new file is created for the device definition and the device definition is catalogued.

285

SUBROUTINE SX4
(DEV-DEF)

SET
INPUT
TO
PRIMARY

RESOLVE
SECOND
OPERAND

SET
EXIT
SWITCH
OFF

SECOND
OPERAND
IS LIB

Y 70

NCAT

Y

SET
EXIT
SWITCH
ON

20

CAT

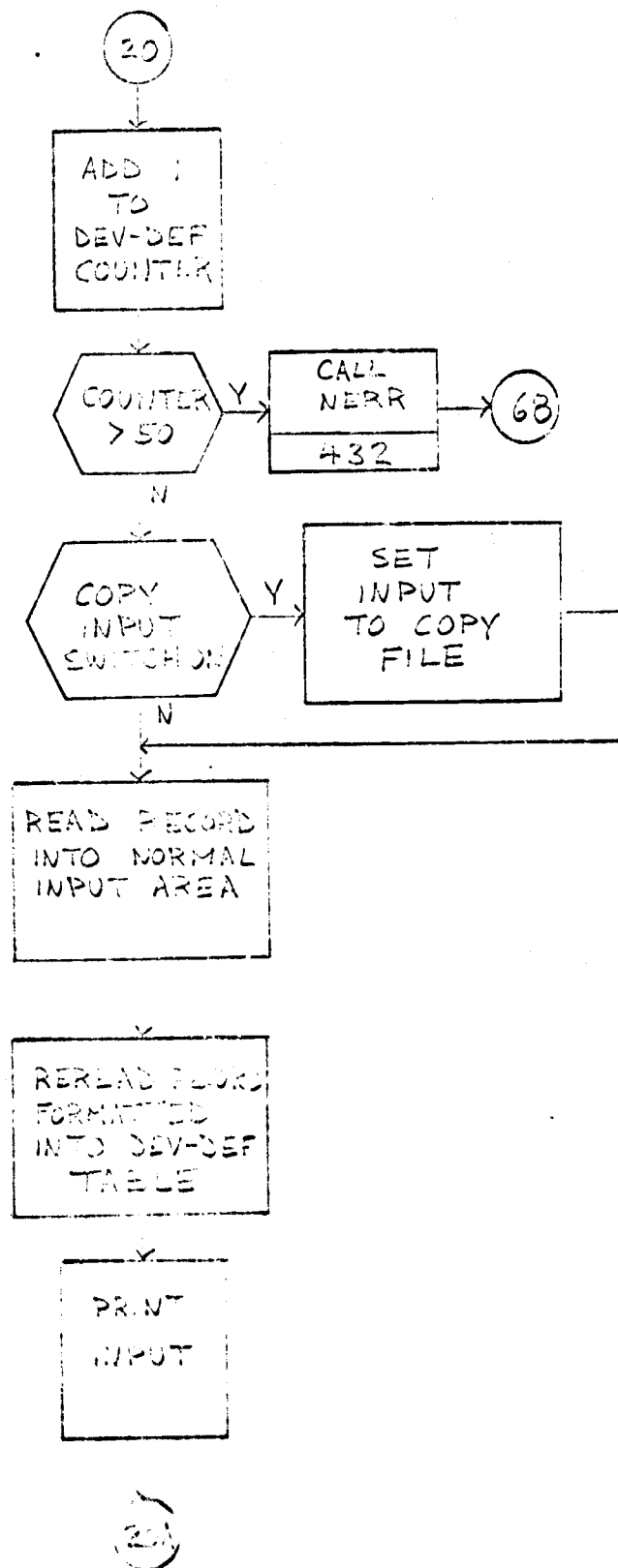
Y

20

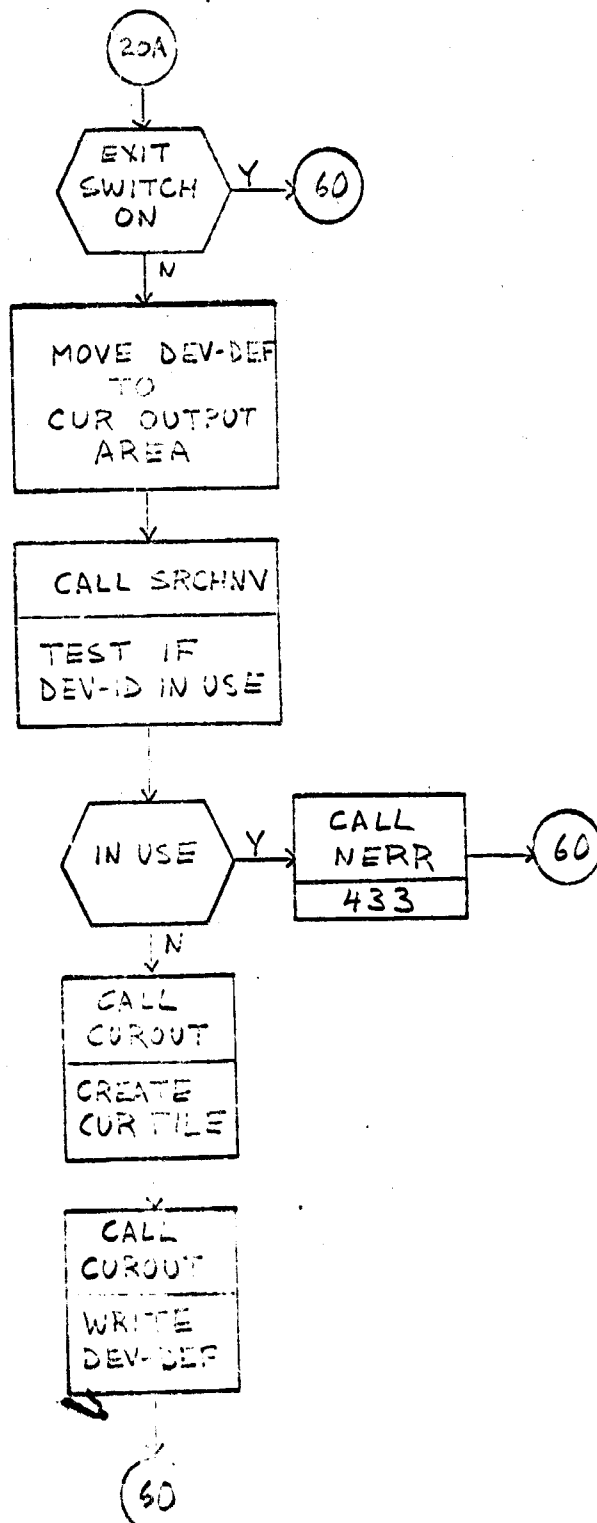
CALL
NCHK
405

67

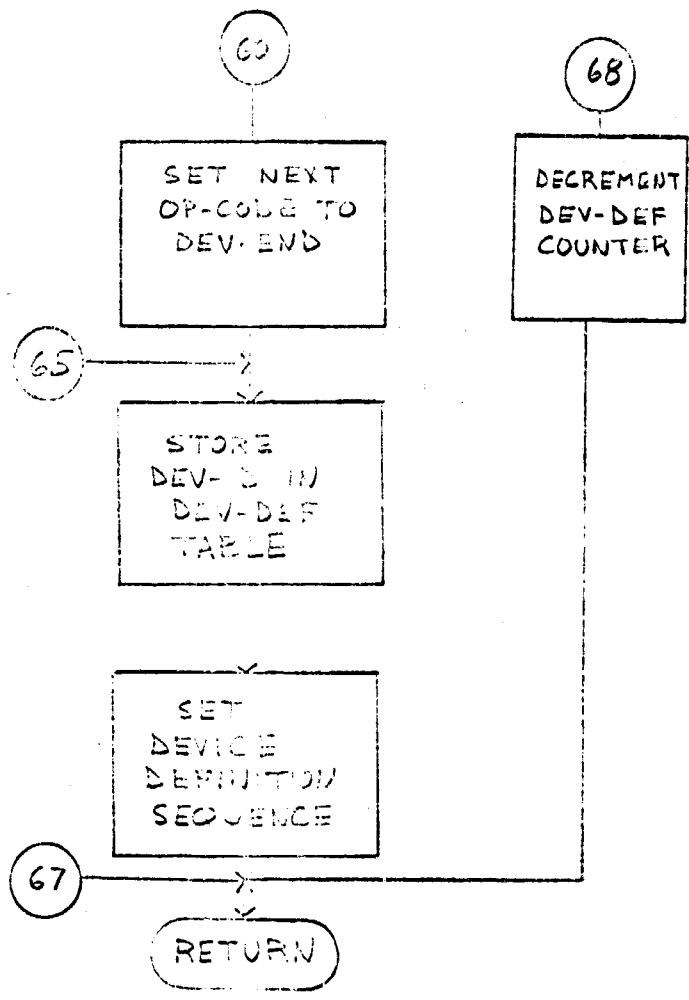
286

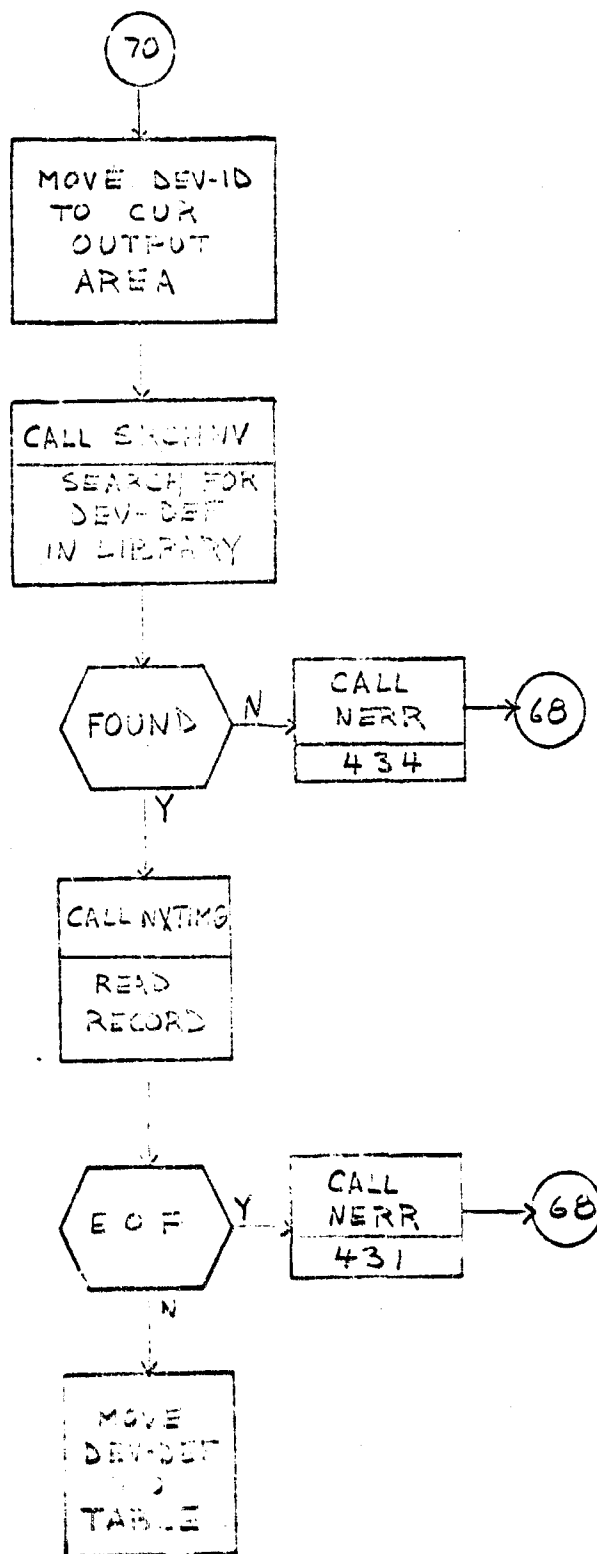


287



288



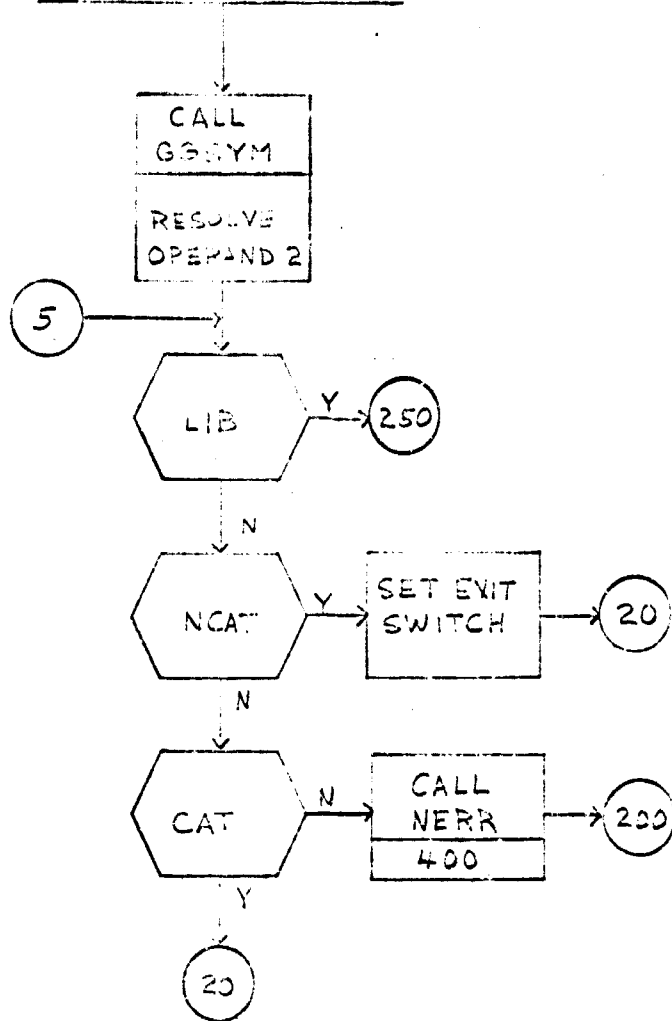


SUBROUTINE SX5

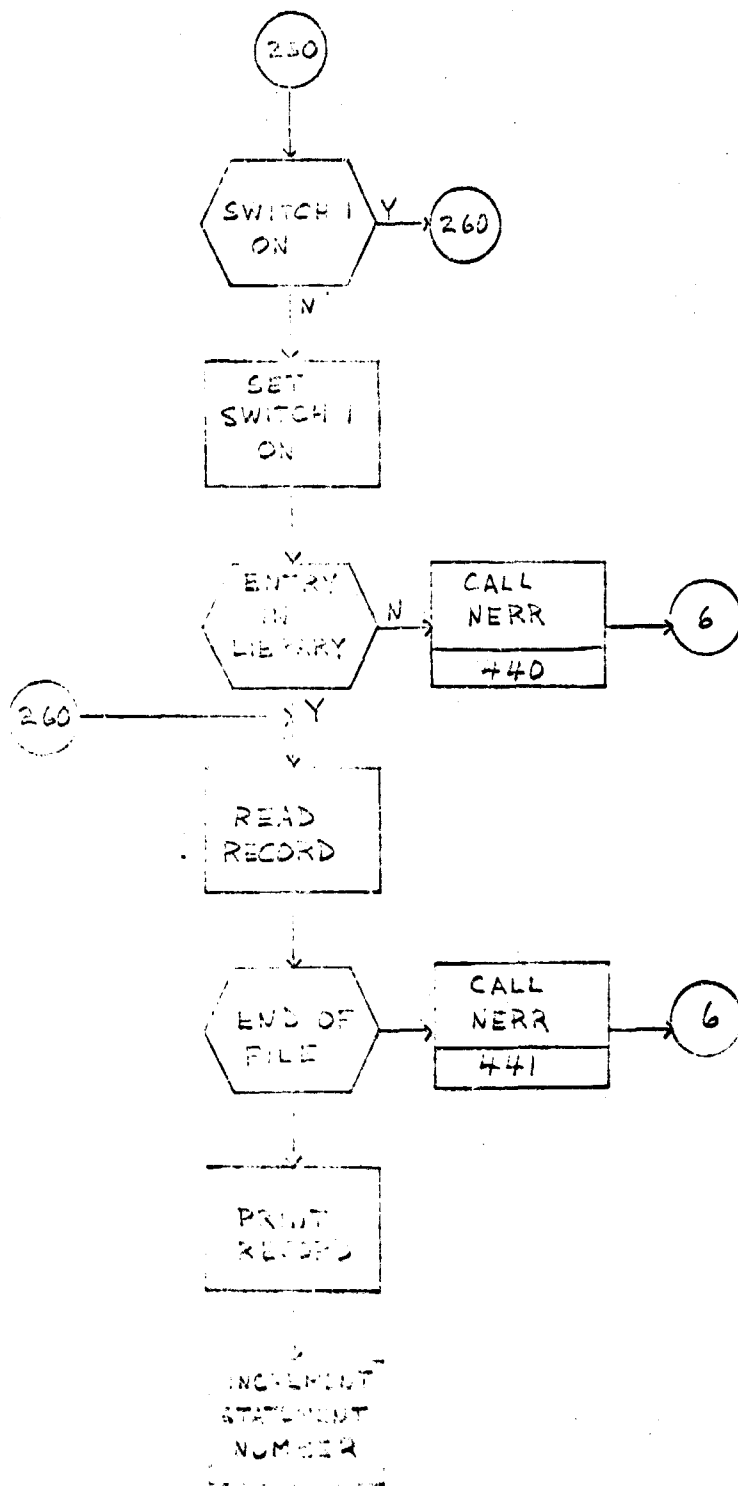
This routine processes the configuration card. The second operand is checked to see whether the configuration is in the library or, if it is in the input stream, whether it is to be catalogued or not. The next configuration card is then obtained from the appropriate input file, printed out, and stored in an appropriate table in common. As each subsequent input is read in, this routine calls other subroutines for further processing according to whether the input is for CPU, MEM, CHAN, CTL, or DEV configuration information. This routine also checks for the CONFIG-END card, at which point a return is made to the first pass subroutine, AS'11.

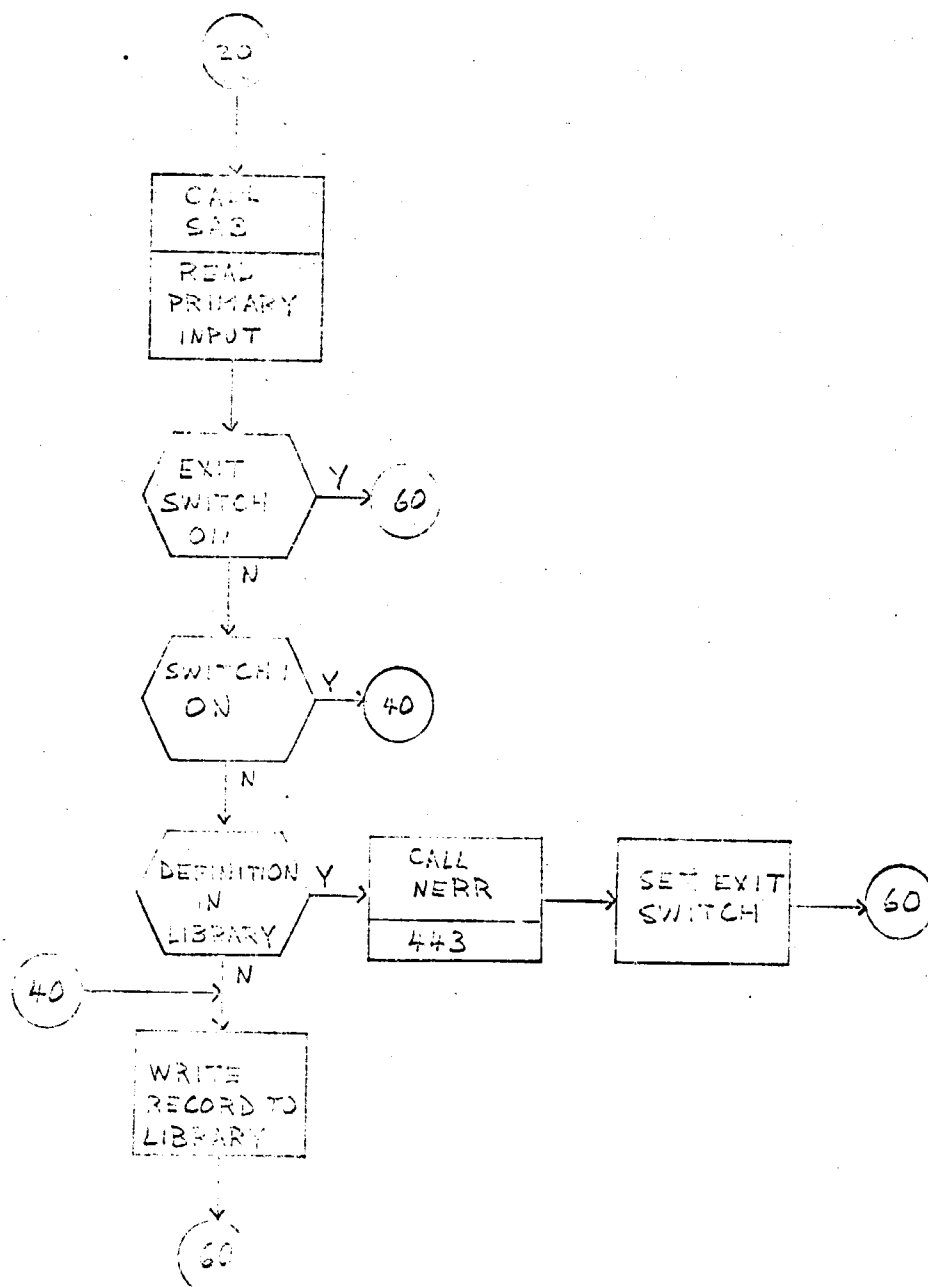
291

SUBROUTINE SX5
(CONFIG CARD)

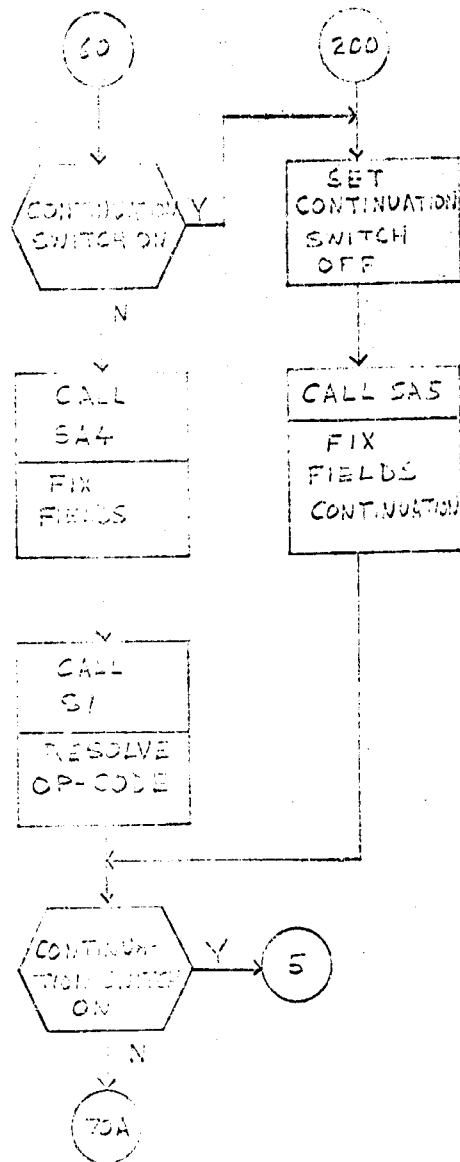


299

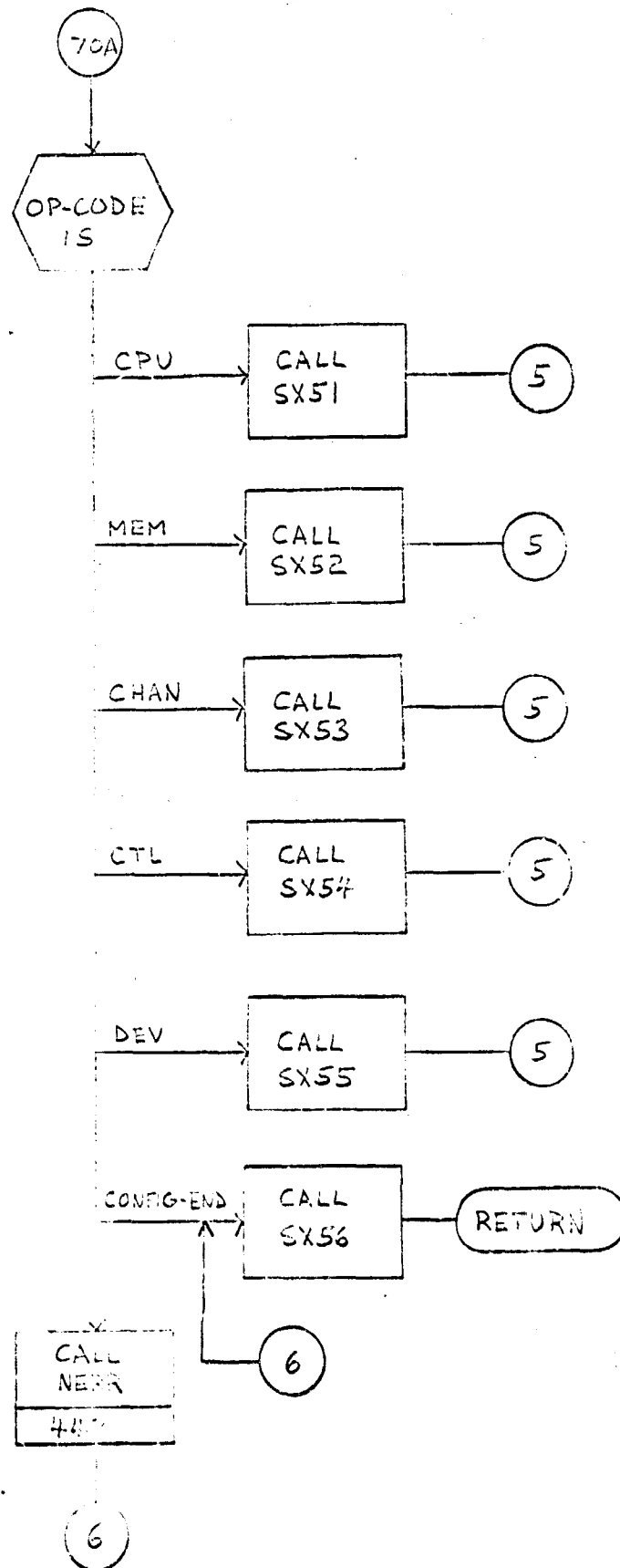




287



285



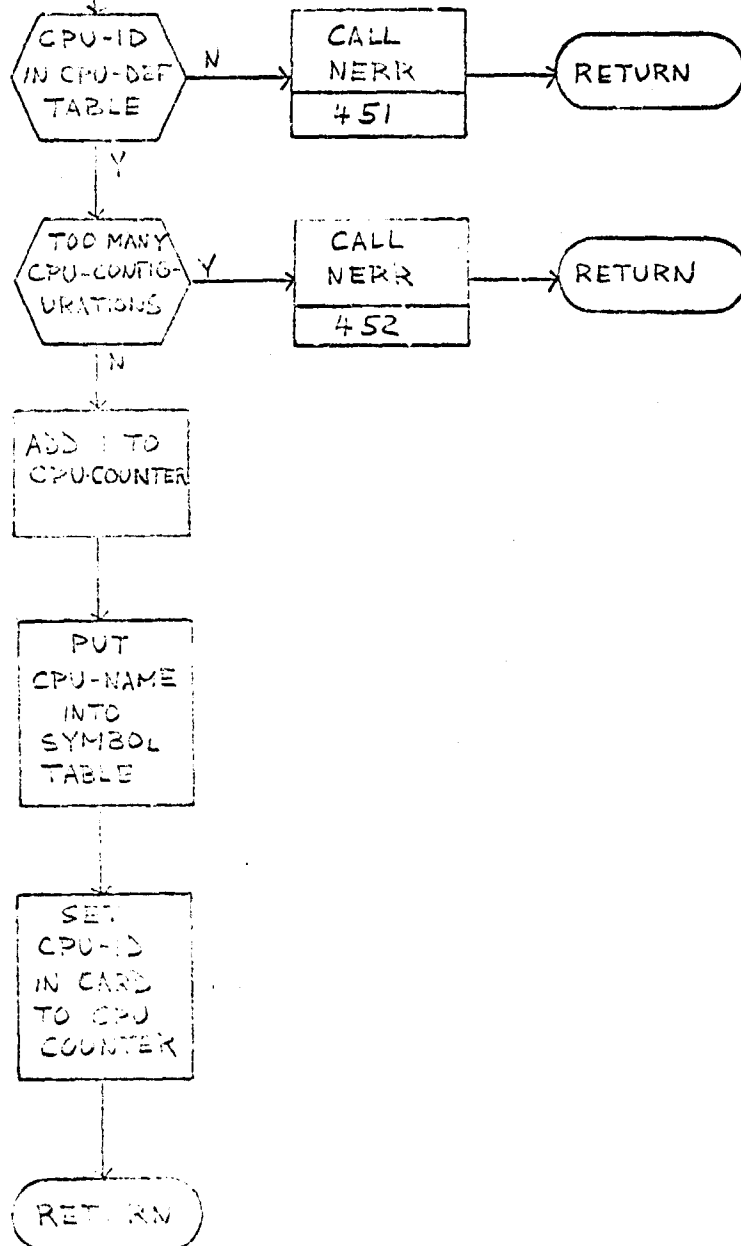
235 .

SUBROUTINE SX51

This routine processes the CPU configuration card. It first checks the CPU definition table to determine if the CPU-ID is valid. After finding the name, it is placed into the global symbol table. Some additional housekeeping is performed.

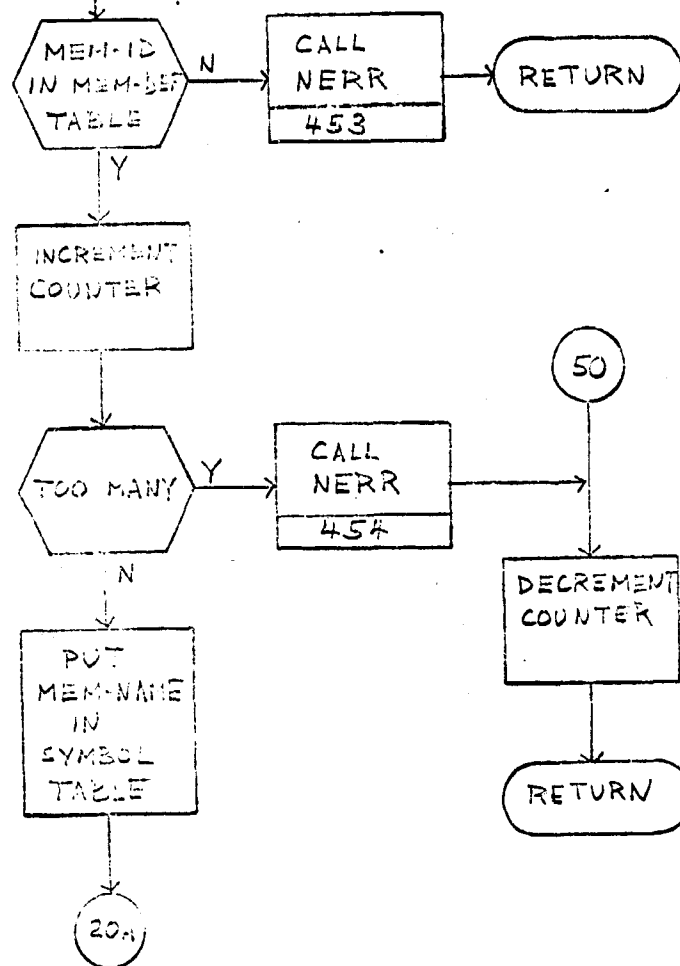
287

SUBROUTINE SX51
(CPU CONFIGURATION)

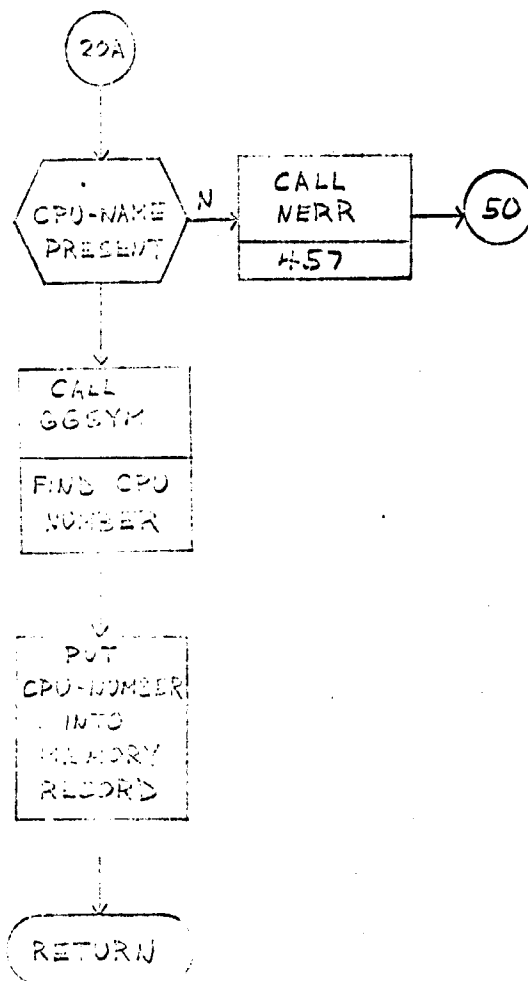


SUBROUTINE SX52

This routine processes the memory configuration card. The MEMORY-ID table is searched to verify that the MEMORY-ID is valid. A test is made to see that the maximum number of memory names is not exceeded. A test is made to see if the associated CPU name is valid.

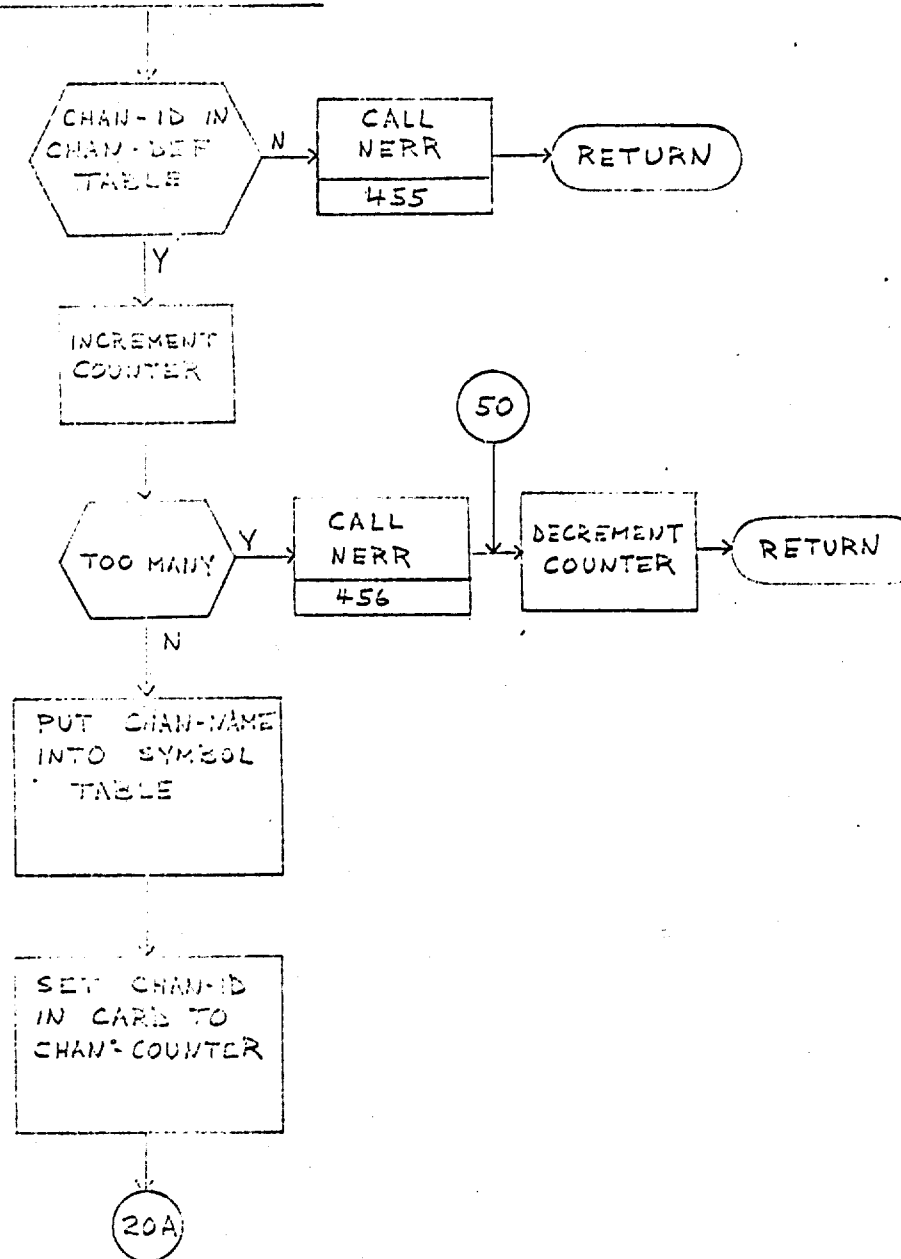
SUBROUTINE SX52
(MEMORY)

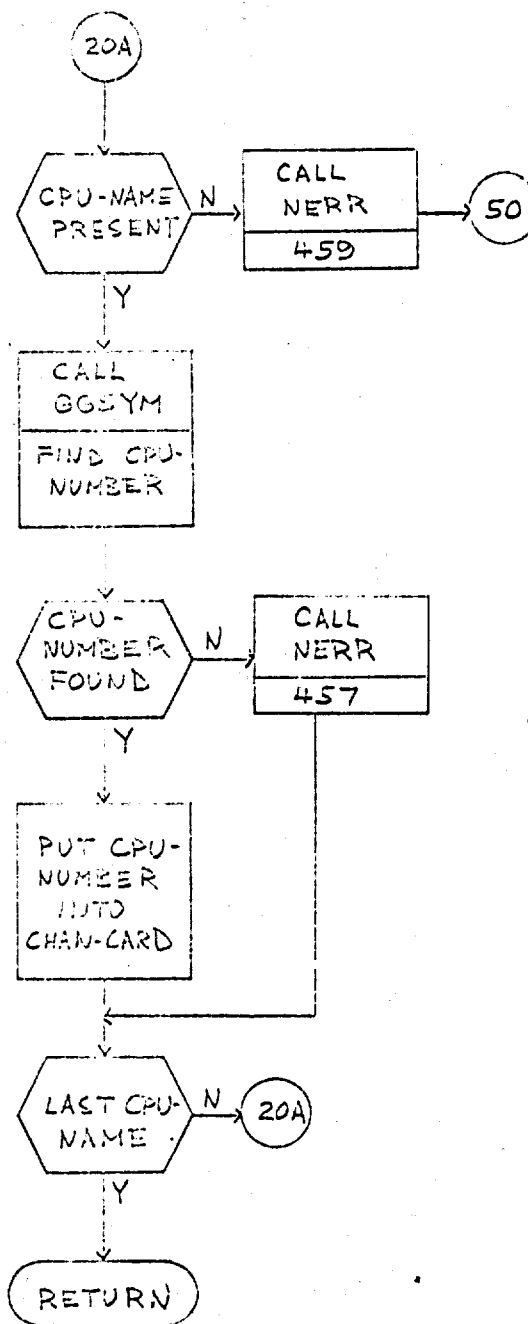
800



SUBROUTINE SX53

This routine processes the channel configuration card. The CHANNEL-ID table is searched to verify that the CHANNEL-ID is valid. A test is made to see that the maximum number of channel names is not exceeded. A test is made to see if the associated CPU name is valid.

SUBROUTINE SX53
(CHANNEL)

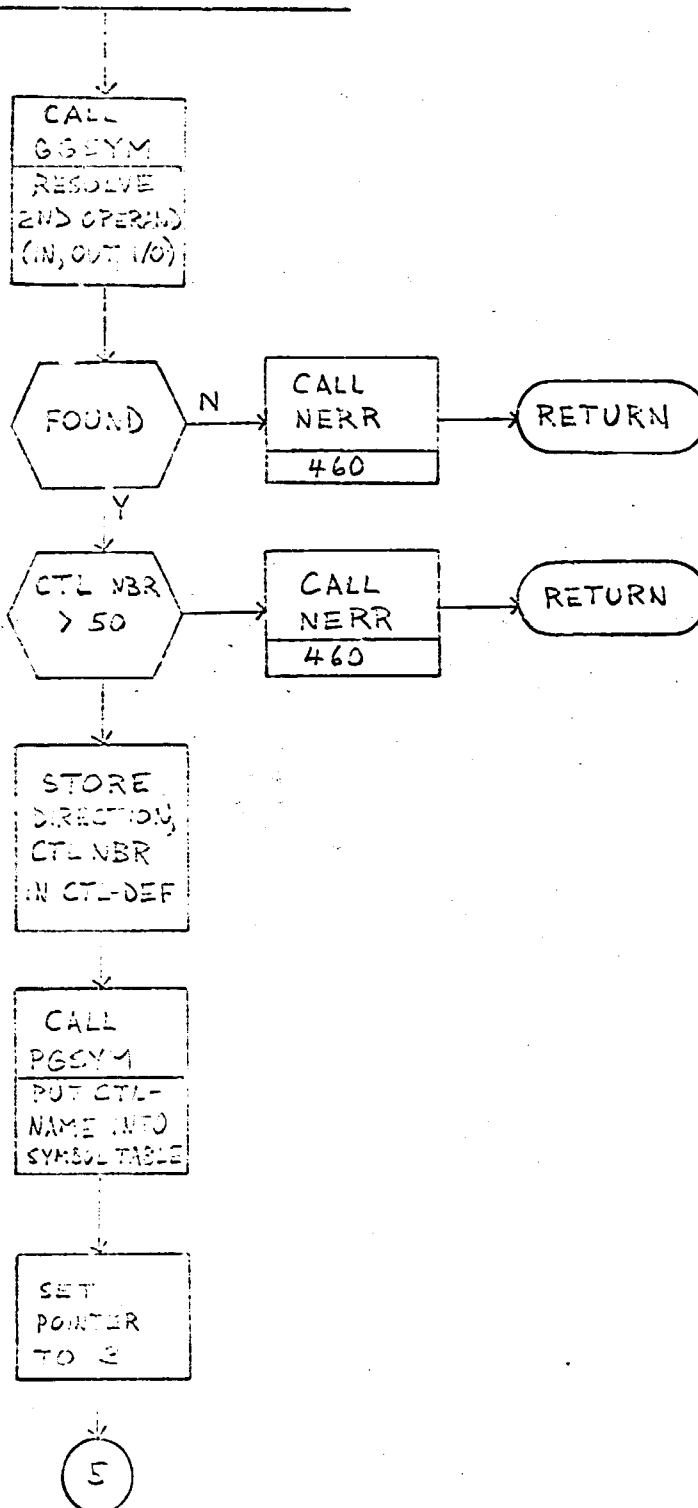


SUBROUTINE SX54

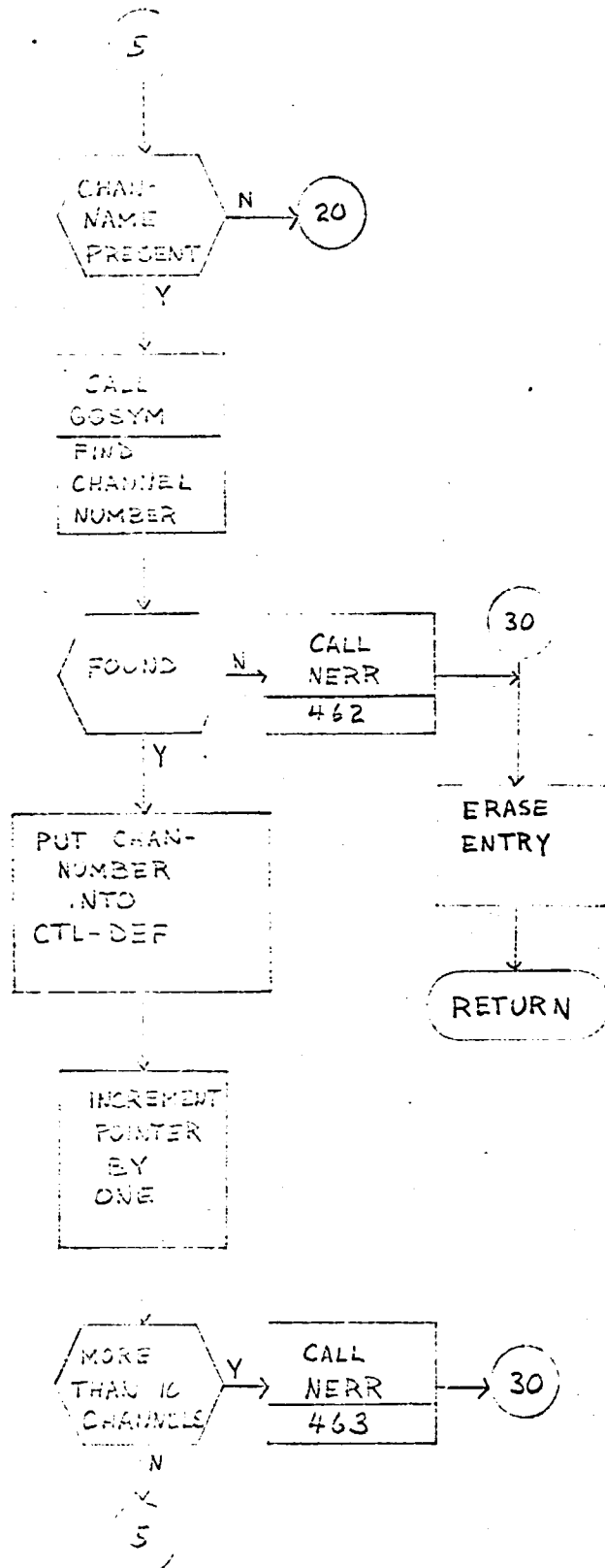
This routine processes the control configuration card. The CTL-ID table is searched to verify that the CTL-ID is valid. A test is made to see that the maximum number of control names is not exceeded. A test is made to see if the associated CPU name is valid.

305

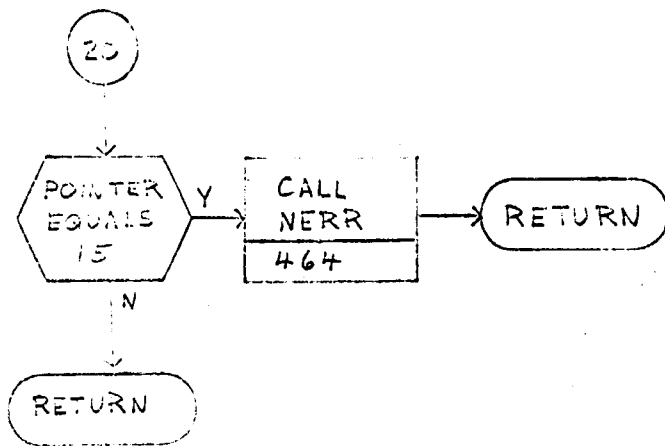
SUBROUTINE SX54-
(CONTROL)



302



307



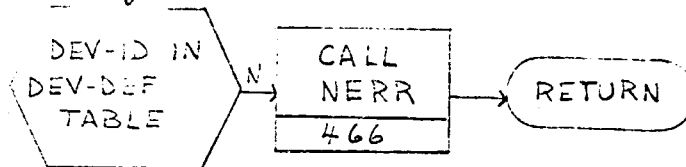
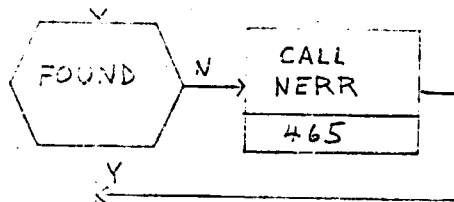
SUBROUTINE SX55

This routine processes the device configuration card. The first step is to resolve the SEIZE, NOSEIZE operand. A check is then made to verify that the device definition is valid. A check is made to see that the number of device definitions does not exceed the maximum. The device name is then placed in the global symbol table. The control names are resolved and appropriate housekeeping is performed.

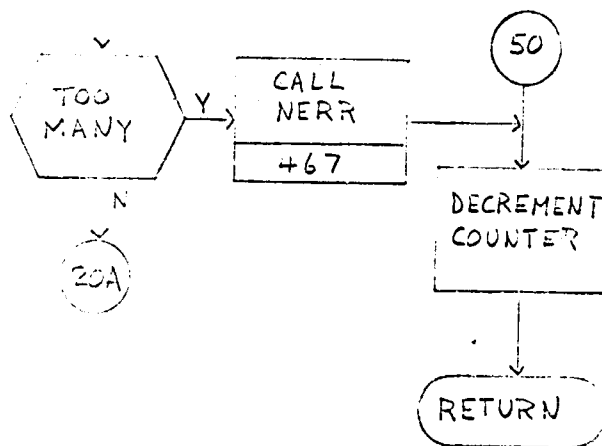
309

SUBROUTINE SX55
(DEVICE)

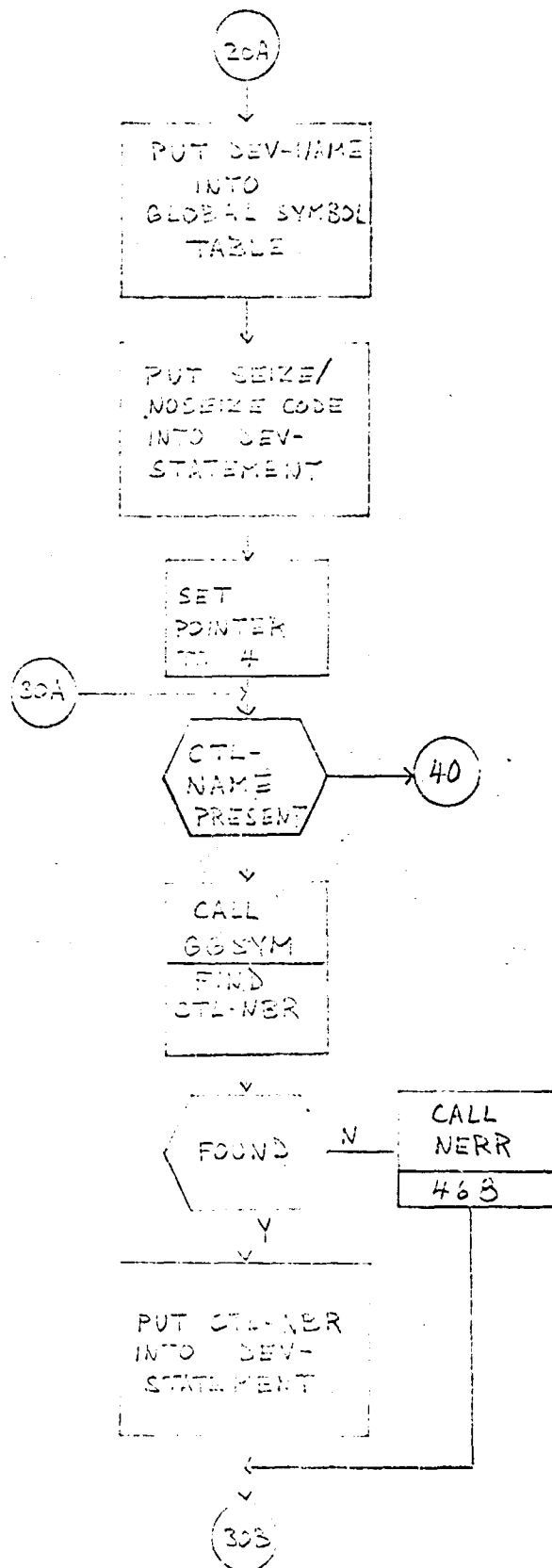
CALL
GGSYM
RESOLVE 2ND
OPERAND
(SEIZE, NO. IZE)



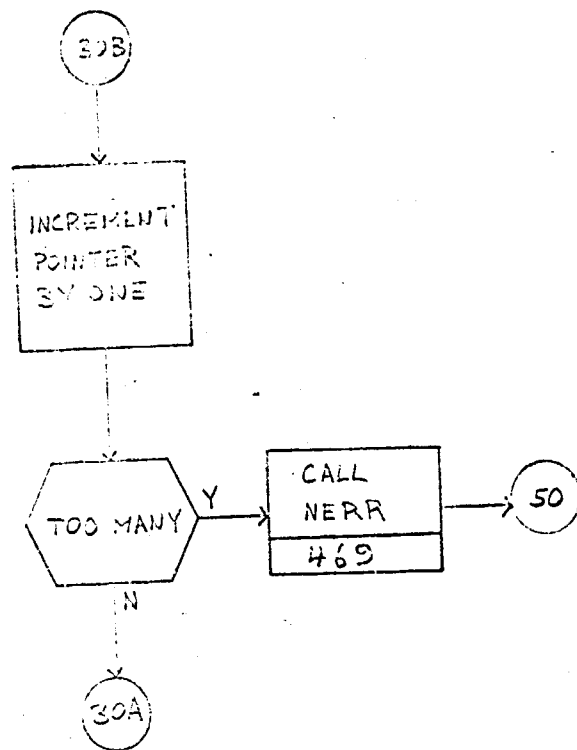
ADD ONE TO
USE COUNTER
FOR DEV-DEF



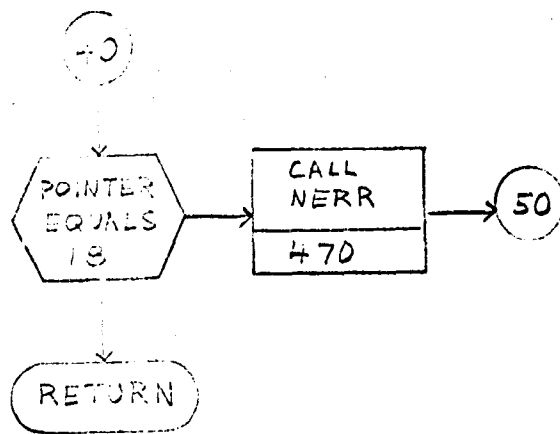
310



316



312



SUBROUTINE SX56

This routine processes the CONFIG-END card. Upon receiving this card, the hardware configuration tables built up by other subroutines are written out onto the simulator input tape. A return is then made to the first pass routine.

314

SUBROUTINE SX56
(CONFIG - END CARD)

↓
WRITE OUT
CPU
STATEMENTS

↓
WRITE OUT
MEMORY
STATEMENTS

↓
WRITE OUT
CHANNEL
STATEMENTS

↓
WRITE OUT
CTL
STATEMENTS

↓
WRITE OUT
DEV
STATEMENTS

↓
10A

315

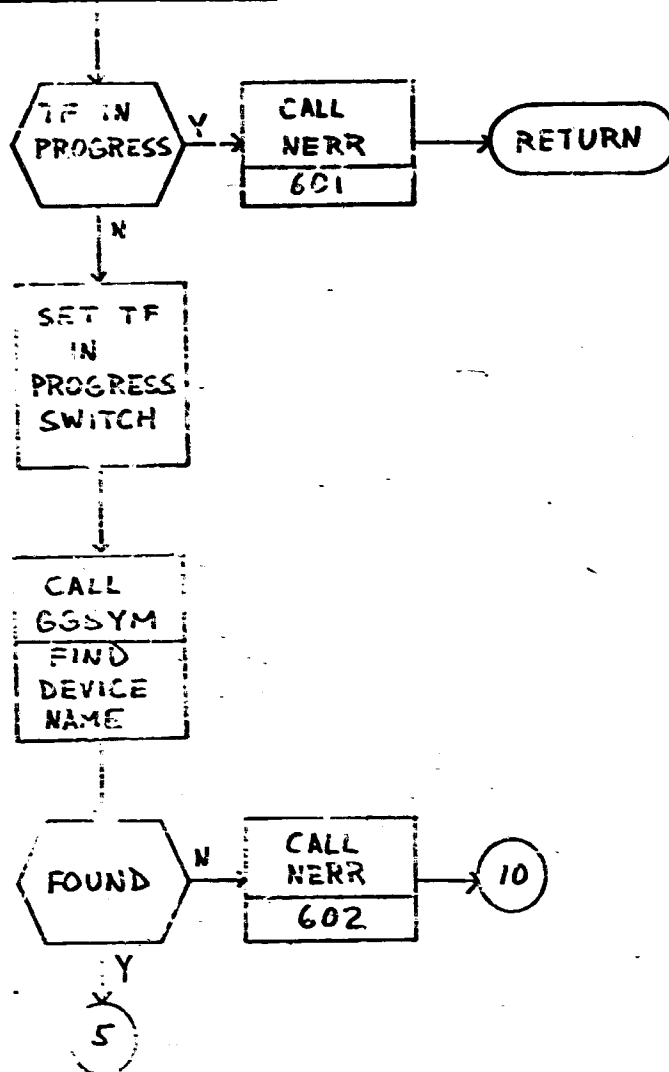


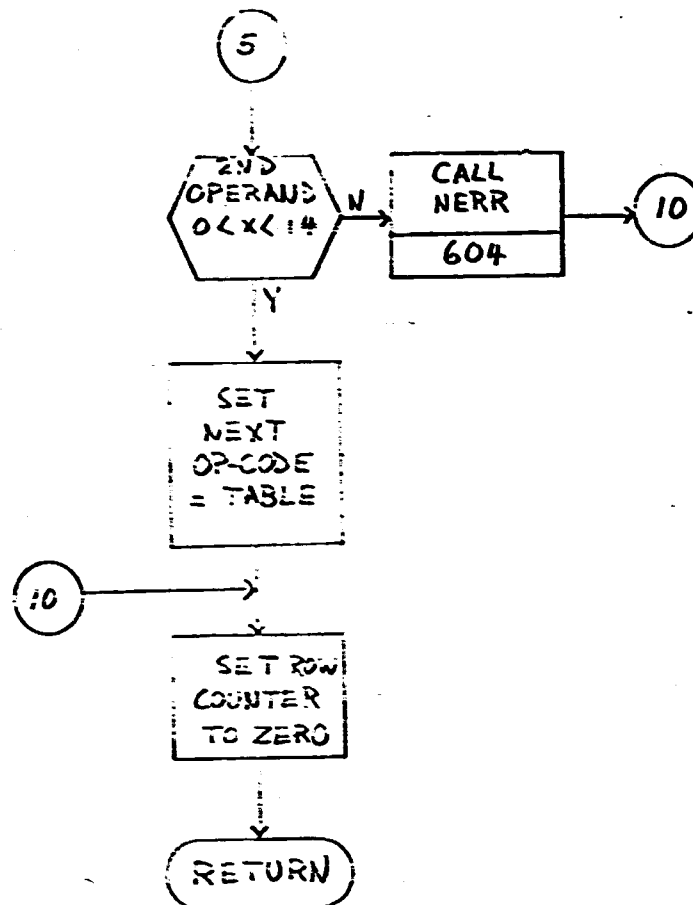
SUBROUTINE SX6

This routine processes the TF definition card. The first step is to test whether a TF definition is already in progress. If it is, an error is declared. If not, a TF definition is declared to be in progress. A check is then made for the device name and number of files. The next anticipated op-code is set to be TABLE. The row counter is then zeroed, and a return is made to the calling program.

317

SUBROUTINE SX6
(TF-DEF CARD)





SUBROUTINE SX7

This routine processes the TABLE statements.

The row counter is incremented, and a test is made to see if there are more rows than files. The output area is then zeroed. The device number, row count, and number of files are stored in an appropriate table. The table entries are then converted using the CONV routine. The table is then written out to the simulator input tape.

370

SUBROUTINE SX7
(TABLE)

ADD ONE
TO ROW
COUNTER

TO-ROW
COUNTER
> NBR OF
FILES

Y

CALL
NERR
606

50

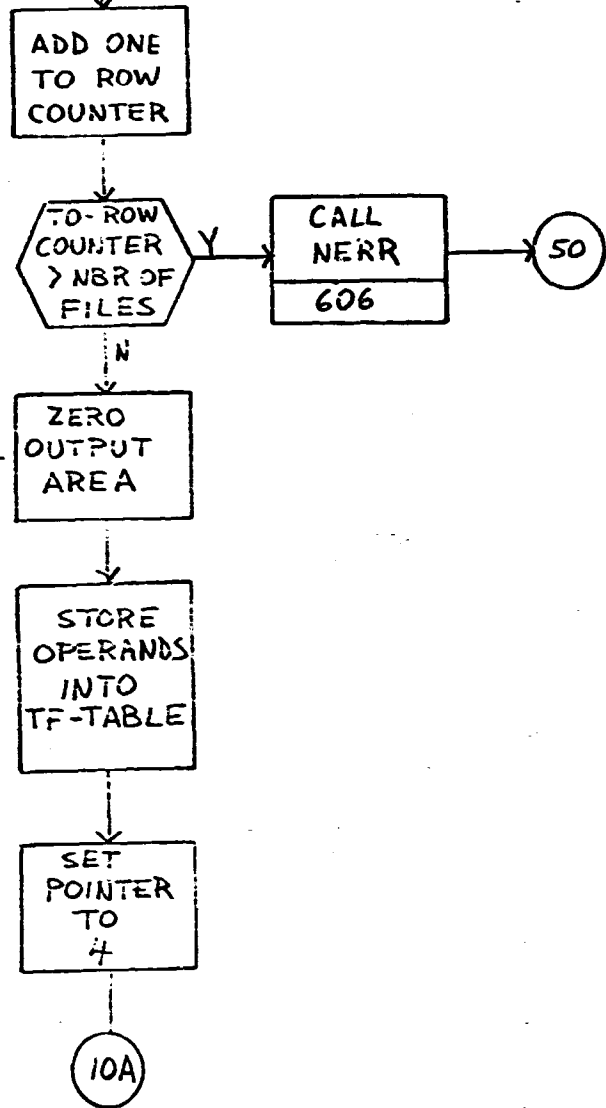
N

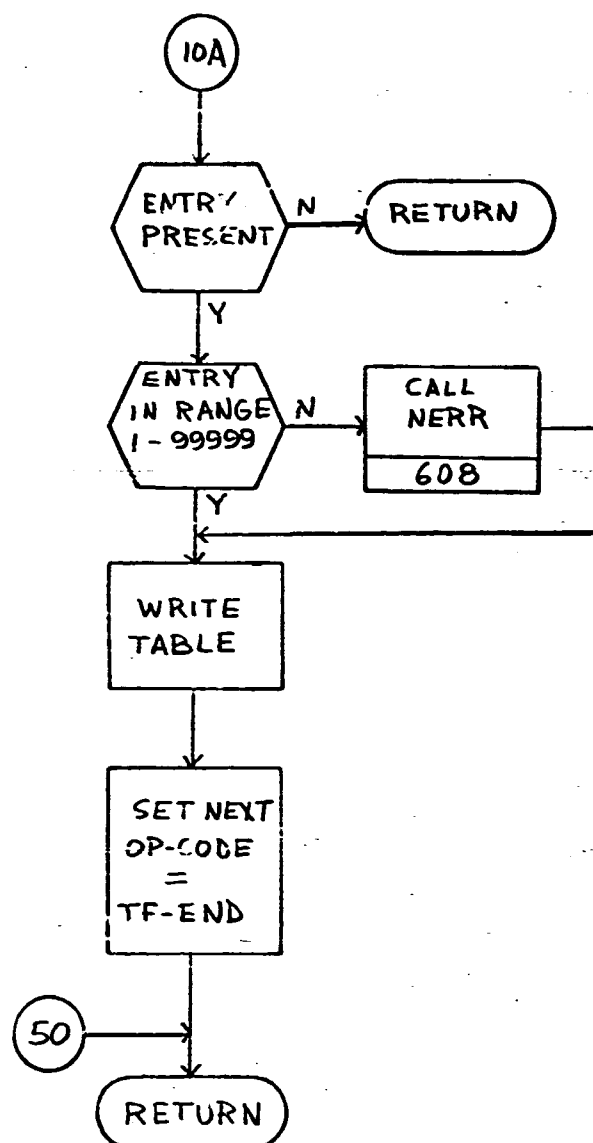
ZERO
OUTPUT
AREA

STORE
OPERANDS
INTO
TF-TABLE

SET
POINTER
TO
4

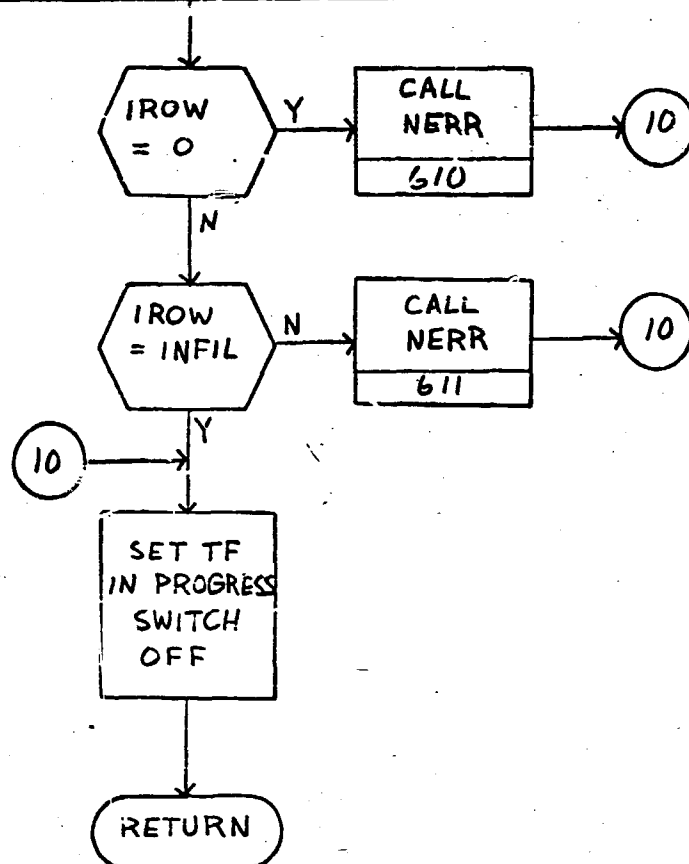
10A





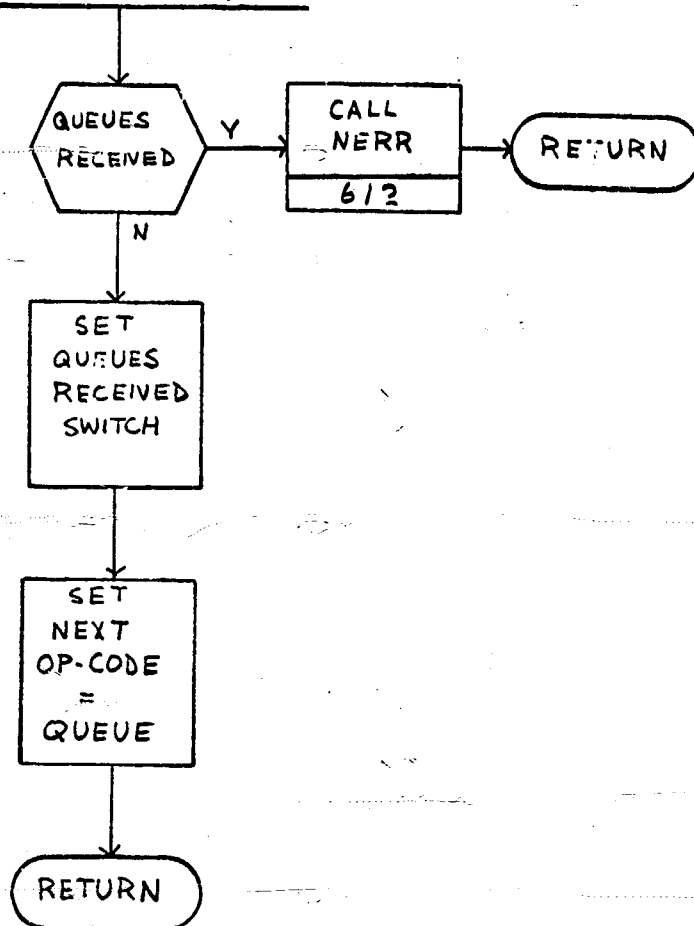
SUBROUTINE SX8

This routine processes the TF-END statement. If no table statements have been received, an error is written. A switch is set to indicate all table statements have now been received. A return is then made to the assembler first pass routine.

SUBROUTINE SX8
(TF-END CARD)

SUBROUTINE SX9

This routine is used to process the Q-DEF statement. It first tests to see if the queues have already been received. If they have, an error is written. If not, a switch is set to indicate that queues may now be received. The next anticipated op-code is set to be the QUEUE statement. A return is then made to the assembler first pass subroutine.

SUBROUTINE SY9
(Q-DEF CARD)

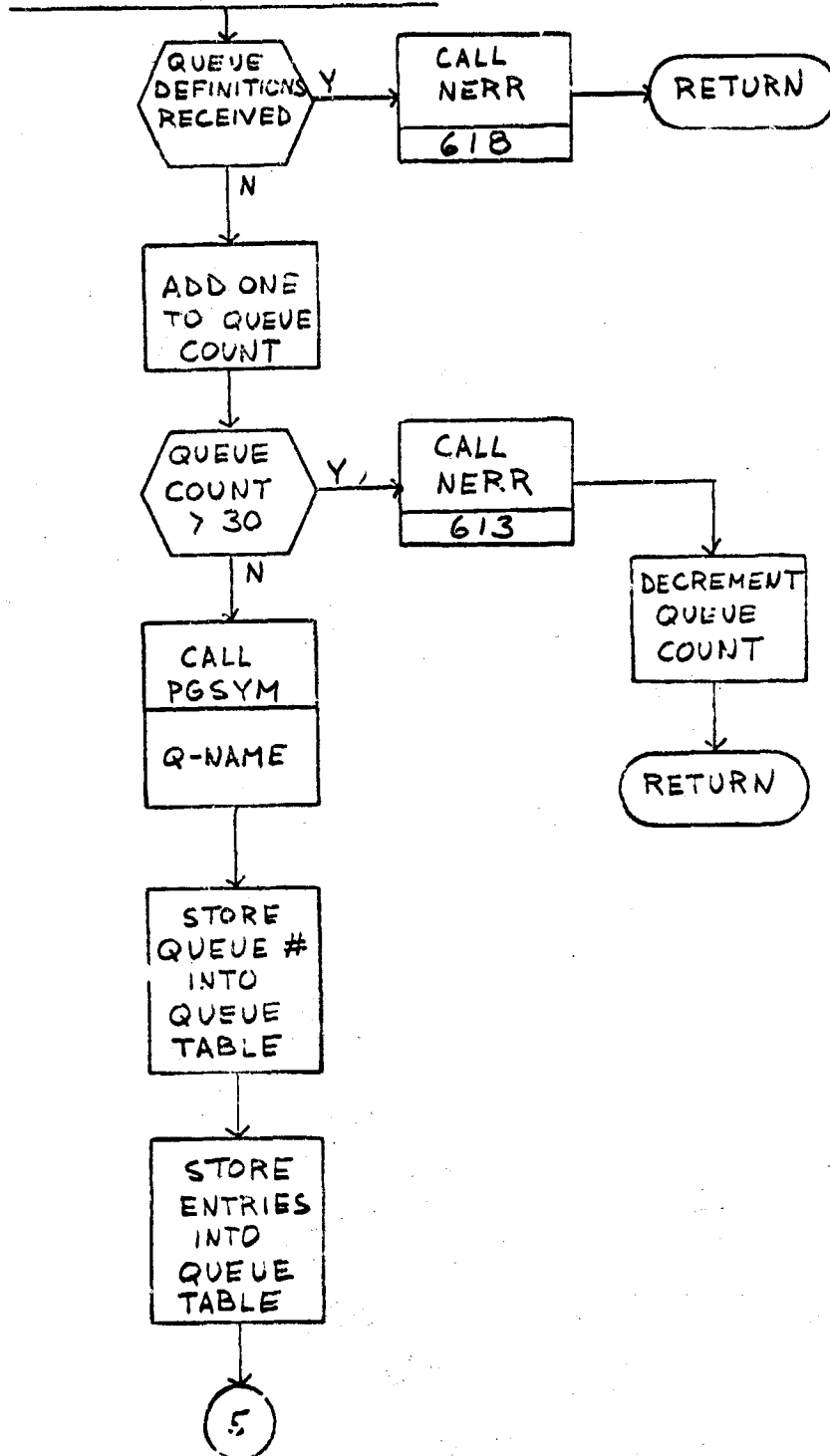
SUBROUTINE SX10

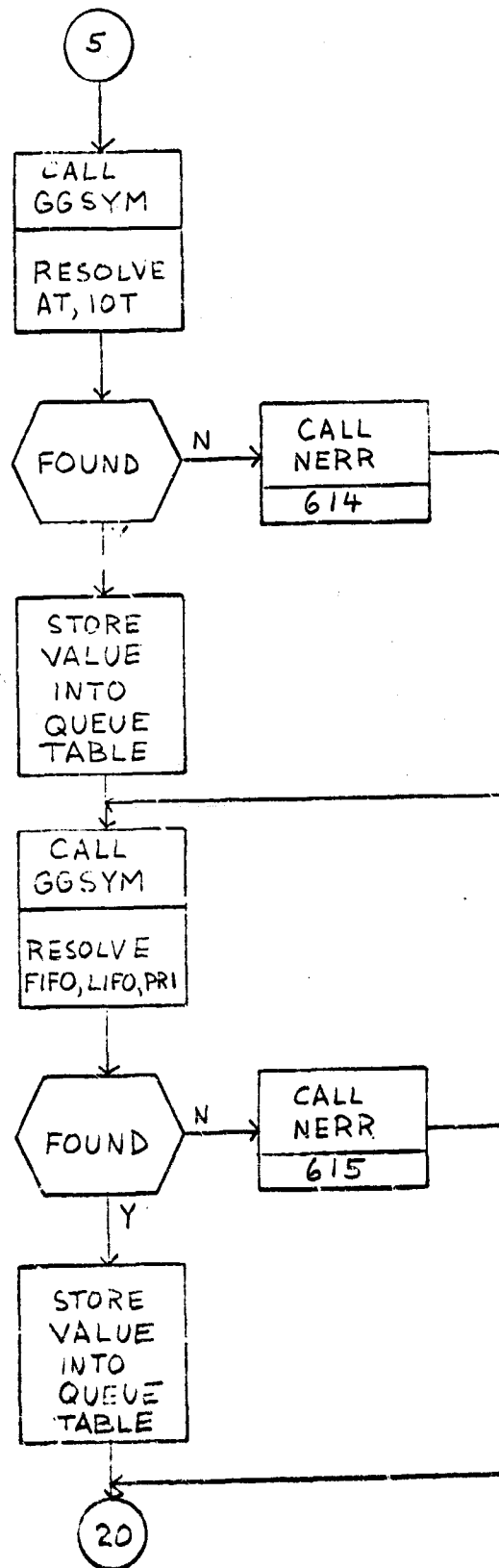
This routine processes the queue statement.

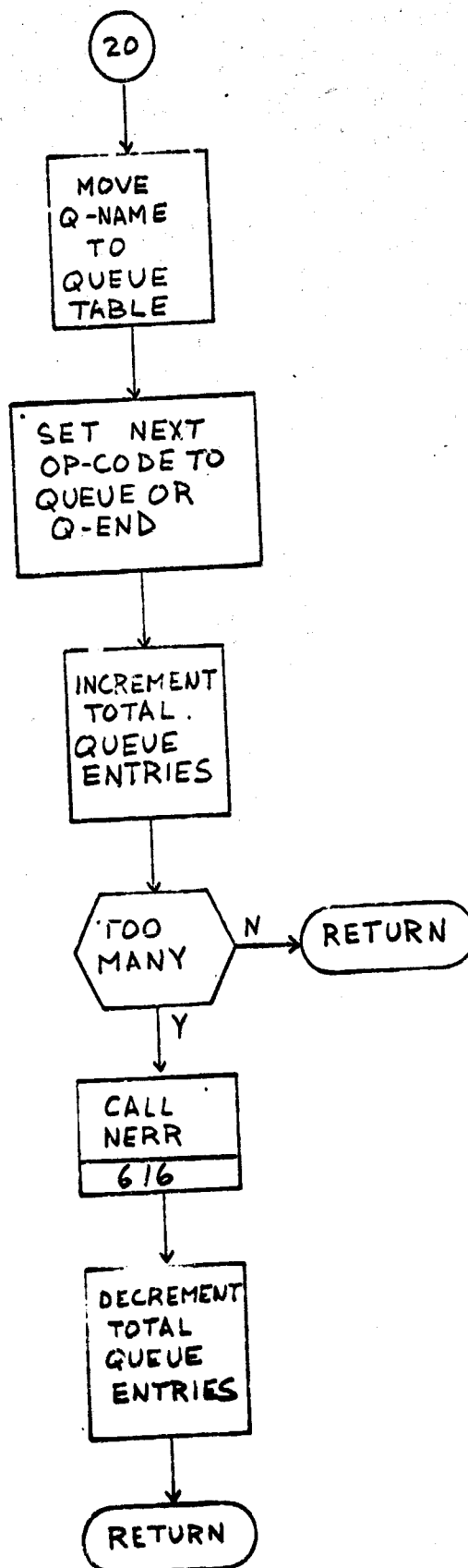
It places a queue name into the global symbol table. After performing some housekeeping functions, a queue name is moved to the queue table. The next anticipated op-code is set to be either another QUEUE or a Q-END statement. A count is made of the total number of queue entries and a test is made to see if the maximum number has been exceeded.

327

SUBROUTINE SX10
(QUEUE)







330

SUBROUTINE SX11

The IQR switch is set to 2 indicating that
all queues have been received. This routine is called
when the Q-END statement is received.

331

SUBROUTINE SX11
(Q-END)

SET
QUEUES
RECEIVED
SWITCH

RETURN

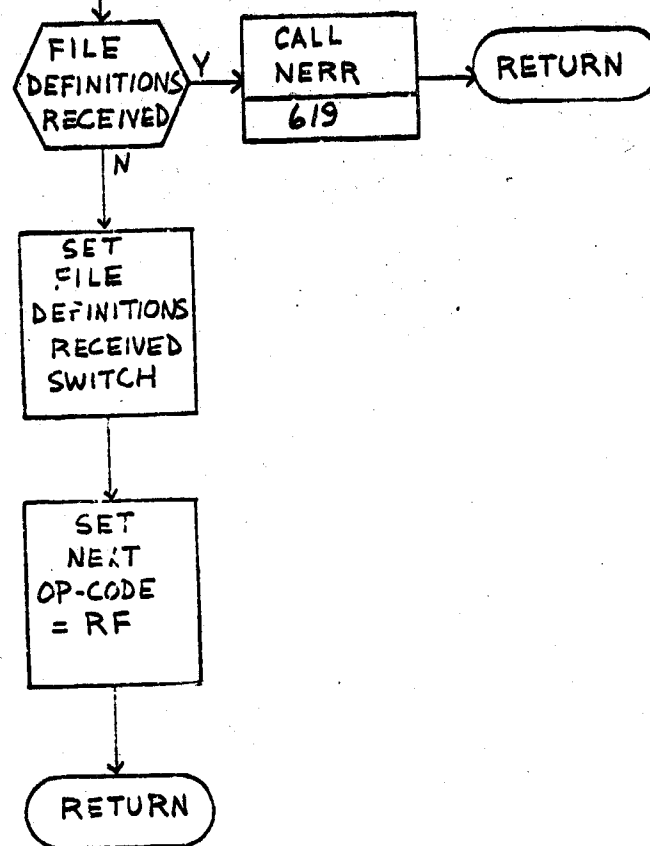
SUBROUTINE SX12

This routine processes the FILE-DEF statement.

A test is made to see if the file definitions have already been received. If they have, an error is declared. If not, a switch is set indicating that files are now to be received. The next op-code switch is set to accept RF statements.

332

SUBROUTINE SX12
(FILE-DEF)

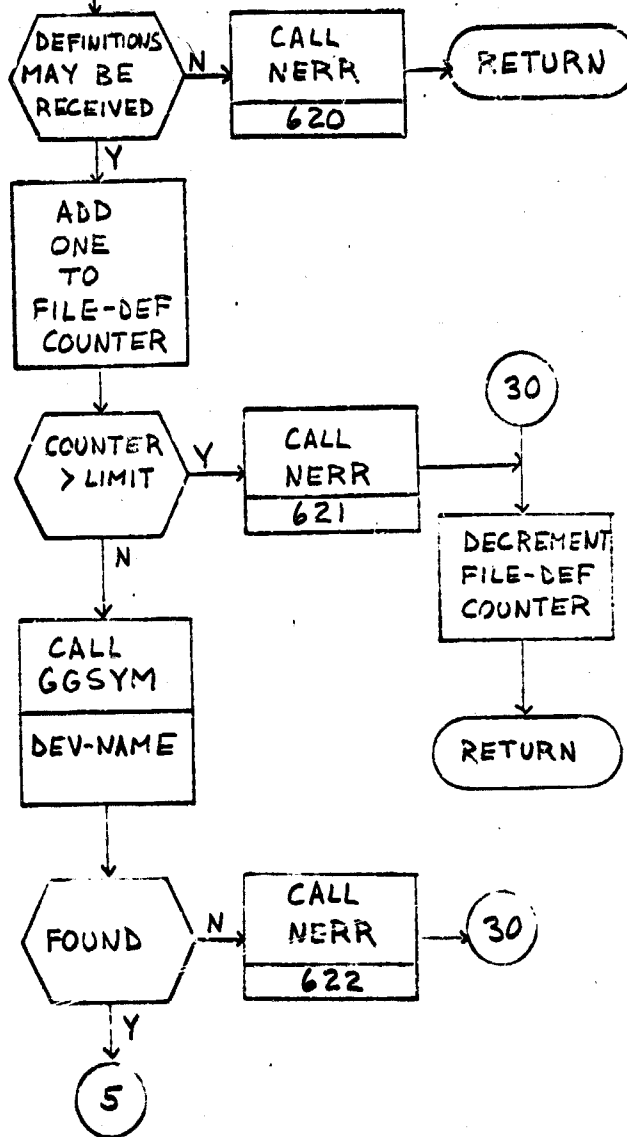


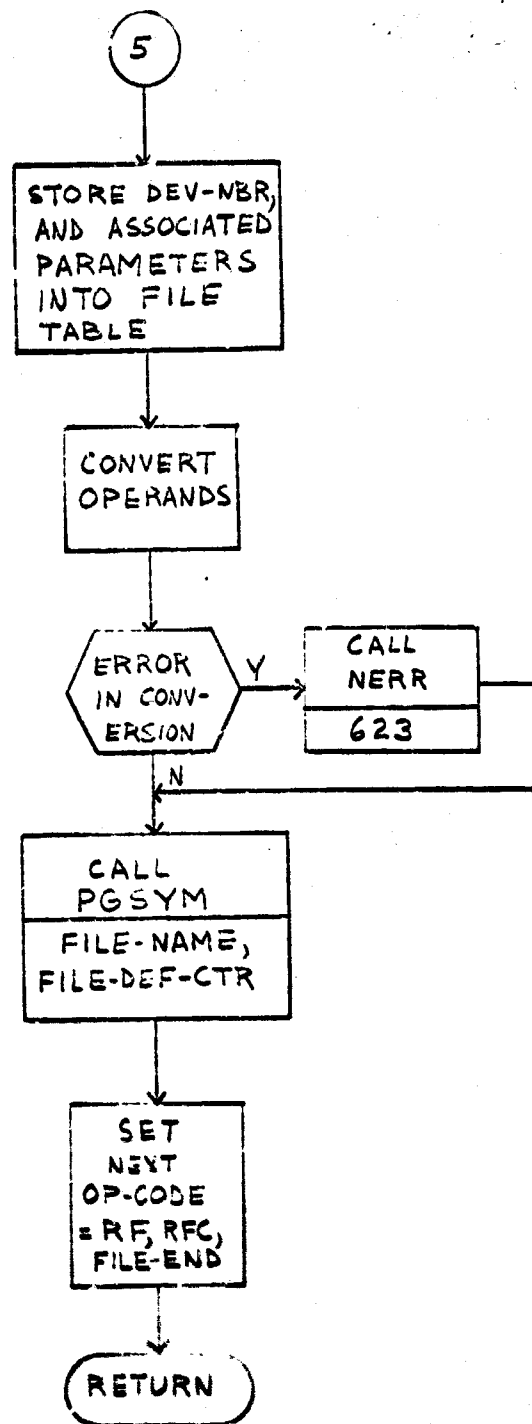
SUBROUTINE SX13

This routine processes RF statements. A test is made to see if file definitions are now in progress. A test is made to see if the maximum number of files is being exceeded. The device name is resolved, the file name and converted operands are stored in an appropriate table. The next op-code is set to be either another RF, RFC, or FILE-END statement.

335

SUBROUTINE SX13
(RF)



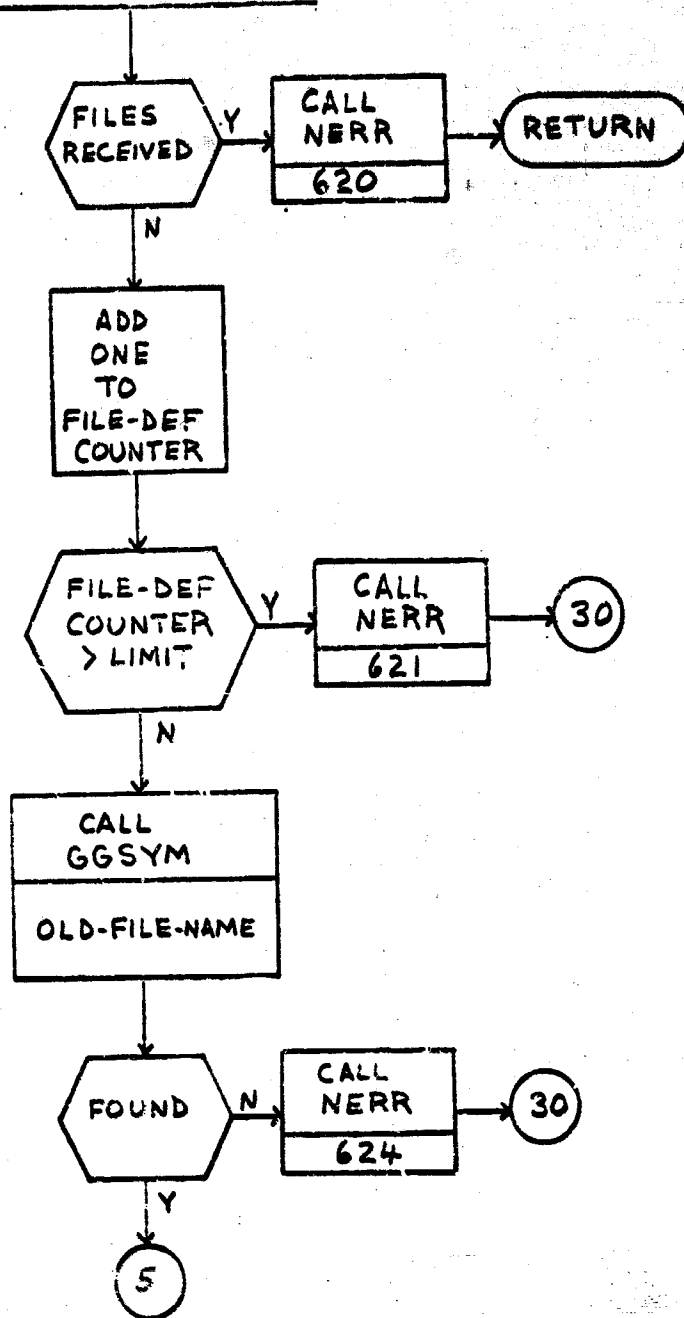


SUBROUTINE SX14

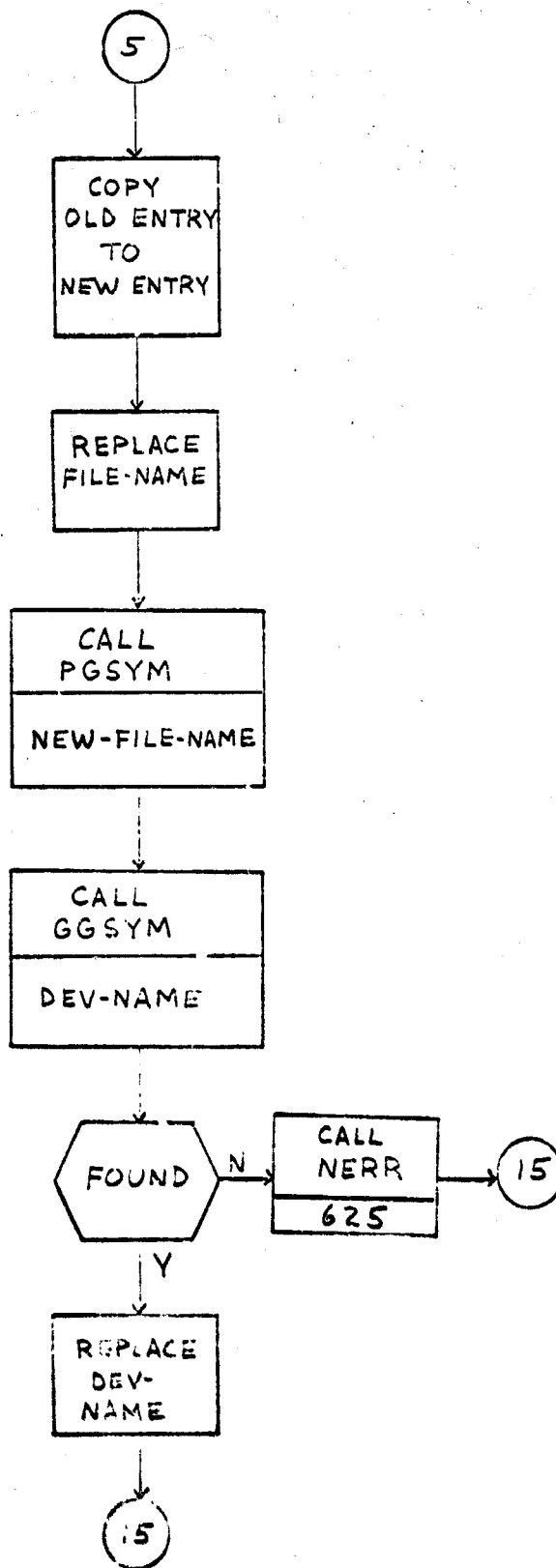
This routine processes the RFC statement. A test is first made to see if the file definition is in progress. A check is made to see that the maximum number of files has not been exceeded. A test is made to see if the c'd file name is valid. The new file name is put into the file table and the global symbol table. The device name is resolved into a device number and stored in the file table. The next expected op-codes are set to be RF, RFC, or file end.

338

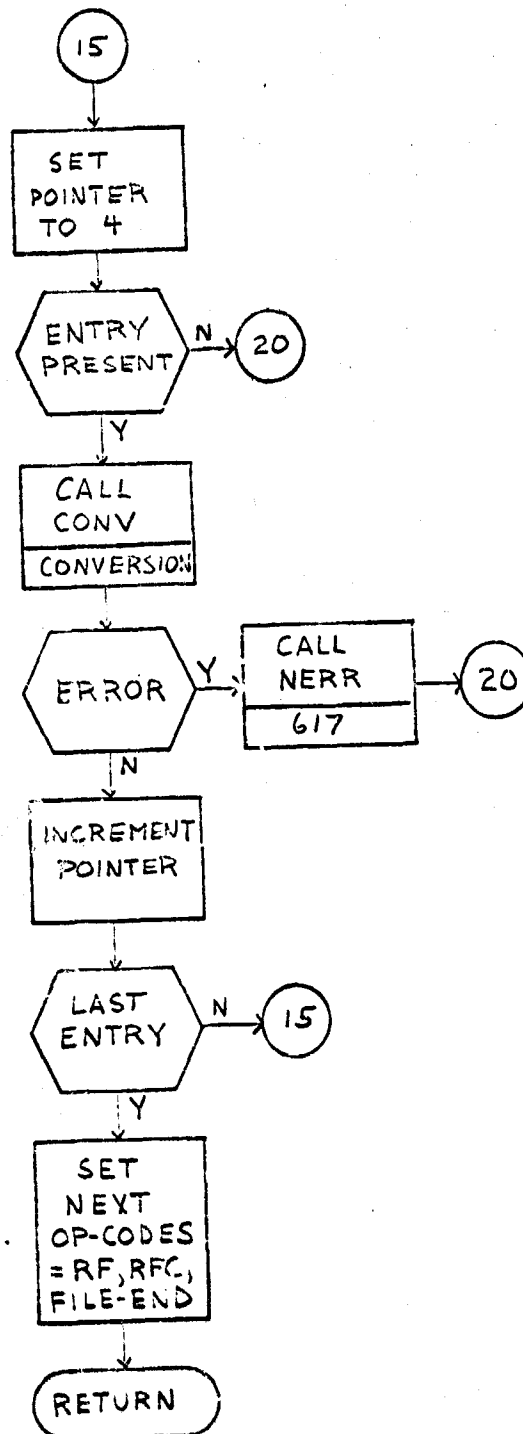
SUBROUTINE SX14
(RFC)



329



340



SUBROUTINE SX15

This routine processes the file end statement. Its functions are to clear the next anticipated op-code area, and to set a switch indicating that file definitions have been received. A return is then made to the first pass subroutine.

342

SUBROUTINE SX15
(FILE-END)

↓
CLEAR
NEXT
OP-CODES

↓
SET
FILES
RECEIVED
SWITCH

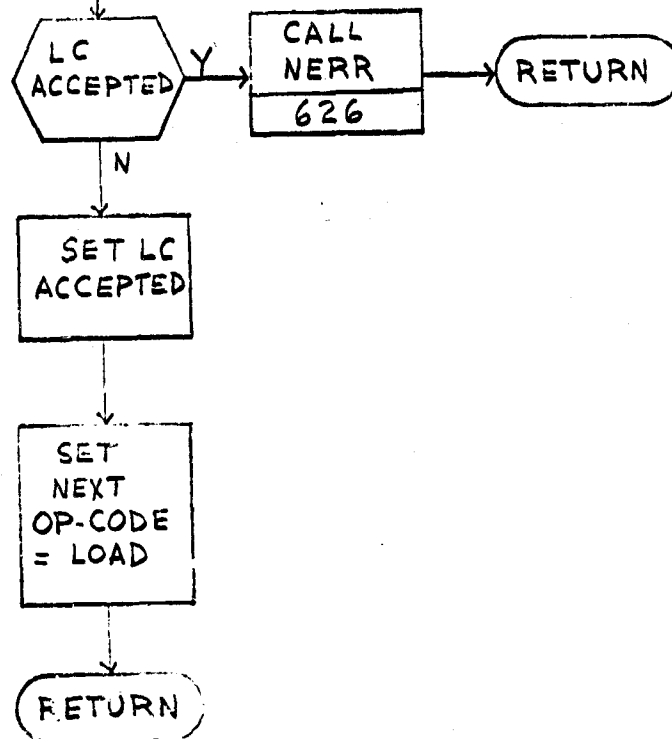
↓
RETURN

SUBROUTINE SX16

This routine processes the load class definition statement. A test is made to see if the load class information has already been received. If it has, an error is written. Otherwise, the load class switch is set to indicate that load class information may now be received. The next anticipated op-code is set to be the LOAD op-code. A return is made to the first pass subroutine.

344

SUBROUTINE SX16
(LC-DEF)

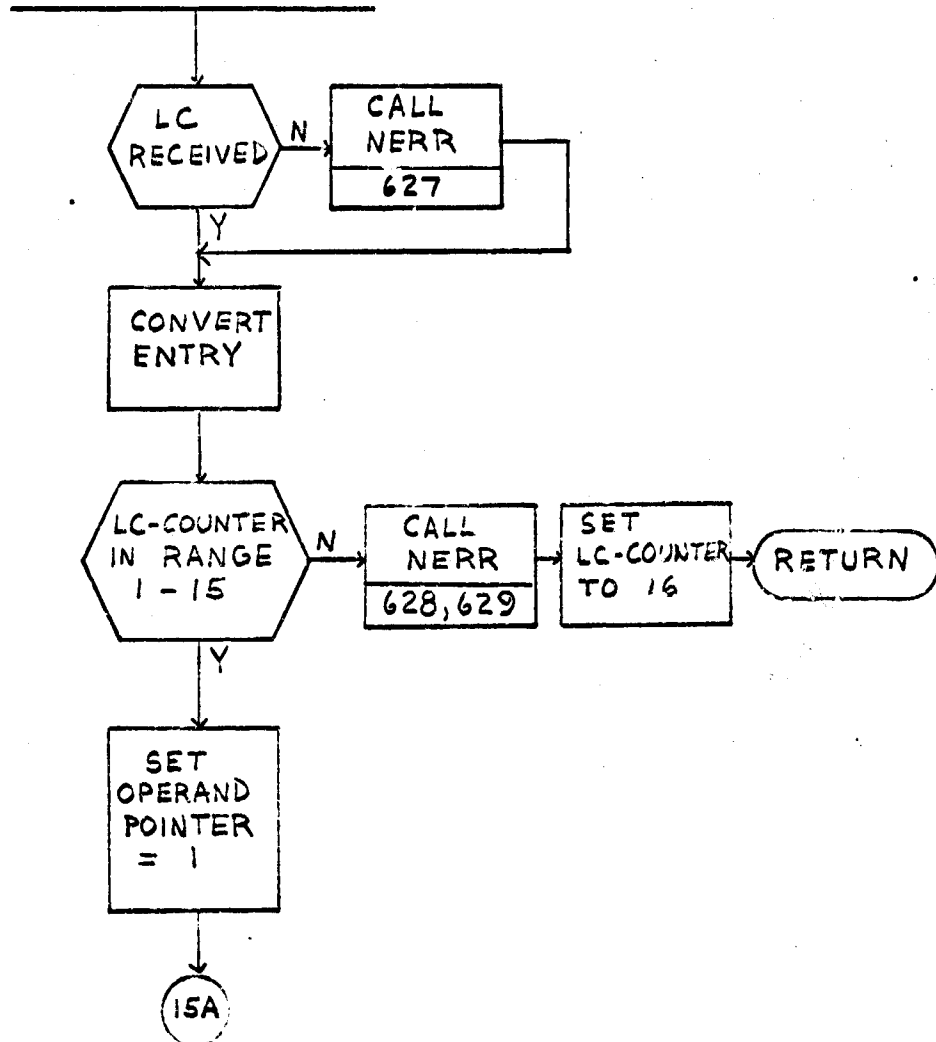


SUBROUTINE SX17

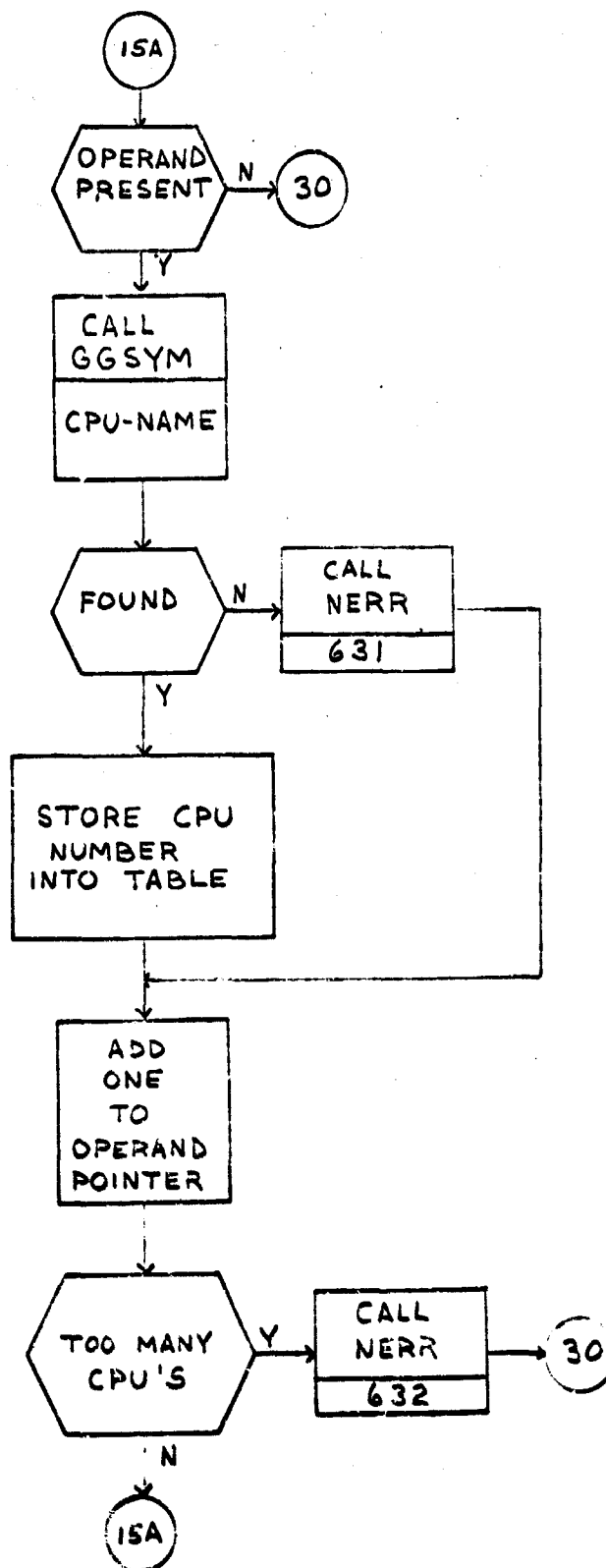
This routine processes the LOAD statement. A test is made to see if the load class information may now be received. The run class identification is then checked to see if it is within range, and if it has already been used. The CPU-IDs associated with this load class are then identified by using the global symbol table. Information is stored in an appropriate table. The next anticipated op-code is set to be either LOAD or LC-END.

346

SUBROUTINE SX17
(LOAD)



347



348

30

SET NEXT
OP-CODE
= LOAD,
LC-END

RETURN

SUBROUTINE SX18

This routine processes the LC-END statement. This routine indicates that the end of the load class data has been received. This is indicated by setting a switch. A return is then made to the first pass sub-routine.

350

SUBROUTINE SX1B
(LC-END)

SET
LC-END
RECEIVED

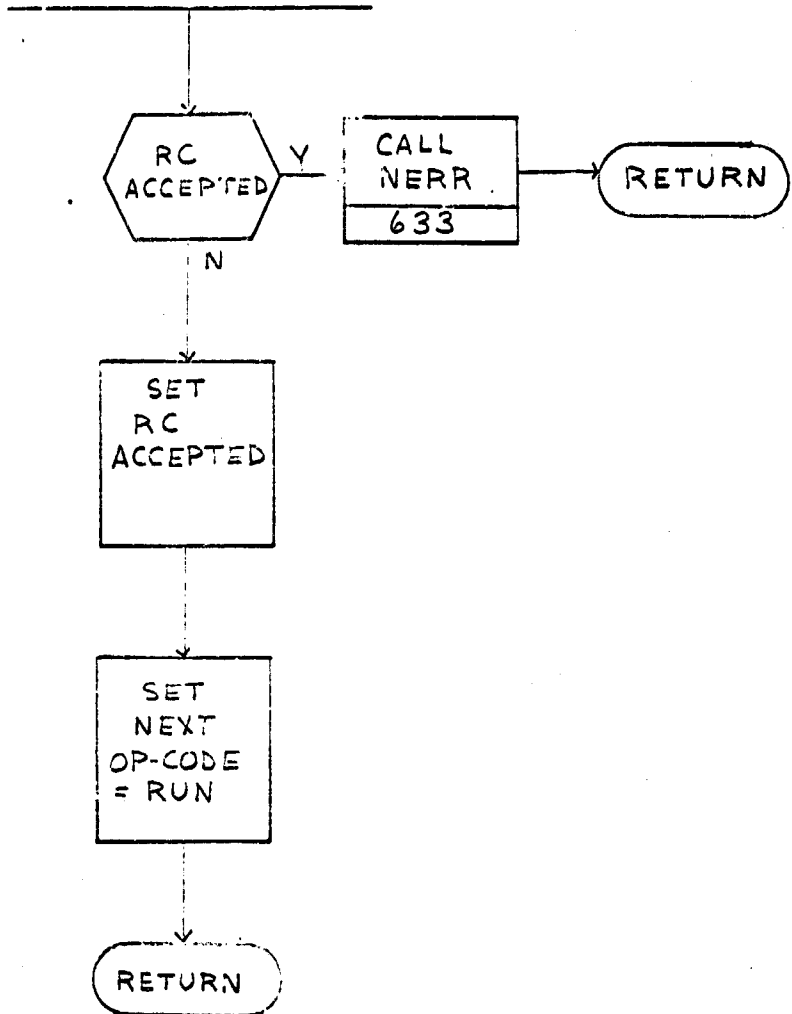
RETURN

SUBROUTINE SX19

This routine processes the run class definition card. A test is first made to see whether the run class information has already been accepted. If it has, then an error is written. Otherwise, a switch is set to indicate that run class definitions may now be received. The next anticipated op-code is set to RUN. A return is made to the first pass subroutine.

352

SUBROUTINE SY19
(RC-DEF)



SUBROUTINE SX20

This routine processes the RUN statement.

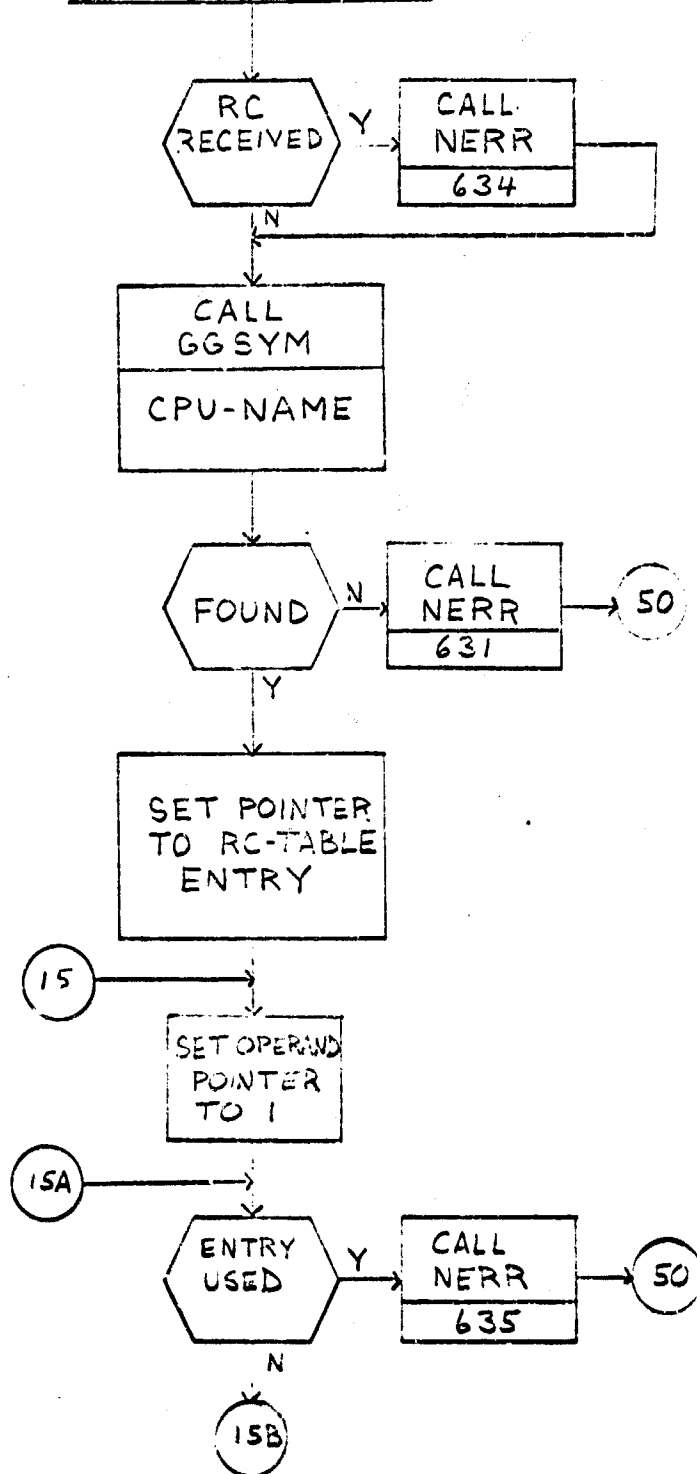
A test is first made to see whether run definitions may now be accepted. If not, an error is written.

The CPU names that are associated with this run card are identified by use of the global symbol table. The identity of the CPUs is stored in an appropriate table.

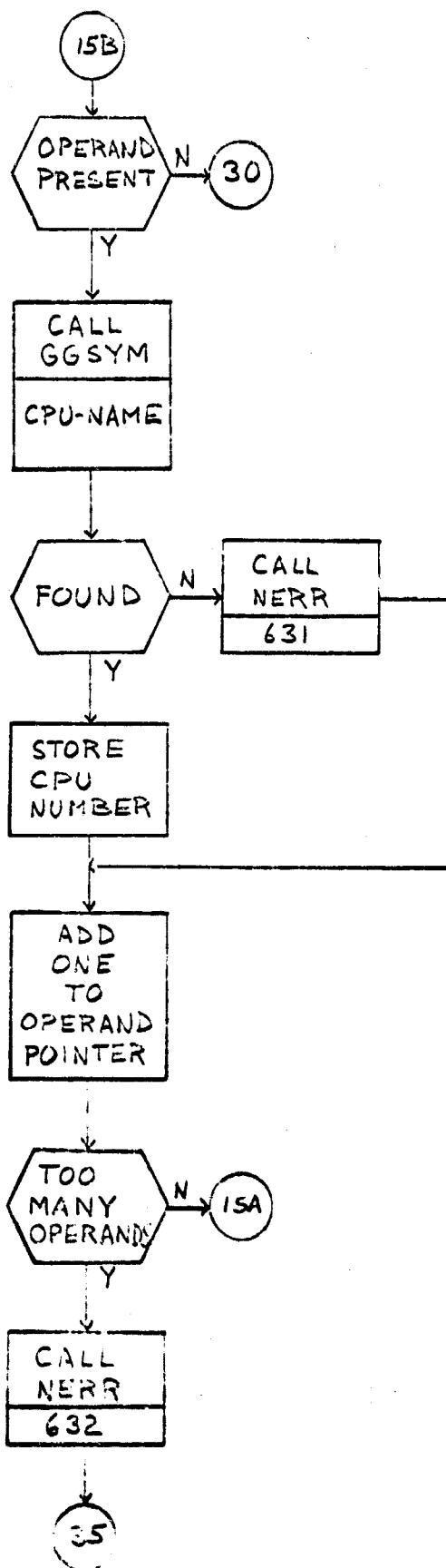
The next op-code is set to be either RUN or RC-END.

354

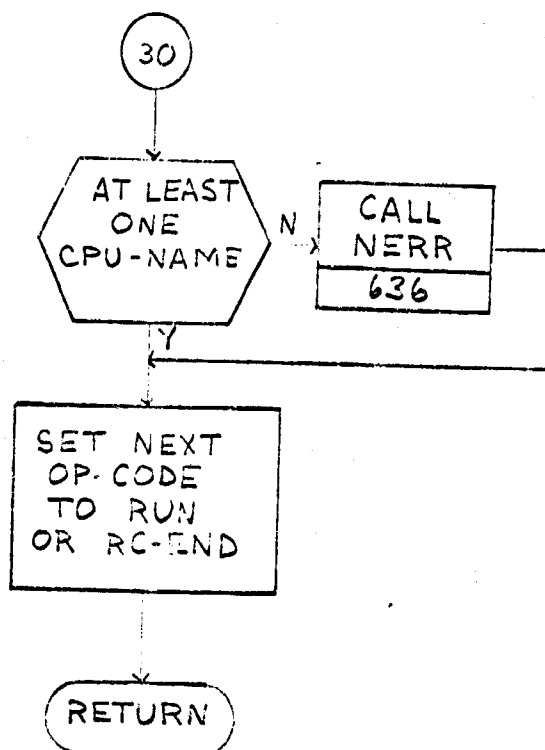
SUBROUTINE SX20
(RUN)



355



356



SUBROUTINE SX21

This routine processes the RC-END statement. This statement denotes the end of the RUN CLASS definition information. This condition is indicated by setting a switch. A return is then made to the first pass subroutine.

358

SUBROUTINE SX21
(RC-END)

SET
RC-END
RECEIVED
SWITCH

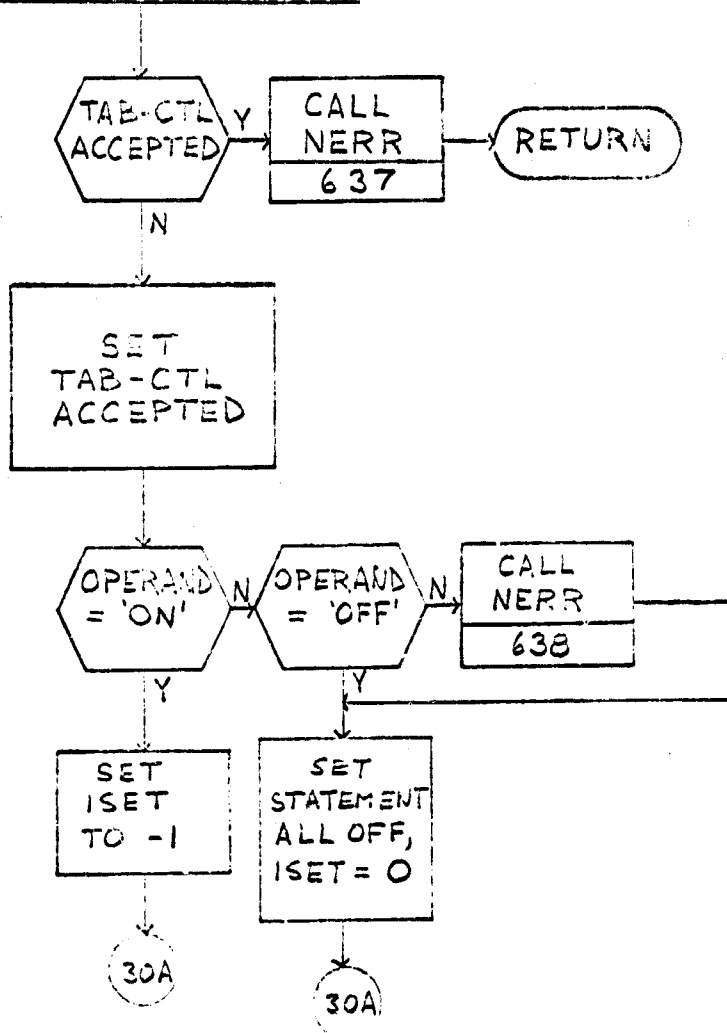
RETURN

SUBROUTINE SX22

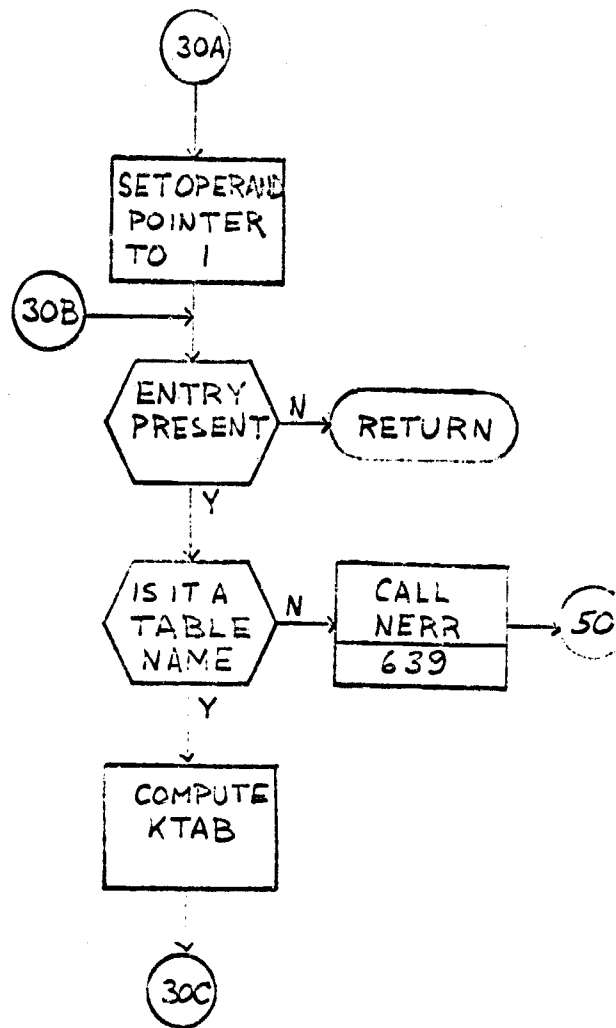
This routine processes the table control statement. A check is first made to see whether or not a table control statement has already been accepted. If it has, an error is written. If not, a switch is set to indicate that the table control statement has been accepted. A test is made to see whether or not a normal condition is ON or OFF. Subsequent operands, denoting tables, are identified and their print status is set. A return is made to the first pass subroutine.

360

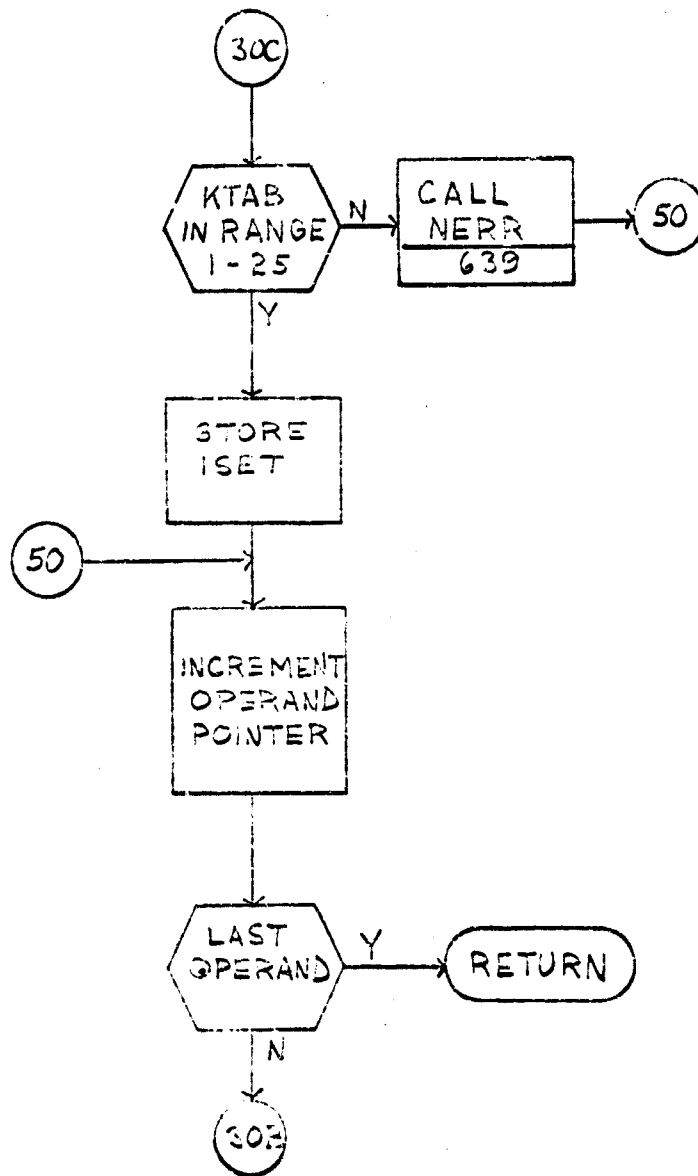
SUBROUTINE SX22
(TAB-CTL)



361



362

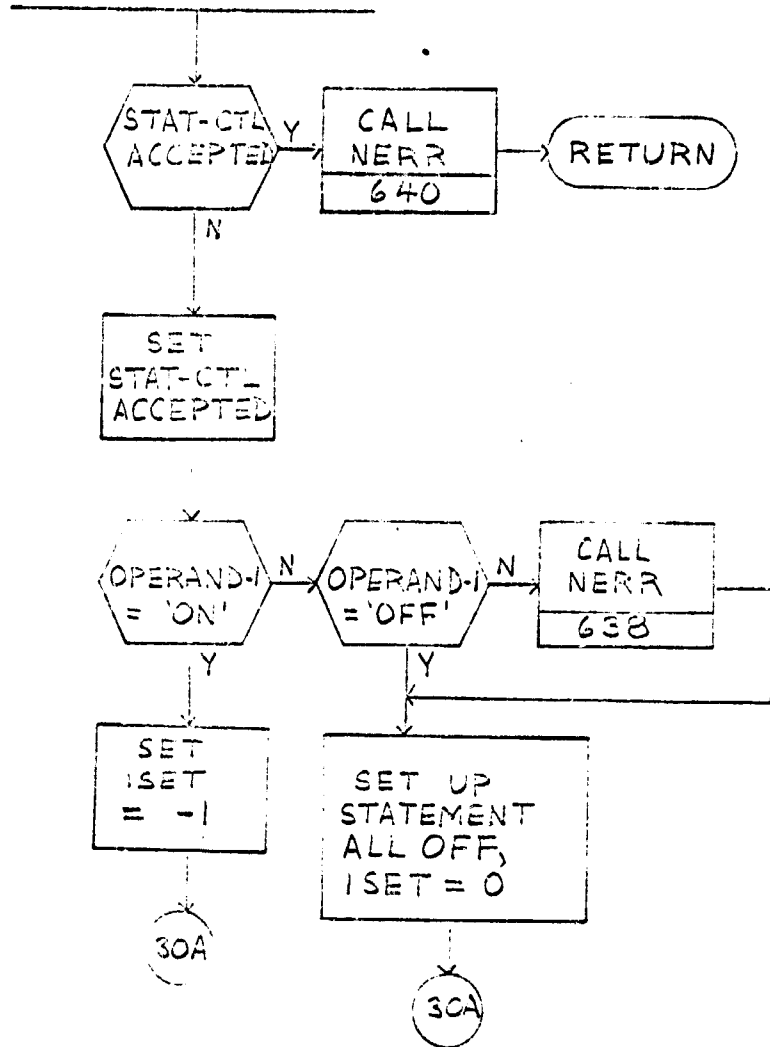


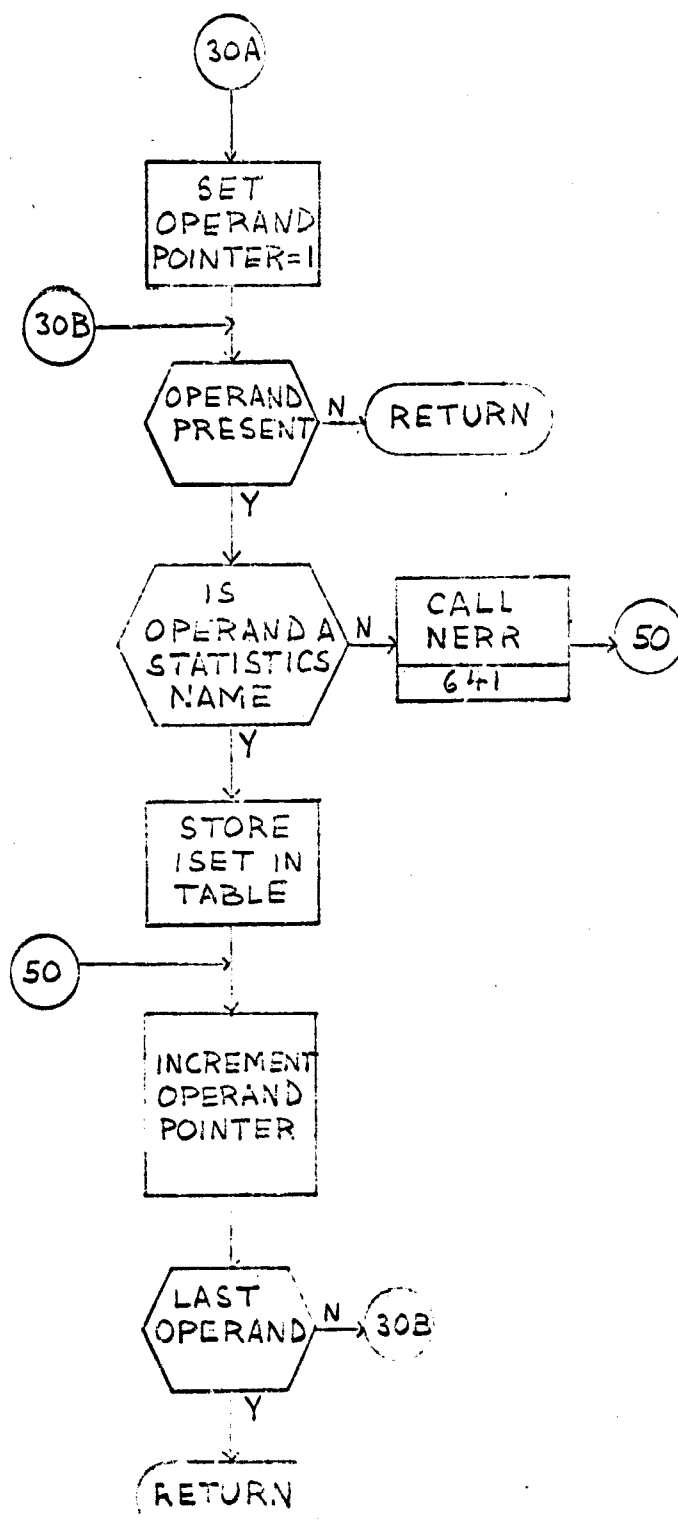
SUBROUTINE SX23

This routine processes the statistics control statement. A test is first made to see whether or not the statistics control statement has already been received. If it has, an error condition is raised. If not, a switch is set to indicate that the statistics control statement has been received. A test is then made to see whether the normal condition for printing out a statistics table is ON or OFF. Subsequent operands, denoting statistics tables to be printed, are identified and their print status is set. A return is then made to the first pass subroutine.

364

SUBROUTINE SX23
(STAT-CTL)



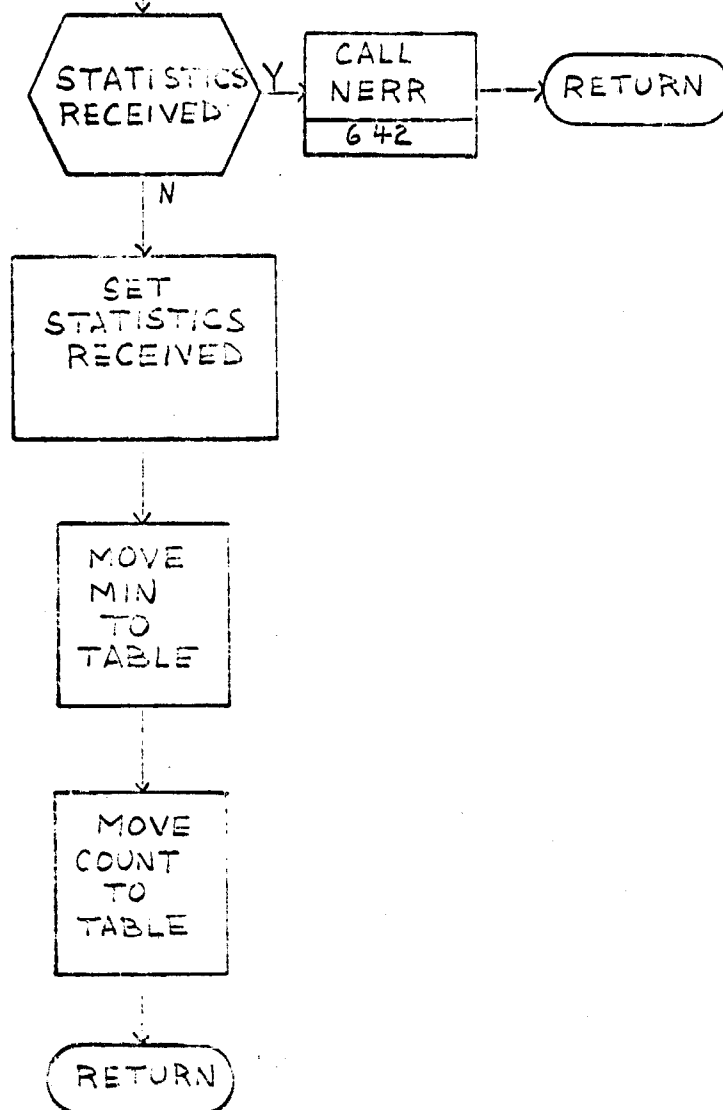


SUBROUTINE SX24

This routine processes the STATISTICS statement. This routine obtains the first and second operands denoting the statistics interval and the number of such intervals to be used during this simulation, and stores them in appropriate fields. A return is then made to the calling subroutine.

367

SUBROUTINE SX24
(STATISTICS)

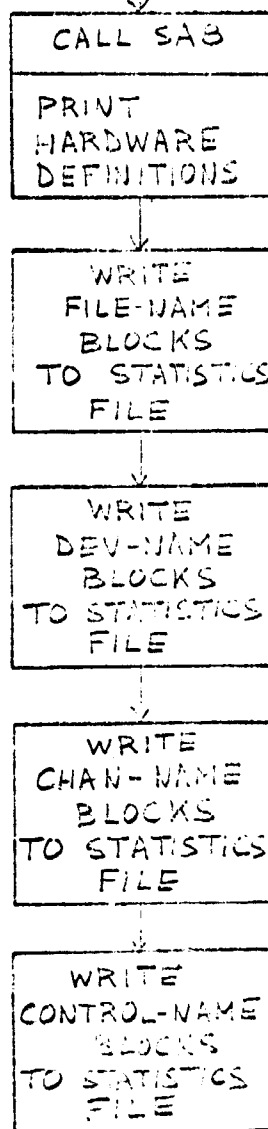


SUBROUTINE SX25

This routine processes the ASSEMBLY statement. It first calls SA8 which is used to print the hardware definitions. It then writes several blocks of data to the statistics file. These blocks contain file names, device names, channel names, control name blocks, queue name blocks, and memory name blocks. Common is then cleared. The program distribution table is then initialized. The load class and run class tables are also initialized. The assembly switch is set OFF to permit processing of worker program routines. A test is then made to see whether the operand in the assembly statement was NOPRINT, and the appropriate print switch setting is made. A return is then made to the first pass subroutine.

369

SUBROUTINE SX25
(ASSEMBLY)



110

370

110

WRITE
QUEUE-NAME
BLOCKS TO
STATISTICS
FILE

WRITE
MEMORY-NAME
BLOCKS TO
STATISTICS
FILE

CLEAR
COMMON
FOR
PHASE 2
AND 2+3

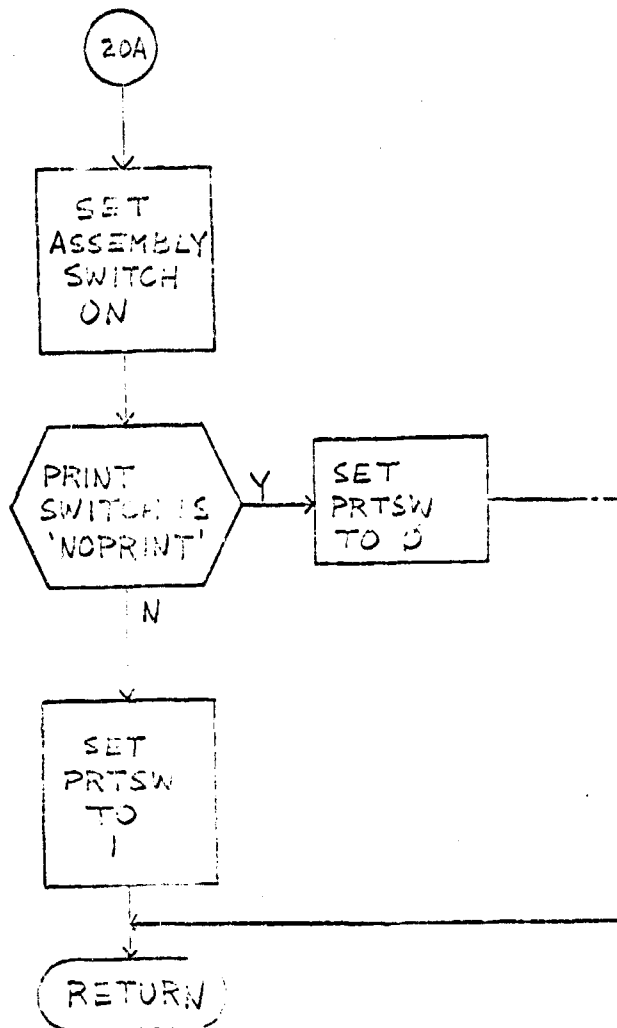
INITIALIZE
PROGRAM
DISTRIBUTION
TABLE

INITIALIZE
LOAD-CLASS
TABLE

INITIALIZE
RUN-CLASS
TABLE

20A

375



SUBROUTINE SX26

This routine processes the TITLE statement. The next card is read from the input stream and its contents stored in the page title area of memory. The printer is then advanced to the top of a new page and the title printed out there. A return is then made to the first pass subroutine.

373

SUBROUTINE EXR6
(TITLE CARD)

READ
NEXT
CARD

MOVE
TITLE
TO
HEADING

SET
NEW
PAGE
SWITCH

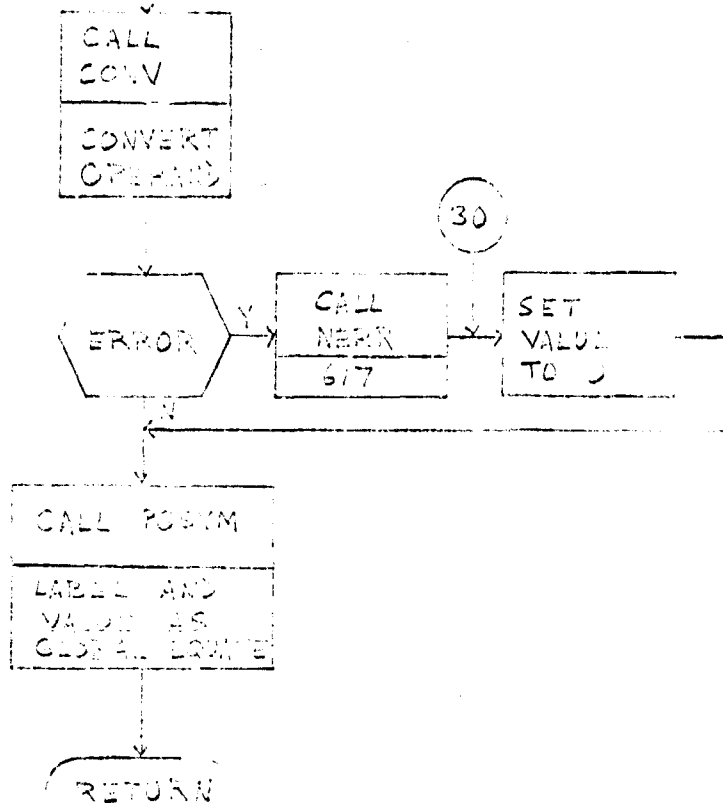
RETURN

SUBROUTINE SX27

This routine processes the GEQU statement. This statement is used to define global equates. The first operand is converted to internal binary form, and the statement label and this value are stored in the global symbol table as a global equate. A return is then made to the calling subroutine.

375

SUBROUTINE SX27
(GENU CARD)

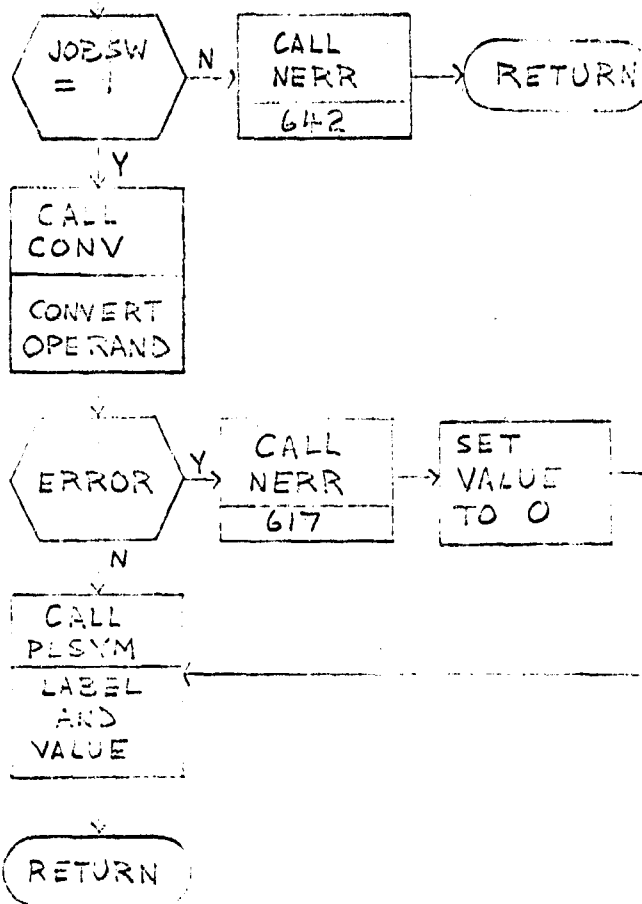


SUBROUTINE SX28

This routine processes the LEQU statement. A test is first made to see whether a job statement has preceded this LEQU statement. If not, an error is written. Otherwise, the first operand is converted to internal binary form, and this value and the statement label are stored in the local symbol table as a local equate. A return is then made to the first pass subroutine.

377

SUBROUTINE SX28
(LEQU CARD)

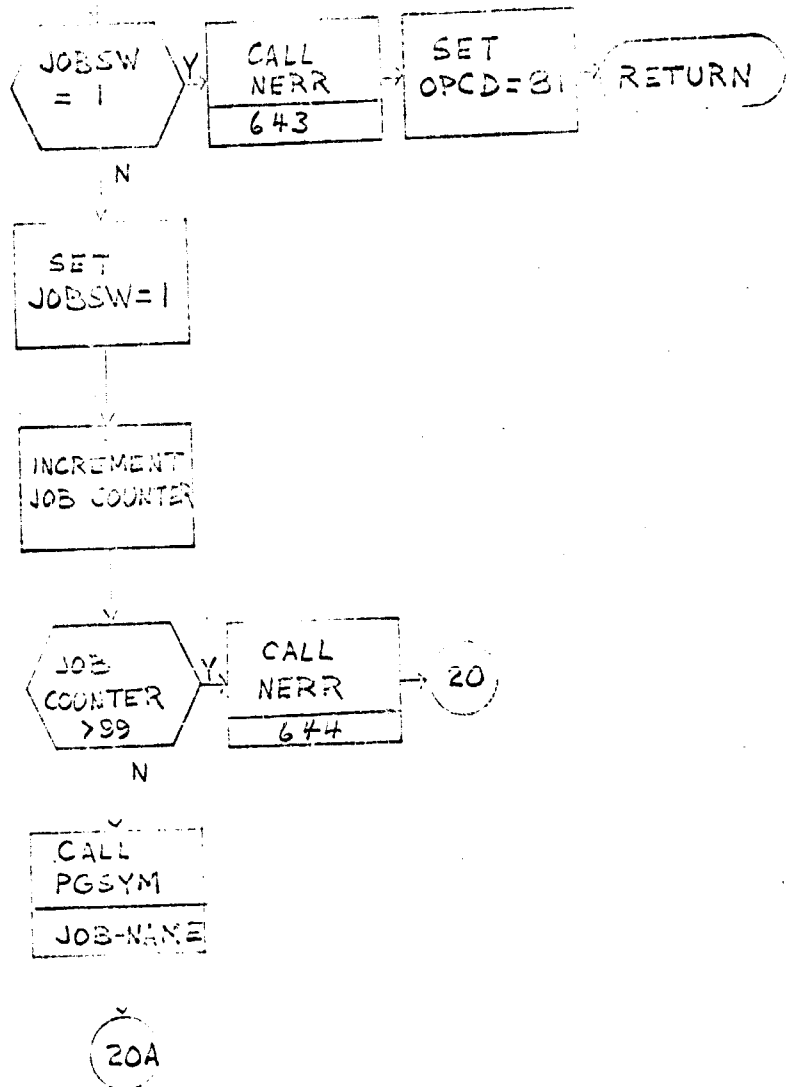


SUBROUTINE SX29

This routine processes the JOB statement. A test is made to see whether this statement has occurred in the middle of another job. If it has, an error is written. If not, a switch is set to indicate that a job has been begun. The number of jobs is then incremented by one. A test is then made to see whether the number of jobs received has exceeded the maximum. The job name is stored in the global symbol table. Some internal housekeeping is then performed. The local job statement number is then set to zero. A return is then made to the first pass subroutine.

379

SUBROUTINE SX29
(JOB CARD)



200

20A

STORE 102
COUNTER IN
SECOND HALF
OF THE SECOND
OPERAND

SHIFT 2ND
AND 3RD
OPERANDS
TO 1ST AND
2ND

SET LOCAL
STATEMENT
NUMBER
COUNTER
TO ZERO

20

SUBROUTINE SX30

This routine processes the OF statement. The number of OF statements received is incremented by one. The OF name and its numeric value are then stored in the local symbol table. The op-code switch is set to 81 denoting an OF statement has been received. A return is then made to the first pass subroutine.

387

SUBROUTINE SX30
(OF CARD)

INCREASE
OF
COUNTER
BY
1

PUT FIRST
OPERAND IN
THE LOCAL
SYMBOL AS
OF NAME

ALL
OPCD
TO
82

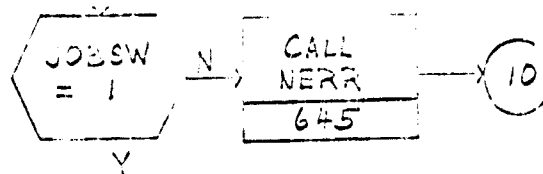
RETURN

SUBROUTINE SX31

This routine processes the END statement. A check is first made to see whether a job is in progress. If not, an error message is written. The job switch is then set to zero, denoting that no job is in progress. The job identification number, the local hash table, and the local symbol table are then written out onto file number 8. The ordinal file counter is reset to zero. The op-code switch is then set to 86, denoting the fact that an END statement has been received. A return is then made to the first pass subroutine.

324

SUBROUTINE SX31
(END CAPD)



SET
JOBSW
= 0

WRITE JOB
NUMBER, HASH
TABLE, AND
LOCAL SYNTAX

SET
OFCTREQ

10

805

10

SET
OPCD
= 33

ZERO
LOCAL
HASH
TABLE

SET FIRST
ENTRY IN
LOCAL SYNTAB
AS NEXT FREE
ENTRY

RETURN

303

SUBROUTINE SX32

This routine processes the COPY statement. The copy name is moved to the CUR output area where it is used to search for the copy file name. If the copy file name is not found an error is written. If it is found, then the file is copied from the library into the copy input file. A switch is then set to indicate that there is input in the copy file, and a return is made to the main first pass routine.

SUBROUTINE SX32
(COPY)

MOVE
COPY-NAME
TO
CUR OUTPUT
AREA

RIGHT-FILL
WITH
BLANKS

REWIND
COPY
FILE

CALL SX04V
SEARCH
FOR
COPY-NAME

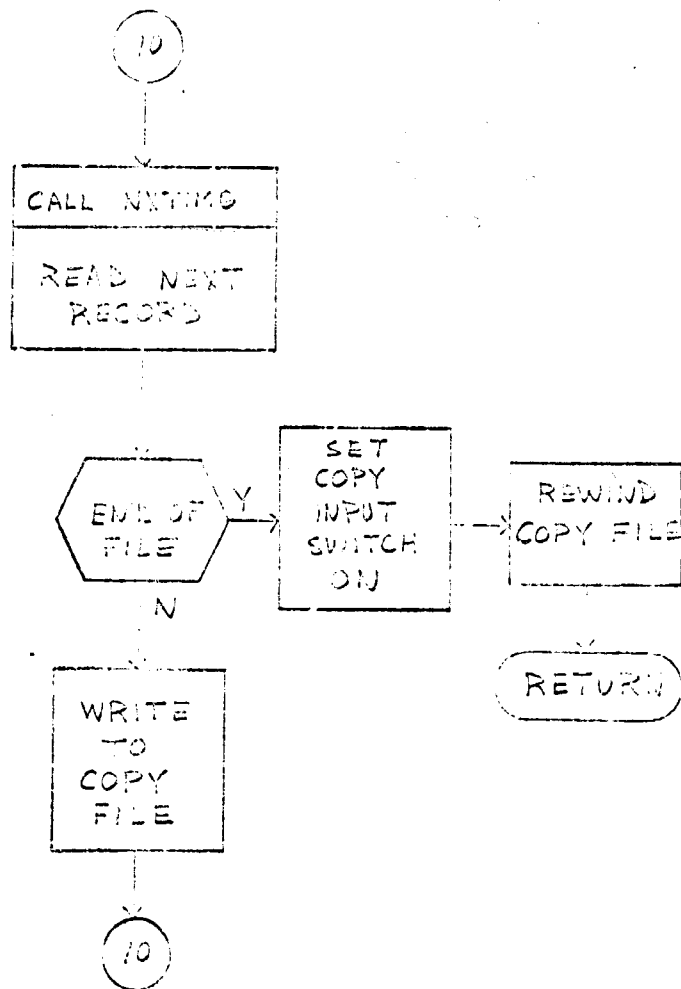
FOUND

CALL
NERR
647

RETURN

Y
10

378

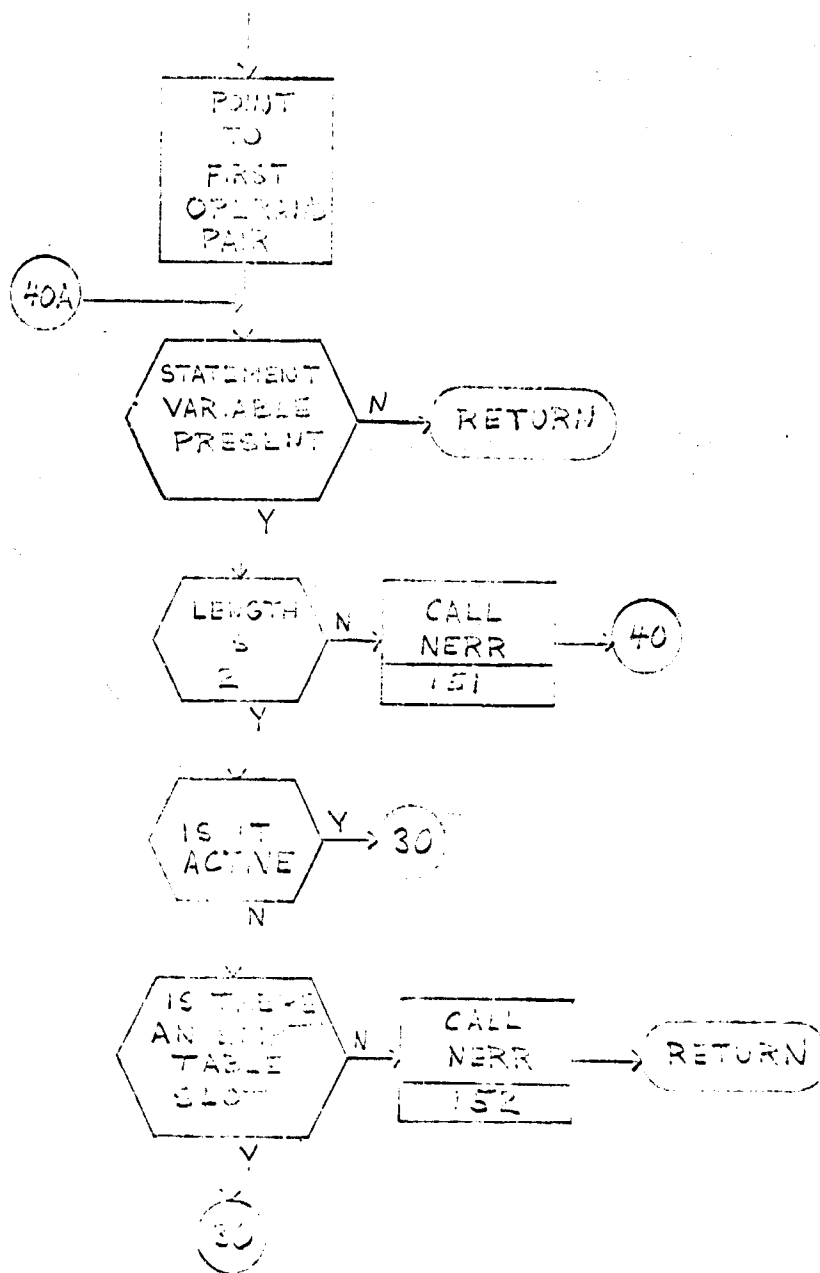


SUBROUTINE SX33

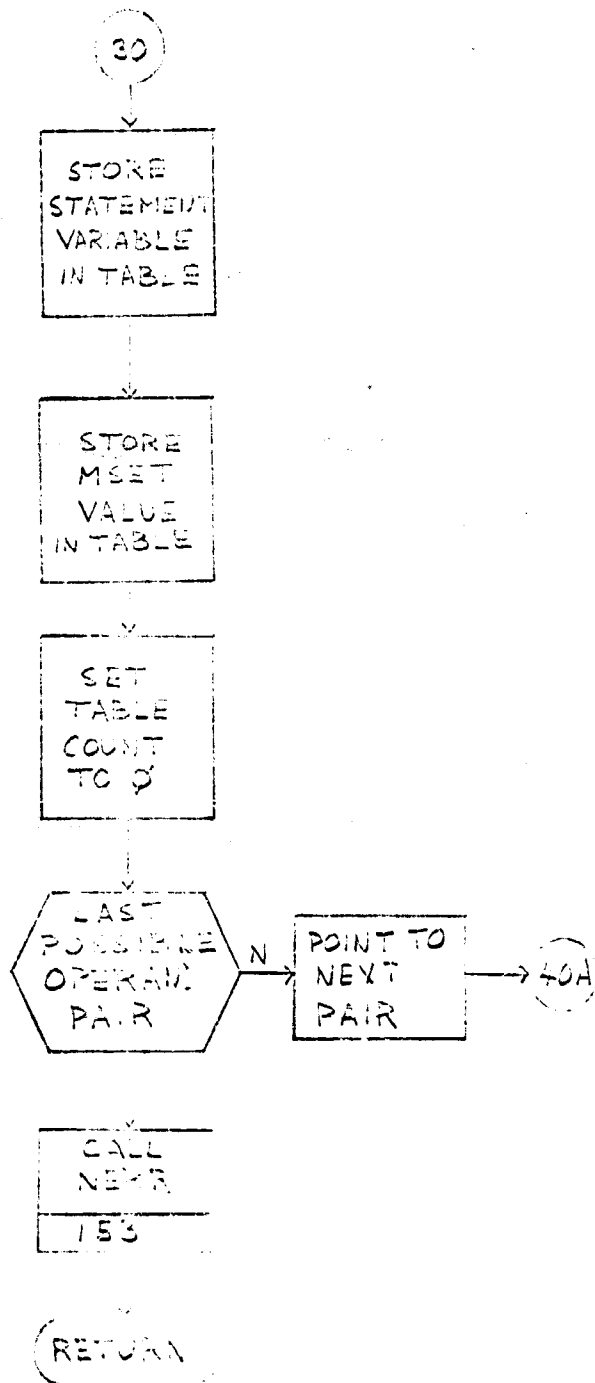
This routine processes the MSET statement. The statement variables, occurring as operands in the MSET statement, are checked for proper format. The table of active statement variables is then searched to see if there is a match for each operand in the MSET statement. If there is a match, the new value is stored in the table and the count zeroed. Otherwise, an empty slot is searched for, and the statement variable placed in this slot with its value and zero count. A check is made to see that no more than ten statement variables are active at any one time. A return is then made to the calling subroutine.

000

SUBROUTINE SY33
(M00T)



391

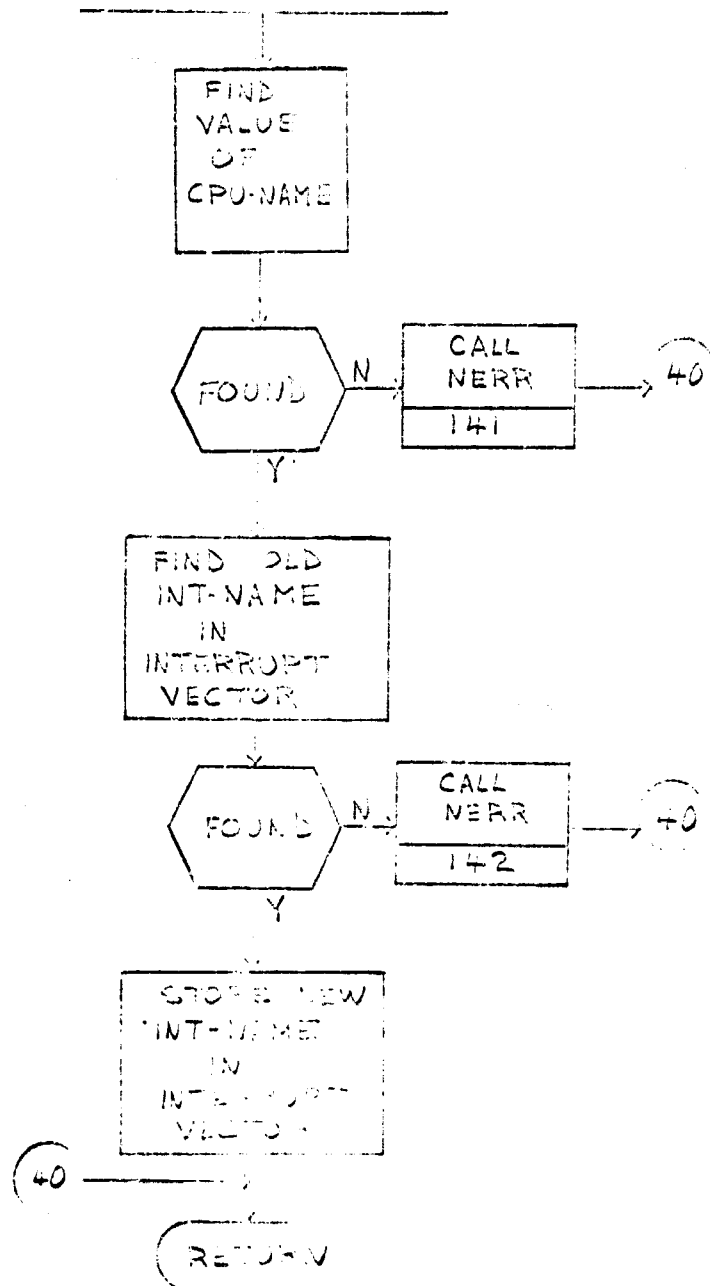


SUBROUTINE SX35

This routine processes the IV statement. The global symbol table is searched for the CPU name, and the interrupt vector is searched for the old interrupt name. The new interrupt name is then stored in the interrupt table.

393

SUBROUTINE SY35
(IV STATEMENT)



354

SUBROUTINE SX36

This routine processes the RCV statement. The global symbol table is searched for the CPU name and its associated value. The load class number which is the second operand in the RCV statement, is stored along with the CPU value in the job distribution table.

335

SUBROUTINE SX26
(RCV)

FIND
VALUE
OF
CPU-NAME

FOUND

CALL
NERR
140

RETURN

STORE CPU-
NAME VALUE
AND LOAD-CLASS
IN JOE DISTRIB-
UTION TABLE

RETURN

245

SUBROUTINE SX37

This routine is used to store the job number and memory name value into the memory assignment table, and to process interrupt names in the interrupt table. This routine is called when the OS statement is encountered.

807

SUBROUTINE SX37
(MEM-ASSIGNMENT & INTERRUPT)

FIND VALUE
OF MEM NAME
IN GLOBAL
SYMBOL TABLE

FOUND

CALL
NERP

143

20

STORE VALUE
AND ADDRESS
IN FIRST
EMPTY MEMORY
ASSIGNMENT

POINT TO
FIRST
CPU-
NAME

20

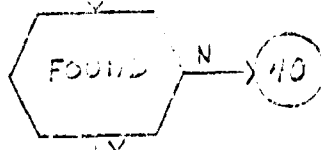
FIND
VALUE
OF
CPU-NAME

30

304

302

FOR

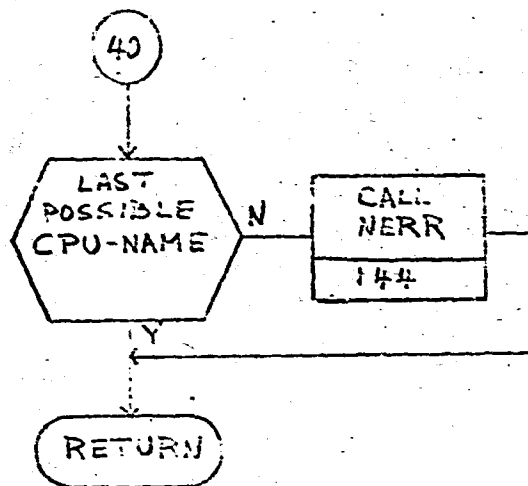


STORE VALUE
OF C U-NAME
AND LOCATION
IN INTERRUPT
TABLE

POINT TO
NEXT
CPU-TIME

30

399

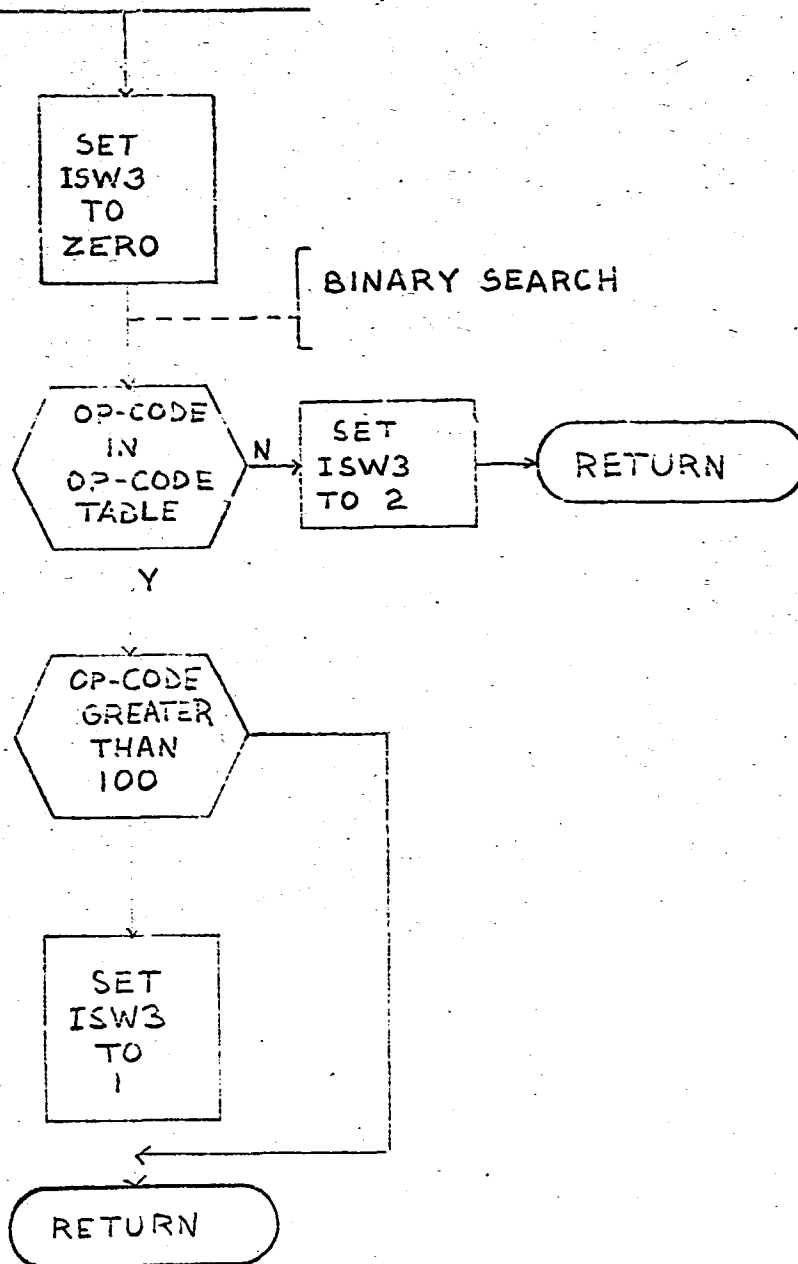


SUBROUTINE S1

The function of this subroutine is to resolve op-codes. Resolution implies the following. A binary search of a table containing all of the valid op-codes is made to see if the op-code contained in the given statement is in the table. If it is, a switch is set indicating a normal op-code. If it is not in the table, then a switch is set to indicate this fact. If an op-code is not in the table, it may either be an invalid op-code or a macro op-code. The determination of which is the case will be made by the SMACRO routine.

401

SUBROUTINE S1
(RESOLVE OP-CODE)



WORKING PAPER

402

SECTION III

FILE DESCRIPTIONS

WORKING PAPER

FORTRAN I/O UNIT ASSIGNMENTS

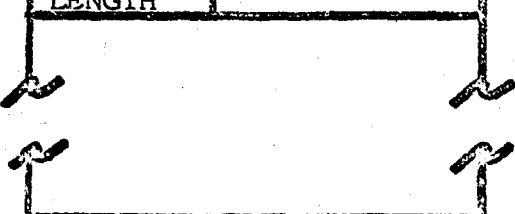
<u>UNIT</u>	<u>FILE</u>
5	CARD READER
6	PRINTER
7	ASSEMBLER INTERMEDIATE FILE
8	LOCAL SYMBOL TABLE FILE
9	COPY INPUT FILE
10	MACRO FILE A
11	MACRO FILE B
12	SECOND PASS OUTPUT
13	FIRST PASS ERROR OUTPUT
14	SECOND PASS ERROR OUTPUT
15	CUR FILE, TAPE 'C'
16	NAME FILES, TAPE 'B'

INTERMEDIATE ASSEMBLY FILE

The Intermediate Assembly File contains the output from the first pass and the input to the second pass. Records on the intermediate assembly file are variable in length, ranging from five words to 23 words. The first word in each record contains the length, in words, of the current record. The variable portion of an intermediate assembly file record consists of statement operands. Since S3 assembler statements may have from 0 to 6 operands, a record in the intermediate assembly file contains only those operands actually present in the current statement.

A detailed description of a record in the intermediate assembly file is shown by the following record layout.

FORTTRAN UNIT 7INTERMEDIATE ASSEMBLY FILEWORD

1	RECORD LENGTH IN WORDS
2	RECORD CODE
3	NUMBER OF OPERANDS
4	STATEMENT NUMBER
5	OP-CD
6	OPERAND-1 PART I
7	OPERAND-1 PART II
8	OPERAND LENGTH
	
21	OPERAND-6 PART I
22	OPERAND-6 PART II
23	OPERAND LENGTH

RECORD CODES

1 = STATEMENT

9999 = END OF FILE

LOCAL SYMBOL TABLE FILE

Each entry in the Local Symbol Table file consists of three physical records. The first record contains a single word which provides the number of the job which created this local symbol table, or else, four nines to indicate end of file.

The second physical record contains a variable number of five word local symbol table entries. The first word of each block contains a count of the number of five word entries contained in this block. Only those symbols obtained from the current job are written out in any one local symbol table entry.

The third physical record is the 90 word local hash table which provides pointers to the local symbol table.

A detailed description of the local symbol table file may be found in the following record layout.

FORTTRAN UNIT 8LOCAL SYMBOL TABLE FILE

<u>BLOCK</u>	<u>WORD</u>
1	1

JOB #

<u>BLOCK</u>	<u>WORD</u>
2	1

LOCAL SYMBOL TABLE

ENTRY COUNT

SYMBOL PART I

SYMBOL PART II

SYMBOL TYPE

SYMBOL VALUE

CHAIN INDEX

<u>BLOCK</u>	<u>WORD</u>
3	1
	2

LOCAL SYMBOL TABLE
POINTERLOCAL SYMBOL TABLE
POINTERLOCAL SYMBOL TABLE
POINTER

00

COPY INPUT FILE

The Copy Input File consists of 14 word records exactly as they are obtained from the library by the COPY statement.

COPY INPUT FILE

1
2
3
4
5
6
7
8
9
10
11
12
13
14

[illegible]

MACRO FILES A & B

Macro Files A & B are used by the SMACRO subroutine during the processing of macro instructions. Records on the macro files are in the exact same format as they appear in table 3. The first word of each record indicates the total number of words in the current record. The records are variable since there may be a variable number of operands in macro statements.

FORTTRAN UNITS 10 & 11MACRO FILES A & BWORDS

1
2
3
4

WORD COUNT

CONTENTS OF

TABLE 3 UP TO

WORD COUNT LIMIT

FIRST PASS ERROR OUTPUT

The First Pass Error Output file is written by the SERR and NERR subroutines. Each record consists of four words. The first word contains the number of the error as passed to the error subroutine. The second word contains the current statement number at the time the error was detected. The third and fourth words of each error record may contain additional information about the error as passed to the error handling subroutine.

413

FORTTRAN UNIT 13

FIRST PASS ERROR OUTPUT

WORD

1

ERROR NUMBER

2

STATEMENT NUMBER

3

ARG-1

4

ARG-2

444

SECOND PASS ERROR OUTPUT FILE

The Second Pass Error Output file is exactly the same as the first pass error file. The format of this file is described in the following file layout.

415

FORTRAN UNIT 14

SECOND PASS ERROR OUTPUT

WORD

1

2

3

4

ERROR NUMBER
STATEMENT NUMBER
ARG-1
ARG-2

MACRO DEFINITION FILE

The Macro Definition File consists of 14 word entries in the PCF library. Since a single statement in a macro definition may require more information than may be stored in 14 words, a single statement may require more than one record in this file. The first word of each record in the macro definition file contains control information. The next 13 words of each record consists of 13 words as copied from table 3.

The first six bits of the first word contain an integer value of one as required by the EXEC 2 system. Bits 12 through 23 contain the number of operands in the macro prototype statement. This allows the SMACRO subroutine to insure that excess operands are not used in calling a macro. The last 12 bits in the first word of each record contains a record code. Records are numbered ordinally for each statement contained in the macro definition file. A comment record is indicated by a record code 99. A detailed description of macro definition file records follows.

MACRO DEFINITION FILE

WORD	6 bits	6 bits	6 bits	6 bits	6 bits	6 bits
1	1	0	NUMBER OF OPERANDS IN PROTOTYPE		RECORD CODE = 1	
2	OPERANDS IN THIS STATEMENT				STATEMENT VARIABLE	
3	LABEL		PART I			
4	LABEL		PART II			
5	LABEL LENGTH	0 = FIXED LABEL N = VARIABLE SUBSCRIPT		0 = NO MACRO # 99 = ADD MACRO #		0
6	OP-CD		PART I			
7	OP-CD		PART II			
8	OP-CD LENGTH	0 = FIXED OP-CD N = VARIABLE SUBSCRIPT		0 = NO MACRO # 99 = ADD MACRO #		0
9	OPERAND 1		PART I			
10	OPERAND 1		PART II			
11	OPERAND-1 LENGTH	0 = FIXED OPERAND N = VARIABLE SUBSCRIPT		0 = NO MACRO # 99 = ADD MACRO #		0
12	OPERAND 2		PART I			
13	OPERAND 2		PART II			
14	OPERAND-2 LENGTH	0 = FIXED OPERAND N = VARIABLE SUBSCRIPT		0 = NO MACRO # 99 = ADD MACRO #		0

STATEMENT

HEADER RECORD

MACRO DEFINITION FILE

WORD	6 bits	6 bits	6 bits	6 bits	6 bits	6 bits
1	1	0	NUMBER OF OPERANDS IN PROTOTYPE		RECORD CODE = 2-14	
2	OPERAND-N PART I					
3	OPERAND-N PART II					
4	LENGTH	0 = FIXED OPERAND N = VARIABLE SUBSCRIPT		0 = NO MACRO # 99 = ADD MACRO #		0
5	OPERAND-N PART I					
6	OPERAND-N PART II					
7	LENGTH	0 = FIXED OPERAND N = VARIABLE SUBSCRIPT		0 = NO MACRO # 99 = ADD MACRO #		0
8	OPERAND-N PART I					
9	OPERAND-N PART II					
10	LENGTH	0 = FIXED OPERAND N = VARIABLE SUBSCRIPT		0 = NO MACRO # 99 = ADD MACRO #		0
11	OPERAND-N PART I					
12	OPERAND-N PART II					
13	LENGTH	0 = FIXED OPERAND N = VARIABLE SUBSCRIPT		0 = NO MACRO # 99 = ADD MACRO #		0
14						

STATEMENT

TRAILER RECORD

449

MACRO DEFINITION FILE

WORD	6 bits	6 bits	6 bits	6 bits	6 bits	6 bits
1	1	0	NUMBER OF OPERANDS IN PROTOTYPE		RECORD CODE = 98	
2	*	COMMENT COL 1-78				
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						

STATEMENT

COMMENT RECORD

490

DIAGNOSTICS FILE

The Diagnostics File consists of 14 word records in the PCF library. The first word of each record consists of control information. The next 12 words contain the format statement used to print the error message. The last word contains the error number in the first 12 bits.

A record layout for the diagnostics file is shown on the following page.

428

DIAGNOSTICS FILE

WORD	6 bits	6 bits	6 bits	6 bits	6 bits	6 bits
1	63	NUMBER OF OPERANDS	PRINT LINES	CONTINU- ATION COUNT		
2	ERROR FORMAT					
3	STATEMENT					
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14	ERROR NUMBER					