2000 E
CORE LAY-OUT

1320 – 1324	NOT USED. (65)	
24002 – 24013	NOT USED. (12)	
31501 – 31534	NOT USED (EF) ??	

12337 – 13777	USER'S BUFFERS (FOR 50 WORDS ONLY)	
14000 – 22033	BASIC INTERPRETER + EXECUTOR	
22033 – 27152	UTILITY + SUBROUTINES	
27153 – 27507	OUTPUT ROUTINES	
27510 – 30764	LIBRARY FUNCTIONS	
30765 – 31333	ASCII TO BINARY NUMBER CONVERSION	
31334 – 31534	DISC DRIVER	
31535 – 31776	PRINT NAME TABLE	
31777 – 32065	X POWER FAIL	
32066 – 32173	PHONES LOGIC	
32174 – 32272	X START-OF-SYSTEM	
32273 – 32503	SYSTEM CONSOLE DRIVER	
32504 – 33226	MULTIPLEXER DRIVER	
33227 – 33246	TTY POINTER TABLE	
33247 – 34166	TTY TABLES	
34167 – 35557	SCHEDULER	
35560 – 35721	PROCESS OUTPUT CHAR. ROUTINE	
35722 – 36147	COMMAND TABLE	
36150 – 36205	ABORT CHECK	
36206 – 37277	LIBRARY ROUTINES	
37300 – 37677	OVERLAY AREA FOR COMMANDS	

USER AREA 1324 – 12252 (8)

$(67)_8 = 55_{10}$
$(106)_8$
$(77)_8 = 63_{10}$
$\Sigma = 166_8 = 118_{10}$

SYSTEMS ANALYST'S SEMINAR

INTRODUCTION TO THE 2000E

August, 1972
Al Pare

2000E SEMINAR OUTLINE

I. Introduction

II. Pre-Sales Information

A. Advantages

1. user-operator communication via the MESSAGE and ANNOUNCE commands. This did not exist on the 2000 A or B.
2. 2000E CPU is equipped with floating point hardware. This significantly improves execution times of computation programs and hence is more cost effective.
3. the utility routine allows an operator to do disc to disc transfers,
4. multi-speed terminals of 10, 15, and 30 chars/sec. The A, B, and C do not have a 30CPS transfer rate.
5. system hardware can be utilized to perform other functions by converting it into:
    1. 2120 DOS
    2. 9600 real time exec.
    3. BCS

   This transformation can be made simply by configuring the hardware the same as the 2000E at system generation time.

B. Disadvantages

1. there is less user area for program development than in existing time-share systems.
2. no formatted output capability
3. backup of user files is a more lengthy procedure if system is not equipped with mag tape (standard configuration).

C. Sales Price (approximate)

1. $50,000 minimum configuration
2. $1,600/month - lease
3. $1,575/month - rental

D. Applications
   1. mainly education
   2. some data centers

III. System Hardware Configuration

A. Illustrative diagrams
   1. the minimum configuration as well as the optional hardware available for a 2000E is shown in Figure 1.
   2. Figure 2 shows a system block diagram of the E and its various options.

IV. System Commands

A. Operator commands
   1. A cumulative operator command chart for the 2000A, B, C, and E is shown in Figure 3.
      a. ANNOUNCE - in 2000 C and E but not A or B.
      b. DIRECTORY - this command is similiar to the DIRECTORY command on existing TSB systems except that the disc subchannel number is an input parameter, The operator has the option of having a specific disc subchannel's DIRECTORY or all disc DIRECTORYs for a particular idcode listed at the system console.
      c. DISC - UP DN, 1-3 serves same function as the DISC command on other time-share systems. On the E, this command allows the operator to interchange user disc cartridges on a disc drive by declaring the DISC - DN, replacing the cartridge, and then delcaring DISC - UP.
      d. MOVE - enables the operator to transfer user files from one disc subchannel to another.
      e. PORT - is a new command which outputs to the system console the stop bit and baud rate configuration for each port. This allows determination of the operating speeds of each port.
      f. SPEED - informs system of baud rate and no. of stop bits for each or all ports.

g. there is a number of commands which are not used on the 2000E. Some of these are commands which are relevant only to systems having a drum or commands which were replaced by new ones, e.g. SPEED for FAST and SLOW. There are, however, some which were eliminated altogether, i.e. MLOCK, MUNLOCK, HIBERNATE, and STATUS. The DUMP, LOAD, AND COPY commands are part of the utility program in the E.

B. User Commands

1. Figure 4 shows a list of user commands for the A, B, C and E systems.

   a. DISC - a new command which lists the no. of sectors used and the total no. allocated to a user at his console.
   b. CSAVE, GROUP - nonexistent on the E.
   c. MESSAGE - in C and E, but not A or B.
   d. XPUNCH - in C and E, but not A or B.

C. Utility Commands

1. Figure 5 shows the utility commands (issued by the system operator)

   a. LOAD - loads the contents of a mag tape onto disc(s).
   b. COPY - copies the contents of one disc subchannel to another disc subchannel.
   c. SLOAD - selectively loads a file from mag tape to a specified disc subchannel. If a file value is not input, the contents of the first file is transferred to the disc subchannel.
   d. SDUMP - selectively dumps an entire disc subchannel contents to a specified file on mag tape. If no file is input, it is dumped to the first file on tape.
   e. FORMAT - formats a user disc subchannel other than 0. Builds the ADT and DIRECTORY table which resides on the disc.
   f. PACK - eliminates unused spaces on a disc created by use of the KILL command.

V. User/System Limitations

A. Files

1. the maximum no. of files is 4 per program. (16 on 2000 B, C 8 on A).
2. there is 48 records maximum/file, each record being 128 words in length. The record size can not be specified by the OPEN command as in the 2000C. (2000A, B = 64 words/record, 2000C = 64-256 words/record; 2000A, B 90 or 128 max. # records/file, 2000C depends on system peripherals).
3. each file listed in a FILES statement utilizes 128 words of the user area. (2000A, B = 64 words, 2000C as many words as there are words in each logical record of the file).
4. no ASSIGN statement. This can be gotten around by using the CHAIN statement.

B. Programs can not be saved in a semi-compiled form. This is unfortunate since this speeds up the chaining process.

C. Tracks can not be locked and unlocked by system commands.

D. Syntax - Figure 6

1. no ASSIGN statement
2. no ENTER statement
3. no PRINT USING statement
4. no MAT PRINT USING statement

E. No line printer capabilities

F. User area = 4,180 words (2000A, B = 5,120 words, 2000C = 10,000 words).

G. Program conversion from other time-share systems to the 2000E

1. Considerations
    a. program size
    b. no formatted output

c. no ENTER statement
d. no ASSIGN statement
e. no line printer
f. file handling
    1. FILES statement
    2. number of records per file
    3. size of records

VI. System Software Overview

A. System routines

1. Figure 7 shows a simplified overview of the 2000E system software modules.

a. scheduler - executive module
    1. schedules/initiates/suspends/terminates tasks
    2. entered every 100 milliseconds by TBG
    3. handles service requests from other modules
    4. optimizes allocation of CPU time

b. interpreter
    1. syntax checking/program execution
    2. re-enterable processor
    3. calls scheduler upon completion of task

c. swapper
    1. swap users in and out of core
    2. swap library programs into core

d. multiplexer routines
    1. multiplexes input/output from/to the 16 user teletypes
    2. communicate with scheduler via MPCOM
    3. uses TTY tables and buffers for I/Ø communications with each port

e. I/Ø drivers
    1. disc driver
    2. console TTY driver
    3. mag tape driver

f. power fail
    1. calls other modules to reinstate status of system

2. Origin of system routines
    a. scheduler - 2000A
    b. multiplexer routines - 2000C hi-speed
    c. library subroutines
        1. XPUNCH, RND, CHAIN - 2000C
        2. HELLO - new
        3. remainder - 2000B
    d. disc driver - new
    e. utility program - new
    f. loader - 2000A
    g. remainder - mostly 2000B
3. Overlays
    a. library
        1. HELLO, OPEN, SLEEP, KILLID
        2. PURGE, DISC, SAVE, MOVE
    b. important to know which overlay was in core when system crashes
4. Figure 8 shows a system core map

B. System Tables
1. DIREC and EQUIPMENT tables are shown in Figure 9
    a. DIREC table - same format as other time-share systems differing only in length. However the disc addresses are addresses pertaining to the system and user discs. This is because a DIRECTORY table exists on every disc subchannel in the system.
    b. EQUIPMENT table - resembles the 2000A EQUIPMENT table with slight variations. IDTTA points to the location which contains the 4 disc addresses of the ID tracks. IDTRL points to the location which contains the track lengths of the 4 ID tracks.
2. ID and AD tables are shown in Figure 10
    a. IDT - 2000B format, uses 4 tracks on system disc
    b. ADT - 2000B format, exists on all disc subchannels to allow the interchanging of user

discs in the system. Requires 1 track on each disc subchannel

3. FUSS table and COMTABLE are illustrated in Figure 11
    a. FUSS - is different from previous FUSS tables in that each entry for a file contains 2 words, 1 word for the disc address of a file and the other word for the length of the file in sectors. 128 words in length, resides on system disc.
    b. COMTABLE - 2000B format, 66 words in length, core resident.
4. LOGGR and the DIRECTORY table are shown in Figure 12
    a. LOGGR - 2000B format, 32 words long, core resident
    b. DIRECTORY - 2000B format, requires 2 tracks on each disc subchannel to permit the interchanging of user disc cartridges.
5. TTY table - Figure 13
    a. TTY table - new format 29 words per table, resembles 2000A TTY tables.
    b. new words
        1. ?TNUM - port number
        2. ?DCNT - CR/LF delay counter
        3. ?CDLY - CR delay
        4. ?LDLY - LF delay
        5. ?RPRM - receive channel parameters
        6. ?SPRM - send channel parameters
        7. ?PPRM - phone parameter

C. Mag Tape Formats

A. SLEEP tape format - Figure 14

1. system disc resident library and tables are dumped to mag tape first and terminated with an EOF.
2. each user disc subchannel (that is up) is written out to mag tape and seperated by an EOF marker.
3. the system routines, in segments, are dumped last

to mag tape as well as the system library overlay routines.

B. Selective dump mag tape format (utility routines) - Figure 14-1.
   1. a disc subchannel is dumped out by the SDUMP command as 1 file on mag tape. This includes the ADT, DIRECTORY table, user library, and other data stored on that disc subchannel. It is terminated with an EOF marker.

VII. Disc Organization
   A. Replaceable disc cartridge
      1. Figure 15 - 17
         a. user file only
      2. User disc track assignments - Figure 18

   B. Non-replaceable disc cartridge
      1. Figure 15 - 16
         a. drive 0 - system disc
         b. drive 1 - user files (if 7900A disc)
      2. Track assignments for the system disc - Figure 19

   C. Disc address format
      1. Figure 20 - format of disc addresses as stored in system tables

VIII. Emergency Resucitation
   A. Disc errors
      1. Non-replaceable disc - subchannel 0
         a. recovery procedures depend upon system hardware configuration and the status of each disc subchannel at the time of the disc error, i.e., whether disc was UP or DN. In any case, this is a very catastrophic situation.
         b. tracks 0 - 2, replace fixed disc
         c. tracks 3 - 202, regenerate system. The loader will lock out the bad tracks on the fixed cartridge by not entering them in its ADT.

2. Replaceable disc - subchannels 1 and 3. Non-replaceable disc - subchannel 2.
   a. perform emergency resucitation (VIII.D.2.)
   b. bring system up from the SLEEP disc or tape
   c. get a DIRECTORY for the bad disc subchannel
   d. use the MOVE command to move all files and programs to another user disc or the system disc.
   e. tracks 0 - 2, replace disc
   f. tracks 3 - 202, reformat disc.

B. Disc error HALTS
   1. During system operations
      a. display register = 102010
      b. A reg = disc addr, B reg = core addr
      c. system tries 10 times before HALT
      d. if the "RUN" button is pressed, the disc operation will be tried an additional 10 times.
   2. During utility program operations
      a. display register = 102011
      b. A reg = disc addr, B reg = core addr
      c. system tries 10 times before HALT
      d. reload the utility program and re-issue the command.

C. Power fail
   1. During system operation
      a. all users come up in syntax mode
      b. ports come up as configured prior to power failure
   2. During SLEEP operation
      a. if DONE message was output to console, no operator action is required.
      b. if a disc SLEEP was in process, the disc transfer is terminated and the operator must re-enter the SLEEP command.
   3. During mag tape SLEEP operation
      a. if the first write operation had taken place:

1. load the bootstrap loader
2. bring the system up from system disc subchannel 0.
3. SLEEP the system

b. if no write operation had been started:
1. perform emergency recovery procedures
2. SLEEP the system

D. System crashes
1. Pre-recovery checks
a. assure table entries are in proper order
1. DIREC
2. EQUIPMENT
b. attempt to determine cause of crash
1. core/disc dump utility program
2. some important core locations are shown in Figure 21
2. Recovery procedures
a. if system table entries are in order and core locations appear to be undisturbed, SLEEP the system after using one of the following options:
*1. POWER FAIL restart routine
2. Start at TSB point 32173_8.
b. if system table entries are in order but other core locations thought to be destroyed:
1. load TSB loader
2. start at the emergency resucitation point - location 3000_8.
3. load system tapes
4. SLEEP the system.
c. if system table entries are not in order and it is felt the system can not be saved (slept), bring system up using most recent SLEEP tape(s) or disc cartridge.

---

*if the POWER FAIL option is selected and does not run to completion, i.e., system "READY" message is not output to console TTY, check to see if it is hung up trying to complete a disc transfer. If this is the case, halt the computer and use option 2.

IX. Benchmarks

- A. Compute bound programs
  - 1. BTEST, FPTS9 - Figure 22
- B. Results - Figure 23

X. References

- A. 2000E I/O configuration (attached)
- B. HALTS (attached)
- C. Selective core/disc dump utility listings (attached).
- D. Flow charts (attached)
- E. IMS's
  - 1. 2000A
  - 2. 2000B
  - 3. 2000C Hi-speed
- F. 2000E ERS

XI. Lab

A. Formatting of disc cartridges

B. System generation/update

C. Operator commands

D. Hands on

E. SLEEP procedures

F. Utility commands

2000E BASIC SYSTEM HARDWARE

HP 2100 Digital Computer with the following:

- 16K core Memory
- Floating Point Arithmetic Hardware
- Direct Memory Access
- Time Base Generator
- Telephone Auto-Disconnect for 16 lines

7900 Cartridge Disc Drive (4.8 megabyte) and Interface

High Speed Tape Reader and Interface

System Teletype (modified ASR-33) and Interface

Hardware Multiplexer (16 terminals)

Single Bay Cabinet with door (115V, 60Hz power)

System Integration Software and Accessories

2000E SYSTEM OPTIONS

Additional Cartridge Disc Storage - 1 Disc Drive (4.8 megabytes)

9-Channel Magnetic Tape (30,000 char/sec.), Interface and Cabinet

System Operation with 230V, 50Hz

Heavy Duty System Teleprinter (modified ASR-35) and Interface

Heavy Duty System Teleprinter (modified ASR-35) and Interface for 230V, 50Hz operation

Two Bay System Cabinet x with door

2000E OPTIONAL PERIPHERAL EQUIPMENT

Teleprinter Terminal - HP 2749A Teleprinter (modified teletype ASR-33 with X-ON/X-OFF reader control options)

Keyboard Display Terminal - HP 2600A

General Electric "Terminet 300"

"Memorex 1240"communications terminal with 10/15/30 transfer rates

"Execuport 300" Transceiver Terminal

ASR-37 with paper tape reader/punch

Univac DCT 500 terminal (type 8541-99 standard version)
Must be equipped with ASCII printwheel and ASCII keytop

Figure 1

OPERATOR COMMANDS

ANNOUNCE			C	E
BESTOW			C	
CHANGE	A	B	C	E
COPY			C	
DESECRATE			C	
DIRECTORY	A	B	C	E
DISC	A	B	C	E
DRUM			C	
DUMP			C	
FAST		B	C	
HIBERNATE			C	
KILLID	A	B	C	E
LOAD			C	
LOCK	A	B	C	
MAGTAPE	A	B	C	
MLOCK			C	
MOVE				E
MUNLOCK			C	
NEWID	A	B	C	E
PHONES	A	B	C	E
PORT				E
PROTECT	A	B	C	E
PURGE	A	B	C	E
REPORT	A	B	C	E
RESET	A	B	C	E
ROSTER	A	B	C	E
SANCTIFY			C	
SDIRECTORY			C	
SLEEP	A	B	C	E
SLOW		B	C	
SPEED				E
STATUS	A	B	C	
UNLOCK	A	B	C	
UNPROTECT	A	B	C	E

Figure 3

USER COMMANDS

APPEND	A	B	C	E
BREAK	A	B	C	E
BYE	A	B	C	E
CATALOG	A	B	C	E
CSAVE		B	C	
DELETE	A	B	C	E
DISC				E
ECHO	A	B	C	E
GET	A	B	C	E
GROUP			C	
HELLO	A	B	C	E
KEY	A	B	C	E
KILL	A	B	C	E
LENGTH	A	B	C	E
LIBRARY	A	B	C	E
LIST	A	B	C	E
MESSAGE			C	E
NAME	A	B	C	E
OPEN	A	B	C	E
PUNCH	A	B	C	E
RENUMBER	A	B	C	E
RUN	A	B	C	E
SAVE	A	B	C	E
SCRATCH	A	B	C	E
TAPE	A	B	C	E
TIME	A	B	C	E
XPUNCH			C	E

Figure 4

2000E UTILITY PROGRAM COMMANDS

COMMAND	FUNCTION
LOAD, select code	Load system from mag tape
COPY, subchannel#, subchannel#	Copies disc to disc
SLOAD, subchannel#, select code, (file)	Selective disc load from mag tape
SDUMP, subchannel#, select code, (file)	Selective disc dump to mag tape
FORMAT, subchannel#	Formats a user disc, builds the AI and DIRECTORY on the disc
PACK, subchannel#	Packs a user disc

(file) - is an optional value representing a specific file position on the mag tape. A file is the contents of one disc as dumped out either by a mag tape sleep or by a selective dump.

Figure 5

SYNTAX

ASSIGN	C	
CHAIN	C	E
COMMON	C	E
DATA	C	E
DIM	C	E
END	C	E
ENTER	C	
FOR	C	E
GO TO	C	E
GO TO . . . . OF	C	E
GOSUB	C	E
GOSUB . . . OF	C	E
IF . . . . THEN	C	E
IMAGE	C	
INPUT	C	E
LET	C	E
MAT INPUT	C	E
MAT PRINT	C	E
MAT PRINT USING	C	
MAT READ	C	E
NEXT	C	E
PRINT	C	E
PRINT USING	C	
READ	C	E
REM	C	E
RESTORE	C	E
RETURN	C	E
STOP	C	E

Figure 6

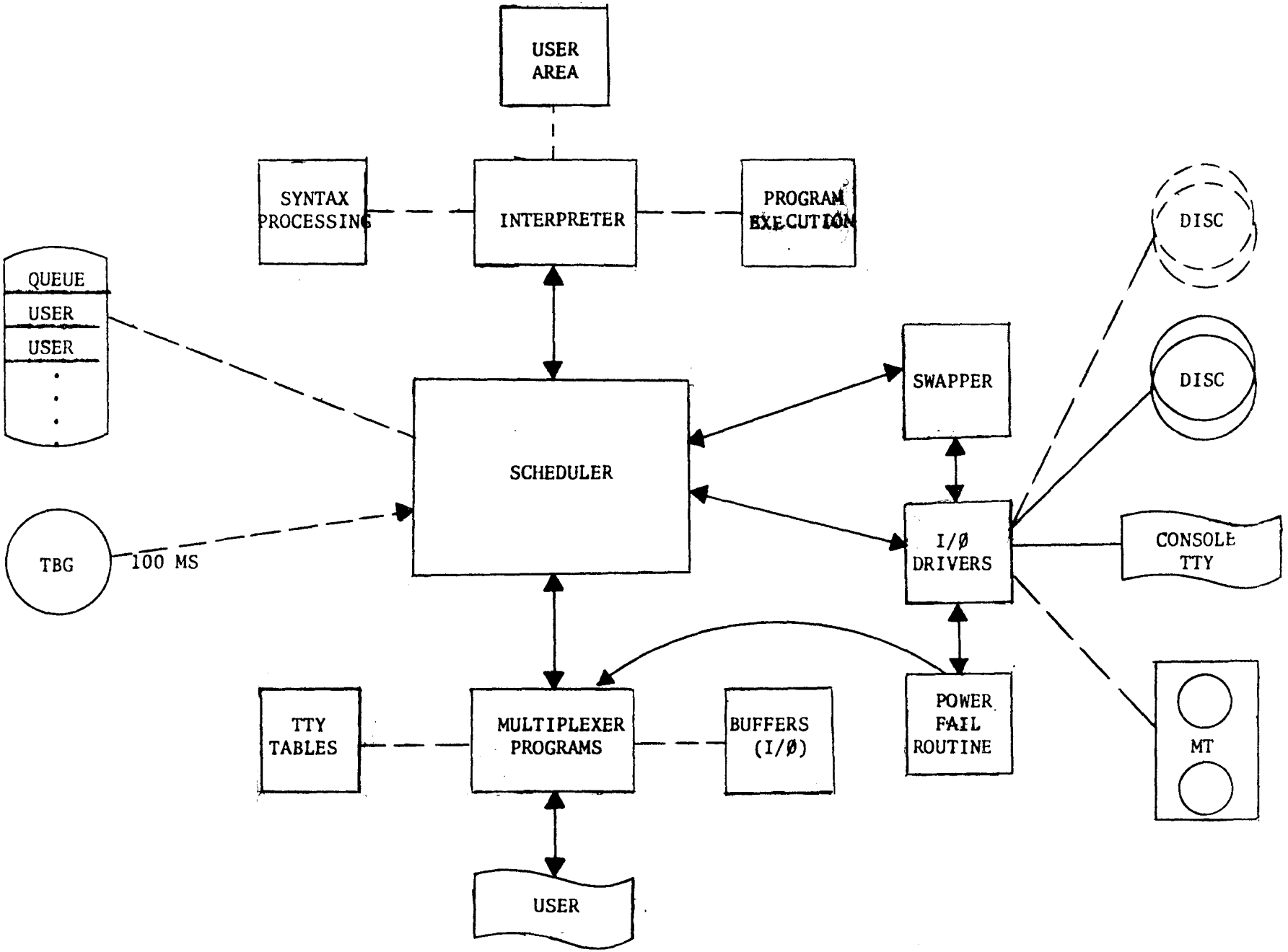
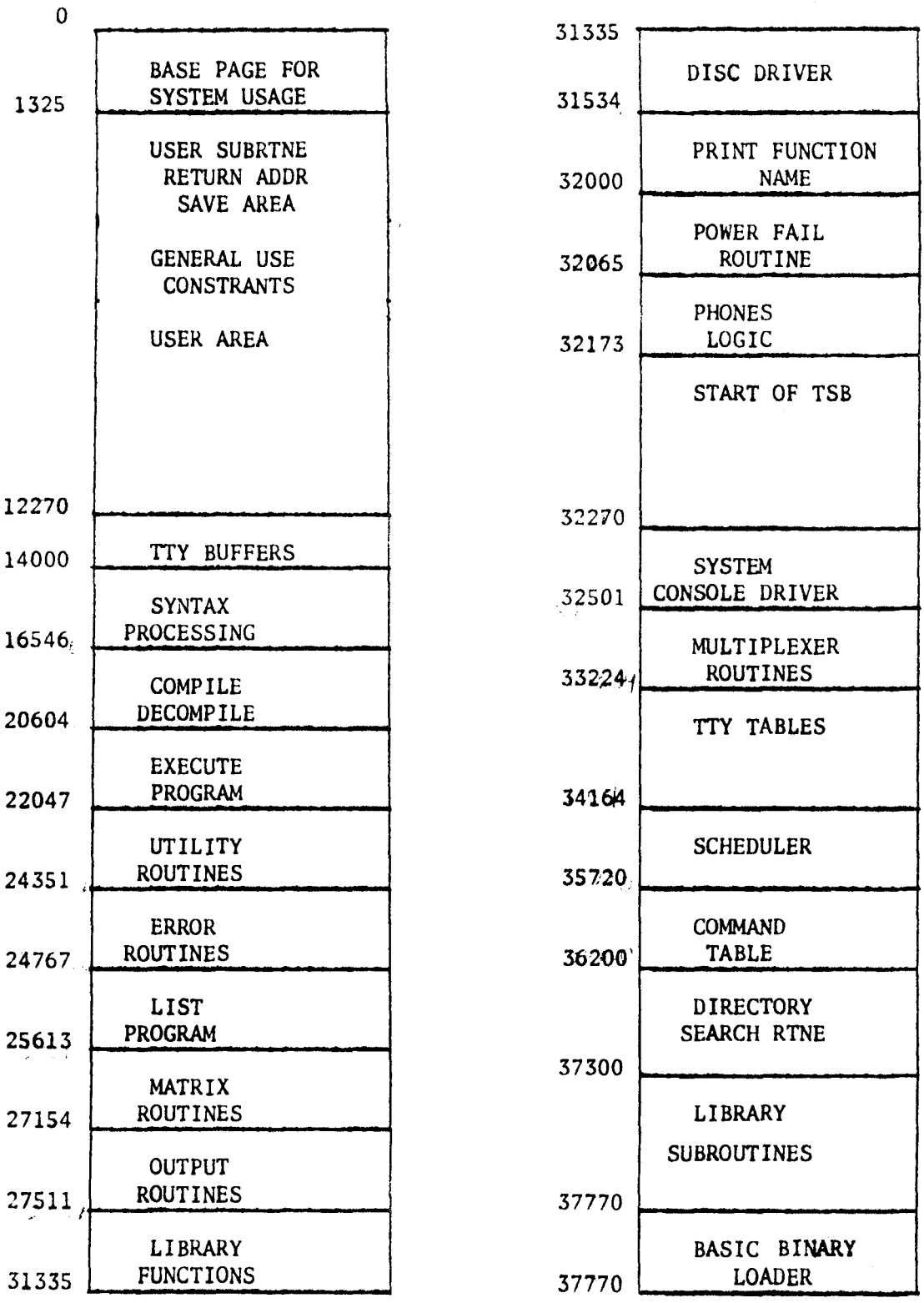


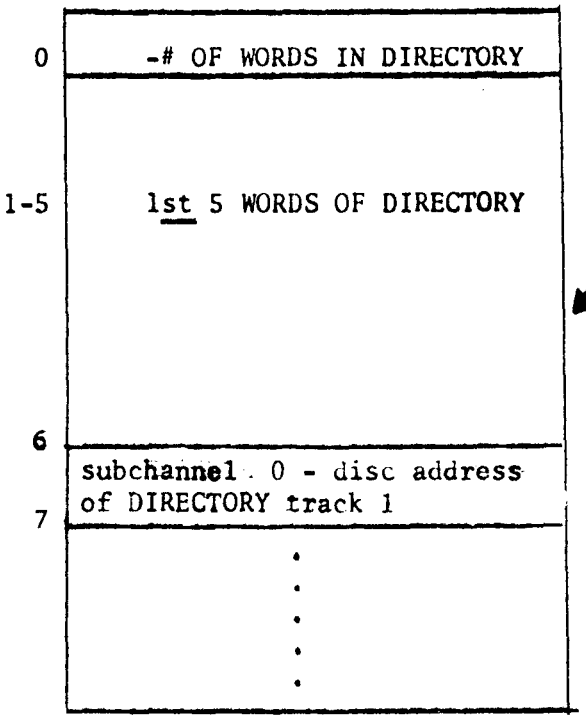
Figure 7

SYSTEM SOFTWA OVERVIEW



2000E 2100 CORE MAP
Figure 8

DIREC
TABLE



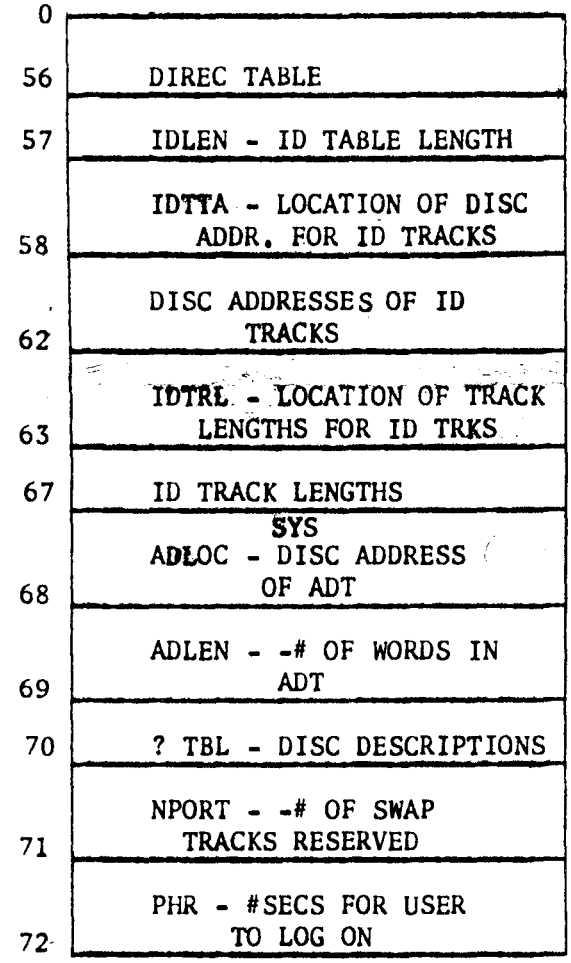
56 WORDS TOTAL
2000B FORMAT

CORE RESIDENT
2-7 WORD ENTRIES
FOR EACH DISC
SUBCHANNEL IN
SYSTEM

table order

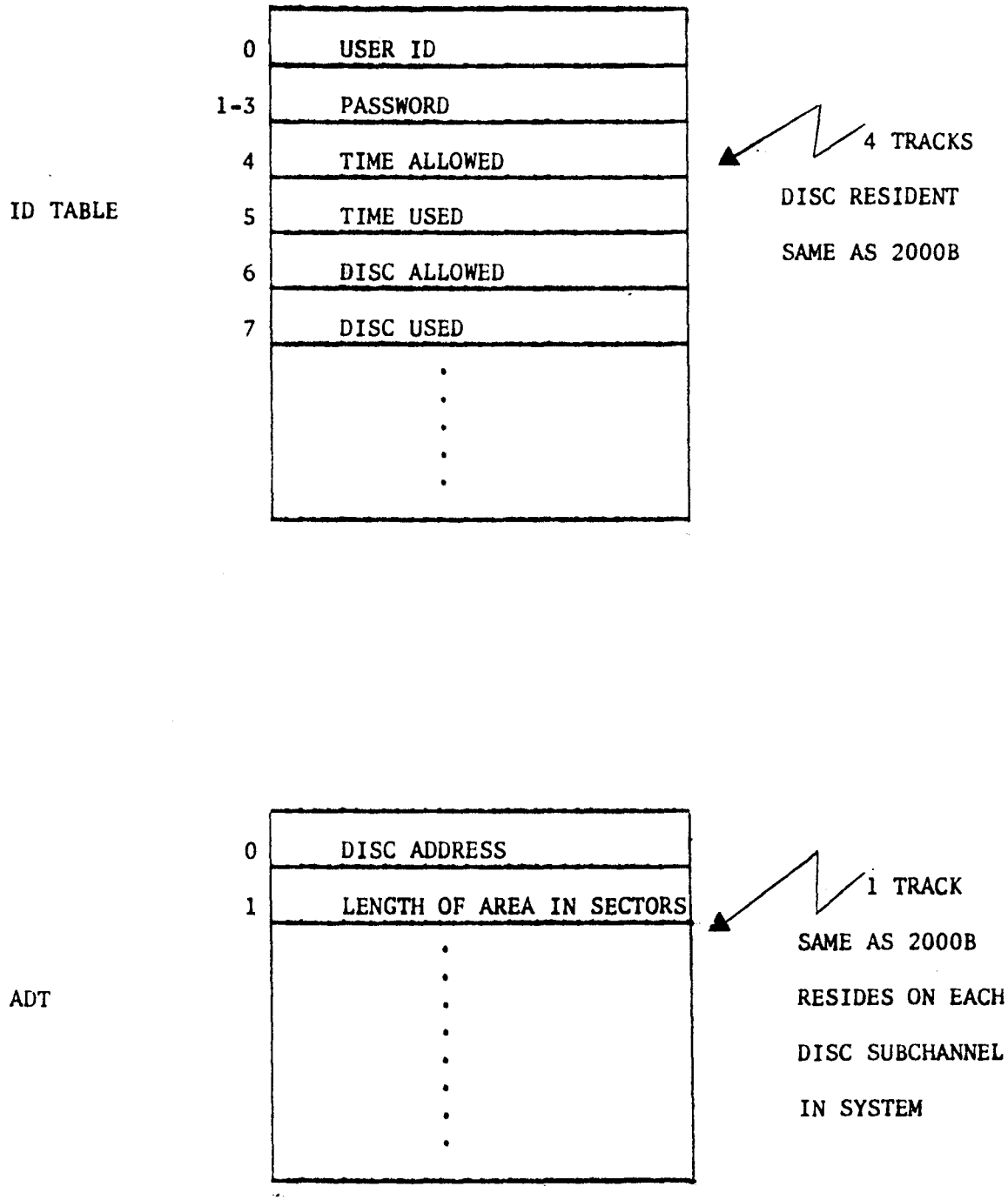
subchannel 0	DIRECTORY trk 1
subchannel 0	DIRECTORY trk 2
subchannel 3	DIRECTORY trk 1
subchannel 3	DIRECTORY trk 2

EQUIPMENT
TABLE



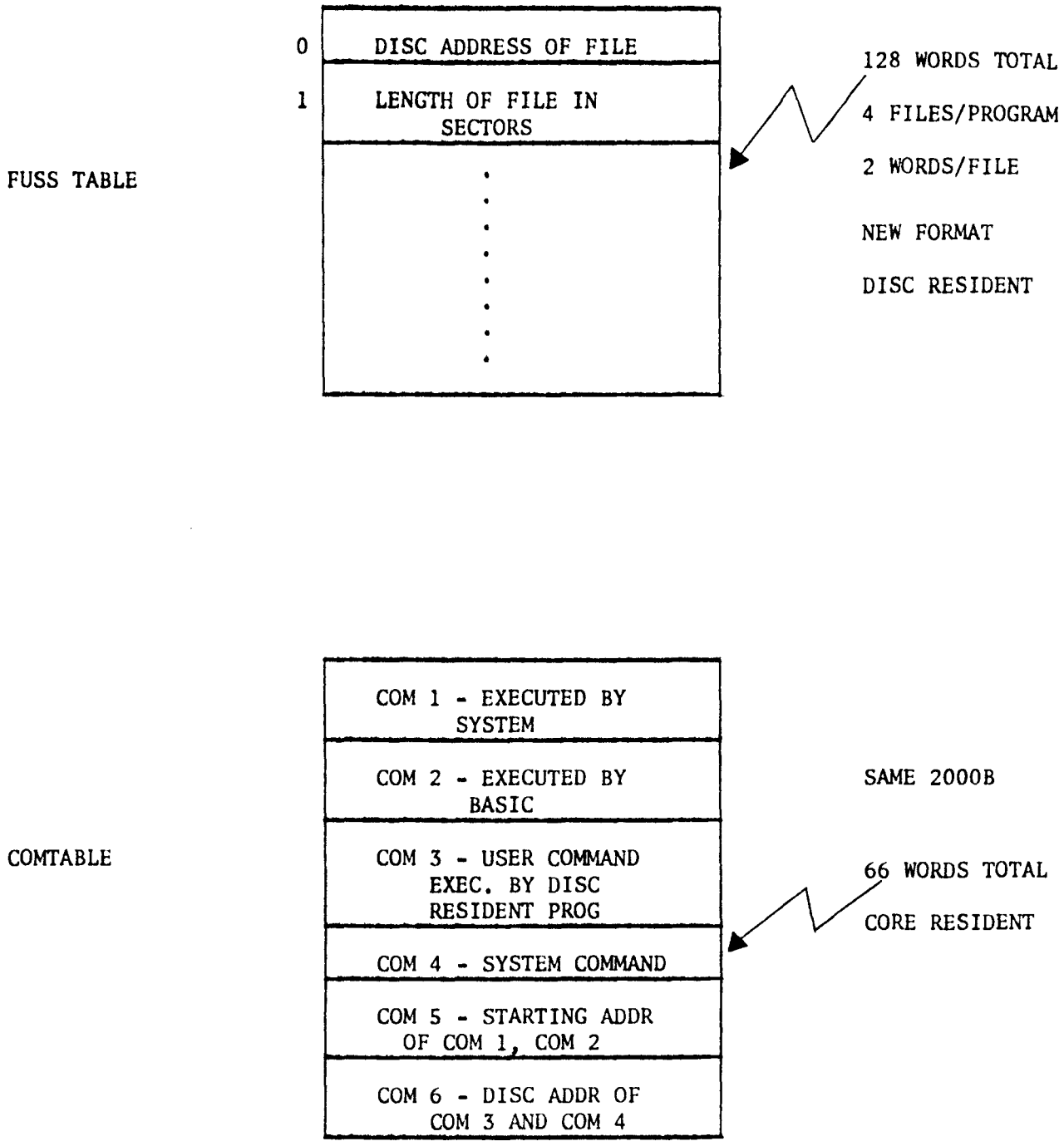
72 WORDS
2000A WITH MODS
CORE RESIDENT

DIREC AND EQUIPMENT TABLES
Figure 9



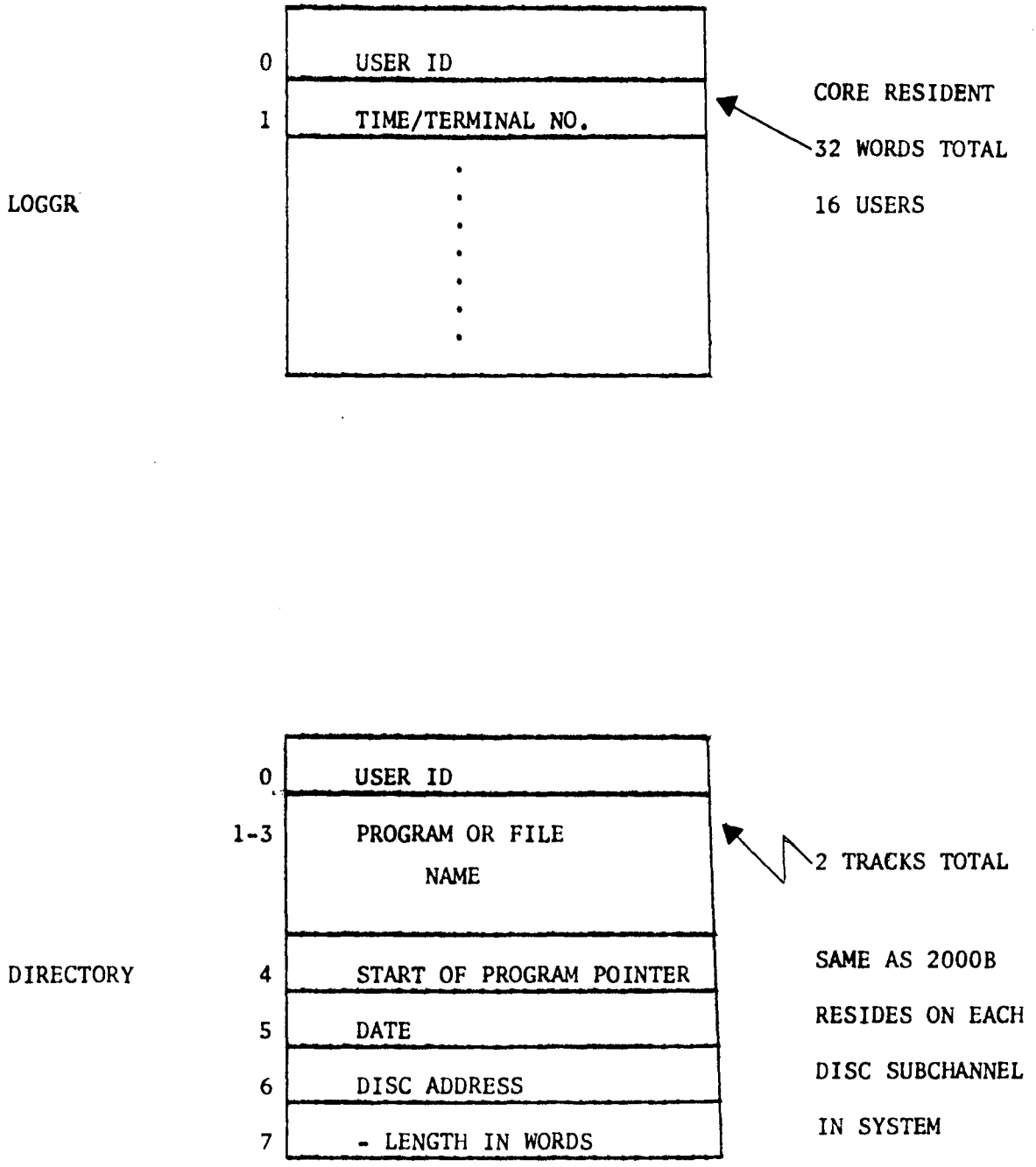
IB AND AD TABLES

Figure 10



FUSS TABLE AND COMTABLE

Figure 11



LOGGR AND DIRECTORY

Figure 12

TTY
TABLE

1	? TNUM - Port Number
2	? CCNT - Char. Cntr
3	? BPNT - Char. Loc. PTR
4	? BSTR - Char. PTR in Buffer
5	? BHED - PTS to Next Char.
6	? BGIN - PTS to Beginning of Bufr
7	? BEND - PTS to 1st Char. after Bufr
8	? TSTA - Status Word
9	? DCNT - CR/LF Delay CNTR
10	? CDLY - CR Delay
11	? LDLY - LF Delay
12	? PHON - Time CNTR For Phones
13	? RPRM - Receive Channel Parms
14	? SPRM - Send Channel Parms
15	? PPRM - Phone Parameter
16	? MASK - 2 For User N
17	? DISC - Disc Address
18	? PROG - Points To Last Core Word
19	? ID - User ID
20	? NAME - Program Name
21	
22	
23	? Time - Starting Time
24	
25	? CLOC - User's Timeout Clock
26	? RSTR - Restart Address
27	? STAT - User's Status
28	? LINK - PTS To Next Entry on Q
29	? PLEV - Priority Level

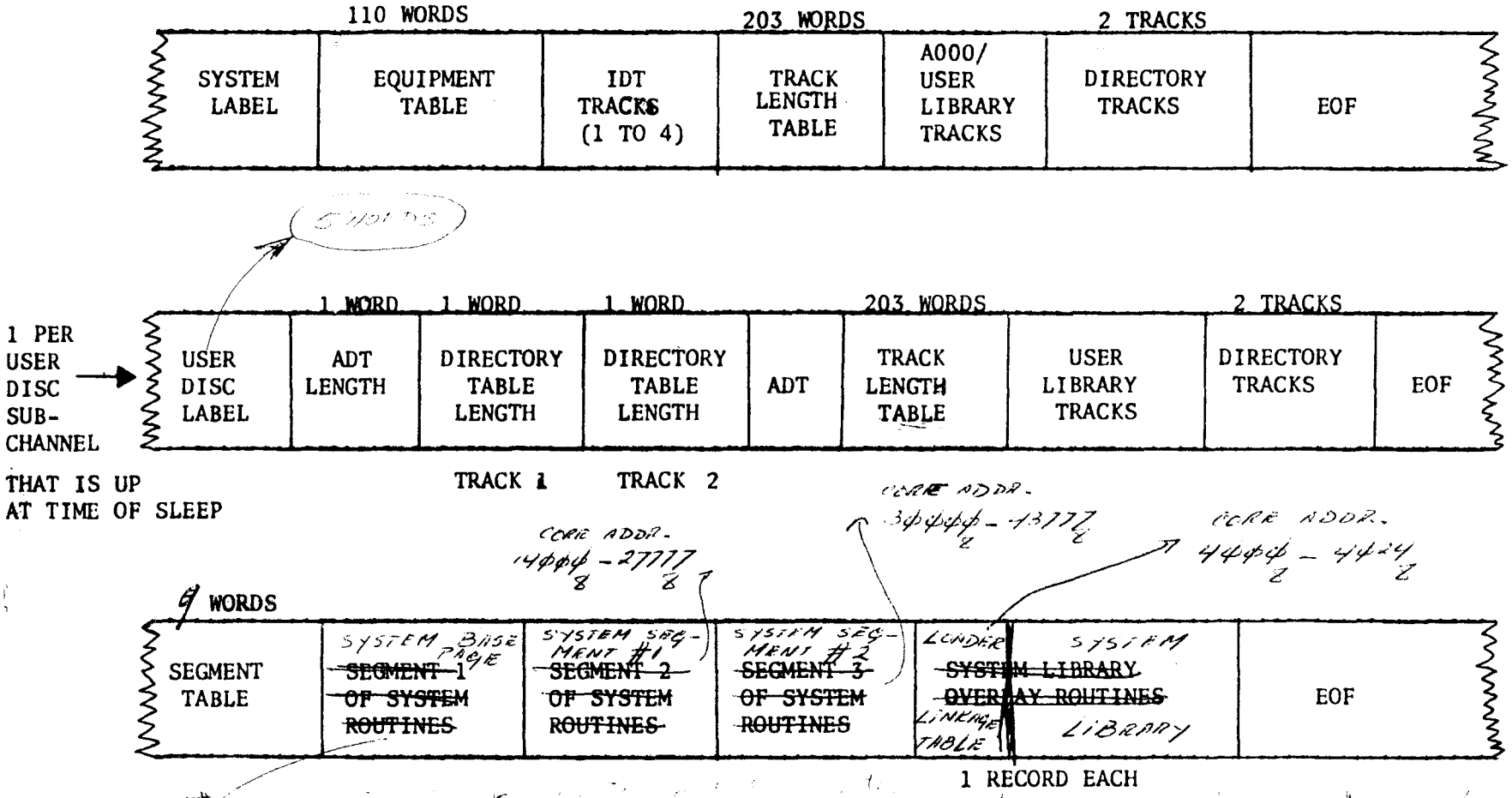
CORE RESIDENT

29 WORDS/PORT

NEW FORMAT

RESEMBLES 2000A

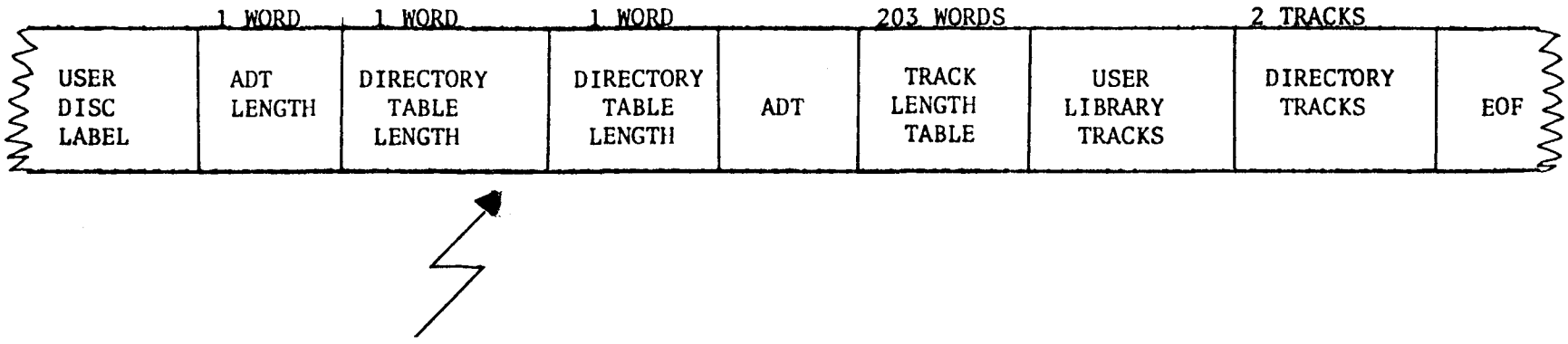
TTY TABLE



NOTE: EACH TRACK IS WRITTEN OUT TO MAG TAPE AS 1 OR 2 RECORDS. IF THE TRACK LENGTH TABLE INDICATES A TRACK IS OVER 3072 WORDS LONG, IT IS PUT INTO 2 RECORDS. THE FIRST RECORD IS 3072 WORDS, THE REMAINDER ON THE SECOND RECORD.

| IS NOT AN EOR MARKER

SLEEP TAPE FORMAT

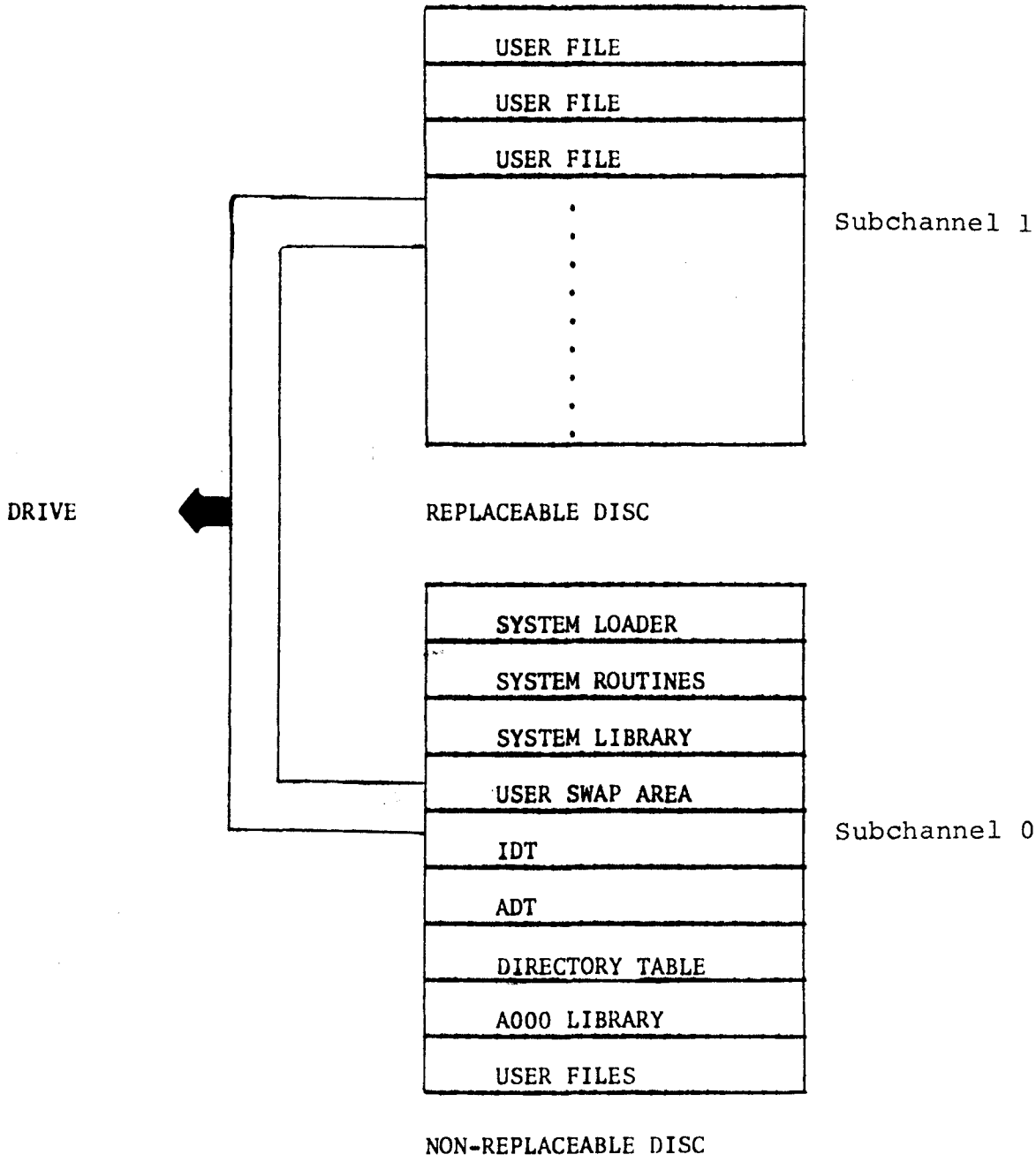


NOTE: EACH TRACK IS WRITTEN OUT TO MAG TAPE AS 1 OR 2 RECORDS. IF THE TRACK LENGTH TABLE INDICATES A TRACK IS OVER 3072 WORDS LONG, IT IS PUT INTO 2 RECORDS. THE FIRST RECORD IS 3072 WORDS, THE REMAINDER ON THE SECOND RECORD.

IS NOT AN EOR MARKER

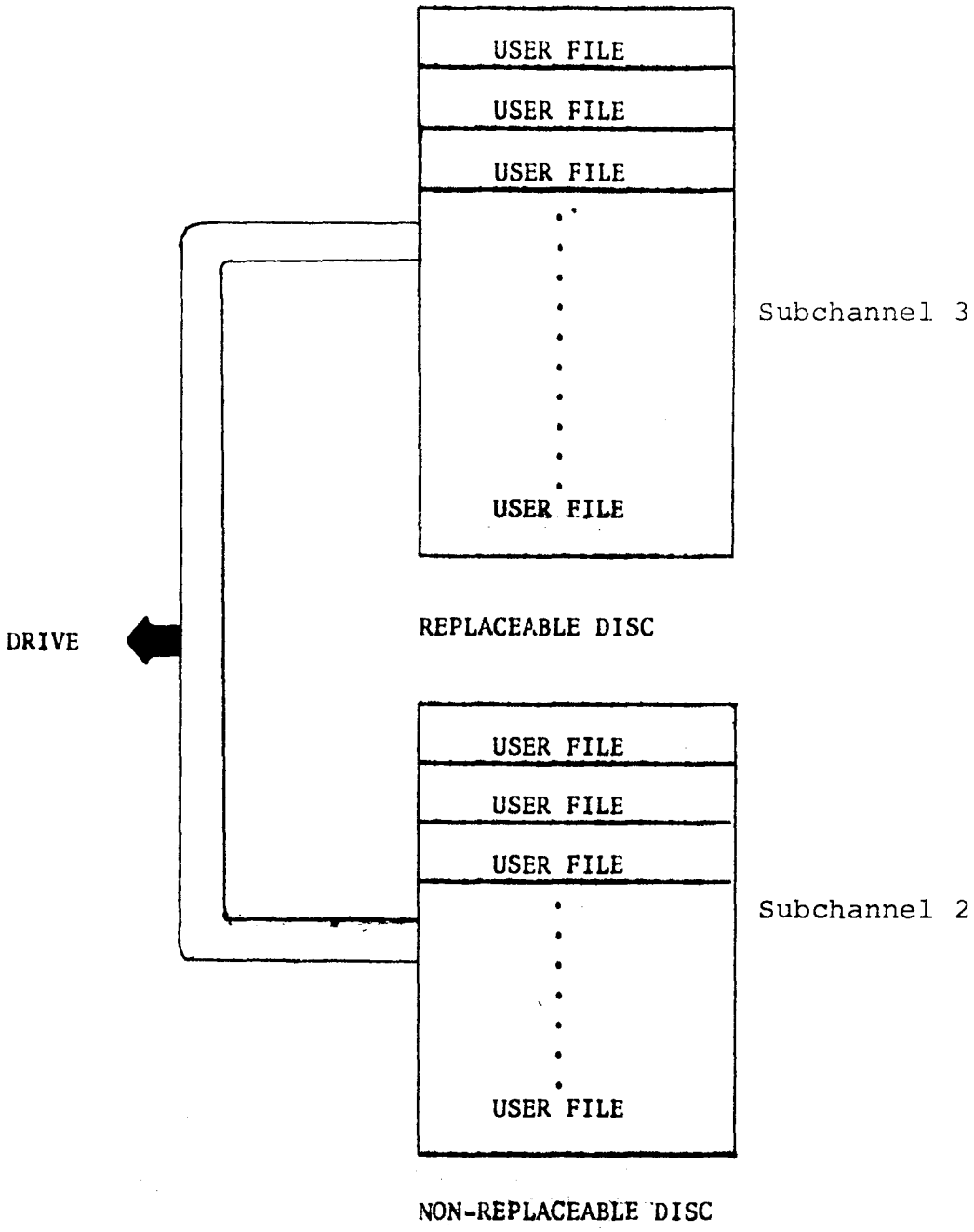
SELECTIVE DUMP MAG TAPE FORMAT (1 FILE)

Figure 14-1



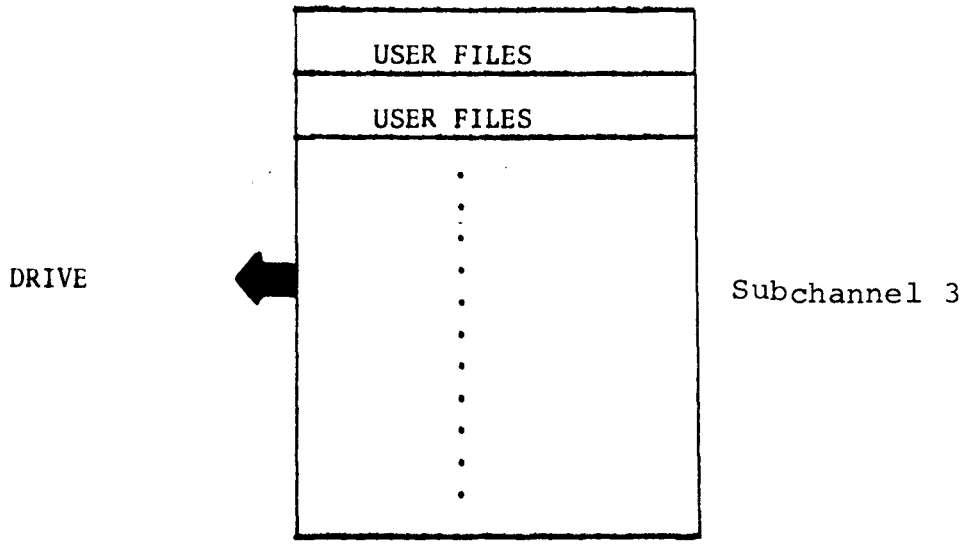
7900A DISC ORGANIZATION DRIVE 0

Figure 15



7900A DISC ORGANIZATION DRIVE 1

Figure 16



7901 DISC ORGANIZATION DRIVE 1

Figure 17

SYSTEM DISC (SUBCHANNEL ϕ)

TRACK	SECTOR	ASSIGNMENT
0	0	SYSTEM LABEL
0	1-8	SYSTEM LOADER (NOT BOOT-STRAP!! Loc. $2\phi\phi3_8 - 4\phi\phi1_8$
0	12-19	SYSTEM BASE PAGE Loc. $\phi - 1777_8$
1	0-47	LOADER UTILITY Loc. $4\phi\phi\phi_8$ - [illegible] $2\phi177_8$
2	0-47	SYSTEM SEGMENT #1 Loc. $14\phi\phi\phi_8 - 27777_8$
3	0-47	SYSTEM SEGMENT #2 Loc. $3\phi\phi\phi\phi_8 - 43777_8$
4	0-47	SYSTEM LIBRARY ROUTINES
5	0-47	
6	0-47	USER SWAP AREA
⋮	⋮	
21	0-47	
22	0-23	IDT
⋮	⋮	
25	0-23	
26	0-23	ADT
27	0-23	DIRECTORY
28	0-23	
29	0-47	$A\phi\phi\phi$ LIBRARY/USER FILES
⋮	⋮	
202	0-47	

USER DISC (SUBCHANNEL 1, 2 OR 3)

TRACK	SECTOR	ASSIGNMENT
0	0	USER DISC LABEL
0	1	LENGTH OF ADT LENGTH OF 1ST DIRECTORY TRACK LENGTH OF 2ND DIRECTORY TRACK
0	2-23	ADT
1	0-23	DIRECTORY
2	0-23	
3	0-47	USER FILES
⋮	⋮	
202	0-47	

SUBCHANNELS 1, 2, AND 3

TRACK	SECTOR	ASSIGNMENT
0	0	DISC LABEL
0	1	ADT LENGTH
		DIRECTORY TABLE, ON TRACK 1, LENGTH
		DIRECTORY TABLE, ON TRACK 2, LENGTH
0	2 - ~~47~~ 24	ADT
1	0 - 47	DIRECTORY TABLE
2	0 - 47	DIRECTORY TABLE
3 - 202	0 - 47	USER FILES

USER DISC TRACK ASSIGNMENTS

7900A AND 7901 DISC DRIVES

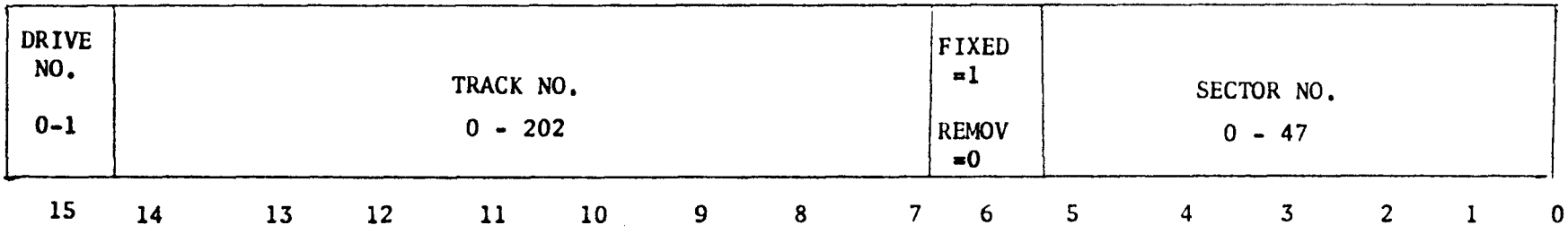
Figure 18

SUBCHANNEL 0

TRACK NO.		ASSIGNMENT
0	SECTOR 0	SYSTEM LABEL (sector 0)
0	SECTOR 1-8	SYSTEM LOADER
1		LOADER/UTILITY
1 ~~& 2~~	2-3	SYSTEM ROUTINES
~~3 & 4~~	4-5	SYSTEM LIBRARY ROUTINES
~~5 thru 20~~	6-21	USER SWAP AREA
~~21 thru 24~~	22-25 SECTORS 0-23	IDT
~~25~~	26 SECTORS 0-23	ADT
~~26 & 27~~	27-28 SECTORS 0-23	DIRECTORY TABLE
~~28 & 202~~	29-202	A000 LIBRARY/USER FILES

7900 FIXED DISC CARTRIDGE (SYSTEM)
TRACK ASSIGNMENTS

Figure 19



HEAD 0,2 - SECTORS 0 - 23

HEAD 1,3 - SECTORS 24 - 47

DISC ADDRESS FORMAT

Figure 20

2000E SYSTEM CRASH CORE LOCATIONS

LOC	VARIABLE	MEANING
1325	AREG	A-REGISTER AT LAST PROGRAM SUSPEND
1326	BREG	B-REGISTER AT LAST PROGRAM SUSPEND
1327	EREG	E-REGISTER AT LAST PROGRAM SUSPEND
1330	PREG	P-REGISTER AT LAST PROGRAM SUSPEND
264	MPCOM	BITS INDICATE TERMINALS ATTEMPTING TO COMMUNICATE WITH THE SCHEDULER
260	MAIN	ADDRESS OF TTY TABLE FOR PORT WHOSE SWAP TRACK IS CURRENTLY IN CORE (0-NO SWAP TRACK)
261	LIB	ADDRESS OF A WORD CONTAINING THE DISC ADDRESS OF THE LIBRARY PROGRAM OR OVERLAY CURRENTLY LOADED IN CORE AT ADDRESS 37300
262	ENDSK	0-NO DISC TRANSFER , 1-DISC TRANSFER INITIATED
65	WORD	WORD COUNT (-WORDS) OF LAST DISC TRANSFER
355	MLINK	BASIC QUEUE ENTRY
356	MLINK+1	POINTS TO HEAD OF QUEUE , IF=MLINK NOBODY IN QUEUE
307	DRIVE	DRIVE #, 0 OR 1
31335	LDISC	RETURN ADDRESS FROM LAST CALLER TO DISC DRIVER
31417	DINT	INTERRUPT RETURN ADDRESS FOR DISC DRIVER
306	FAIL	DISC RETURN COUNTER (-10 to 0)
273	DADDR	DISC ADDRESS OF LAST DISC TRANSFER
32000	POW	POWER FAIL INTERRUPT RESTART ADDRESS
1625	.LNUM	PROGRAM STATEMENT NUMBER UNDER EXECUTION
34164	CLKIN	CLOCK INTERRUPT RETURN ADDRESS
20	LTEMP	USED BY SYSTEM LIBRARY ROUTINES.
43	MOVES	SOURCE ADDRESS FROM MOVEW ROUTINE
44	MOVED	DESTINATION ADDRESS FOR MOVEW
100	DIREC	DIREC TABLE AND START OF EQUIPMENT TABLE
32270	?TT35	LAST CALLER TO SYSTEM CONSOLE DRIVER

Figure 21

BTEST

```
100   READ L
110   IF L<0 THEN 999
120   DATA 60000.
130   DATA -1
140   PRINT "START=";TIM(0)
150   X=0
160   X=X+1
170   IF X<L THEN 160
180   PRINT "X=";X;"STOP=";TIM(0)
190   GOTO 100
999   END
```

FPTS9

```
10   DATA 5000
20   READ L
30   K=0
40   PRINT "START=";TIM(0)
50   I=0
60   J=50
70   I=I+J+100*(I-J)↑2
80   I=I+I+I+I+I+I+I+I+I+I+I+I
90   J=I-J+I-J+I-J+I-J
100   I=J↑(-2)+I*I*I
110   K=K+1
120   IF K<L THEN 50
130   PRINT "STOP=";TIM(0)
140   END
```

Figure 22

PROGRAM NAME

System	BTEST	FPTS9	#Active Ports
C'	2 min. 2.8 secs	1 min. 58.2 secs	1
F	1 min. 26.5 secs	1 min. 16.4 secs	1
C	2 min. 2.5 secs	1 min. 58.2 secs	1
*E	1 min. 23.6 secs	1 min. 15.4 secs	1
**E	2 min. 56.8 secs	_____	2

*the difference between 2000E and 2000F can be attributed primarily to the following factors:

1. Scheduler idle loop is shorter
2. No communications processor on the 2000E

**with 2 ports active it would be expected that the run time be twice as long; anytime over this amount taken to be the swap time.

expected run time: 2 X (1 min. 23.6 secs) = 2 min. 47.2 secs
total swap time: 2 min. 56.8 secs - 2 min. 47.2 secs = 9.6 secs

expected no. of swaps: 1/sec run time or $\approx$ 83 swaps
swap time: 9.6 secs $\div$ 83 = 157 milliseconds
average seek time: 55 milliseconds

Figure 23

REFERENCES

2000E I/O CONFIGURATION

I/O Channel	
10	TBG
11-12	7900A DISC INTERFACE
13	OPERATOR'S CONSOLE
14	PAPER TAPE READER
15-16	MULTIPLEXER DATA BOARDS
17	MULTIPLEXER CONTROL BOARD

	CONNECT SLOT ON MUX BOARD
MULTIPLEXER DATA BOARD -	J18
CONTROL CARD CONN. P1 -	J16
CONTROL CARD CONN. P2 -	J20

HALTS

DISPLAY REGISTER	REASON
102004	Power failure
102005	Parity error
102010	Disc error - system routines
102011	Disc error - utility routines After "INSERT CARTRIDGE..." msg during SLEEP operation. Checksum error from BBL.
102033	After bootstrap operation of transferring system from subchannel 1 to subchannel 0.
102077	END OF TAPE during system generation. Successful completion of a SLEEP. Successful load when using BBL.
102066	Checksum error during system generation
102015	Sense switch 15 up during system generation
102001	Follows an error message being output to system console - utility routines
102055	Invalid address encountered during system generation Illegal address when using BBL.

```
0001                    ASMB,A,B,L,T
INIT1   010000
GET     010004
GOP     010012
LOOP    010027
PRNT    010040
P       010047
T       010063
TTYCW   010072
BEG     010073
WC      010074
M5      010075
CNT     010076
MSK     010077
.60B    010100
M9      010101
CR      010102
LF      010103
LINE    010104
BL      010105
FILL    010106
TTY35   000013
INIT2   011000
STAT    011052
STCMD   011070
DC      000011
CC      000012
POSN    011071
CW1     011072
CW2     011073
CW3     011074
RDCMD   011075
LOCN    011076
.128    011077
DRIVE   011100
** NO ERRORS*
```

```
0001                    ASMB,A,B,L,T
0002***
0003***
0004*************************$*********************@*****************************
0005***
0006***     THIS PROGRAM SELECTIVELY DUMPS TO THE SYS. CONSOLE CORE
0007***     OR 128 WORD SECTORS FROM THE MOVING HEAD DISC. LOAD USING
0008***     THE BRL (SA=37700)
0009***
0010***                 CORE DUMP :
0011***
0012***
0013***                         1. SET P=10000
0014***                         2. PUSH BOTH PRESETS AND RUN
0015***                         3. SET A= START ADDR.
0016***                         4. SET B= NO. OF WORDS TO BE DUMPED
0017***                         5. PUSH RUN
0018***                         6. HALT 77B ON COMPLETION
0019***
0020***                         TO RUN ANOTHER CASE - REPEAT STEPS 3-5
0021***
0022***
0023***
0024***                 DISC SECTOR DUMP
0025***
0026***
0027***                         1. SET P= 11000
0028***                         2. PUSH BOTH PRESETS AND RUN
0029****                        3. SET A=CYL. # BITS 0-7,DRIVE # IN BIT 15
0030***                         4. SET B=HEAD NO. BITS 8-9,SECTOR BITS 0-4
0031***                         5. PUSH RUN
0032***                         6. HALT 66B ON COMPLETION
0033***
0034***                         TO RUN ANOTHER CASE - REPEAT STEPS 3-5
0035***
0036***     SETTING S REGISTER BIT 15 = 1 CAUSES OUTPUT TO TERMINATE
0037***     AT THE END OF THE LINE IN PROCESS.
0038***
0039***
0040*************************$*********************@*****************************
0041***     CORE DUMP ENTRY PT = 10000
0042*************************$*********************@*****************************
0043***
0044***
0045  10000                 ORG 10000B
0046  10000 000000  INIT1   NOP           ENTRY
0047  10001 102077          HLT 77B       RELOAD A&B-REGISTERS
0048  10002 016004          JSB GET       GO TO DUMP
0049  10003 026000          JMP INIT1     LOOP FOR ANOTHER DUMP
0050***
0051*************************$*********************@*****************************
0052***     GET - IS USED TO PUT CORE CONTENTS ONTO THE SYSTEM CONSOLE -
0053*************************$*********************@*****************************
0054***
0055  10004 000000  GET     NOP           ENTRY
0056  10005 072073          STA BEG       SAVE BEGINING CORE LOCATION
```

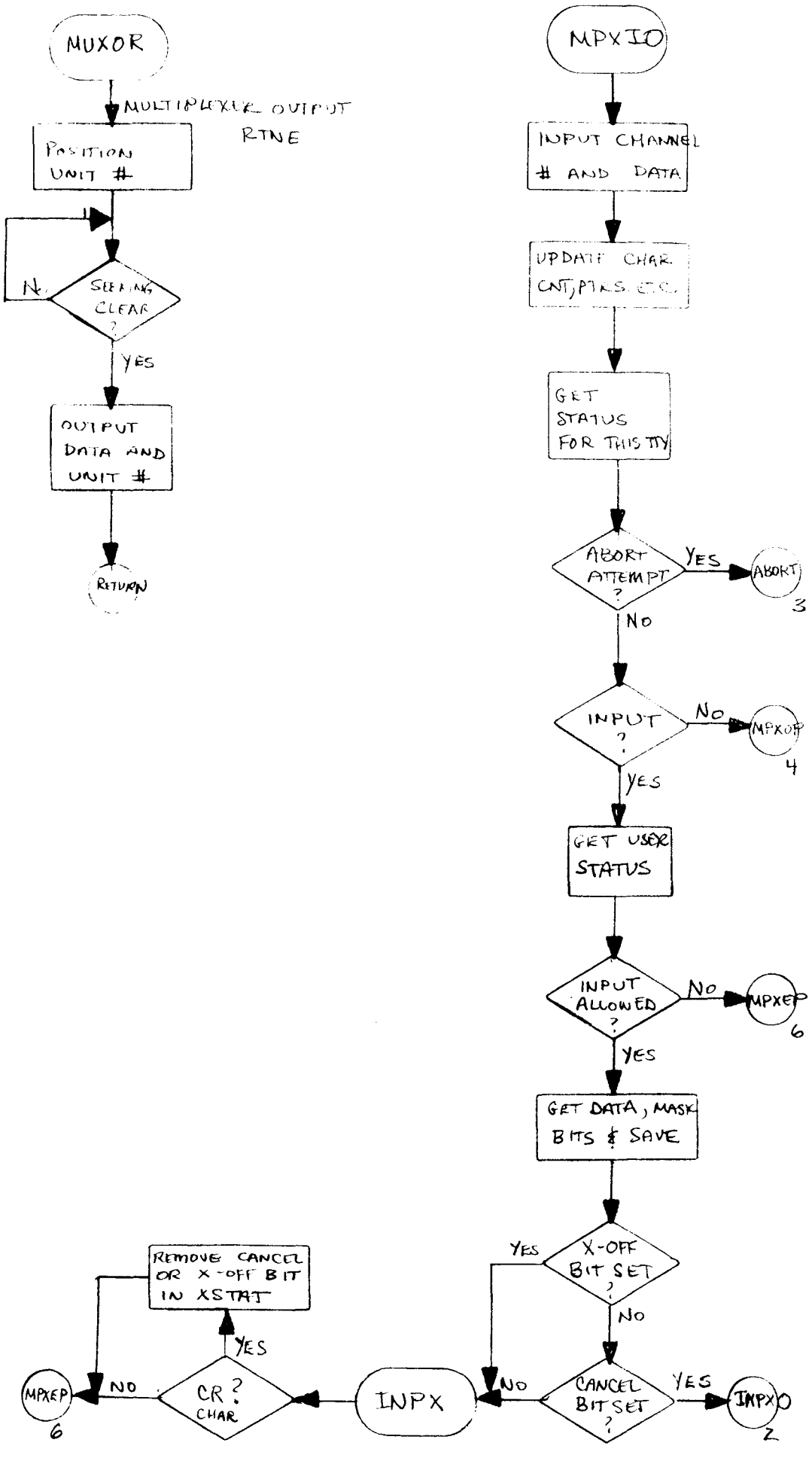
```
0057  1^006  007004          CMB,INB       NEGATE WORD COUNT
0058  1^007  076074          STB WC            AND SAVE
0059  1^010  062072          LDA TTYCW     OUTPUT PRINT CONTROL
0060  1^011  102613          OTA TTY35         WORD TO SYSTEM CONSOLE
0061  1^012  102501  GOP     LIA 1         CHECK BIT 15 OF SWITCH
0062  1^013  002020          SSA               REGISTER--IS IT SET?
0063  1^014  126004          JMP GET,I             YES, RETURN
0064  1^015  066101          LDB M9                NO
0065  1^016  076104          STB LINE      INITIALIZE COLUMN COUNT
0066  1^017  062102          LDA CR        OUTPUT
0067  1^020  016063          JSB T           A CARRIAGE-RETURN,
0068  1^021  062103          LDA LF            A LINE-FEED,
0069  1^022  016063          JSB T               AND A
0070  1^023  062106          LDA FILL              FILLER CHARACTER
0071  1^024  016063          JSB T                   TO THE CONSOLE
0072  1^025  066073          LDB BEG       PRINT THE CORE
0073  1^026  016040          JSB PRNT        LOCATION IN FIRST COLUMN
0074  1^027  036104  LOOP    ISZ LINE      INCREMENT COLUMN COUNTER
0075  1^030  026032          JMP *+2           MORE ON THIS LINE?
0076  1^031  026012          JMP GOP               NO, START NEW LINE
0077  1^032  166073          LDB BEG,I             YES
0078  1^033  016040          JSB PRNT      PRINT NEXT CORE-LOCATION CONTENTS
0079  1^034  036073          ISZ BEG       INCREMENT CORE ADDRESS
0080  1^035  036074          ISZ WC        IS THIS ALL?
0081  1^036  026027          JMP LOOP          NO, MORE WORDS TO GO
0082  1^037  126004          JMP GET,I         YES, RETURN
0083***
0084*****************************$*****************@*****************************
0085***     PRNT - IS A SUBROUTINE THAT PRINTS CONTENTS OF THE B REG.
0086***     AS AN OCTAL NUMBER.
0087*****************************$*****************@*****************************
0088***
0089  1^040  000000  PRNT    NOP           ENTRY
0090  1^041  002400          CLA           DECIDE WHETHER
0091  1^042  006020          SSB             SIGN BIT
0092  1^043  002004          INA               IS ZERO OR ONE
0093  1^044  016063          JSB T                 AND GO PRINT IT
0094  1^045  062075          LDA M5
0095  1^046  072076          STA CNT       SET UP COUNTER FOR 5-DIGITS
0096  1^047  005700  P       BLF
0097  1^050  006001          LDA 1
0098  1^051  005300          RRR
0099  1^052  012077          AND MSK       GET NEXT NUMERAL AND
0100  1^053  016063          JSB T           PRINT IT
0101  1^054  036076          ISZ CNT       MORE TO PRINT?
0102  1^055  026047          JMP P             YES
0103  1^056  062105          LDA BL            NO
0104  1^057  016063          JSB T         PRINT
0105  1^060  062105          LDA BL              TWO
0106  1^061  016063          JSB T             BLANKS
0107  1^062  126040          JMP PRNT,I    RETURN
0108***
0109*****************************$*****************@*****************************
0110***     T - IS A SUBROUTINE THAT PRINTS ONE CHAR SUPPLIED IN THE A
0111***     REGISTER ONTO THE SYSTEM CONSOLE
0112*****************************$*****************@*****************************
```

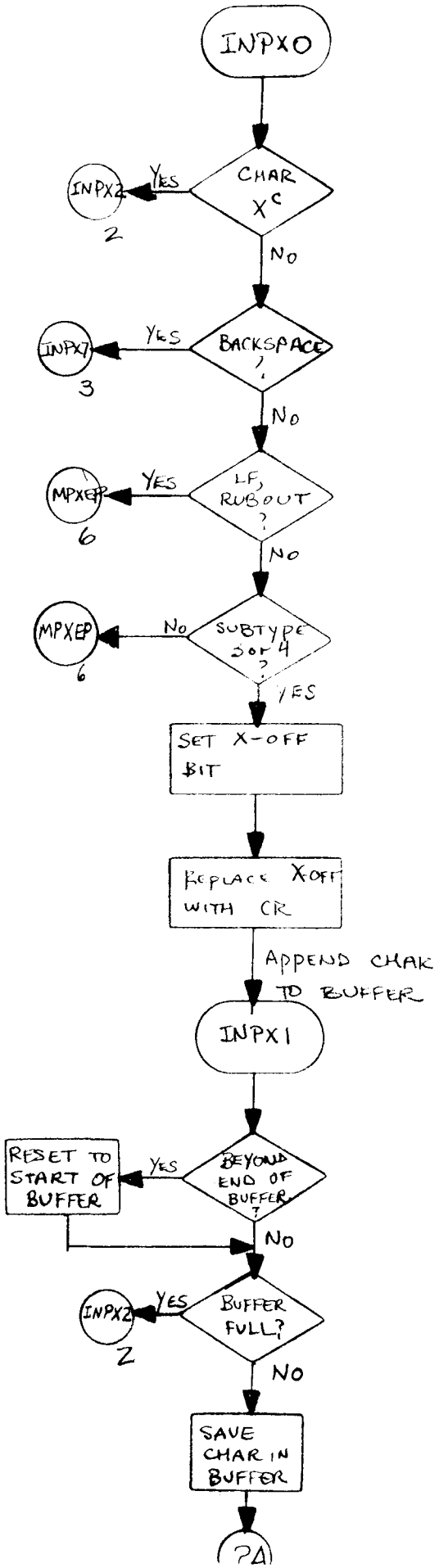
```
0113***
0114  1^063 000000  T      NOP             ENTRY
0115  1^064 042100         ADA .60B        CONVERT TO ASCII
0116  1^065 102613         OTA TTY35       OUTPUT
0117  1^066 103713         STC TTY35,C       TO
0118  1^067 102313         SFS TTY35           SYSTEM
0119  1^070 026067         JMP *-1               CONSOLE
0120  1^071 126063         JMP T,I         RETURN
0121***
0122***
0123*********************$*****************@*****************************
0124***    CONSTANTS,TEMPORARIES,EQUATES
0125*************************************************************************
0126***
0127***
0128  1^072 130000  TTYCW  OCT 130000      SYSTEM CONSOLE CONTROL WORD
0129  1^073 000000  BEG    BSS 1           BEGINING CORE LOCATION
0130  1^074 000000  WC     BSS 1           NEGATIVE WORD COUNT
0131  1^075 177773  M5     DEC -5          DIGITS PER WORD CONSTANT
0132  1^076 000000  CNT    BSS 1           COUNTER FOR DIGITS PER WORD
0133  1^077 000007  MSK    OCT 7           MASK FOR SINGLE OCTAL DIGIT
0134  1^100 000060  .60B   OCT 60          ASCII CONVERSION CONSTANT
0135  1^101 177767  M9     DEC -9          CONSTANT FOR COLUMNS PER LINE
0136  1^102 177732  CR     OCT 177732      CARRIAGE-RETURN CHARACTER
0137  1^103 177735  LF     OCT 177735      LINE-FEED CHARACTER
0138  1^104 000000  LINE   BSS 1           COUNTER FOR COLUMNS PER LINE
0139  1^105 177760  BL     OCT 177760      BLANK CHARACTER
0140  1^106 177720  FILL   OCT 177720      FILLER CHARACTER
0141  00013         TTY35  EQU 13B         SYSTEM CONSOLE SELECT CODE
0142*
0143***
0144***
0145***
0146***
0147***
0148*************************************************************************
0149***    DISC SECTOR DUMP ENTRY PT = 11000
0150*************************************************************************
0151***
0152***
0153  11000                 ORG 11000B
0154  11000 000000  INIT2  NOP             ENTRY
0155  11001 002400         CLA                 SETUP FOR DRIVE 0
0156  11002 073100         STA DRIVE
0157  11003 102066         HLT 66B         RELOAD A&B REGISTERS
0158  11004 002420         SSA                 IS THIS DRIVE 0
0159  11005 037100         ISZ DRIVE           YES
0160  11006 102611         OTA DC          OUTPUT CYL # TO DISC CU
0161  11007 103711         STC DC,C          AND SET CONTROL
0162  11010 063071         LDA POSN        OUTPUT POSITION CMND
0163  11011 043100         ADA DRIVE           SETUP FOR PROPER DRIVE
0164  11012 102612         OTA CC            TO DISC CU
0165  11013 106712         CLC CC          ENSURE RESPONSE
0166  11014 103712         STC CC,C          AND SET CONTROL
0167  11015 102311         SFS DC          CYL # ACCEPTED?
0168  11016 027015         JMP *-1           NO, WAIT
```

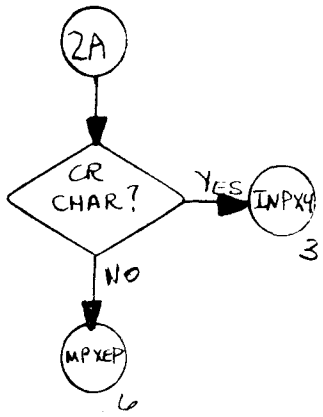
```
0169  11017 106611          OTB DC          YES, OUTPUT HD/SECT #
0170  11020 103711          STC DC,C          TO CU AND SET CONTROL
0171  11021 102312          SFS CC        IS POSITIONING COMPLETE?
0172  11022 027021          JMP *-1         NO, WAIT
0173  11023 017052          JSB STAT        YES, GO CHECK STATUS
0174  11024 063072          LDA CW1       STATUS IS OKAY
0175  11025 102606          OTA 6         SET
0176  11026 106702          CLC 2           UP
0177  11027 063073          LDA CW2          DMA
0178  11030 102602          OTA 2              FOR
0179  11031 102702          STC 2               INPUT
0180  11032 063074          LDA CW3                FROM
0181  11033 102602          OTA 2                   DISC
0182  11034 063075          LDA RDCMD     SEND READ COMMAND
0183  11035 043100          ADA DRIVE          SETUP FOR PROPER DRIVE
0184  11036 102612          OTA CC          TO DISC CU,
0185  11037 106712          CLC CC          ENSURE RESPONSE
0186  11040 103711          STC DC,C        PREPARE DISC DATA CH,
0187  11041 103706          STC 6,C         START DMA, AND
0188  11042 103712          STC CC,C        SIGNAL DISC CU
0189  11043 102312          SFS CC        HAS ALL DATA BEEN INPUT?
0190  11044 027043          JMP *-1           NO, WAIT
0191  11045 017052          JSB STAT          YES, CHECK STATUS
0192  11046 063076          LDA LOCN      STATUS OK, GET DATA LOCATION
0193  11047 067077          LDB .128      GET WORD COUNT
0194  11050 016004          JSB GET       GO DUMP TO SYSTEM CONSOLE
0195  11051 027001          JMP INIT2+1   LOOP FOR MORE DUMPS
0196***
0197*****************************************************************************
0198***     STAT - CHECKS THE STATUS OF THE DISC
0199*****************************************************************************
0200***
0201  11052 000000  STAT    NOP           ENTRY
0202  11053 063070          LDA STCMD     GET STATUS COMMAND CODE
0203  11054 043100          ADA DRIVE          SETUP FOR PROPER DRIVE
0204  11055 103711          STC DC,C      PREPARE DATA CH TO RECEIVE STATUS
0205  11056 102612          OTA CC        SEND COMMAND TO DISC CU,
0206  11057 106712          CLC CC          ENSURE RESPONSE AND
0207  11060 103712          STC CC,C        SET CONTROL
0208  11061 102311          SFS DC        IS STATUS WORD THERE?
0209  11062 027061          JMP *-1           NO, WAIT
0210  11063 102511          LIA DC            YES, GET IT
0211  11064 106711          CLC DC
0212  11065 002020          SSA           IS STATUS ERROR BIT SET?
0213  11066 102011          HLT 11B           YES, HALT
0214  11067 127052          JMP STAT,I        NO, RETURN
0215***
0216***
0217*****************************************************************************
0218***     CONSTANTS,TEMPORARIES,EQUATES
0219*****************************************************************************
0220***
0221***
0222  11070 000000  STCMD   OCT 00000      STATUS COMMAND CODE
0223  00011         DC      EQU 11B       DISC DATA CH SELECT CODE
0224  00012         CC      EQU 12B       DISC COMMAND CH SELECT CODE
```

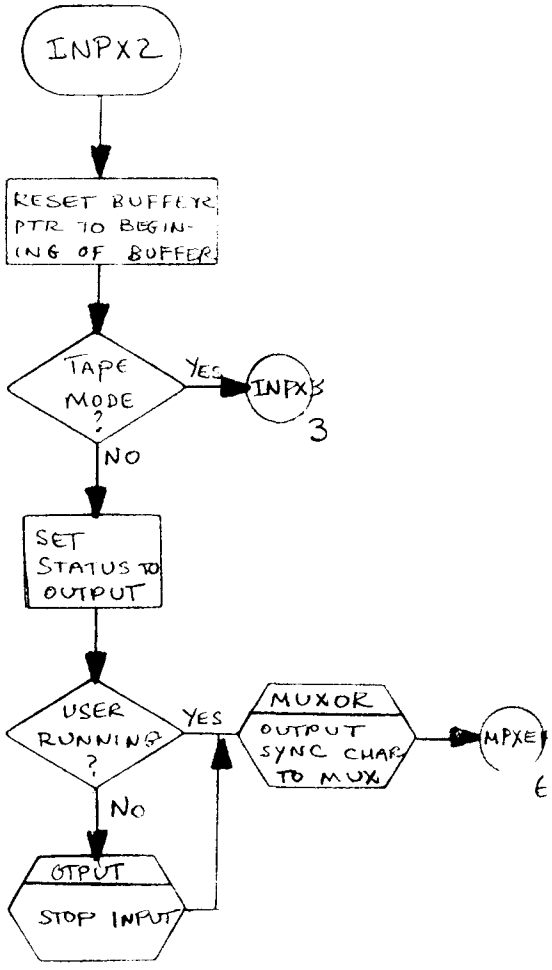
```
0225   11071 030000   POSN   OCT 30000     POSITION COMMAND CODE
0226   11072 120011   CW1    OCT 120011    DMA CW1
0227   11073 112000   CW2    OCT 112000    DMA CW2
0228   11074 177600   CW3    DEC -128      DMA CW3
0229   11075 020000   RDCMD  OCT 20000     READ COMMAND CODE
0230   11076 012000   LOCN   OCT 12000     CORE LOCATION OF DATA
0231   11077 000200   .128   DEC 128       WORD COUNT OF DUMP
0232   11100 000000   DRIVE  BSS 1               DRIVE NO., 0 OR 1
0233                         END
** NO ERRORS*
```

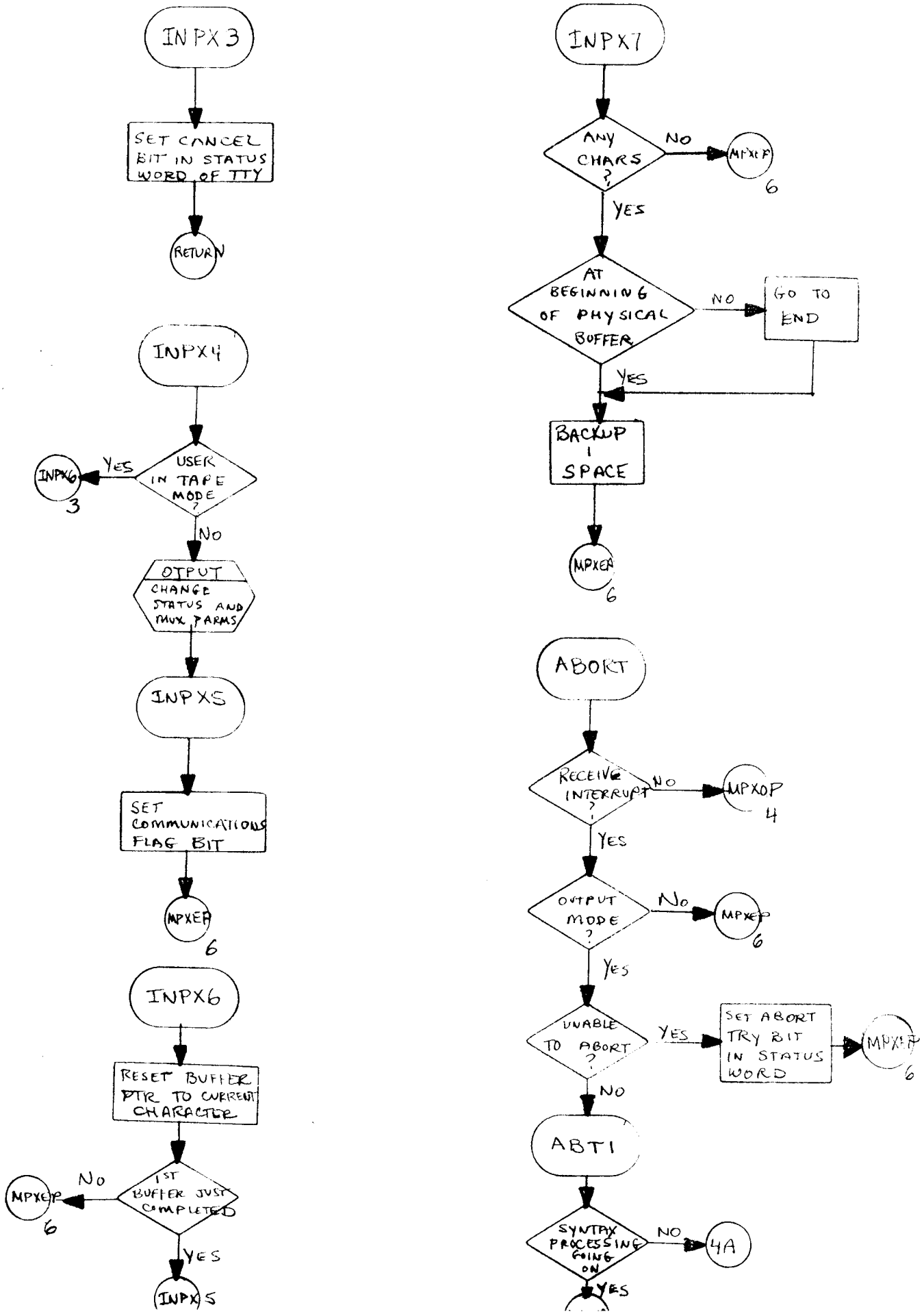
MULTIPLEXER ROUTINES

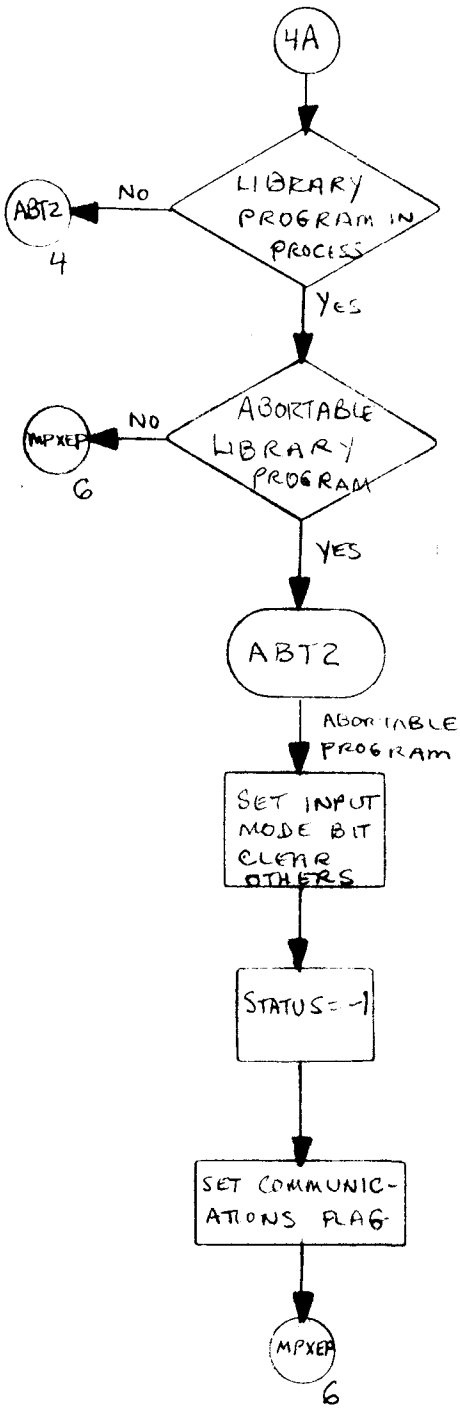


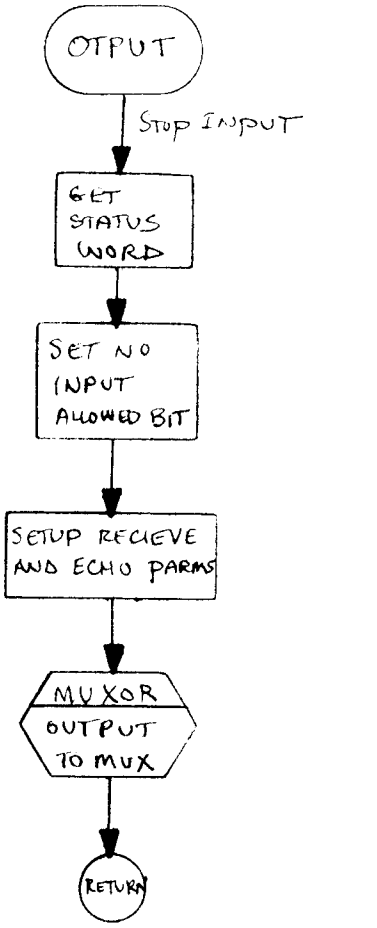

INPX0
CHAR X^C
YES
INPX2
2
NO
BACKSPACE ?
YES
INPX1
3
NO
LF, RUBOUT ?
YES
MPXEP
6
NO
SUBTYPE 3 OR 4 ?
NO
MPXEP
6
YES
SET X-OFF BIT
REPLACE XOFF WITH CR
APPEND CHAR TO BUFFER
INPX1
BEYOND END OF BUFFER ?
YES
RESET TO START OF BUFFER
NO
BUFFER FULL?
YES
INPX2
2
NO
SAVE CHAR IN BUFFER
2A

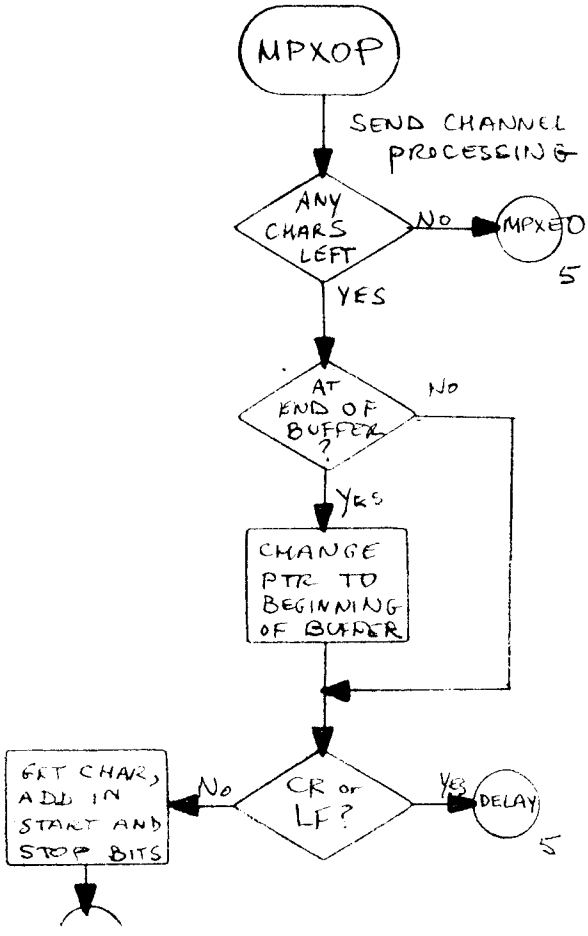

2A
CR CHAR?
YES
INPX4
3
NO
MPXEP
6

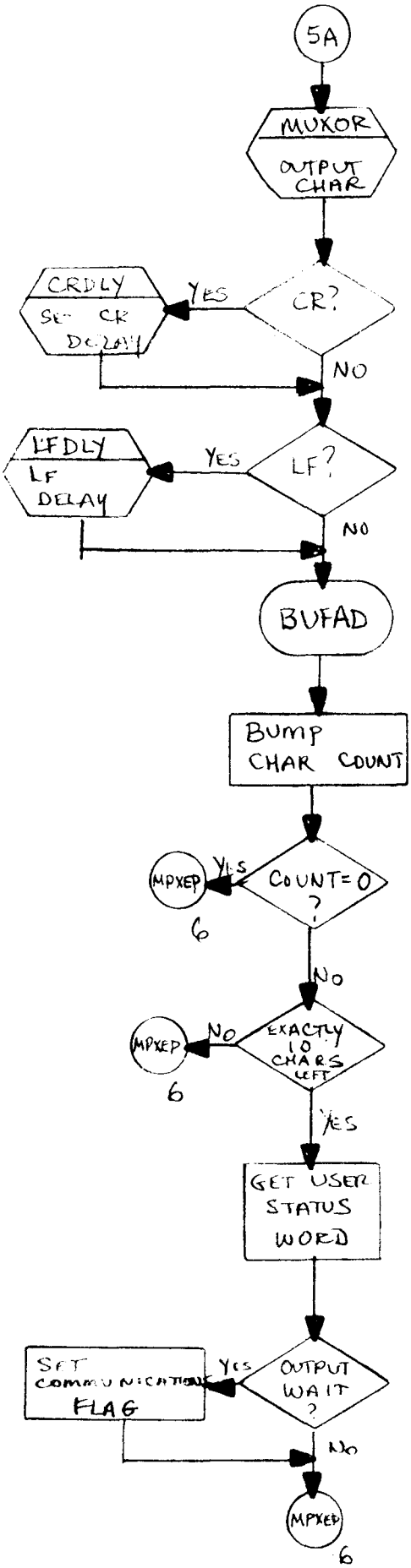

INPX2
RESET BUFFER PTR TO BEGINING OF BUFFER
TAPE MODE ?
YES
INPX3
3
NO
SET STATUS TO OUTPUT
USER RUNNING ?
YES
MUXOR OUTPUT SYNC CHAR TO MUX
MPXEP
6
No
OTPUT STOP INPUT

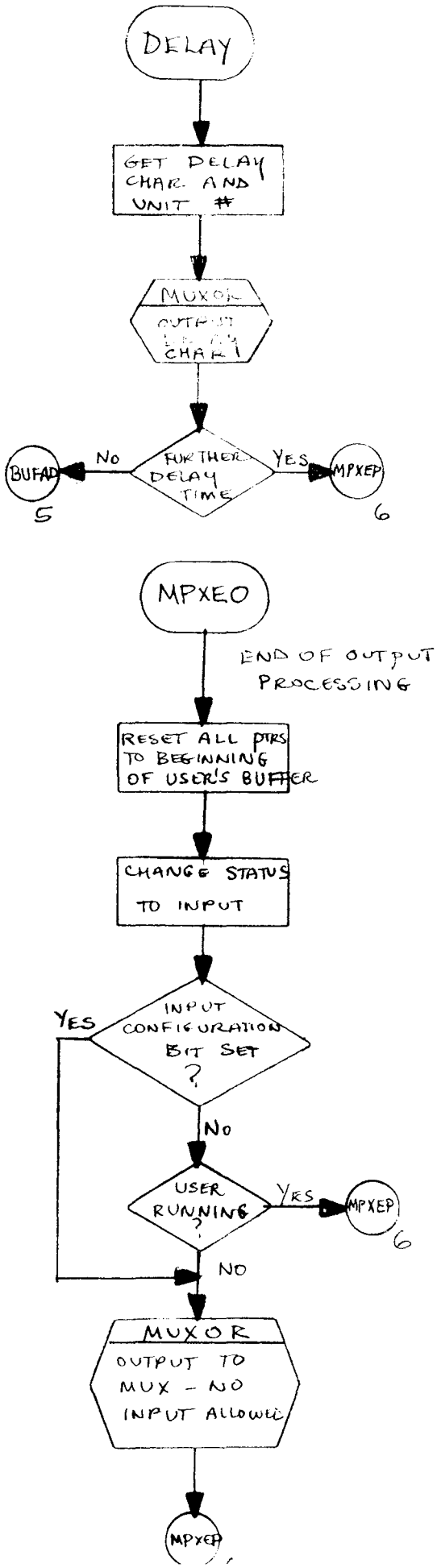

INPX3
SET CANCEL BIT IN STATUS WORD OF TTY
RETURN
INPX4
USER IN TAPE MODE ?
YES
INPX6
3
NO
OTPUT
CHANGE STATUS AND MUX PARMS
INPX5
SET COMMUNICATIONS FLAG BIT
MPXEP
6
INPX6
RESET BUFFER PTR TO CURRENT CHARACTER
1ST BUFFER JUST COMPLETED
NO
MPXEP
6
YES
INPX5
INPX7
ANY CHARS ?
NO
MPXEP
6
YES
AT BEGINNING OF PHYSICAL BUFFER
NO
GO TO END
YES
BACKUP 1 SPACE
MPXEP
6
ABORT
RECEIVE INTERRUPT ?
NO
MPXOP
4
YES
OUTPUT MODE ?
NO
MPXEP
6
YES
UNABLE TO ABORT ?
YES
SET ABORT TRY BIT IN STATUS WORD
MPXEP
6
NO
ABT1
SYNTAX PROCESSING GOING ON
NO
4A
YES

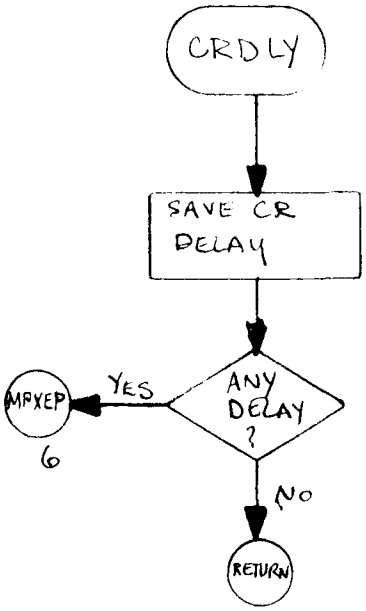

4A
LIBRARY PROGRAM IN PROCESS
No
ABT2
4
Yes
ABORTABLE LIBRARY PROGRAM
No
MPXEP
6
YES
ABT2
ABORTABLE PROGRAM
SET INPUT MODE BIT CLEAR OTHERS
STATUS = -1
SET COMMUNICATIONS FLAG
MPXEP
6

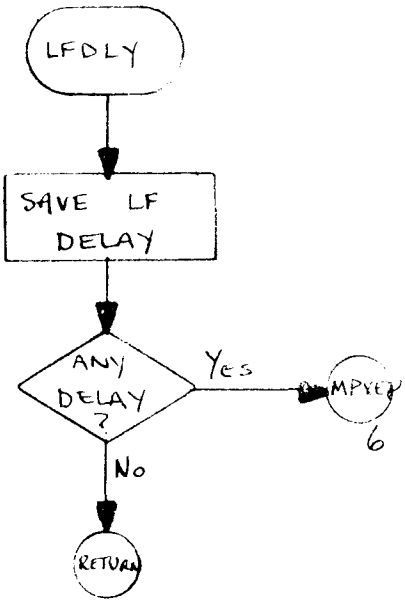

OTPUT
STOP INPUT
GET STATUS WORD
SET NO INPUT ALLOWED BIT
SETUP RECIEVE AND ECHO PARMS
MUXOR OUTPUT TO MUX
RETURN

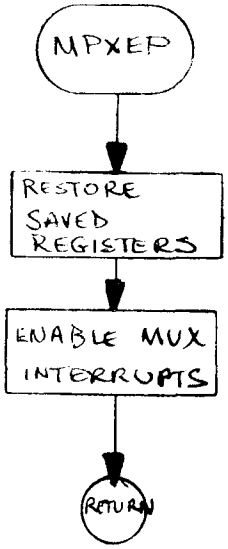

MPXOP
SEND CHANNEL PROCESSING
ANY CHARS LEFT
No
MPXEO
5
YES
AT END OF BUFFER ?
No
Yes
CHANGE PTR TO BEGINNING OF BUFFER
GET CHAR, ADD IN START AND STOP BITS
No
CR or LF ?
Yes
DELAY
5


5A
MUXOR
OUTPUT CHAR
CR?
YES
CRDLY
SET CR DELAY
NO
LF?
YES
LFDLY
LF DELAY
NO
BUFAD
Bump CHAR COUNT
COUNT=0 ?
YES
MPXEP
6
NO
EXACTLY 10 CHARS LEFT
NO
MPXEP
6
YES
GET USER STATUS WORD
OUTPUT WAIT ?
YES
SET COMMUNICATIONS FLAG
NO
MPXEP
6

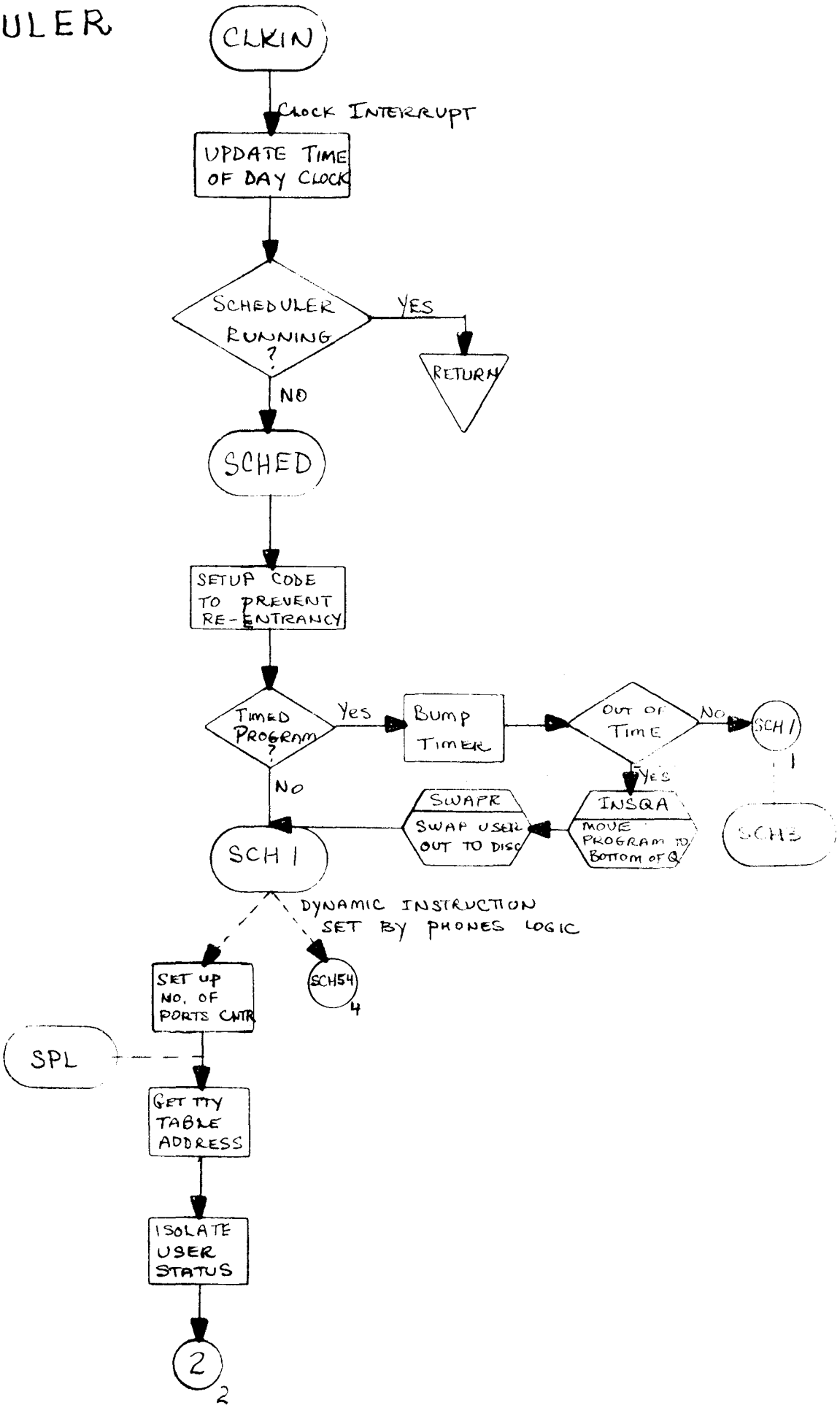

DELAY
GET DELAY CHAR. AND UNIT #
MUXOR
OUTPUT CHAR
FURTHER DELAY TIME
NO
BUFAD
5
YES
MPXEP
6
MPXEO
END OF OUTPUT PROCESSING
RESET ALL PTRS TO BEGINNING OF USER'S BUFFER
CHANGE STATUS TO INPUT
INPUT CONFIGURATION BIT SET ?
YES
NO
USER RUNNING ?
YES
MPXEP
6
NO
MUXOR
OUTPUT TO MUX - NO INPUT ALLOWED
MPXEP

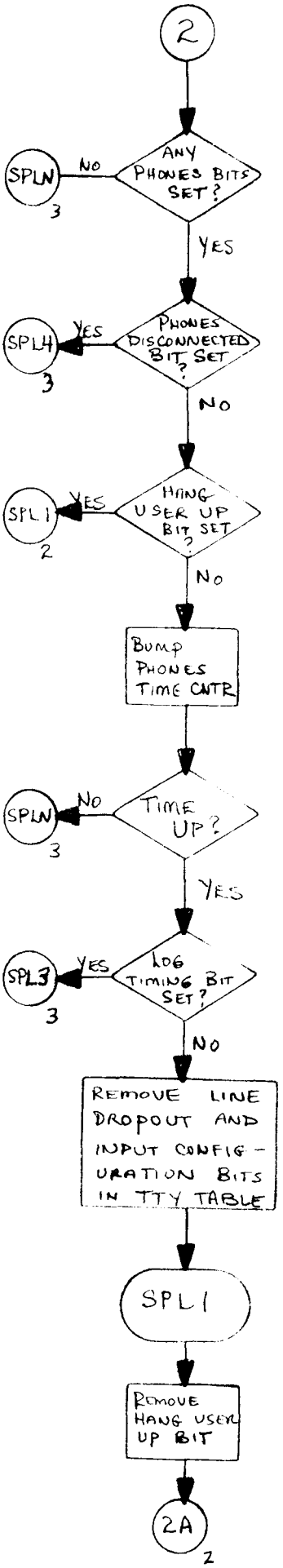

CRDLY
SAVE CR DELAY
ANY DELAY ?
YES
MPXEP
6
No
RETURN

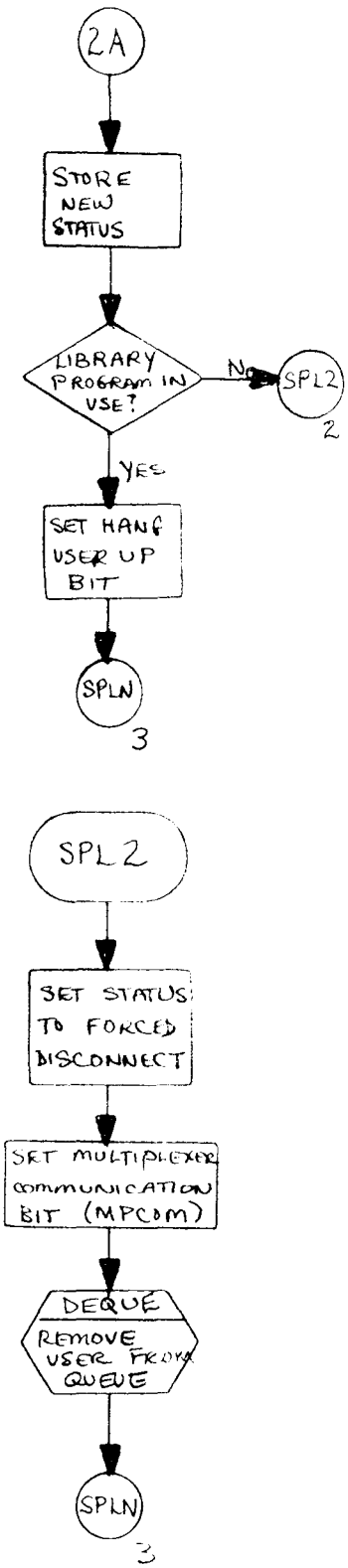

LFDLY
SAVE LF DELAY
ANY DELAY ?
Yes
MPXEP
6
No
RETURN

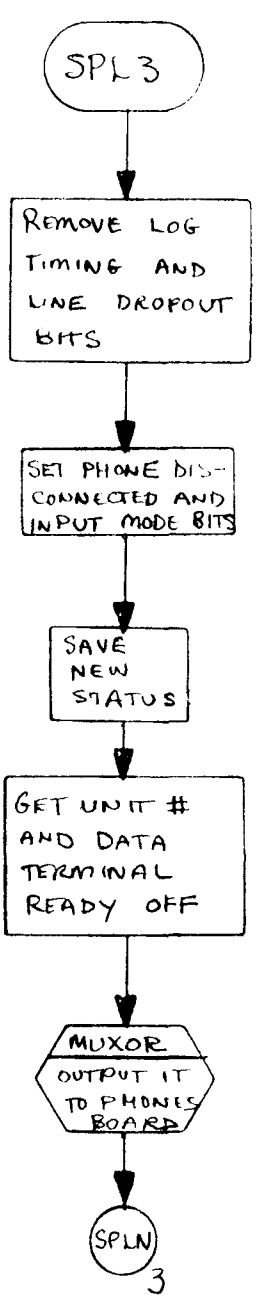

MPXEP
RESTORE SAVED REGISTERS
ENABLE MUX INTERRUPTS
RETURN

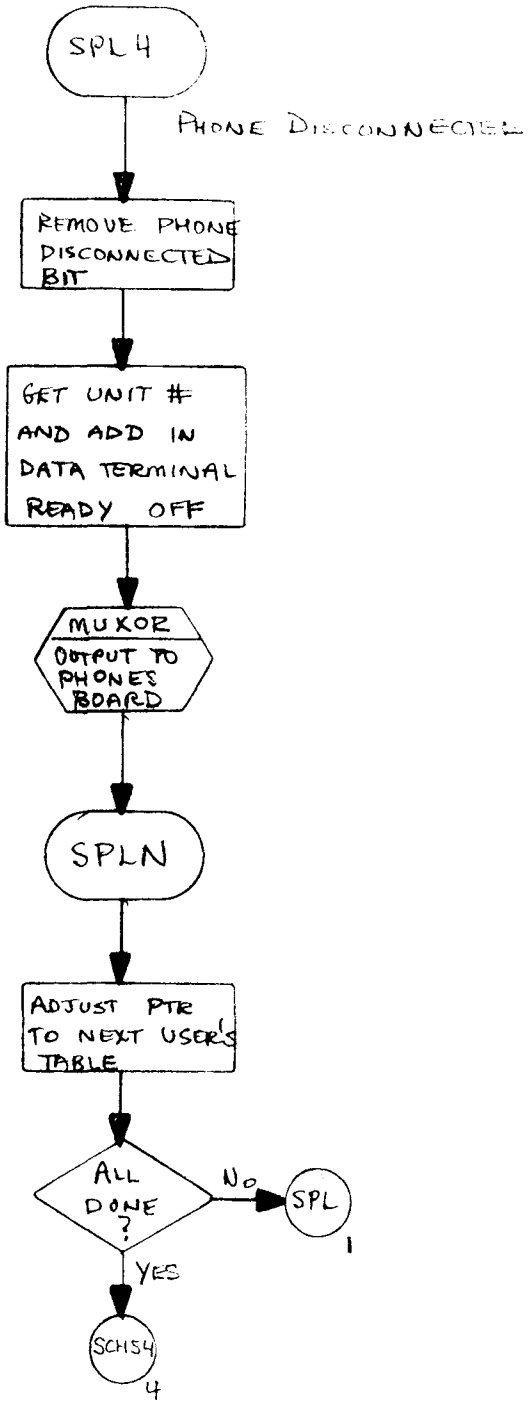
SCHEDULER

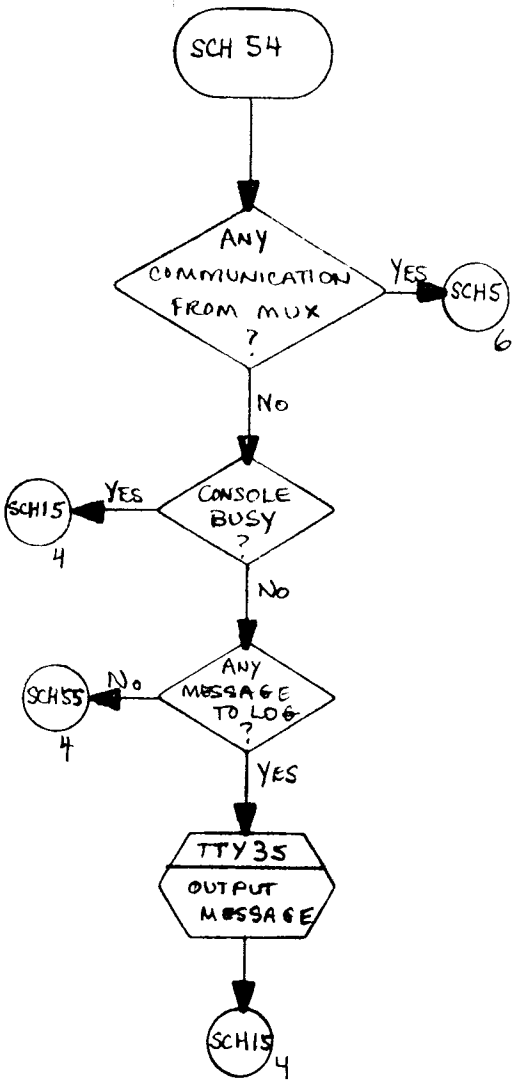

CLKIN
CLOCK INTERRUPT
UPDATE TIME OF DAY CLOCK
SCHEDULER RUNNING ?
YES
RETURN
NO
SCHED
SETUP CODE TO PREVENT RE-ENTRANCY
TIMED PROGRAM ?
Yes
Bump Timer
OUT OF TIME
NO
SCH 1
YES
INSQA
MOVE PROGRAM TO BOTTOM OF Q
SCHE
SWAPR
SWAP USER OUT TO DISC
NO
SCH 1
DYNAMIC INSTRUCTION SET BY PHONES LOGIC
SET UP NO. OF PORTS CNTR
SCH54
4
SPL
GET TTY TABLE ADDRESS
ISOLATE USER STATUS
2
2

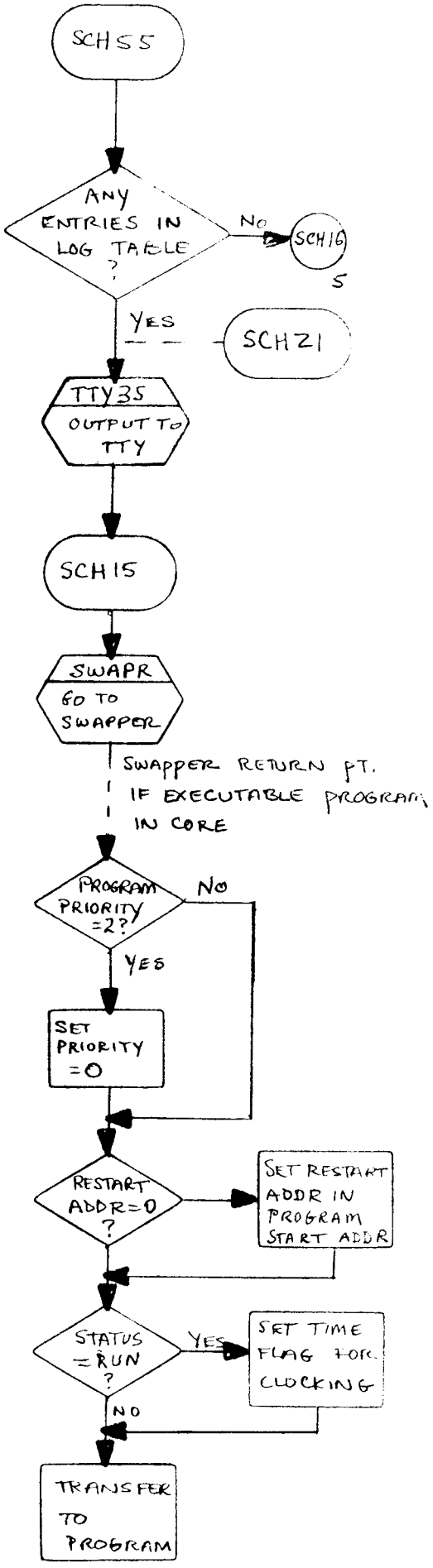

2
ANY PHONES BITS SET?
No
SPLN
3
Yes
PHONES DISCONNECTED BIT SET?
Yes
SPLH
3
No
HANG USER UP BIT SET?
Yes
SPL1
2
No
BUMP PHONES TIME CNTR
TIME UP?
No
SPLN
3
Yes
LOG TIMING BIT SET?
Yes
SPL3
3
No
REMOVE LINE DROPOUT AND INPUT CONFIGURATION BITS IN TTY TABLE
SPL1
REMOVE HANG USER UP BIT
2A
2

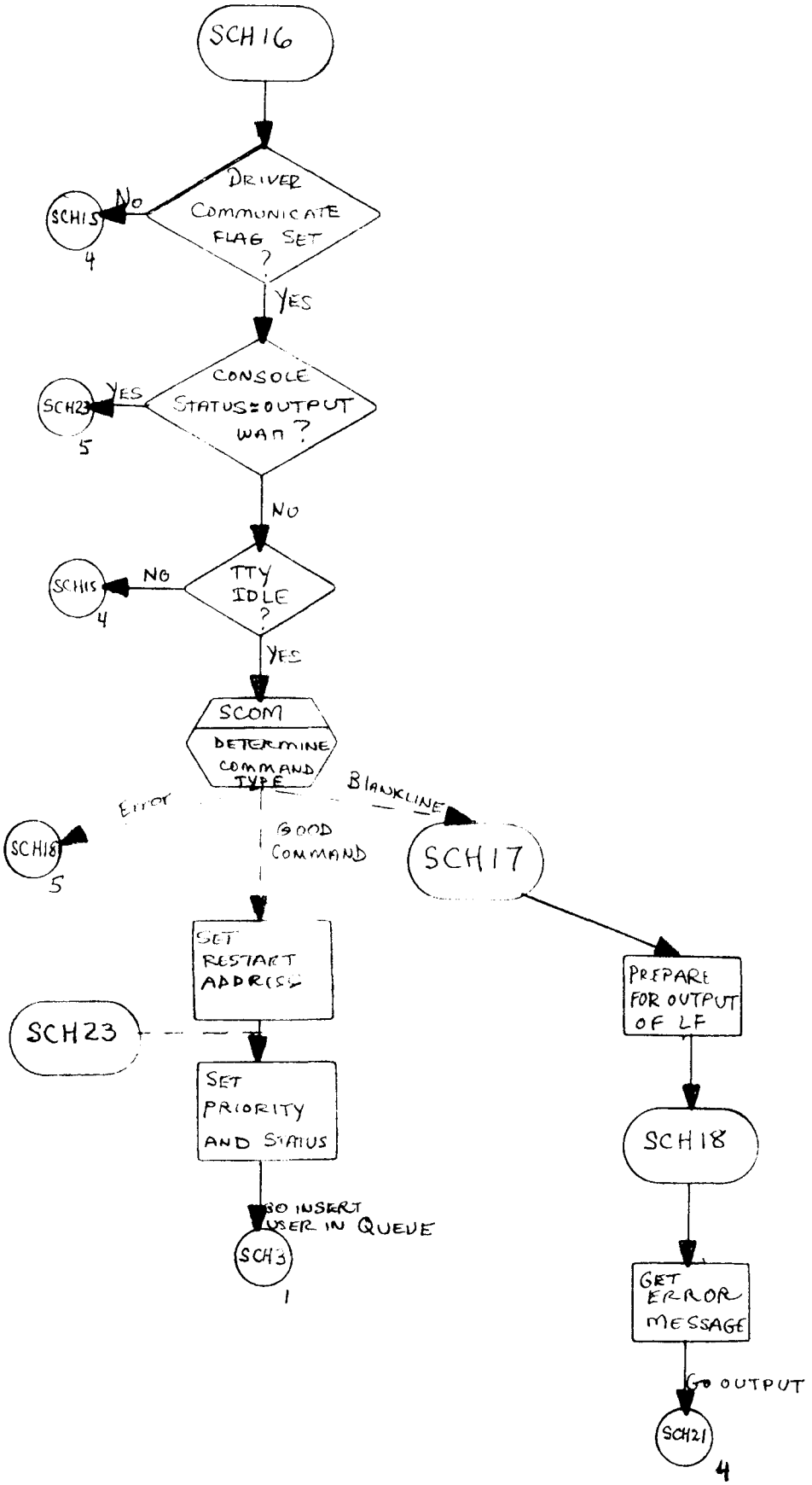

2A
STORE NEW STATUS
LIBRARY PROGRAM IN USE?
No
SPL2
2
Yes
SET HANG USER UP BIT
SPLN
3
SPL2
SET STATUS TO FORCED DISCONNECT
SET MULTIPLEXER COMMUNICATION BIT (MPCOM)
DEQUE
REMOVE USER FROM QUEUE
SPLN
3

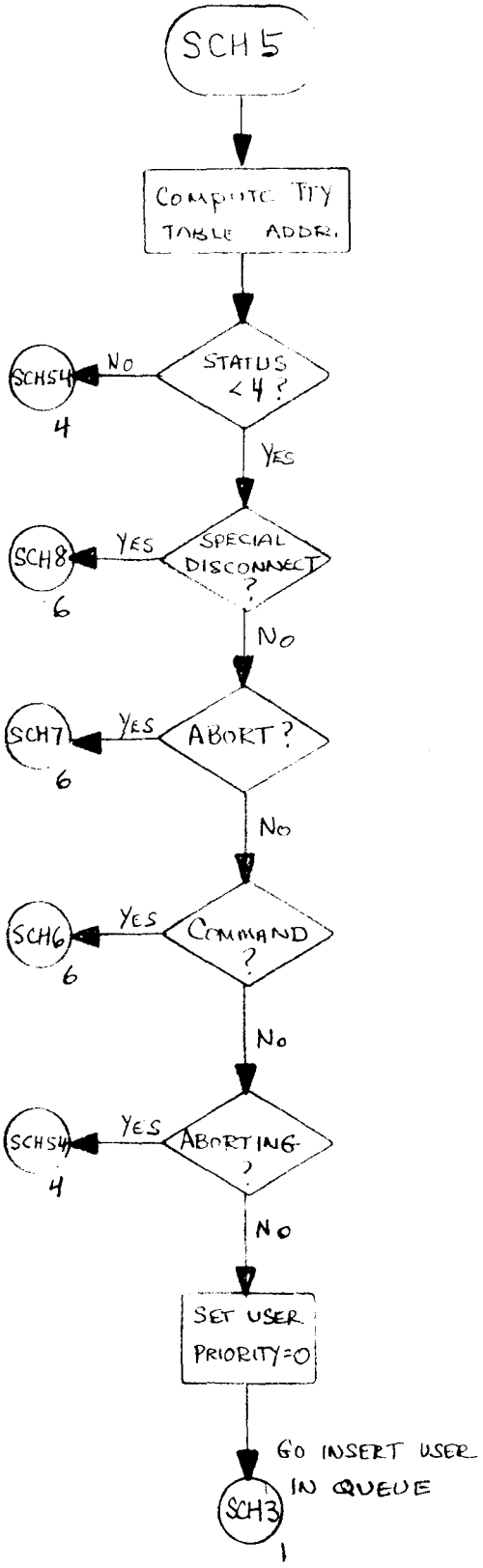

SPL3
REMOVE LOG TIMING AND LINE DROPOUT BITS
SET PHONE DIS-CONNECTED AND INPUT MODE BITS
SAVE NEW STATUS
GET UNIT # AND DATA TERMINAL READY OFF
MUXOR
OUTPUT IT TO PHONES BOARD
SPLN
3

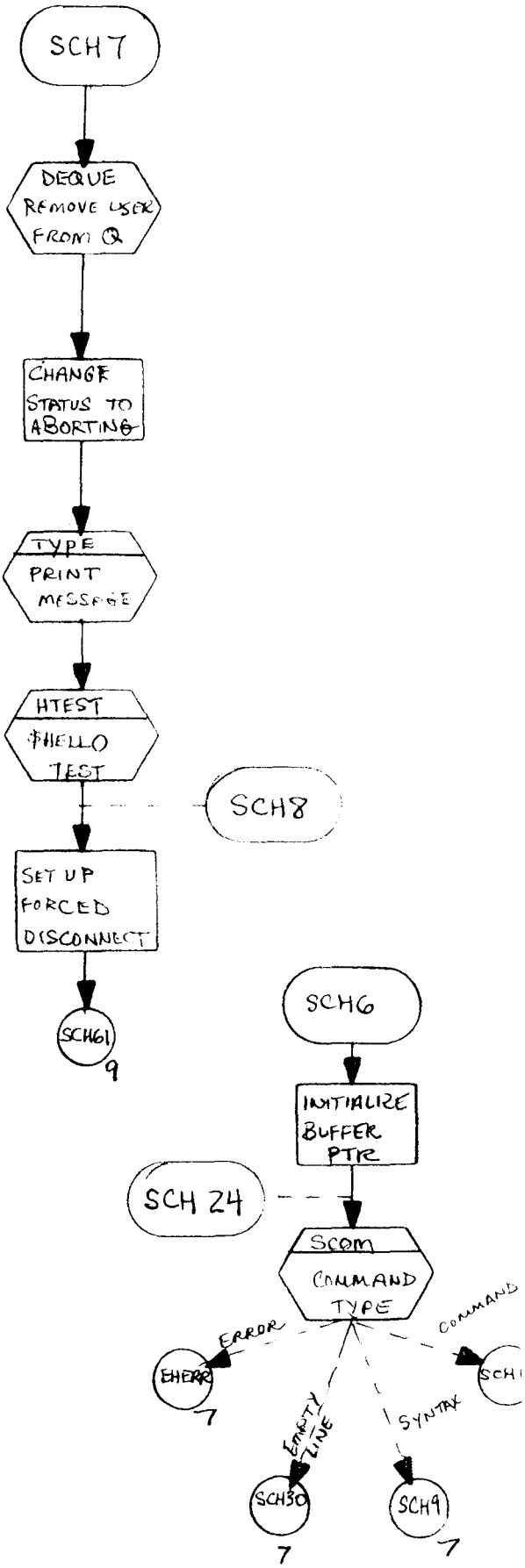

SPL4
PHONE DISCONNECTED
REMOVE PHONE DISCONNECTED BIT
GET UNIT # AND ADD IN DATA TERMINAL READY OFF
MUXOR
OUTPUT TO PHONES BOARD
SPLN
ADJUST PTR TO NEXT USER'S TABLE
ALL DONE ?
NO
SPL
1
YES
SCHS4
4

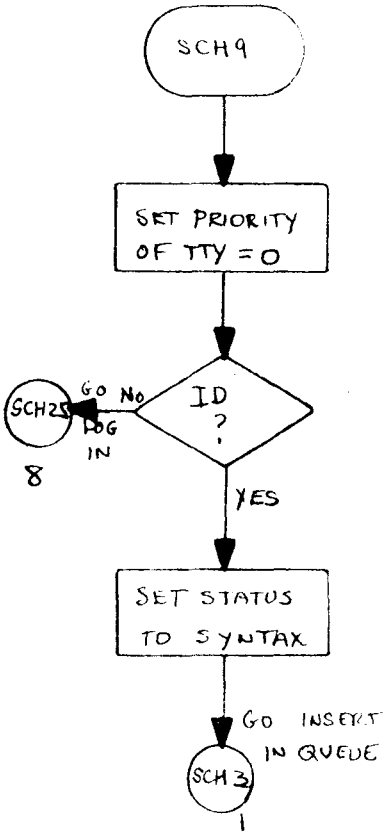

SCH 54
ANY COMMUNICATION FROM MUX ?
YES
SCH5
6
No
CONSOLE BUSY ?
YES
SCH15
4
No
ANY MESSAGE TO LOG ?
No
SCH55
4
YES
TTY35
OUTPUT MESSAGE
SCH15
4

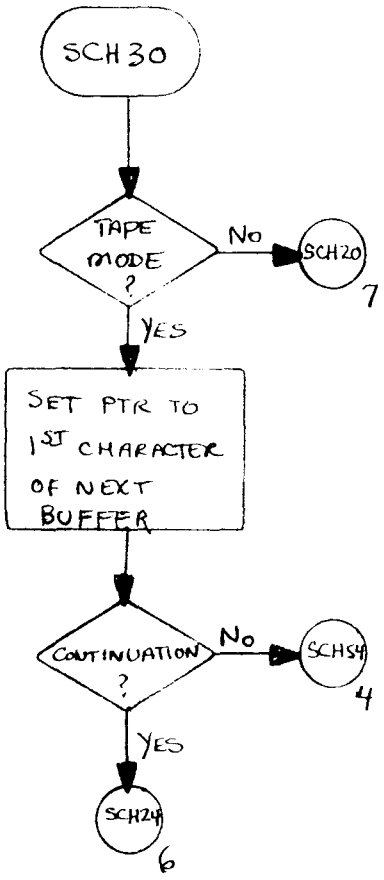

SCH55
ANY ENTRIES IN LOG TABLE ?
No
SCH16
5
YES
SCH21
TTY35
OUTPUT TO TTY
SCH15
SWAPR
GO TO SWAPPER
SWAPPER RETURN PT. IF EXECUTABLE PROGRAM IN CORE
PROGRAM PRIORITY =2?
No
YES
SET PRIORITY =0
RESTART ADDR=0 ?
SET RESTART ADDR IN PROGRAM START ADDR
STATUS =RUN ?
YES
SET TIME FLAG FOR CLOCKING
NO
TRANSFER TO PROGRAM

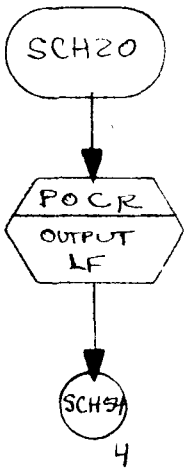

SCH16
DRIVER COMMUNICATE FLAG SET ?
No
SCH15
4
YES
CONSOLE STATUS=OUTPUT WAIT ?
YES
SCH23
5
NO
TTY IDLE ?
NO
SCH15
4
YES
SCOM
DETERMINE COMMAND TYPE
ERROR
SCH18
5
GOOD COMMAND
BLANKLINE
SCH17
SET RESTART ADDRESS
SCH23
SET PRIORITY AND STATUS
GO INSERT USER IN QUEUE
SCH3
1
PREPARE FOR OUTPUT OF LF
SCH18
GET ERROR MESSAGE
GO OUTPUT
SCH21
4

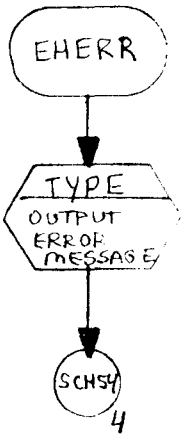

SCH 5
COMPUTE TTY TABLE ADDR.
STATUS < 4 ?
No
SCH54
4
Yes
SPECIAL DISCONNECT ?
Yes
SCH8
6
No
ABORT ?
Yes
SCH7
6
No
COMMAND ?
Yes
SCH6
6
No
ABORTING ?
Yes
SCH54
4
No
SET USER PRIORITY=0
GO INSERT USER IN QUEUE
SCH3
1

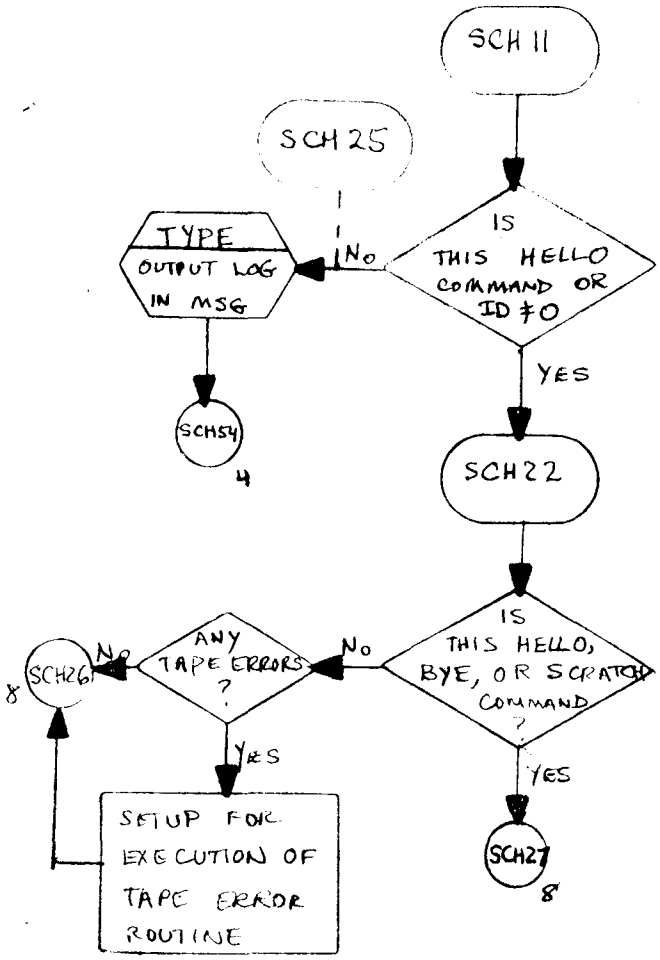

SCH 7
DEQUE REMOVE USER FROM Q
CHANGE STATUS TO ABORTING
TYPE PRINT MESSAGE
HTEST $HELLO TEST
SCH8
SET UP FORCED DISCONNECT
SCH61
9
SCH6
INITIALIZE BUFFER PTR
SCH 24
Scom
COMMAND TYPE
ERROR
COMMAND
EMPTY LINE
SYNTAX
EHERR
7
SCH30
7
SCH9
7

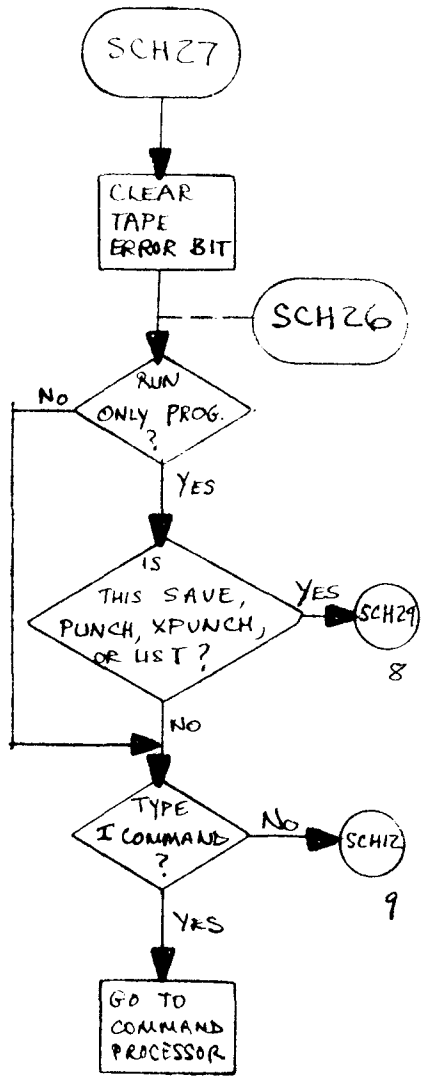

SCH9
SET PRIORITY OF TTY = 0
ID ?
GO LOG IN
No
SCH25
8
YES
SET STATUS TO SYNTAX
GO INSERT IN QUEUE
SCH3
1

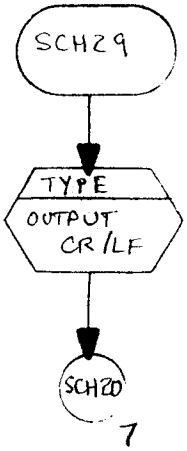

SCH30
TAPE MODE ?
No
SCH20
7
YES
SET PTR TO 1ST CHARACTER OF NEXT BUFFER
CONTINUATION ?
No
SCH54
4
YES
SCH24
6

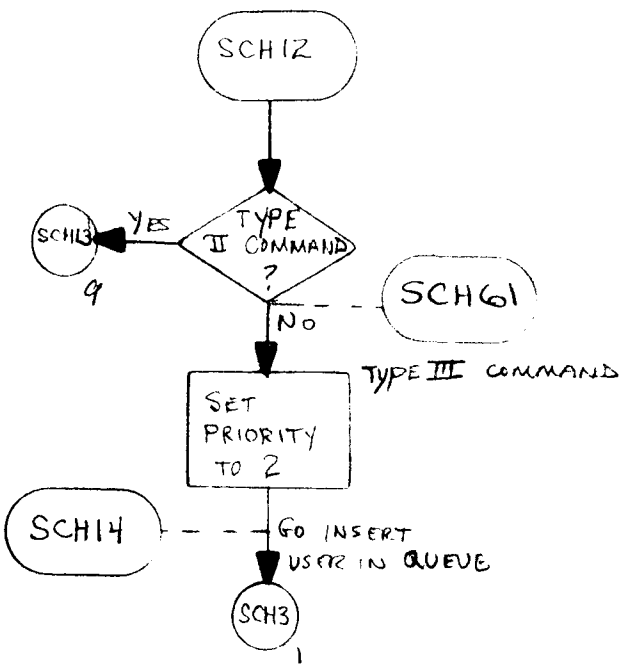

SCH20
POCR
OUTPUT LF
SCH54
4


EHERR
TYPE
OUTPUT ERROR MESSAGE
SCH54
4

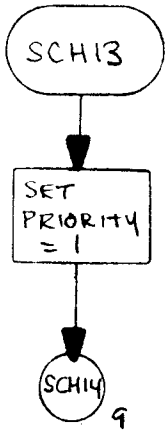

SCH 11
SCH 25
IS THIS HELLO COMMAND OR ID ≠ 0
No
TYPE OUTPUT LOG IN MSG
SCH54
4
YES
SCH 22
IS THIS HELLO, BYE, OR SCRATCH COMMAND ?
No
ANY TAPE ERRORS ?
No
SCH26
8
YES
SETUP FOR EXECUTION OF TAPE ERROR ROUTINE
YES
SCH27
8

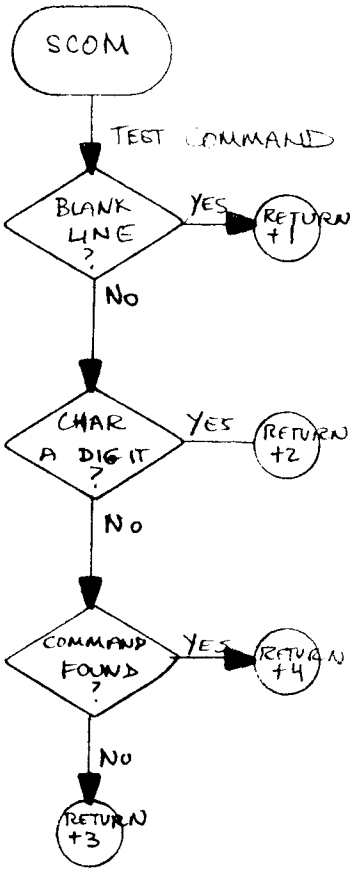

SCH 27
CLEAR TAPE ERROR BIT
SCH 26
RUN ONLY PROG. ?
No
YES
IS THIS SAVE, PUNCH, XPUNCH, OR LIST ?
YES
SCH29
8
NO
TYPE I COMMAND ?
No
SCH12
9
YES
GO TO COMMAND PROCESSOR

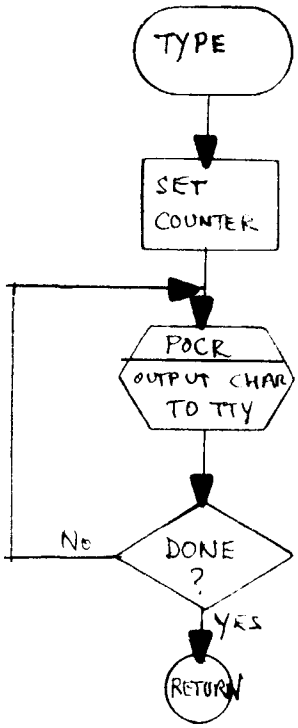

SCH 29
TYPE OUTPUT CR/LF
SCH20
7

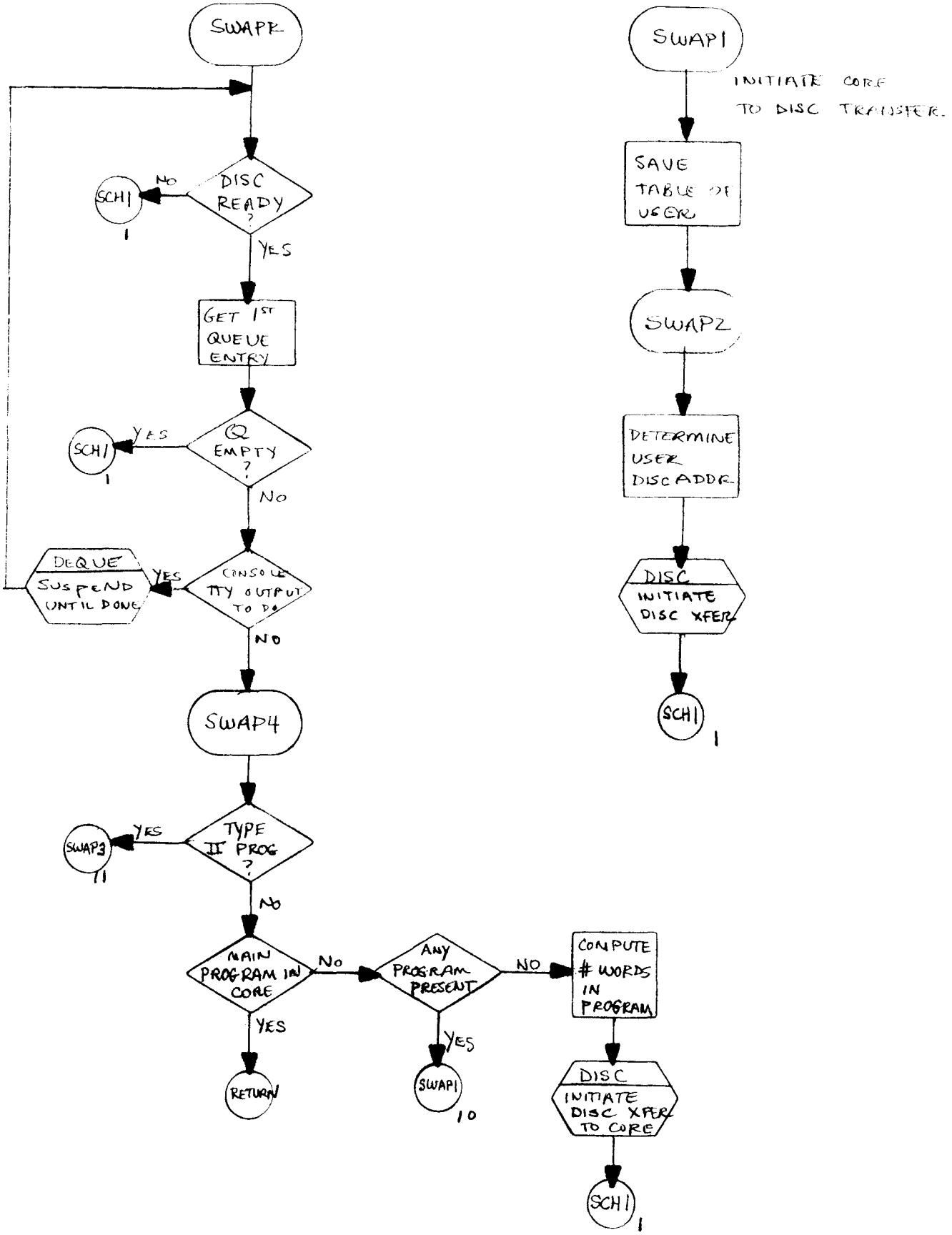

SCH12
TYPE II COMMAND ?
YES
SCH13
9
No
SCH61
TYPE III COMMAND
SET PRIORITY TO 2
SCH14
GO INSERT USER IN QUEUE
SCH3
1

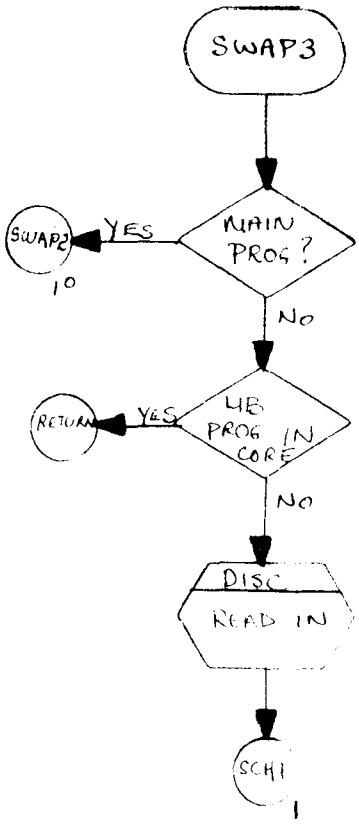

SCR
SCRAT
SCRATCH PROGRAM
SCH20
7

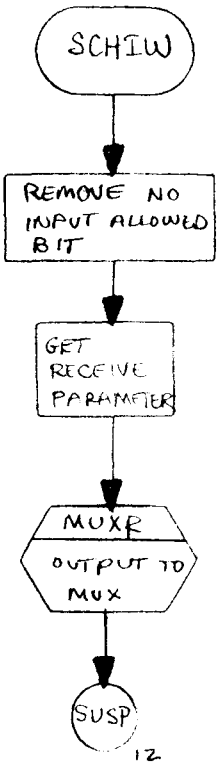

SCH13
SET PRIORITY = 1
SCH14
9

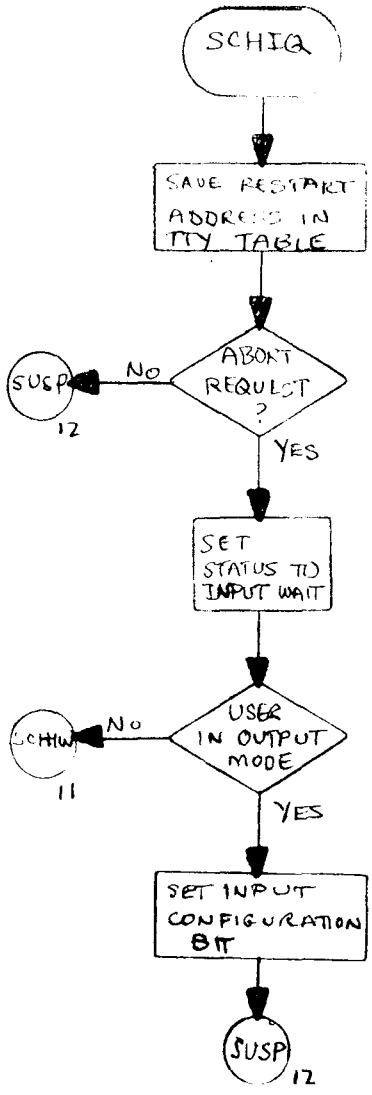

SCOM
TEST COMMAND
BLANK LINE ?
YES
RETURN +1
No
CHAR A DIGIT ?
YES
RETURN +2
No
COMMAND FOUND ?
YES
RETURN +4
No
RETURN +3

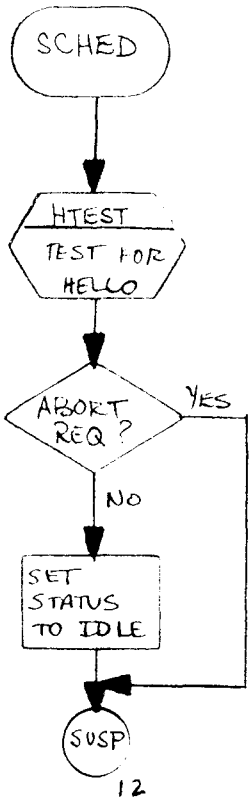

TYPE
SET COUNTER
POCR
OUTPUT CHAR TO TTY
No
DONE ?
YES
RETURN

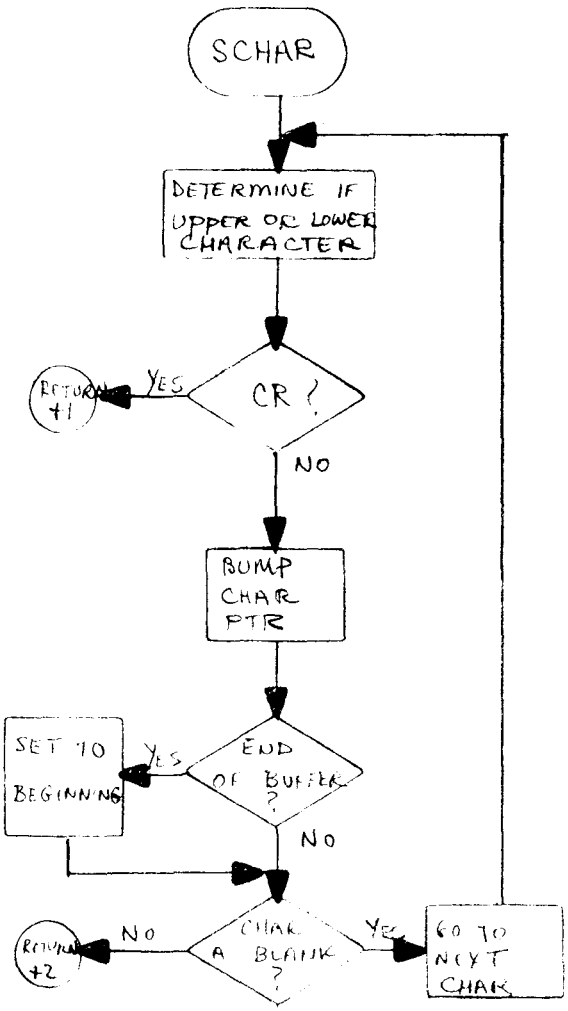

SWAPR
DISC READY ?
NO
SCH1
1
YES
GET 1ST QUEUE ENTRY
Q EMPTY ?
YES
SCH1
1
NO
CONSOLE TTY OUTPUT TO DO
YES
DEQUE SUSPEND UNTIL DONE
NO
SWAP4
TYPE II PROG ?
YES
SWAP3
11
NO
MAIN PROGRAM IN CORE
NO
ANY PROGRAM PRESENT
NO
YES
RETURN
YES
SWAP1
10
COMPUTE # WORDS IN PROGRAM
DISC INITIATE DISC XFER TO CORE
SCH1
1
SWAP1
INITIATE CORE TO DISC TRANSFER.
SAVE TABLE OF USER
SWAP2
DETERMINE USER DISC ADDR.
DISC INITIATE DISC XFER
SCH1
1

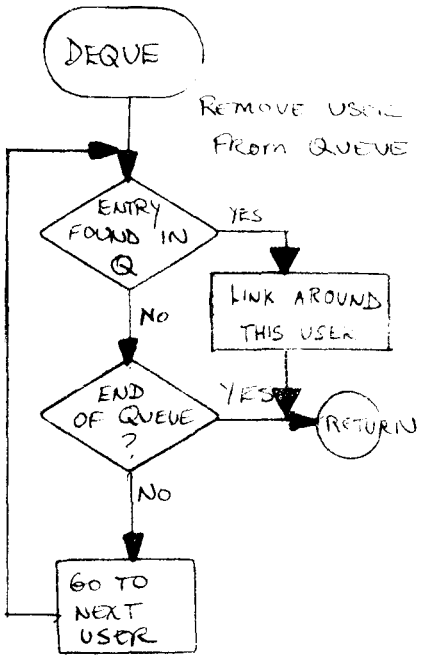

SWAP3
MAIN PROG ?
YES
SWAP2
10
No
HB PROG IN CORE
YES
RETURN
No
DISC
READ IN
SCH1
1

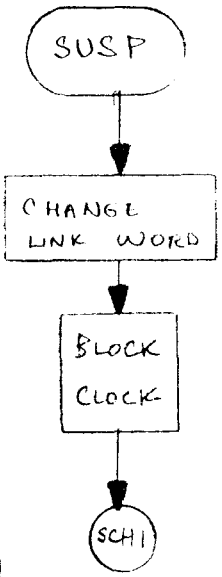

SCHIW
REMOVE NO INPUT ALLOWED BIT
GET RECEIVE PARAMETER
MUXR
OUTPUT TO MUX
SUSP
12

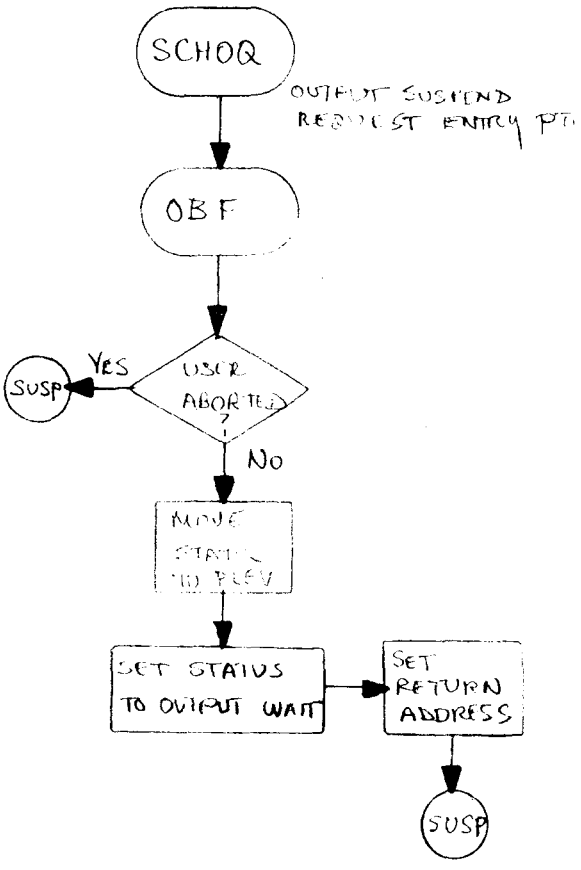

SCHIQ
SAVE RESTART ADDRESS IN TTY TABLE
ABORT REQUEST ?
No
SUSP
12
YES
SET STATUS TO INPUT WAIT
USER IN OUTPUT MODE
No
SCHIW
11
YES
SET INPUT CONFIGURATION BIT
SUSP
12

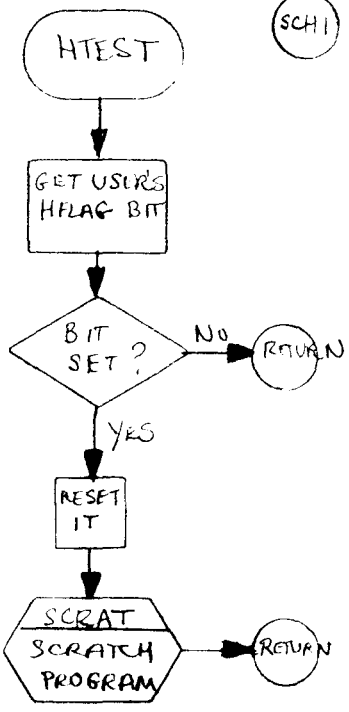

SCHED
HTEST
TEST FOR HELLO
ABORT REQ ?
YES
No
SET STATUS TO IDLE
SUSP
12


SCHAR
DETERMINE IF UPPER OR LOWER CHARACTER
CR ?
YES
RETURN +1
NO
BUMP CHAR PTR
END OF BUFFER ?
YES
SET TO BEGINNING
NO
CHAR A BLANK ?
NO
RETURN +2
YES
GO TO NEXT CHAR

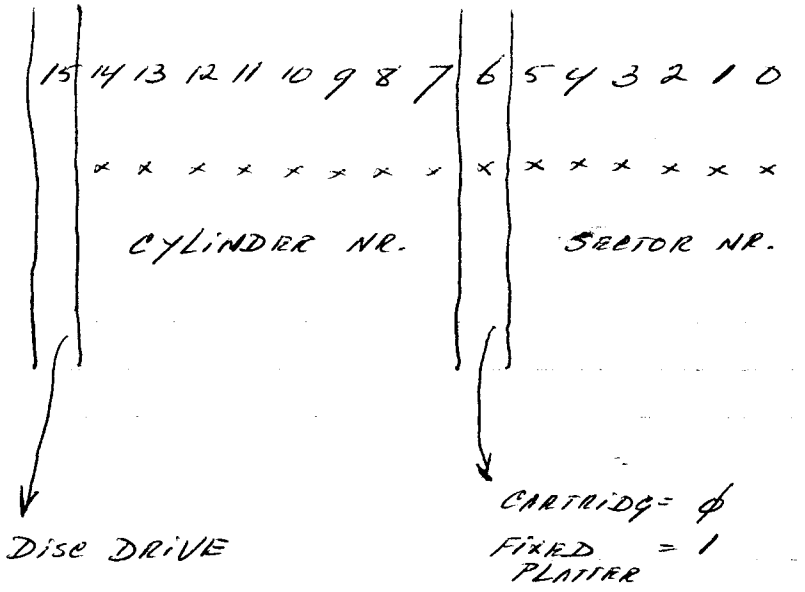

DEQUE
REMOVE USER FROM QUEUE
ENTRY FOUND IN Q
YES
LINK AROUND THIS USER
NO
END OF QUEUE ?
YES
RETURN
NO
GO TO NEXT USER

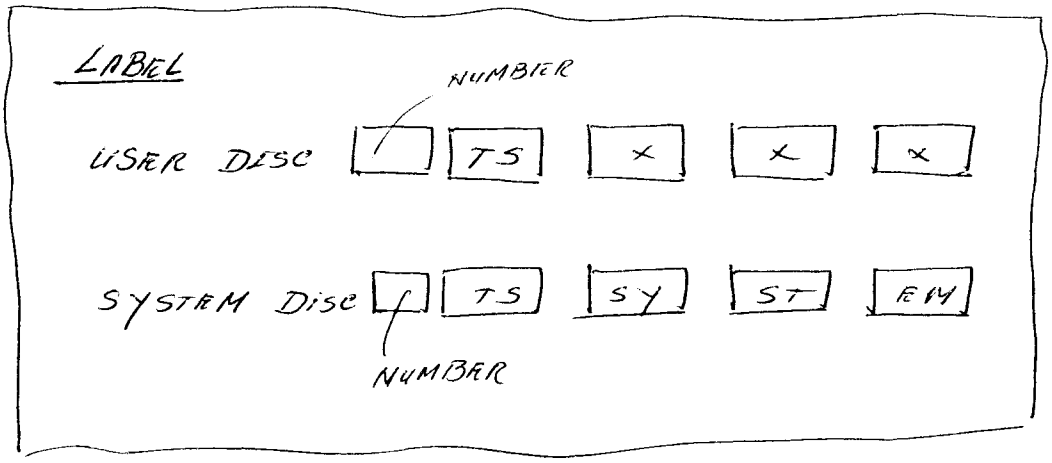

SUSP
CHANGE LINK WORD
BLOCK CLOCK
SCH1


SCHOQ
OUTPUT SUSPEND REQUEST ENTRY PT
OBF
USER ABORTED ?
YES
SUSP
NO
MOVE STATUS TO PREV
SET STATUS TO OUTPUT WAIT
SET RETURN ADDRESS
SUSP


HTEST
GET USER'S HFLAG BIT
BIT SET ?
NO
RTURN
YES
RESET IT
SCRAT
SCRATCH PROGRAM
RETURN

2000 E
Disc ADDRESS FORMAT





15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

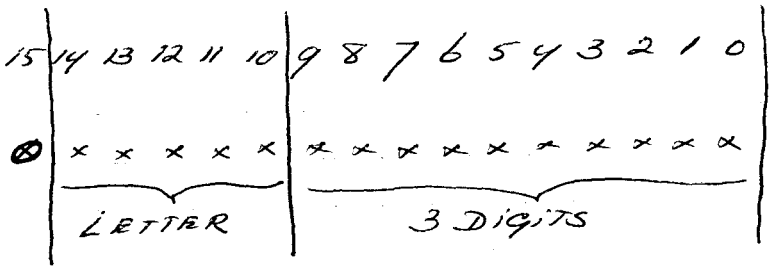
0 x x x x x x x x 1 x x x x x x → DISC # 0

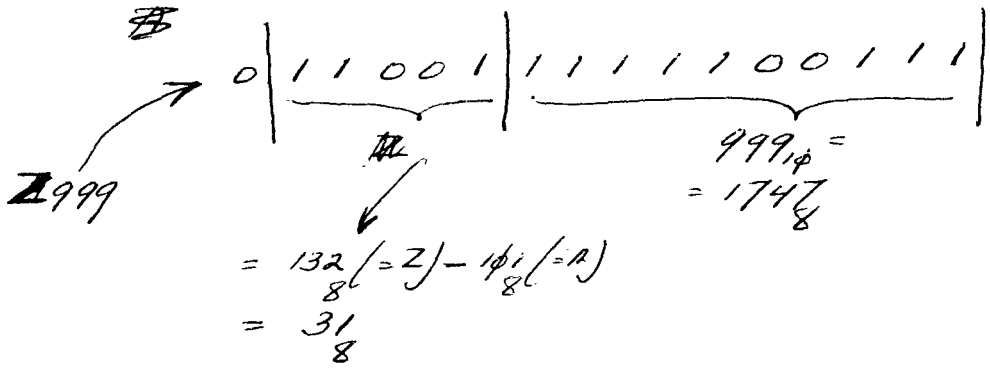
0 x x x x x x x x 0 x x x x x x → DISC # 1

1 x x x x x x x x 1 x x x x x x → DISC # 2

1 x x x x x x x x 0 x x x x x x → DISC # 3

INTERNAL FORMAT
OF ID CODE (IN ID TABLE)





FORMAT OF SYSTEM DISC

TRACK	SECTOR	MODULE	CORE LOCATION		WORD COUNT
ϕ	ϕ	LABEL			
ϕ	1 – $10_8 = 8$	SYSTEM LOADER	$2\phi\phi2_8$	THRU $4\phi\phi1_8$	(1024
ϕ	$14_8 = 12$ – $23_8 = 19$	BASE PAGE	ϕ	THRU 1777_8	(1024
1	ϕ – $57_8 = 47$	LOADER	$4\phi\phi\phi_8$	THRU 17777_8	(6144
2	ϕ – $57_8 = 47$	BASIC SEGMENT #1	$14\phi\phi\phi_8$	THRU 27777_8	(6
3	ϕ – $57_8 = 47$	BASIC SEGMENT #2	$34\phi\phi\phi_8$	THRU 43777_8	(614

WHEN STARTING UP FROM DISC

LOAD INTO CORE:

MODULE	CORE LOCATION	WORD COUNT
BASE PAGE	ϕ THRU 1777_8	1024_{10}
BASIC SEGMENT #1	14000_8 THRU 27777_8	6144_{10}
BASIC SEGMENT #2	30000_8 THRU 43777_8	6144_{10}
LOADER SEGMENT	40000_8 THRU 13277_8	3776_{10}

2000E TIME SHARED BASIC

INTERNAL MAINTENANCE SPECIFICATIONS

NOTE

THE 2000 E CONSIDERS ONE TRACK TO BE HALF A CYLINDER = 48 SECTORS.
HOWEVER TRACKS FOR THE ADT, IDT AND DIRECTORY A CONSIDERED TO BE 24 SECTORS LONG.

REASON:

SYSTEM SHOULD BE ABLE TO READ ONE DIRECTORY TRACK INTO USER AREA WHICH IS 4K WORDS LONG.

ONE DIRECTORY TRACK = 384 × 8 = 3072 WORDS.

↓

= NR. OF ENTRIES

IN "DISC-UP" COMMAND, IF NEW ADT ON SPECIFIED T WILL BE > 3072 WORDS THE ADT OF THE [illegible] IS SHORTENED OF 1.

RICH PEARSON
KILE BAKER

SEPTEMBER 16, 1972

Contents

INTRODUCTION

THE 2000E TIME SHARED BASIC SYSTEM CONSISTS OF THREE SEPARATE PROGRAMS. THE SYSTEM CONTRINS THE BASIC INTERPRETER, EXECUTIVE, AND LIBRARY ROUTINES AND HANDLES THE MULTIPLEXED I/O FROM THE USER TERMINALS. THE LOADER/UTILITY IS RESPONSIBLE FOR GENERATING INITIAL SYSTEMS, BACKING UP THE SYSTEM ON MAG TAPE AND/OR DISCS, AND RELOADDING THE SYSTEM AND USER LIBRARY FROM MAG TAPE. THE BOOTSTRAP LOADER IS RESPONSIBLE FOR LOADING THE SYSTEM AND USE LIBRARY FROM DISC.

HARDWARE CONFIGURATION

I/O CHANNEL	DEVICE
10	TIME BASE GENERATOR
11-12	HP 7900A DISC INTERFACE
13	OPERATOR'S CONSOLE
14	PHOTOREADER
15-16	HP 12920 ASYNCHRONOUS MULTIPLEXER
17	PHONES CONTROL

DIRECTORY

SECTORS 0-23 OF EACH OF 2 TRACKS

The directory is a table which contains all necessary information about each program or file in the system library. It resides on the disc and may occupy from 1 to 2 disc tracks, ~~depending upon how many discs there are on the system.~~ A core resident table called DIREC contains information on the directory itself.

A directory entry consists of 8 words and has the following format:

WORD		
	0	user id
	1	program or
	2	file
	3	name
	4	start of program pointer
	5	date
	6	disc address
	7	-length in words

BIT 15 = 1 if protected, 0 if unprotected.
BIT 15 = 1 if file, 0 if program
BIT 15 = 1 if semi-compiled, 0 if uncompiled.

The directory entries are kept sorted on words 0-3. BIT 15 of words 1 and 2 and 3 are not considered in the sorting. Names of fewer than 6 characters are filled out with spaces (40_8). The date is the most recent date on which the program or file was referred to.

The directory contains 2 pseudo entries which are the first and last entries in the table. They have the following form:

	FIRST ENTRY	LAST ENTRY
0	0	177777
1	0	177777
2	0	177777
3	0	177777
4	0	0
5	177777	177777
6	0	0
7	0	0

A. DIREC

DIREC IS A CORE RESIDENT TABLE WHICH CONTAINS INFORMATION ABOUT THE DISC DIRECTORY. IT HAS THE FOLLOWING STRUCTURE:

WORD 0	- LENGTH IN WORDS OF FIRST DIRECTORY TRACK DISC 0
1-4	SAME AS FIRST 4 WORDS OF FIRST DIRECTORY TRACK DISC 0
5	UNUSED
6	DISC ADDRESS OF FIRST DIRECTORY TRACK DISC 0
7-13	SAME AS 0-6 BUT APPLIED TO 2ND DIRECTORY TRACK DISC 0
14-20	" " " " TO 1ST " " DISC 1
21-27	" " " " TO 2ND " " "
28-34	" " " " TO 1ST " " DISC 2
35-41	" " " " TO 2ND " " DISC 2
42-48	" " " " TO 1ST " " DISC 3
49-55	" " " " TO 2ND " " DISC 3

IN THE RK05 A TRACK IS CONSIDERED TO BE HALF A CYLINDER = 48 SECTORS. THE DIRECTORY HOWEVER USES 1/2 TRACK = 24 SECTORS.

A DISC ADDRESS OF 0 IMPLIES THAT THERE IS NO SUCH DIRECTORY TRACK. WHEN WORD 0 IS 0, WORDS 1-4 ARE MEANINGLESS.

THE DISK ADDRESS OF A DIRECTORY IS ALWAYS SECTOR 0 OF A TRACK. EACH DIRECTORY TRACK CONTAINS 24 SECTORS = 3072 WORDS = 438 DIRECTORY ENTRIES

EACH DISC IN THE SYSTEM HAS TWO DIRECTORY TRACKS. DIRECTORY TRACKS REFER TO THE DISC ON WHICH THEY RESIDE. ALL THE DIRECTORY TRACKS LINKED TOGETHER FORM THE DIRECTORY.

11. ID TABLE

4 ERRORS/REVOLV SECTORS 4-23 OF EACH OF 4 TRACKS

The ID table (IDT) is a disc resident table which contains one 8-word entry for each ID code on the system. The entries are kept sorted according to the ID codes. An entry has the following format:

WORD	0	user id	
	1-3	password	(filled with 0's if fewer than 6 characters).
	4	time allowed	(in minutes)
	5	time used	(in minutes)
	6	disc allowed	(in sectors)
	7	disc used	(in sectors)

Words 4-7 are 16 bit quantities with values between 0 and 65535.

~~The following 2 words in core refer to the IDT:~~

~~IDLOC = disc address of IDT.~~

~~IDLEN = length in words of IDT.~~

THE FOLLOWING TWO FOUR WORD TABLES IN CORE REFER TO THE IDT:

IDTTA - TABLE OF DISC ADDRESSES OF IDT

IDTRL - TABLE OF TRACK LENGTHS OF IDT

THE TRACKS ARE HANDLED IN REVERSE ORDER (i.e. ADDRESS OF TRACK 4 COMES FIRST.)

III. AVAILABLE DISC TABLE

The available disc table (ADT) is a disc resident table which contains one two-word entry for each area of the disc which is unallocated. An entry has the following form:

WORD	0	disc address
	1	length of area in sectors

Entries are sorted according to word 0. Each entry may refer to as much as one full track, and no two consecutive entries ever refer to two adjacent disc areas (two tracks are not considered to be adjacent).

Besides the entries for unallocated areas, there is also one ADT entry for each of the ~~five~~ four tracks on which the system itself resides, and for each of the up to ~~32~~ 16 tracks allocated for user swapping. Word 1 of each of these entries is 0 so that they will never be allocated. The purpose of having these entries is to indicate to the system dump that they may be released at that time, and ~~also to indicate to the LOCK and UNLOCK routines that these tracks have special significance.~~

At the end of the ADT is one additional entry having the form:

0	177777
1	0

Since track 0 is always allocated as a system track, any possible disc address is guaranteed to be bounded by two ADT entries.

The following two memory locations refer to the ADT:

ADLOC = disc address of ADT

ADLEN = -length in words of ADT

~~The IDT and ADT always reside on the same track. The IDT is at the beginning of the track (sector 0), and the ADT begins at the first sector that is unused by the IDT.~~

P3072 165 [illegible] 3073/32 371/32 47/33 [illegible]

SYS# OF USER'S DISC (SUBSYS. 1, 2 ETC)

"ADT" IS ON TRACK 5 SECTOR 2 - 4 & 24

MAXIMUM LENGTH OF ADT = 22 SECTORS =
= 22 × 128 = (2816 WORDS)

2 WORDS/ENTRY → MAX. $\frac{2816}{2}$ = 1408 ENTRIES (INCL DUMMY)

∴ (1407 REAL ENTRIES (WITHOUT DUMMY))

(1408 ENTRIES (WITH DUMMY))

SYSTEM DISC (SUBSYS. 0)

ADT IS ON TRACK 26

MAX. LENGTH OF ADT = 24 SECTORS =
= 24 × 128 = (3072 WORDS)

2 WORDS/ENTRY → MAX. $\frac{3072}{2}$ = 1536 ENTRIES (INCL DUMMY)

1 TRACK = 24 SECTORS =
= 24 × 128 = 3072_{10} WORDS

∴ (1535 REAL ENTRIES (WITHOUT DUMMY))

(1536 ENTRIES (WITH DUMMY))

USER AREA

$1325_8 - 12252_8$

LENGTH = $12252 - 1324 = 10726_8 = 4566_{10}$ WORDS.

4K = 4 × 1024 = 4096 WORDS

IV. FUSS

The FUSS table is a ~~512~~ 128 word table which resides on the disc. Its disc address can be obtained by the instruction.

```
LDA FUSS,1
```

FUSS is divided into ~~32~~ 16 sections of ~~16~~ 8 words each. The ~~16~~ 8 words in each section are the disc addresses AND FILE LENGTHS of the user files currently being accessed by the user corresponding to that table. Addresses of 0 indicate no file. ~~Disc addresses with bit 7=1~~ BIT 15 OF A FILE LENGTH WORD indicates that the user has read only access.

The purpose of maintaining this table is to:

1.) Prevent simultaneous write access by two users to one file;
2.) Prevent KILLing a file when some user has access to it.

A user's FUSS (i.e. his area of the FUSS table) is set by the FILES routine, which is called from BASIC at the beginning of execution of a program containing a FILES statement. It is cleared by BYE, HELLO, KILLID, and sometimes by KILL.

V. COMTABLE

The COMTABLE is a list of all user and system commands containing their ASCII codings and disc locations or core addresses. The structure of the COMTABLE is as follows:

COM1	codes for commands which are executed immediately by the system	
COM2	codes for commands which are executed by BASIC	
COM3	user commands which are executed by disc resident programs	
COM4	system commands - - all are executed by disc resident programs	
COM5	starting addresses for those commands which are listed under COM1 and COM2	
COM6	disc addresses for those commands which are listed under COM3 and COM4	(this section is filled by the loader)

Since each command is recognized only by its first 3 letters, the scanner converts each letter into a number from 0 to 31_8, and then packs the three codes into one word as three 5-bit bytes. In addition, bit 15 is set for system commands. Codes of -1 in sections 2, 3, and 4 do not correspond to any possible 3-letter code. Their purpose is to generate room in COM6 for disc addresses of routines that are called indirectly, or for tables like FUSS. In the case of CTAPR, the purpose is to generate a status type for printing compiler tape errors without a direct command from the user.

VI. LOGGR

LOGGR is a ~~64~~ 32-word queue which contains codes for printing LOGON/OFF messages. Entries are placed on the queue by HELLO,BYE, and SLEEP. Each entry consists of 2 words, with the following format:

WORD Ø: user id (BIT 15=0 for ON, 1 for OFF)

1: bits 15-5 = 60 x hrs + mins

bits 4-0 = terminal number

The representation of a user id is as follows:

BITS 14-10 = letter (A = 1, B = 2, ..., Z = 32_8)

BITS 9-0 = number (0-999)

The following variables are relevant:

LOGCT = # of unprocessed entries in LOGGR

LOGP1 = points to word 1 of last processed entry

LOGP2 = points to word 1 of last unprocessed entry

Note that LOGCT = 0 <=> LOGP1=LOGP2

VIII. EQUIPMENT TABLE

The Equipment Table is the area of core which describes the resources available to the system. It resides at locations 100 - 206

100 - 167 :	DIREC	(discussed elsewhere)
170 :	IDLEN	total length of ID table
171 :	IDTTA	Pointer to ID Track addresses
172 - 175 :	BSS 4	ID TRACK addresses
176 :	IDTRL	Pointer to ID track lengths
177 - 202 :	BSS 4	ID track lengths
203 :	ADLOC	(discussed elsewhere)
204 :	ADLEN	" "
205 :	?TBL	number of sectors/disc track
206 :	NPORT	- number of ports on the system

VII. TELETYPE TABLES

THE TELETYPE TABLES CONTAIN IN CORE INFORMATION FOR SYSTEM USERS. EACH OF THE USERS HAS ONE TABLE CONTAINING THE FOLLOWING ENTRIES:

WORD	0	TNUM
	1	CCNT
	2	BPNT
	3	BSTR
	4	BHED
	5	BGIN
	6	BEND
	7	TSTA
	10	DCNT
	11	CDLY
	12	LDLY
	13	PHON
	14	RPRM
	15	SPRM
	16	PPRM
	17	MASK
	20	DISC
	21	PROG
	22	ID
	23-25	NAME
	26-27	TIME
	30	CLOC
	31	RSTR
	32	STAT

33	LINK
34	PLEV

TNUM: TELETYPE NUMBER IN BITS 12-8

CCNT: USED BY THE MULTIPLEXER FOR COUNTING OUTPUT CHARACTERS. THE COUNT EQUALS -# OF CHARACTERS, INCLUDING THE CURRENT ONE.

BPNT: ON INPUT - POINTS TO THE CHARACTER LOCATION INTO WHICH THE NEXT CHARACTER WILL BE DEPOSITED.
ON OUTPUT - POINTS TO THE LAST CHARACTER TRANSMITTED.

BSTR: ON INPUT - POINTS TO THE FIRST CHARACTER OF THE MOST RECENT BUFFER.
ON OUTPUT - POINTS TO THE LOCATION INTO WHICH THE NEXT CHARACTER WILL BE PLACED BY THE POC ROUTINE.

BHED: ON INPUT - POINTS TO THE NEXT CHARACTER TO BE FETCHED.

BGIN: POINTS TO THE BEGINNING OF THE PHYSICAL BUFFER.

BEND: POINTS TO THE FIRST CHARACTER FOLLOWING THE PHYSICAL BUFFER.

TSTA: TELETYPE STATUS. THIS WORD CONTAINS THE FOLLOWING INFORMATION:

BIT	NAME	MEANING IF = 1
0	IOBT	USER IS IN INPUT MODE
1	NIBT	NO INPUT ALLOWED
2	ICBT	INPUT CONFIGURATION NEEDED
3	RNBT	USER IS RUNNING
4	CXBT	'CONTROL X' WAS HIT
5	XOBT	X-OFF WAS READ FROM TERMINET OR ASR-
6	LTBT	WAIT FOR LOG TIMING
7	LDBT	LINE DROPOUT OCCURRED
8	HUBT	HANG USER UP
9	PDBT	PHONE DISCONNECTED
10	UNABT	USER NOT ABLE TO ABORT
11	ABTRY	USER TRIED TO ABORT
12-14		TELETYPE SUBTYPES

DCNT: 'CR' AND 'LF' DELAY COUNTER.

CDLY: CARRIAGE RETURN DELAY (NEGATIVE).

LDLY: LINE FEED DELAY (NEGATIVE)

PHON: USED AS TIME COUNTER FOR PHONES LOGIC

RPRM: RECEIVE CHANNEL PARAMETER

SPRM: SEND CHANNEL PARAMETER

PPRM: PHONE PARAMETER

MASK: EQUALS 2↑N FOR USER N

DISC: disc address of user's swap area

PROG: when user is on the disc, PROG points to the last core location used by the program. When the user is loaded into core, PROG is placed into PBPTR. When he is written back to disc, PBPTR is copied into PROG. BASIC is required to maintain PBPTR as a bound on the core it is using.

ID: user's id, 0 if none

NAME: a three word entry containing the user's program name. It is set by the routine NAME & GET & CHAIN, and cleared by HELLO. When fewer than 6 characters are in the name, blanks are appended.

CLOC: this is the timeout clock used to determine the length of a user's time slice. See the discussion on scheduling for further information.

RSTR: this is set, when a user is placed on the queue, to his starting address in core. When the user is actually initiated, RSTR is set to 0. Whenever RSTR = 0, the transfer address of the user can be found in location PREG.

STAT: indicates user's status. The user's status is as follows:

-3,	enter timeout
-2,	system disconnect
-1,	user abort request
0,	idle
1,	system abort
2,	input wait
3,	output wait
4,	syntax processing
>4,	command processing

When a command is being processed, STAT indicates the command. STAT values are assigned in order of entries in the COMTABLE, so that

RUN = 5

LIST = 6

PUNCH = 7, etc.

LINK: the LINK words in the tables are used to form a queue of active users. All users whose status is ≥ 4 are in the queue. See discussion on scheduling for further information.

PLEV: this word gives the priority level of the user when he is on the queue. When the user's status is set to 2 or 3, the previous value of STAT is copied into PLEV, and the user removed from the queue. The possible values of PLEV are as follows:

0: highest priority, used for syntax, users returning from I/O suspend, and for disc resident routines once they begin. This includes FILES and CHAIN.

1: used for commands RUN,LIST,PUNCH

2: used for disc resident routines until they reach the top of the queue

4: used for long running programs.

~~RTIM: the length of time in seconds that it took the user to respond to an <ENTER statement>.~~

Associated with each item in these tables is a symbol which is EQUated to the corresponding number of the item. For example:

```
?FLAG  EQU   0
?TNUM  EQU   1
?RTIM  EQU   15
```

These symbols are primarily used for adjusting pointers to the table. For example, if the B register contains a pointer to the LINK entry of some user, the instruction

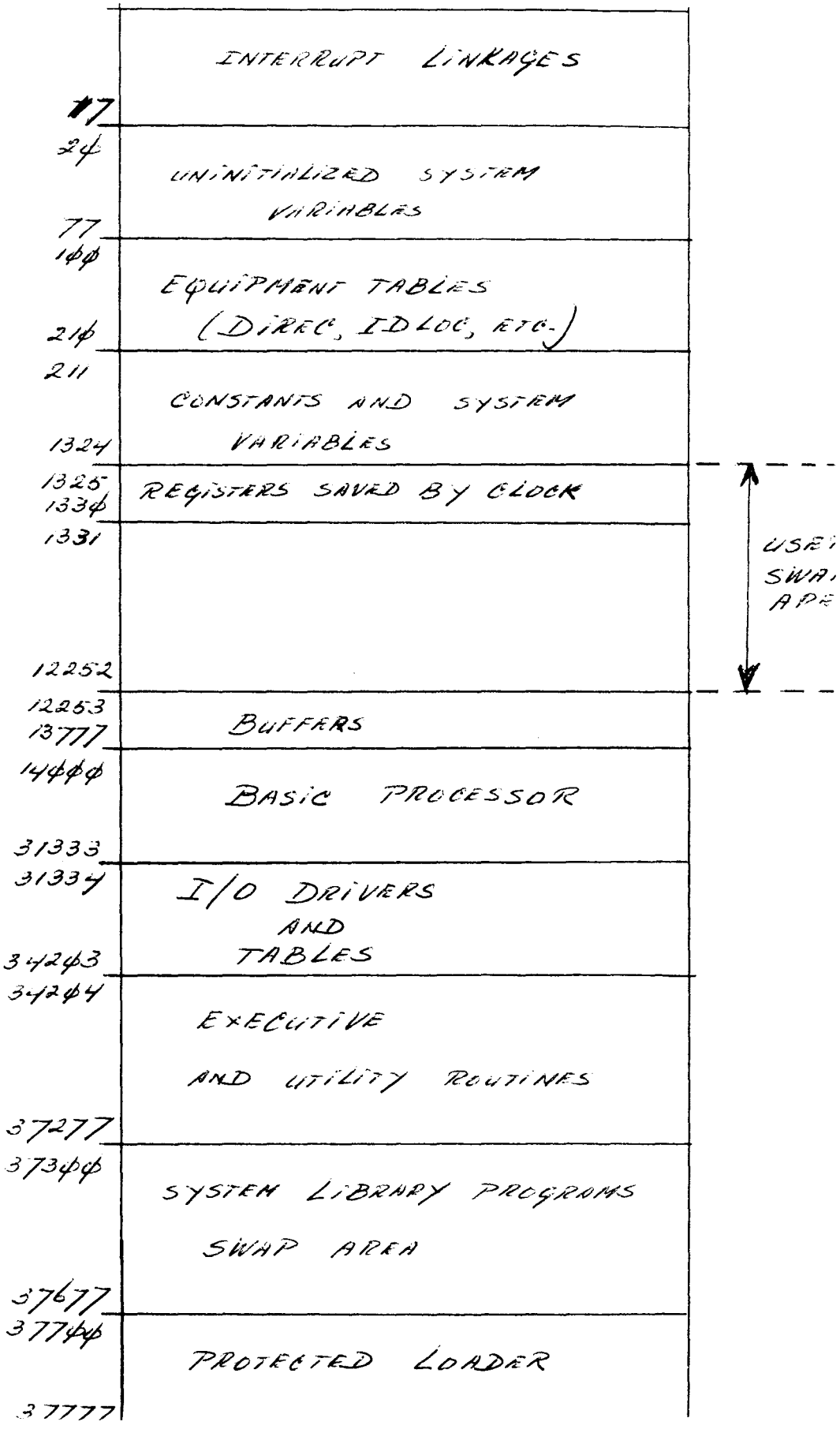
```
ADB   .+?  ID - ? LINK
```

will point B to his ID entry.

. is a symbol located in base page at the 0 entry of a table of constants from -26 to + 49. A word containing the value N, where $-26 \leq N \leq 49$ can be referenced by .+N.

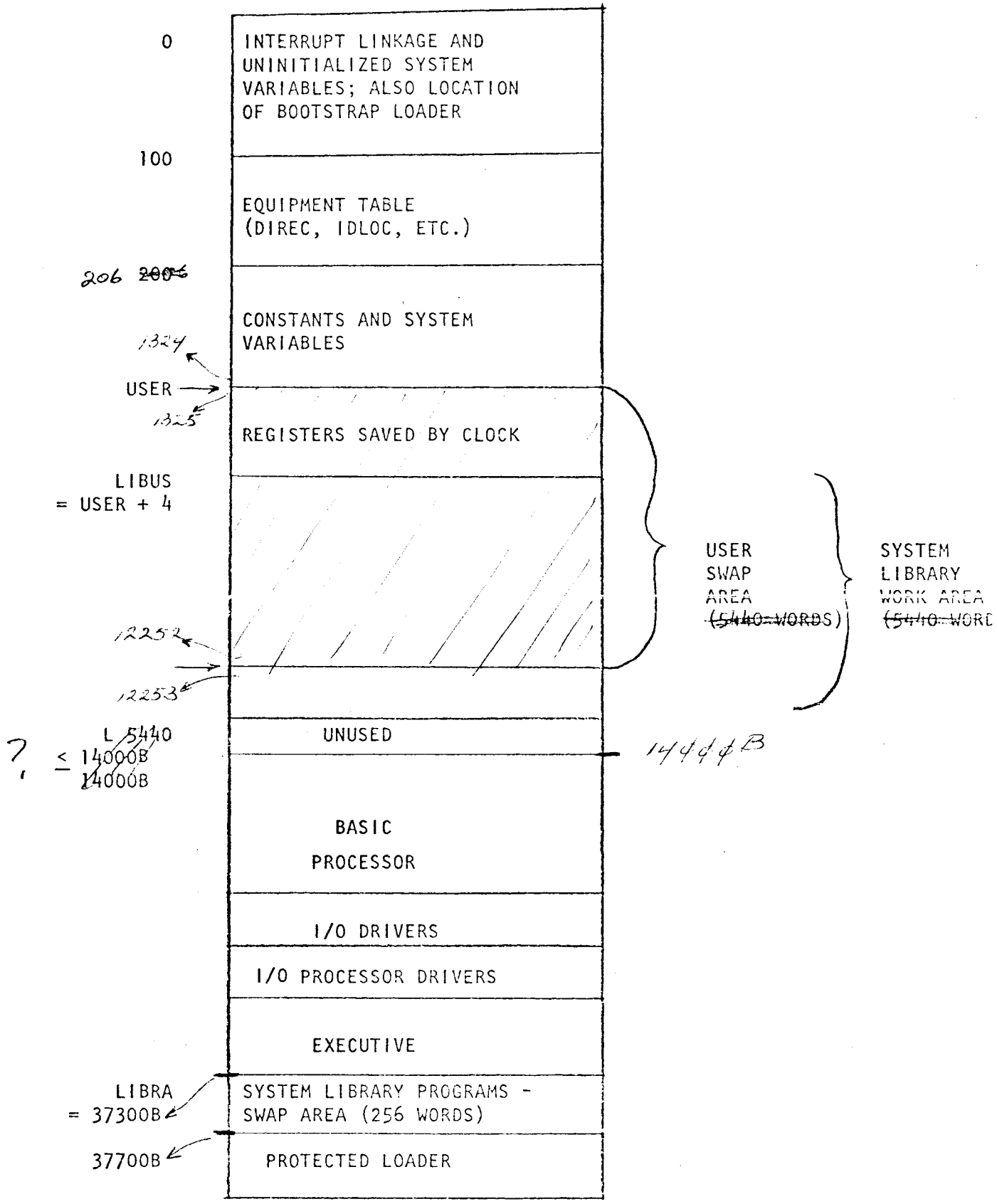
2440 E TSB

CORE MAP



Rich - check these addresses at end of user area

CORE MAP



Disc Organization

The system requires an HP 7900 disc and may have one additional drive, either a 7900 or a 7901 disc drive. Each disc platter is assigned a subchannel number (0-3). Subchannel 0 is always the non-removable platter of the first drive (drive 0). Subchannel 0 is referred to as the system disc and contains the TSB system itself, the loader/utility, IDT tracks, ADT track and two directory tracks plus the 16 swap tracks. The remainder of the disc is available for storage of user programs and files.

SYSTEM DISC

Disc label + temporary storage for loading	(1 track - must be track 0)
Loader/utility	(1 track) → = 48 SECTORS
Resident system	(2 tracks) → = 96 SECTORS
System library routines	(2 tracks) → = 96 SECTORS
IDT	(4 tracks)
ADT	(1 track) = 24 SECTORS
User swap tracks	(16 tracks)
Directory	(2 tracks)

All remaining tracks are available for storage of user programs and files. The ADT contains

an entry for each available area.

The disc addresses of the individual system library routines are stored into the COMTABLE during loading.

USER DISCS

Subchannel 1 and optionally subchannels 2 and 3 are referred to as user discs. Each user disc has a label, a table of lengths (3 words) which gives the length of the ADT track and the two directory tracks), the available disc table (ADT) for the disc, and two directory tracks.

label	(1 sector: must be track 0, sector 0
lengths table	(1 sector: must be track 0, sector 1)
ADT	(##22 sectors: must be track 0, sector 2)
Directory	(2 tracks: must be tracks 1 and 2

The remaining tracks are available for user programs and files.

During running, each user track contains a copy of the area from core location USER through the core location specified by its ?PROG entry. This includes all variable data which is relevant to that user's program, and his program itself. The location of various sections in his program is discussed elsewhere.

Programs and files are each required to be stored as contiguous blocks of disc. Since the disc is allocated by sectors, each program may cause part of its last sector to be wasted. When a program is stored (by the SAVE routine), it is first decompiled and is stored in that form. Only the encoded text is stored, so that a program may require as little as 3 words of disc space. ~~When a program is stored (by the CSAVE routine) it is saved in a semi-compiled form, i.e. the form it is in after the symbol table is built. Both the encoded text and the symbol table are stored, plus 6 words of necessary information.~~

Files always occupy an integral number of sectors (1 - 128), each file occupying a contiguous area on the disc. BASIC does not treat the individual sectors in the same logical sequence as the physical sequence, but rather interleaves the sectors, as follows:

even # of sectors

Physical sequence:	1	2	3	4	...	2n-2	2n-1	2n
Logical sequence:	1	n+1	2	n+2	...	2n-1	n	2n

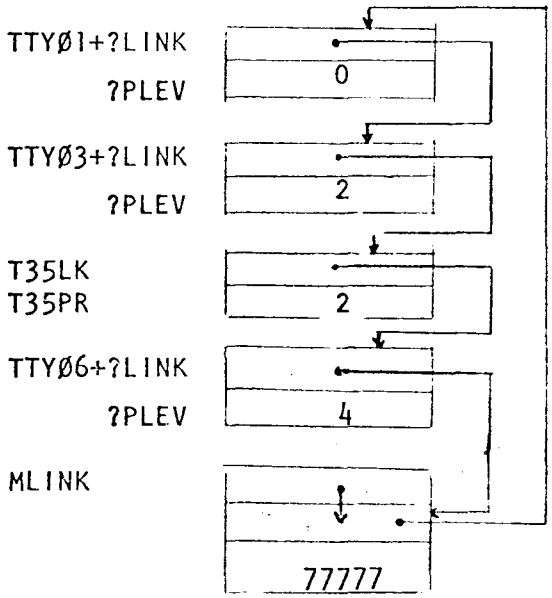
odd # of sectors

Physical sequence:	1	2	3	4	...	2n-2	2n-1
Logical sequence:	1	n+1	2	n+2	...	2n-1	n

This format tends to decrease disc seek time when sectors are accessed in a logically ascending order.

SCHEDULING

The basic philosophy of the TSB scheduling algorithm is to provide short response times for short, interactive jobs at the possible cost of delays in longer running jobs. The implementation of this involves a queue of jobs to run which is ordered according to a priority scheme. The queue is a linked list of from 1 to 34 entries, each entry pointing to the next entry, and the last entry pointing back to the first. The 34 possible entries in the queue are the 32 user LINK entries, a LINK word in a truncated TELETYPE table reserved for the system console, and a queue head. The queue head consists of the locations MLINK (0:2), and is always in the queue. The queue head has a priority of 77777_8, which is stored in location MLINK+2, and so it is always the last entry in the queue. As an example of how this works, assume that users 1, 3 and 6 are on the queue in that order and so is the system console, in a position between users 3 and 6. Then the queue will have the following appearance:



Since the MLINK entry is always the last entry on the queue, MLINK+1 is a pointer to the first entry, which in this case is TTYØ1. In the case of an empty queue, MLINK+1 will point to itself, i.e., CONTENTS(MLINK+1) = CONTENTS(MLINK). Each entry on the queue has a priority no greater in numerical value than that of the one it points to. When an entry is added to the queue, this ordering is always preserved by placing the new entry just ahead of the first entry with a larger priority number. Note that when the first entry in the queue has priority 0, it will remain at the head of the queue until it is removed from the queue entirely.

The following rules are used to assign (and reassign) priorities:

1. Upon first entering the queue, jobs are assigned priorities as follows:
   SYNTAX lines and jobs returning from I/O suspend: 0
   BASIC commands (RUN, LIST, PUNCH) : 1
   Commands for disc-resident routines (GET, BYE, etc): 2

2. Priorities of jobs are reassigned in the following way:
   Jobs of priority 2, when they reach the top of the queue, are reassigned priority 0.

   RUN jobs, when they exceed their time slice, are re-assigned priority 4, and repositioned in the queue according to that priority. Each RUN job is assigned a time slice of two seconds, and if it exhausts that it is assigned another. When executing a <CHAIN statement> or a <FILES statement>, a RUN job is reassigned a priority of Ø.

The scheduler always chooses to run the job on top of the queue, so that whenever a job is running, MLINK + 1 is pointing to its link word. The two locations MAIN and LIB are control variables which tell what is presently in core. MAIN refers to one of the ~~32~~ 16 user programs. It is a pointer to WORD 0 of the TTY table of the user program currentiy in core. If none is in core, MAIN = 0.

LIB points to the location in the COMTABLE of the disc address of the library routine in core. LIB = 0 when none is present.

The following conditions must exist for the scheduler to permit execution:

A) for Syntax and BASIC commands:
MAIN set to point to correct user table

B0 for disc resident commands:
MAIN = 0
LIB set to correct disc resident routines.

The scheduler routine SWAPR is responsible for creating these conditions, and makes its decisions according to the values of MAIN, LIB, and the entry on top of the queue.

COMMUNICATION BETWEEN SYSTEM MODULES

There are six system modules that communicate with each other in various ways: the disc driver, ~~I/O Processor~~ MULTIPLEXER driver, system console driver, scheduler, BASIC, and system library routines (HELLO, BYE, KILLID, etc.).

1. Disc Driver.

Any section of the system may call the disc driver to perform a disc transfer. Three parameters are passed:

A = disc address	(bits	(15:14)	= disc number
	bits	(13:8)	= track number
	bit	7	= 0
	bits	(6:0)	= sector number)
B = core address	(bits	(14:0)	= core address
	bit	15	= 1 for disc input
			0 for disc output)

WORD = -# of words to be transferred (may be 0, in which case no actual transfer is performed).
Called by JSB DISC,1

It is the responsibility of the caller to insure that the disc is not busy when the call takes place. This is no hardship since while BASIC or a system library routine is running, no other module ever initiates disc transfers. As a result, the disc will appear to be busy only if the module itself has initiated the transfer.

Upon initiation of a disc transfer, the variable ENDSK is set to 1, and it is cleared upon completion. A complete transfer can be performed by:

```
JSB DISC,1
LDA ENDSK
SZA
JMP *-2
```

The system never suspends a program for a disc transfer because the high speed of the disc does not cause any great overhead.

The value of WORD is not modified by the driver.

II. 2100A Asynchronous Channel Multiplexer

A. Multiplexer Initialization

Each port of the multiplexer interface must be primed with two parameters before data transmission or data reception can take place. Onced primed, the receive and send parameters will remain in each channel's memory until the power goes down or a "master clear" is executed. The 16 bit parameters have the following functions:

1. Send Channel Parameter

Bits	Function
0-7	The rate at which the data bits will be transmitted.
8-10	The least significant bits of the number of bits, including stop bits, in a character.
11	Not used
12	If set, ASCII parity will be generated.
13	If set, interrupt on completion of transmission of data will be enabled.
14	Must be set
15	Must be set

2. RECEIVE CHANNEL PARAMETER

BITS	FUNCTION
0-7	THE RATE AT WHICH THE DATA BITS WILL BE RECEIVED.
8-10	SAME AS ABOVE
11	NOT USED
12	IF SET, RECEIVED DATA IS ECHOED BACK TO THE TERMINAL.
13	IF SET, INTERRUPT ON RECEPTION OF DATA IS ENABLED.
14	MUST BE EQUAL TO 0
15	MUST BE SET

B. MULTIPLEXER INTERFACE BOARDS

THE MULTIPLEXER CONSISTS OF TWO BOARDS, A DATA BOARD AND A STATUS BOARD. THEY MUST BE LOCATED IN TWO CONSECUTIVE I/O SLOTS; THE DATA BOARD IN THE HIGHER PRIORITY SLOT (LOWER NUMBERED SELECT CODE) AND THE STATUS BOARD IN THE LOWER PRIORITY SLOT (HIGHER NUMBERED SELECT CODE.)

1. DATA BOARD

DATA OUTPUT TO THE SEND CHANNEL MUST BE IN THE FOLLOWING FORMAT:

BITS			FUNCTION
0-10	DATA:	0-6	ASCII CHARACTER
		7	MUST BE Ø SINCE EVEN PARITY IS GENERATED
		8-10	MUST BE ONES
11			MUST BE Ø
12-13			IMMATERIAL
14			MUST BE SET
15			MUST BE Ø

THE ONLY EXCEPTION TO THE ABOVE FORMAT IS THE SYNCHRONIZING CHARACTER. THIS NON-PRINTING CHARACTER IS OUTPUT AT THE BEGINNING OF EVERY TRANSMISSION TO INSURE TELETYPE SYNCHRONIZATION. ADDITIONALLY, IT IS USED TO PROVIDE LINE FEED AND CARRIAGE RETURN DELAYS. THE SYNCHRONIZING WORD IS Ø77577.

DATA RECEIVED FROM A TERMINAL IS IN THE FOLLOWING FOR

BITS	FUNCTION
0-6	DATA BITS
7-9	IMMATERIAL

10-14	UNIT NUMBER
15	NOT USED

2. STATUS BOARD

THE DATA WORD OBTAINED FROM THE STATUS BOARD SUPPLIES THE FOLLOWING INFORMATION:

BITS	FUNCTION
0	IF = 1, THE INTERRUPT CAME FROM THE COMPLETION OF A CHARACTERS TRANSMISSION (SEND CHANNEL.)
	IF = 0, THE INTERRUPT CAME FROM THE COMPLETION OF A CHARACTER RECEPTION (RECEIVE CHANNEL
1	NOT USED
2	IF = 1, A "BREAK" SIGNAL WAS RECEIVED.
3-9	NOT USED
10-14	UNIT NUMBER
15	IF SET, THIS BIT INDICATES THAT A SEEK OPERATION IS TAKING PLACE IN THE CIRCULATING MEMORY OF THE

INTERFACE. NO DATA OR PARAMETERS SHOULD BE OUTPUT UNTIL THIS BIT CLEARS.

C. TRANSMISSION OPERATION

DATA AND SEND AND RECEIVE PARAMETERS ARE OUTPUT BY THE FOLLOWING METHOD:

1. LIA SC CHECK THE SEEKING BIT AND WAIT FOR IT TO CLEAR.

2. OTA SC THE DATA WORD.

3. OTA SC+1 THE UNIT NUMBER.

4. STC SC TO TRANSMIT THE DATA.

5. CLF SC TO ACKNOWLEDGE THE PREVIOUS INTERRUPT.

III. Multiplexer Driver

The multiplexer driver has three main processing sections: Receive, send, and input. A check of the multiplexer status determines which processing section is needed to service the interrupt.

A. Receive Channel Processing

The multiplexer supplies whole characters, each of which are examined on reception and echoed back to the terminal. Certain characters (rubouts, line feeds, feed frames, and X-off) are ignored. 'Control X' signals that the current line is to be deleted. If the character is '←', the buffer pointer is backed up one position. All other characters are appended to the user's buffer.

Upon reception of a carriage return, the system scheduler is notified that the user has entered a complete line and further character input is blocked.

B. Send Channel Processing

If there are characters left in the user's buffer, a test is made to see if there is line feed or carriage return delay pending. If so, a synchronizing character is output and the delay counter is bumped. If not, the us

NEXT CHARACTER IS PLUCKED FROM HIS BUFFER AND SENT TO HIS PORT. IF THE CHARACTER WAS A LINE FEED OR A CARRIAGE RETURN, THE APPROPRIATE DELAY IS SET UP. IF EXACTLY TEN CHARACTERS REMAIN IN THE USER'S BUFFER AND IF HIS STATUS IS OUTPUT WAIT, THE SCHEDULAR IS NOTIFIED THAT HIS BUFFER IS ALMOST EMPTY.

IF NO CHARACTERS REMAIN IN THE USER'S BUFFER: HE IS PLACED IN AN IDLE MODE IF HIS PROGRAM IS STILL RUNNING; OR HE IS PLACED IN INPUT MODE.

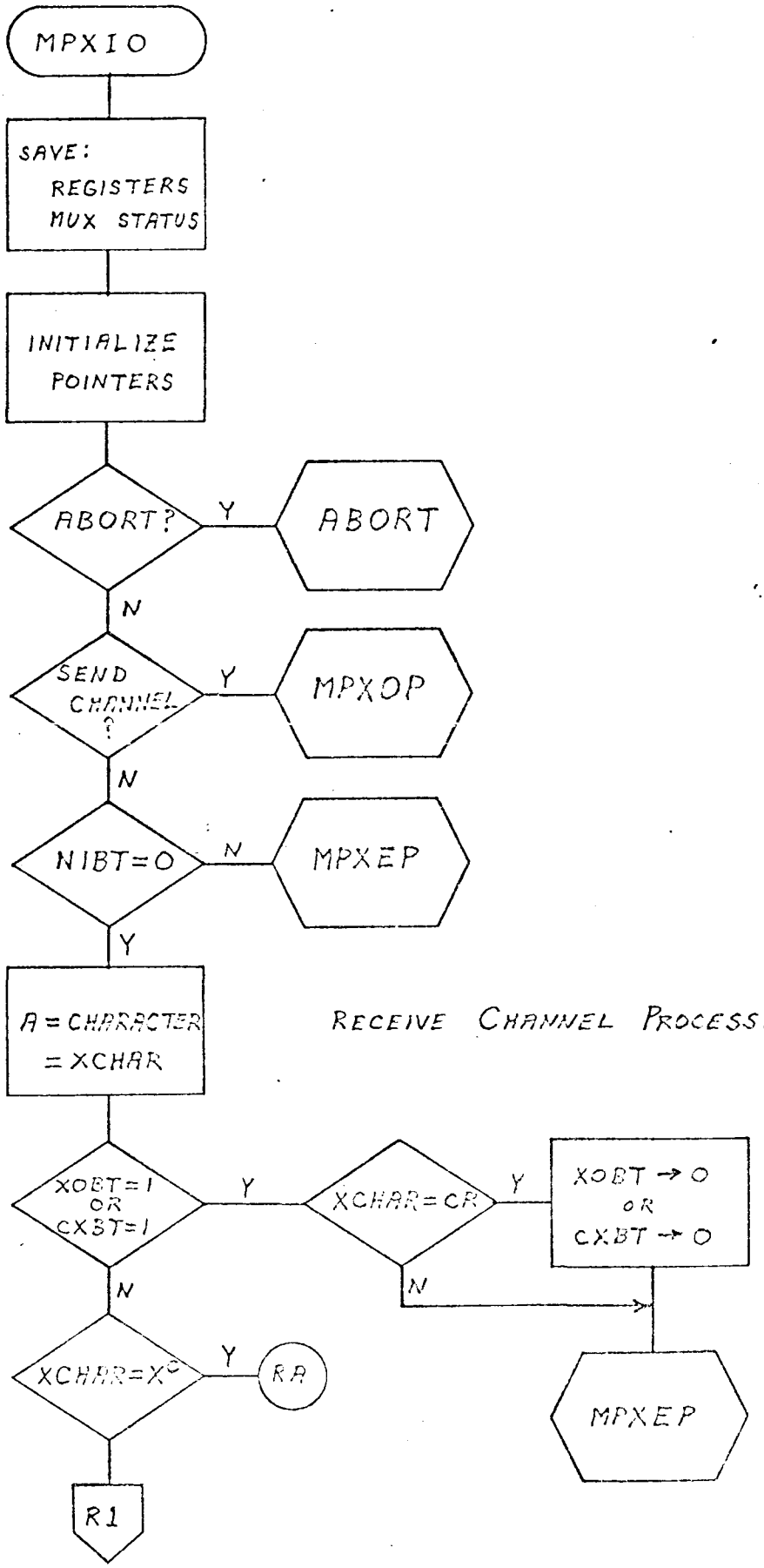
C. ABORT PROCESSING

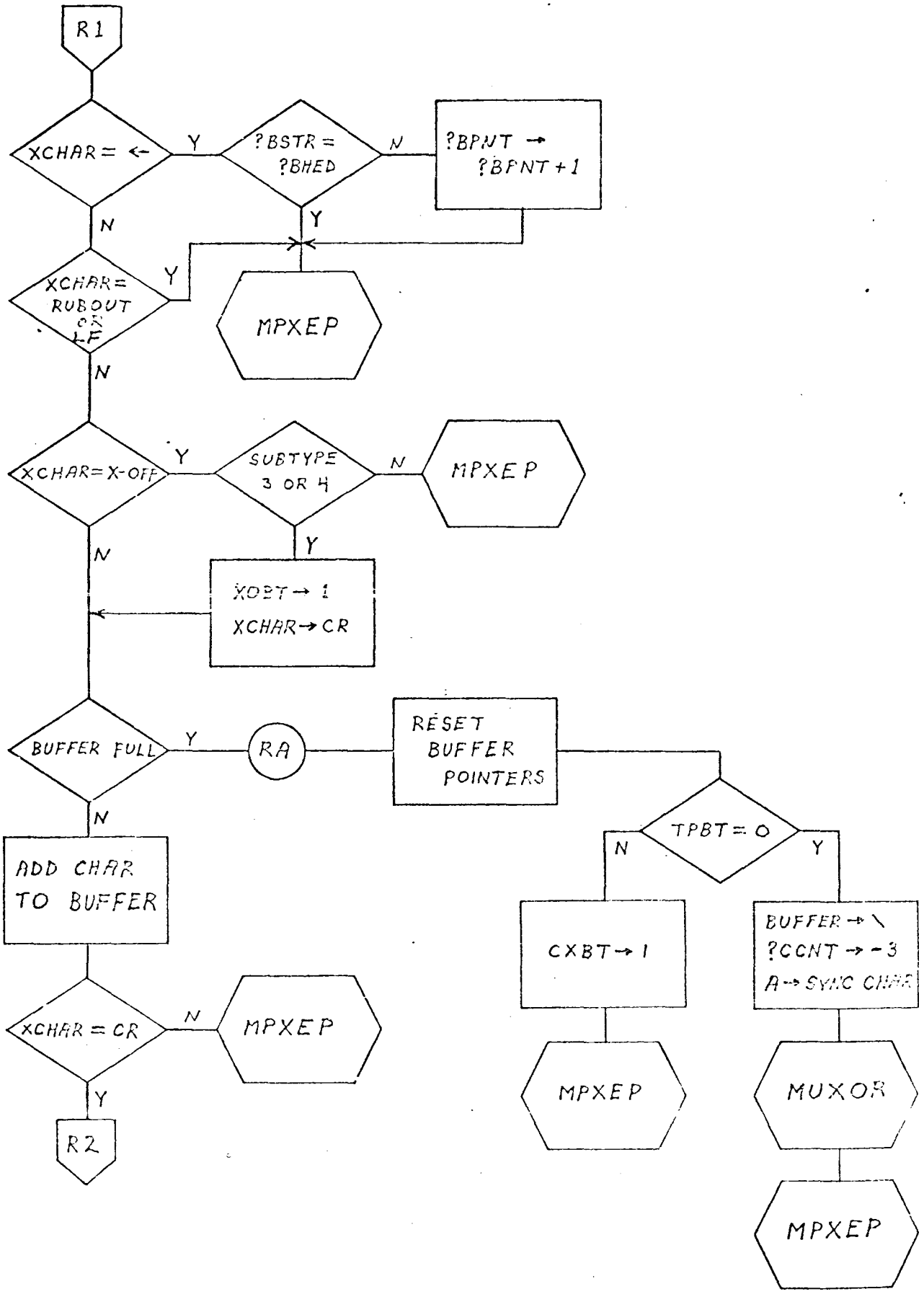
UNLESS THE ABORT SIGNAL OCCURRED ON THE RECEIVE CHANNEL WITH THE USER IN OUTPUT MODE, IT IS IGNORED. IF THE USER IS RUNNING A LIBRARY PROGRAM (EXCEPT CATALOG OR LIBRARY), THE ABORT IS IGNORED, SINCE THE ROUTINE MAY BE IN THE PROCESS OF UPDATING SYSTEM TABLES. AT OTHER TIMES WHEN ABORTING COULD CAUSE TROUBLE, THE UNABT BIT IN THE !TSTA WORD OF THE TTY TABLE IS SET. WHEN THE ABORT IS SEEN, THE ABTRY BIT IS SET. ROUTINES WHICH SET UNABT HAVE THE RESPONSIBILITY OF CALLING ABCHK WHEN ABORTS NO LONGER CAUSE HARM. ABCHK ABORTS THE USER IF ABTRY WAS SET.

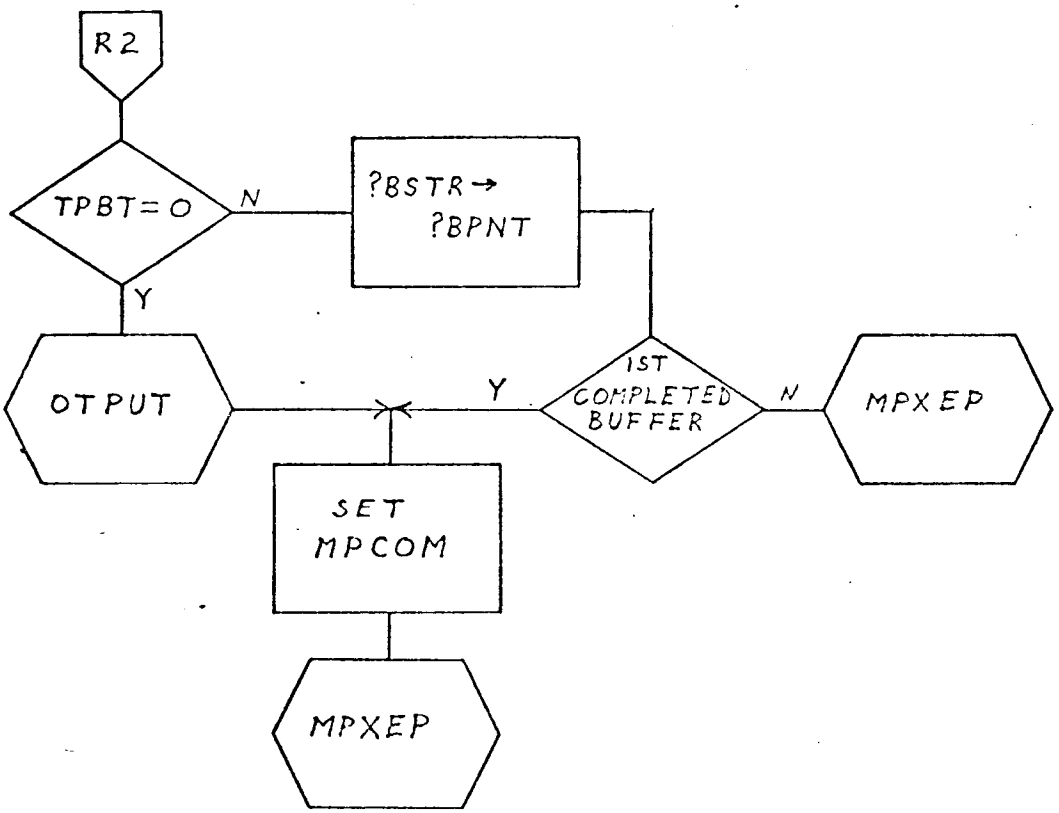
D. PROCESS OUTPUT CHARACTER ROUTINE

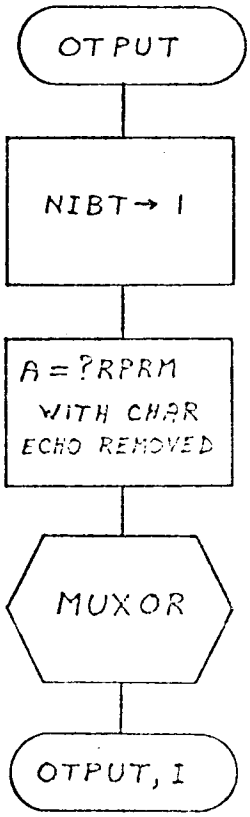
THE POC ROUTINE PLACES CHARACTERS INTO THE USER'S BUFFER UNTIL IT IS FILLED (98 CHARACTERS) AT WHICH POINT THE USER IS SUSPENDED BY POC. THIS IS NO PROBLEM FOR BASIC, BUT DUE TO RE-ENTRANCY PROBLEMS THIS MUST NOT BE ALLOWED BY OTHER MODULES. THE BUFFER IS ALWAYS EMPTY WHEN A LIBRARY ROUTINE IS INITIATED, SO THEY NORMALLY DO NOT HAVE TO WORRY ABOUT IT. CATALOG AND LIBRARY, WHICH MAY FILL THE BUFFER, GET AROUND THE PROBLEM BY SUSPENDING THEMSELVES AT AN APPROPRIATE TIME.

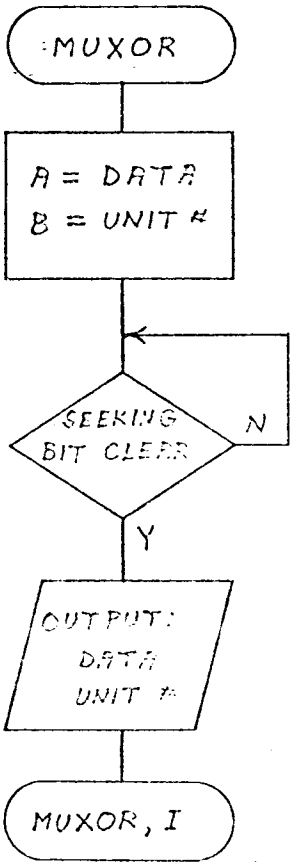
INITIALIZATION SECTION




R1
XCHAR = ←
Y
?BSTR = ?BHED
N
?BPNT → ?BPNT + 1
Y
N
XCHAR = RUBOUT OR LF
Y
MPXEP
N
XCHAR = X-OFF
Y
SUBTYPE 3 OR 4
N
MPXEP
Y
N
XOBT → 1
XCHAR → CR
BUFFER FULL
Y
RA
RESET BUFFER POINTERS
N
ADD CHAR TO BUFFER
TPBT = 0
N
Y
CXBT → 1
BUFFER → \
?CCNT → -3
A → SYNC CHAR
XCHAR = CR
N
MPXEP
MPXEP
MUXOR
Y
R2
MPXEP

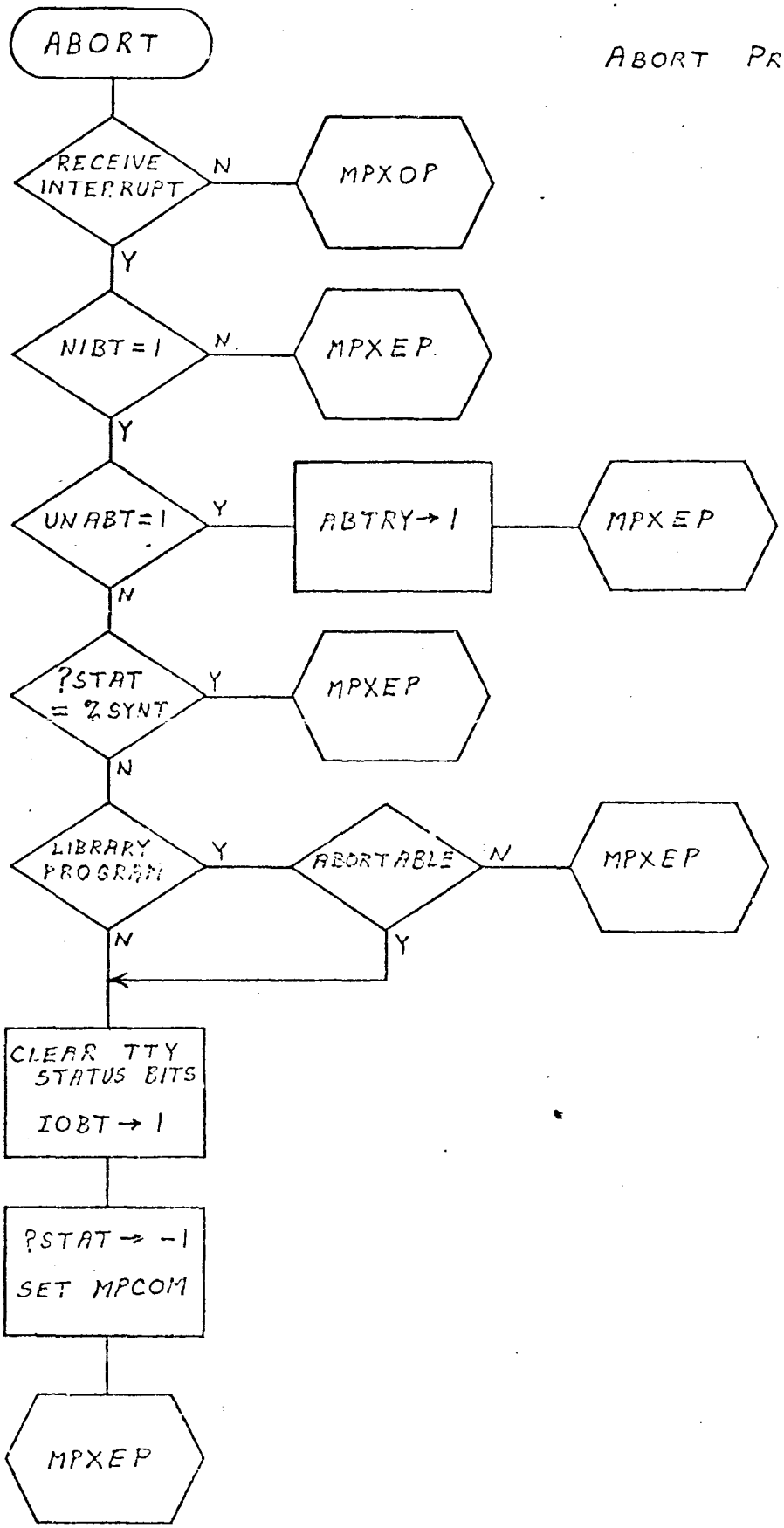




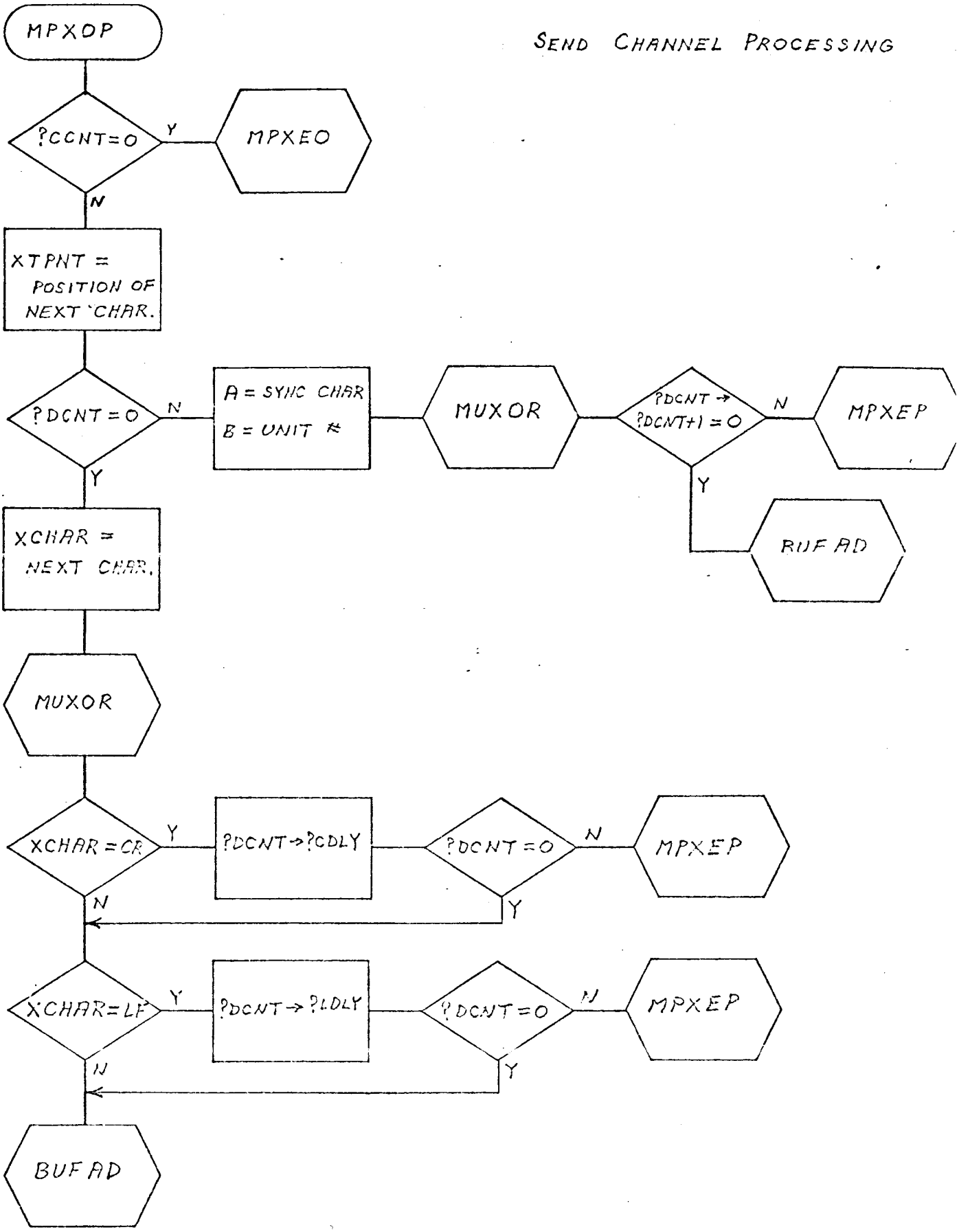


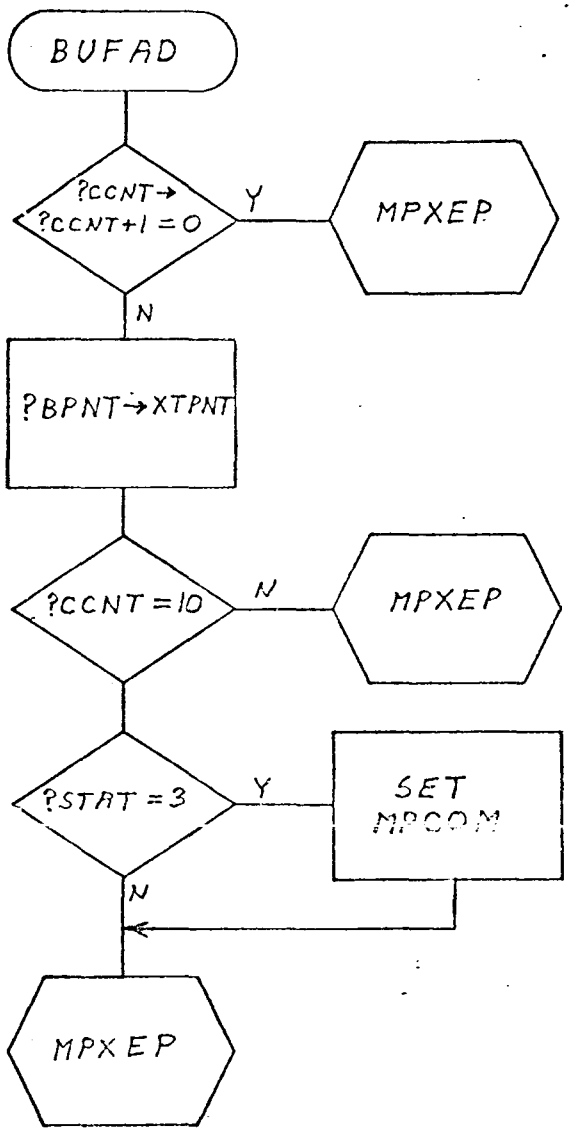
MULTIPEXER
OUTPUT
ROUTINE

ABORT PROCESSING

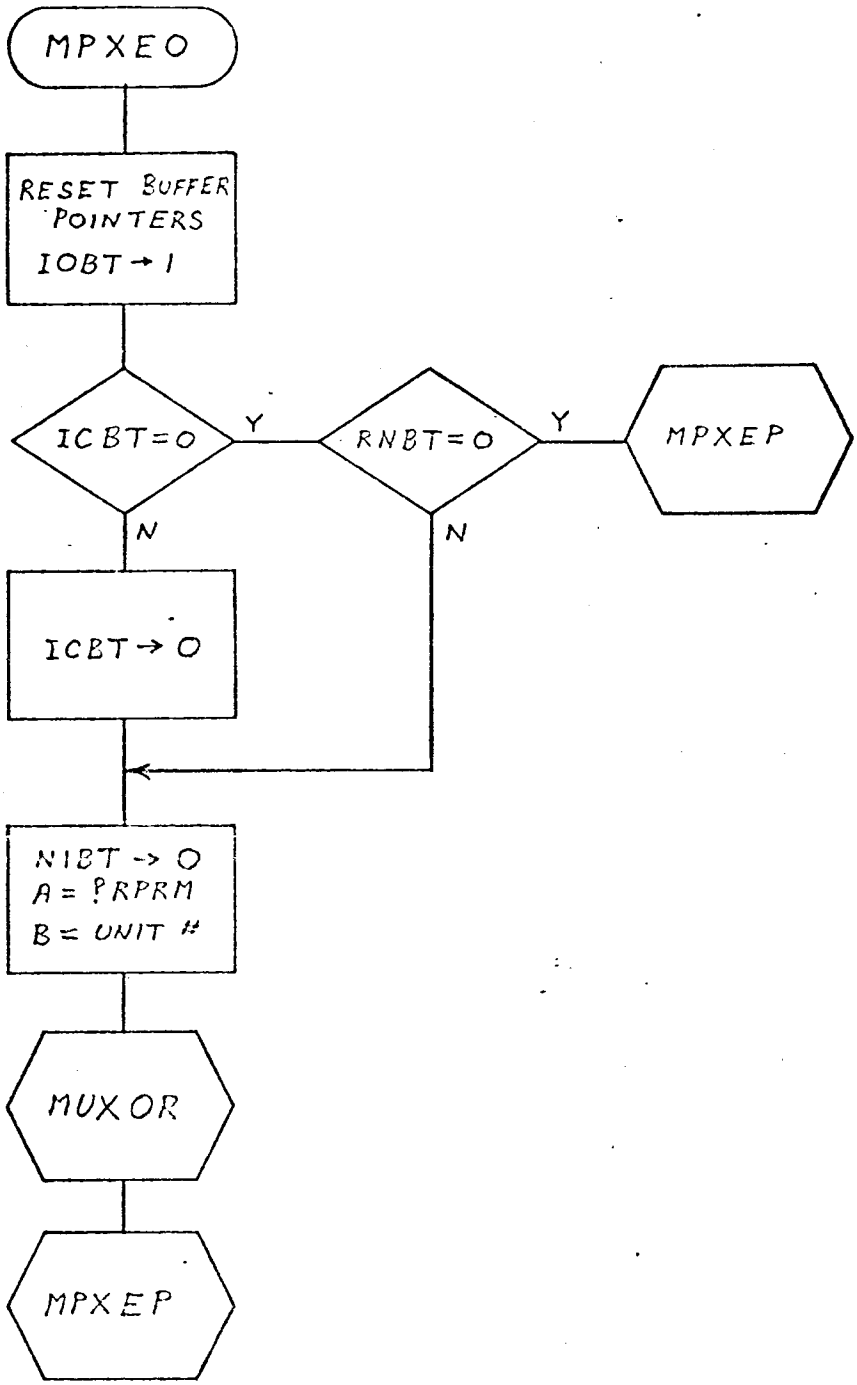


Send Channel Processing




BUFAD
?CCNT→
?CCNT+1 = 0
Y
MPXEP
N
?BPNT→XTPNT
?CCNT = 10
N
MPXEP
?STAT = 3
Y
SET
MPCOM
N
MPXEP


MPXEP
RESTORE
REGISTERS
ENABLE
INTERRUPTS
MPXIO, I


MPXEO
RESET BUFFER POINTERS
IOBT → 1
ICBT=0
Y
RNBT=0
Y
MPXEP
N
N
ICBT → 0
NIBT → 0
A = ?RPRM
B = UNIT #
MUXOR
MPXEP

IV. 2100 Data Set Control Board

The data set control board is used in the scan mode so that an interrupt will only occur if a change in either of the two signals (carrier or data set ready) has been detected. To prime the board for an interrupt, it is necessary to output a parameter with the following format:

Bits	Function
0	Data Set Ready If 0, an interrupt will occur when "data set ready" comes up. If 1, an interrupt will occur when "data set ready" drops
1	Carrier Detect If 0, an interrupt will occur when the carrier comes up. If 1, an interrupt will occur when the carrier drops.
2	Enable bit for comparison logic If 1 and if the comparison logic detects a change in "data set ready", the flag will be set and scanning is stopped.

BITS	FUNCTION
3	ENABLE BIT FOR COMPARISON LOGIC SAME AS BIT 2 BUT APPLIES TO CARRIE DETECT.
4	1 – "DATA TERMINAL READY" ON 0 – "DATA TERMINAL READY" OFF
5	MUST BE SET
6	ENABLE BIT FOR "DATA TERMINAL READY" IF THIS BIT AND BIT 14 ARE SET, "DATA TERMINAL READY" WILL BE SENT TO THE INTERFACE.
7	MUST BE SET
8-9	NOT USED
10-13	CHANNEL NUMBER
14	IF SET, BITS 0-3 WILL BE SENT TO THE INTERFAC
15	IF SET, OPERATION IS IN THE SCAN MODE.

STATUS OBTAINED AFTER AN INTERRUPT HAS THE FOLLOWING MEANING:

BITS	FUNCTION
0	IF 0, "DATA SET READY" HAS DROPPED IF 1, "DATA SET READY" HAS COME UP
1	IF 0, CARRIER HAS DROPPED IF 1, CARRIER HAS COME UP
2-3	HAVE THE SAME VALUES AS THE PARAMETERS OUTPUT TO THE INTERFACE
4-7	0
8-9	NOT USED
10-13	CHANNEL NUMBER ON WHICH THE INTERRUPT OCCURRED
14-15	1

III. System Console Driver

The system console driver maintains two flags, T35F1 and T35F2, which determine its status. The meanings of these flags are as follows:

REPLACE WITH (A)

T35F1: = -1 during output, 0 otherwise

T35F2: Normally 0, it is set to -1 by the driver at the conclusion of input, and cleared to 0 externally. The combined values of these is more significant:

F1	F2	
0	0	driver is accepting input
0	-1	1) input command received and being processed, or 2) output terminated from a system command which is to be reinitiated
-1	0	outputting
-1	-1	outputting, at the end of which the current system command will be reinitiated.

When F2 = -1, the driver will not accept any input. This guarantees system library programs that they will not be interfered with. These routines are responsible for clearing F2 when they call the driver for the last time. F2 and the console status (T35ST) are also cleared if a key is struck on the console during output. This will effectively terminate such things as DIRectories, REPorts and STAtuses.

INSERT (B)

The calling sequence is:

A: bit 15 = 0 if CRLF is to be appended, bits (14:0) = # of chars.

B: bit 15 = 1 if punching is to take place in addition to printing, bits (14:0) = core address of output buffer.

```
JSB TTY35,I
```

The driver uses the 36 word buffer T35BF as an input buffer. Most of the library routines use it for output, and occasionally for temporary storage between lines of output.

(A)

V. SYSTEM CONSOLE DRIVER

THE SYSTEM CONSOLE DRIVER MAINTAINS THREE FLAGS T35F1, T35F2, AND T35F3, WHICH DETERMINE ITS STAT THE MEANING OF THESE FLAGS ARE AS FOLLOWS:

T35F1: = -1 DURING OUTPUT, Ø OTHERWISE

T35F2: NORMALLY Ø, IT IS SET TO -1 BY THE DRIVER AT THE CONCLUSION OF INPUT, AND CLEARED TO Ø EXTERNALLY.

T35F3: NORMALLY Ø, IT IS SET TO -1 BY THE DRIVER AT THE CONCLUSION OF INPUT, AND CLEARED TO Ø BY THE DRIVER AFTER OUTPUT HAS BEEN INITIATED.

THE COMBINED VALUES OF THESE FLAGS ARE MORE SIGNIFICA

F1	F2	F3	
Ø	Ø	Ø	DRIVER IS ACCEPTING INPUT
Ø	-1	-1	INPUT COMMAND RECEIVED AND IS BEING PROCESSED, BUT OUTPUT HAS NOT BEEN INITIATED.
Ø	-1	Ø	OUTPUT TERMINATED FROM A SYSTEM COMMAND WHICH IS TO BE REINITIATED
-1	Ø	Ø	OUTPUTTING
-1	-1	Ø	OUTPUTTING, AT THE END OF WHICH THE CURRENT SYSTEM COMMAND WILL BE REINITIATED.

(B)

WHEN F3 = -1, LOG-ON AND LOG-OFF REPORTS AS WELL AS THE MESSAGE QUEUE ARE HELD OFF. THIS GUARANTEES THAT THESE MESSAGES WILL NOT BE INTERFERED WITH BY SYSTEM LIBRARY PROGRAM OUTPUT.

VI Input and Termination Requests

BASIC may obtain input from a user console by performing the instruction

```
JSB SCHIN,1
```

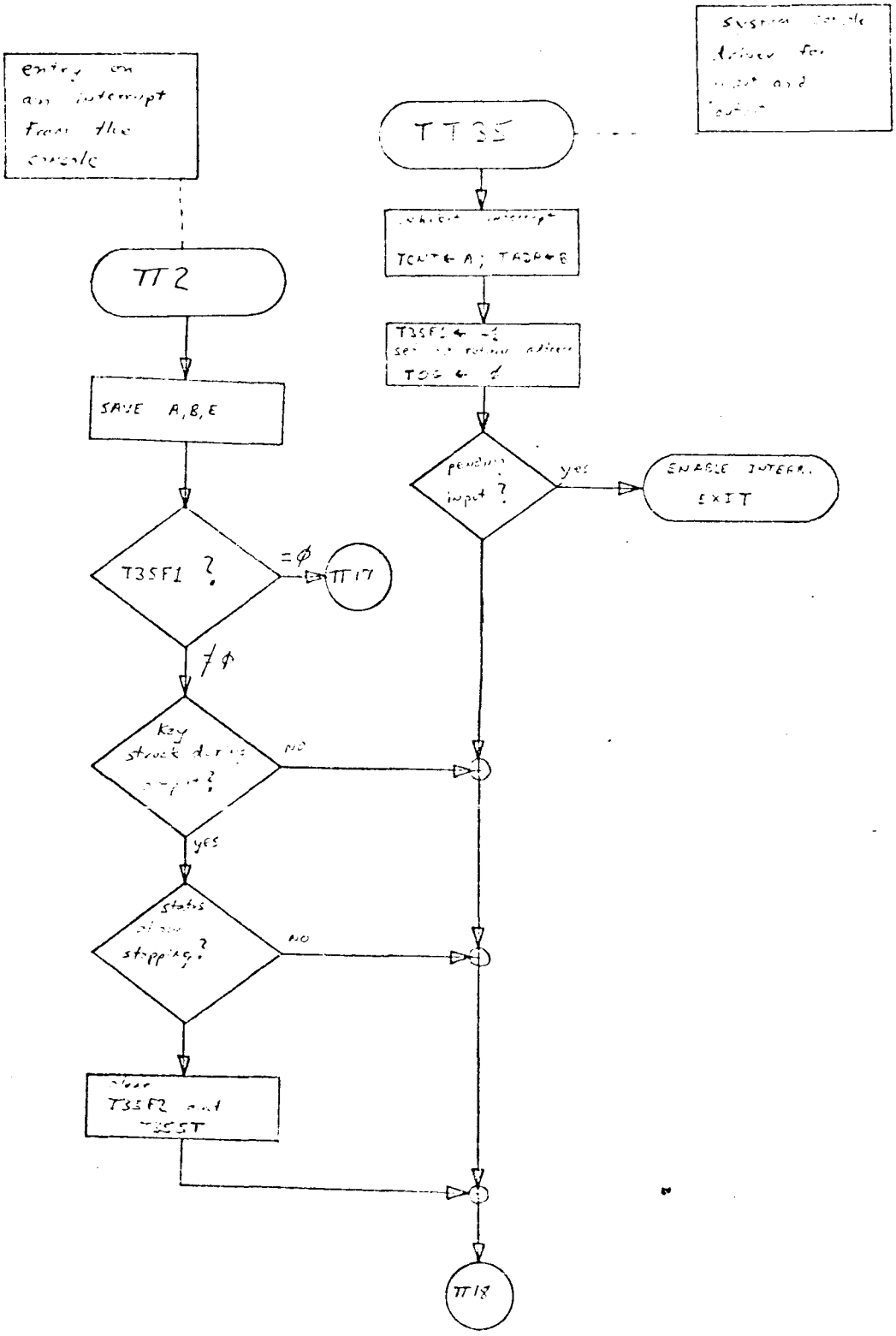
Either BASIC or a system library routine terminates by:

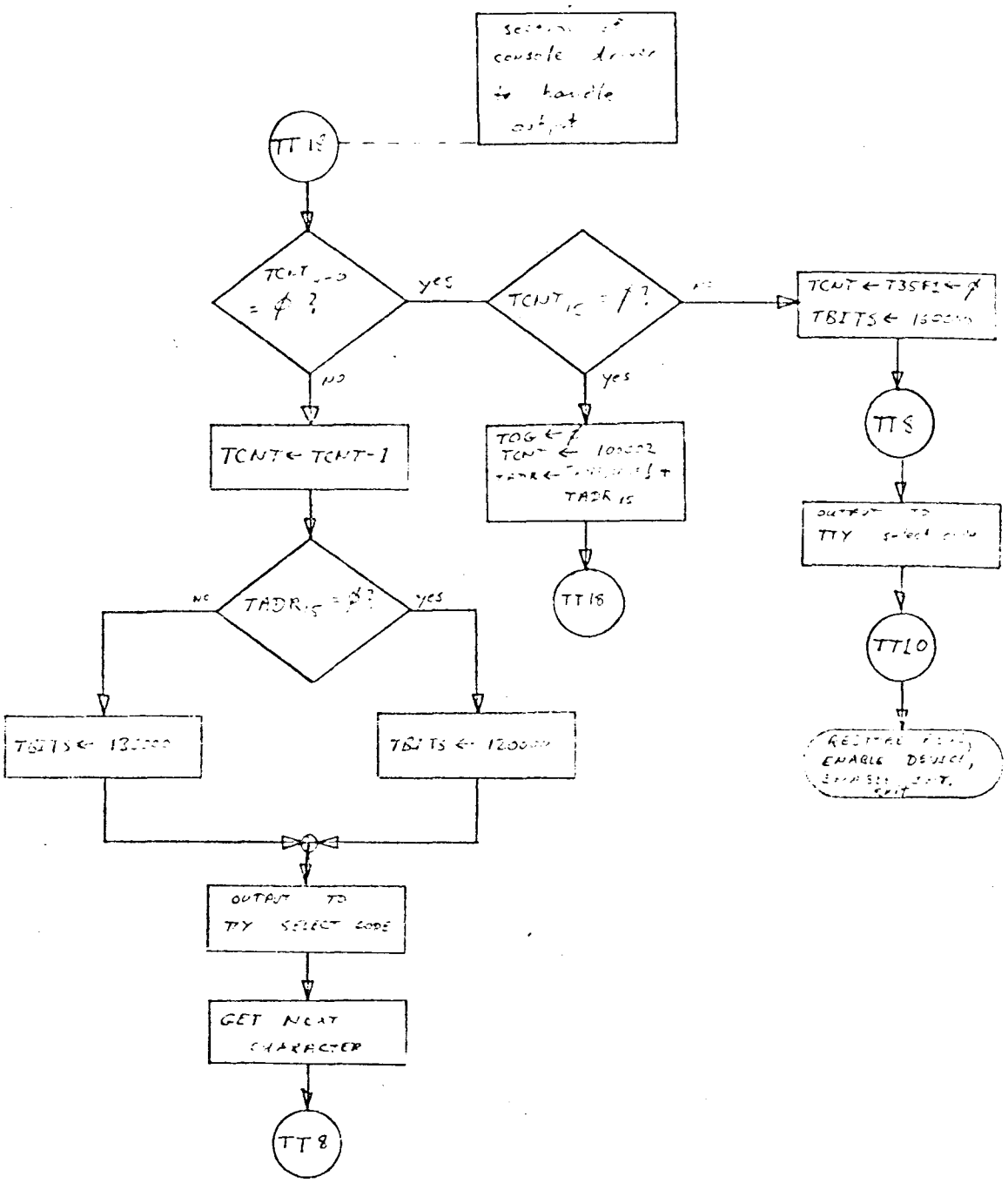
```
JSB SCHEN,1
```

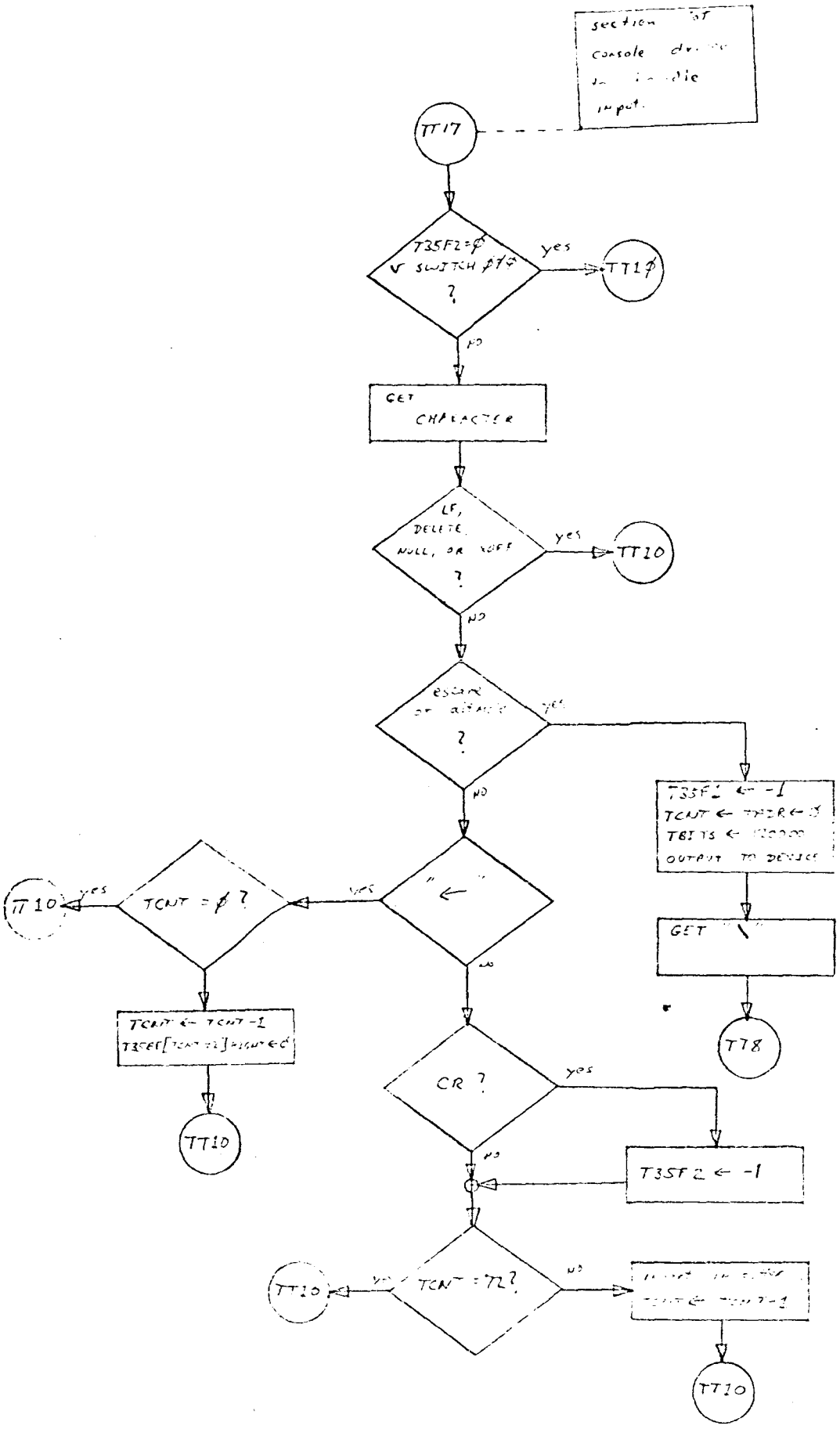
It is possible for BASIC to call a system library routine directly by executing:

```
JSB SCHLB,1
DEF <location in COMTABLE of disc address of program>
```

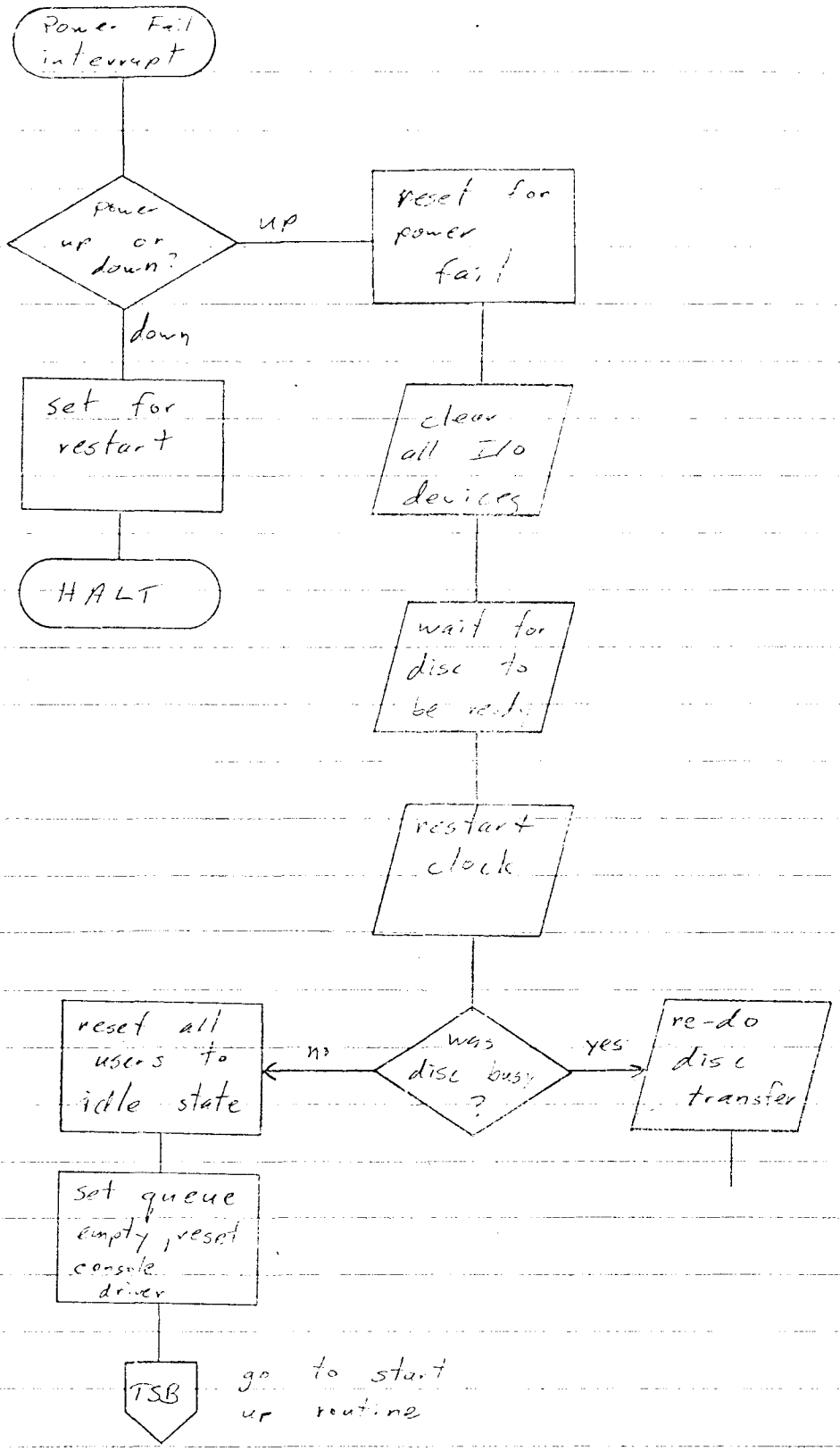
This is done with the FILES and CHAIN routines. It is necessary that the library routine cooperate with BASIC, i.e., not any program can be so called.

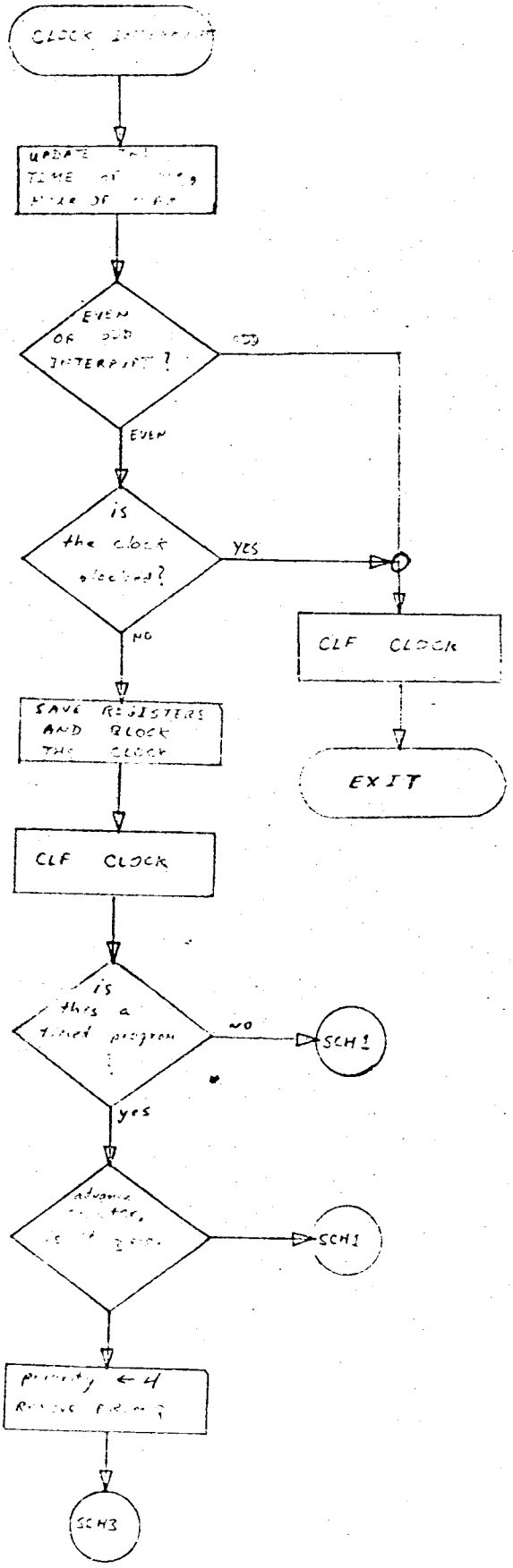

entry on an interrupt from the console
TT2
SAVE A,B,E
T35F1 ?
=0
TT17
≠0
key struck during ESC ?
NO
yes
status shows stopping?
NO
clear T35F2 and T35ST
TT18
TT35
inhibit interrupt
TCNT ← A; TA2R ← B
T35F1 ← -1
set up return address
T02 ← 0
pending input ?
yes
ENABLE INTERR.
EXIT


TT 1?
yes
NO
TCNT ← TCNT-1
TCNT ← T35F2 ← 0
TBITS ← 1500
TT 5
OUTPUT TO TTY SELECT CODE
TT10
TOG ← 1
TCNT ← 10000
TADR ← TADR IS
TT 18
TADR.5 = 0?
NO
yes
TBITS ← 13200
TBITS ← 12000
OUTPUT TO TTY SELECT CODE
GET NEXT CHARACTER
TT 8


section of console driver to handle input.
TT17
T35F2=0 V SWITCH 0/1=0 ?
yes
TT10
no
GET CHARACTER
LF, DELETE, NULL, OR XOFF ?
yes
TT10
no
ESCAPE OR ALTMODE ?
yes
no
T35F2 ← -1
TCNT ← TPTR ← 0
TBITS ← 100000
OUTPUT TO DEVICE
GET "\"
T78
" ← "
yes
TCNT = 0 ?
yes
TT10
TCNT ← TCNT-1
T35BF[TCNT+1] RIGHT ← 0
TT10
no
CR ?
yes
T35F2 ← -1
no
TCNT = 72 ?
yes
TT10
no
TT10

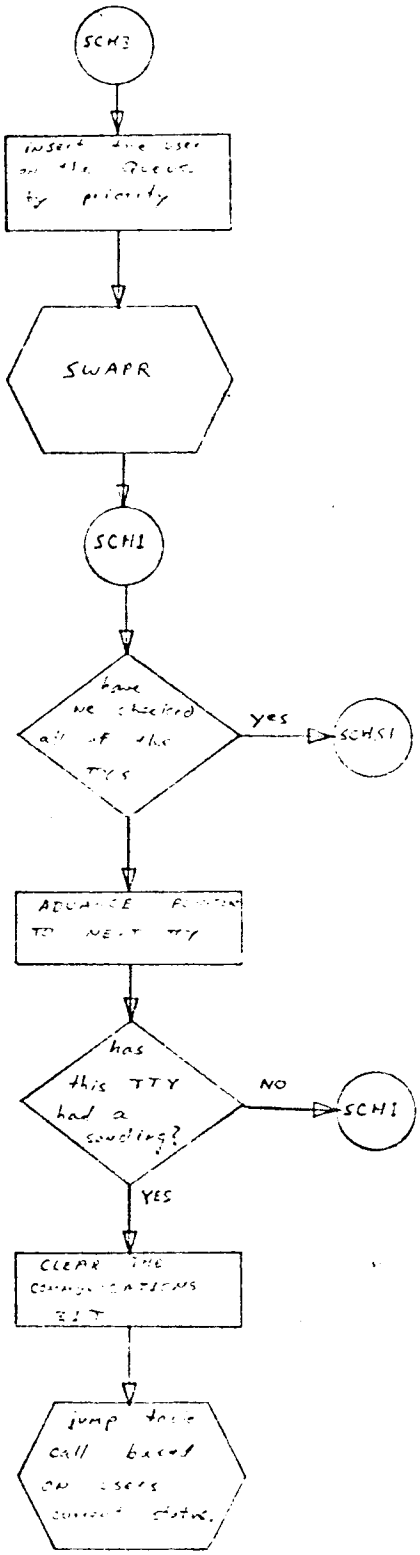
Power Fail

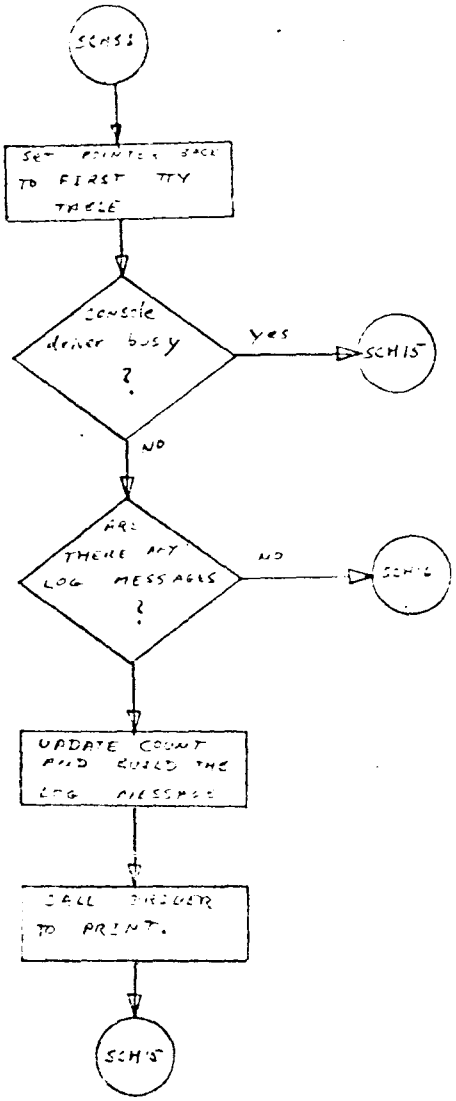

Power Fail interrupt
power up or down?
up
down
reset for power fail
set for restart
clear all I/O devices
HALT
wait for disc to be rdy
restart clock
was disc busy ?
no
yes
reset all users to idle state
re-do disc transfer
set queue empty, reset console driver
TSB
go to start up routine

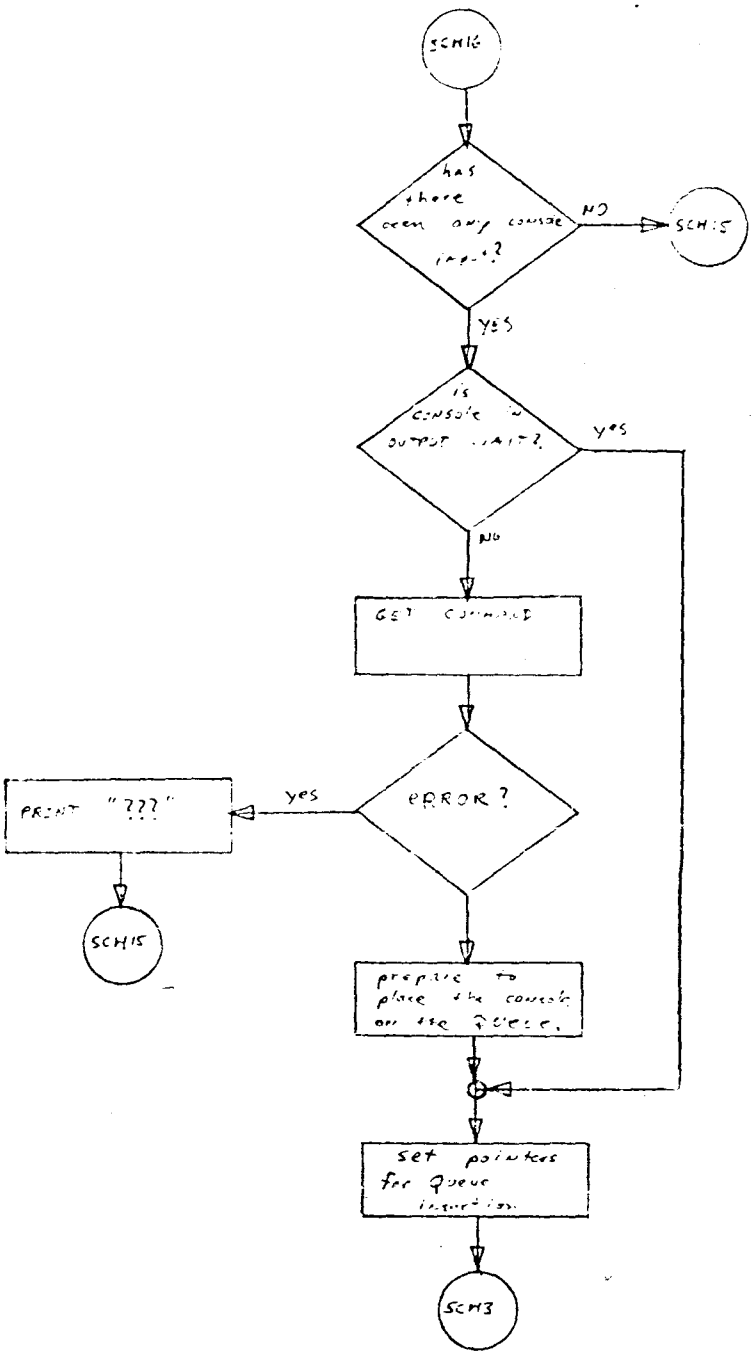

CLOCK INTERRUPT
UPDATE
TIME OF
EVEN OR ODD INTERRUPT?
ODD
EVEN
IS the clock blocked?
YES
NO
CLF CLOCK
EXIT
SAVE REGISTERS AND BLOCK THE CLOCK
CLF CLOCK
IS this a timed program
NO
SCH1
yes
SCH1
SCH3

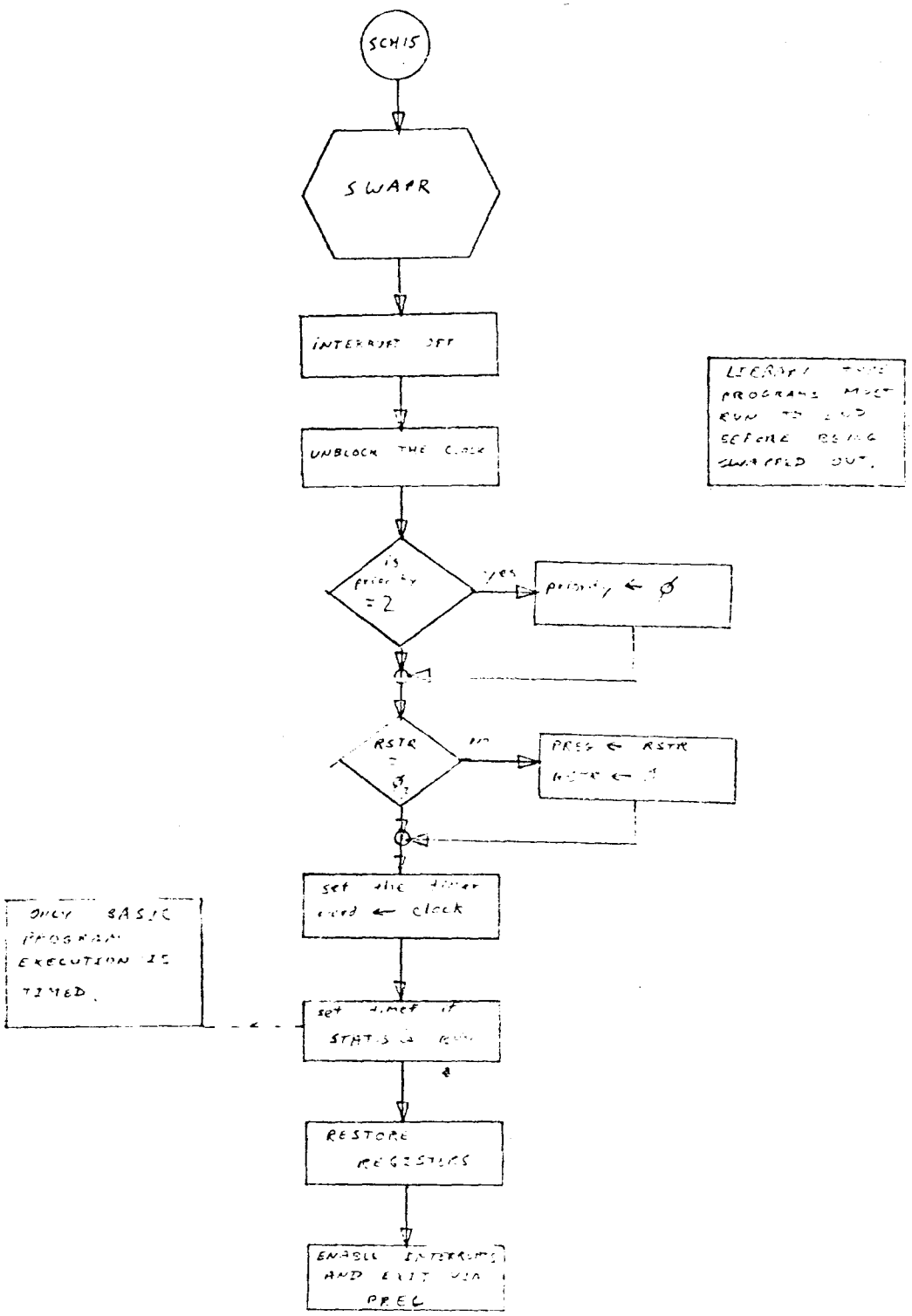
SCHEDULER

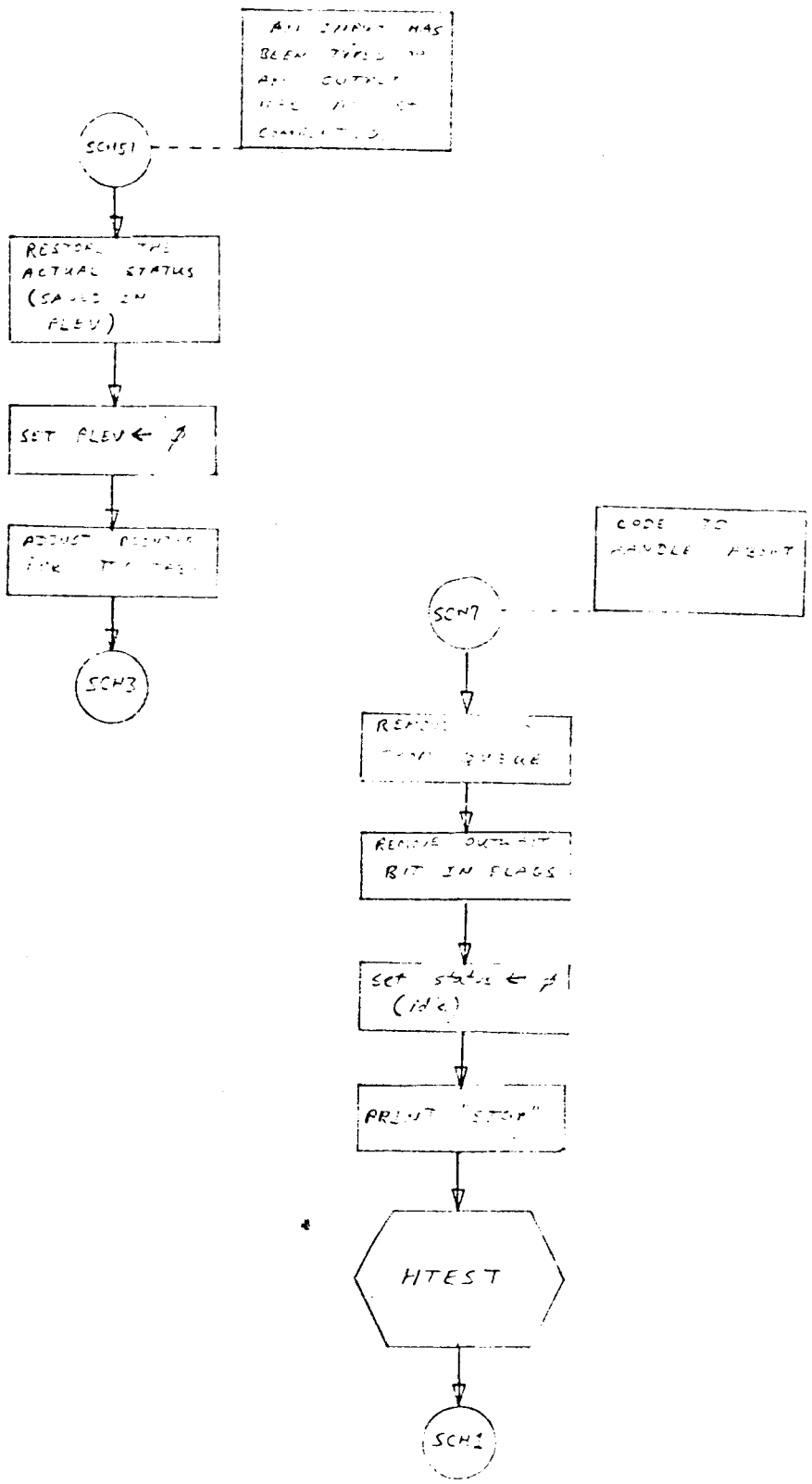
pg 2 of 11

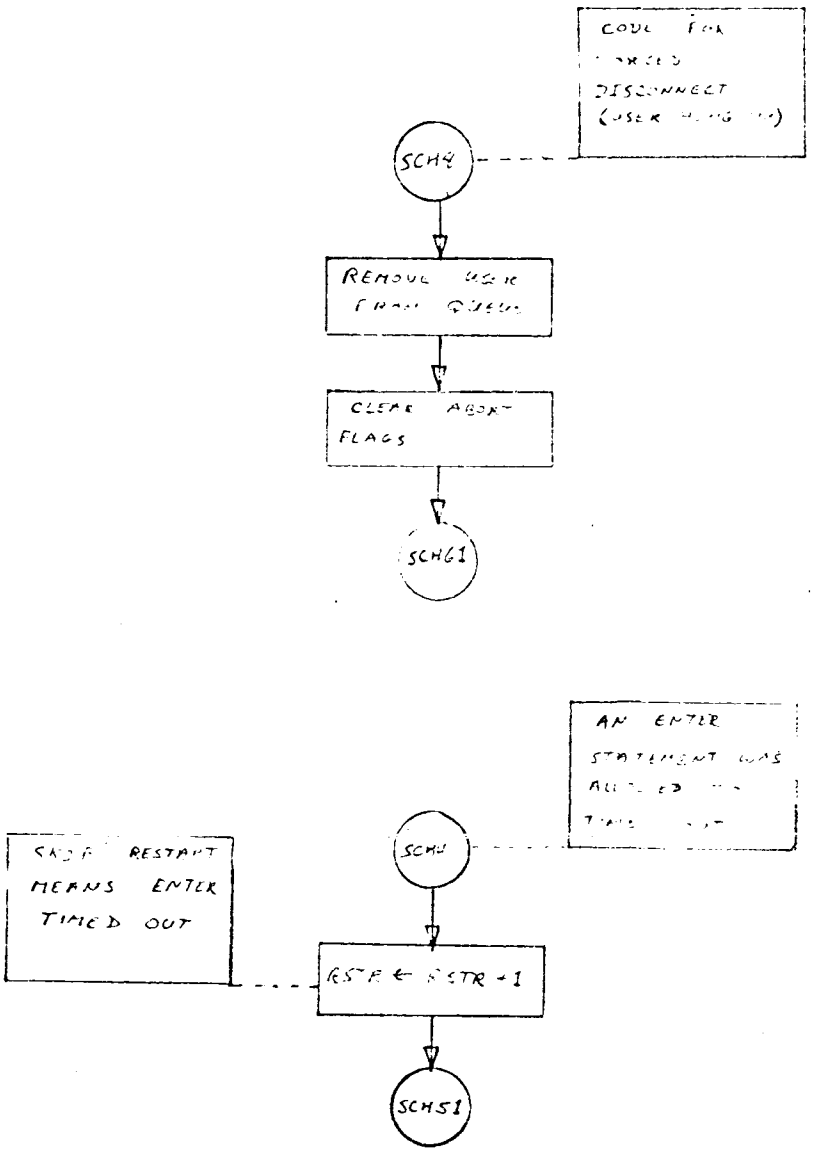


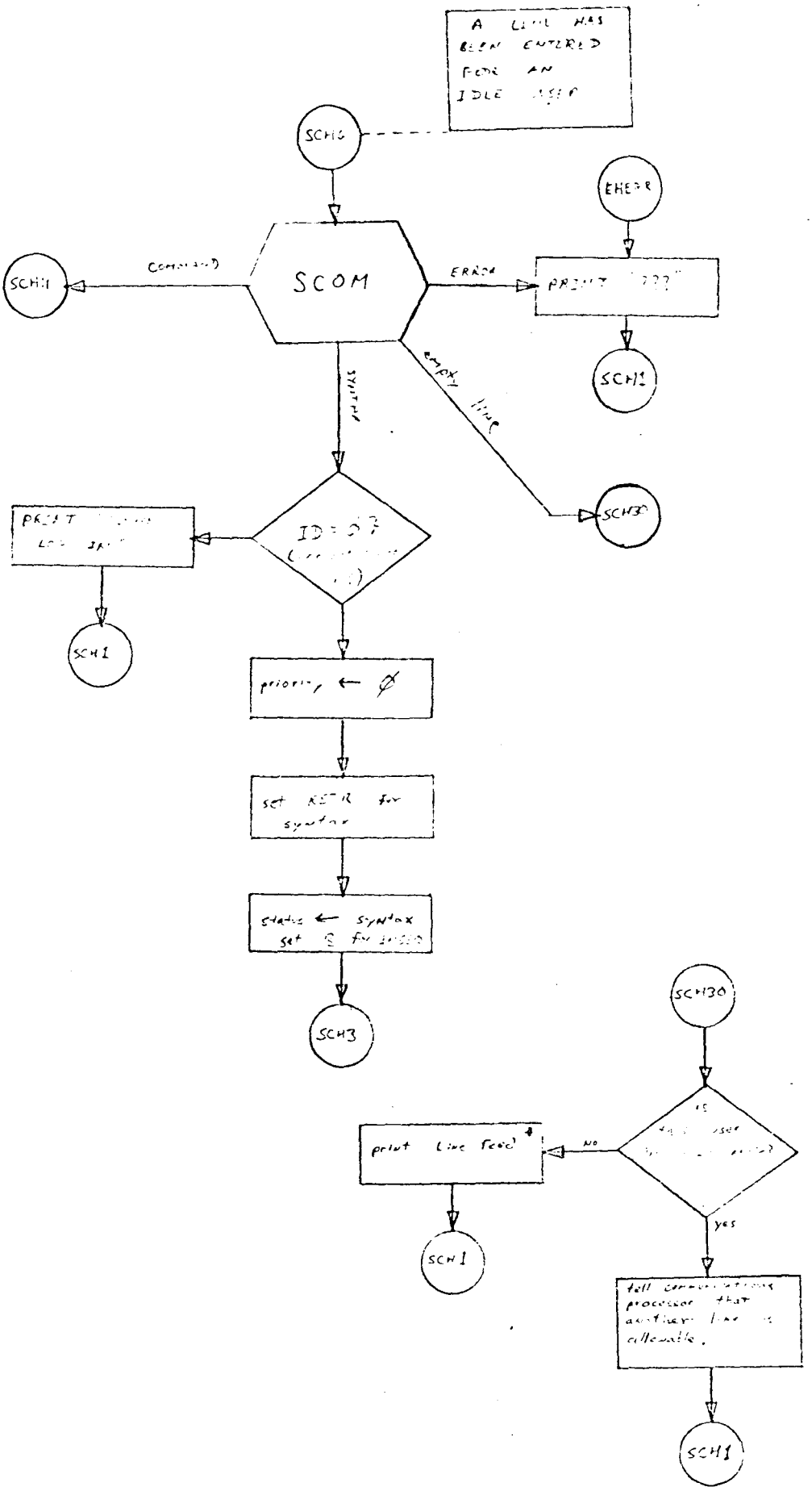

SCHSS
SET POINTER BACK TO FIRST TTY TABLE
CONSOLE driver busy ?
Yes
SCHIS
NO
ARE THERE ANY LOG MESSAGES ?
NO
SCHIS
UPDATE COUNT AND BUILD THE LOG MESSAGE
CALL DRIVER TO PRINT.
SCHIS

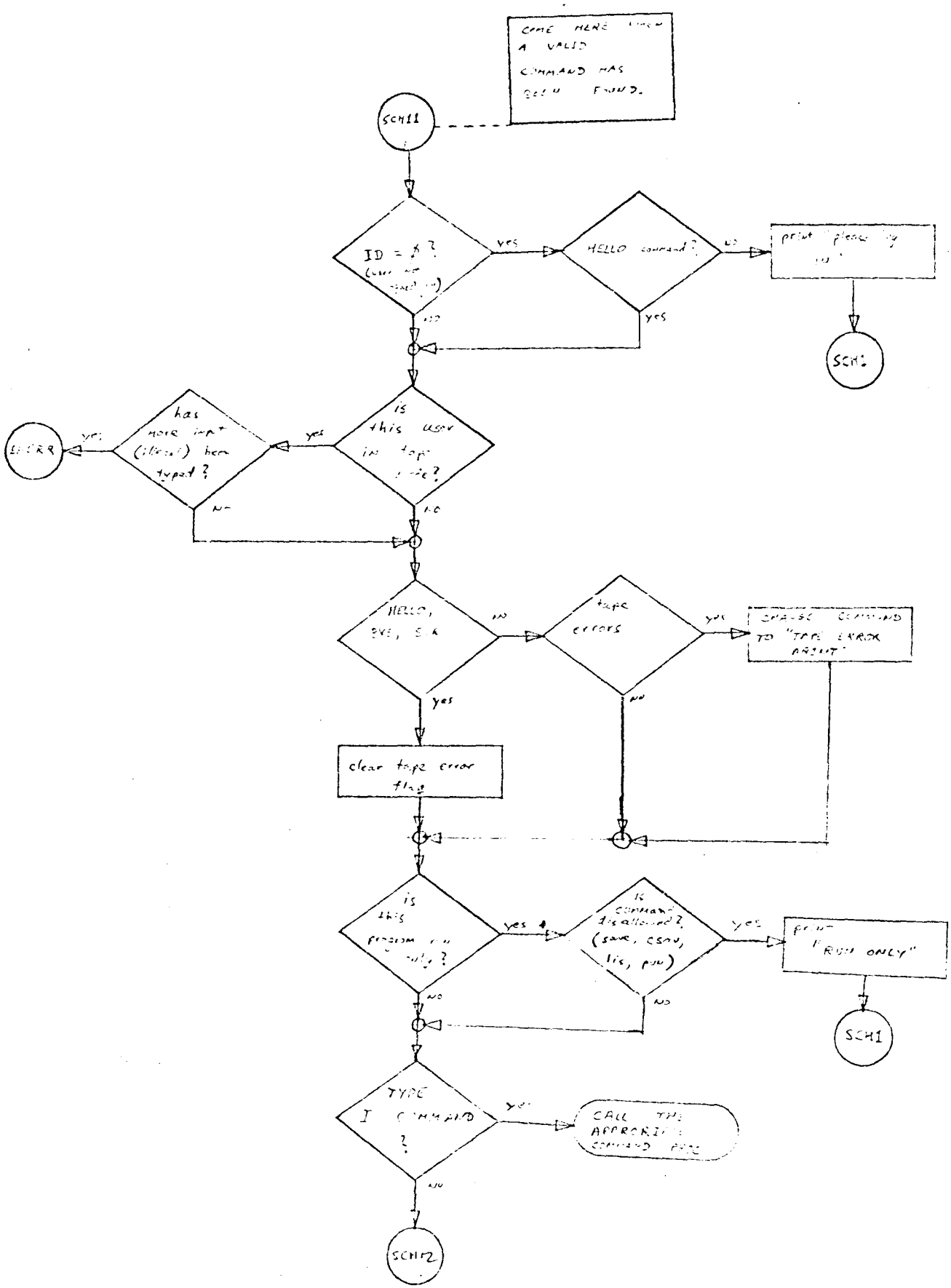

SCH16
has there been any console input?
NO
SCH15
YES
is console in output wait?
yes
NO
GET COMMAND
ERROR?
yes
PRINT "???"
SCH15
prepare to place the console on the queue.
set pointers for queue insertion.
SCH3

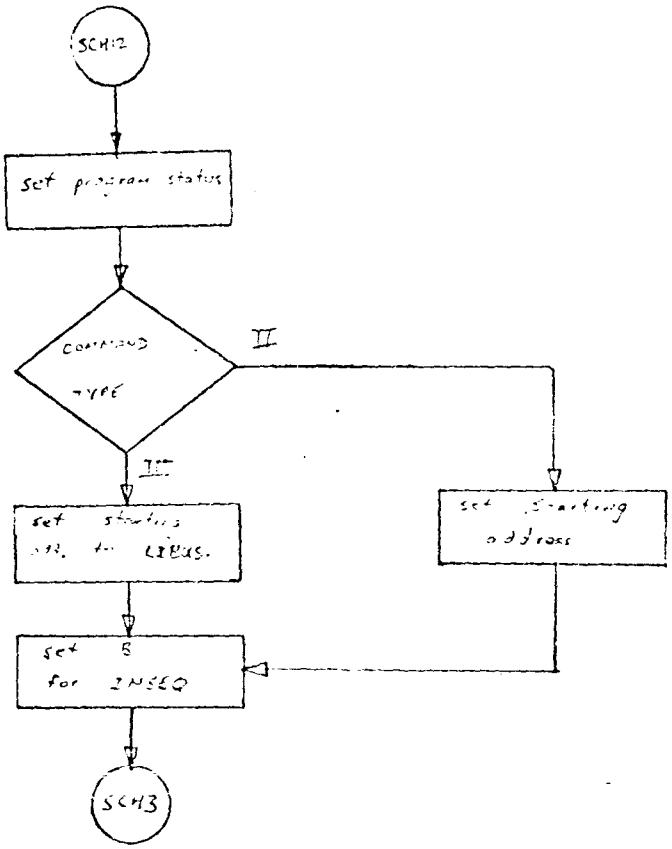

SCHIS
SWAPR
INTERRUPT OFF
UNBLOCK THE CLOCK
IS PRIORITY = 2
yes
PRIORITY ← 0
RSTR = 0
no
PRES ← RSTR
set the timer
next ← clock
ONLY BASIC PROGRAM EXECUTION IS TIMED.
set timer if STATUS = RUN
RESTORE REGISTERS
ENABLE INTERRUPTS AND EXIT VIA PREC

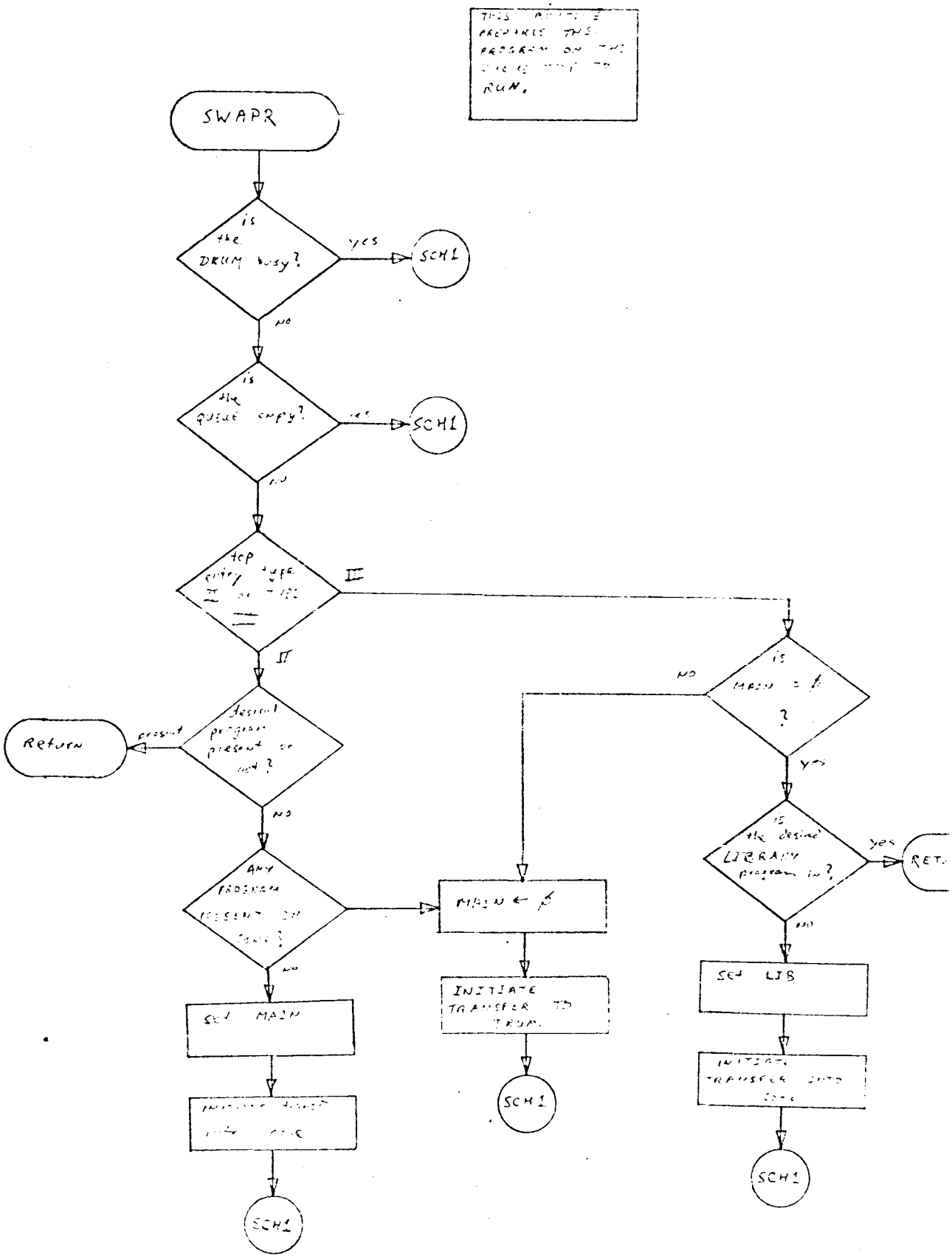

SCH51
RESTORE THE ACTUAL STATUS (SAVED IN FLEV)
SET FLEV ← ∅
SCH3
SCH7
HTEST
SCH2


SCH4
REMOVE USER FROM QUEUE
CLEAR ABORT FLAGS
SCHG1
SCH5
RSTR ← RSTR + 1
SCH51


A LINE HAS BEEN ENTERED FROM AN IDLE USER
SCH0
ERROR
SCOM
COMMAND
ERROR
PRINT "???"
SCH1
SCH2
EMPTY LINE
SCH30
STRING
ID = ??
PRINT
SCH1
priority ← ∅
set ASTR for syntax
status ← syntax
SCH3
SCH30
print Line Feed
NO
SCH1
YES
SCH1


COME HERE WHEN A VALID COMMAND HAS BEEN FOUND.
SCH11
ID = 0 ?
yes
HELLO command ?
yes
print "please log in"
SCH2
has more input (illegal) been typed ?
yes
SCERR
is this user in tape mode ?
yes
no
no
HELLO, BYE, S.A
no
tape errors
yes
CHANGE COMMAND TO "TAPE ERROR ABORT"
yes
no
clear tape error flag
is this program run only ?
yes
is command disallowed ? (save, csav, lic, pw)
yes
print "RUN ONLY"
SCH1
no
no
TYPE I COMMAND ?
yes
CALL THE APPROPRIATE COMMAND PROC
no
SCH12


SCH2
set program status
command type
II
III
set starting address
set starting add. to LIBUS.
set S for INSEQ
SCH3


SWAPR
is the DRUM busy?
yes
SCH1
no
is the QUEUE empty?
yes
SCH1
no
top entry type II or III
III
II
desired program present or not?
present
Return
not present
ANY PROGRAM PRESENT SW TRUE?
no
SCH MAIN
SCH1
yes
MAIN ← p
INITIATE TRANSFER TO DRUM
SCH1
no
is MAIN = p ?
yes
is the desired LIBRARY program in?
yes
RET
no
SCH LIB
INITIATE TRANSFER INTO CORE
SCH1

FILES

The FILES routine is used by BASIC to process FILES statements in a user's program. The function of the FILES routine is to translate the file names in the user's program into a table for use during execution. This table contains a 7-word entry for each file. Its format is:

1. Physical length in sectors (BIT 15 = 1 if read only)
2. Disc address of last logical sector
3. Not set by FILES routine
4. Disc address of first sector

5-7. Not set by FILES routine

During operation of the FILES routine, a temporary buffer is used as a table to store intermediate data. Three words of the buffer are used for each file. The operation is as follows:

1. Translate characters in FILES statements into the buffer table. The FILES statements is pointed to by VALTB ~~a four word table in the user swap area which is pointed to by DFILT. FILCT = -5+ # of FILES statements. There may be up to 4 such statements~~. Filenames are extended to six characters, if necessary, and those which are specified to be public files are marked by setting Bit 15 of their first word to 1. Possible errors found in this step are:

   a. File name of 0 or > 6 characters
   b. More than ~~15~~ 4 files requested

2. Perform directory search for each file. DIRWD is set to the disc address of the directory track in core so that DLOOK doesn't have to read and write the directory for each file. Change the

last two words of its entry in the buffer table to the disc address and length in sectors. The read-only bit is set if the file is a public file and the user is not A000. An error occurs if the file is nonexistent or protected. Update the date word in the directory entry for this file.

3. Test to make sure that there is sufficient room in the user area for the file table.
4. Scan the FUSS table to see if any other user has write capability on the files requested. Mark any such files as read-only. This test is skipped if the user's ID has a letter prefix 'A'. Copy the disc addresses and lengths of the requested files into the user's portion of FUSS. Indicate read-only files by marking bit 7 in FUSS.
5. Build the table specified above. FILTB is a pointer to the beginning of the table. Upon exit, VALTB and PBPTR both point to the first word following the table.

CHAIN

The CHAIN routine is used by BASIC to process a CHAIN statement in a user's program. The function of the CHAIN routine is to find the program named in the CHAIN statement, retrieve it from the disc, and begin execution. It operates as follows:

1. Dump file buffers.
2. Translate name of program from CHAIN statement. Invalid names exit to error. If preceded by a $, set up AØØØ search; otherwise set for searching on user's ID.
3. Perform directory search. Exit to error if not found.
4. Check if entry is a file. If so, exit to error. Also check that program fits. If not, exit to error.
5. Update date entry in directory and write directory track back to disc. Copy the program name into the user's table, and if this is a run-only program, set the run-only bit, unless the user is AØØØ.
6. Read in the basic portion of the previous program, including the common area and then append the new program. Call SEMIC, which sets up pointers for the language processor, ~~dependent upon whether the program is uncompiled or semi-compiled.~~
7. Check if an abort was attempted during the previous steps, and if so, abort the user.
8. Bump the user's timeout clock. If he times out, take him off the queue and reinsert him with priority = 4 and jump into the scheduler. Otherwise, jump to the compiler.

SAVE

The SAVE routine is called by a user to save a program in the library. Its operation is as follows:

1. Test for the existence of a program name and a non-null program.
2. If the user's program is in compiled form (CFLAG bit = 1), call DCMPL to put it into the form in which we will save it.
3. Check if the common area has been allocated. If not, call ALCOM, which computes the amount of space required for common. This is used to determine the start-of-program pointer which is saved in word 4 of the directory entry, a device which keeps the common area from being overwritten on GET's and CHAIN's.
4. Test to see that the user has sufficient disc space allocated to save the program. The test to be satisfied is:

   (disc currently in use) + (length of program in sectors) $\leq$ (disc allowed).
5. Search the ADT for the first entry large enough to hold the program. Remember the address of the entry in SAVA.
6. Perform a directory search on the program to be saved. Fail if such an entry already exists.
7. If the directory track is full, call the SUPERSAVE routine to attempt to reallocate the directory. ~~SUPERSAVE will perform step 8 itself and proceed to step 9.~~
8. Insert a new directory entry into the directory.
9. Update the IDT and ADT.
10. Copy the user's program to its library area.

SUPERSAVE

The SUPERSAVE routine is called by SAVE, OPEN and MOVE routines when they want to make a directory entry on a track that is already full. SUPERSAVE attempts to redistribute the two two directory tracks for the indicated disc so that they will be equal in length. The operation is as follows:

1. Compute total length of both tracks
2. Check if both tracks are full. If so emit system standard message and terminate
3. Divide by 2 to determine new track lengths
4. Redistribute tracks (see flow chart)
5. Update DIREC entries
6. Return to calling routine to make the new entry.

GET

The GET routine is called by a user to load a program from the library. The operation is as follows:

1. Translate name of program from user's input. If preceded by a $, set up for A000 search; otherwise set for searching on user's id.
2. Perform directory search. Print error if not found.
3. Fail if entry is a file (BIT 15 of word 2 of entry is 1). Check that the program will fit into the user area. This is necessary in case a program which was saved under an old version of the system can no longer fit with the current version.
4. Set the date into word 5 of the directory entry and write it back. Copy the program name into the user's table, and if this is a run-only program, set the run-only bit, unless the user is A000.
5. Read in the basic portion of the user area and the common area. Append the library program, reading it in starting with the word specified by the start of program pointer (word 4 of the directory entry).
6. Call SEMIC, which sets up pointers as follows: ~~For uncompiled programs, clear CFLAG bit and set SYMTB = 0. For semicompiled programs, set CFLAG bit, move 4 pointers to FILES statements into FLSTS, set FILCT, set SYMTB to point to the first word of the symbol table and set SPTR=0. For both types of programs~~ set MAIN to point to this user, set SPROG to the start of program pointer and set PBPTR to point past the last word used by the program.

APPEND

The APPEND routine is called by a user to append a library program onto his current program. The operation is the same as GET for steps 1-3, and then continues as follows:

4. Check that the program to be appended is not semicompiled and has no common area. Set the date into word 5 of the directory entry and write it back.
5. Load user's current program and call DCMPL. Check that the program to be appended will fit, and if so, read it in at the end of the current program.
6. If the current program is not null, search it for the sequence number of the last statement, and insist that it be smaller than the sequence number of the first statement of the appended program. If okay, update PBPTR and exit.

HELLO

The HELLO command is used to log a user on to the system. Its operation is as follows:

1. If the current id is 0, there is no user to log off, so go to step 2. Otherwise, clear the user's section of FUSS, set the phones control board to time the user log on. and ~~tell the I/O processor that a new user called~~. This will force the user to be disconnected if he does not successfully log on.
2. ~~Read the IDT~~. If there is no user to be logged off, go to step 3. Find the old user's IDT entry and update his total time used. Add an entry to LOGGR to be printed on the system console. Set the user's ID word to 0.
3. Translate the new idcode and search for it in the IDT. If not found, print an error message and terminate. Compare the password typed to the correct one, and fail if they disagree. Also, check that the time used to date is less than the time allowed.
4. Add a LOGON entry to LOGGR, and set the starting time into the user's table. Also insert the idcode, clear the name, clear the program, ~~and tell I/O processor of successful logon.~~
5. Search the directory for a public library HELLO program. If not found, or if it is a file, or if it won't fit in core, print READY and terminate.
6. Read in the fixed user area and append $HELLO. Call SEMIC, which sets pointers as in SAVE. Change the user's status to RUN, set TIMEF, and transfer to BASIC.

4. Get the terminal type if there is one and set up teletype table for proper terminal type. If type is invalid go to fail exit.

BYE

This command is used to log a user off. It operates as follows:

1. If the user id is 0, go to step 2. Otherwise clear the user's FUSS table and read in the IDT. Compute the time used and update his IDT entry. Create a LOGOFF entry in LOGGR. Clear the user's id entry and output a message.
2. ~~Tell the I/O processor to~~ restore this port to full duplex and disconnect him and then terminate.

KILL

The KILL routine is called by a user to delete a program or a file from the library. Files which are being accessed by another user are not allowed to be killed. The operation is as follows:

1. Translate the program or file name and perform a directory search. Fail if illegal name or the search fails.
2. If the entry is a file, search the FUSS table to see if any other user has access to the file. If so, print a message and terminate. If not, clear the user's section of FUSS.
3. Delete the entry from the directory and adjust DIREC. Subtract the program length from the user's IDT entry, and restore the space to the ADT.
4. If a file was killed, read the user's program in and decompile it. This guarantees that any old references to the file will disappear.

RENUMBER

The function of RENUMBER is to assign a new set of sequence numbers to a user program. The user may specify the sequence number of the first statement and the increment between statements. If unspecified, these are set to 10.

There are actually two sets of numbers that must be modified. One set is the sequence numbers themselves, each of which occupies the first word of its statement. The other is the set of references, which are labels in GO TO, GOSUB, RESTORE, and IF statements. Each of these also occupies one word. For programs in compiled mode, they are pointers to the statement they reference; in decompiled mode they are the actual statement number.

The primary technique used is to change all the references to absolute pointers (if in decompiled mode), then to change all the sequence numbers, and then (if in decompiled mode) to change the references to the new statement numbers. References to nonexistent labels are left unchanged.

Because the process of changing all the references to absolute pointers can become quite time consuming (due to the search that must be performed for each reference), a table is built in advance essentially dividing the program into 32 parts, each containing the same number of statements. For large programs with many references, this effectively cuts the time down by a factor of close to 32.

The subroutine RENSK is used to scan for references. It maintains two pointers, P and Q. Whenever it is called, it moves P to the next reference, and sets Q to point at the statement following the one that P is pointing at. It takes advantage fo the fact that any references within a statement are always the last word or words of the statement. Before calling RENSK for the first time, Q is set to point at the beginning of the program, and P is set to Q-1.

The operation of RENUMBER is as follows:

1. If null program, terminate immediately. Otherwise, read in user program.
2. Translate and check parameters M and N.
3. Scan through program and make sure that the new sequence numbers will not exceed 9999.
4. If program is in compiled mode, go to step 7. Otherwise, set up a table in ERSEC which divides the program into 32 parts. The result is that for each I from 0 to 31

   ERSEC [I] = sequence number of first statement in part I,

   ERSEC [I+32] = Absolute address of that statement

   If there are 32K + L statements ($0 \leq L \leq 31$) in the program, ERSEC [I] is the sequence number of statement.

   (K + 1) I + 1, if I< L

   KI + L + 1, if I≥ L, K 0

   L if I≥ L, K=0

   Set Q = SPROG, P + Q-1. (SPROG points to the first statement).
5. Call RENSK to find the next statement reference. If there are none left, go to step 7. Find the largest I for which ERSEC [I] ≤ (RENP). If there is none, the statement referenced does not exist, so go to step 6. Otherwise, test all statements from (ERSEC [I + 32]) to either (ERSEC [I + 33]) or PBPTR, depending upon whether I < 31 or I = 31. If found, set (RENP) to the location of the statement referred to, and repeat this step. Otherwise, go to step 6.
6. Set (RENP) = (RENP) + 100000_8 and go back to step 5.
7. Change the sequence numbers of all statements, according to the M and N parameters. If compiled mode, terminate. Otherwise, set Q = PBUFF, P = Q-1, and go to step 8.
8. Call RENSK to find the next statement reference. If none left, terminate. If (RENP)<0, the reference was undefined, so set (RENP) = (RENP)-100000_8, and repeat this step. Otherwise, set RENP = ((RENP)) and repeat this step.

NAME

The NAME routine is called by a user when he wants to assign a name to his program. The program name is placed in his teletype table. The operation is as follows:

1. Get an input character. If a carriage return change it to a blank. If a control character, ignore it and repeat this step. If a "$", and this is the first character, print an error message and terminate.
2. Add the character to the user's name area. If <6 characters, go back to step 1. Otherwise, restore the RUN-ONLY bit, and get one more character. If not a blank, print an error message. Then terminate.

CATALOG

The CATALOG routine prints a list of all programs and files in the user library. The operation is as follows:

1. Perform directory search on the program with all nulls. Get first directory entry following the one sought.
2. If the entry does not belong to this user, output a CRLF and terminate. Otherwise, output the 6 characters of the name one at a time, then a ~~'C' if a semi-compiled program or~~ an 'F' if a file (or a blank if ~~neither~~ a program), then the 4 digits comprising the length of the program or file, and the 2 more blanks.
3. If <5 names have been printed on the line, advance to the next directory entry and return to step 2. Otherwise, copy ~~the name of the last one output~~ all pointers into ERSEC ~~(0:2)~~ in the user area, output a carriage return and suspend until the buffer is almost empty.
4. Read ~~the name of the last program printed~~ the pointers from ERSEC (0:2) in the user area, ~~and perform a directory search. The reason for doing this in this way rather than saving a pointer to the directory is that during the time CATALOG was suspended, the directory may have been changed in any way. Get the first directory entry following and go back to step 2.~~

   Read the directory track and go back to step 2.

If the entries for the indicated user are exhausted for one disc, set the pointers to the next available user disc and continue at step 1.

LIBRARY

The LIBRARY routine prints a list of all programs and files in the public library. Its operation is identical to that of CATALOG except that A000 is used for directory searches instead of the user's id.

DELETE

The DELETE command allows a user to delete a section of his program. He can specify two parameters, M and N. M refers to the first line to be deleted, N to the last. If N is not specified, the entire program is deleted, starting at line M. The operation is as follows:

1. Translate and check parameters. If N is not specified, set it to 9999.
2. Decompile program.
3. Locate range of statements to be deleted.
4. Move portion of program following deleted area up against portion preceding.
5. Reset PBPTR and exit.

DISC

The DISC command prints the user's allowed disc space and total disc space used.

1. Print "DISC ALLOWED ="
2. Find the ED in IOT
3. Print disc allowed (in sectors)
4. Print "DISC USED ="
5. Print disc space used
6. Exit

MESSAGE

The MESSAGE routine is used to transfer a character string from a user terminal to the console.

1. Check message flag to see if there is already a message in the queue. If so exit "Busy" message and terminate.
2. Get the port number and output to the message teletype buffer.
3. Transfer message text from the terminal buffer to the message buffer.
4. Set the message flag to indicate that a message is in the message buffer and terminate.

PROTECT

The PROTECT command allows user A000 to protect a program or file. Program protection means that no other user may list or save the program. File protection means that no other user may access the file. A000 files are always protected against other users writing on them. The operation is as follows:

1. Check for A000.
2. Translate and check the program or file name.
3. Perform a directory search on the specified program. Fail if not found.
4. Set the protect bit (BIT 15 of word 1 of the directory entry), write the directory back to the disc, and terminate.

UNPROTECT

This is identical to PROTECT except that it clears the protect bit.

OPEN

The OPEN command is used to open data files. The user must specify the filename and file length in sectors (1 to ~~128~~ 48). The operation is as follows:

1. Translate and check the file name and length.
2. Check the IDT and ADT to see if a) the user has enough disc allocated to him to satisfy the command; and b) there is an area on the disc which is large enough to accommodate the file. Save the location of the ADT entry and its information, but don't update it until we know that there is room in the directory.
3. Perform a directory search on the file name. If found, this is a duplicate entry, so terminate. Otherwise, if the directory track is not full, insert the new entry. If it is full, call in SUPERSAVE to restructure the directory and insert the entry.
4. Update the IDT and ADT appropriately.
5. Initialize the file so that a -1 (end-of-file) is at the beginning of every sector. Write the file to the disc and then terminate.

LENGTH

The LENGTH command prints the length of the user's program, as it would be if saved. This is only the length of the source area of the program, and includes neither the fixed portion nor any of the tables used at run time. The length is determined in one of two ways:

1. if the user is in decompiled mode, length = PROG-SPROG. PROG is just a copy of PBPTR, which points to the last word +1 of the program. PBUFF points to the first word.
2. if the user is in compiled mode, length = SYMTB-SPROG.

ECHO

The ECHO command is used to control the computer echo of teletype input. Echoing is determined by the user's bit in the word PLEX or PLEX1 in the I/O processor. Bit = 0 implies no echo, 1 implies echo. The user will want echoing if and only if his teletype is full duplex. The command format is:

ECHO-ON for full duplex.
ECHO-OFF for half duplex.

REPORT

The REPORT command prints IDT information on the system console. From each IDT entry, the user id, time consumed, and disc consumed are printed. The entries are printed three per line. Note that the time printed on the console does not include any time for currently active users, since these are not added to the IDT until the user logs off. The operation of REPORT is as follows:

1. Print heading and suspend
2. Read ~~portion~~ track of IDT containing next three IDT entries.
3. Translate id, time, and disc of next three entries into output buffer. If less than three left, only do those.
4. Print and suspend if necessary, otherwise terminate.
5. Go back to step 2.

RESET

The RESET command modifies the time to date of a user's IDT entry. The format is: RES-IDCODE, TIME

1. Read IDT.
2. Set ID = T = 0.
3. If IDCODE = "ALL", go to step 4, otherwise get ID = the specified IDCODE.
4. If no time specified, go to step 5. Otherwise, set T = specified time.
5. If ID = 0, set word 5 of all IDT entries to T. Otherwise, locate specified id and set word 5 to T.
6. Write IDT back to disc and terminate.

SPEED

THE SPEED COMMAND IS USED TO CONFIGURE THE USER PORTS.

THE FORMAT IS: SPE- BAUD RATE, STOP-BIT , PORT LIST

BAUD RATE — THE DATA TRANSFER RATE OF THE USER TERMINAL FOR WHICH THE PORT IS TO BE CONFIGURED. THE BAUD RATE CAN BE COMPUTED BY THE FOLLOWING FORMULA:

$$\text{BAUD RATE} = \frac{14,400}{\text{BIT RATE}} - 1$$

STOP-BIT — THE TOTAL NUMBER OF STOP BITS INCLUDED IN THE BIT COMPOSITION OF A CHARACTER.

PORT

THE PORT COMMAND IS USED TO PRINT THE NUMBER OF STOP BITS AND THE BAUD RATE FOR EACH OF THE 11 PORTS. THE FORMAT IS TWO EIGHT ENTRY LINES WHERE EACH ENTRY HAS THE FOLLOWING FORM:

STOP-BITS – BAUD RATE

CHANGEID

The CHANGEID command is used to modify any or all of the parameters in an IDT entry. The parameters that can be specified are: password, time allowed, disc allowed. The operation is as follows:

1. Translate id specified. Read IDT and locate the specified id. Fail if not found.
2. If password specified, insert into IDT entry. If followed by comma, go to step 3, otherwise to step 5.
3. If time specified, insert into entry. If followed by comma, go to step 4, otherwise to step 5.
4. Insert new disc value.
5. Write IDT back to disc and terminate.

DIRECTORY

The DIRECTORY routine prints a list of directory entries. The entries are printed one per line. The items printed are: id, name, date, disc subchannel, track + sector, length (in sectors), file indicator and protect indicator.

1. Check to see if subchannel was specified. If so, print only directory for this subchannel & go to step 3.
2. Check to see if idcode was specified. If so print only the directory entries for this id.
3. Print heading and suspend
4. Set up parameters for appropriate subchannel and id code
5. Search directory. If entry is the dummy terminator, then if all subchannels are to be listed set pointers to next subchannel. If all entries have been output then exit. If the idcode was specified and the entry does not belong to that entry go to next entry.
6. Print entry. If idcode is the same as last entry printed then replace it with blanks, otherwise print the idcode. Suspend.
7. Go to step 5.

ANNOUNCE

The ANNOUNCE routine is used to transfer a message from the console to a user terminal (or to all active terminals).

1. Check first parameter if it is "ALL" set counter to NPORT (= # of ports on system), set port number to 0. Otherwise check if parameter is a valid number between 0 and 15. If it is, set port number to this value and set counter to -1. If invalid emit error message and exit.
2. Get character from console teletype buffer and output to terminal specified in the port number. If no characters left then terminate.
3. Increment port number and counter. If counter is zero go to step 2, otherwise output character to port specified. Continue step 3 until counter is zero.

SLEEP

The SLEEP command is used for system shutdown.

1. Print message "MAG TAPE SLEEP?" and wait for reply on console teletype.
2. If reply is "Y" go to step 4
3. Check if any user discs are UP. If so print the message "REMOVE DISC SUBCHANNEL: n" where n is the subchannel number (1-3). Exit from SLEEP
4. Remove all users from the queue + output the Sleep message.
5. Disconnect all users and turn off multiplexor
6. Update IDT entry for each active user + create a logoff entry in LOGGR.
7. Wait for console to finish
8. Read in 1st overlay
9. Clear FUSS table

The user tracks of the system disc are now packed so that the only unused area is at the end of the track.

8. Read in ADT set T=1 (T is the track number)
9. If the track is a system track or is empty ignore it. T ← T+1 if T = 200 go to step 16

10. write ADT back to disc; set S=R= disc addr. Set P=Q=STAB, P and Q point to a table which will serve as a subdirectory. Each program on track T will cause a two word entry to be created, the first word is the old disc address, and the second is the new disc address of the program following.

11. Search directory for the next program on track T. If none left go to step 12. Otherwise set MEM(P) = old disc address of program, set disc address in directory entry to SLES, P=P+1, MEM[P] = SLES = SLES + length in sectors of progam, P = P+1, and repeat this step.

12. Read in programs. If Q=P, we have read in all programs, go to step 13. otherwise read in MEM[Q+1] - R sectors from disc address MEM[Q], Q = Q+1, and repeat this step.

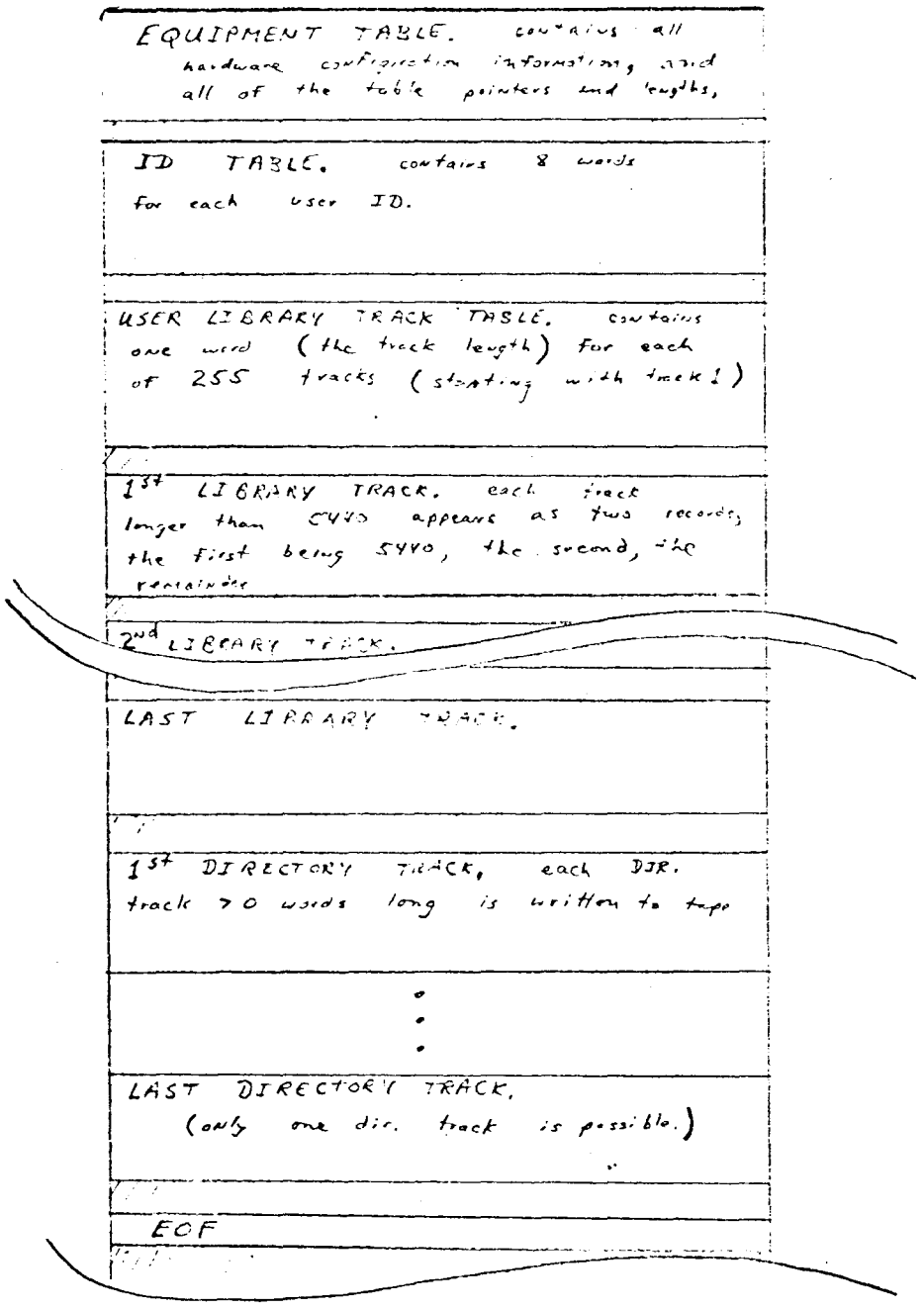
13. write R-T sectors to disc address T from core address LIBUS

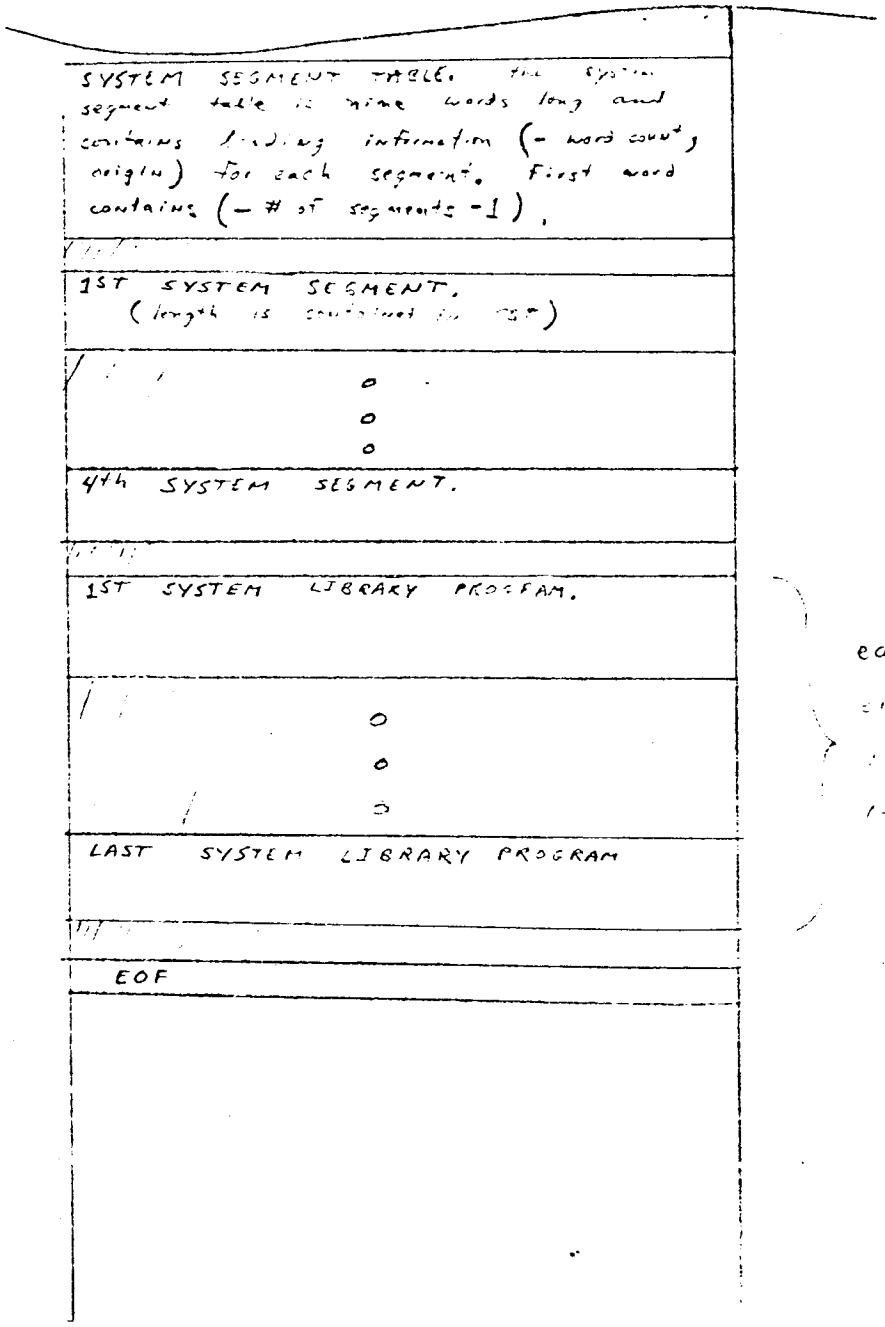
14. Read in ADT and replace all entries for track T by either no entries if the track is full, or one entry with values R and # of sectors/track + T-R.

15. Set T=T+1 go to step 9

16. Read 2nd overlay
17. clear base page data, set queue empty,
18. write equipment table to disc
19 If mag tape sleep go to step 25
20. Print message "INSERT CARTRIDGE FOR SYSTEM DISC DUMP." "PRESS RUN WHEN DISC READY"
21. HALT
22 Read a track from system disc & write it to cartridge. Do this for each track (0 - 302)
23 Print message "SLEEP COMPLETE"
24 HALT (end of sleep)
25 Load Loader/utility and transfer control to the SLEEP section.

2000B SLEEP TAPE FORMAT




SYSTEM SEGMENT TABLE. The system segment table is nine words long and contains loading information (- word count, origin) for each segment. First word contains (- # of segments -1).
1ST SYSTEM SEGMENT.
(length is contained in TBL)
o
o
o
4th SYSTEM SEGMENT.
1ST SYSTEM LIBRARY PROGRAM.
o
o
o
LAST SYSTEM LIBRARY PROGRAM
EOF
each record is

NEWID

The NEWID routine adds an entry to the IDT. The operation is as follows:

1. If the IDT is at full capacity, print an error message and terminate.
2. Read in the IDT.
3. Translate the parameters.
4. Search the IDT for the specified id. Fail if found. Otherwise insert the new entry in its appropriate position, update IDLEN, ~~write the IDT back to disc, and terminate.~~ and the appropriate track length and terminate.

Note: if the track where the id is to be added is full, then the following sequence is used.

1. Read in the correct track
2. Separate entries at the location where the new-id is to be inserted.
3. write out the maximum track length (this will leave the last entry on the track unwritten.
4. Move the entry left over to the low end of core (LIBUS) and read the next track of the IDT in after it. If the next track was full go to step 2. otherwise write the update track to the disc, update the length entry for this track and terminate.

KILLID

The KILLID routine removes a specified id from the system. The operation is as follows:

1. Get the id. If the id is A000, fail. This is because the files belonging to A000 may be accessed by other users, and removing them would be almost impossible.
2. Search the IDT for the specified id. If not found, terminate. Otherwise, delete the entry from the IDT and write it back to the disc.
3. If any user with the specified id is currently on the system, set the id item of his TTYTABLE to 0, set his status to -2 and his COM14 bit to force him to be disconnected, and remove his from the queue if he is on it. Also, zero out his section of the FUSS table.
4. Load the overlay section. This section will remove from the directory any entries belonging to the user being killed, and will release the space occupied to the system.
5. Remove all directory entries belonging to this user, and build a table which will be used to patch the ADT. For each directory entry, two words are placed in the table, the disc address and length of the released area.
6. Update the ADT, using the patch table information.

MOVE

The MOVE routine is used to transfer user programs and files from one disc subchannel to another subchannel.

1. Get the ideafa and make sure it is valid
2. Get the program or file name
3. Get the subchannel # and make sure it is valid
4. Do a directory search for the specified program (or file). If not present emit error message and terminate.
5. Get current disc address and length of program save disc address in MOUDI and length in MOVWD. Compute sector length and save in MOVLN.
6. Move correct program or file name to temporary buffer complete with file and protect bits.
7. If entry is already on the correct subchannel terminate.
8. Otherwise read ADT and search for an entry on the specified subchannel which has enough room to accomodate the program. If room is found save the ADT entry. Otherwise emit disc overload message
9. Find location for the directory entry on the directory track for the new subchannel
10. Load overlay of MOVE

11. check if directory track is full, if so call SUPERSAVE. If supersave is called it returns to step 9.

12. Put directory entry in its new location and update DIREC on base page, update track length.

13. Read ADT, make a new ADT entry for the disc space released by the move and delete the entry for the disc space used.

14. Load second overlay

16. Find old directory entry and delete it.

15. Read in program or file and transfer it to its new location

17. Exit

PURGE

The PURGE routine is used to delete from the library all programs or files which have not been referenced since a certain date. The operation is as follows:

1. If HELLO program exists, assign it today's date. This is because the HELLO routine does not perform this function.
2. Interpret parameters and set DT to the purge date. Make sure that DT $\leq$ today's date.
3. Make sure that FUSS is empty. This is to avoid killing any active files.
4. Set ID = - max (LIBUS-IDLEN, LIBUS-ADLEN)-4. This is used to determine when the update table described below has reached the point when the updates must be made.
5. Set P = ~~LIBUS~~ USEND + ~~5440~~, I = DIRDO. P is a pointer to the update table. Each entry in the update table contains 3 words:
   a) id
   b) disc address
   c) length in sectors
6. Read directory. If LIBUS-MEM [I] >P, the directory won't fit, so call PURFX to remedy the situation. Then read the directory. Set MOVED=MOVES-LIBUS, D = LIBUS-MEM [I].
7. Test next entry. If MOVES = D, we're done with this directory track, so go to step 11. If MEM [MOVES + 5] $\geq$DT, we don't want to delete the entry, so perform an 8 word move and repeat this step.
8. Entry deletion. Set T = MEM [MOVES], T1 = MEM [MOVES + 6], T2 = (- MEM [MOVES + 7] + 63) $\div$ 64, MOVES = MOVES + 8. If P-3 $\geq$D, we have room for another update entry, so go to step 9. Otherwise, set N = MOVED, perform a move of D-MOVES words, set D=MOVED, MOVED = MOVES = N.
9. If P + ID $\geq$0, we can add a new update and still be able to load the IDT and ADT, so go to step 10. Otherwise, write LIBUS through D-1 to the disc, call PURFX, and read back LIBUS through D-1.

10. Make entry in update table. Set MEM [P-1] = T2, MEM [P-2] = T1, MEM [P-3] = T, P = P - 3, and go back to step 7.
11. End of directory track. Set MEM [I] = LIBUS-MOVED, update DIREC and write the directory back to the disc. If I ≠ DIRD3, set I = I + 7 and go to step 6. Otherwise, call PURFX once more and then terminate.

The PURFX routine is brought in as an overlay. It operates as follows:

1. Save MOVED and MOVES in M and M1.
2. Read the IDT, set B = LIBUS-IDLEN-8, set PP=P.
3. If PP=LIBUS+5440, write back the IDT, read in the ADT, and go to step 5.
4. Search for ID. If MEM [PP] ≠ MEM [B], set B = B-8 and repeat this step. Otherwise, set MEM [B + 7] = MEM [B + 7] - MEM [PP+2], set PP=PP+3, and go back to step 3.
5. Update ADT. If P = ~~LIBUS + 5440~~ LSEND, set MOVED = M, MOVES = M1, write the ADT back to disc, set ID = max(LIBUS-IDLEN, LIBUS-ADLEN)-4, and exit. Otherwise, insert into the ADT the entry specified by MEM [P + 1] and MEM [P + 2], set P = P + 3, and repeat this step.

ROSTER

The ROSTER routine prints a listing of the id codes of all active users. These are obtained from the ID word in the ~~32~~ 16 TTYTABLES. The absence of a user is indicated by the word being zero.

DISC (console)

The DISC command is used by the operator from the console teletype to inform the system that a user disc is being added to the system or being removed from the system.

1. Check whether disc is going up or down. If going up go to step 10
2. Disc is going down. Check subchannel. If subchannel 0 is specified emit error message.
3. Check FUSS table to make sure there are no busy files on the specified disc. If there are emit error message and terminate.
4. Clear DIREC entries for this subchannel
5. Read ADT + extract the portion referring to this subchannel.
6. write the 3 word table of lengths to the disc on track 0 sector 1.
7. write the ADT portion for this disc to track 0, sector 2.
8. Compact the rest of the ADT and rewrite it to the system ADT track. Update ADLEN.
9. Terminate

10. Load the DISC-UP overlay
11. Check the subchannel for validity
13. Check label on the indicated subchannel to make sure it is a user disc. If not

14. Check the disc id number and compare with the system id number. If they don't compare output warning message.

15. Read lengths of ADT and directory tracks for this disc.

(12). If this subchannel is already up exit.

16. Read the system ADT into core. Find the location for the ADT entries for the new disc. Separate system ADT at that point and read in the ADT for the new disc, write the new system ADT to the system disc and update ADLEN.

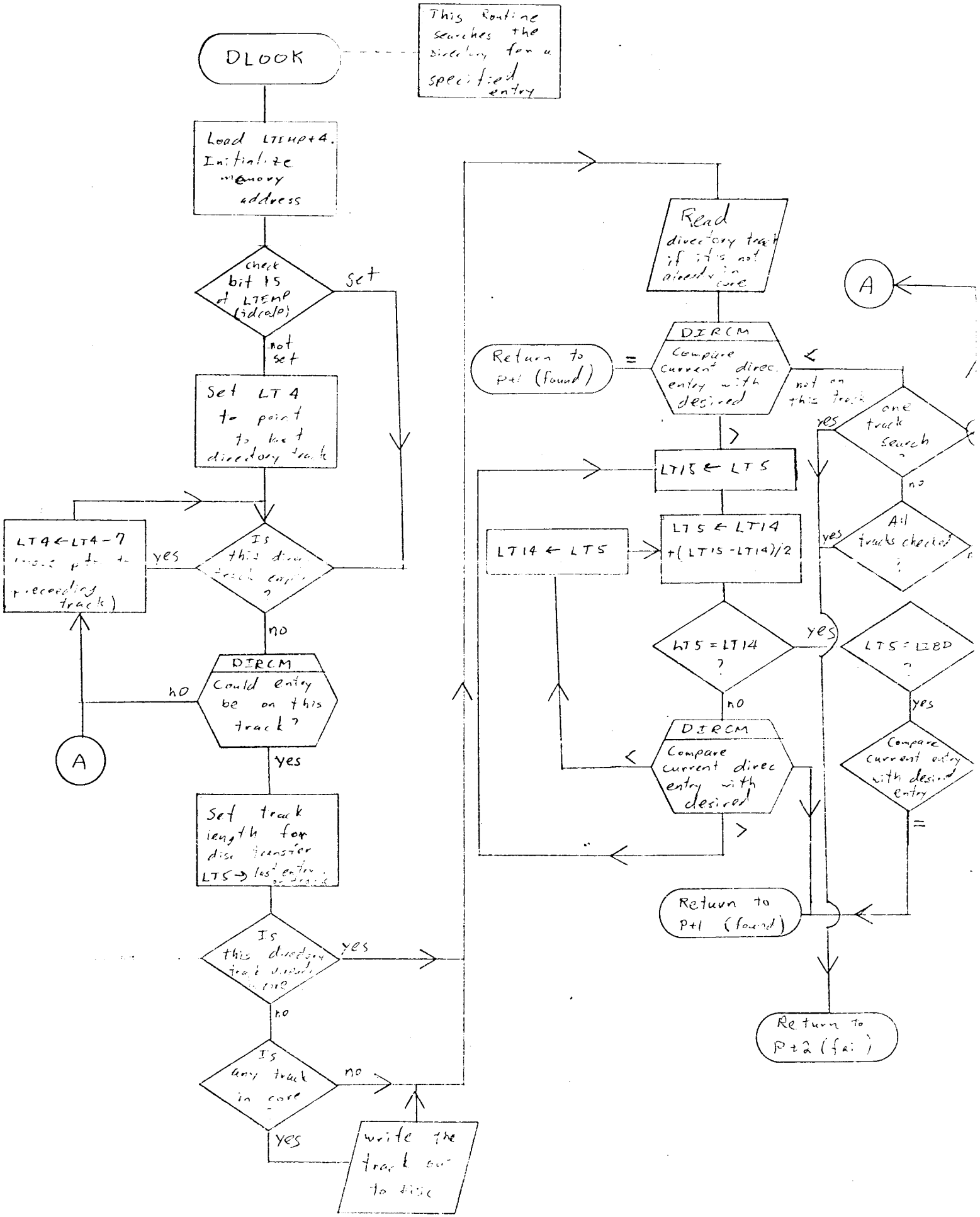
17. Read in the first entry on the first directory track and set the appropriate words in DIREC. Read in the first entry of the second directory track (if it is not empty) and set DIREC.

18. Check if warning message is being printed if so exit to LENDR, if not exit to LEND to print (CR)-(LF).

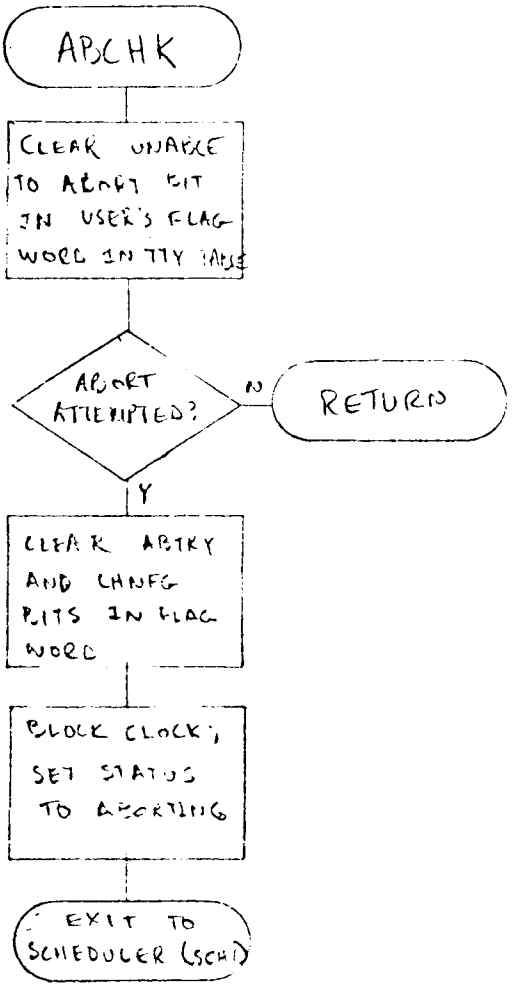
PHONES

The PHONES command is used to tell the I/O processor how long to allow the user to try to successfully log on before disconnecting him. It is originally assumed to be 120 seconds. It can be reset to from 1 to 255 seconds by the PHONES command.

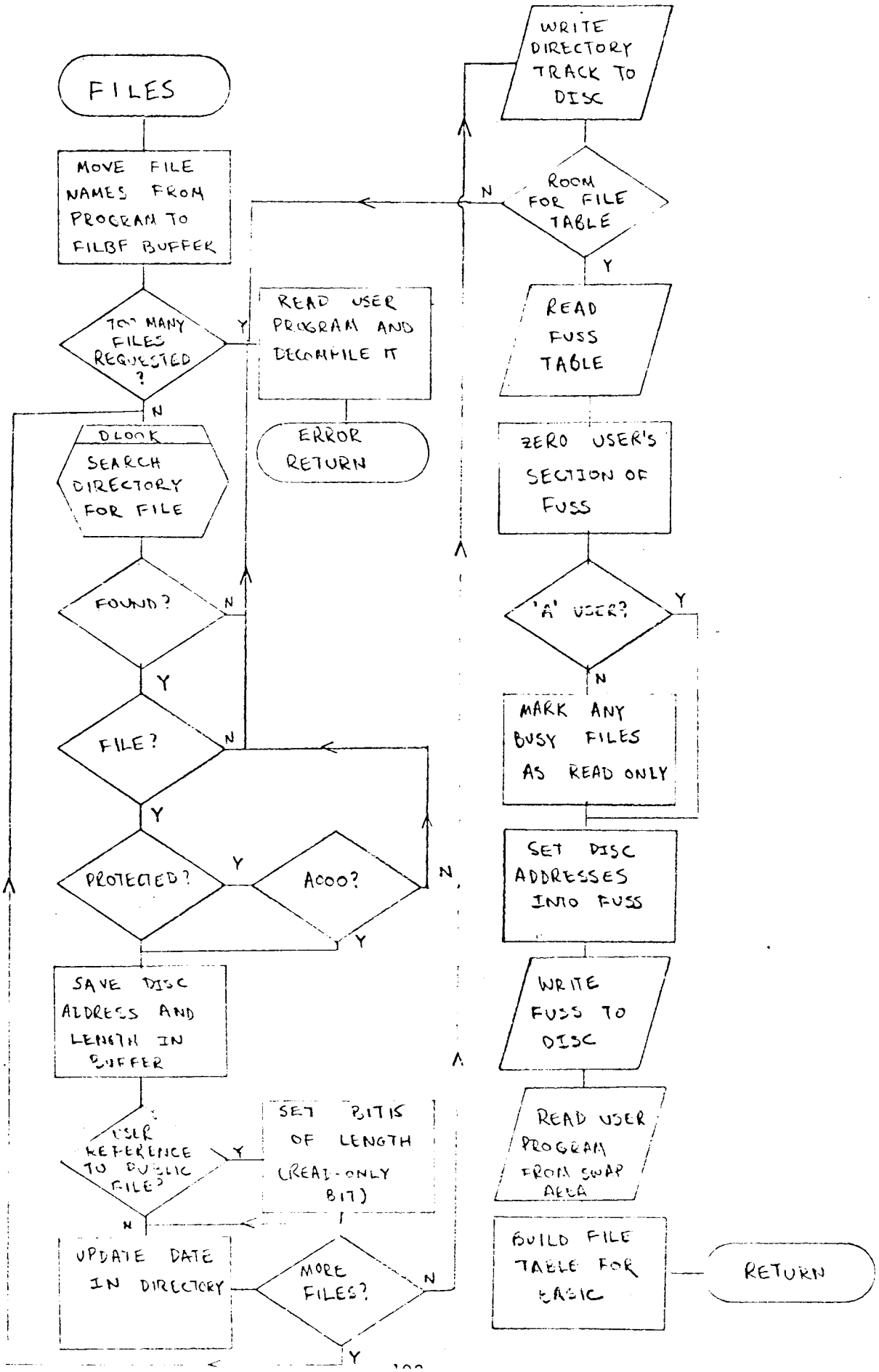
Directory Search Routine



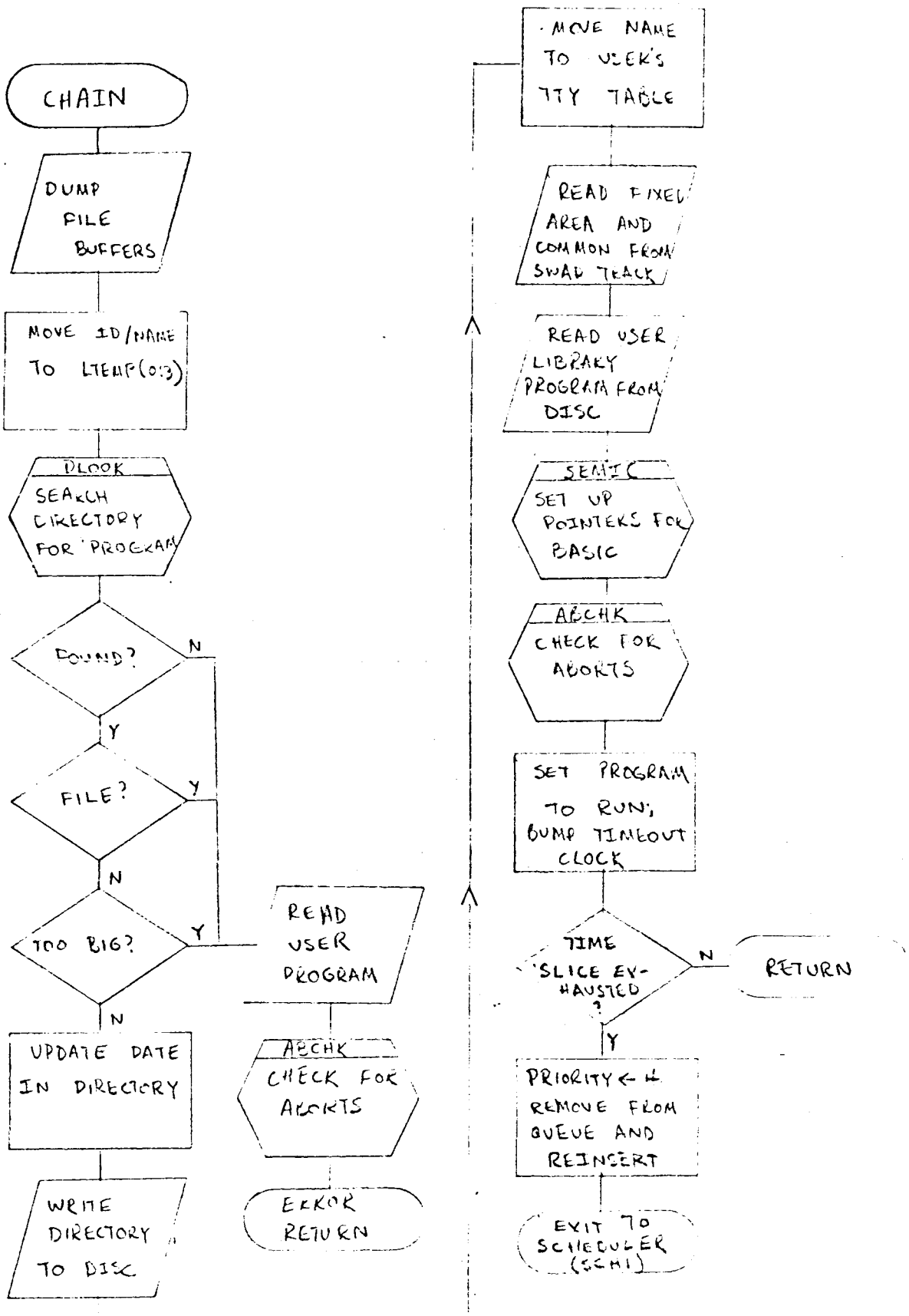
ABCHK


ABCHK
CLEAR UNABLE TO ABORT BIT IN USER'S FLAG WORD IN TTY TABLE
ABORT ATTEMPTED?
N
RETURN
Y
CLEAR ABTKY AND CHNFG BITS IN FLAG WORD
BLOCK CLOCK; SET STATUS TO ABORTING
EXIT TO SCHEDULER (SCHD)

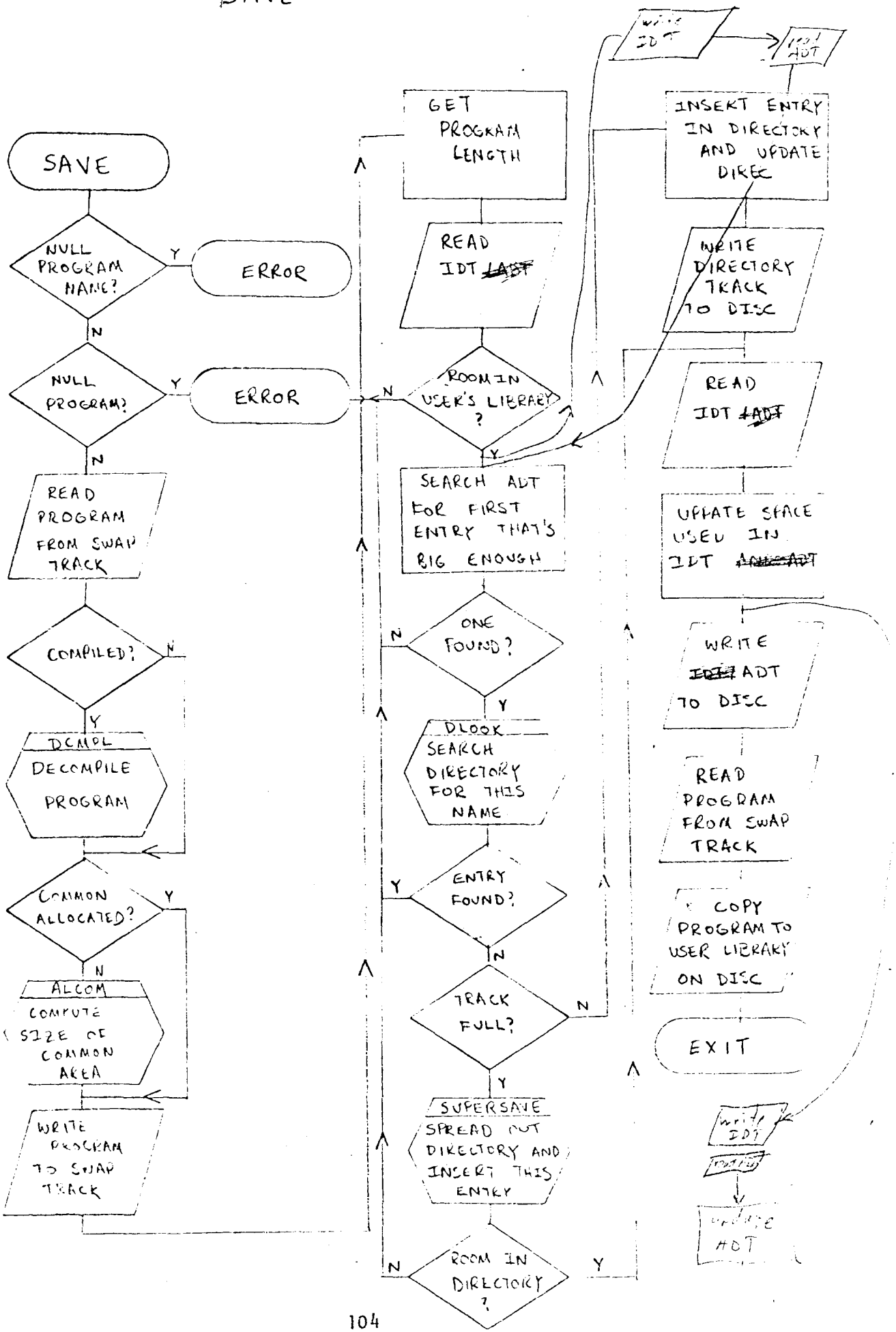
FILES


FILES
MOVE FILE NAMES FROM PROGRAM TO FILBF BUFFER
TOO MANY FILES REQUESTED ?
Y
N
READ USER PROGRAM AND DECOMPILE IT
ERROR RETURN
DLOOK
SEARCH DIRECTORY FOR FILE
FOUND?
N
Y
FILE?
N
Y
PROTECTED?
Y
ACOO?
N
Y
SAVE DISC ADDRESS AND LENGTH IN BUFFER
USER REFERENCE TO PUBLIC FILE?
Y
N
SET BITIS OF LENGTH (READ-ONLY BIT)
UPDATE DATE IN DIRECTORY
MORE FILES?
N
Y
WRITE DIRECTORY TRACK TO DISC
ROOM FOR FILE TABLE
N
Y
READ FUSS TABLE
ZERO USER'S SECTION OF FUSS
'A' USER?
Y
N
MARK ANY BUSY FILES AS READ ONLY
SET DISC ADDRESSES INTO FUSS
WRITE FUSS TO DISC
READ USER PROGRAM FROM SWAP AREA
BUILD FILE TABLE FOR BASIC
RETURN

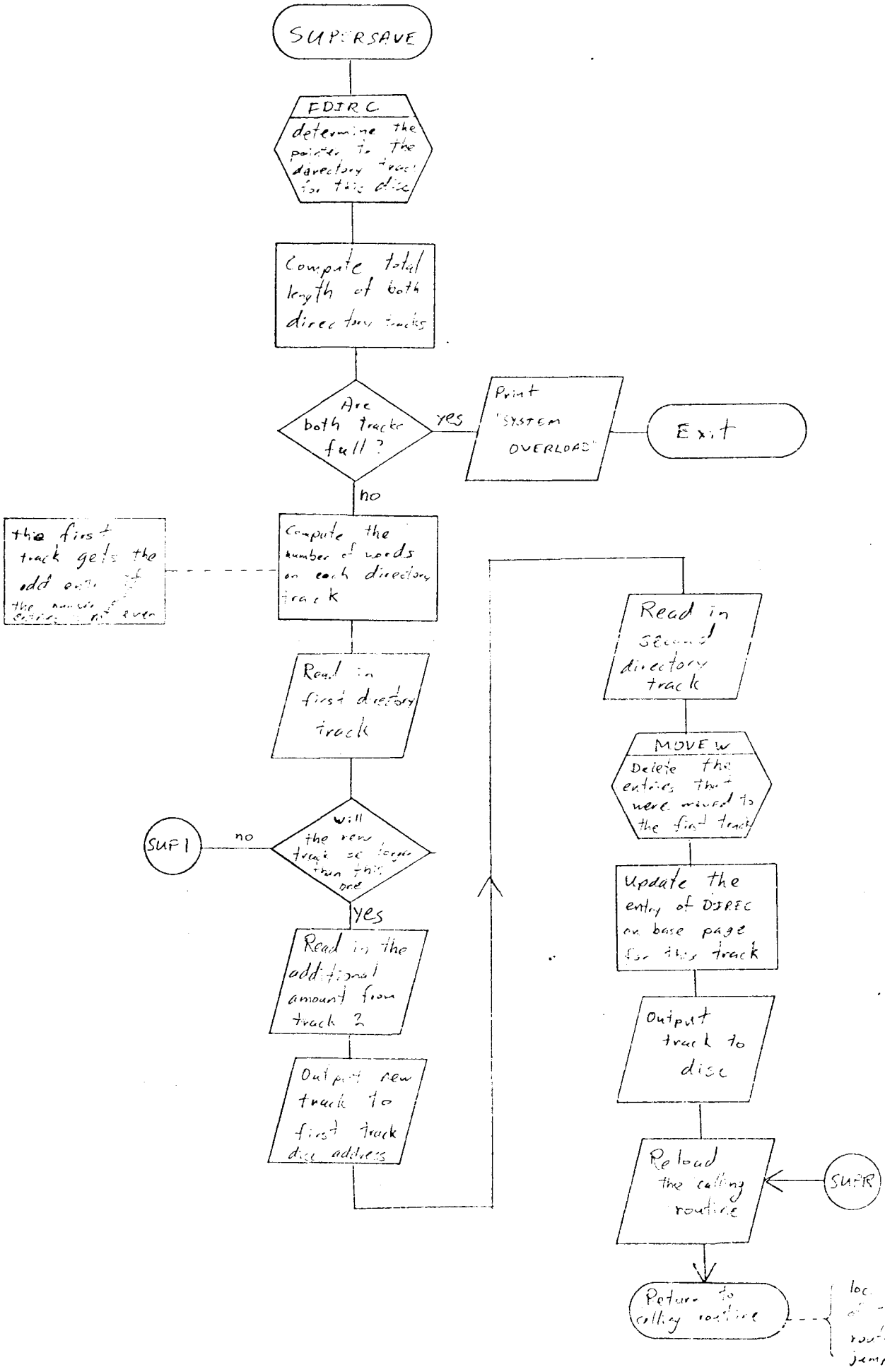
CHAIN

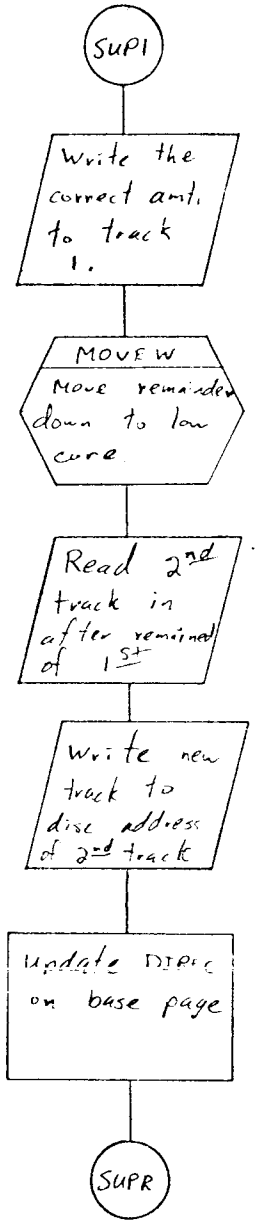

CHAIN
DUMP FILE BUFFERS
MOVE ID/NAME TO LTEMP(0:3)
DLOOK
SEARCH DIRECTORY FOR PROGRAM
FOUND?
N
Y
FILE?
Y
N
TOO BIG?
Y
N
UPDATE DATE IN DIRECTORY
WRITE DIRECTORY TO DISC
READ USER PROGRAM
ABCHK
CHECK FOR ABORTS
ERROR RETURN
MOVE NAME TO USER'S TTY TABLE
READ FIXED AREA AND COMMON FROM SWAP TRACK
READ USER LIBRARY PROGRAM FROM DISC
SEMIC
SET UP POINTERS FOR BASIC
ABCHK
CHECK FOR ABORTS
SET PROGRAM TO RUN; BUMP TIMEOUT CLOCK
TIME SLICE EXHAUSTED ?
N
RETURN
Y
PRIORITY ← 4
REMOVE FROM QUEUE AND REINSERT
EXIT TO SCHEDULER (SCHI)

SAVE

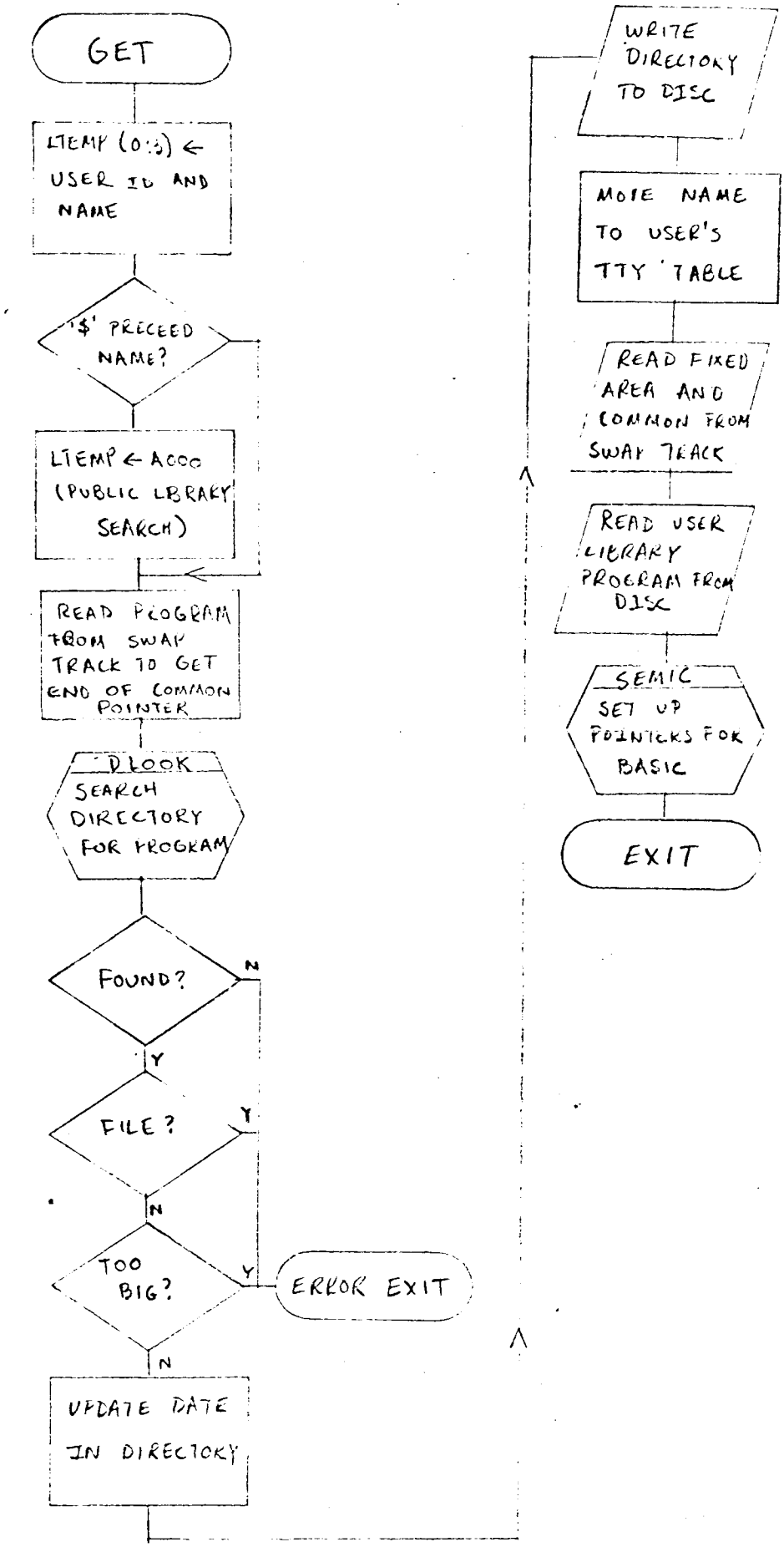

SAVE
NULL PROGRAM NAME?
Y
ERROR
N
NULL PROGRAM?
Y
ERROR
N
READ PROGRAM FROM SWAP TRACK
COMPILED?
N
Y
DCMPL
DECOMPILE PROGRAM
COMMON ALLOCATED?
Y
N
ALCOM
COMPUTE SIZE OF COMMON AREA
WRITE PROGRAM TO SWAP TRACK
GET PROGRAM LENGTH
READ IDT
ROOM IN USER'S LIBRARY ?
N
Y
SEARCH ADT FOR FIRST ENTRY THAT'S BIG ENOUGH
ONE FOUND?
N
Y
DLOOK
SEARCH DIRECTORY FOR THIS NAME
ENTRY FOUND?
Y
N
TRACK FULL?
N
Y
SUPERSAVE
SPREAD OUT DIRECTORY AND INSERT THIS ENTRY
ROOM IN DIRECTORY ?
N
Y
INSERT ENTRY IN DIRECTORY AND UPDATE DIREC
WRITE DIRECTORY TRACK TO DISC
READ IDT
UPDATE SPACE USED IN IDT
WRITE ADT TO DISC
READ PROGRAM FROM SWAP TRACK
COPY PROGRAM TO USER LIBRARY ON DISC
EXIT

SUPERSAVE

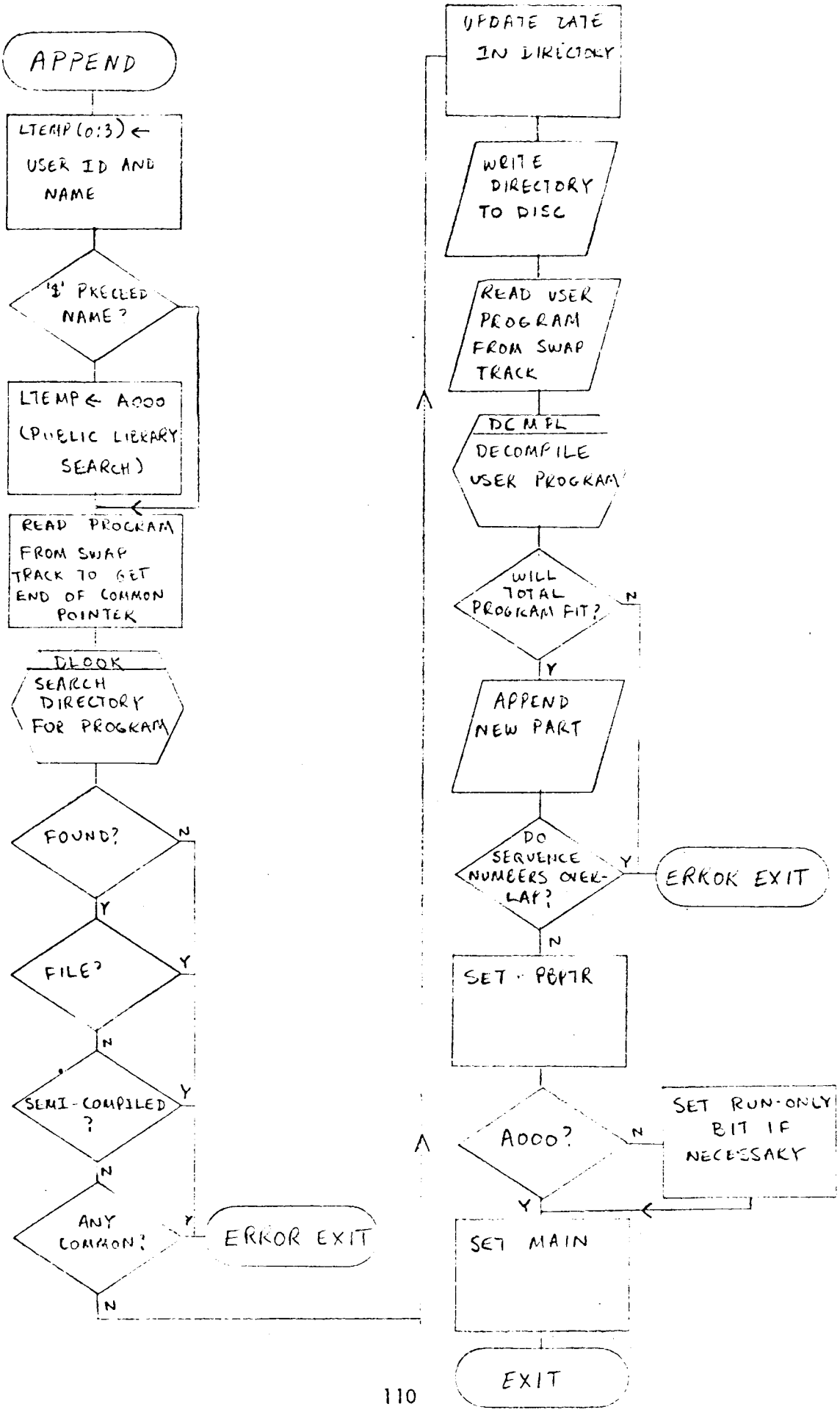

SUPERSAVE
EDIRC
determine the pointer to the directory track for this disc
Compute total length of both directory tracks
Are both tracks full?
yes
Print 'SYSTEM OVERLOAD'
Exit
no
Compute the number of words on each directory track
this first track gets the odd extra if the number of entries is not even
Read in first directory track
Will the new track be longer than this one
no
SUF1
yes
Read in the additional amount from track 2
Output new track to first track disc address
Read in second directory track
MOVEW
Delete the entries that were moved to the first track
Update the entry of DIREC on base page for this track
Output track to disc
Reload the calling routine
SUPR
Return to calling routine
loc 254 of the calling routine has a jump to the right location


SUP1
Write the correct amt. to track 1.
MOVE W
Move remainder down to low core
Read 2nd track in after remainder of 1st
Write new track to disc address of 2nd track
Update DIRC on base page
SUPR

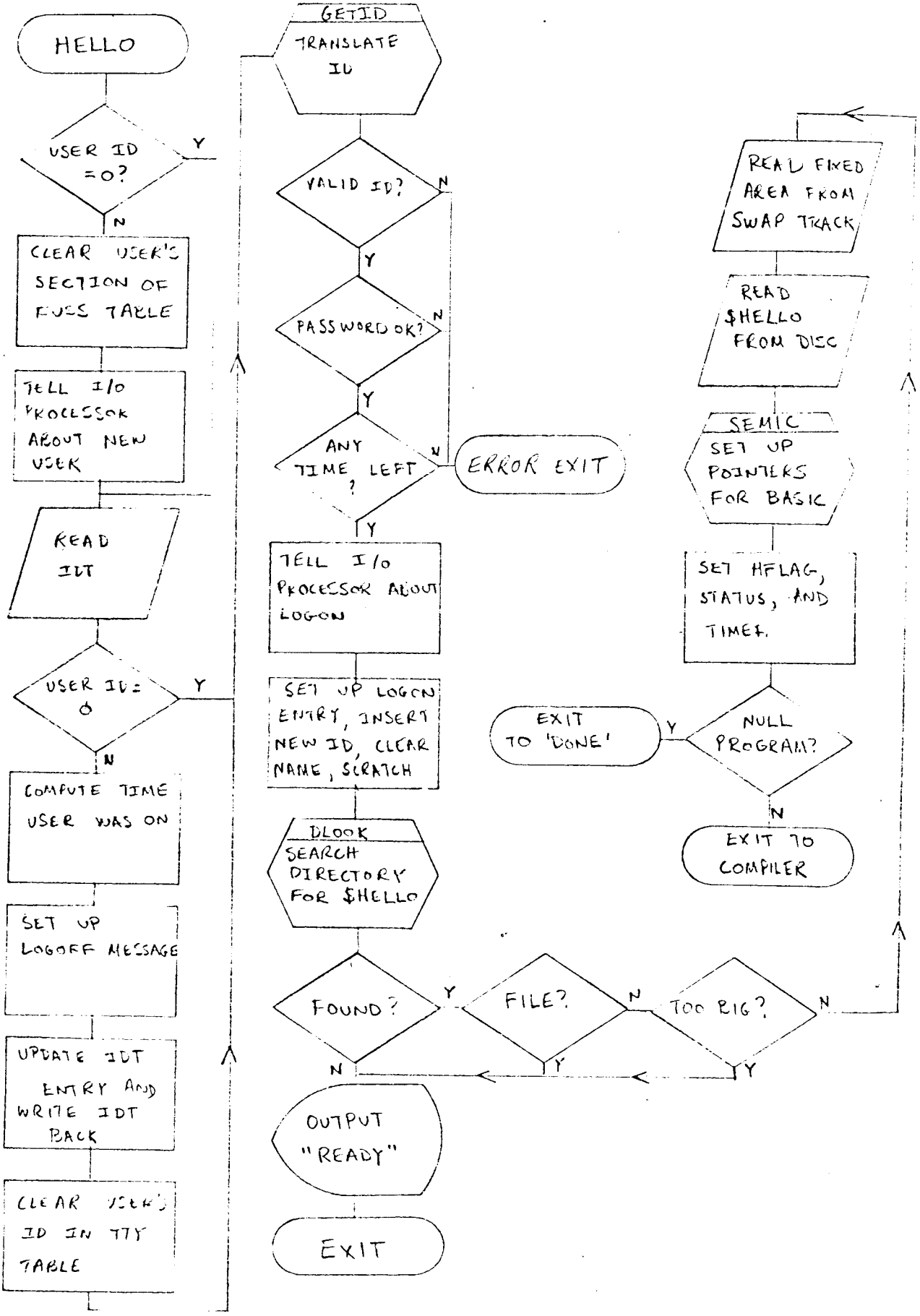
GET


GET
LTEMP (0:3) ← USER ID AND NAME
'$' PRECEED NAME?
LTEMP ← ACCC (PUBLIC LBRARY SEARCH)
READ PROGRAM FROM SWAP TRACK TO GET END OF COMMON POINTER
DLOOK SEARCH DIRECTORY FOR PROGRAM
FOUND?
N
Y
FILE?
Y
N
TOO BIG?
Y
ERROR EXIT
N
UPDATE DATE IN DIRECTORY
WRITE DIRECTORY TO DISC
MOVE NAME TO USER'S TTY TABLE
READ FIXED AREA AND COMMON FROM SWAP TRACK
READ USER LIBRARY PROGRAM FROM DISC
SEMIC SET UP POINTERS FOR BASIC
EXIT

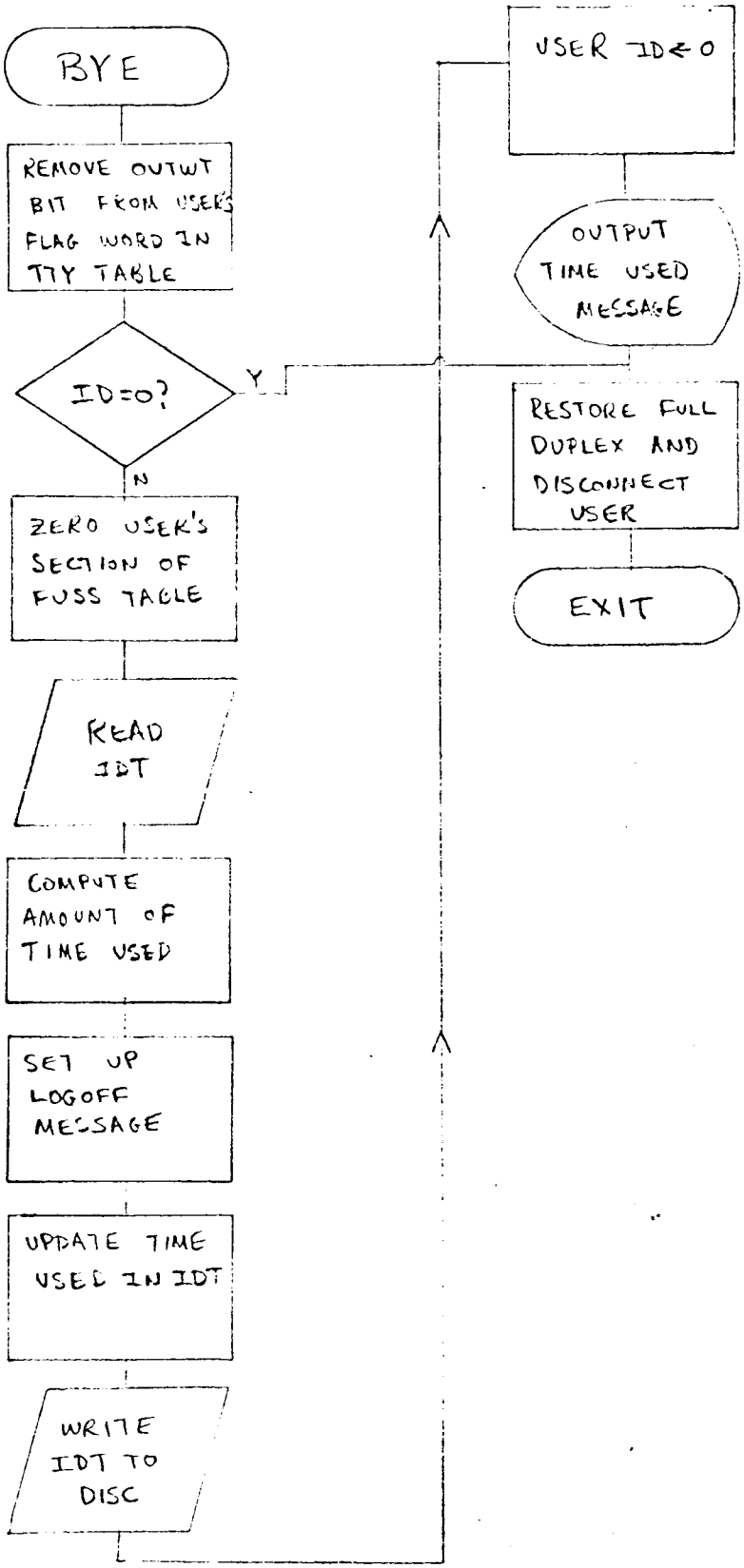
APPEND


APPEND
LTEMP(0:3) ← USER ID AND NAME
'$' PRECEED NAME?
LTEMP ← A000 (PUBLIC LIBRARY SEARCH)
READ PROGRAM FROM SWAP TRACK TO GET END OF COMMON POINTER
DLOOK SEARCH DIRECTORY FOR PROGRAM
FOUND?
FILE?
SEMI-COMPILED ?
ANY COMMON?
ERROR EXIT
UPDATE DATE IN DIRECTORY
WRITE DIRECTORY TO DISC
READ USER PROGRAM FROM SWAP TRACK
DCMPL DECOMPILE USER PROGRAM
WILL TOTAL PROGRAM FIT?
APPEND NEW PART
DO SERVENCE NUMBERS OVER-LAP?
ERROR EXIT
SET · PBPTR
A000?
SET RUN-ONLY BIT IF NECESSARY
SET MAIN
EXIT
N
Y

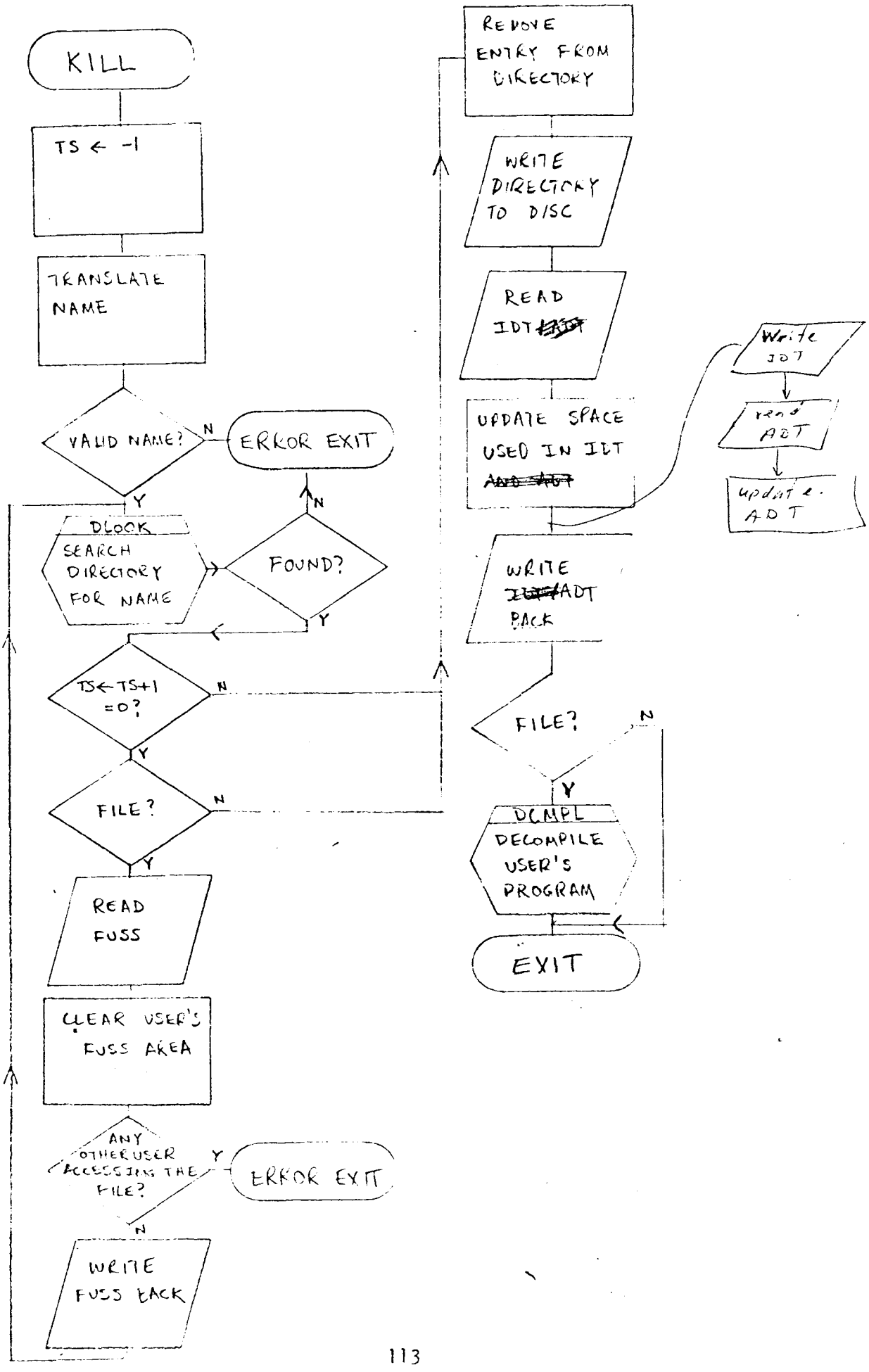
HELLO


HELLO
USER ID =0?
Y
N
CLEAR USER'S SECTION OF FUSS TABLE
TELL I/O PROCESSOR ABOUT NEW USER
READ IDT
USER ID = 0
Y
N
COMPUTE TIME USER WAS ON
SET UP LOGOFF MESSAGE
UPDATE IDT ENTRY AND WRITE IDT BACK
CLEAR USER'S ID IN TTY TABLE
GETID
TRANSLATE ID
VALID ID?
N
Y
PASSWORD OK?
N
Y
ANY TIME LEFT ?
N
ERROR EXIT
Y
TELL I/O PROCESSOR ABOUT LOGON
SET UP LOGON ENTRY, INSERT NEW ID, CLEAR NAME, SCRATCH
DLOOK
SEARCH DIRECTORY FOR $HELLO
FOUND?
Y
N
FILE?
N
Y
TOO BIG?
N
Y
OUTPUT "READY"
EXIT
READ FIXED AREA FROM SWAP TRACK
READ $HELLO FROM DISC
SEMIC
SET UP POINTERS FOR BASIC
SET HFLAG, STATUS, AND TIMER
NULL PROGRAM?
Y
N
EXIT TO 'DONE'
EXIT TO COMPILER

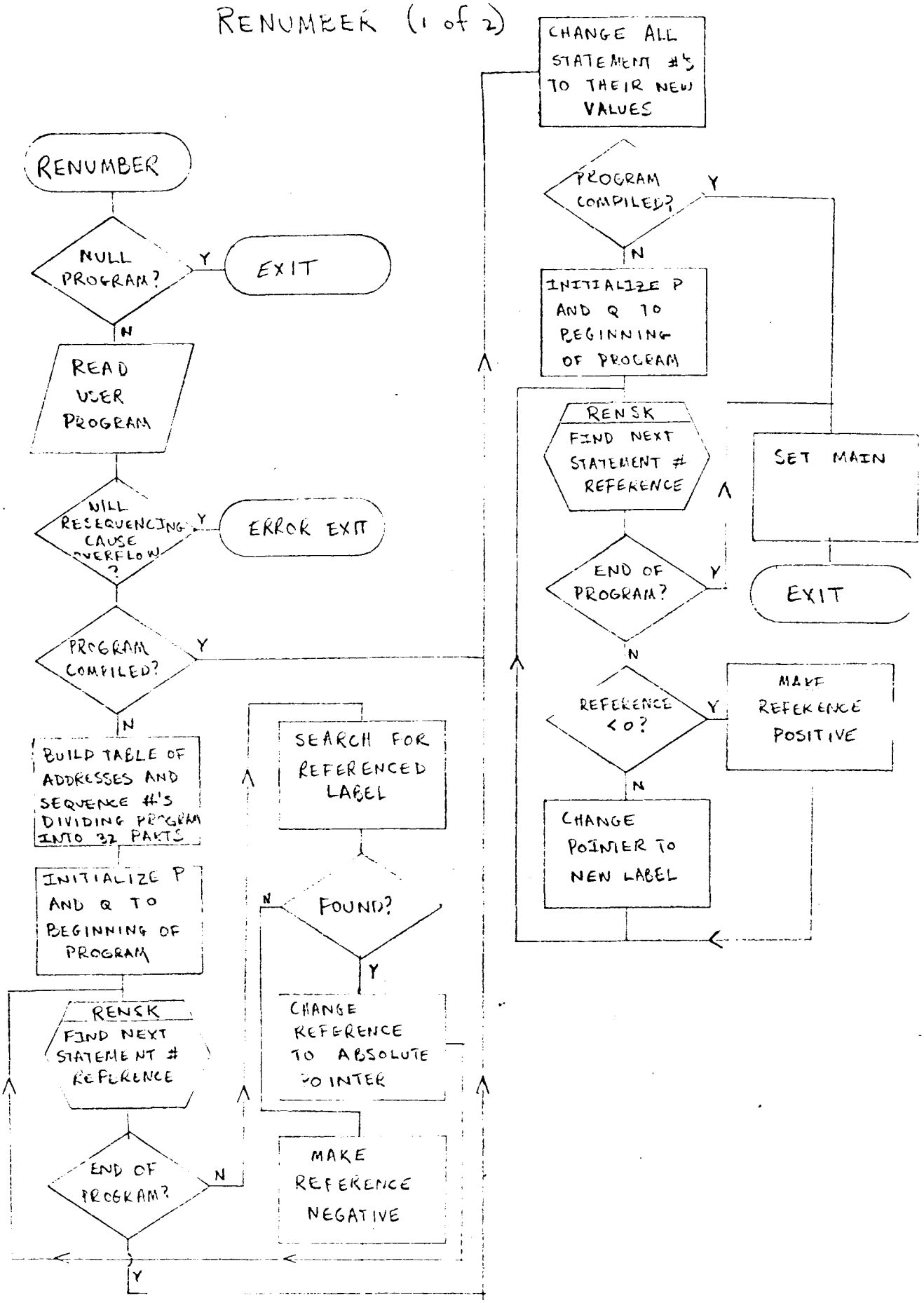
BYE


BYE
REMOVE OUTWT BIT FROM USER'S FLAG WORD IN TTY TABLE
ID=0?
Y
N
ZERO USER'S SECTION OF FUSS TABLE
READ IDT
COMPUTE AMOUNT OF TIME USED
SET UP LOGOFF MESSAGE
UPDATE TIME USED IN IDT
WRITE IDT TO DISC
USER ID←0
OUTPUT TIME USED MESSAGE
RESTORE FULL DUPLEX AND DISCONNECT USER
EXIT

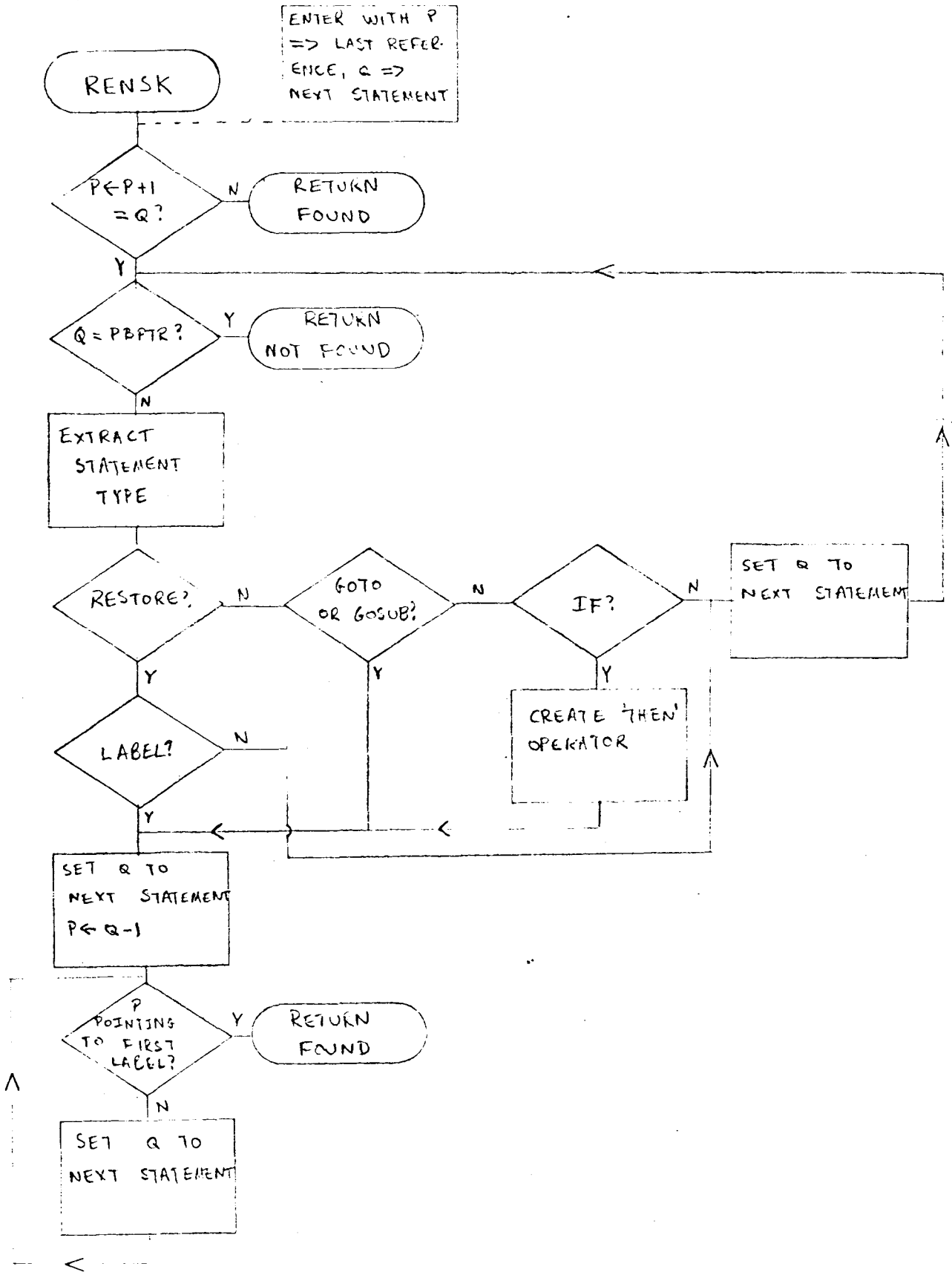
KILL

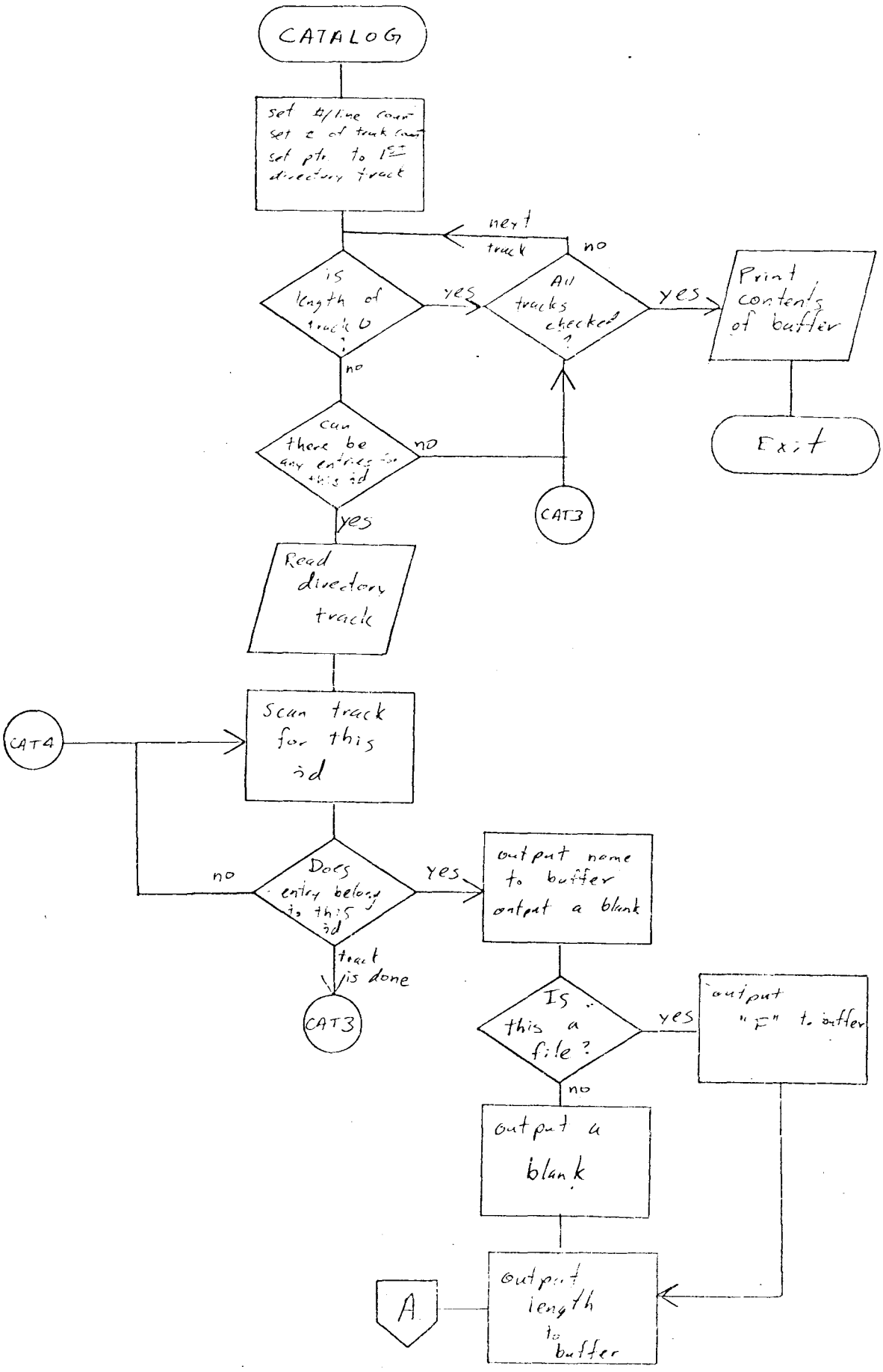

KILL
TS ← -1
TRANSLATE NAME
VALID NAME?
N
ERROR EXIT
Y
DLOOK
SEARCH DIRECTORY FOR NAME
FOUND?
N
Y
TS←TS+1 =0?
N
Y
FILE?
N
Y
READ FUSS
CLEAR USER'S FUSS AREA
ANY OTHER USER ACCESSING THE FILE?
Y
ERROR EXIT
N
WRITE FUSS BACK
REMOVE ENTRY FROM DIRECTORY
WRITE DIRECTORY TO DISC
READ IDT
UPDATE SPACE USED IN IDT
WRITE ADT BACK
FILE?
N
Y
DCMPL
DECOMPILE USER'S PROGRAM
EXIT
Write IDT
read ADT
update ADT

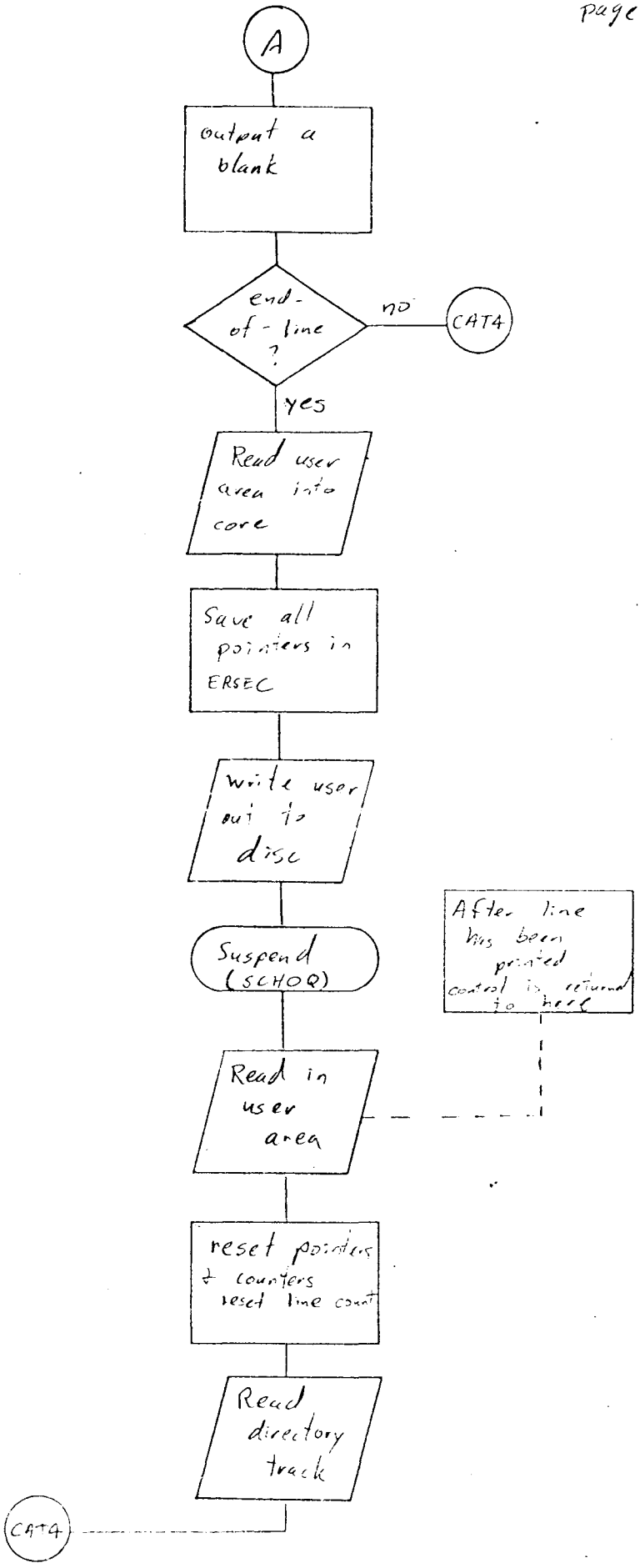
RENUMBER (1 of 2)

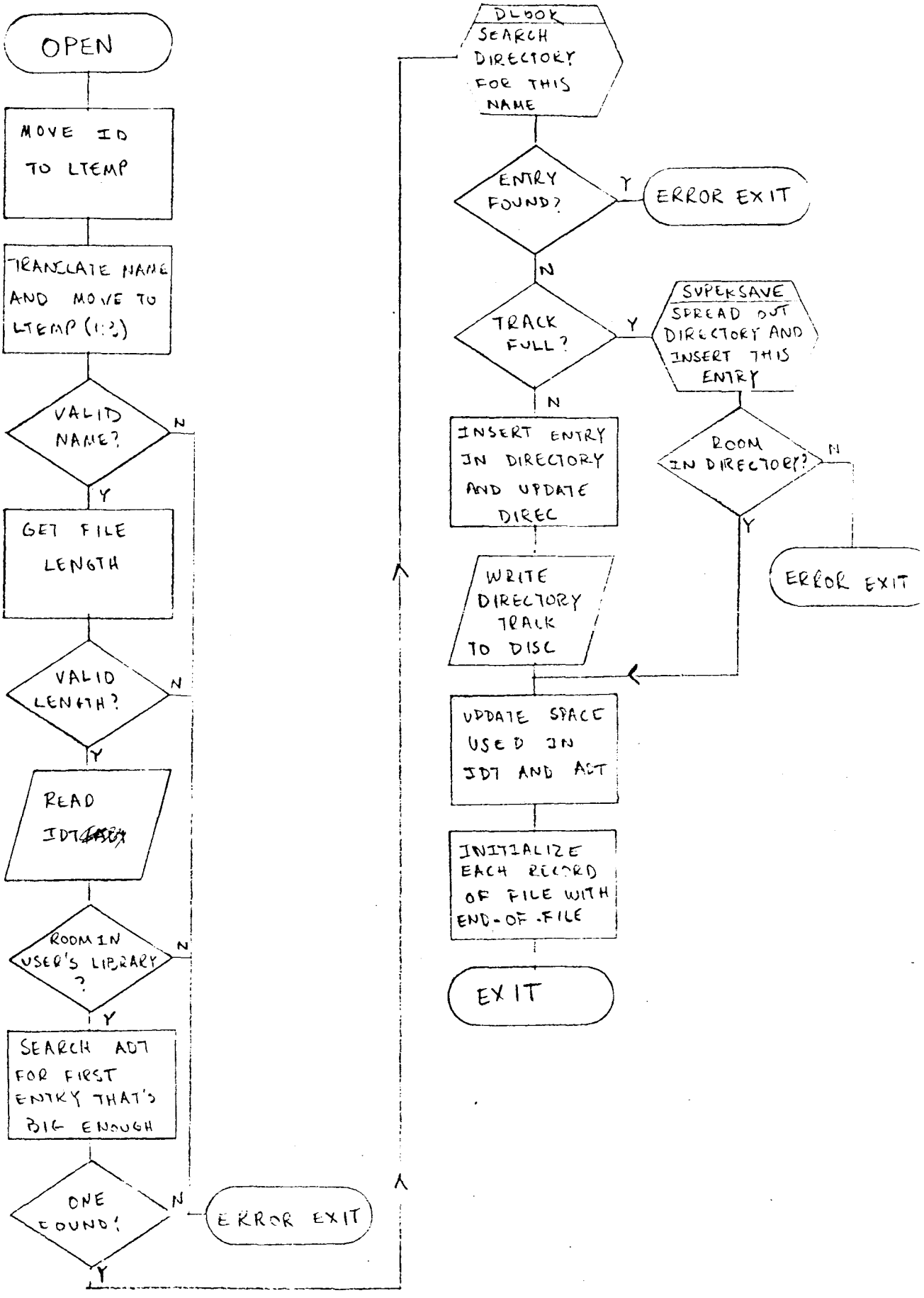


RENUMBER (2 of 2)

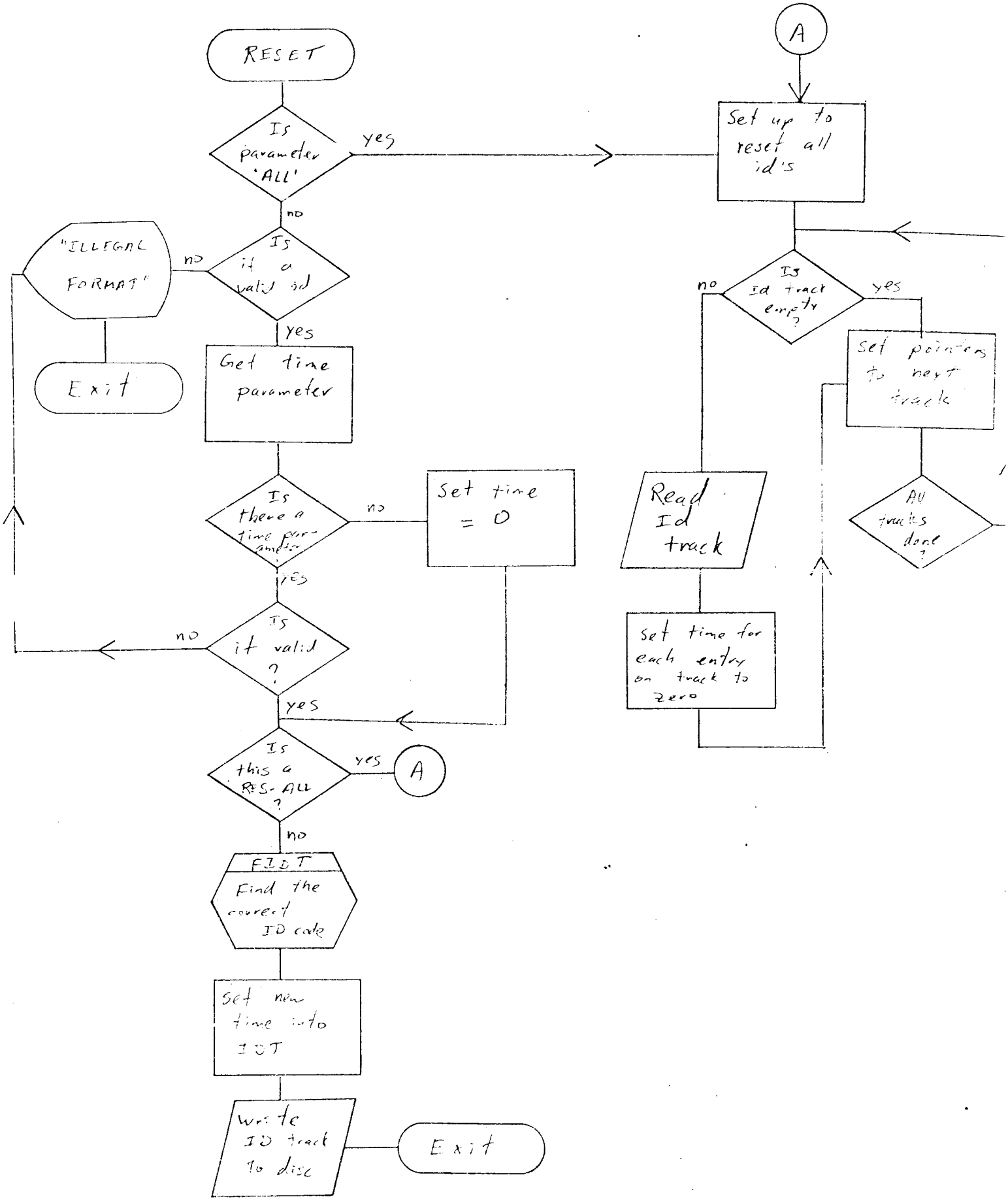



CATALOG
set t/line count
set c of track count
set ptr to 1st directory track
next track
no
is length of track 0 ?
yes
All tracks checked ?
yes
Print contents of buffer
Exit
no
can there be any entries for this id
no
CAT3
yes
Read directory track
CAT4
Scan track for this id
no
Does entry belong to this id
yes
output name to buffer
output a blank
track is done
CAT3
Is this a file ?
yes
output "F" to buffer
no
output a blank
A
output length to buffer

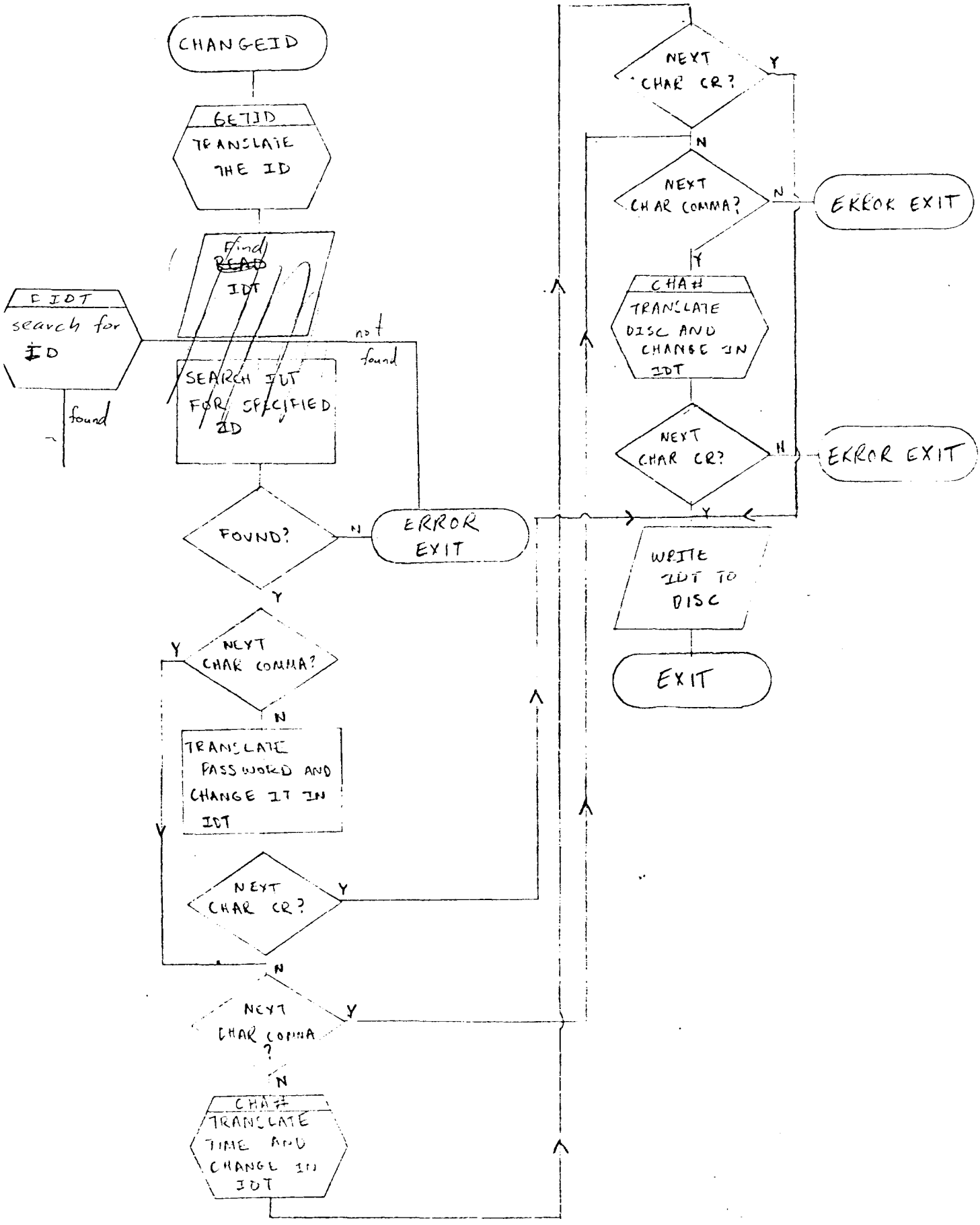

A
output a blank
end-of-line ?
no
CAT4
yes
Read user area into core
Save all pointers in ERSEC
write user out to disc
Suspend (SCHOQ)
After line has been printed control is returned to here
Read in user area
reset pointers & counters reset line count
Read directory track
CAT4


OPEN
MOVE ID TO LTEMP
TRANSLATE NAME AND MOVE TO LTEMP (1:)
VALID NAME?
N
Y
GET FILE LENGTH
VALID LENGTH?
N
Y
READ IDT
ROOM IN USER'S LIBRARY ?
N
Y
SEARCH ADT FOR FIRST ENTRY THAT'S BIG ENOUGH
ONE FOUND?
N
ERROR EXIT
Y
DLOOK
SEARCH DIRECTORY FOR THIS NAME
ENTRY FOUND?
Y
ERROR EXIT
N
TRACK FULL?
Y
SUPERSAVE
SPREAD OUT DIRECTORY AND INSERT THIS ENTRY
N
INSERT ENTRY IN DIRECTORY AND UPDATE DIREC
ROOM IN DIRECTORY?
N
ERROR EXIT
Y
WRITE DIRECTORY TRACK TO DISC
UPDATE SPACE USED IN IDT AND ADT
INITIALIZE EACH RECORD OF FILE WITH END-OF-FILE
EXIT

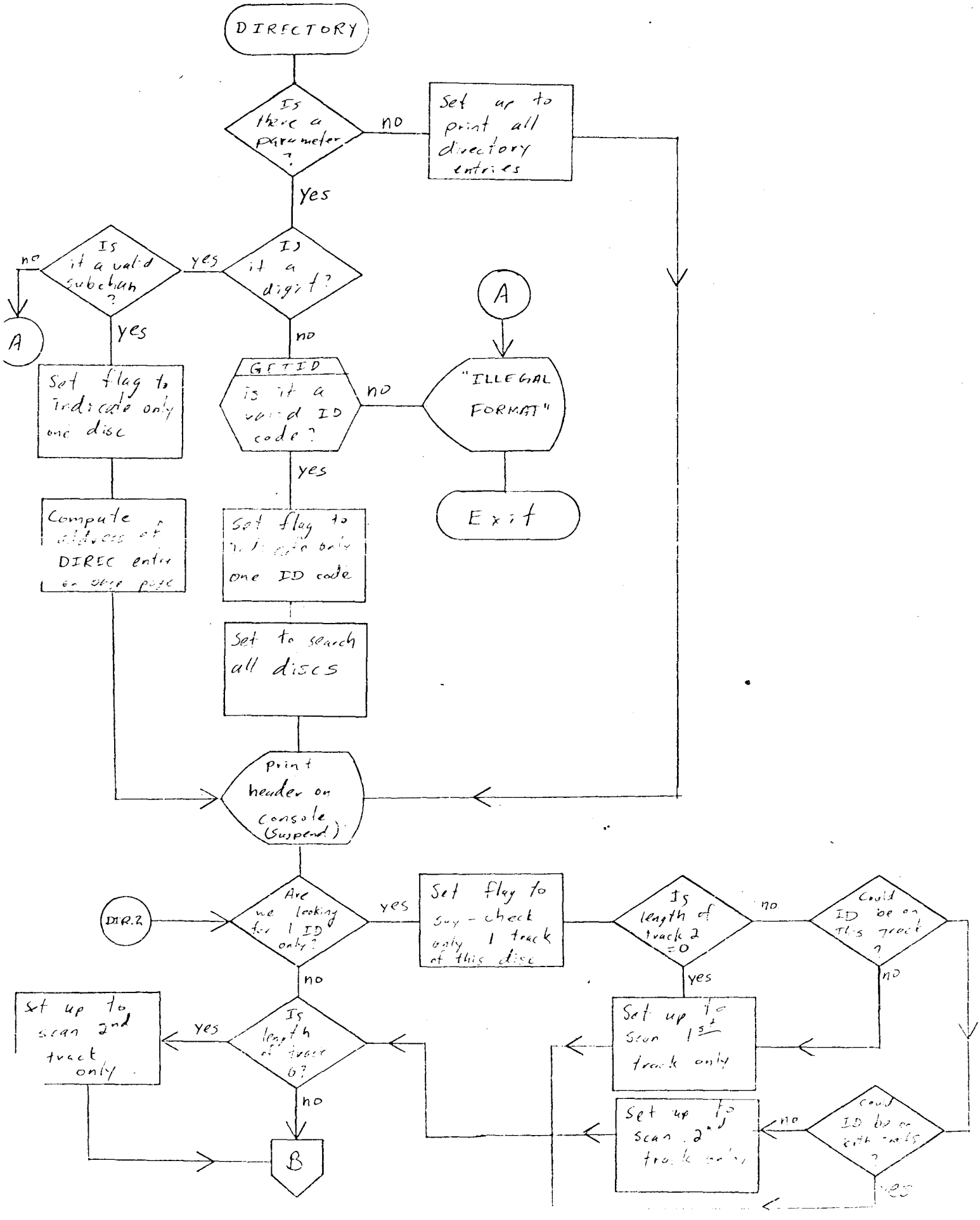
RESET

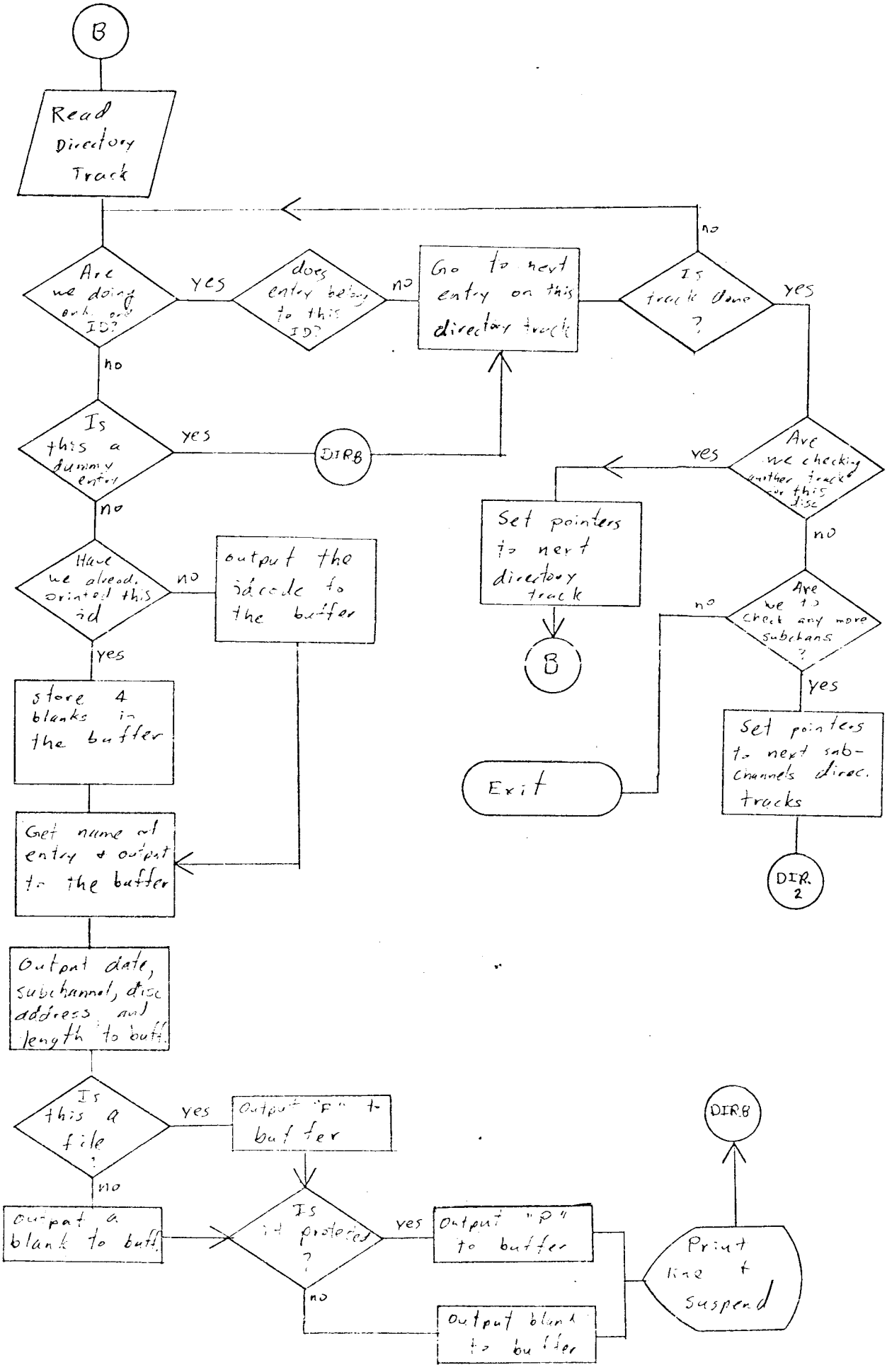

RESET
Is parameter 'ALL'
yes
no
"ILLEGAL FORMAT"
no
Is it a valid id
yes
Exit
Get time parameter
Is there a time parameter
no
Set time = 0
yes
Is it valid ?
no
yes
Is this a RES-ALL ?
yes
A
no
FIDT
Find the correct ID code
Set new time into IDT
write ID track to disc
Exit
A
Set up to reset all id's
Is id track empty ?
no
yes
Read Id track
Set time for each entry on track to zero
Set pointers to next track
All tracks done ?

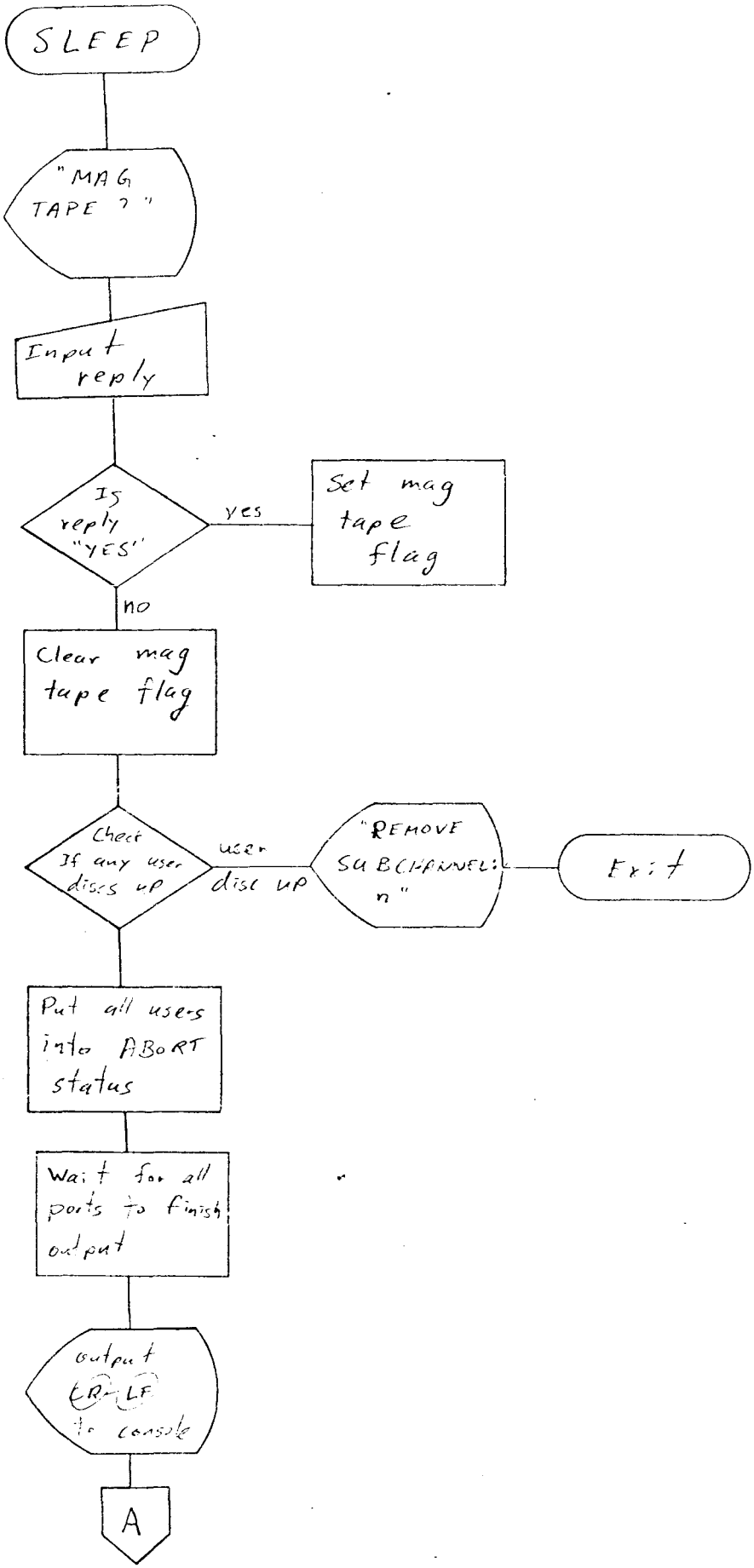
CHANGEID

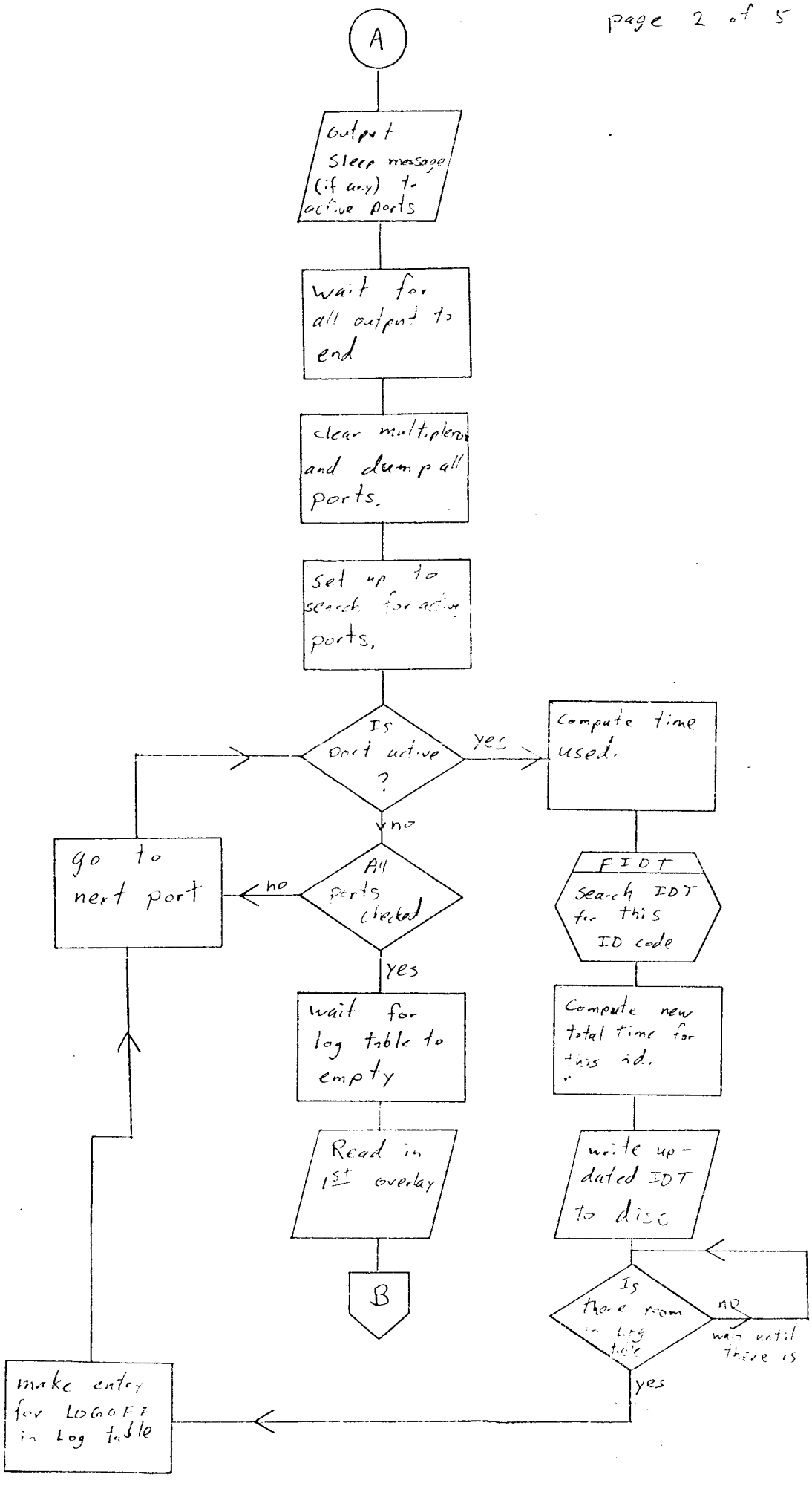

CHANGEID
GETID
TRANSLATE THE ID
find
READ
IDT
EIDT
search for ID
not found
found
SEARCH IDT FOR SPECIFIED ID
FOUND?
N
ERROR EXIT
Y
Y
NEXT CHAR COMMA?
N
TRANSLATE PASSWORD AND CHANGE IT IN IDT
NEXT CHAR CR?
Y
N
NEXT CHAR COMMA ?
Y
N
CHA#
TRANSLATE TIME AND CHANGE IN IDT
NEXT CHAR CR?
Y
N
ERROR EXIT
NEXT CHAR COMMA?
Y
N
ERROR EXIT
CHA#
TRANSLATE DISC AND CHANGE IN IDT
NEXT CHAR CR?
Y
N
ERROR EXIT
WRITE IDT TO DISC
EXIT

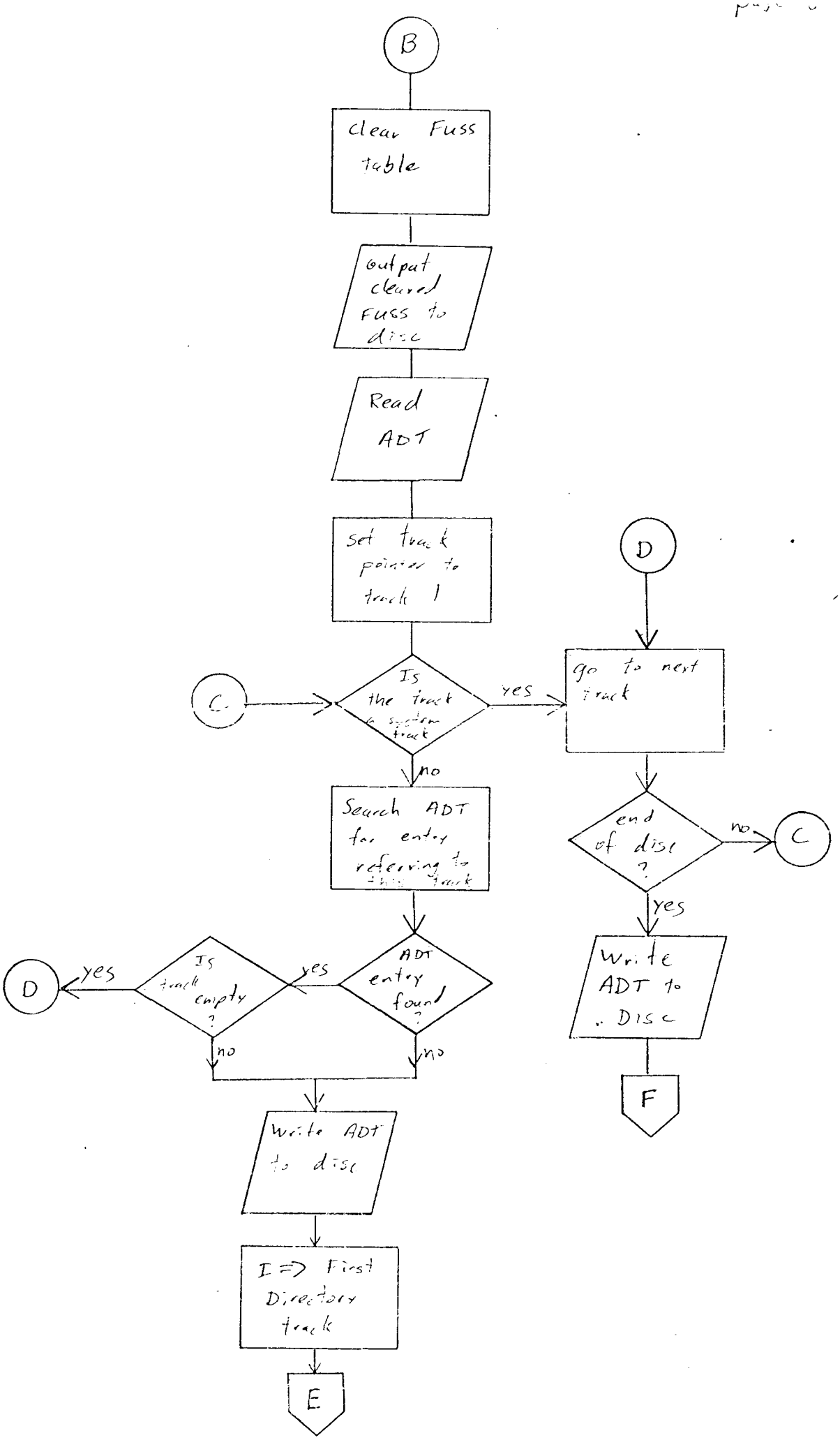
DIRECTORY

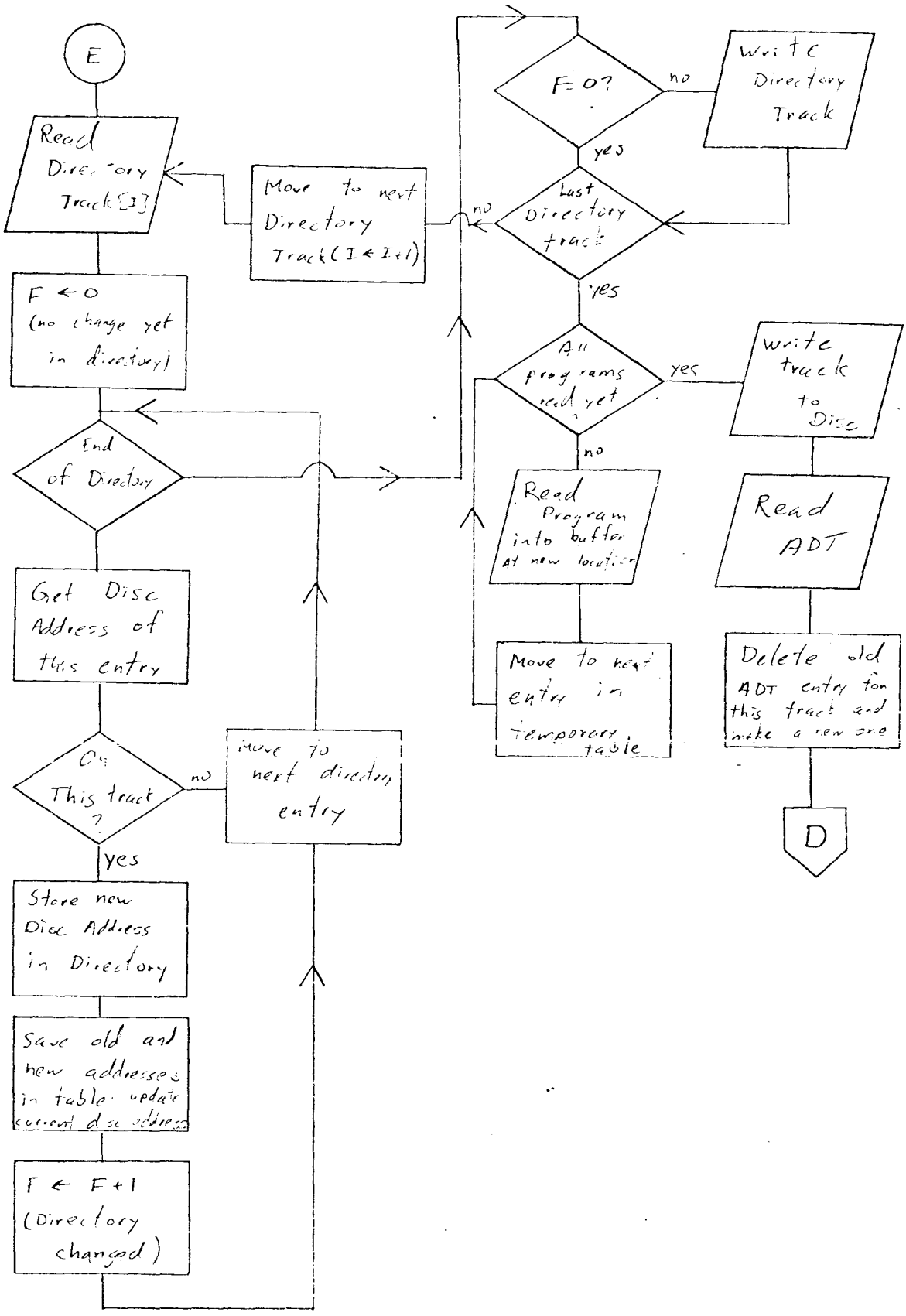

DIRECTORY
Is there a parameter?
no
Set up to print all directory entries
yes
Is it a valid subchan?
yes
Is it a digit?
no
yes
A
A
Set flag to indicate only one disc
GETID
Is it a valid ID code?
no
"ILLEGAL FORMAT"
yes
Exit
Compute address of DIREC entry on other page
Set flag to indicate only one ID code
Set to search all discs
Print header on console (Suspend)
DIR.2
Are we looking for 1 ID only?
yes
Set flag to say - check only 1 track of this disc
Is length of track 2 = 0
no
Could ID be on this track?
yes
no
Set up to scan 1st track only
Set up to scan 2nd track only
Is length of track 0?
yes
no
Could ID be on 4th track?
no
yes
B

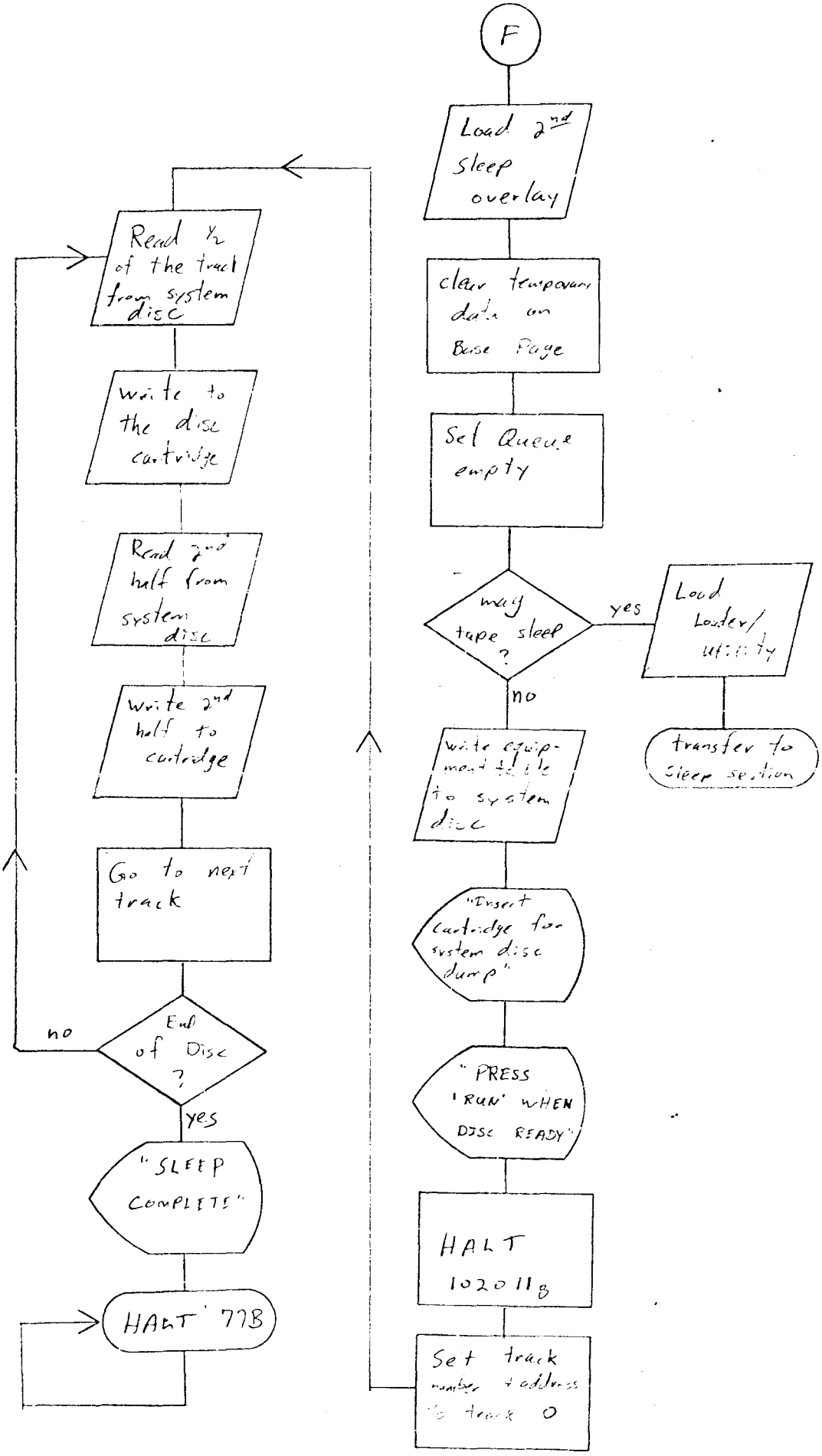

B
Read Directory Track
Are we doing anything with ID?
yes
does entry belong to this ID?
no
Go to next entry on this directory track
Is track done?
yes
no
no
Is this a dummy entry
yes
DIRB
no
Have we already printed this id
no
output the id code to the buffer
yes
store 4 blanks in the buffer
Get name of entry & output to the buffer
Output date, subchannel, disc address and length to buff.
Is this a file?
yes
Output "F" to buffer
no
output a blank to buff.
Is it protected?
yes
Output "P" to buffer
no
Output blank to buffer
Print line & suspend
DIRB
Are we checking another track on this disc
yes
Set pointers to next directory track
B
no
Are we to check any more subchans?
no
Exit
yes
Set pointers to next sub-channels direc. tracks
DIR. 2


SLEEP
"MAG TAPE ?"
Input reply
Is reply "YES"
yes
Set mag tape flag
no
Clear mag tape flag
Check If any user disks up
user disk up
"REMOVE SUBCHANNEL: n"
Exit
Put all users into ABORT status
Wait for all ports to finish output
output CR-LF to console
A

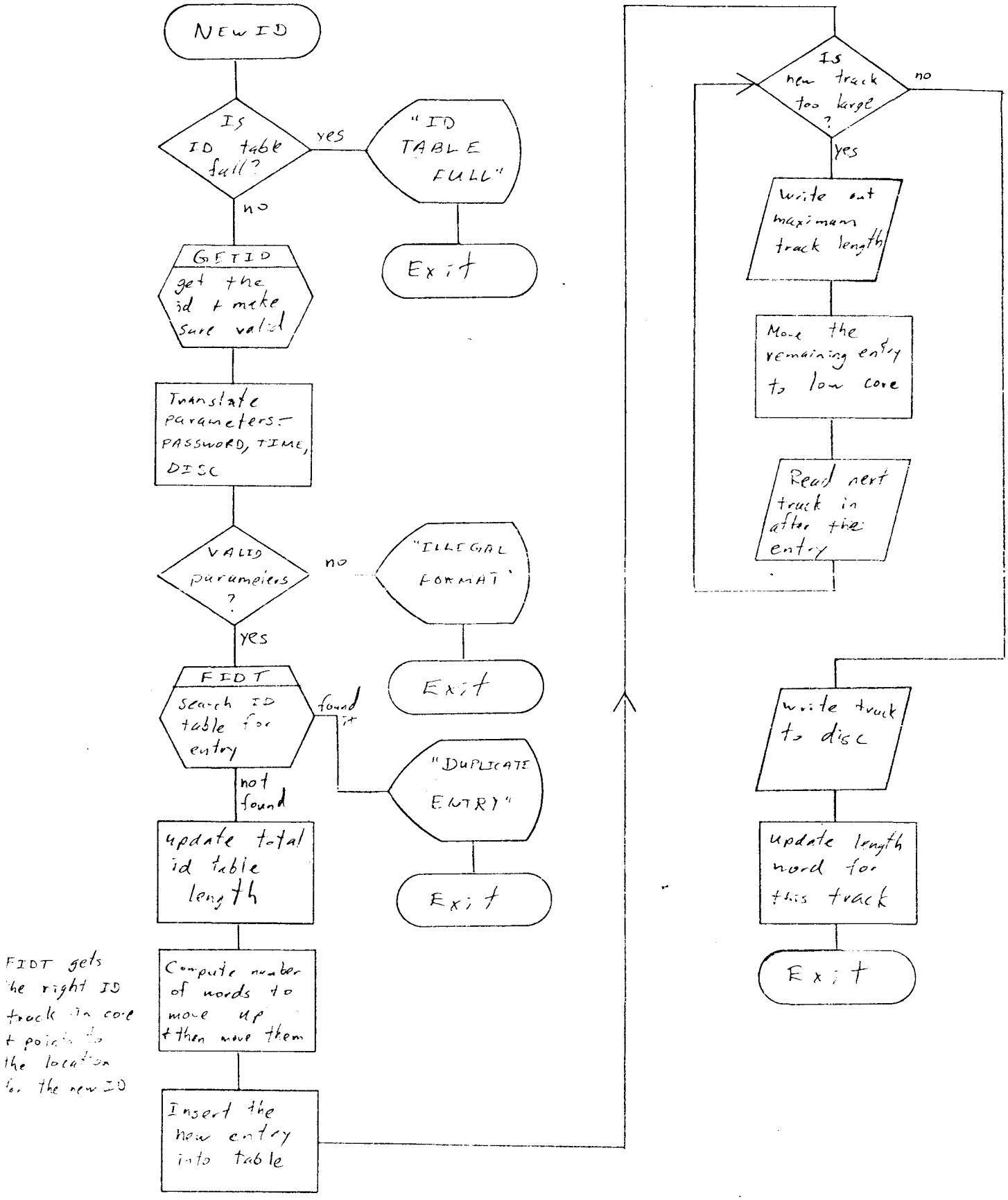

A
Output Sleep message (if any) to active ports
wait for all output to end
clear multiplexer and dump all ports.
set up to search for active ports.
Is port active ?
yes
Compute time used.
no
go to next port
no
All ports checked
yes
FIDT
Search IDT for this ID code
wait for log table to empty
Compute new total time for this id.
Read in 1st overlay
write updated IDT to disc
B
Is there room in Log table
no
wait until there is
yes
make entry for LOGOFF in Log table


B
clear Fuss table
output cleared Fuss to disc
Read ADT
set track pointer to track 1
C
Is the track a system track
yes
no
D
go to next track
end of disc ?
no
C
yes
Write ADT to Disc
F
Search ADT for entry referring to this track
ADT entry found ?
yes
Is track empty ?
yes
D
no
no
Write ADT to disc
I => First Directory track
E

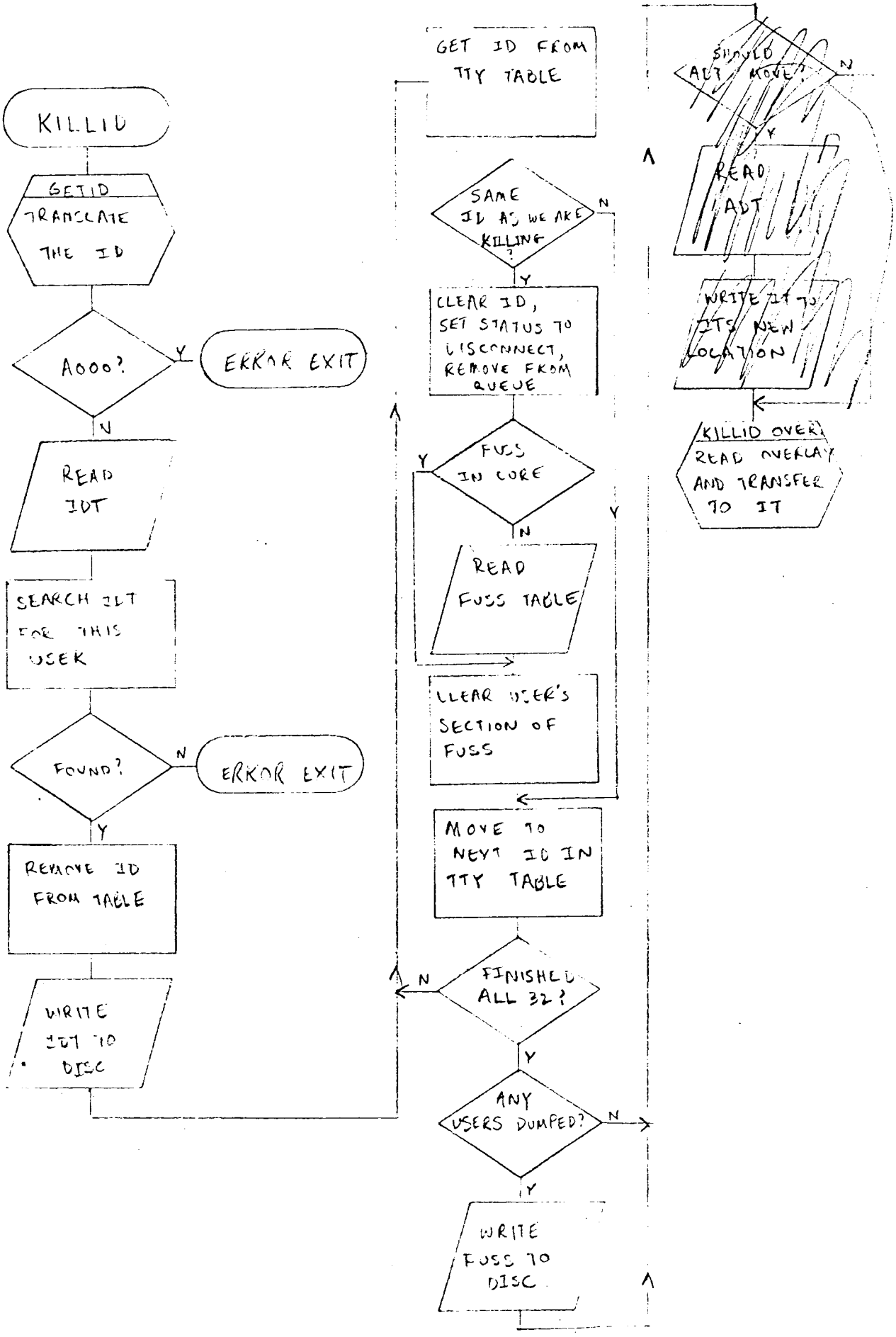

E
Read Directory Track [I]
F ← 0 (no change yet in directory)
End of Directory
Get Disc Address of this entry
On This track ?
no
move to next directory entry
yes
Store new Disc Address in Directory
Save old and new addresses in table update current disc address
F ← F+1 (Directory changed)
Move to next Directory Track (I ← I+1)
F=0?
no
write Directory Track
yes
Last Directory track
no
yes
All programs read yet ?
yes
write track to Disc
no
Read Program into buffer At new location
Move to next entry in temporary table
Read ADT
Delete old ADT entry for this tract and make a new one
D


F
Load 2nd sleep overlay
clear temporary data on Base Page
Set Queue empty
may tape sleep ?
yes
Load Loader/ utility
transfer to sleep section
no
write equip-ment table to system disc
"Insert cartridge for system disc dump"
"PRESS 'RUN' WHEN DISC READY"
HALT 102011B
Set track number address to track 0
Read ½ of the track from system disc
write to the disc cartridge
Read 2nd half from system disc
write 2nd half to cartridge
Go to next track
End of Disc ?
no
yes
"SLEEP COMPLETE"
HALT '77B

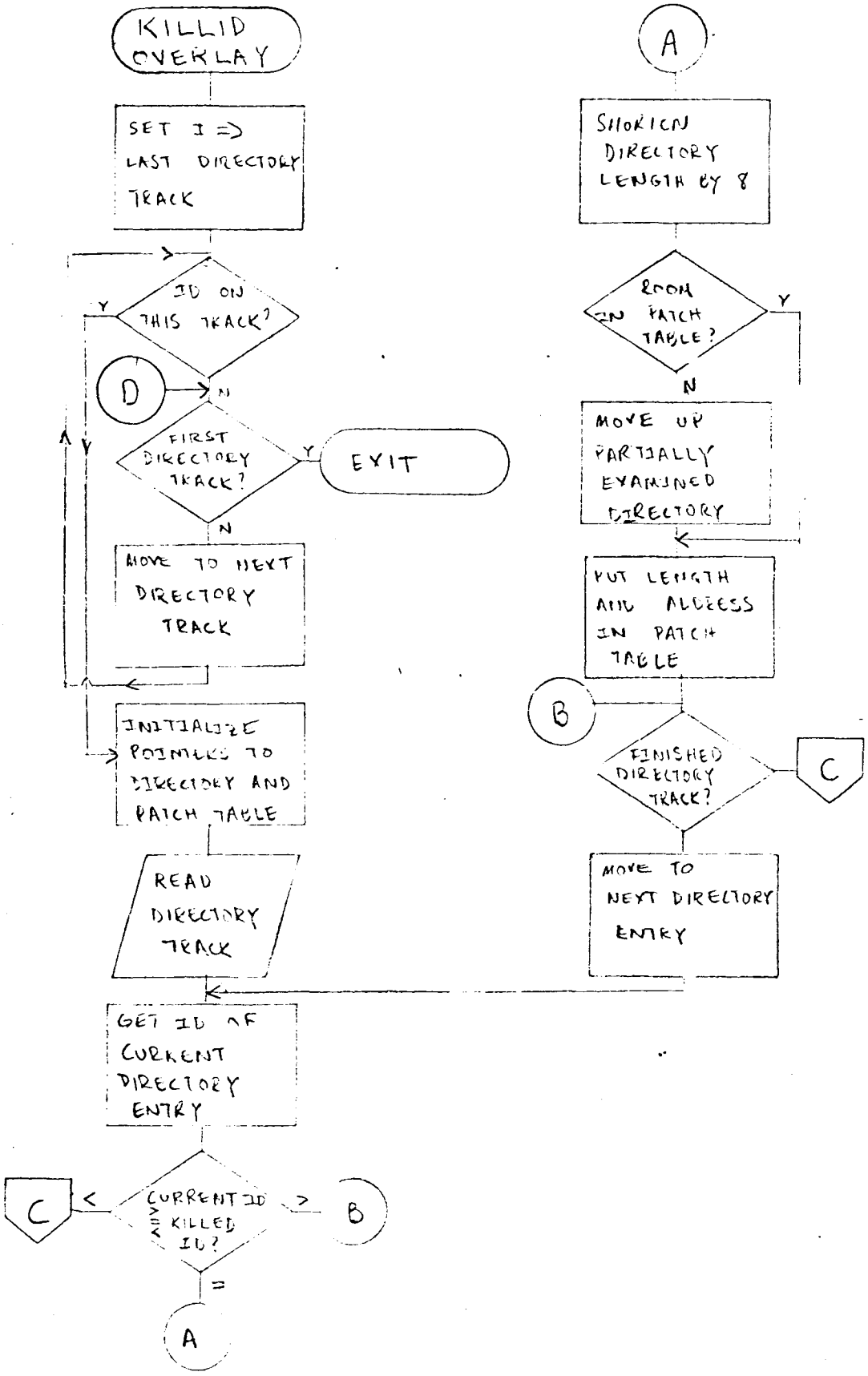
NEW ID


NEW ID
Is ID table full?
yes
"ID TABLE FULL"
Exit
no
GETID
get the id + make sure valid
Translate parameters: PASSWORD, TIME, DISC
VALID parameters ?
no
"ILLEGAL FORMAT"
Exit
yes
FIDT
search ID table for entry
found it
"DUPLICATE ENTRY"
Exit
not found
update total id table length
Compute number of words to move up + then move them
Insert the new entry into table
FIDT gets the right ID track in core + points to the location for the new ID
Is new track too large ?
no
yes
write out maximum track length
Move the remaining entry to low core
Read next track in after the entry
write track to disc
update length word for this track
Exit

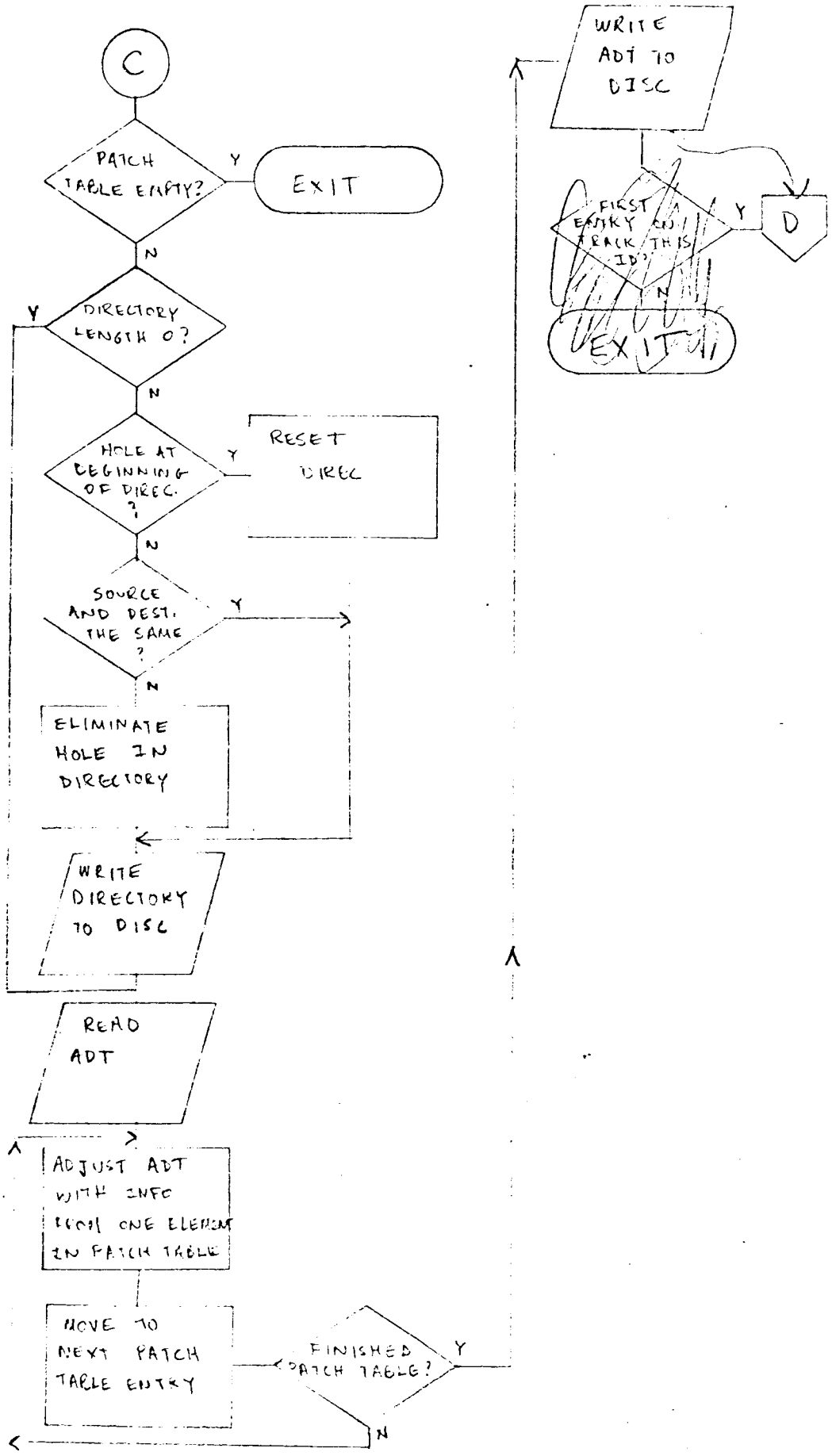
KILLID


KILLID
GETID
TRANSLATE THE ID
A000?
Y
ERROR EXIT
N
READ IDT
SEARCH IDT FOR THIS USER
FOUND?
N
ERROR EXIT
Y
REMOVE ID FROM TABLE
WRITE IDT TO DISC
GET ID FROM TTY TABLE
SAME ID AS WE ARE KILLING?
N
Y
CLEAR ID, SET STATUS TO DISCONNECT, REMOVE FROM QUEUE
FUSS IN CORE
Y
N
READ FUSS TABLE
CLEAR USER'S SECTION OF FUSS
MOVE TO NEXT ID IN TTY TABLE
FINISHED ALL 32?
N
Y
ANY USERS DUMPED?
N
Y
WRITE FUSS TO DISC
SHOULD ADT MOVE?
N
Y
READ ADT
WRITE IT TO ITS NEW LOCATION
KILLID OVER
READ OVERLAY AND TRANSFER TO IT

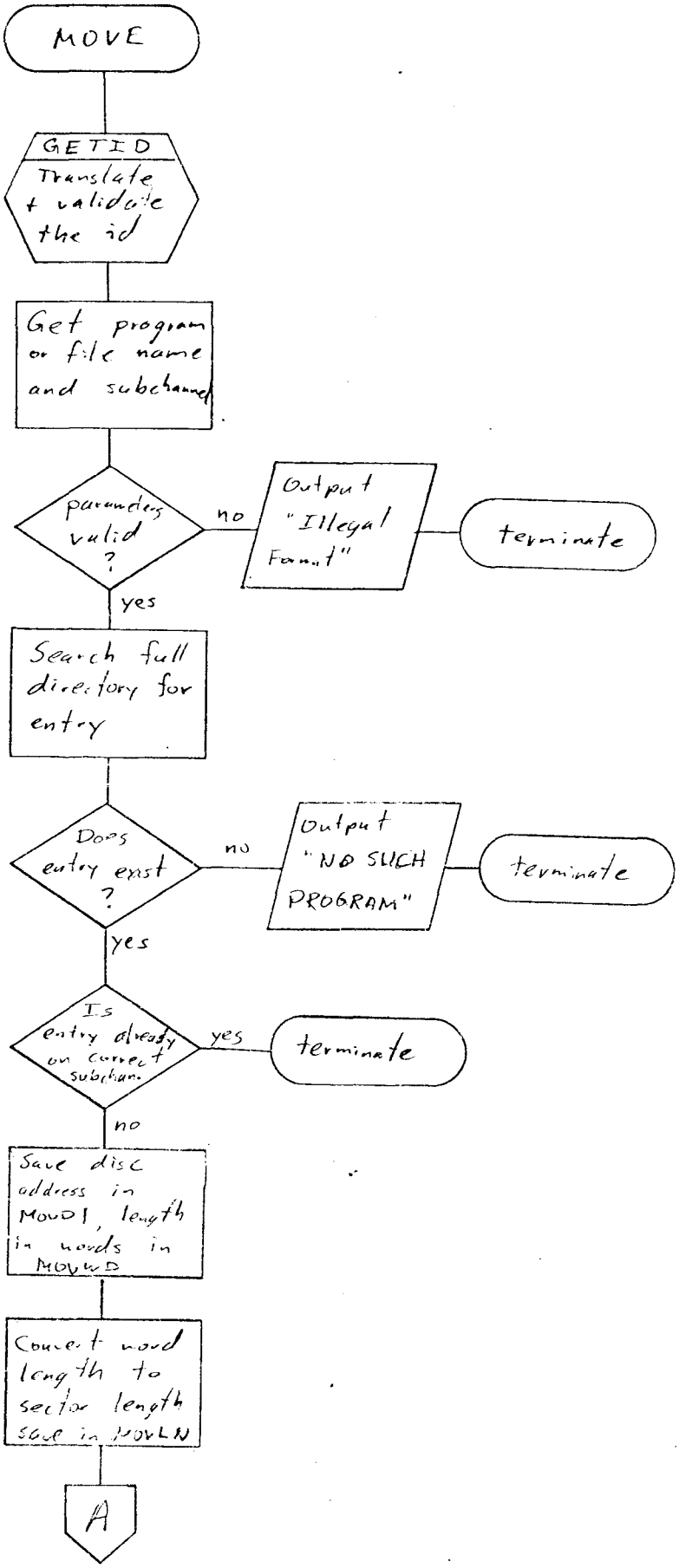
KILLID OVERLAY (1 of 2)

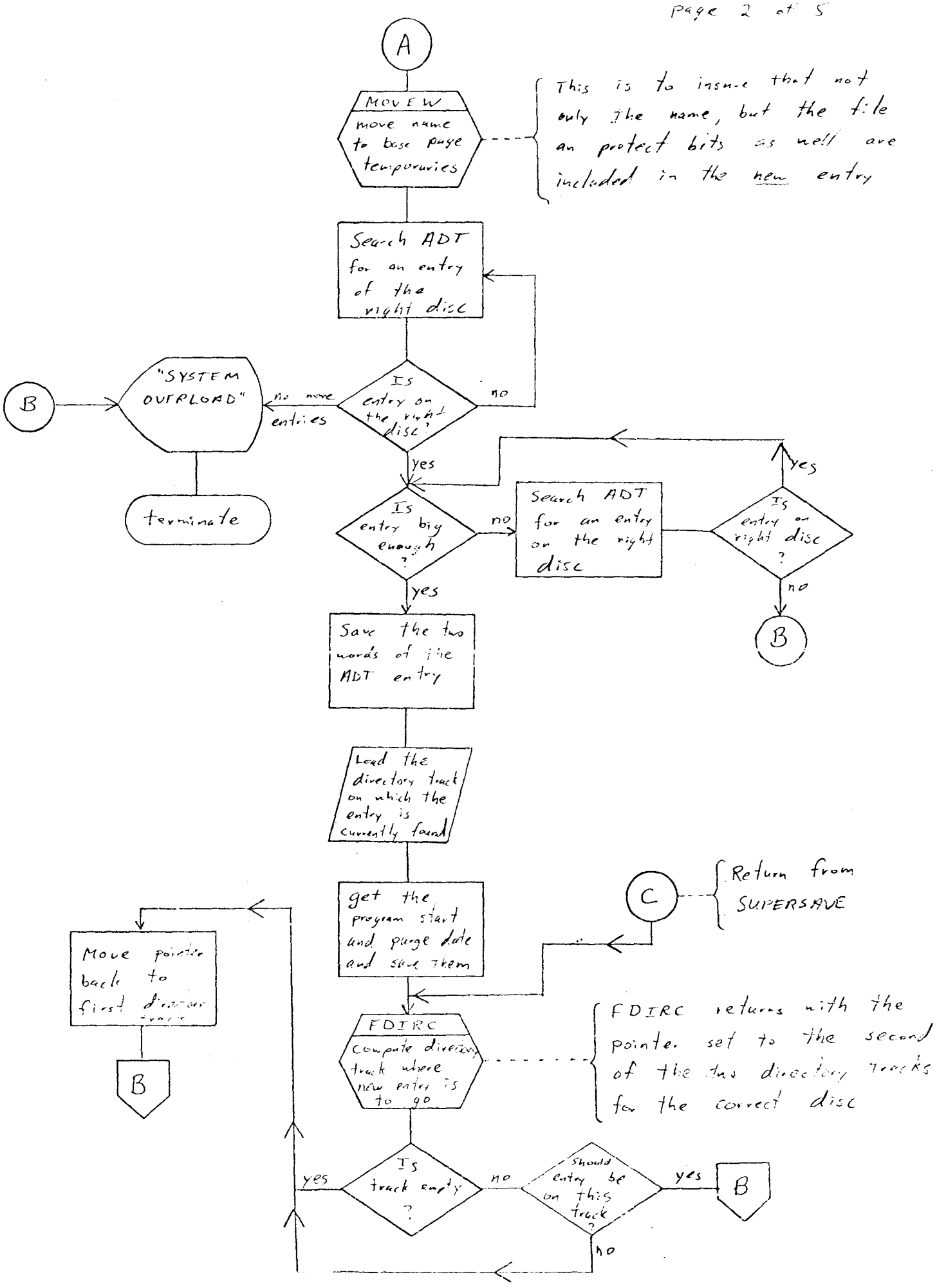


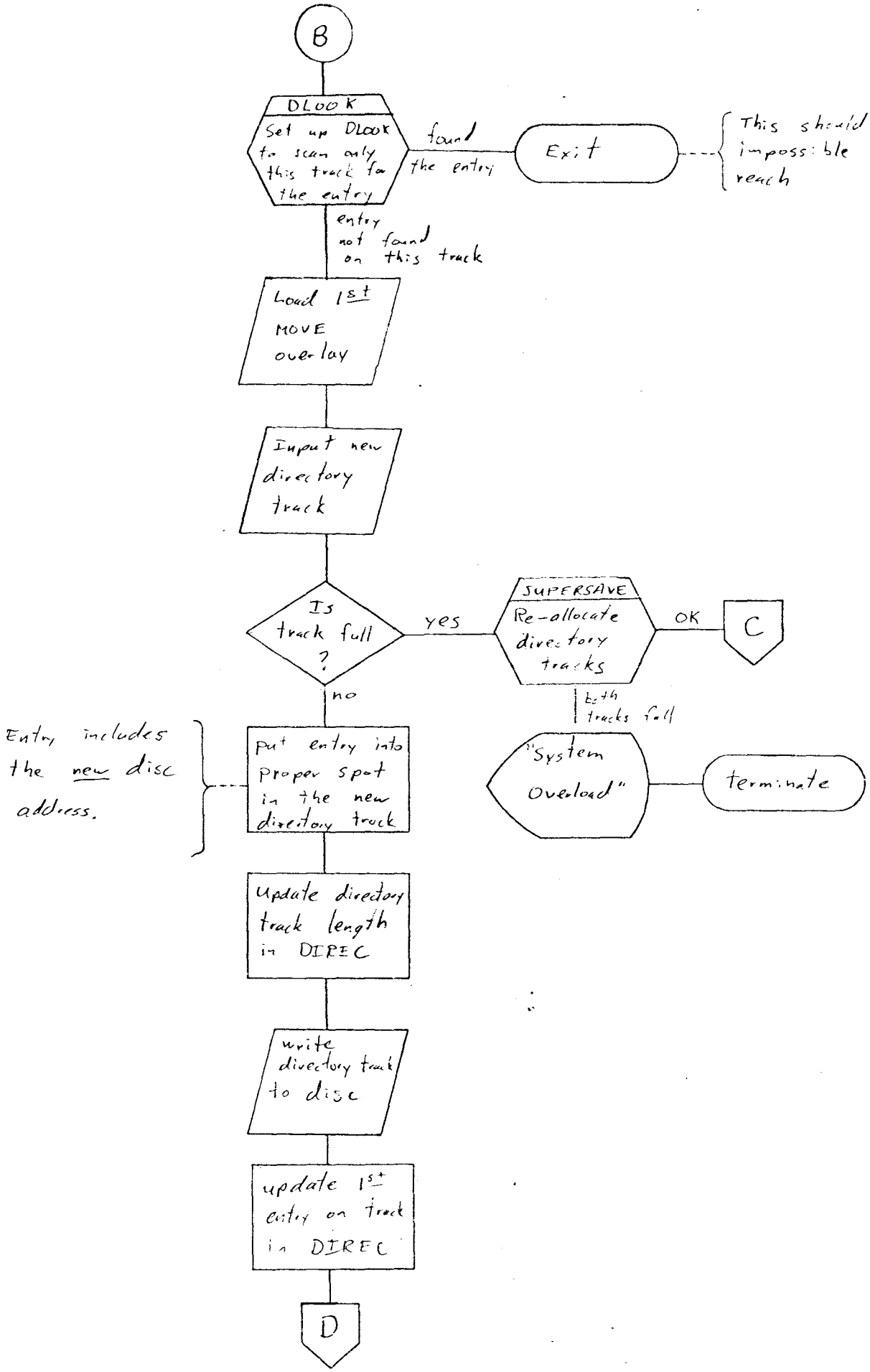
KILLID OVLKLAY (2 of 2)

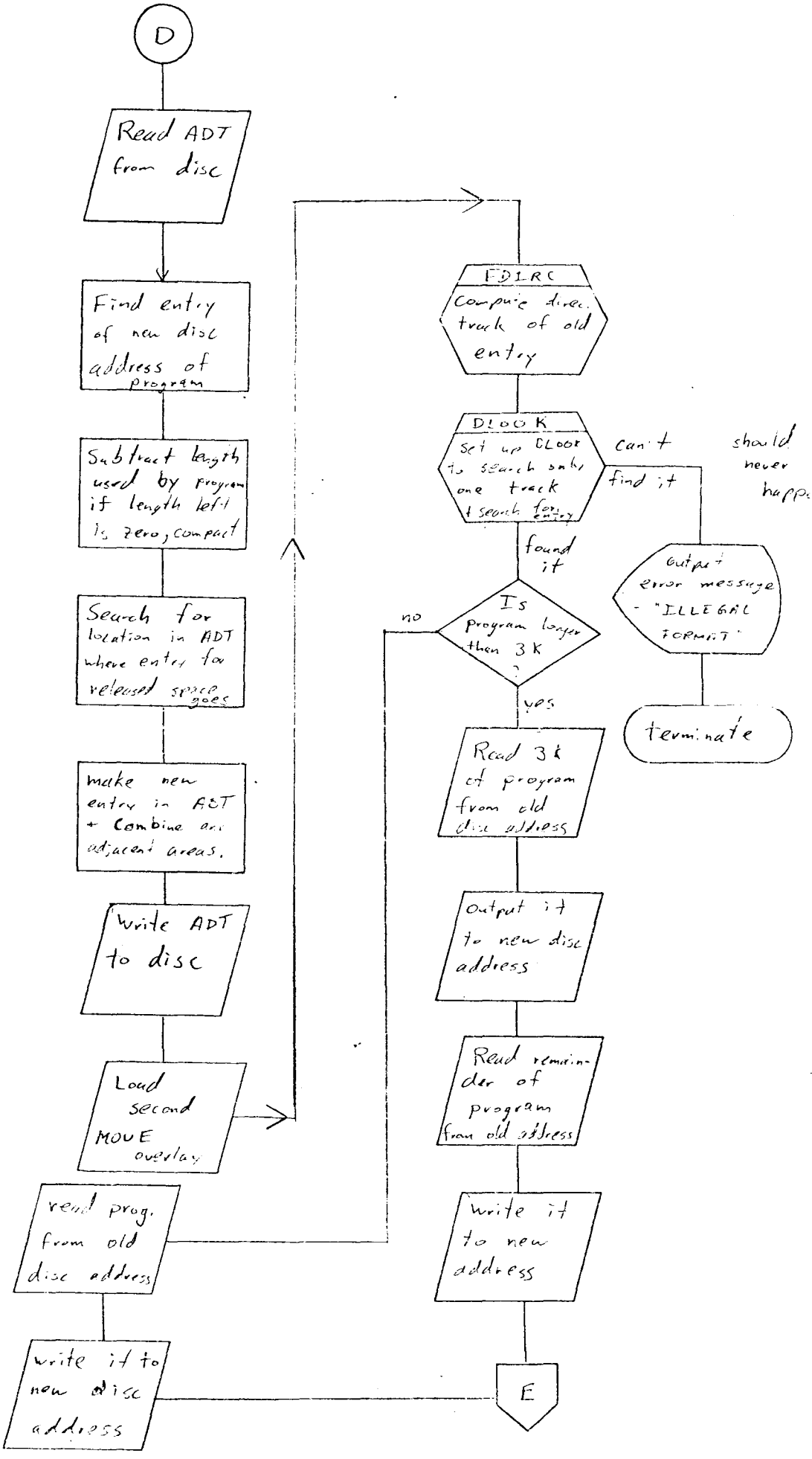


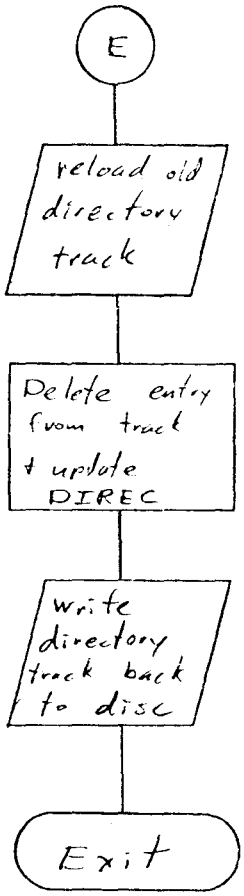
MOVE


MOVE
GETID
Translate & validate the id
Get program or file name and subchannel
parameters valid ?
no
Output "Illegal Format"
terminate
yes
Search full directory for entry
Does entry exist ?
no
Output "NO SUCH PROGRAM"
terminate
yes
Is entry already on correct subchan.
yes
terminate
no
Save disc address in MOVD1, length in words in MOVWD
Convert word length to sector length save in MOVLN
A

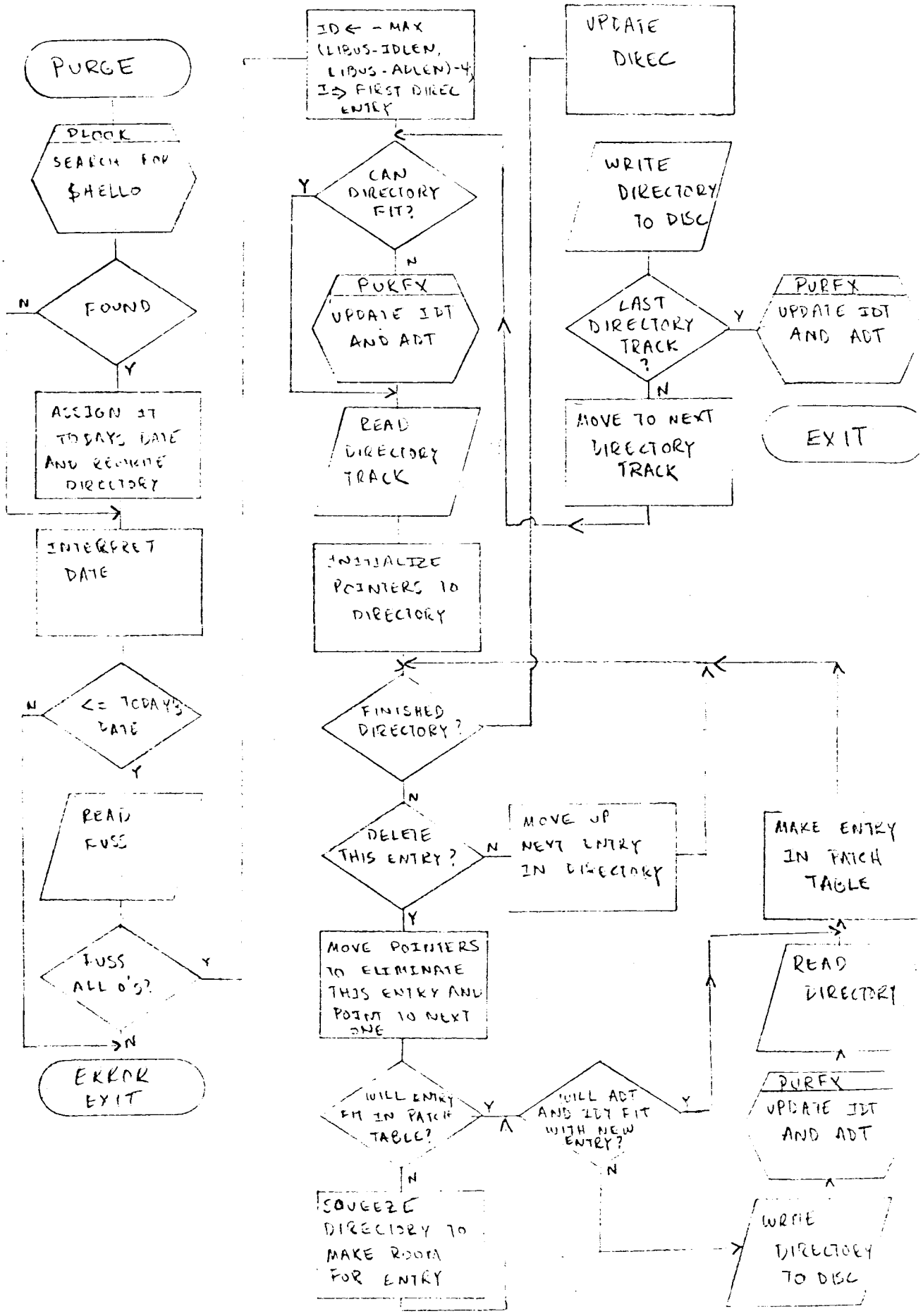

A
MOVEW
move name to base page temporaries
This is to insure that not only the name, but the file an protect bits as well are included in the new entry
Search ADT for an entry of the right disc
B
"SYSTEM OVERLOAD"
terminate
no more entries
Is entry on the right disc ?
no
yes
Is entry big enough ?
no
Search ADT for an entry on the right disc
Is entry on right disc ?
yes
no
B
yes
Save the two words of the ADT entry
Load the directory track on which the entry is currently found
get the program start and purge date and save them
C
Return from SUPERSAVE
Move pointer back to first directory entry
B
FDIRC
Compute directory track where new entry is to go
FDIRC returns with the pointer set to the second of the two directory tracks for the correct disc
Is track empty ?
yes
no
Should entry be on this track ?
yes
B
no


B
DLOOK
Set up DLOOK to scan only this track for the entry
found the entry
Exit
This should be impossible to reach
entry not found on this track
Load 1st MOVE overlay
Input new directory track
Is track full ?
yes
SUPERSAVE
Re-allocate directory tracks
OK
C
both tracks full
"System Overload"
terminate
no
Entry includes the new disc address.
put entry into proper spot in the new directory track
Update directory track length in DIREC
write directory track to disc
update 1st entry on track in DIREC
D

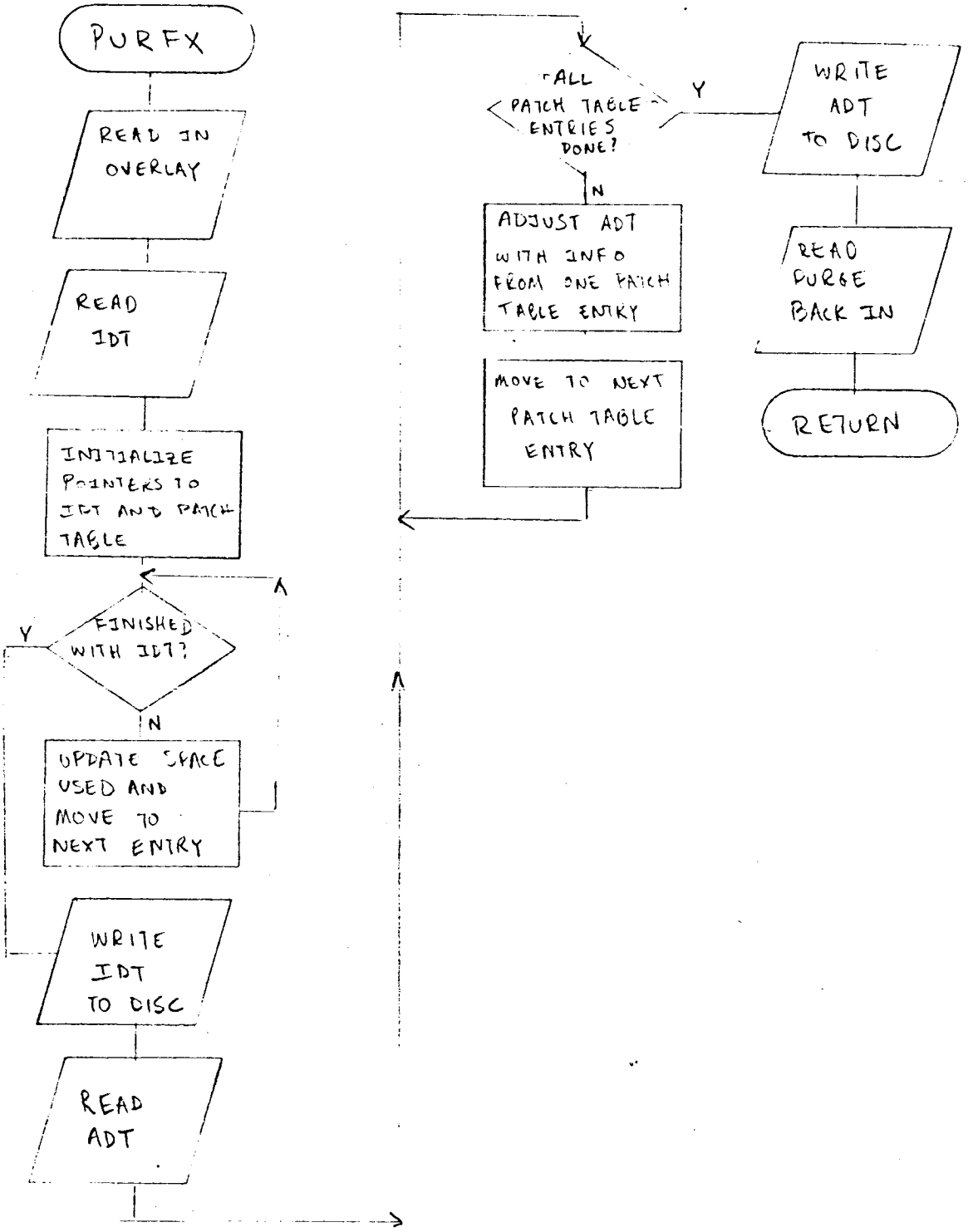

D
Read ADT from disc
Find entry of new disc address of program
Subtract length used by program if length left is zero, compact
Search for location in ADT where entry for released space goes
make new entry in ADT + Combine any adjacent areas.
Write ADT to disc
Load second MOVE overlay
read prog. from old disc address
write it to new disc address
E
FDIRC
Compute direc. track of old entry
DLOOK
Set up DLOOK to search entire one track & search for entry
can't find it
should never happ
found it
Is program longer then 3K ?
no
yes
output error message - "ILLEGAL FORMAT"
terminate
Read 3k of program from old disc address
Output it to new disc address
Read remainder of program from old address
write it to new address

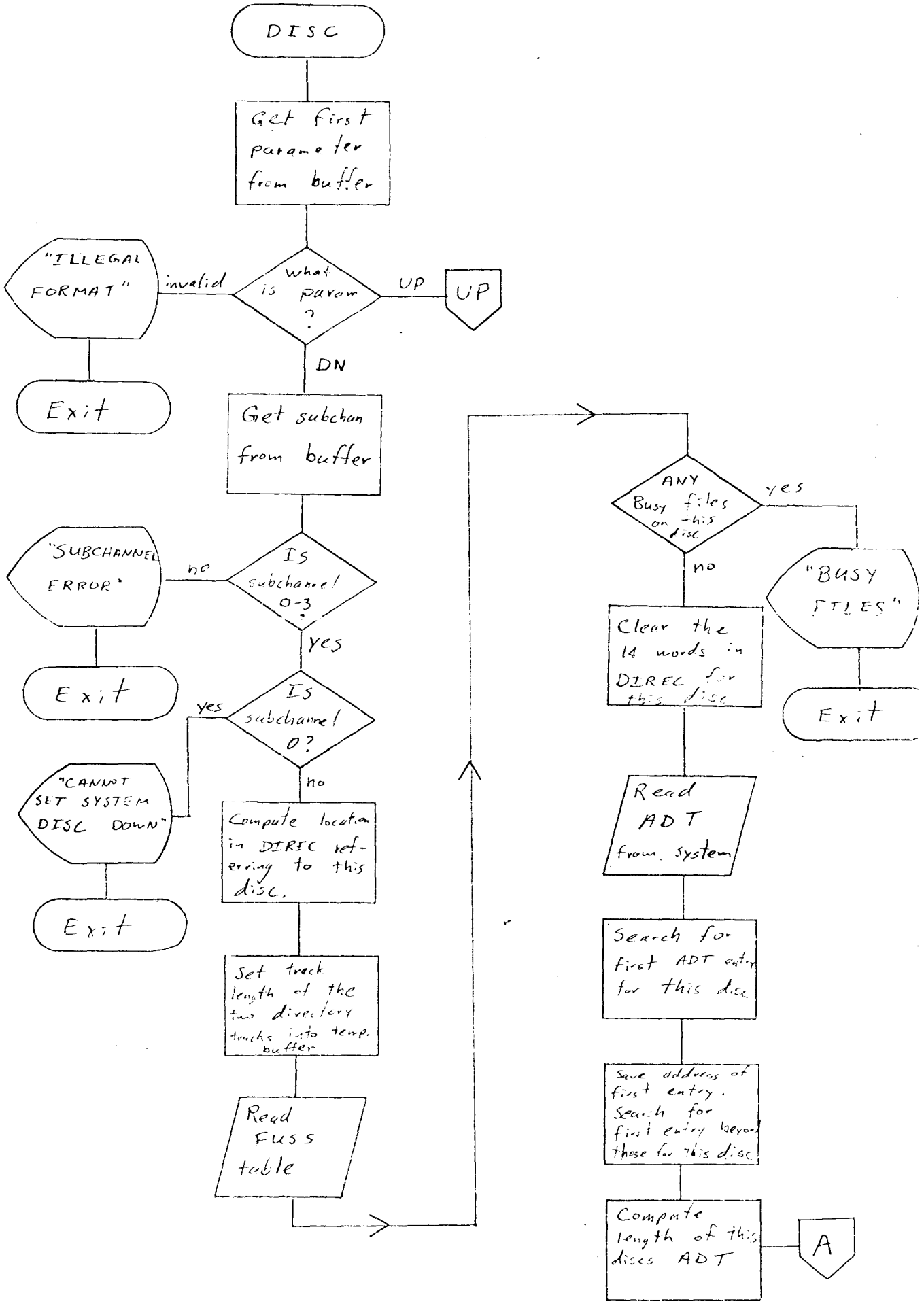

E
reload old directory track
Delete entry from track + update DIREC
write directory track back to disc
Exit

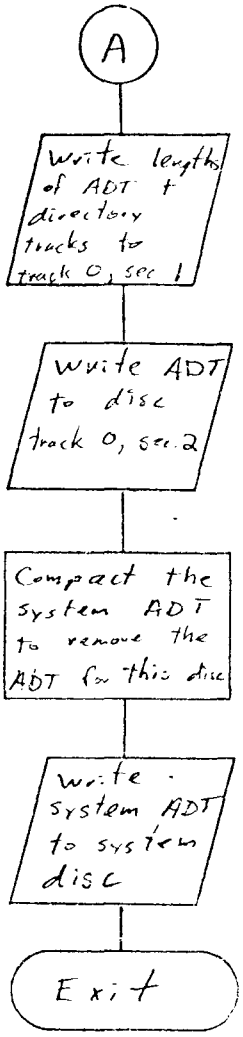
PURGE

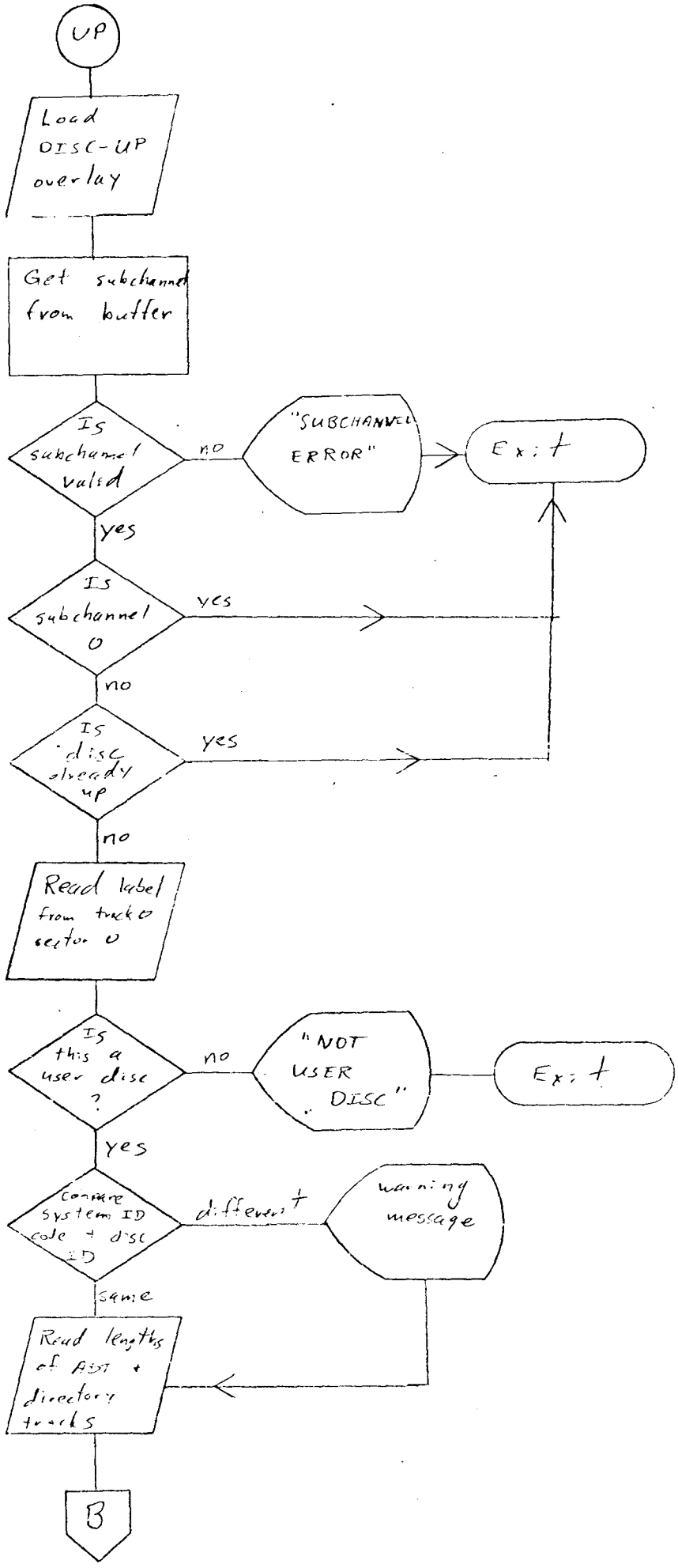

PURGE
BLOCK SEARCH FOR $HELLO
FOUND
N
Y
ASSIGN IT TODAYS DATE AND REWRITE DIRECTORY
INTERPRET DATE
<= TODAY'S DATE
N
Y
READ FUSS
FUSS ALL 0'S?
Y
N
ERROR EXIT
ID ← -MAX (LIBUS-IDLEN, LIBUS-ALLEN)-4; I⇒ FIRST DIREC ENTRY
CAN DIRECTORY FIT?
Y
N
PURFX UPDATE IDT AND ADT
READ DIRECTORY TRACK
INITIALIZE POINTERS TO DIRECTORY
FINISHED DIRECTORY?
N
DELETE THIS ENTRY?
N
MOVE UP NEXT ENTRY IN DIRECTORY
Y
MOVE POINTERS TO ELIMINATE THIS ENTRY AND POINT TO NEXT ONE
WILL ENTRY FIT IN PATCH TABLE?
Y
WILL ADT AND IDT FIT WITH NEW ENTRY?
Y
N
SQUEEZE DIRECTORY TO MAKE ROOM FOR ENTRY
UPDATE DIREC
WRITE DIRECTORY TO DISC
LAST DIRECTORY TRACK?
Y
N
PURFX UPDATE IDT AND ADT
EXIT
MOVE TO NEXT DIRECTORY TRACK
MAKE ENTRY IN PATCH TABLE
READ DIRECTORY
PURFX UPDATE IDT AND ADT
WRITE DIRECTORY TO DISC

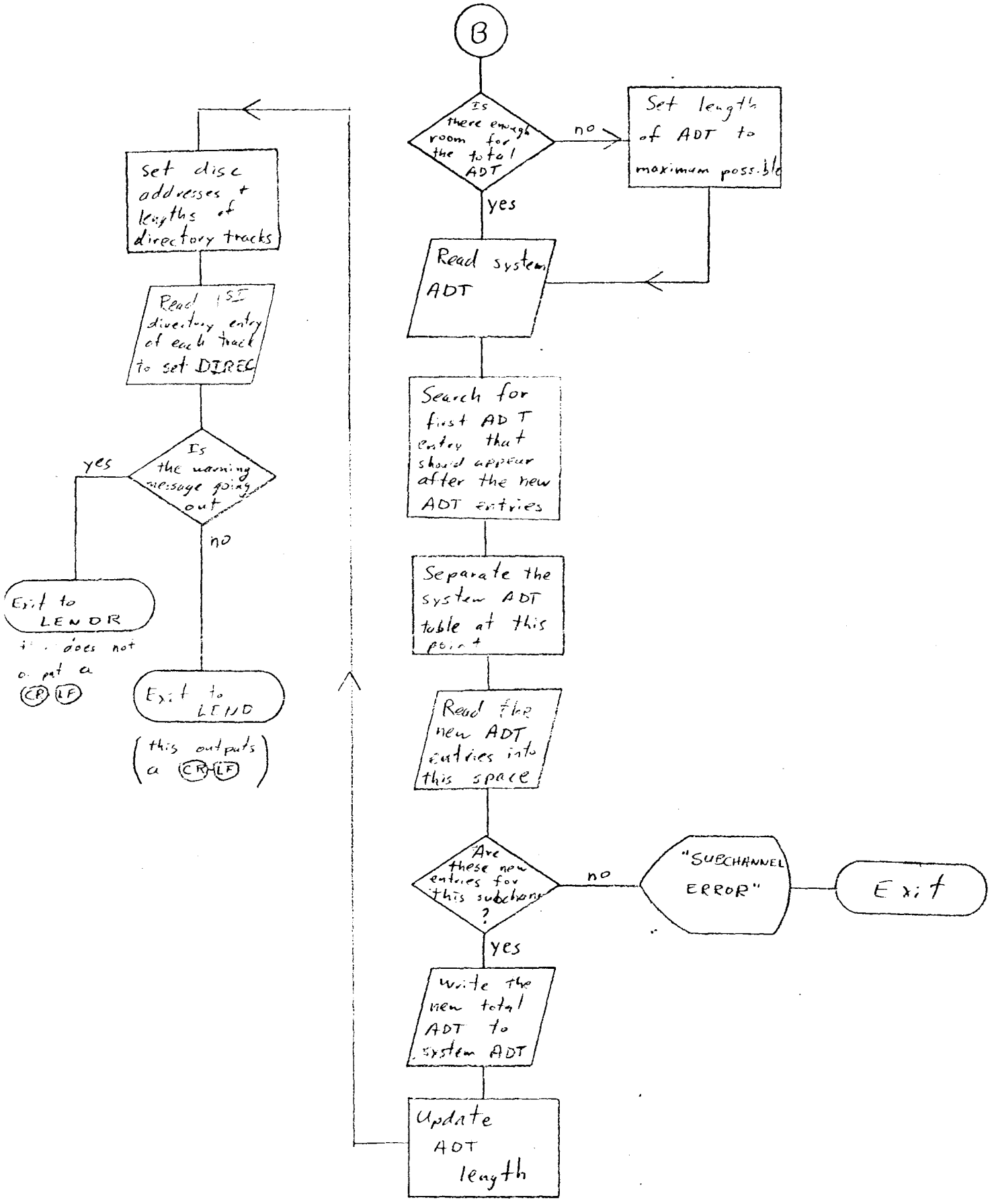
PURGE OVERLAY




DISC
Get first parameter from buffer
What is param ?
invalid
"ILLEGAL FORMAT"
Exit
UP
UP
DN
Get subchan from buffer
Is subchannel 0-3 ?
no
"SUBCHANNEL ERROR"
Exit
yes
Is subchannel 0?
yes
"CANNOT SET SYSTEM DISC DOWN"
Exit
no
Compute location in DIREC referring to this disc.
Set track length of the two directory tracks into temp. buffer
Read Fuss table
ANY Busy files on this disc
yes
"BUSY FILES"
Exit
no
Clear the 14 words in DIREC for this disc
Read ADT from system
Search for first ADT entry for this disc
Save address of first entry. Search for first entry beyond those for this disc
Compute length of this discs ADT
A


A
write lengths of ADT + directory tracks to track 0, sec 1
write ADT to disc track 0, sec. 2
Compact the system ADT to remove the ADT for this disc
write system ADT to system disc
Exit


UP
Load DISC-UP overlay
Get subchannel from buffer
Is subchannel valid
no
"SUBCHANNEL ERROR"
Exit
yes
Is subchannel 0
yes
no
Is disc already up
yes
no
Read label from track 0 sector 0
Is this a user disc ?
no
"NOT USER DISC"
Exit
yes
Compare system ID code + disc ID
different
warning message
same
Read lengths of AST + directory tracks
B

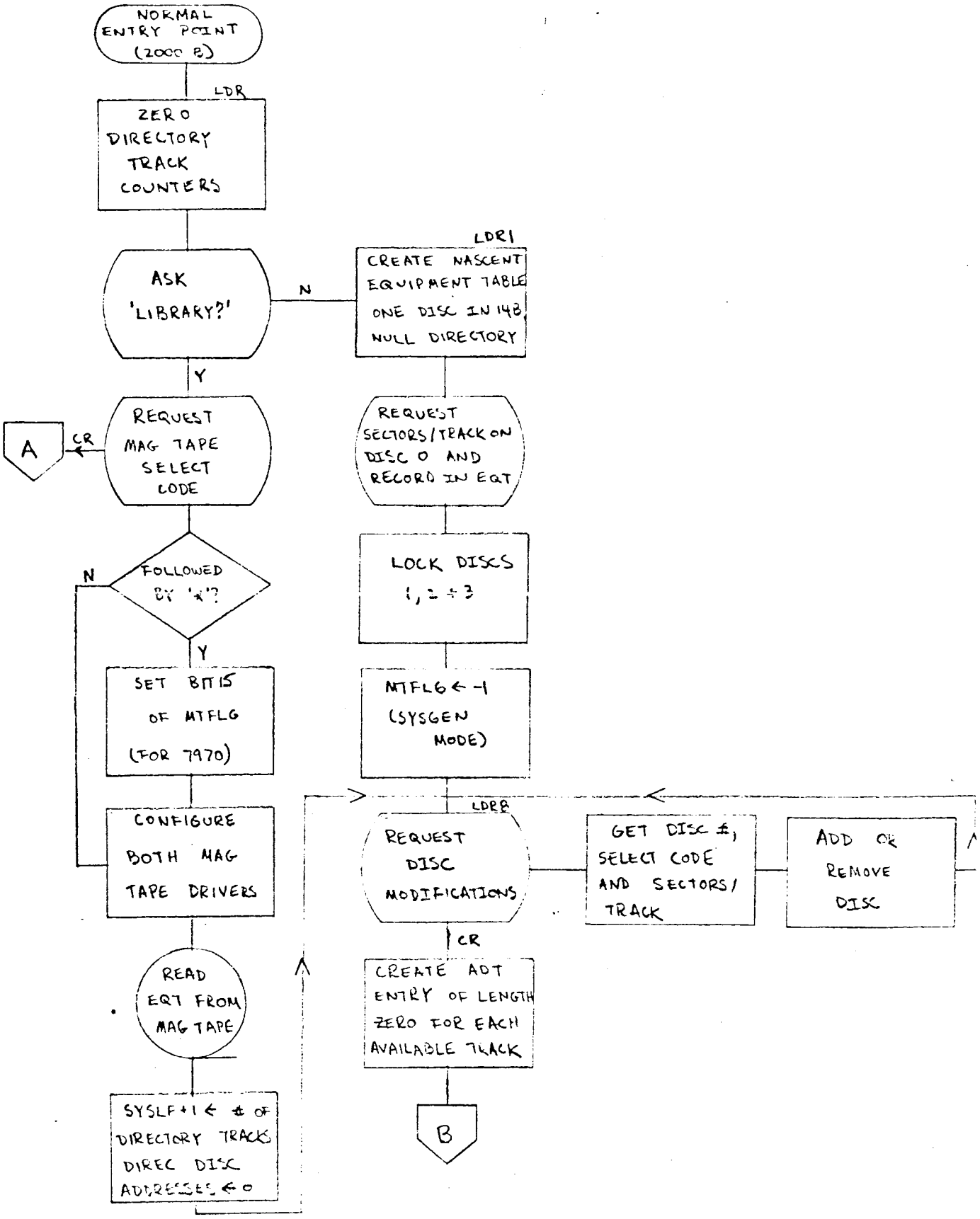

B
Is there enough room for the total ADT
no
Set length of ADT to maximum possible
yes
Read system ADT
Search for first ADT entry that should appear after the new ADT entries
Separate the system ADT table at this point
Read the new ADT entries into this space
Are these new entries for this subchannel ?
no
"SUBCHANNEL ERROR"
Exit
yes
Write the new total ADT to system ADT
Update ADT length
set disc addresses + lengths of directory tracks
Read ISE directory entry of each track to set DIREC
Is the warning message going out
yes
Exit to LENDOR
it does not output a CR LF
no
Exit to LEND
(this outputs a CR LF)

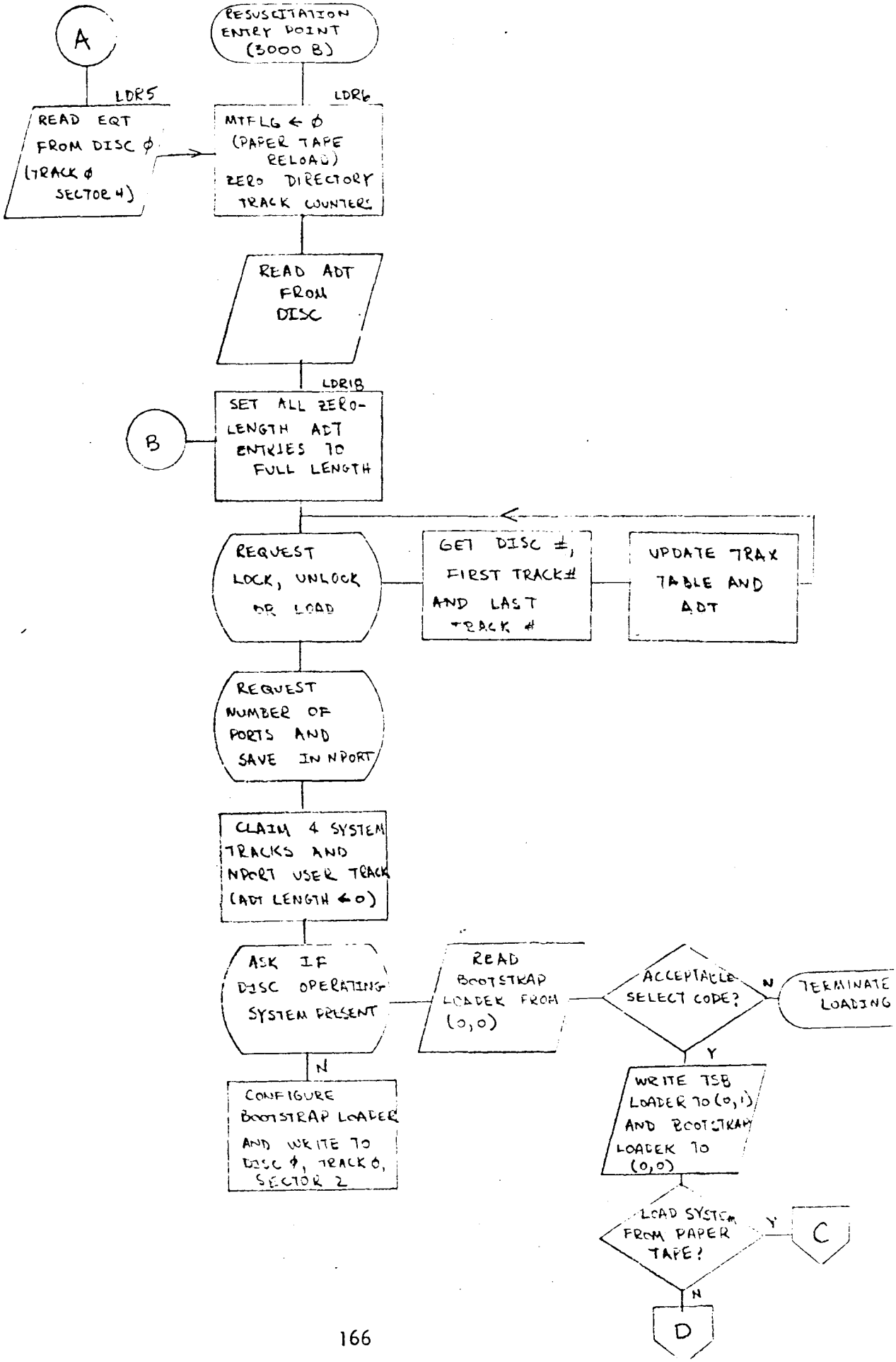
2000B TIME SHARED BASIC LOADER

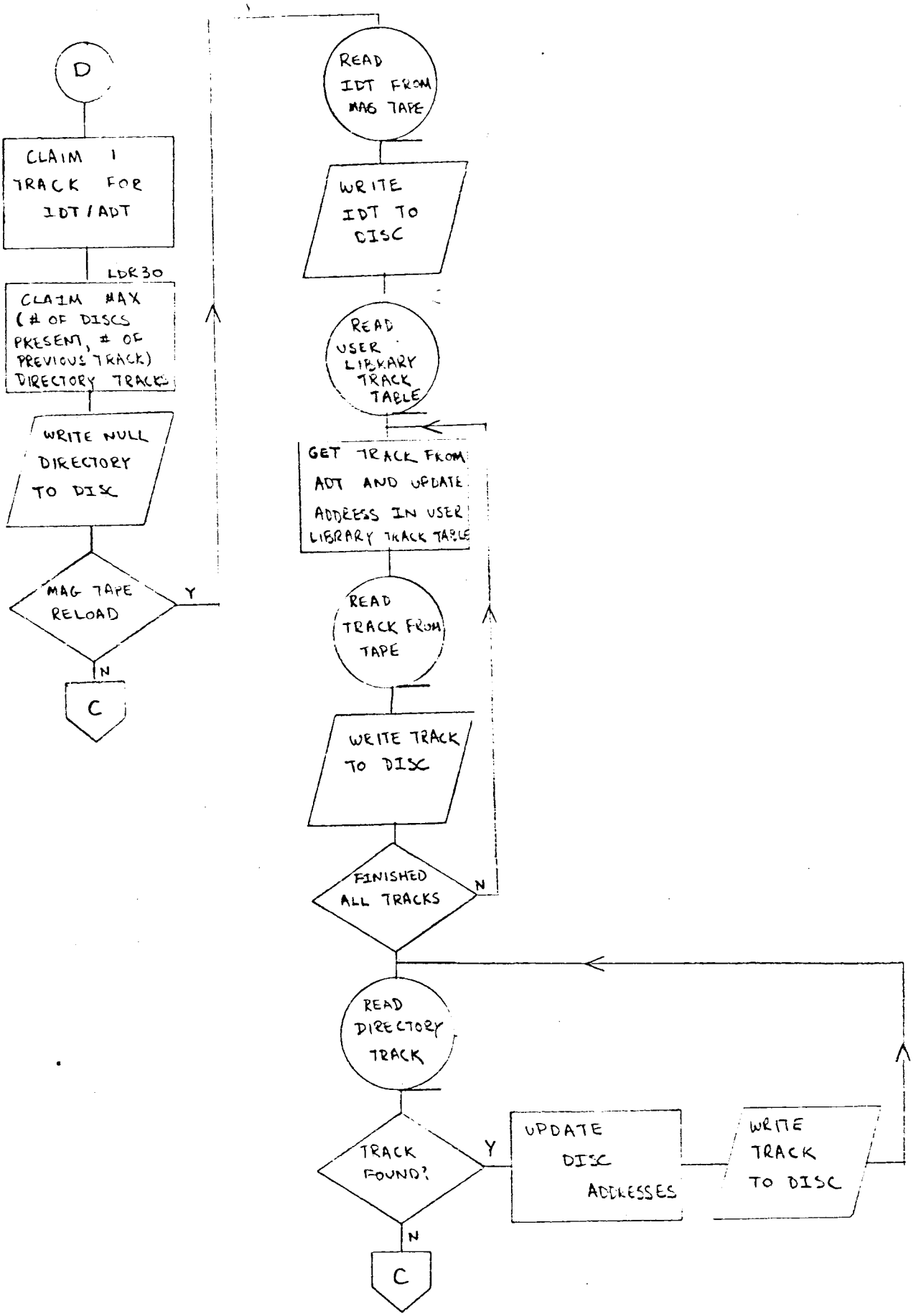
The 2000B Time Shared Basic Loader is a separate program which runs on the main processor. It performs the following functions:

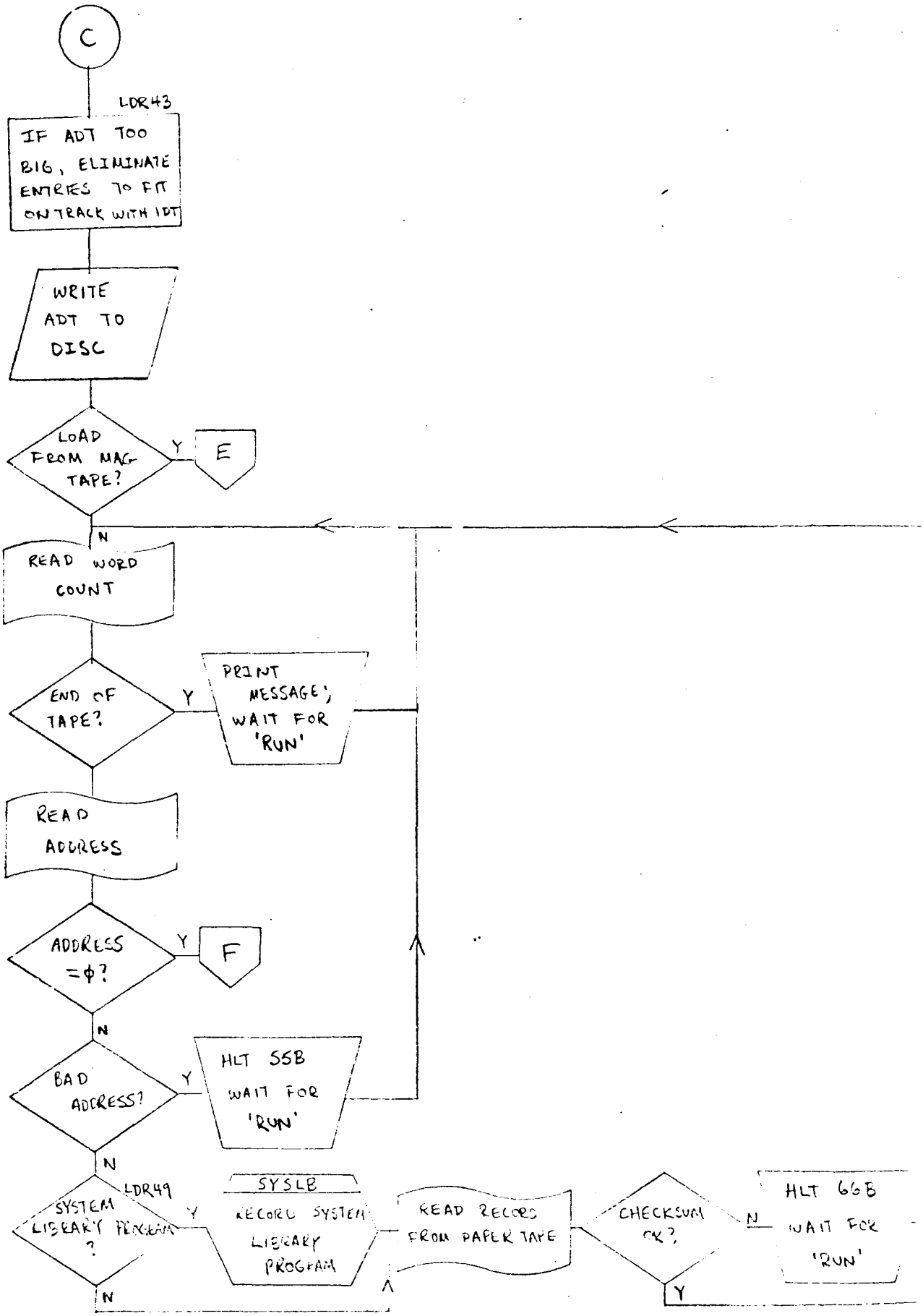
1. Generate a new system from paper tape.
2. Reload the system from mag tape following a mag tape sleep.
3. Reload the system from disc following a sleep.
4. Link a new system (on paper tape) with the library of an older system which is on the disc.
5. Resuscitate the system following a software blowup, machine parity halt, operator error, etc.
6. Dump the system to mag tape. This code is in the loader and is retrieved from the disc by the SLEEP command.

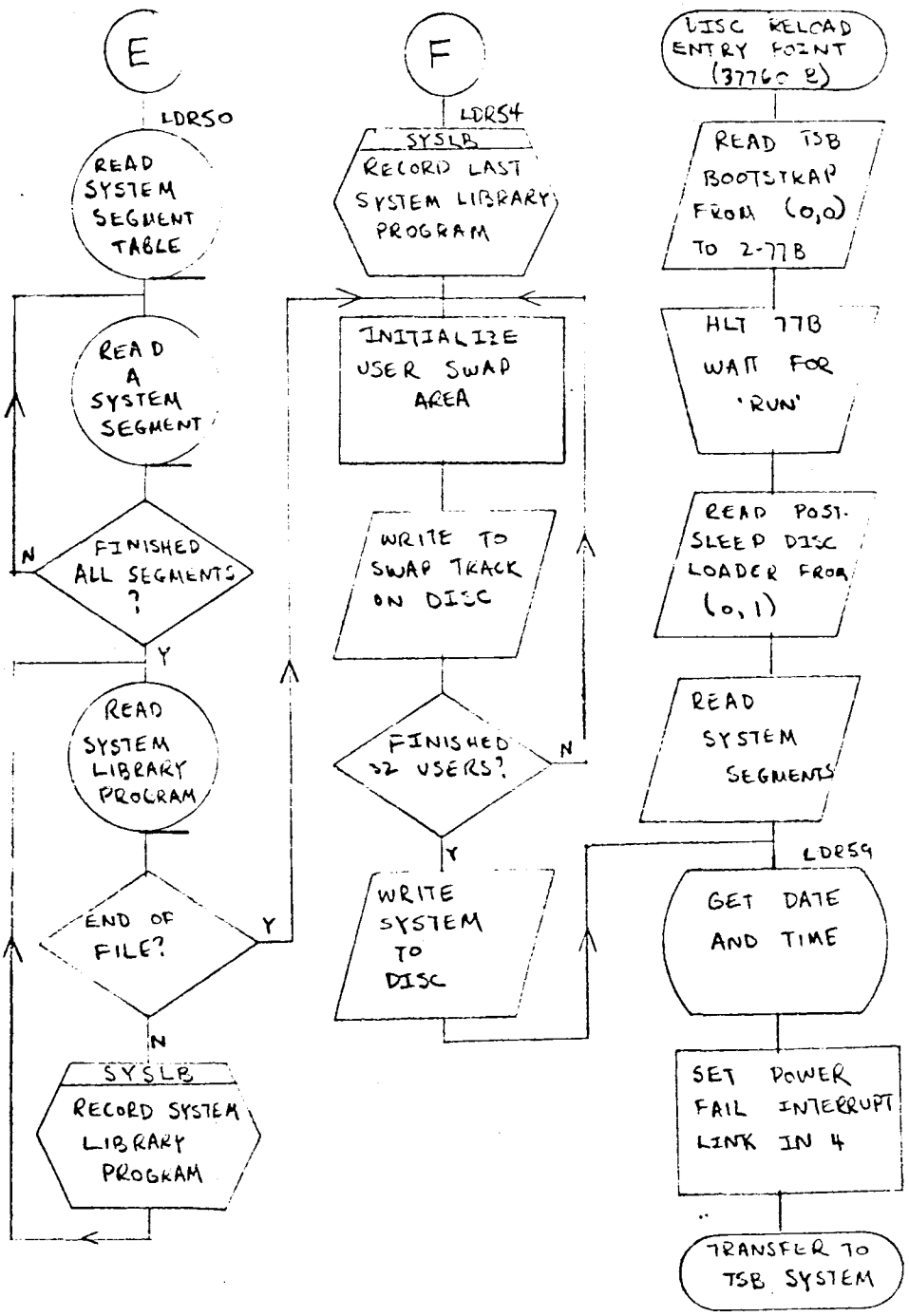
The operation of the loader is straightforward and can be gleaned by studying the listing and the attached flowcharts.

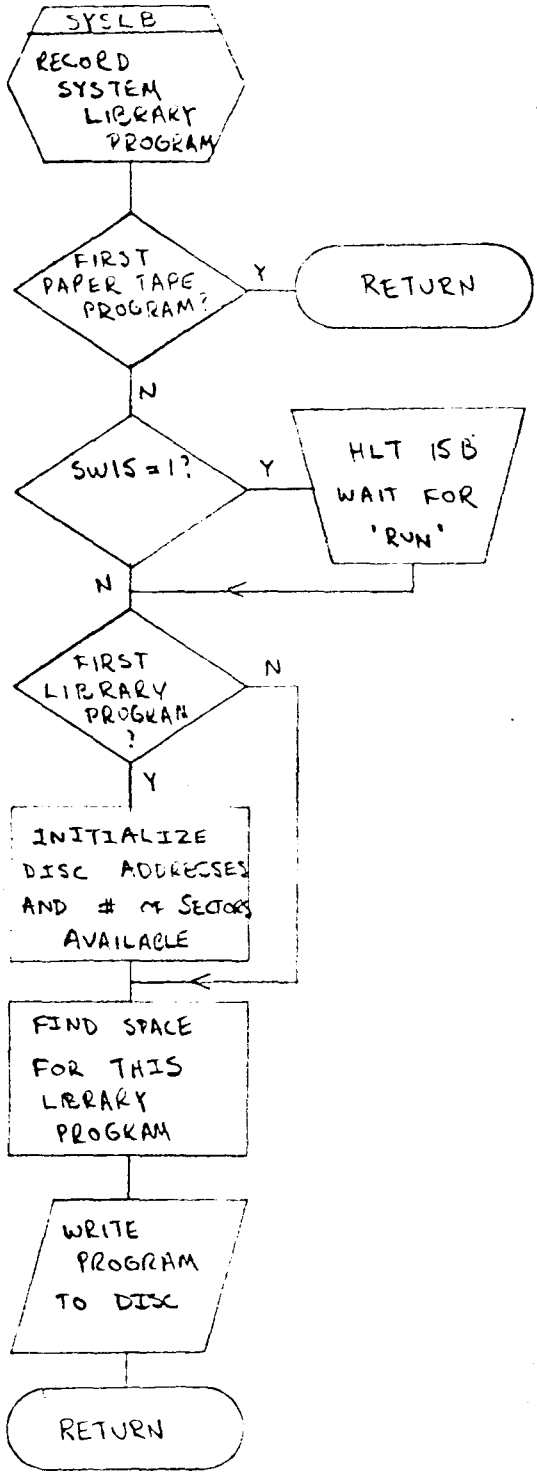

NORMAL ENTRY POINT (2000 B)
LDR
ZERO DIRECTORY TRACK COUNTERS
ASK 'LIBRARY?'
N
LDR1
CREATE NASCENT EQUIPMENT TABLE ONE DISC IN 14B, NULL DIRECTORY
Y
REQUEST MAG TAPE SELECT CODE
CR
A
REQUEST SECTORS/TRACK ON DISC 0 AND RECORD IN EQT
FOLLOWED BY '*'?
N
Y
LOCK DISCS 1, 2 & 3
SET BIT 15 OF MTFLG (FOR 7970)
MTFLG ← -1 (SYSGEN MODE)
CONFIGURE BOTH MAG TAPE DRIVERS
LDR2
REQUEST DISC MODIFICATIONS
GET DISC #, SELECT CODE AND SECTORS/TRACK
ADD OR REMOVE DISC
CR
READ EQT FROM MAG TAPE
CREATE ADT ENTRY OF LENGTH ZERO FOR EACH AVAILABLE TRACK
SYSLF+1 ← # OF DIRECTORY TRACKS DIREC DISC ADDRESSES ← 0
B


A
RESUSCITATION ENTRY POINT (3000 B)
LDR5
READ EQT FROM DISC ϕ (TRACK ϕ SECTOR 4)
LDR6
MTFLG ← ϕ (PAPER TAPE RELOAD) ZERO DIRECTORY TRACK COUNTERS
READ ADT FROM DISC
LDR18
B
SET ALL ZERO-LENGTH ADT ENTRIES TO FULL LENGTH
REQUEST LOCK, UNLOCK OR LOAD
GET DISC #, FIRST TRACK# AND LAST TRACK #
UPDATE TRAX TABLE AND ADT
REQUEST NUMBER OF PORTS AND SAVE IN NPORT
CLAIM 4 SYSTEM TRACKS AND NPORT USER TRACK (ADT LENGTH ← 0)
ASK IF DISC OPERATING SYSTEM PRESENT
READ BOOTSTRAP LOADER FROM (0,0)
ACCEPTABLE SELECT CODE?
N
TERMINATE LOADING
Y
WRITE TSB LOADER TO (0,1) AND BOOTSTRAP LOADER TO (0,0)
N
CONFIGURE BOOTSTRAP LOADER AND WRITE TO DISC ϕ, TRACK ϕ, SECTOR 2
LOAD SYSTEM FROM PAPER TAPE?
Y
C
N
D


D
CLAIM 1 TRACK FOR IDT/ADT
LDR30
CLAIM MAX (# OF DISCS PRESENT, # OF PREVIOUS TRACK) DIRECTORY TRACKS
WRITE NULL DIRECTORY TO DISC
MAG TAPE RELOAD
Y
N
C
READ IDT FROM MAG TAPE
WRITE IDT TO DISC
READ USER LIBRARY TRACK TABLE
GET TRACK FROM ADT AND UPDATE ADDRESS IN USER LIBRARY TRACK TABLE
READ TRACK FROM TAPE
WRITE TRACK TO DISC
FINISHED ALL TRACKS
N
READ DIRECTORY TRACK
TRACK FOUND?
Y
UPDATE DISC ADDRESSES
WRITE TRACK TO DISC
N
C


C
LDR43
IF ADT TOO BIG, ELIMINATE ENTRIES TO FIT ON TRACK WITH IDT
WRITE ADT TO DISC
LOAD FROM MAG TAPE?
Y
E
N
READ WORD COUNT
END OF TAPE?
Y
PRINT MESSAGE; WAIT FOR 'RUN'
READ ADDRESS
ADDRESS =ϕ?
Y
F
N
BAD ADDRESS?
Y
HLT 55B WAIT FOR 'RUN'
N
LDR49
SYSTEM LIBRARY PROGRAM ?
Y
SYSLB
RECORD SYSTEM LIBRARY PROGRAM
READ RECORD FROM PAPER TAPE
CHECKSUM OK?
N
HLT 66B WAIT FOR 'RUN'
Y
N


E
LDRSO
READ SYSTEM SEGMENT TABLE
READ A SYSTEM SEGMENT
FINISHED ALL SEGMENTS ?
N
Y
READ SYSTEM LIBRARY PROGRAM
END OF FILE?
Y
N
SYSLB
RECORD SYSTEM LIBRARY PROGRAM
F
LDRS4
SYSLB
RECORD LAST SYSTEM LIBRARY PROGRAM
INITIALIZE USER SWAP AREA
WRITE TO SWAP TRACK ON DISC
FINISHED 32 USERS?
N
Y
WRITE SYSTEM TO DISC
DISC RELOAD ENTRY POINT (37760 B)
READ TSB BOOTSTRAP FROM (0,0) TO 2-77B
HLT 77B
WAIT FOR 'RUN'
READ POST-SLEEP DISC LOADER FROM (0,1)
READ SYSTEM SEGMENTS
LDRS9
GET DATE AND TIME
SET POWER FAIL INTERRUPT LINK IN 4
TRANSFER TO TSB SYSTEM


SYSLB
RECORD SYSTEM LIBRARY PROGRAM
FIRST PAPER TAPE PROGRAM?
Y
RETURN
N
SW15 = 1?
Y
HLT 15B WAIT FOR 'RUN'
N
FIRST LIBRARY PROGRAM?
N
Y
INITIALIZE DISC ADDRESSES AND # OF SECTORS AVAILABLE
FIND SPACE FOR THIS LIBRARY PROGRAM
WRITE PROGRAM TO DISC
RETURN

SUPPLEMENTARY NOTES ON BASIC

I SYNTAX

The general process of analyzing an input to the language processor is displayed in the section on flow charts. The annotations in the listing explain the actions of the subroutines, while the core map and section on internal representation describe the objects/structures being created or manipulated. The BASIC syntax, in conjunction with the listing, explains the method of identification and recognition of legitimate BASIC statements from the input string.

II Phase 2

A. Compilation

The preliminary section of CMPLE prepares for execution of the program following a successful compilation. Null programs require no processing. If a sequence number follows the RUN (e.g., RUN - 22Ø) the interpreter's program counter is set to the first statement whose sequence number equals or exceeds the reference, otherwise it is set to the first statement of the user program. If the common area has not been allocated, ALCOM is called to compute the space needed and move the program accordingly. If the program is already compiled (SYMTB=SPTR≠0) PBPTR is set back to the first word following the value table (FCORE) and phase 2 simply reinitializes all of the variables to undefined. If the program is semi-compiled (SPTR=0, SYMTB≠0) we may skip building the symbol table. Otherwise FILTB is set to Ø so PRNST will not terminate compilation by mistaking it for decompilation.

The symbol table is then built as explained in the listing (Refer to the flow chart for general logic flow and to BASIC Variable Storage Allocation for a visual example). Also, at this time statement number references are replaced by absolute addresses. This is facilitated by dividing the program into 32 parts and building an 64 word table in ERSEC containing the first statement number and address of each part. During compilation SPTR points to the program word being processed. Pointers to <FILES statements> are stored in

FLSTS and a count of them is kept in FILCT. An error in compilation will cause a call to DCMPL to restore the source form of the program followed by a call to the error routine. If after a successful compilation at least one <FILES statement> has been found, BASIC calls the system, which analyzes the <FILES statements> and builds the file table, filling in the first, second, and fourth words of each entry.

The symbol routine has two entry points: SSYMT is used for functions and simple variables and ASYMT is used for array and string variables. Because the dimensionality of an array variable may not be known locally (e.g., MAT A = some symbols may have two entries. If this is the case, the "don't know" entry will always be farther down in the table (i.e., have a higher core address) than its dimensioned counterpart.

B. Value

VALUE is responsible for detecting deficiencies in the symbol table, allocating storage for the values of symbols (i.e., building the value table and common area), and initializing the values of all variables except those in common. Only the last of these functions is performed if a program is already compiled when a RUN command is received. The process of building the value table is described in the listing. Note that for arrays in common, the declared dimensions in the <COM statement> are checked against those in the common area. If they match and the dynamic dimensions are consistent (i.e., less than or equal to the declared ones) then the values are left alone. Otherwise they are set to undefined and both sets of dimensions are set equal to those in the <COM statement>. For strings, the physical length is checked against the declared length and the logical length tested to be less than or equal to the physical length. If these tests fail the physical length is set to the declared length and the logical length is set to zero. Simple variables in common are left untouched.

Several errors may be encountered while building the value table. The occurrence of a null symbol (bit pattern of Ø) in the symbol table means that an array symbol is used in the program, but never in such a way that its dimensionality can be determined. If the second word of a function entry is zero, no <DEF statement> for that function appears in the program. Arrays

of more than 2500 elements are not allowed. For all errors the program is decompiled before the call to the error routine.

C. Decompilation

Programs are decompiled when any error occurs during compilation, building of the file table, building of the value table, or when the program is to be modified or saved in the user library. Since in the first of these only a portion of the program is compiled, the pointer SPTR is used to determine how much to be decompiled (A fully compiled program always has SPTR pointing to the first word following the program). The program is moved so that SPROG=PBUFF (no common area). The process is explained in the listing.

D. The routine PRNST

PRNST is used by both CMPLE and DCMPL to scan the program and skip over those portions not affected by compiling. PRNST assumes responsibility for recognizing extra <FILES statements> and <COM statements> that are out of order. If such an error condition is encountered, SPTR is set to point before the statement which caused the error (it hasn't been compiled). Then PRNST calls DCMPL, which calls PRNST. The statement causing the error is not seen this time, so PRNST and DCMPL can exit correctly.

III EXECUTION

A. Main Loop

Upon completion of the value assignment in phase 2, control transfers to XEC. FCORE saves a pointer to the first word following the value table (used in repeated RUNS of a program). After printing the program name (unless the program was CHAINED to) XEC proceeds to initialize the file table. A 64-word buffer is allocated for each file and pointers to the word following it are placed in words 5 and 6 of the file table. The disc address of the record in the buffer (word 3) is set to -1 to indicate that no record is present. Word 7 is set to 0, indicating that no end-of-record/end-of-file exit has been specified. If the file is read-only a message to this effect is printed, following the program name, unless the program was CHAINED to.

Following the preparation of files the initial execution status is set. The initial execution stacks are claimed from free user space and pointers are set to the first constant of the first <DATA statement>, if such exists. The internal print position counter (CHRCT) is set to zero by outputting a carriage return. Phase 2 has already set the BASIC program pointer (PRGCT) to the first statement to be executed.

Execution of a statement simulates the execution of an instruction on a 'BASIC machine'. The sequence number of the statement referenced by PRGCT is saved for possible use by the error routine. PRGCT is advanced to reference the following statement. The type of the current statement is used to branch to the appropriate routine via a jump table. Individual statement routines return to the top of the loop.

B. Statement execution

<LET statement> execution consists simply of evaluating the formula, which is known to contain at least one assignment operator and to have type compatibility (numeric vs. string) by its acceptance by phase 1.

<IF statement> execution forks on the symbol following the IF. The construction 'IF END' causes the following: the file reference is evaluated and tested for existence as one of the program's requested files; if a legitimate reference, the statement reference following the THEN is placed in the end-of-file word of the file's table entry. If not 'IF END', the decision formula is evaluated and if true the statement reference replaces the value of the interpreter's program counter, PRGCT, via the GOTO mechanism.

<GOTO statement> execution consists of choosing a statement reference to replace the program counter. For simple GOTO's this is done trivially; for multi-branch GOTO's this is done by evaluating the index formula and choosing the statement reference in the corresponding list position. If the index value lies outside the list of statement references, the program counter remains unchanged.

<GOSUB statement> execution follows the pattern for the GOTO except that after choosing the new value for the program counter, the old value is saved on the return stack (stack overflow generating an error condition).

<FOR statement> execution opens an active program loop. The for-stack is searched for an entry with the same for-variable; if found, the entry is eliminated (i.e., the previous <FOR statement> with this for variable is closed). A new entry is set on top of the for-stack (extending the for-stack by six words if no entry was eliminated) and a pointer to the for-variable's value entry is put into word 1. Since the first formula in the FOR contains an assignment operator, the formula evaluator, FORMX, initializes the for-variable when it determines the initial value. A reference to the statement following the <FOR statement> is put into word 6 of the for-stack entry (the start-of-loop address). Words 2 and 3 save the result of evaluating the limit value formula. If a step size formula appears explicitly it is evaluated, otherwise 1.Ø is taken as the step size. In either case the value of the step size is left in words 4 and 5 of the for-stack entry. The program counter is set to the statement following the associated <NEXT statement> and control transfers to the <NEXT statement> execution code to compare the initial and limit values (see flow chart).

<NEXT statement> execution decides whether to iterate a loop or close it. The for-stack is searched for an entry with the same for-variable. If none is found the statement is ignored and control passes to the following statement. If the entry is found, any entries above it (more recent entries) are eliminated i.e., they are assumed to belong to nested loops which were not closed by exceeding their limit value but exited otherwise. The value of the for-variable is then incremented by the step size and the new value tested by subtracting the limit value and using the sign of the step size to determine whether a non-negative or non-positive result indicates 'success'. If the result is 'success', the program counter is loaded from word 6 or the for-stack entry (the reference to the statement following the <FOR STATEMENT>). If the result is not 'success', the for-stack entry is eliminated. At this point the program counter already points to the statement following the <NEXT statement> so exit is simply to the main execution loop.

<RETURN statement> execution merely loads the program counter from the top entry of the return stack. An error condition is generated if the return stack is empty.

<INPUT statement> execution assigns values to the input list for both INPUT and MAT INPUT. INITF = Ø and MCNT is meaningless when executing an <INPUT statement>; For MAT INPUT, INITF = -1 and MCNT holds the number (in 2's complement) of elements of the current array as yet unassigned values. IFCNT holds the ordinal number of the current item in the current record (Note that IFCNT is not cumulative over the entire execution of a statement requesting input unless the request is met entirely by one line from the teletype).

The general approach in execution is to determine the address and type of a variable in the input list and then attempt to satisfy it from the input record. When an error occurs in the above process, it is explained along with any necessary corrective action and the value assignment is attempted again, so that errors in the input record will not terminate program execution. For simple input if the next variable in the list is of numeric type its value table address is placed into SBPTR; for array input the base address of the array is put into SBPTR. After filling a simple variable the next variable from the list is taken and a new address generated; after filling an array element SBPTR has been advanced to the next element by the numeric input routine so no new address need be calculated. When MCNT rolls over to zero (an array has been filled) control exits to the MAT INPUT code, which may return with another array's base address in SBPTR and MCNT reset appropriately. If the input record is empty but the variable list is not yet exhausted a request for additional input is made (signified by '??' rather than the initial '?'). SERR is needed as a flag to indicate if under/overflow occurred while converting the latest numeric input, since the error message will have destroyed any additional information in the input record. When looking for a number, the input record is scanned for the first sign (+ or -), digit, or decimal point, which begins the number. Any other characters will be ignored except the ", which will generate a recoverable error.

String input requires fairly complicated analysis of the data transfer. If the string variable does not specify the transfer length (does not have a double subscript), then the next string in the input record is transferred in its entirety and the logical length of the variable set appropriately. If the' next string does not fit, a message is printed and a new string value requested. If the string variable specifies the transfer length then exactly that much of the next string in the input record will be transferred, either truncated or extended by blanks as necessary to achieve the specified length. The 'next string' in the input record begins with the next non-blank character or, if it is a ", the following character, blanks included. The string ends with the first " (which is not part of the string) encountered or with the carriage return (also not part of the string) if no " appears.

Every data item in the input record must be followed by a comma or carriage return and a comma must be followed by another data item. Failure to observe the above will generate recoverable errors. INTMP holds the type of data being sought, INTMP = Ø for a number or INTMP # 0 for a string, and is used by the error recovery code to prepare for the entry.

<ENTER statement> execution assigns a value to a string variable or a simple variable. If a '#' follows the ENTER, the user's port number (0-31) is assigned to the first variable. The <ENTER statement> is timed and the length of time it took to respond (in seconds) is assigned to another variable. The input analysis proceeds much like an input statement with one variable, with the notable exception that no error messages are printed. Instead, the response time variable is negated if an error occurs. If the user does not respond within the alloted time, the response time variable is set to -256. This is non-ambiguous since response times are between 1 and 255 seconds inclusive. Also, for string input leading blanks are non stripped off and quote marks are allowed as characters.

<READ statement> execution assigns values to variables in the list. FDATA is primed to obtain values from either a file of the <DATA statement>s, depending on the presence or lack of a file reference following the READ. A mismatch in type between the variable and the next data item, or a string too long to fit into its designated destination, will generate an error and terminate execution.

<PRINT statement> execution consists of identifying items in the print list and sending the appropriate media equivalent to the teletype or disc file. An initial file reference identifies the statement as a file write and turns off the end-of-line mode; its absence identifies and teletype write and turns on the end-of-line mode. A comma or semicolon turns off the end-of-line mode and generates enough blanks to advance to the next field of 15 characters, if a teletype write. A literal string is written as a string of characters, less quotes, and turns on the end-of-line mode if a teletype write. An END writes an end-of-file mark on the file; it cannot occur in a teletype write. Formulas in the print string are evaluated and the results examined. Formulas which are string variables evaluate to their contents, which is then treated as a literal string. If not a string variable but within a file write statement, the floating point value of the formula is written on the file in its two-word binary representation. If a teletype write, floating point values are converted to an ASCII character string of the decimal equivalent. TAB can only occur in a teletype write; the evaluation of the TAB itself produces the desired action, so the value returned is thrown away, along with a following comma if one exits. For a teletype write all formulas turn on the end-of-line mode. If the end-of-line mode is on after processing the last print item, a carriage return-line feed is printed (This can only occur in a teletype write.).

Before writing a quantity BASIC insures that sufficient space is available to accommodate it. CHRCT keeps track of the current print position on the teletype line (Ø-71). If the character string sent to the teletype would require non-blank characters to be printed past position 71, a carriage return-line feed is output first and CHRCT set to Ø. If an item sent to a file requires more words than remain in the current record, BASIC automatically advances to the next record if in serial mode or exits to the end-of-record code if in record mode.

<RESTORE statement> execution resets the pointers to the DATA block. Beginning at the statement specified, or at the first statement in the program if none is specified, the pointers are set to the first <DATA statement> found, or to the out-of-data condition if none is found.

<END statement> and <STOP statement> execution terminates the program run. Since each requested file has a 64-word buffer in core, the last record written on a file does not exist on the disc in its updated form. Thus END and STOP must force the buffer of each read/write file onto its proper disc sector. Following this, the word DONE is sent to the teletype and control exits to the scheduler.

<CHAIN statement> execution consists of calling the CHAIN library routine to get the named program from the disc and start execution of it.

<MAT statement> execution involves many disparate tasks. The forms of the <MAT statement> may be classified as array I/O, array assignment, array initialization, and the array functions TRN and INV. For conciseness in coding, all forms other than array I/O use some common program segments.

Array I/O prepares each array in the list in the same fashion. SBPTR is set to the dynamic dimensions of the array (base address -2) and the operator following the array identifier is picked up for examination. At this point MAT PRINT follows a separate path than MAT READ and MAT INPUT. The following operator is noted as spacing the elements (comma or end-of-statement) or packing them (semicolon). VCHK examines the array and generates an error if any of its elements have value 'undefined'. The dynamic row and column lengths are saved in 2's complement. If the MAT PRINT references a file, the array elements are written one by one in rows, each element in its two-word binary form. If the MAT PRINT references the teletype, rows are double spaced and the elements within a row are spaced or packed as noted above, each element in its ASCII decimal form. Both MAT READ and MAT INPUT redimension the array if the following operator is a left bracket (i.e., begins a matrix subscript). MCNT is set to the number of elements in the array, in 2's complement. MAT READ calls FDATA for element values while MAT INPUT transfers to the <INPUT STATEMENT> execution to obtain element values. MTØ acts as a flag for MAT INPUT, differentiating the first call for input from subsequent calls and saving the input character following the last element value used from the input record. After completing I/O on an array, a common section of code prepares the next array in the list or, if no more remain, terminates the

statement execution. MAT INPUT returns to the input code to clean up there, MAT PRINT and MAT READ return directly to the main execution loop.

Array assignment consists of preparing the destination and source arrays and executing a loop which assigns the destination array elements one by one. The general procedure is to assign a jump to the element computation code to MOP, an exit address to MEXIT to use after completing the destination array, and a count of the elements to MCNT, in 2's complement. The code to compute an element returns to MLOP1, MLOP2, or MLOP3 depending on the number of arrays involved which require updating of the element address. Each operation checks the dimensions of the arrays involved to insure that the operation is well-defined; and all elements of the source matrices are checked to make sure none have value 'undefined'. Matrix multiplication does not use the element computation loop, instead it uses row and column counters to tell when it is done and computes destination array elements by inner products of the rows and columns of its source matrices.

Array initialization also uses the element computation loop. The initialization program first redimensions the destination array (if a matrix subscript is given) and then chooses the appropriate constant for the element values. IDN acts like ZER except that it insists that the destination array be 'square' and sets a special counter to choose 1.Ø for the value of main diagonal elements.

TRN and INV are handled apart from the other matrix functions. For both of these, the elements of the source matrix are checked against the 'undefined value'. The source and destination matrices are then checked for transpositional compatibility. If TRN, then proceed to transfer the columns of the source matrix to the rows of the destination matrix.

INV uses the Gauss-Jordan algorithm with row pivoting. This procedure converts a copy of the source matrix into the identity matrix and converts an identity matrix into the inverse by applying the same set of operations to both. Since the source matrix is destroyed in the process, it is first copied into free user space and the copy treated thereafter as the source. A

side effect of the copying produces the element of largest absolute value, which is used to compute a lower bound on the allowable magnitude of pivot elements. INV then calls IDN to set the destination matrix to an identity matrix, having the side effect of checking that the matrix is square.

Diagonalization of the source matrix and production of the inverse now proceeds on a row-by-row basis. The next unreduced column of the source is searched for the pivot element (the largest in magnitude). If necessary, rows are swapped to put the pivot element on the main diagonal (the corresponding rows of the destination matrix must also be swapped). If the pivot element is smaller in magnitude than the previously computed lower bound, the matrix is too nearly singular to invert and execution is terminated. Otherwise, the pivot rows of both matrices are divided through by the pivot element. Now all other elements in the pivot column are eliminated by subtracting the appropriate multiple of the pivot row from each of the other rows. Advantage is taken of those pivot column elements which are already zero and of the fact that elements of the pivot row to the left of the pivot column have been set to zero by previous steps. After diagonalization of the source matrix and consequent creation of the inverse, the user space occupied by the source copy is released.

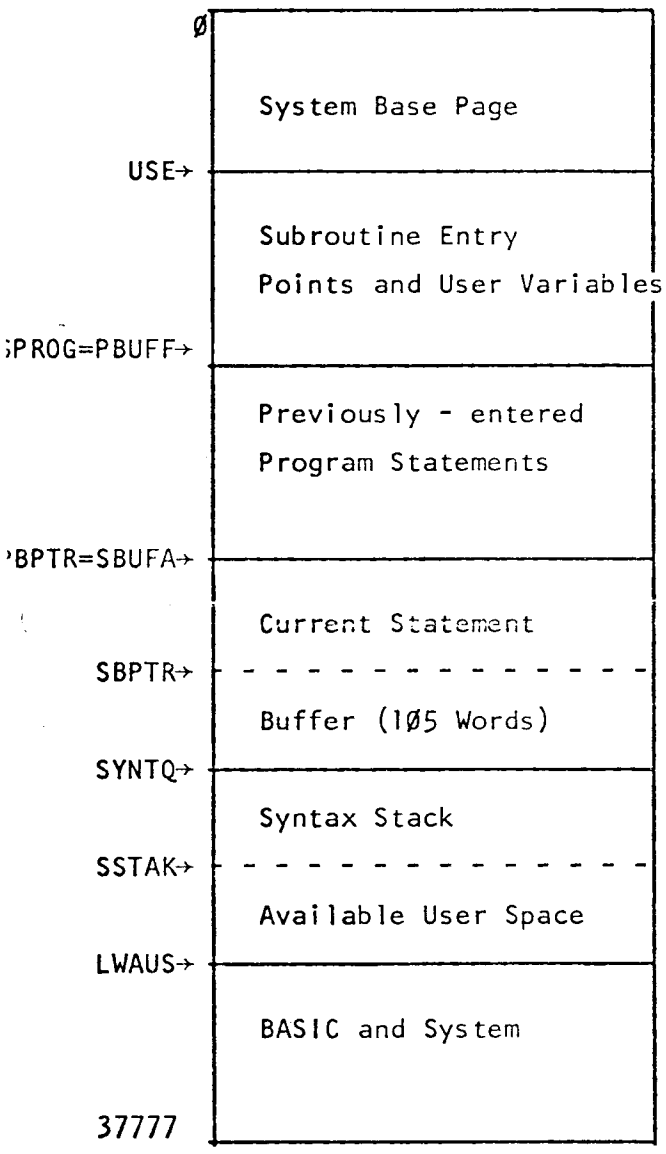
The other statement types are declarative in nature. Execution of them consists solely of skipping over to the statement following.

NOTES ON THE ERROR ROUTINES

Errors are handled routine SERR, reached by a jump through the base page table beginning at SERRS. A JSB SERRS + i,1 signifies detection of error i. The alternative bases RERRS and WERRS are conveniences to denote subsections of the table used for run-time errors and warning-only errors. The actions taken by SERR are explained in the listing; but notice that the 'BAD INPUT' error is singled out, its processing is completed by the input execution routine upon return from SERR.

Syntax errors detected while in tape mode are handled by accepting error psuedo-statements in place of the erroneous statements. Since these psuedo-statements will be replaced by any subsequently received statements with the same line number, provision is made in FNDPS, which returns the location of a statement when given its sequence number, to decrement the error counter (ERRCT) whenever the statement found is an error psuedo-statement (an error psuedo-statement will only be found by FNDPS when another statement with the same sequence number is ready to replace it). Over/underflows detected during number conversions in syntax mode cause warning messages to be issued only after accepting the statement, if it is otherwise correct. Since no printing can be done while in tape mode, the routine CHOUF suppresses setting of the flag and these potential errors are not reported when in tape mode.

SYNTAX (Phase 1)

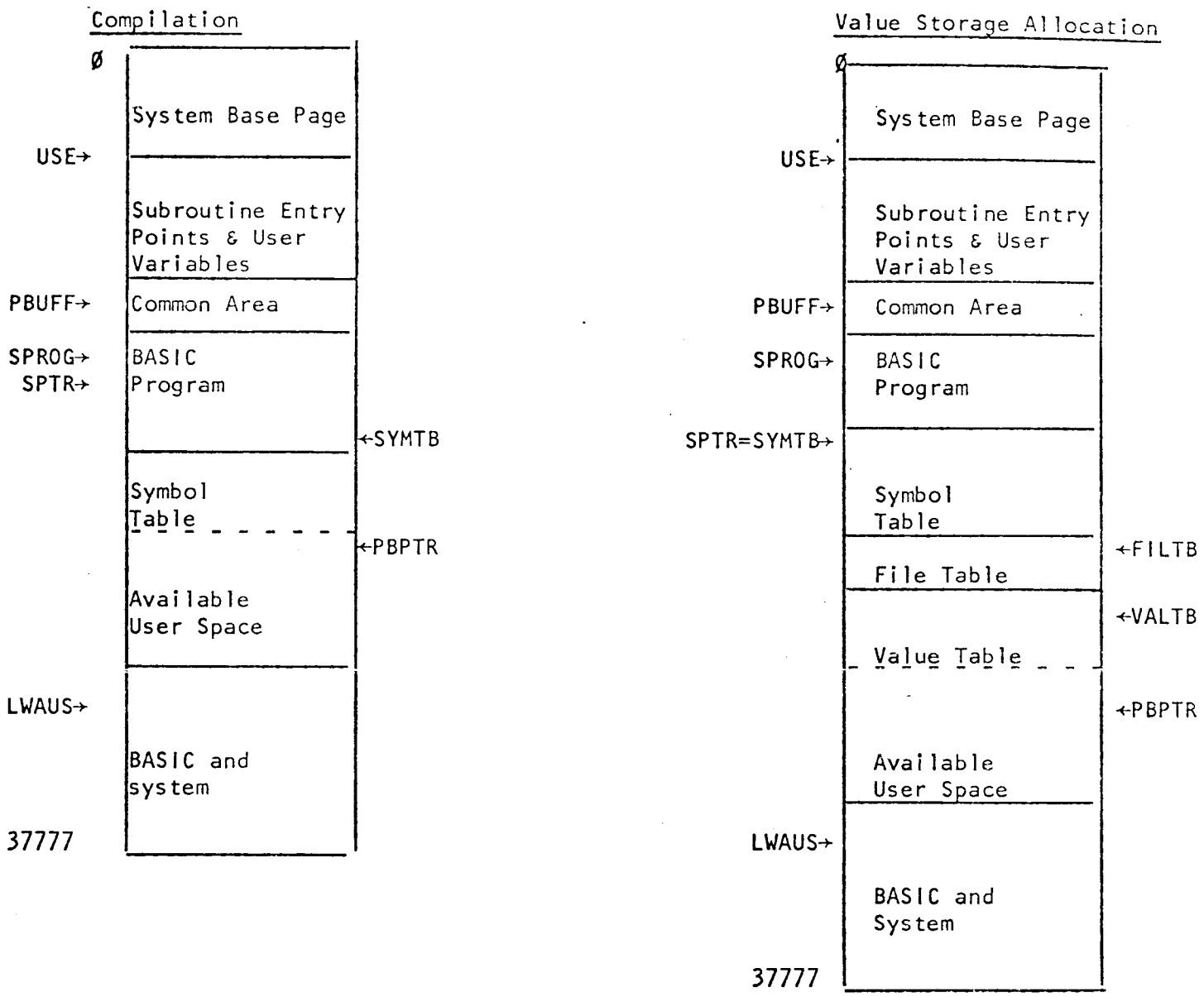


User Swap Area
(5440 Words)

Pointers

USE	Fixed, first word of user swap area.
PBUFF	Fixed, first word of program space.
SPROG	Fixed, first word of program.
SBUFA	Variable, first word of statement being syntaxed.
PBPTR	Variable, first word of program space not used by previously accepted program statements.
SBPTR	Variable, first word not used by statement being syntaxed.
SYNTZ	Variable, first word of syntax stack.
SSTAK	Variable, last word of syntax stack.
LWAUS	Fixed, first word not in user swap area.

COMPILATION (Phase II)



SPROG - Variable, first word of program

SYMTB - Variable, first word of symbol table.

SPTR - Variable, word of program being processed.

FILTB - Variable, first word of file table.

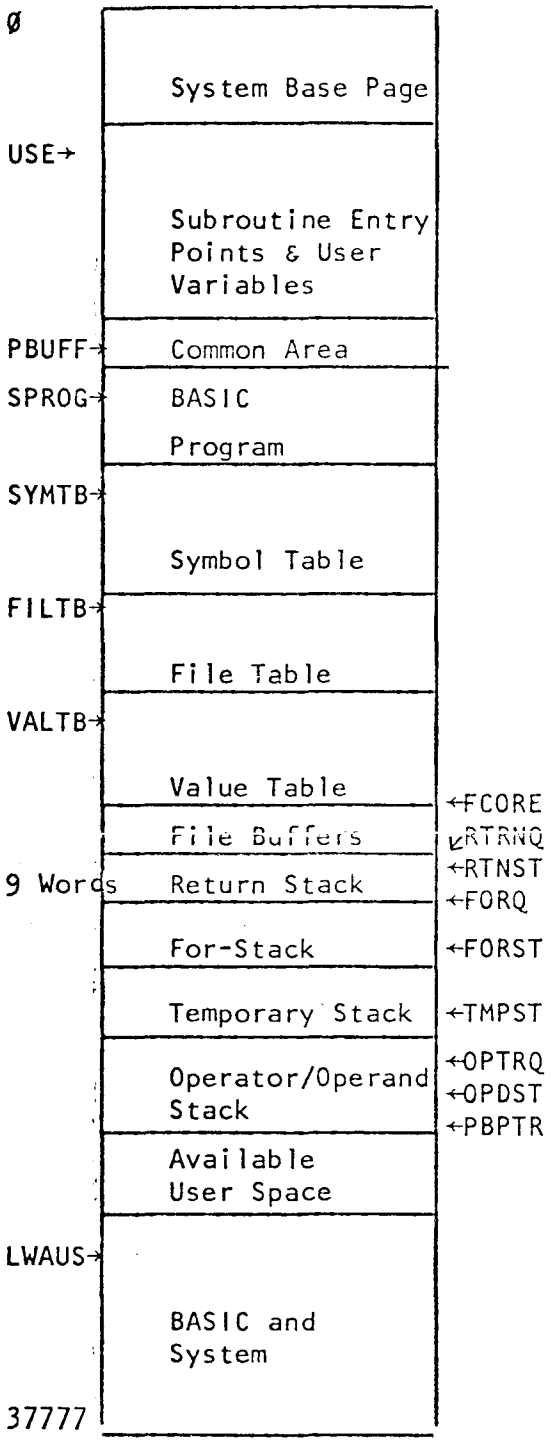
VALTB - Variable, first word of symbol value table
(FILTB = VALTB if no <FILES statement> is in program)

PBPTR - Variable, first word available of user space.

SYMTB and SPTR are not changed after compilation.

FILTB and VALTB are not changed after allocating value storage.

EXECUTION (Phase III)



FCORE - Variable, first word not used ' Phase II

RTRNQ - Variable, bottom of return stack (first word preceding return stac

RTNST - Variable, top of return stack

FORQ - Variable, bottom of for-stack (sixth word preceding for-stack)

FORST - Variable, top of for-stack (points to latest 6-word entry)

TMPST - Variable, top of temporary stack (points to latest 2-word entry)

OPTRQ - Variable, bottom of operator stac

OPDST - Variable, top of operand stack.

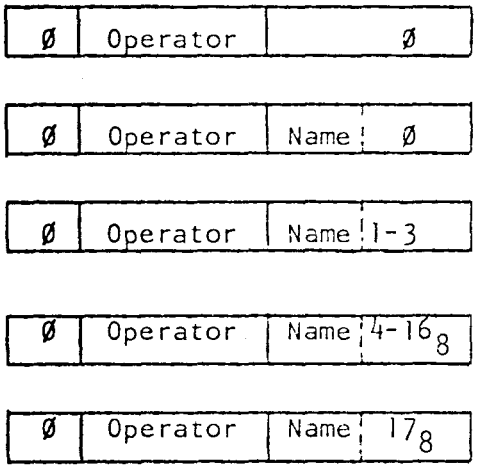
PBPTR - Variable, top of operator stack.

FCORE, RTRNQ, and FORQ are not changed after initiating execution.

Entries on the operator and operand stack are one word each and interleave (i.e., alternate words belong to one stack). All stacks beyond the return stack grow and shrink as needed so long as user space is available.

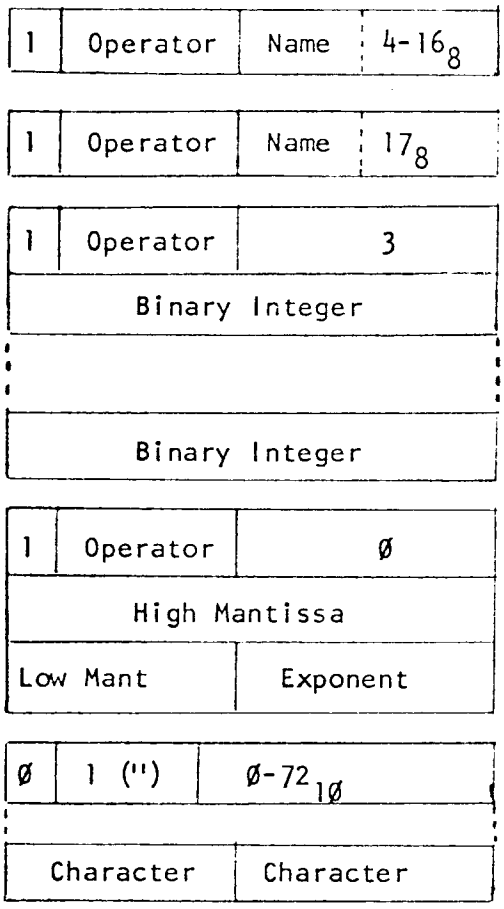
BASIC statements are represented internally by the sequence number followed by the length in words (including the sequence number and length words) followed by the statement body. The statement body is composed almost entirely of operator-operand pairs which occupy from one to three words each. Null operands and operators are used when necessary to maintain the operator-operand correspondence. The operator resides in bits 14-9 of a word; the operand uses bit 15, bits 8-Ø, and sometimes whole additional words immediately following.

'Variable' Operands



Bits 8-Ø are generally divided into two fields as follows: a name field (bits 8-4) and a type field (bits 3-Ø). The name field holds a value between 1 and 32_8 corresponding to A-Z (for functions, corresponding to FNA through FNZ). A type of Ø identifies a string variable (e.g. 3,Ø represents C$). Types 1 and 2 identify array variables of dimensionality one and two respectively (e.g. 4,2 represents D[*,*]) while type 3 identifies an array variable whose dimensionality cannot be determined by its immediate context. Type 4 identifies a simple variable with no digit (e.g. 1,4 represents A) while types $5\text{-}16_8$ identify simple variables whose names include the digit $Ø\text{-}9_{10}$ respectively (e.g. 6,7 represents F2). Type 17_8 identifies a programmer-defined function (e.g. 32_8, 17_8 represents FNZ).

'Constant' Operands



Parameter

Pre-defined Function

Formal Dimension/

Branch Address List

Numerical Constant

String Constant

A parameter (which can only appear inside a <DEF statement>) differs from a simple variable only in that bit 15 is set. The name of a pre-defined function may range, in the standard system, from 1 to 17_8 or 24_8 to 30_8 (TAB to TIM or ZER to TRN). A flagged (bit 15 set) operand of 3 identifies either a formal dimension in a <DIM statement> or <COM statement> (value in following word) or a branch address list (one or more statement sequence numbers in the following words). A flagged operand of Ø indicates that the following two words hold a floating-point constant (all numerical constants within a program are so represented). The operator with internal code 1 is ", which signals the start of a string constant. The operand portion of the word has a value from Ø to $72_{1Ø}$, indicating the number of characters in the constant. The string follows, two characters per word, and the closing " is not explicitly represented internally.

The table below gives the internal representation of the BASIC operators. Those operators which manipulate the formula evaluation stack during execution have associated priorities. All numbers are in octal notation.

BASIC Operators

CODE	PRIORITY	ASCII	CODE	PRIORITY	ASCII	CODE	ASCII
Ø	Ø	(end-of-formula)	26	5	<	54	FOR
1		"	27	5	#	55	NEXT
2		,	3Ø	5	=(equal)	56	GOSUB
3		;	31		(unused)	57	RETURN
4		#(file)	32		(unused)	6Ø	END
5		(unused)	33	4	AND	61	STOP
6		(unused)	34	3	OR	62	DATA
7		(unused)	35	6	MIN	63	INPUT
1Ø	1	)	36	6	MAX	64	READ
11	1	]	37	5	<>	65	PRINT
12	13(1)	[	40	5	>=	66	RESTORE
13	13(1)	(	41	5	<=	67	MAT
14	11	+(unary)	42	11	NOT	7Ø	FILES
15	11	-(unary)	43		(unused)	71	CHAIN
16	2	,(subscript)	44		(unused)	72	ENTER
17	2	=(assignment)	45		COM	73	'IMPLIED' LET
2Ø	7	+	46		LET	74	OF
21	7	-	47		DIM	75	THEN
22	1Ø	*	5Ø		DEF	76	TO
23	1Ø	/	51		REM	77	STEP
24	12	↑	52		GOTO		
25	5	>	53		IF		

Some examples of BASIC statements in their internal form are given below. Note that actual function parameter formulas, <DEF statements> formulas, and subscript formulas appearing in <MAT statements> require end-of-formula operators to signal their end whereas most formulas end either with the first operator which does not manipulate the formula evaluation stack or with the end of the statement. Note also that constants are considered signed only within a <DATA statement>. ASCII numbers are decimal, internal numbers are octal in the presentation below.

```
1Ø    LET W1 = Y = (B = C) ↑ 3*A[1,J+K]

   12                          sequence number
   21                          length
Ø | 46 | 27 | 6      LET W1
Ø | 17 | 31 | 4      =    Y
Ø | 17 |  . | Ø      =
Ø | 13 |  2 | 4      (    B
Ø | 3Ø |  3 | 4      =    C
Ø | 1Ø |    | Ø      )
1 | 24 |    | Ø      ↑
Ø3ØØØØ                3.Ø
ØØØØØ4
Ø | 22 |  1 | 2      *A
Ø | 12 | 11 | 4      [1
Ø | 16 | 12 | 4      ,J
Ø | 2Ø | 13 | 4      +K
Ø |  Ø |    | Ø      (end-of-formula)
Ø | 11 |    | Ø      ]
```

```
2Ø DIM A[5], C[6,12]

   24
   14
Ø | 47 |  1 | 1
1 | 12 |    | 3
        5
Ø | 11 |    | Ø
Ø |  2 |  3 | 2
1 | 12 |    | 3
        6
1 | 16 |    | 3
       14
Ø | 11 |    | Ø
```

```
30  DEF  FNC (X) = X + A0
```

```
      36
       7
0 | 50 |  3 | 17
1 | 13 | 30 |  4
0 | 10 |    |  0
1 | 17 | 30 |  4
  | 20 |  1 |  5
0 |  0 |    |  0
```

```
40  REM  ARK
```

```
      50
       5
0 | 51 | 40
 040522
 045400
```

```
50  GOTO  A  OF  10, 20, 30
```

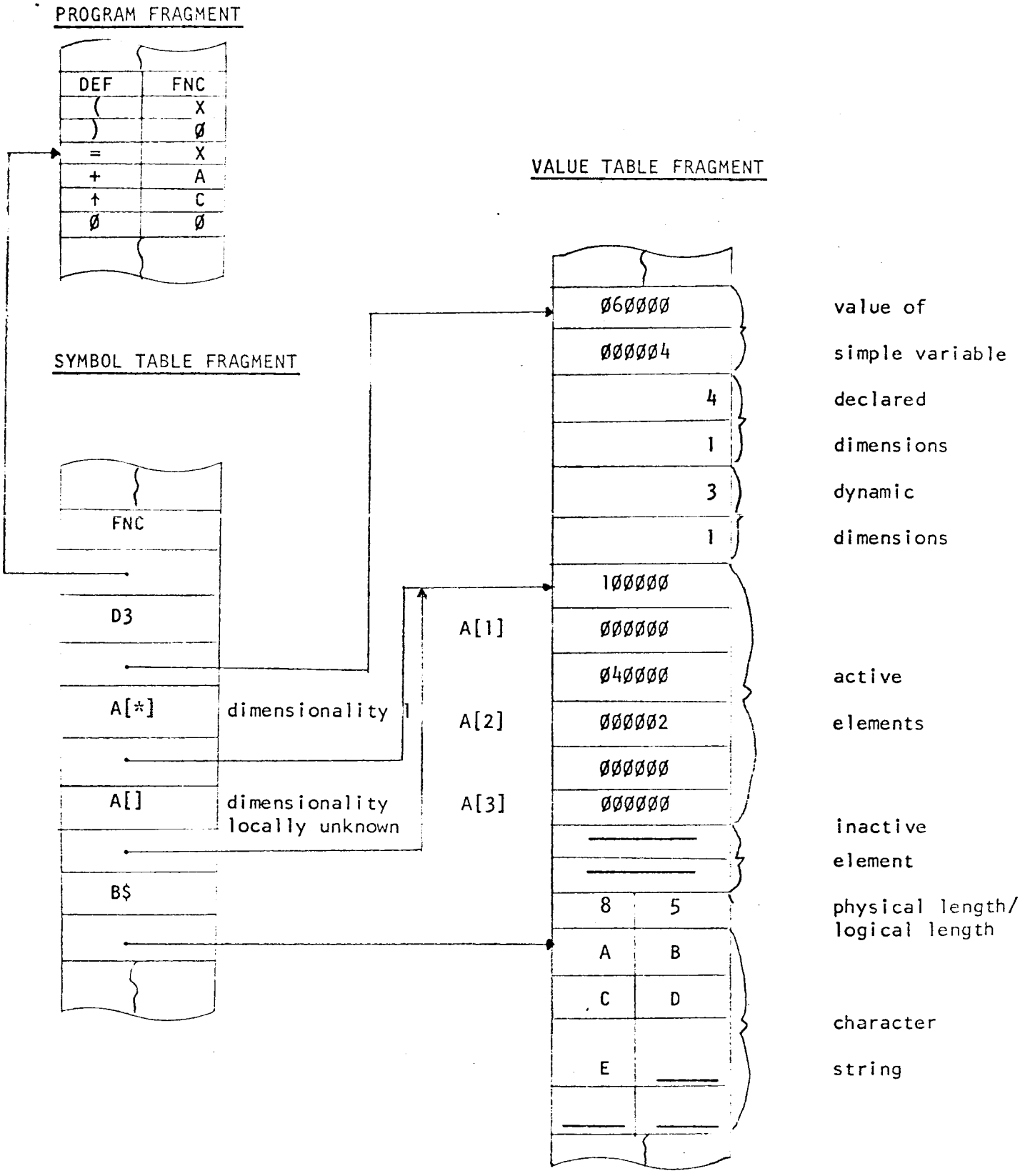
```
      62
       7
0 | 52 |  1 |  4
1 | 74 |    |  3
      12
      24
      36
```

```
60  DATA  -1, "ABC"
```

```
      74
      11
1 | 62 |    | 0
 100000
 000000
0 |  2 |    | 0
0 |  1 |    | 3
 040502
 041400
```

```
70 MAT READ #K;A[1]
```

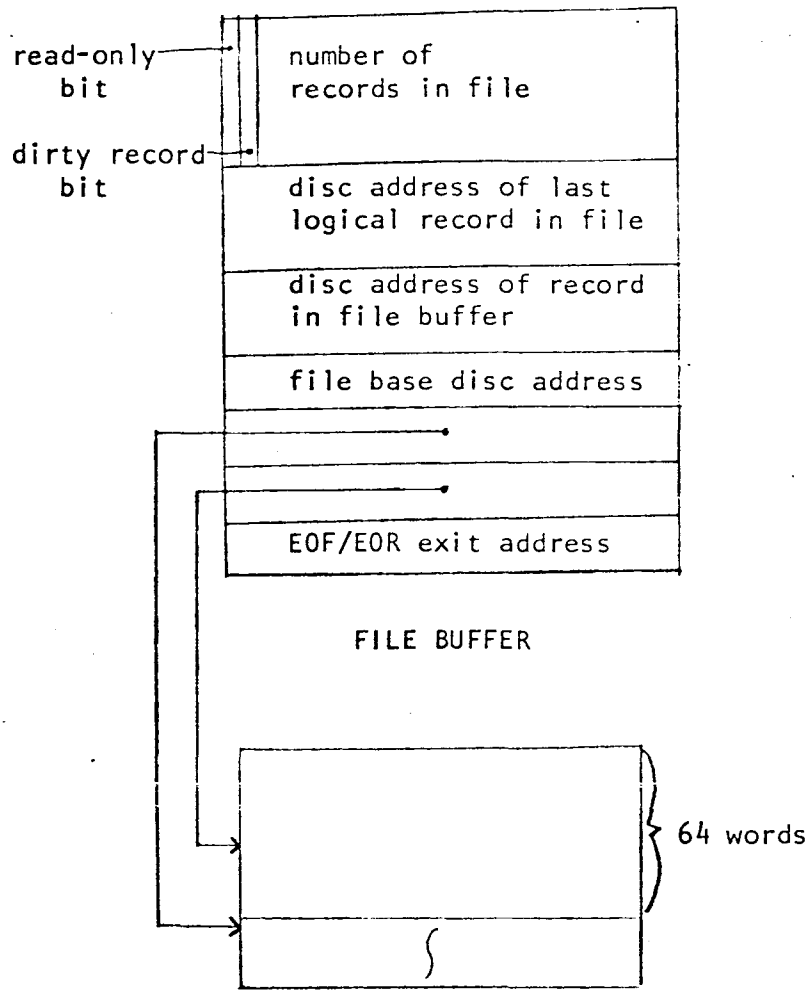
```
    106
     11
0 | 67 |    | 0
0 | 64 |    | 0
0 |  4 | 13 | 4
0 |  3 |  1 | 1
0 | 12 | 11 | 4
0 |    |    | 0
0 | 11 |    | 0
```


PROGRAM FRAGMENT
DEF FNC
(X
) Ø
= X
+ A
↑ C
Ø Ø
SYMBOL TABLE FRAGMENT
FNC
D3
A[*]
dimensionality
A[]
dimensionality
locally unknown
B$
VALUE TABLE FRAGMENT
Ø6ØØØØ
ØØØØØ4
4
1
3
1
1ØØØØØ
ØØØØØØ
Ø4ØØØØ
ØØØØØ2
ØØØØØØ
ØØØØØØ
8 5
A B
C D
E
A[1]
A[2]
A[3]
value of
simple variable
declared
dimensions
dynamic
dimensions
active
elements
inactive
element
physical length/
logical length
character
string

The symbol table consists of two-word entries, one for each unique symbol occurri in the user's program. The first word of an entry is the internal representation of the symbol as previously described. The second word of the entry is a pointer to th value of the symbol. For a programmer-defined function the value is the defining formula in the <DEF statement>. The value of a simple variable is a two-word floating point number. The value pointer of an array is its base address (i.e. the address of its first element); when an array is dynamically redimensioned to occupy less than its physically allocated storage, it occupies a contiguous block justified to the low core portion of its element space. Since array symbols may not have dimensionality locally defined (e.g. MAT A=B), array symbols may have a "don't know" entry in the symbol table in addition to the dimensioned entry. Both entries have the same value pointer. The declared and dynamic dimensions occupy the four words preceding the element space in the value table. The value of a string is also its base address. A string is a character array (packed two elements per word in contra to the two words per element for numerical arrays). Its physical (declared) length and logical (dynamic) length occupy the word immediately preceding its value space.

The value table and common area are simply the concatenation of the values for the symbols in the program, excepting programmer-defined functions.

FILE TABLE ENTRY

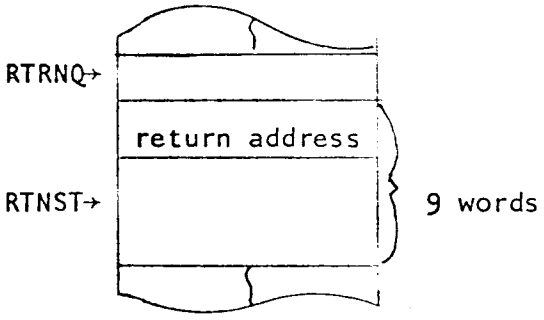


The file table consists of one seven-word entry for each file in the <FILES statement>. Bit 15 of the first word is set if the file was busy when requested or i a public file (available on a rea only basis). Bit 14 of the first word is set when an item is store in the buffer, so that only recor which are changed will be written back to the disc. A 64-word buffer is associated with each fi entry and is accessed through pointers in its file entry. An intr-record pointer designates th next portion of the record to be written or read. A fixed poi to the first word not in the buffer acts as a bound on the intra-buffer pointer.

FILE CONTENTS

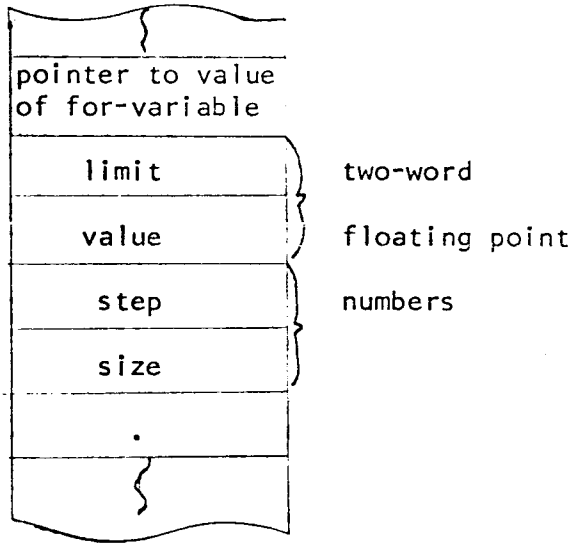
There are 4 data types possible in a file. A string has bit 9 = 1 and the length in characters in the lowest 7 bits of the first word, followed by the string packed 2 characters per word. A two-word floating point number has the upper two bits of the first word different, except for a zero, which has both words zero. An end-of-file is a -1, and an end-of-record is a -2, in the first word.

Return Stack



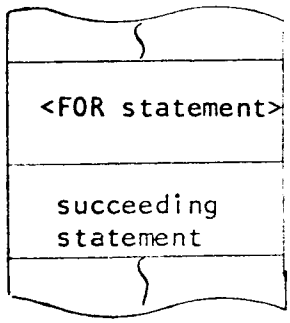
The return stack is of fixed length, holding from Ø to 9 one-word entries at any time. An entry is the absolute address of the statement following the GOSUB which placed the entry on the stack.

For-Stack Entry

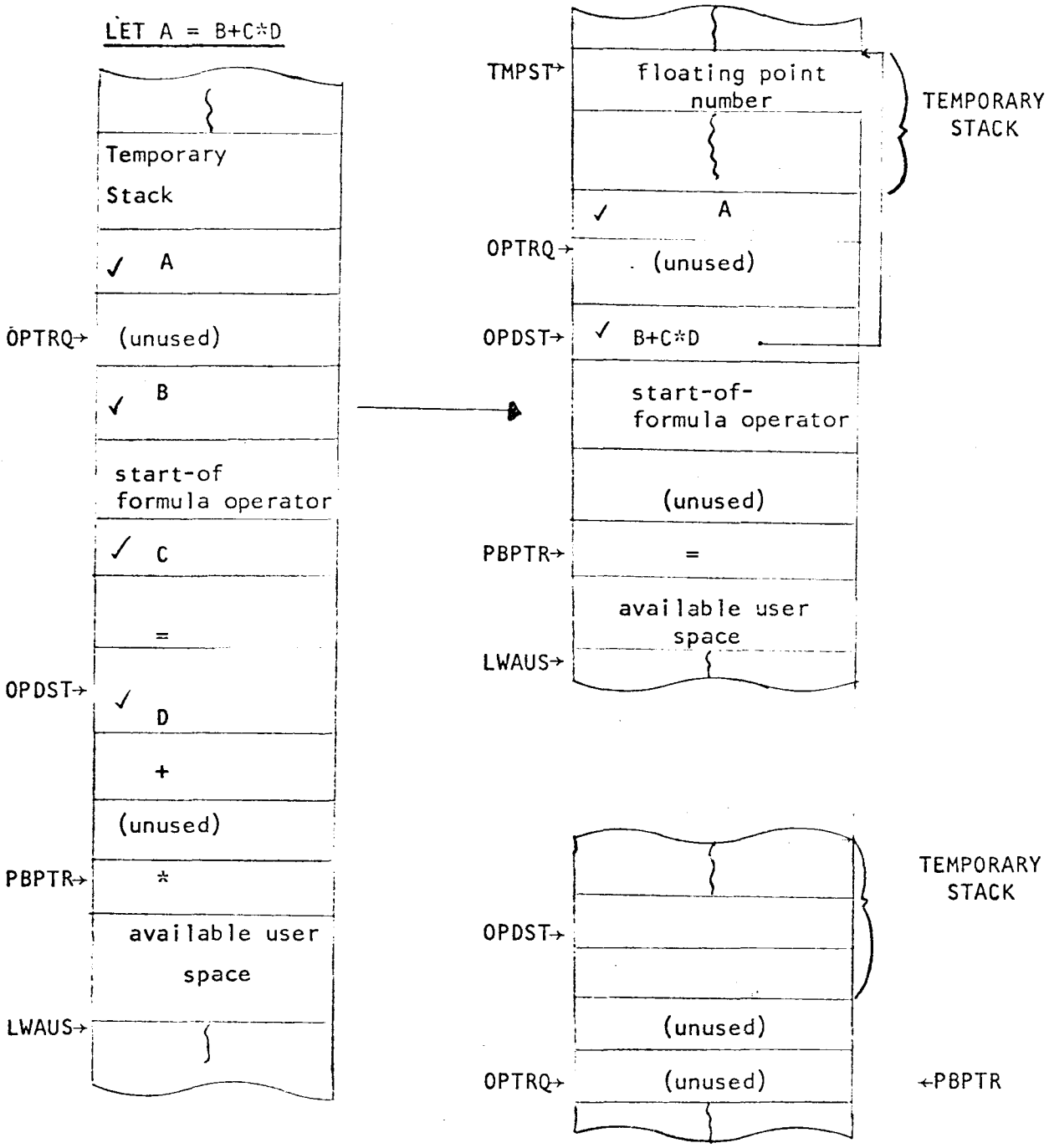


The for-stack is of variable length, containing one six-word entry for each for-loop which is currently active. Since the limit value and step size are kept in the entry, they may not be changed within the for-loop. The value of the for-variable is the one kept in the value table, so this may be altered by statements within the for-loop.

Program Fragment



OPERATOR/OPERAND STACK FRAGMENTS



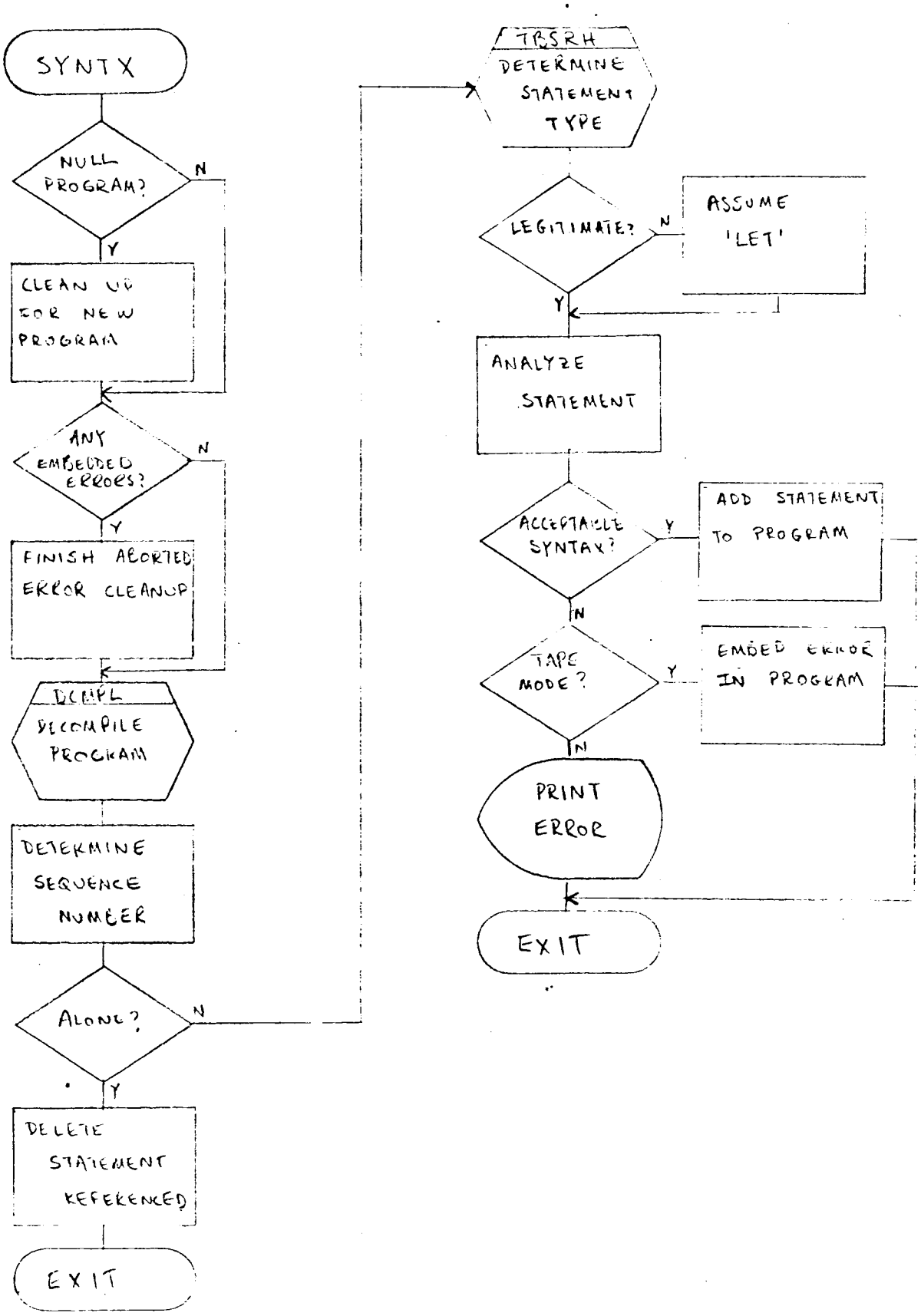
All operands (checked words) are addresses (i.e., C represents a pointer to the value of the simple variable C). Bits 7 - Ø of an operator entry contain the operators identifying code (See 'Basic Operators' Table) while bits 15-8 contain the operator's priority. Note the alternate-word structure of the stacks. The temporary stack holds intermediate values during the formula evaluation.

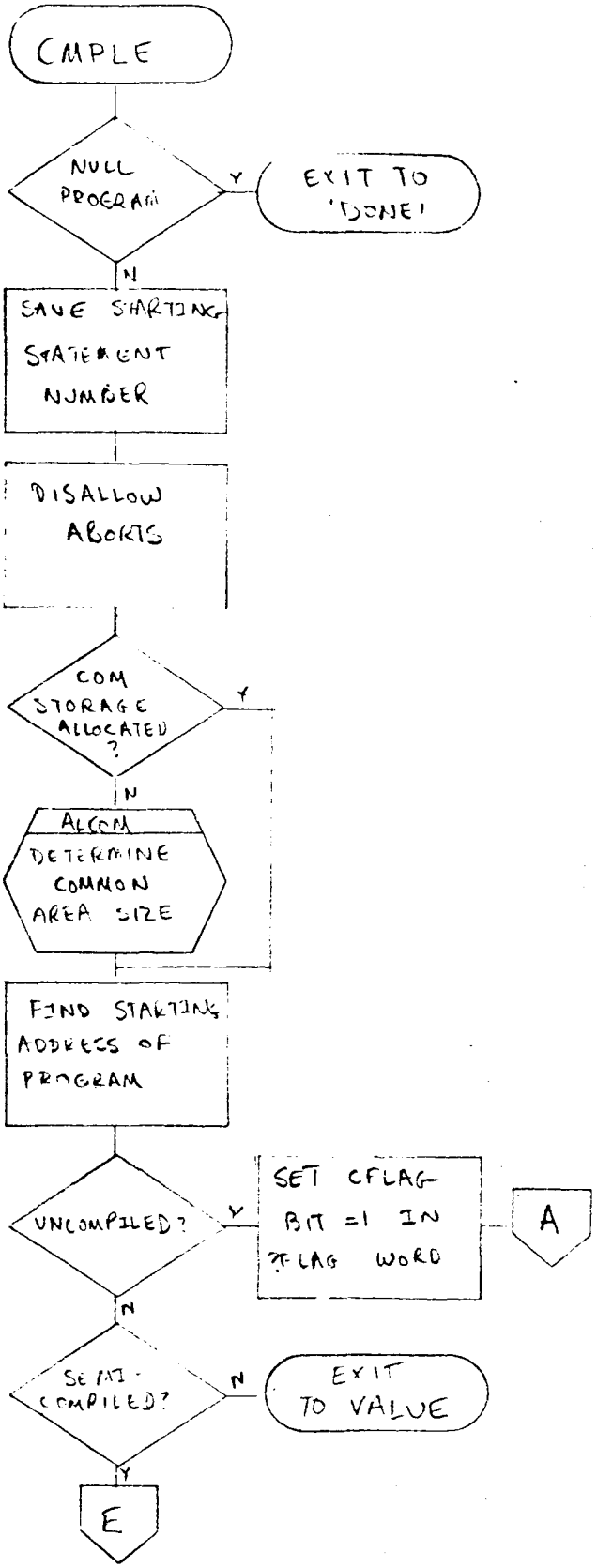
The two areas of core labelled SBJTB and USER contain the mechanism allowing different users to exercise different portions of the language processor without interference. The language processor makes its subroutine calls to the labels in the area beginning with USER. The word following a subroutine entry point is an indirect jump through the appropriate address in the area following SBJTB. When a user is displaced by the system, his registers are saved at USER and the area of core from USER to PBPTR,I inclusive is dumped onto his track of the disc. Thus, a complete record of the language processor's status with respect to him is preserved. The only thing particular to a user which remains when he is swapped out is his own teletype table.

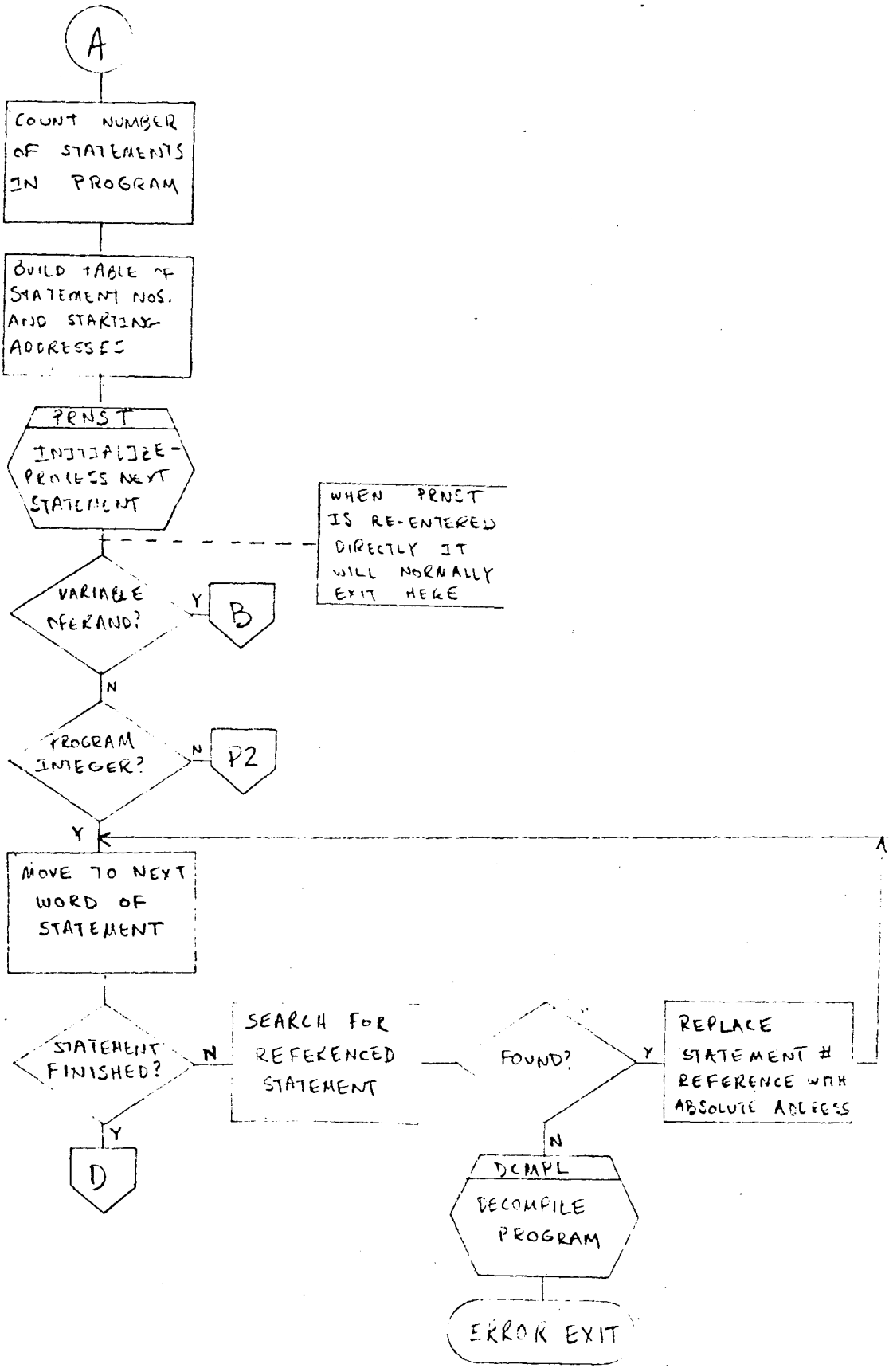
The tables headed by PDFTB (which must be in base page), SYNTB, XECTB, and FOJT are jump tables. The method in the last three cases is to compute a decision number, add the base address of the table, and transfer through the entry thus designated. The pre-defined function table is used by the formula evaluator to enter the code for evaluating pre-defined functions.

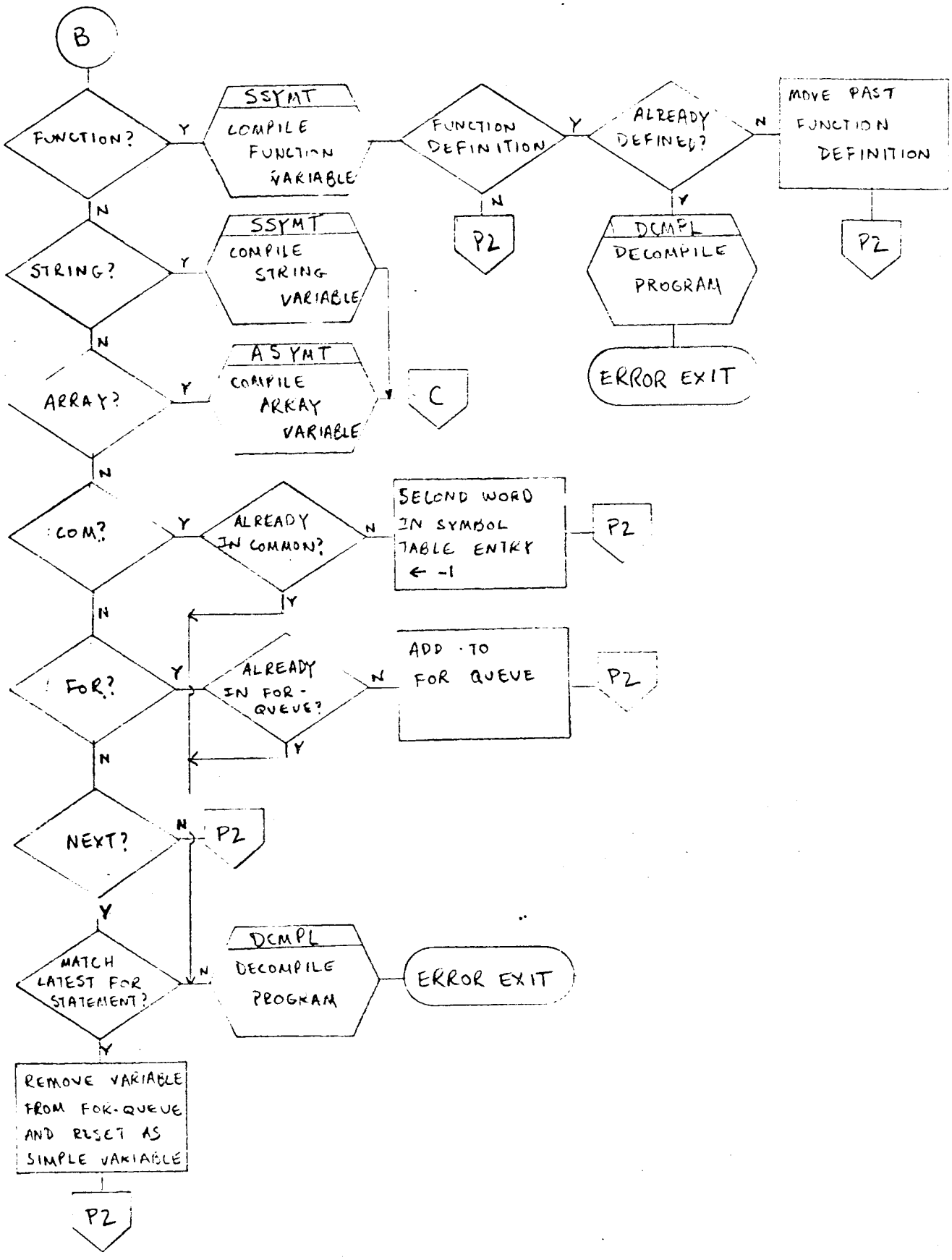
The tables headed by QUOTE and MCBOS have several uses. Their entries are explained in the listing and their use will be explained in thos routines which access them. The Error Jump Table (at SERRS) is explained along with the error routines.

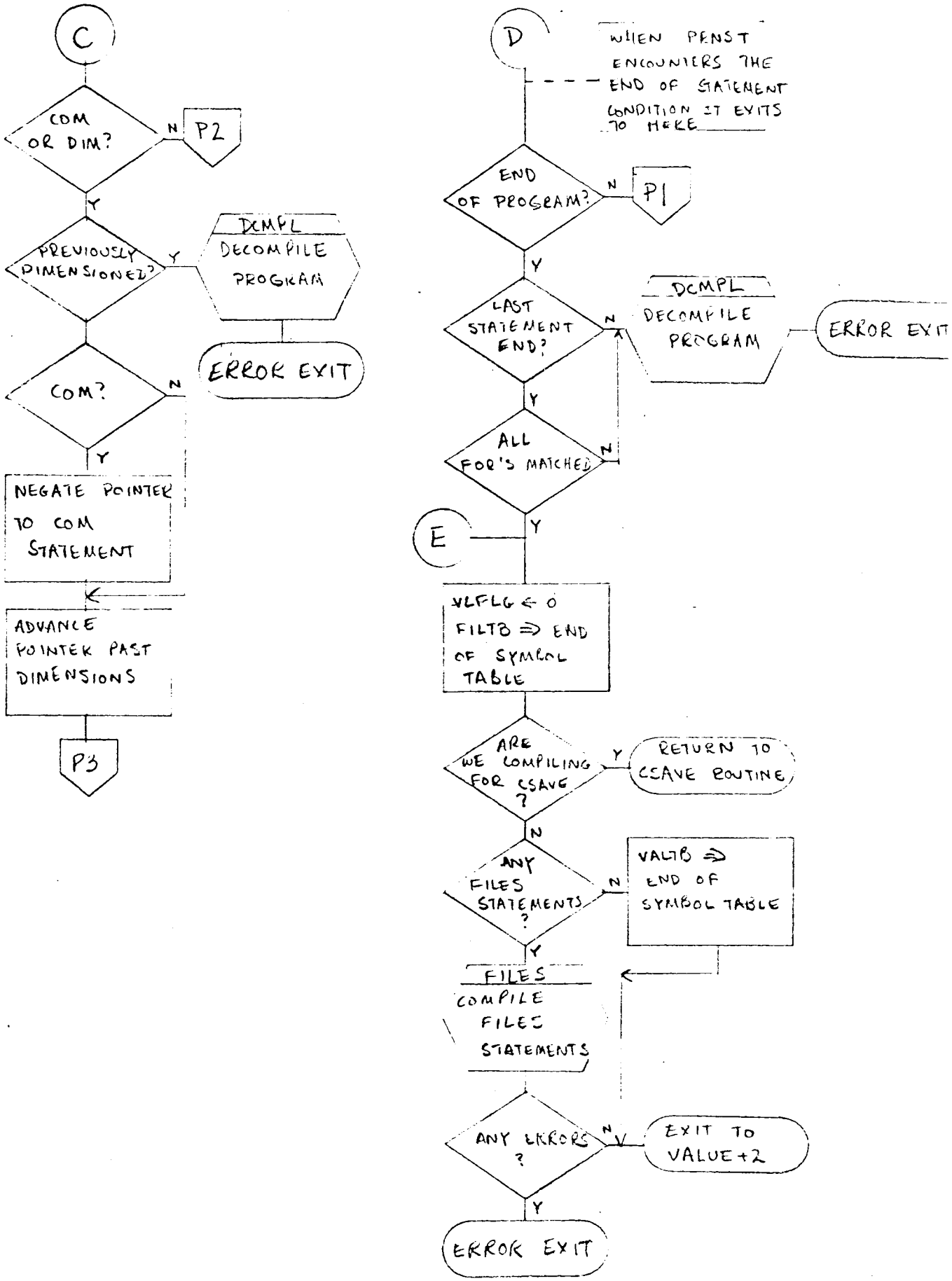
SYNTAX


SYNTX
NULL PROGRAM?
N
Y
CLEAN UP FOR NEW PROGRAM
ANY EMBEDDED ERRORS?
N
Y
FINISH ABORTED ERROR CLEANUP
DCMPL
DECOMPILE PROGRAM
DETERMINE SEQUENCE NUMBER
ALONE?
N
Y
DELETE STATEMENT REFERENCED
EXIT
TRSRH
DETERMINE STATEMENT TYPE
LEGITIMATE?
N
ASSUME 'LET'
Y
ANALYZE STATEMENT
ACCEPTABLE SYNTAX?
Y
ADD STATEMENT TO PROGRAM
N
TAPE MODE?
Y
EMBED ERROR IN PROGRAM
N
PRINT ERROR
EXIT

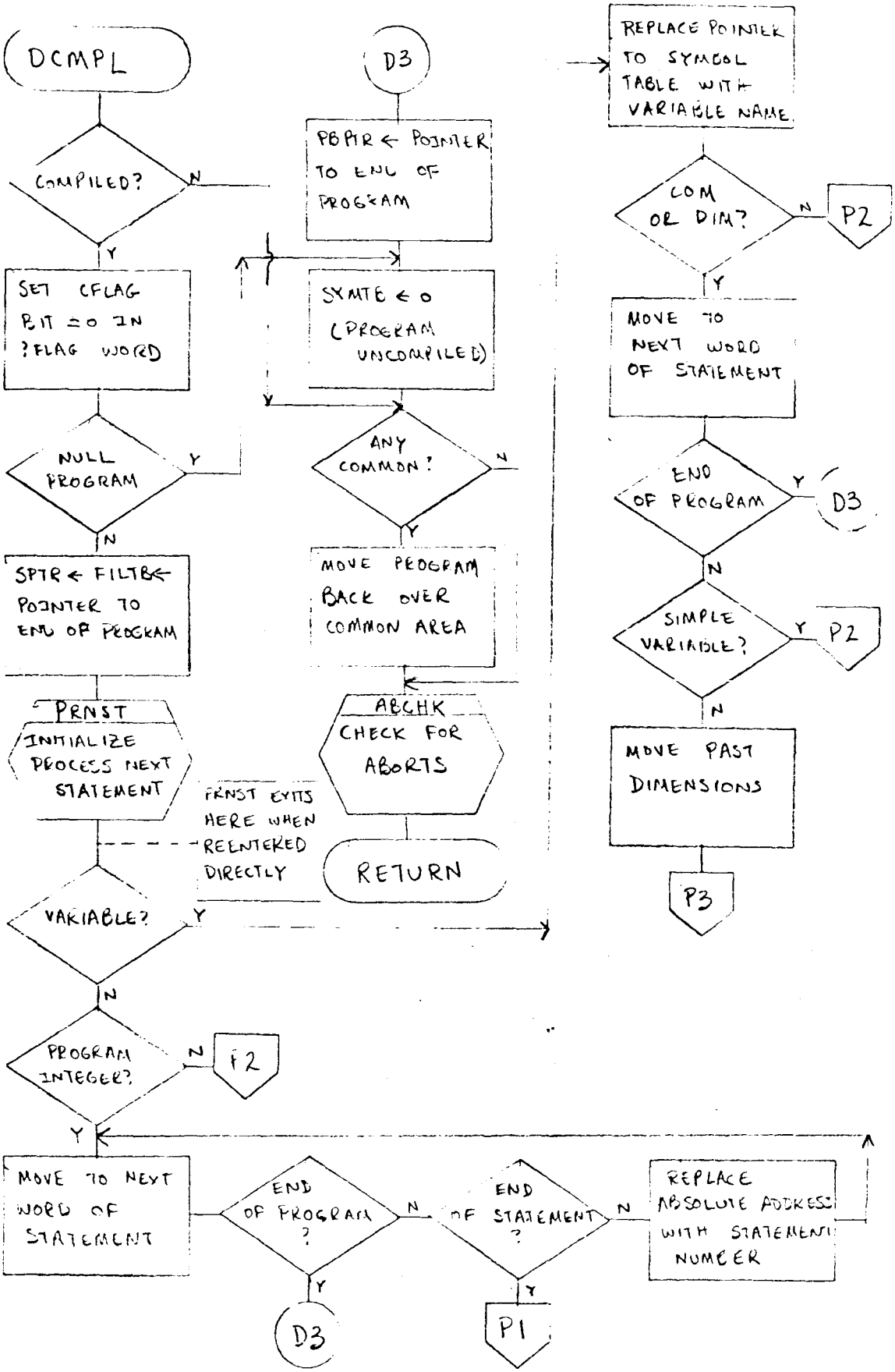

CMPLE
NULL PROGRAM
Y
EXIT TO 'DONE'
N
SAVE STARTING STATEMENT NUMBER
DISALLOW ABORTS
COM STORAGE ALLOCATED ?
Y
N
ALCOM
DETERMINE COMMON AREA SIZE
FIND STARTING ADDRESS OF PROGRAM
UNCOMPILED?
Y
SET CFLAG BIT =1 IN FLAG WORD
A
N
SEMI-COMPILED?
N
EXIT TO VALUE
Y
E

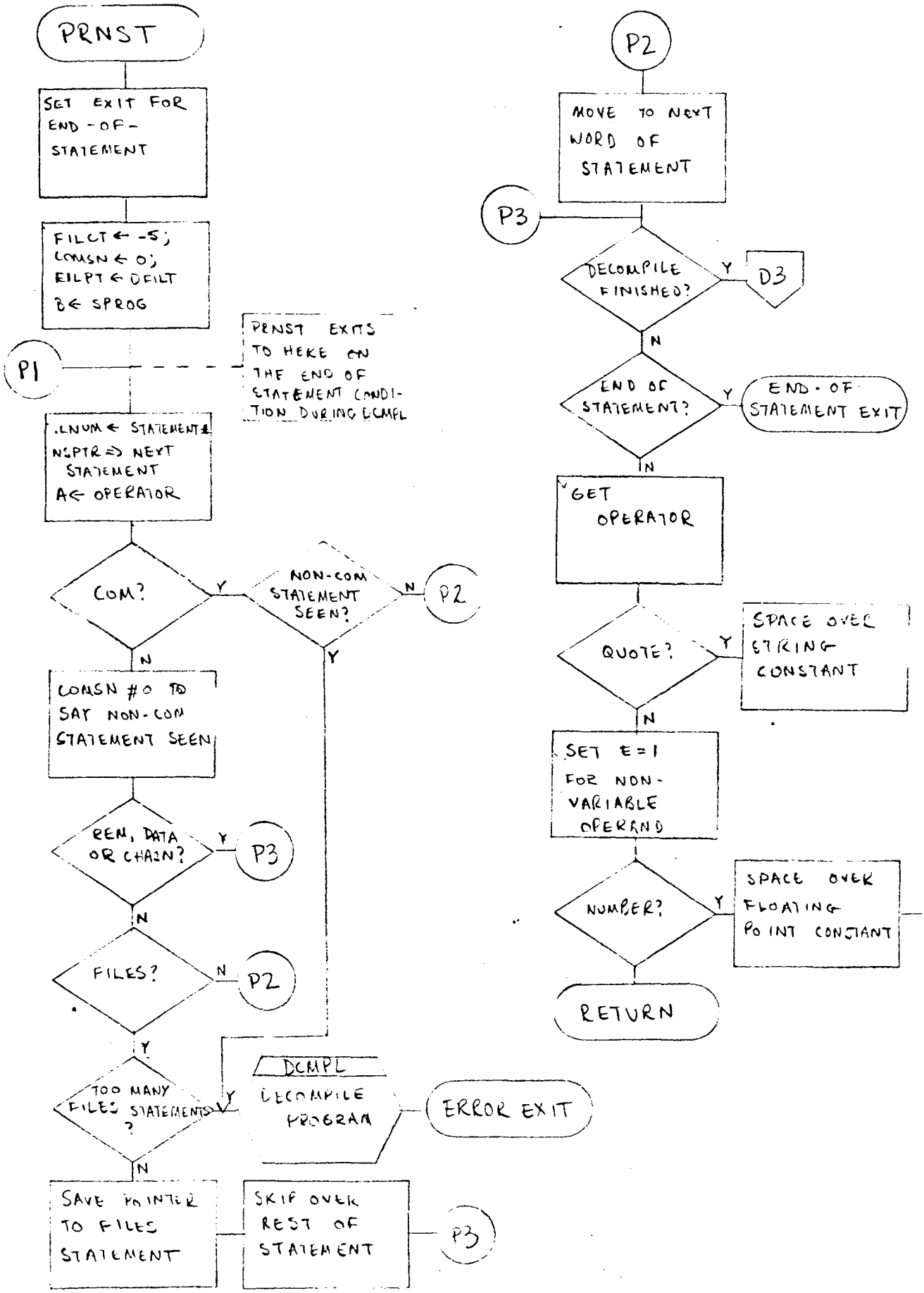

A
COUNT NUMBER OF STATEMENTS IN PROGRAM
BUILD TABLE OF STATEMENT NOS. AND STARTING ADDRESSES
PRNST
INITIALIZE - PROCESS NEXT STATEMENT
WHEN PRNST IS RE-ENTERED DIRECTLY IT WILL NORMALLY EXIT HERE
VARIABLE OPERAND?
Y
B
N
PROGRAM INTEGER?
N
P2
Y
MOVE TO NEXT WORD OF STATEMENT
STATEMENT FINISHED?
N
SEARCH FOR REFERENCED STATEMENT
FOUND?
Y
REPLACE STATEMENT # REFERENCE WITH ABSOLUTE ADDRESS
Y
D
N
DCMPL
DECOMPILE PROGRAM
ERROR EXIT

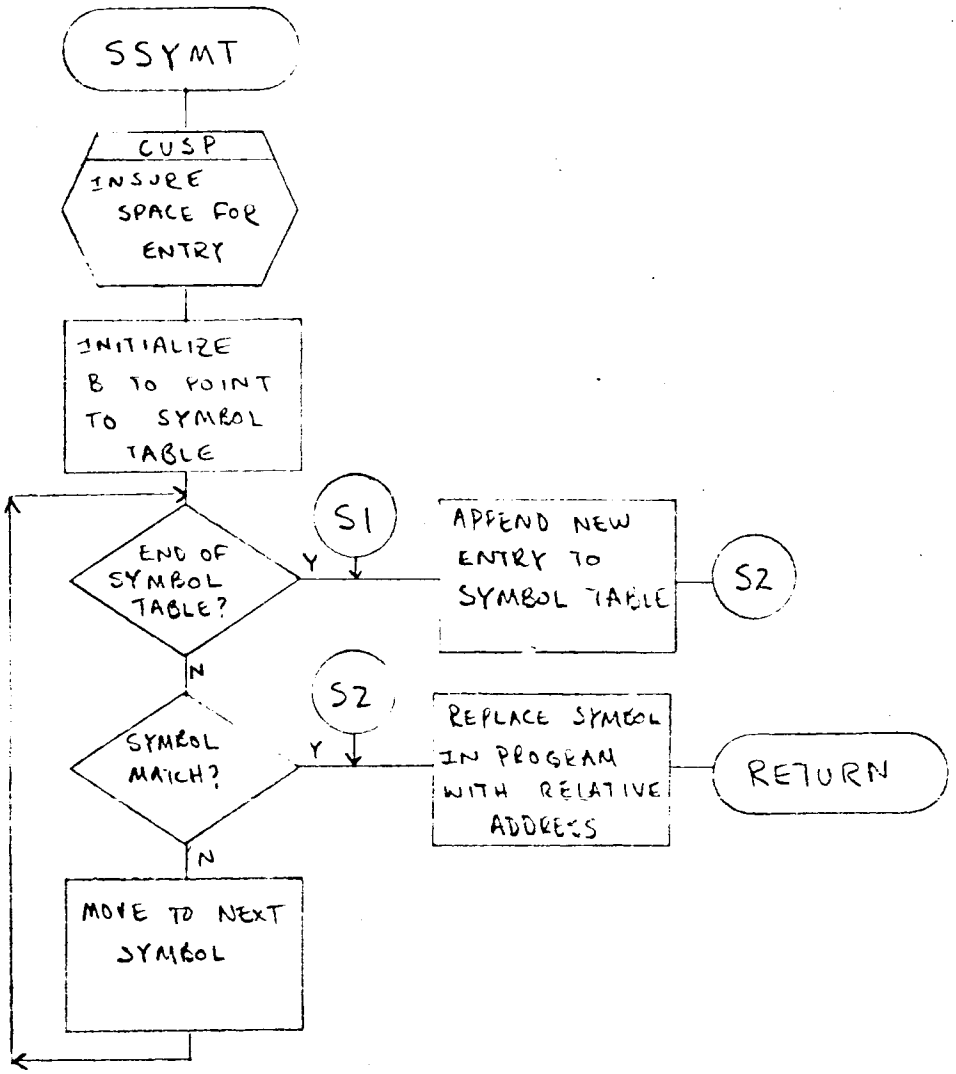

B
FUNCTION?
Y
SSYMT
COMPILE FUNCTION VARIABLE
FUNCTION DEFINITION
Y
ALREADY DEFINED?
N
MOVE PAST FUNCTION DEFINITION
P2
N
P2
Y
DCMPL
DECOMPILE PROGRAM
ERROR EXIT
N
STRING?
Y
SSYMT
COMPILE STRING VARIABLE
N
ARRAY?
Y
ASYMT
COMPILE ARRAY VARIABLE
C
N
COM?
Y
ALREADY IN COMMON?
N
SECOND WORD IN SYMBOL TABLE ENTRY ← -1
P2
Y
N
FOR?
Y
ALREADY IN FOR-QUEUE?
N
ADD TO FOR QUEUE
P2
Y
N
NEXT?
N
P2
Y
MATCH LATEST FOR STATEMENT?
N
DCMPL
DECOMPILE PROGRAM
ERROR EXIT
Y
REMOVE VARIABLE FROM FOR-QUEUE AND RESET AS SIMPLE VARIABLE
P2


C
COM OR DIM?
N
P2
Y
PREVIOUSLY DIMENSIONED?
Y
DCMPL
DECOMPILE PROGRAM
ERROR EXIT
COM?
N
Y
NEGATE POINTER TO COM STATEMENT
ADVANCE POINTER PAST DIMENSIONS
P3
D
WHEN PRNST ENCOUNTERS THE END OF STATEMENT CONDITION IT EXITS TO HERE
END OF PROGRAM?
N
P1
Y
LAST STATEMENT END?
N
DCMPL
DECOMPILE PROGRAM
ERROR EXIT
Y
ALL FOR'S MATCHED
N
Y
E
VLFLG ← 0
FILTB ⇒ END OF SYMBOL TABLE
ARE WE COMPILING FOR CSAVE ?
Y
RETURN TO CSAVE ROUTINE
N
ANY FILES STATEMENTS ?
N
VALTB ⇒ END OF SYMBOL TABLE
Y
FILES
COMPILE FILES STATEMENTS
ANY ERRORS ?
N
Y
EXIT TO VALUE+2
Y
ERROR EXIT

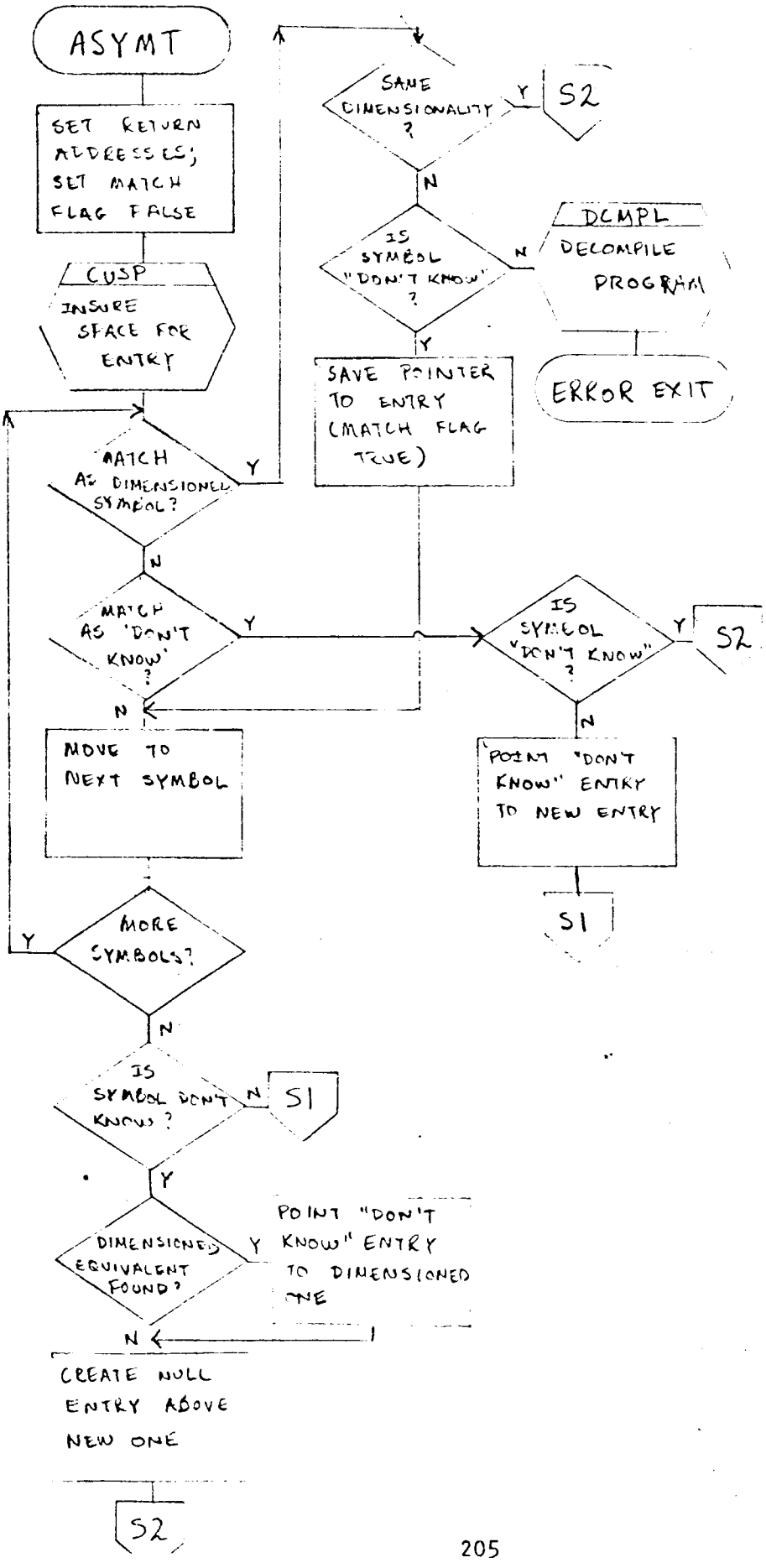
DECOMPILATION



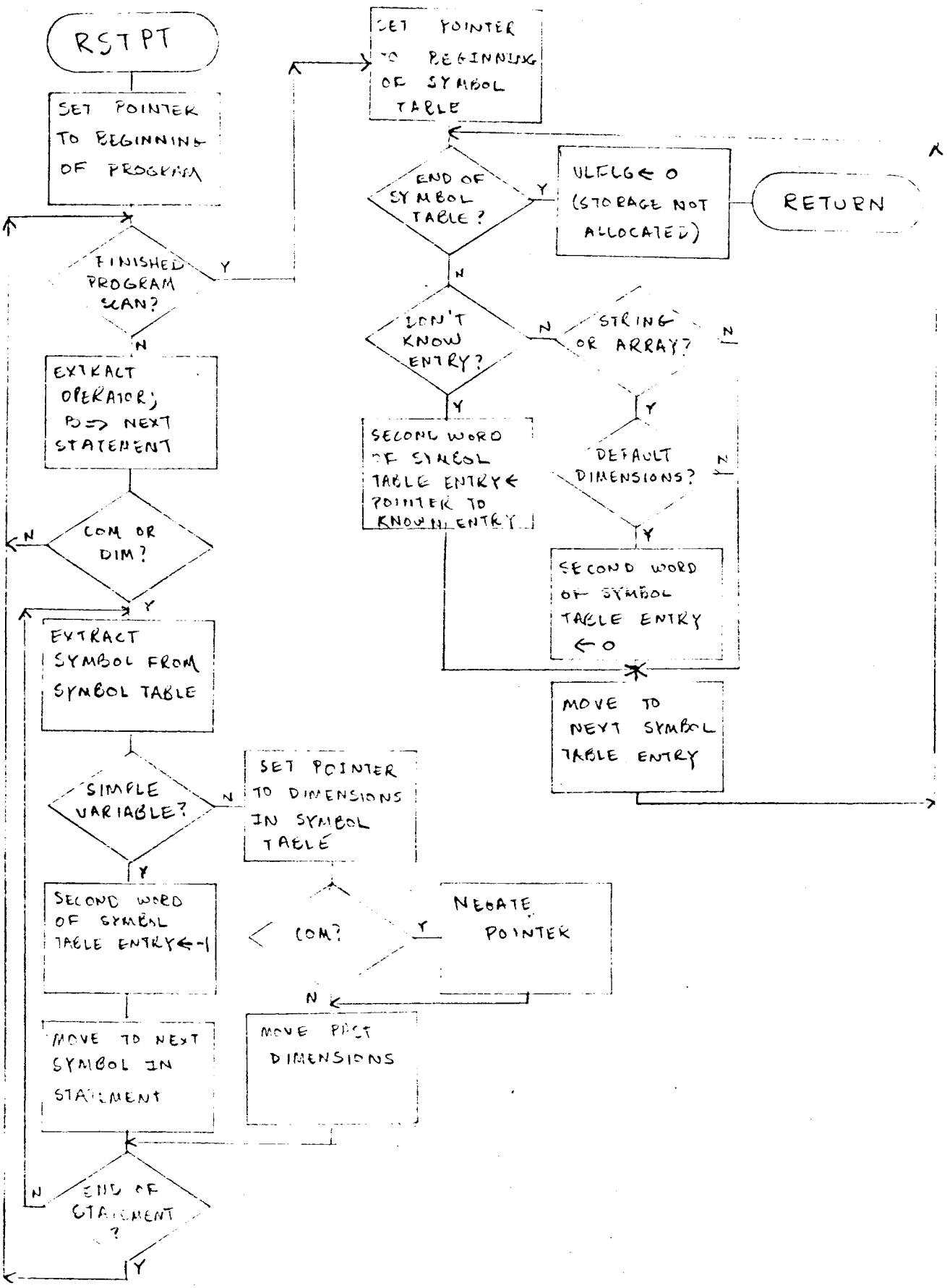

PRNST
SET EXIT FOR END-OF-STATEMENT
FILCT ← -5;
CONSN ← 0;
FILPT ← DFILT
B ← SPROG
P1
PRNST EXITS TO HERE ON THE END OF STATEMENT CONDITION DURING DCMPL
.LNUM ← STATEMENT#
NSPTR ⇒ NEXT STATEMENT
A ← OPERATOR
COM?
Y
NON-COM STATEMENT SEEN?
N
P2
Y
N
CONSN ≠ 0 TO SAY NON-COM STATEMENT SEEN
REM, DATA OR CHAIN?
Y
P3
N
FILES?
N
P2
Y
TOO MANY FILES STATEMENTS?
Y
DCMPL
DECOMPILE PROGRAM
ERROR EXIT
N
SAVE POINTER TO FILES STATEMENT
SKIP OVER REST OF STATEMENT
P3
P2
MOVE TO NEXT WORD OF STATEMENT
P3
DECOMPILE FINISHED?
Y
D3
N
END OF STATEMENT?
Y
END-OF-STATEMENT EXIT
N
GET OPERATOR
QUOTE?
Y
SPACE OVER STRING CONSTANT
N
SET E=1 FOR NON-VARIABLE OPERAND
NUMBER?
Y
SPACE OVER FLOATING POINT CONSTANT
RETURN

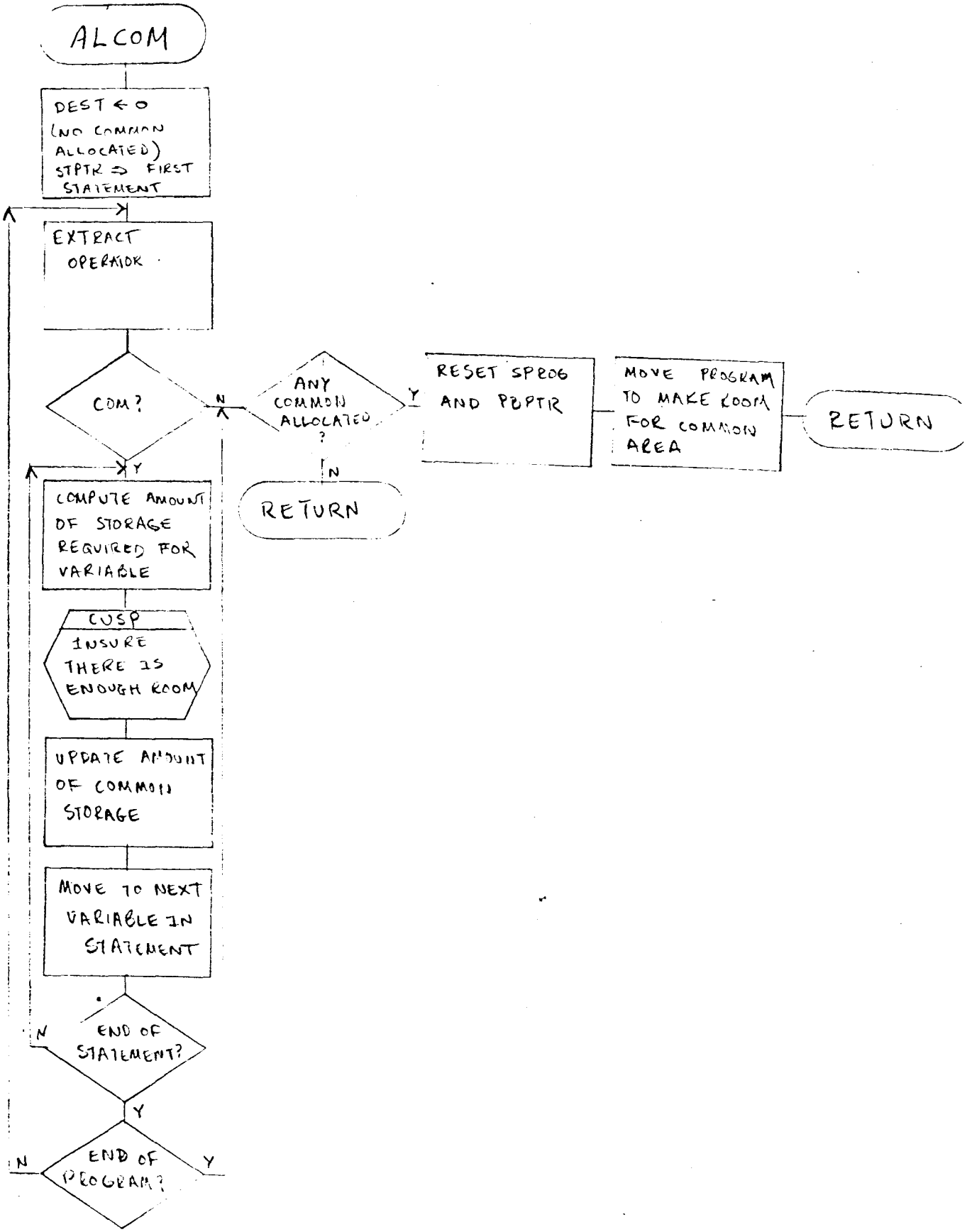

SSYMT
CUSP
INSURE SPACE FOR ENTRY
INITIALIZE B TO POINT TO SYMBOL TABLE
END OF SYMBOL TABLE?
Y
S1
APPEND NEW ENTRY TO SYMBOL TABLE
S2
N
SYMBOL MATCH?
Y
S2
REPLACE SYMBOL IN PROGRAM WITH RELATIVE ADDRESS
RETURN
N
MOVE TO NEXT SYMBOL

ASYMT

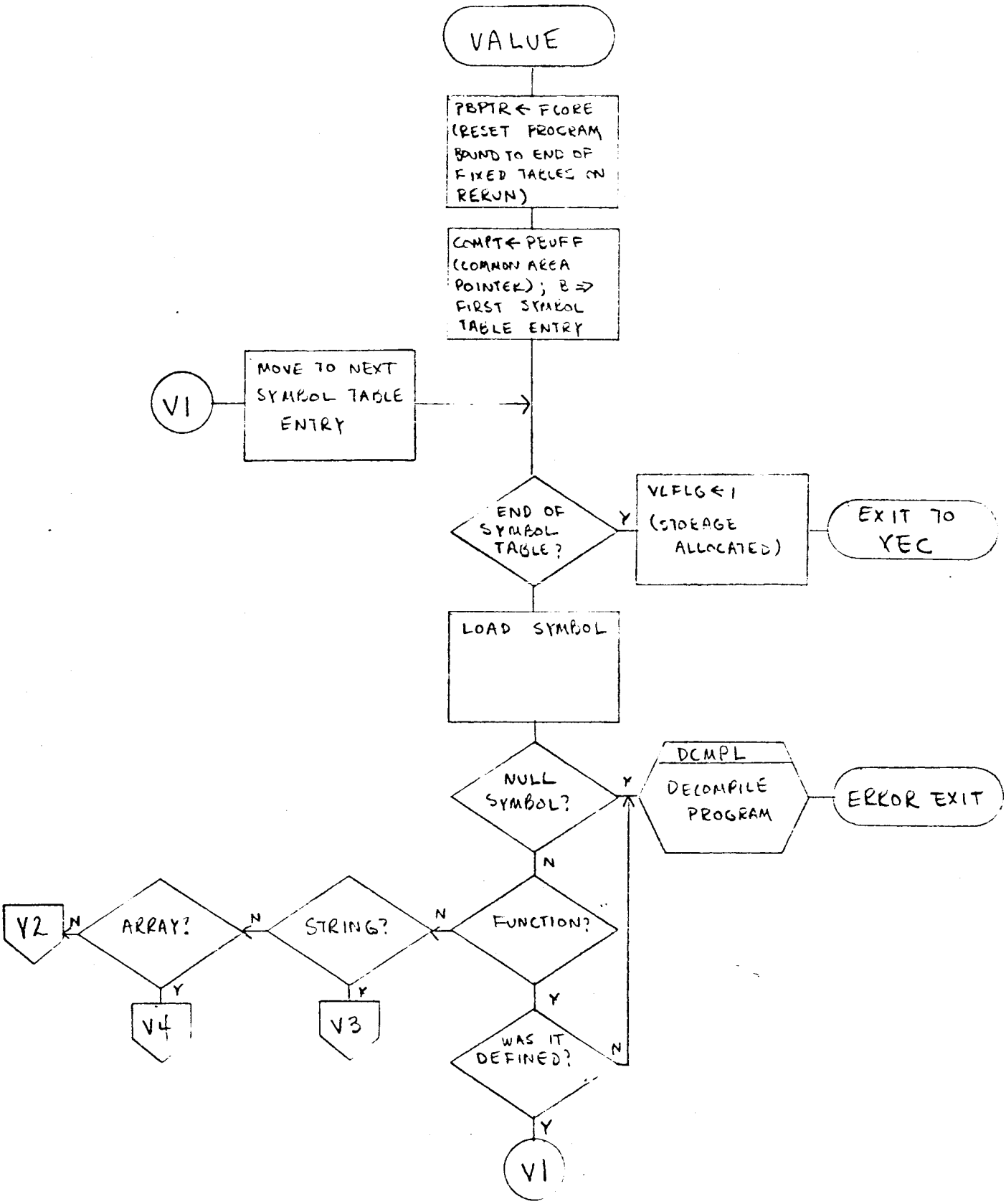

ASYMT
SET RETURN ADDRESSES; SET MATCH FLAG FALSE
CUSP
INSURE SPACE FOR ENTRY
MATCH AS DIMENSIONED SYMBOL?
Y
N
MATCH AS "DON'T KNOW" ?
Y
N
MOVE TO NEXT SYMBOL
MORE SYMBOLS?
Y
N
IS SYMBOL DON'T KNOW?
N
S1
Y
DIMENSIONED EQUIVALENT FOUND?
Y
POINT "DON'T KNOW" ENTRY TO DIMENSIONED ONE
N
CREATE NULL ENTRY ABOVE NEW ONE
S2
SAME DIMENSIONALITY ?
Y
S2
N
IS SYMBOL "DON'T KNOW" ?
N
DCMPL
DECOMPILE PROGRAM
ERROR EXIT
Y
SAVE POINTER TO ENTRY (MATCH FLAG TRUE)
IS SYMBOL "DON'T KNOW" ?
Y
S2
N
POINT "DON'T KNOW" ENTRY TO NEW ENTRY
S1

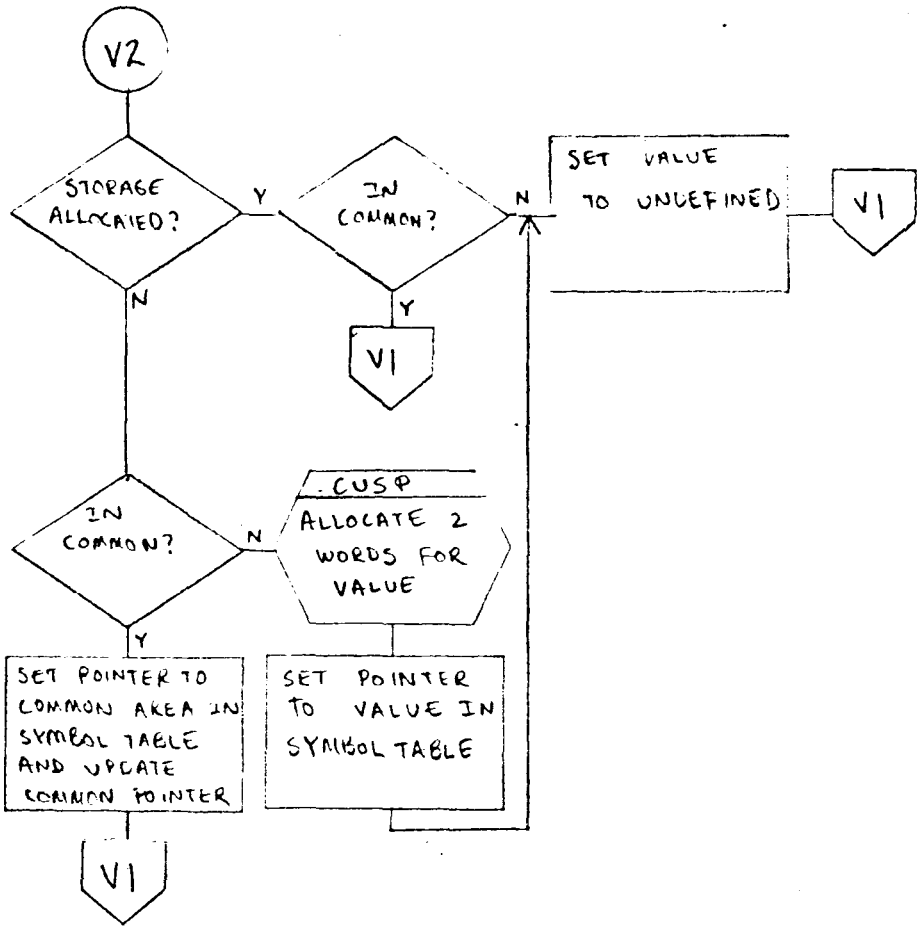
RSTPT

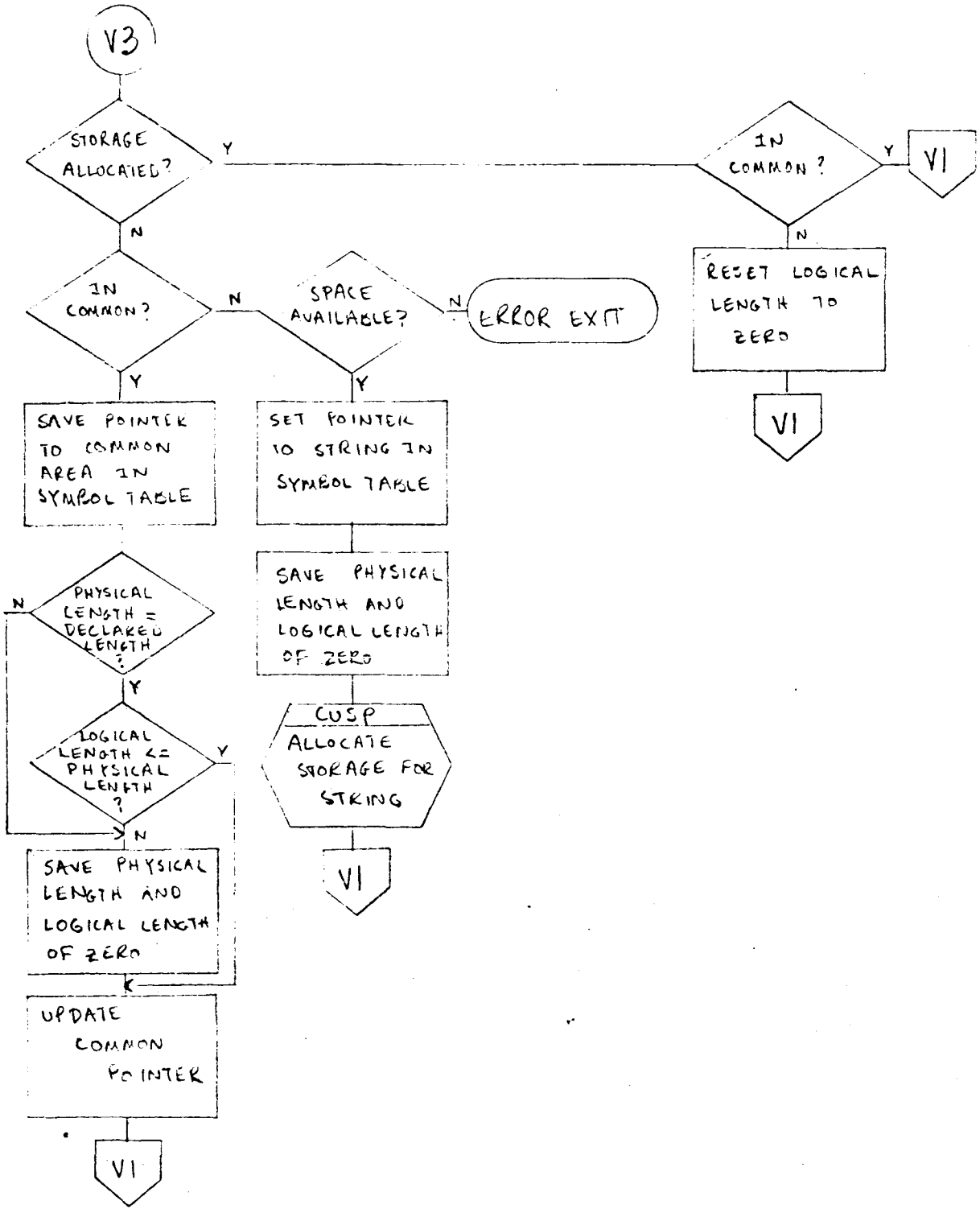

RSTPT
SET POINTER TO BEGINNING OF PROGRAM
FINISHED PROGRAM SCAN?
Y
N
EXTRACT OPERATOR; PO => NEXT STATEMENT
COM OR DIM?
N
Y
EXTRACT SYMBOL FROM SYMBOL TABLE
SIMPLE VARIABLE?
N
Y
SECOND WORD OF SYMBOL TABLE ENTRY ← -1
MOVE TO NEXT SYMBOL IN STATEMENT
END OF STATEMENT?
N
Y
SET POINTER TO DIMENSIONS IN SYMBOL TABLE
< COM?
Y
NEGATE POINTER
N
MOVE PAST DIMENSIONS
SET POINTER TO BEGINNING OF SYMBOL TABLE
END OF SYMBOL TABLE?
Y
ULFLG ← 0 (STORAGE NOT ALLOCATED)
RETURN
N
DON'T KNOW ENTRY?
N
STRING OR ARRAY?
N
Y
SECOND WORD OF SYMBOL TABLE ENTRY ← POINTER TO KNOWN ENTRY
Y
DEFAULT DIMENSIONS?
N
Y
SECOND WORD OF SYMBOL TABLE ENTRY ← 0
MOVE TO NEXT SYMBOL TABLE ENTRY


ALCOM
DEST ← 0 (NO COMMON ALLOCATED) STPTR ⇒ FIRST STATEMENT
EXTRACT OPERATOR
COM?
N
ANY COMMON ALLOCATED ?
Y
RESET SPROG AND PBPTR
MOVE PROGRAM TO MAKE ROOM FOR COMMON AREA
RETURN
N
RETURN
Y
COMPUTE AMOUNT OF STORAGE REQUIRED FOR VARIABLE
CUSP
INSURE THERE IS ENOUGH ROOM
UPDATE AMOUNT OF COMMON STORAGE
MOVE TO NEXT VARIABLE IN STATEMENT
N
END OF STATEMENT?
Y
N
END OF PROGRAM?
Y

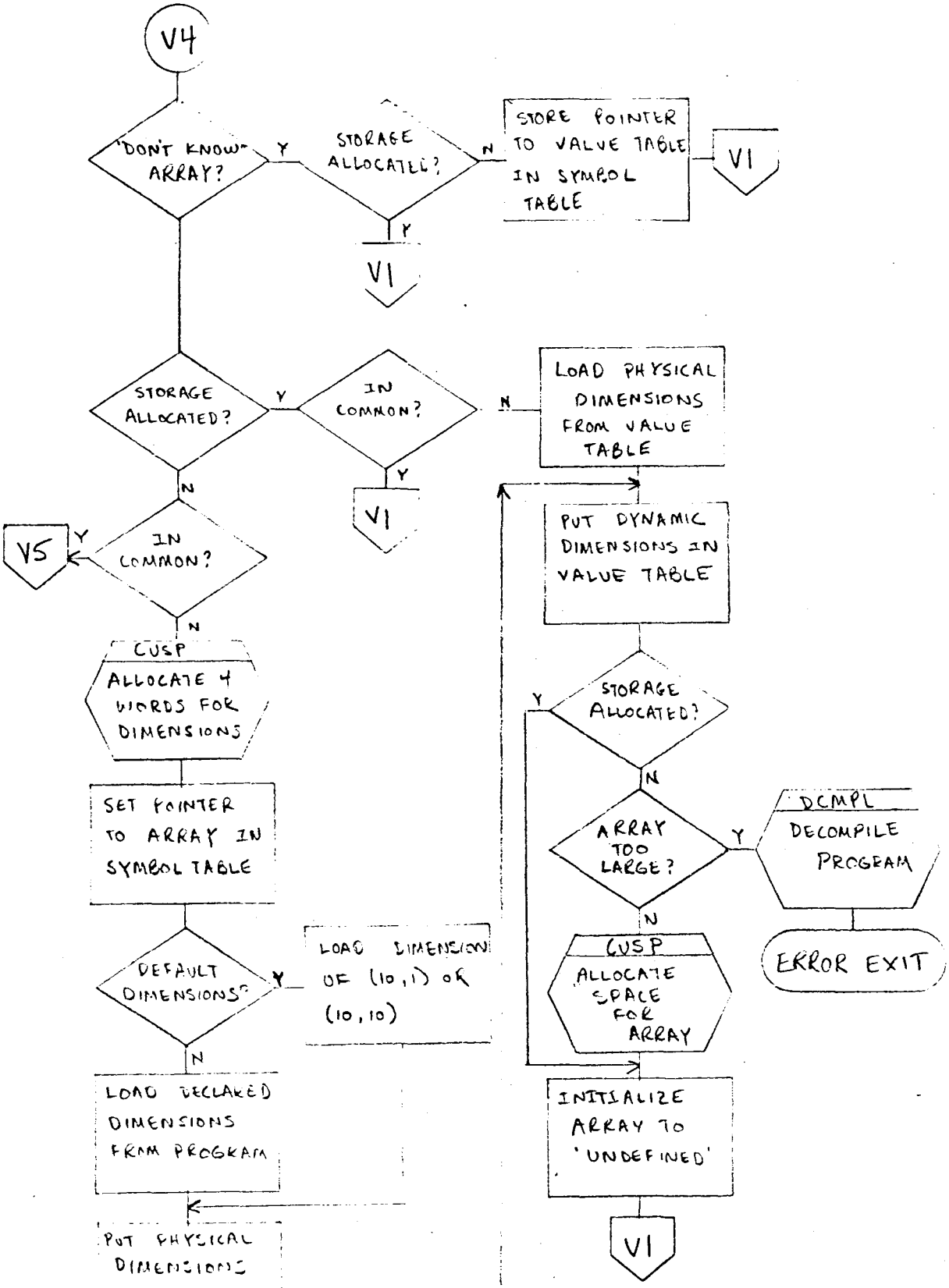
VALUE (1 of 5)


VALUE
PBPTR ← FCORE (RESET PROGRAM BOUND TO END OF FIXED TABLES ON RERUN)
COMPT ← PEUFF (COMMON AREA POINTER); B ⇒ FIRST SYMBOL TABLE ENTRY
V1
MOVE TO NEXT SYMBOL TABLE ENTRY
END OF SYMBOL TABLE?
Y
VLFLG ← 1 (STORAGE ALLOCATED)
EXIT TO YEC
LOAD SYMBOL
NULL SYMBOL?
Y
DCMPL
DECOMPILE PROGRAM
ERROR EXIT
N
FUNCTION?
N
STRING?
N
ARRAY?
N
V2
Y
V4
Y
V3
Y
WAS IT DEFINED?
N
Y
V1

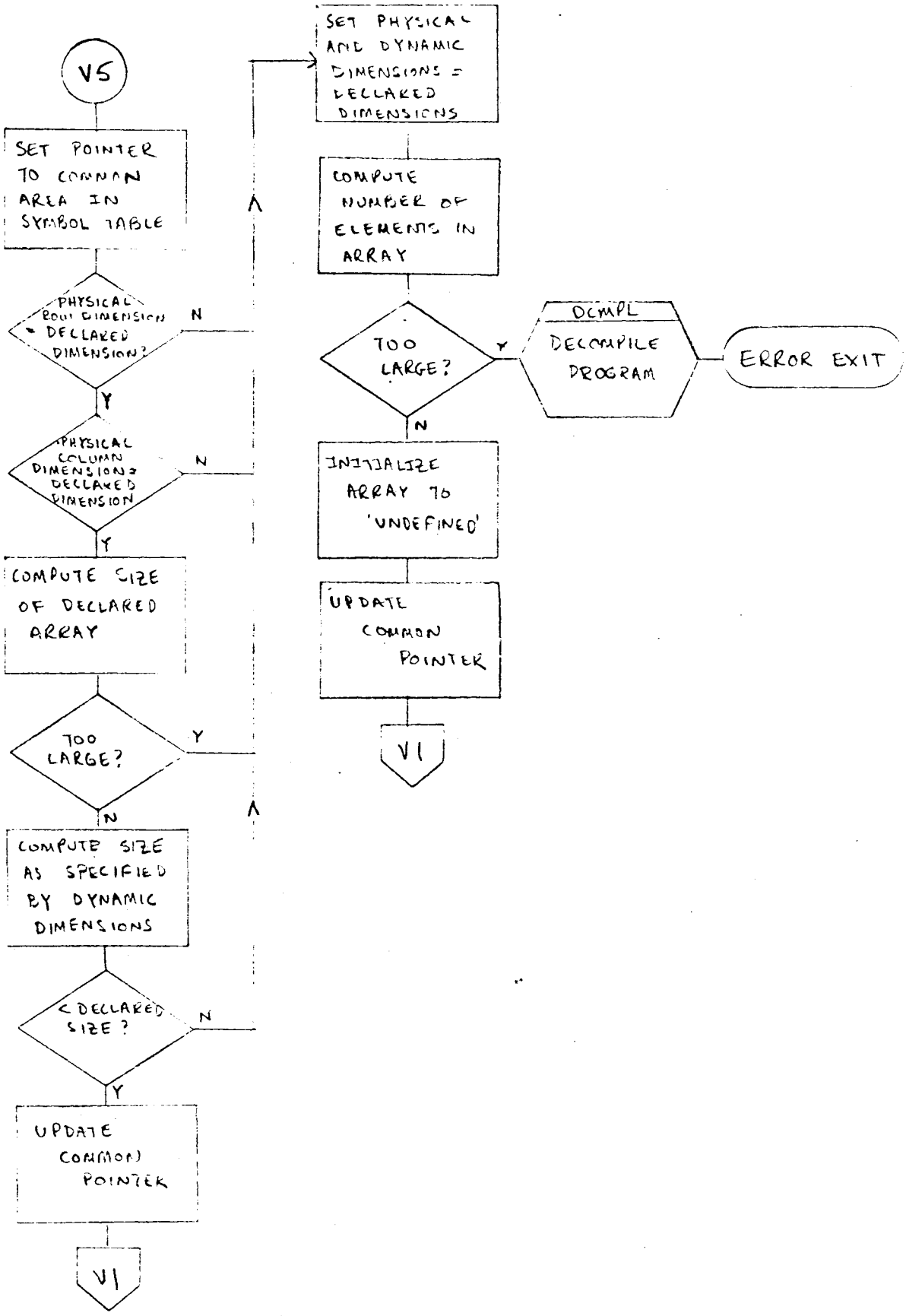

V2
STORAGE ALLOCATED?
Y
IN COMMON?
N
SET VALUE TO UNDEFINED
V1
N
Y
V1
IN COMMON?
N
.CUSP
ALLOCATE 2 WORDS FOR VALUE
Y
SET POINTER TO COMMON AREA IN SYMBOL TABLE AND UPDATE COMMON POINTER
SET POINTER TO VALUE IN SYMBOL TABLE
V1


V3
STORAGE ALLOCATED?
Y
N
IN COMMON?
N
Y
SPACE AVAILABLE?
N
ERROR EXIT
Y
SAVE POINTER TO COMMON AREA IN SYMBOL TABLE
SET POINTER TO STRING IN SYMBOL TABLE
SAVE PHYSICAL LENGTH AND LOGICAL LENGTH OF ZERO
CUSP
ALLOCATE STORAGE FOR STRING
V1
PHYSICAL LENGTH = DECLARED LENGTH ?
N
Y
LOGICAL LENGTH <= PHYSICAL LENGTH ?
Y
N
SAVE PHYSICAL LENGTH AND LOGICAL LENGTH OF ZERO
UPDATE COMMON POINTER
V1
IN COMMON ?
Y
V1
N
RESET LOGICAL LENGTH TO ZERO
V1

VALUE (4 of 5)

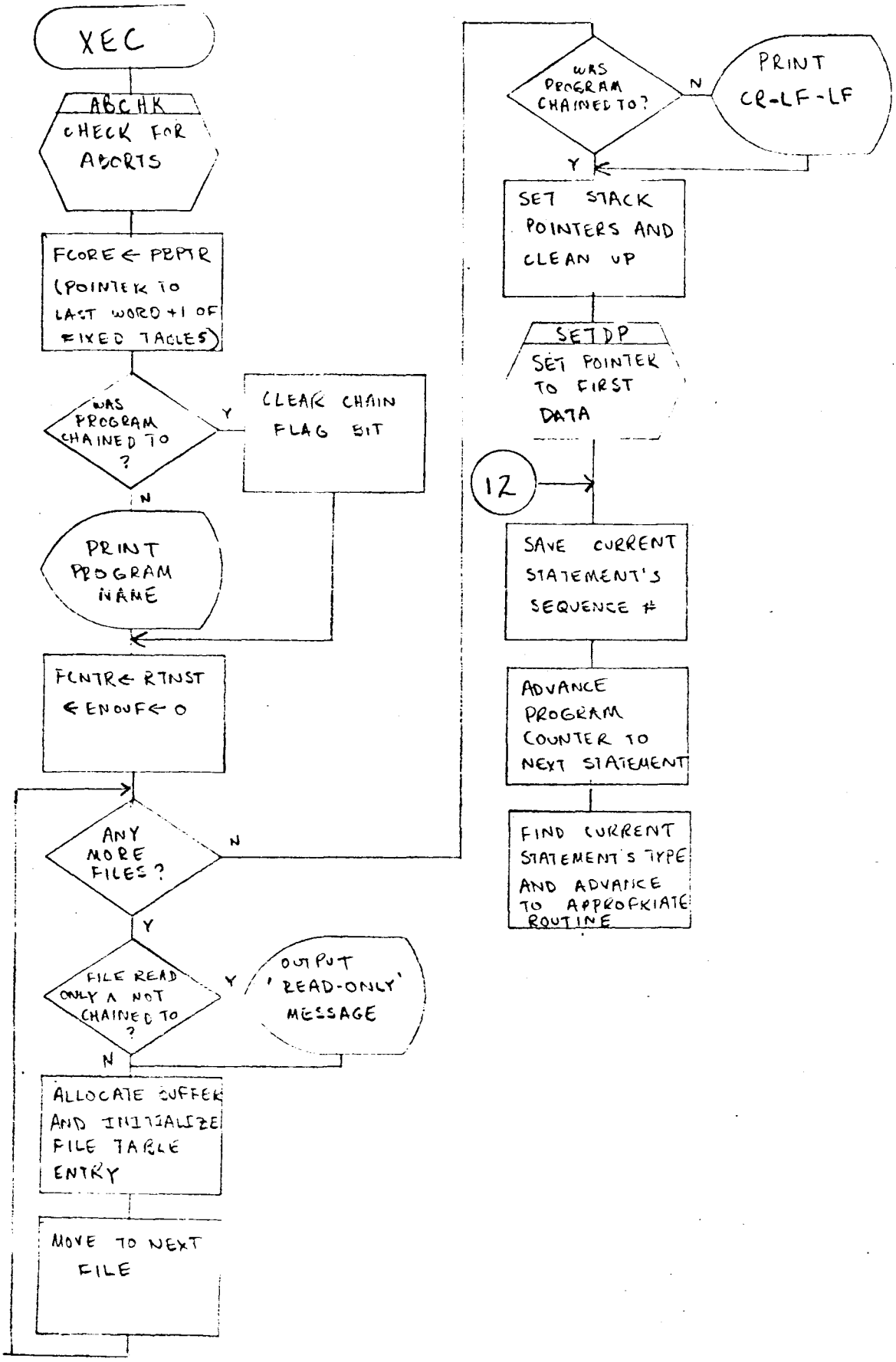

V4
'DON'T KNOW' ARRAY?
Y
STORAGE ALLOCATED?
N
STORE POINTER TO VALUE TABLE IN SYMBOL TABLE
V1
Y
V1
STORAGE ALLOCATED?
Y
IN COMMON?
N
LOAD PHYSICAL DIMENSIONS FROM VALUE TABLE
Y
V1
N
V5
Y
IN COMMON?
N
CUSP
ALLOCATE 4 WORDS FOR DIMENSIONS
SET POINTER TO ARRAY IN SYMBOL TABLE
DEFAULT DIMENSIONS?
Y
LOAD DIMENSION OF (10,1) OR (10,10)
N
LOAD DECLARED DIMENSIONS FROM PROGRAM
PUT PHYSICAL DIMENSIONS INTO VALUE AREA
PUT DYNAMIC DIMENSIONS IN VALUE TABLE
Y
STORAGE ALLOCATED?
N
ARRAY TOO LARGE?
Y
DCMPL
DECOMPILE PROGRAM
ERROR EXIT
N
CUSP
ALLOCATE SPACE FOR ARRAY
INITIALIZE ARRAY TO 'UNDEFINED'
V1

VALUE (5 of 5)

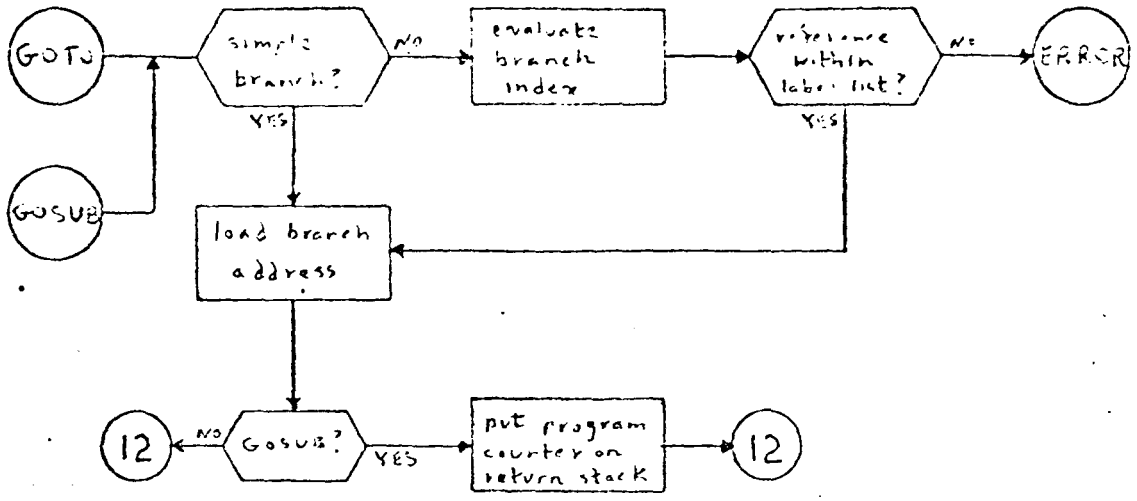

V5
SET POINTER TO COMMAN AREA IN SYMBOL TABLE
PHYSICAL ROW DIMENSION = DECLARED DIMENSION?
N
Y
PHYSICAL COLUMN DIMENSION = DECLARED DIMENSION
N
Y
COMPUTE SIZE OF DECLARED ARRAY
TOO LARGE?
Y
N
COMPUTE SIZE AS SPECIFIED BY DYNAMIC DIMENSIONS
< DECLARED SIZE ?
N
Y
UPDATE COMMON POINTER
V1
SET PHYSICAL AND DYNAMIC DIMENSIONS = DECLARED DIMENSIONS
COMPUTE NUMBER OF ELEMENTS IN ARRAY
TOO LARGE?
Y
DCMPL
DECOMPILE PROGRAM
ERROR EXIT
N
INITIALIZE ARRAY TO 'UNDEFINED'
UPDATE COMMON POINTER
V1

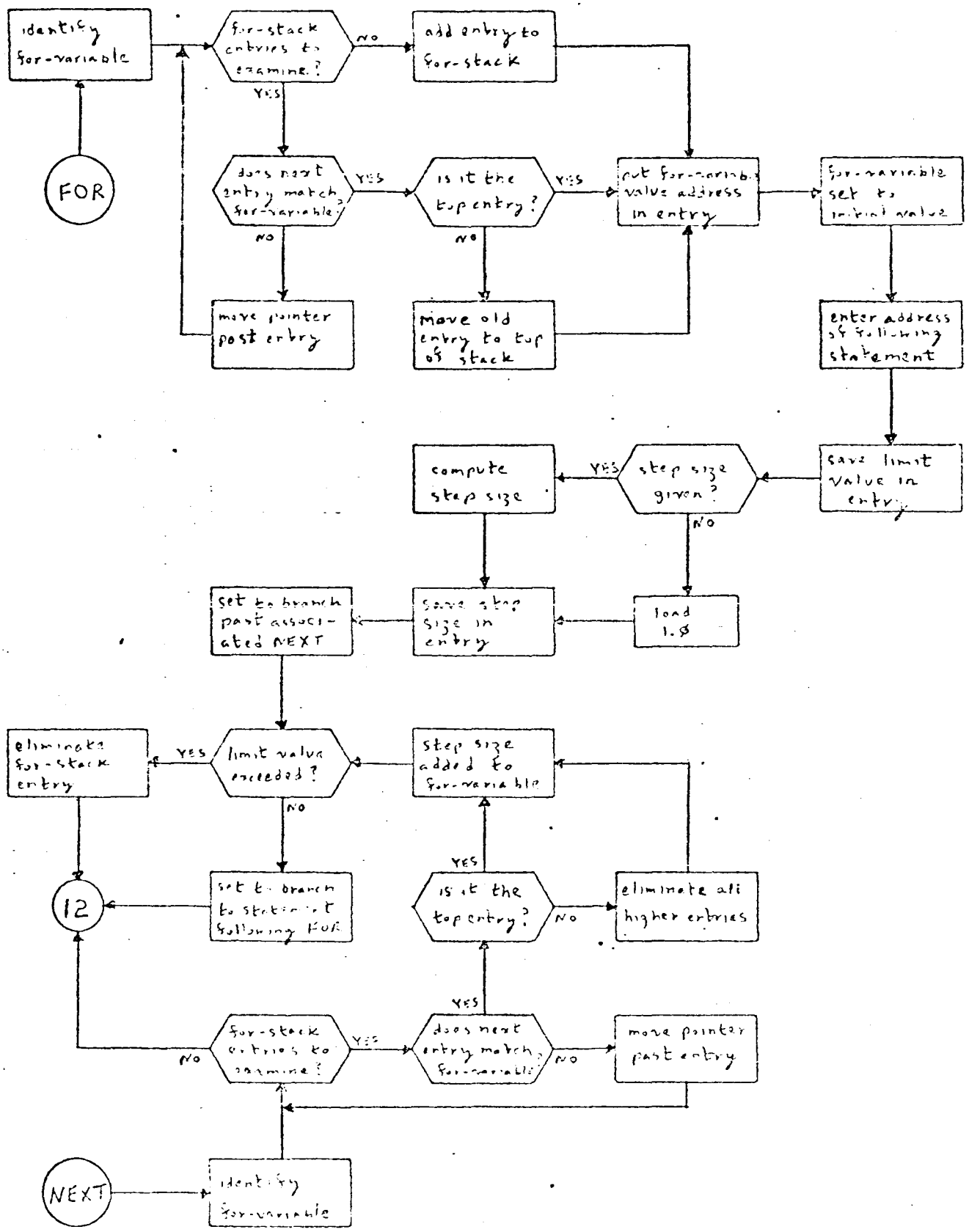
EXECUTION

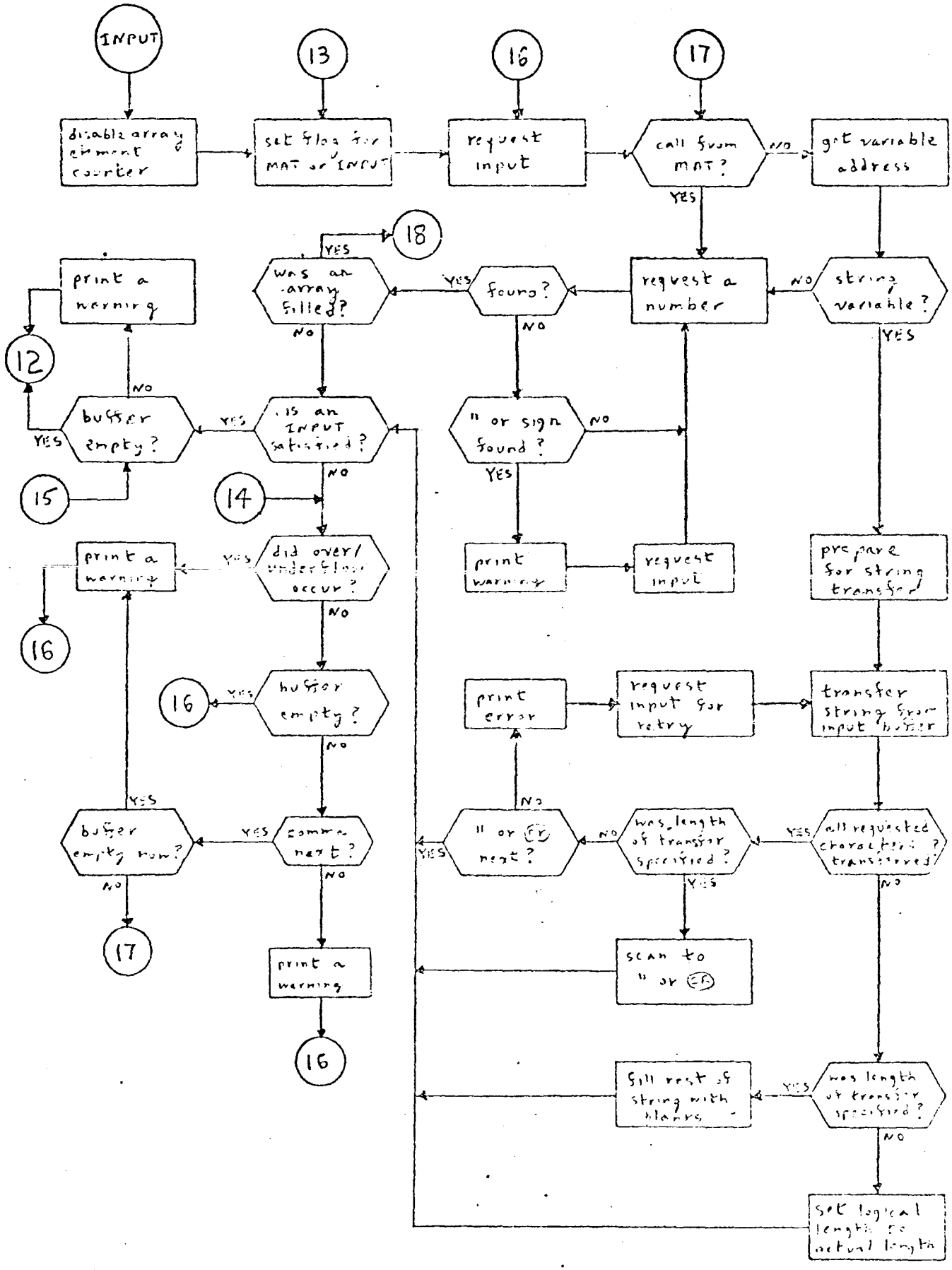
A. Main Loop

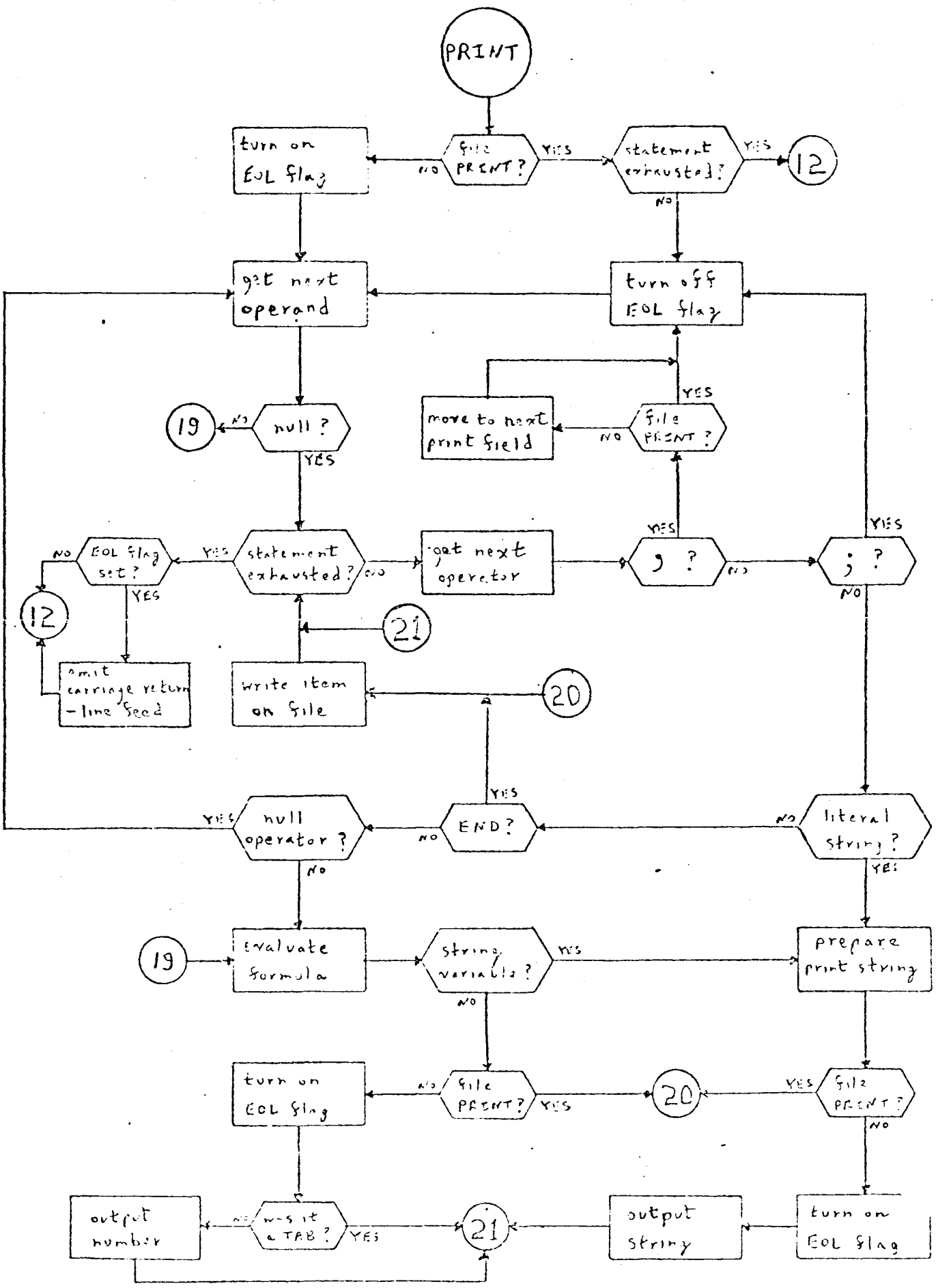


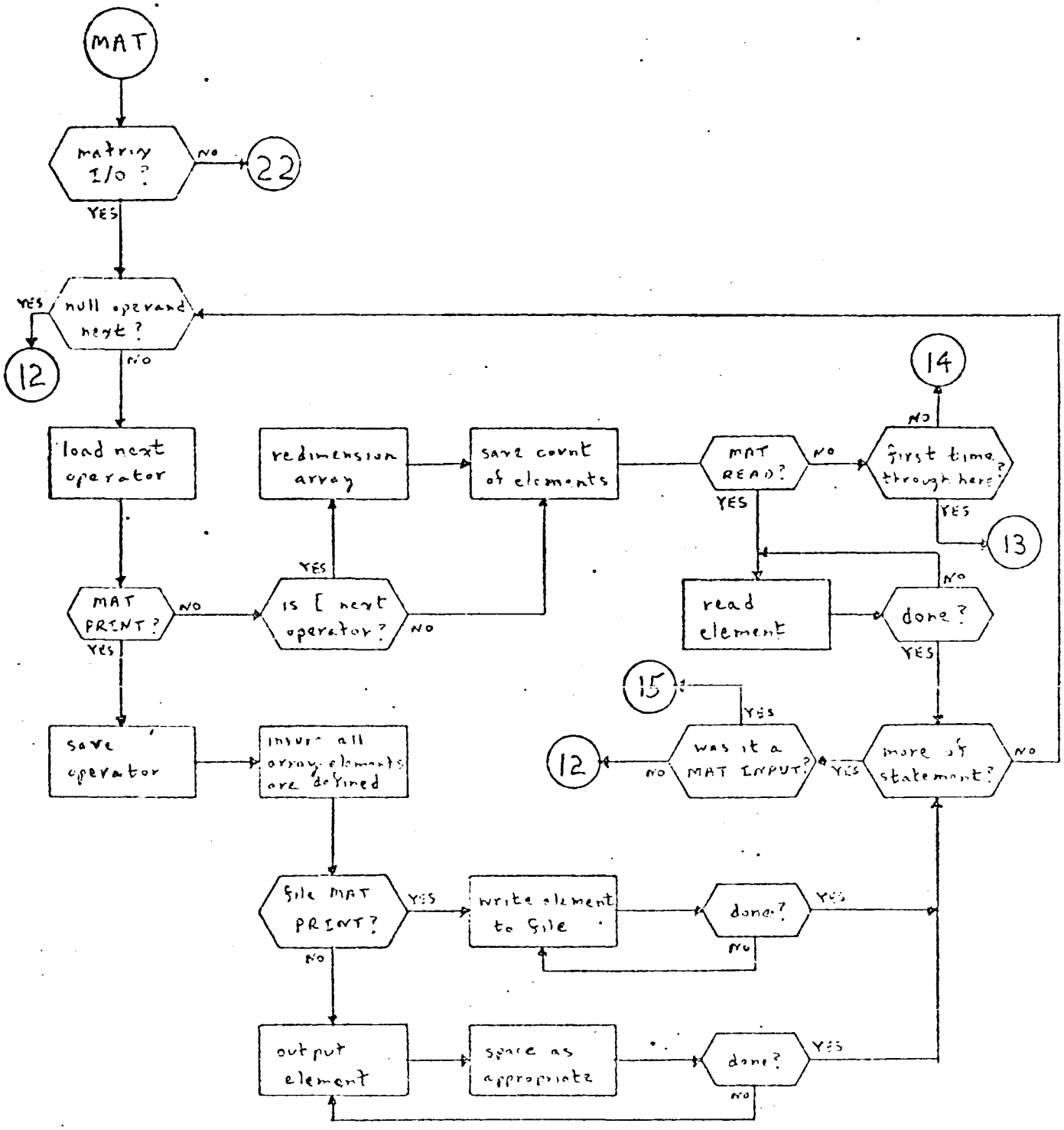
B. Selected Statement Types

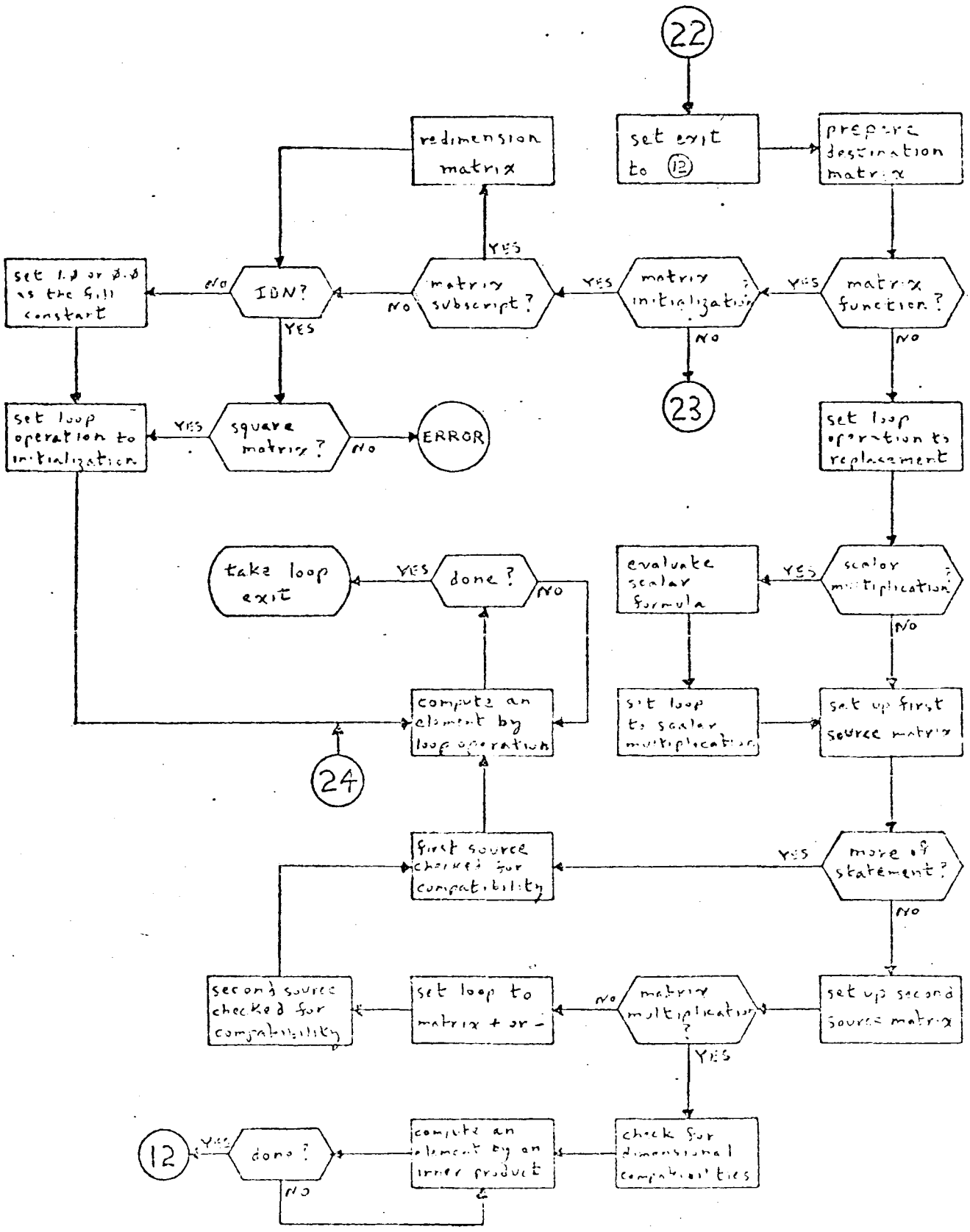


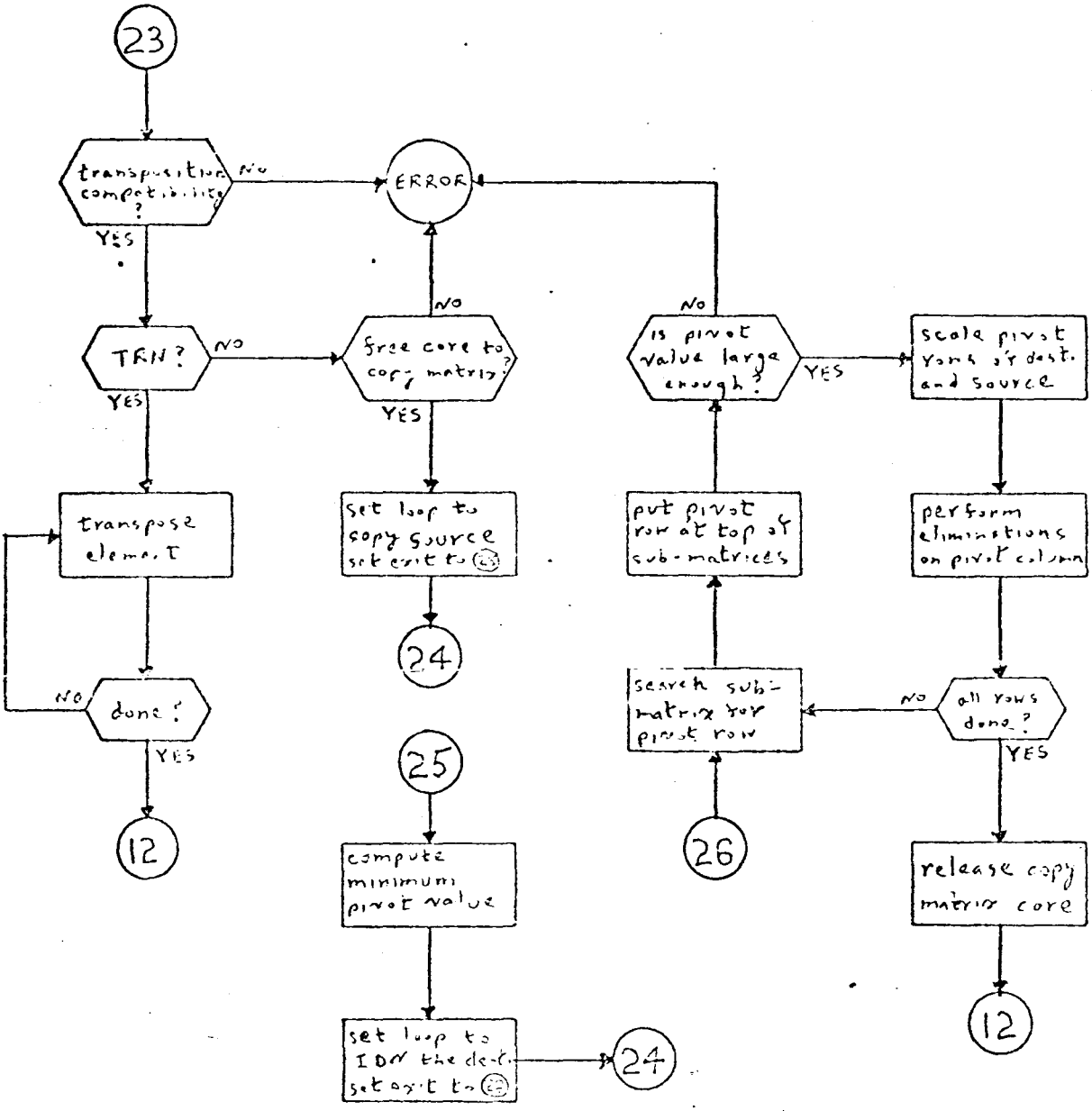

identify for-variable
for-stack entries to examine?
no
add entry to for-stack
YES
FOR
does next entry match for-variable?
YES
is it the top entry?
YES
put for-variable value address in entry
for-variable set to initial value
NO
NO
move pointer past entry
move old entry to top of stack
enter address of following statement
compute step size
YES
step size given?
save limit value in entry
NO
set to branch past associated NEXT
save step size in entry
load 1.0
eliminate for-stack entry
YES
limit value exceeded?
step size added to for-variable
NO
YES
12
set to branch to statement following FOR
is it the top entry?
NO
eliminate all higher entries
YES
for-stack entries to examine?
YES
does next entry match for-variable?
NO
move pointer past entry
NO
NEXT
identify for-variable


INPUT
13
16
17
disable array element counter
set flag for MAT or INPUT
request input
call from MAT?
NO
get variable address
YES
18
YES
print a warning
was an array filled?
YES
found?
request a number
NO
string variable?
YES
12
NO
NO
NO
buffer empty?
YES
is an INPUT satisfied?
YES
" or sign found?
NO
YES
15
14
NO
YES
print a warning
did over/ underflow occur?
YES
print warning
request input
prepare for string transfer
NO
16
16
YES
buffer empty?
print error
request input for retry
transfer string from input buffer
NO
NO
YES
buffer empty now?
YES
comma next?
" or EF next?
NO
was length of transfer specified?
YES
all requested characters transferred?
NO
NO
YES
NO
17
print a warning
scan to " or EF
16
fill rest of string with blanks
YES
was length of transfer specified?
NO
set logical length to actual length


PRINT
file PRINT?
YES
NO
statement exhausted?
YES
12
turn on EOL flag
get next operand
turn off EOL flag
19
null ?
YES
move to next print field
file PRINT ?
YES
NO
EOL flag set?
YES
NO
statement exhausted?
YES
NO
get next operator
, ?
YES
NO
; ?
YES
NO
12
21
emit carriage return - line feed
write item on file
20
null operator ?
YES
NO
END?
YES
NO
literal string ?
YES
NO
19
evaluate formula
string variable ?
YES
NO
prepare print string
turn on EOL flag
file PRINT?
NO
YES
20
YES
file PRINT ?
NO
output number
was it a TAB ?
NO
YES
21
output string
turn on EOL flag


MAT
matrix I/O ?
NO
22
YES
null operand next ?
YES
12
NO
load next operator
redimension array
save count of elements
MAT READ?
NO
first time through here?
14
NO
YES
YES
13
NO
MAT PRINT ?
NO
YES
is [next operator?
NO
YES
read element
done ?
YES
15
YES
save operator
insure all array elements are defined
was it a MAT INPUT?
NO
12
YES
more of statement?
NO
file MAT PRINT?
YES
NO
write element to file
done?
YES
NO
output element
space as appropriate
done?
YES
NO


22
set exit to (12)
prepare destination matrix
redimension matrix
YES
matrix initialization ?
YES
matrix function ?
YES
matrix subscript ?
NO
IDN?
NO
set 1.0 or 0.0 as the fill constant
YES
NO
23
NO
set loop operation to initialization
YES
square matrix ?
NO
ERROR
set loop operation to replacement
take loop exit
YES
done ?
NO
evaluate scalar formula
YES
scalar multiplication ?
NO
compute an element by loop operation
set loop to scalar multiplication
set up first source matrix
24
first source checked for compatibility
YES
more of statement ?
NO
second source checked for compatibility
set loop to matrix + or -
NO
matrix multiplication ?
set up second source matrix
YES
12
YES
done ?
compute an element by an inner product
check for dimensional compatibilities
NO


23
transposition compatibility ?
NO
YES
ERROR
TRN ?
NO
YES
free core to copy matrix?
NO
YES
is pivot value large enough?
NO
YES
scale pivot rows at dest. and source
transpose element
set loop to copy source set exit to 13
put pivot row at top of sub-matrices
perform eliminations on pivot column
NO
done ?
YES
24
search sub-matrix for pivot row
NO
all rows done ?
YES
12
25
26
compute minimum pivot value
release copy matrix core
set loop to IDN the dest. set exit to 23
24
12

SYNTAX REQUIREMENTS OF TSB

LEGEND

::= "is defined as..."

| "or"

< > enclose an element of Time Shared BASIC

LANGUAGE RULES

1. Exponents have 1 or 2 digit integers only.
2. A <parameter> primary appears only in the defining formula of a <DEF statement>.
3. A <sequence number> must lie between 1 and 9999 inclusive.
4. An array bound must lie between 1 and 9999 inclusive; a string variable bound must lie between 1 and 72 inclusive.
5. The character string for a <REM statement> may include the character ".
6. An array may not be transposed into itself, nor may it be both an operand and the result of a matrix multiplication.

Note: Parentheses, (), and square brackets, [], are accepted interchangeably by the syntax analyzer.

Continued on the next page.

SYNTAX REQUIREMENTS OF TSB

<constant>	::=	<number>|+<number>|-<number>|<literal string>
<number>	::=	<decimal number>|<decimal number><exponent part>
<decimal number>	::=	<integer>|<integer>.|<integer>.<integer>|.<integer>
<integer>	::=	<digit>|<integer><digit>
<digit>	::=	Ø|1|2|3|4|5|6|7|8|9
<exponent part>	::=	E<integer>|E+<integer>|E-integer *(see rule 1)*
<literal string>	::=	"<character string>"
<character string>	::=	<character>|<character string><character>
<character>	::=	any ASCII character except null, line feed, return, x-off, alt-mode, escape, ←, " , and rubout
<variable>	::=	<simple variable>|<subscripted variable>
<simple variable>	::=	<letter>|<letter><digit>
<letter>	::=	A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<subscripted variable>	::=	<letter>(<sublist>)
<sublist>	::=	<expression>|<expression>,<expression>
<string variable>	::=	<string simple variable>|<string simple variable>(<sublist
<string simple variable>	::=	<letter>$
<expression>	::=	<conjunction>|<expression>OR<conjunction>
<conjunction>	::=	<relation>|<conjunction>AND<relation>
<relation>	::=	<minmax>|<minmax><relational operator><minmax>
<minmax>	::=	<sum>|<minmax>MIN<sum>|<minmax>MAX<sum>
<sum>	::=	<term>|<sum>+<term>|<sum>-<term>
<term>	::=	<subterm>|<term>*<subterm>|<term>/<subterm>
<subterm>	::=	<denial>|<signed factor>

SYNTAX REQUIREMENTS OF TSB . CONTINUED

<denial>	::=	<factor>|NOT<factor>
<signed factor>	::=	+<factor>|-<factor>
<factor>	::=	<primary>|<factor>↑<primary>
<primary>	::=	<variable>|<number>|<functional>|<parameter> *(rule 2)*| (<expression>)
<relational operator>	::=	<|<=|=|#|<>|>=|>
<parameter>	::=	.<letter>|<letter><digit>
<functional>	::=	<function identifier>(<expression>)| <pre-defined function>(<expression>)| LEN (<string simple variable>)
<function identifier>	::=	FN <letter>
<pre-defined function>	::=	SIN|COS|TAN|ATN|EXP|LOG|ABS|SQR|INT|RND|SGN|TYP|TIM
<source string>	::=	<string variable>|<literal string>
<destination string>	::=	<string variable>
<file reference>	::=	#<file formula>|#<file formula>,<record formula>
<file formula>	::=	<expression>
<record formula>	::=	<expression>
<array identifier>	::=	<letter>
<sequence number>	::=	<integer> *(see rule 3)*
<program statement>	::=	<sequence number><BASIC statement>carriage return
<BASIC statement>	::=	<LET statement>|<IF statement>|<GOTO statement>| <GOSUB statement>|<RETURN statement>|<FOR statement>| <NEXT statement>|<STOP statement>|<END statement>| <DATA statement>|<READ statement>|<INPUT statement>| <ENTER statement>|<PRINT statement>| <RESTORE statement<|>DIM statement>|<COM statement>| <DEF statement>|<FILES statement>|<REM statement>| <CHAIN statement>|<MAT statement>
<LET statement>	::=	LET <leftpart><expression>| LET <destination string>=<source string>| <leftpart><expression>| <destination string>=<source string>
<leftpart>	::=	<variable>=|<leftpart><variable>=
<IF statement>	::=	IF<decision expression>THEN<sequence number>| IF END #<file formula>THEN<sequence number>
<decision expression>	::=	<expression>| <comparison string 1><relational operator> <comparison string 2>

SYNTAX REQUIREMENTS OF TSB . CONTINUED

```
<comparison string 1>    ::=  <string variable>
<comparison string 2>    ::=  <string variable>|<literal string>
<GOTO statement>         ::=  GOTO <sequence number>|
                              GOTO <expression>OF<sequence list>
<sequence list>          ::=  <sequence number>|<sequence list>,<sequence number>
<GOSUB statement>        ::=  GOSUB <sequence number>|
                              GOSUB <expression>OF <sequence list>
<RETURN statement>       ::=  RETURN
<FOR statement>          ::=  FOR <for variable>=<initial value>TO<final value>|
                               FOR <for variable>=<initial value>TO<final value>
                                                        STEP<step size>
<for variable>           ::=  <simple variable>
<initial value>          ::=  <expression>
<final value>            ::=  <expression>
<step size>              ::=  <expression>
<NEXT statement>         ::=  NEXT<for variable>
<STOP statement>         ::=  STOP
<END statement>          ::=  END
<DATA statement>         ::=  DATA<constant>|<DATA statement>,<constant>
<READ statement>         ::=  READ<variable list>|READ<file reference>|
                              READ<file reference>;<variable list>
<variable list>          ::=  <read variable>|<variable list>,<read variable>
<read variable>          ::=  <variable>|<destination string>
<INPUT statement>        ::=  INPUT<variable list>
<ENTER statement>        ::=  ENTER#<variable>|
                              ENTER<variable>,<variable>,<variable>|
                              ENTER<variable>,<variable>,<string variable>|
                              ENTER#<variable>,<variable>,<variable>,
                                                        <string variable>
                              ENTER#<variable>,<variable>,<variable>,<variable>
<PRINT statement>        ::=  <type statement>|<file write statement>|
                              PRINT<file reference>
<type statement>         ::=  <print 1>|<print 2>
<print 1>                ::=  PRINT|<print 2>,|<print 2>;|<print 3>
<print 2>                ::=  <print 1><print expression>|<print 3>
<print 3>                ::=  <type statement><literal string>
<print expression>       ::=  <expression>|TAB(<expression>)|<source string>
```

```
<file write statement>   ::=  PRINT<file reference>;<write expression>|
                              <file write statement>,<write expression>|
                              <file write statement>;<write expression>|
                              <file write statement><literal string>|
                              <file write statement><literal string>
                                                    <write expression>

<write expression>       ::=  <expression>|END|<source string>

<RESTORE statement>      ::=  RESTORE|RESTORE<sequence number>

<DIM statement>          ::=  DIM<dimspec>|<DIM statement>,<dimspec>

<COM statement>          ::=  COM<com list element>|
                                 <COM statement>,<com list element>

<com list element>       ::=  <simple variable>|<string simple variable>|
                                 <dimspec>

<dimspec>                ::=  <array identifier>(<bound>)|
                              <array identifier>(<bound>,<bound>)|
                              <string simple variable>(<bound>)

<bound>                  ::=  <integer> (see rule 4)

<DEF statement>          ::=  DEF<function identifier>(<parameter>)=<expression>

<FILES statement>        ::=  FILES<name>|<FILES statement>,<name>

<name>                   ::=  a string of 1 to 6 printing characters>

<REM statement>          ::=  REM<character string> (see rule 5)

<CHAIN statement>        ::=  CHAIN<name>|<CHAIN $<name>

<MAT statement>          ::=  <MAT READ statement>|<MAT INPUT statement>|
                              <MAT PRINT statement>|<MAT initialization statement>|
                              <MAT assignment statement>

<MAT READ statement>     ::=  MAT READ<actual array>|
                              MAT READ<file reference>;<actual array>|
                              <MAT READ statement>,<actual array>

<actual array>           ::=  <array identifier>|<array identifier>(<dimensions>)

<dimensions>             ::=  <expression>|<expression>,<expression>

<MAT INPUT statement>    ::=  MAT INPUT<actual array>|
                              <MAT INPUT statement>,<actual array>

<MAT PRINT statement>    ::=  <MAT PRINT 1>|<MAT PRINT 2>

<MAT PRINT 1>            ::=  MAT PRINT<array identifier>|
                              MAT PRINT<file reference>;<array identifier>|
                              <MAT PRINT 2><array identifier>

<MAT PRINT2>             ::=  <MAT PRINT 1>,|<MAT PRINT 1>;
```

SYNTAX REQUIREMENTS OF TSB CONTINUED

<MAT initialization statement>::= MAT<array identifier>=<initialization function>|
MAT<array identifier>=<initialization function>
(<dimensions>)

<initialization function>::= ZER|CON|IDN

<MAT assignment statement>*(rule 6)* ::= MAT<array identifier>=<array identifier>|
MAT<array identifier>=<array identifier><mat operator>
<array identifier>|
MAT<array identifier>=INV(<array identifier>)
MAT<array identifier>=TRN(<array identifier>)|
MAT<array identifier>=(<expression>)*<array identifier>

<mat operator> ::= +|-|*