A 11

Xerox Universal Time-Sharing System (UTS)

Sigma 6/7/9 Computers

Reliability and Maintainability
Technical Manual

Xerox Corporation
701 South Aviation Boulevard
El Segundo, California 90245
213 679-4511

XEROX

Xerox Universal Time-Sharing System (UTS)

Sigma 6/7/9 Computers

Reliability and Maintainability

Technical Manual

FIRST EDITION

90 19 90A

February, 1973

Price: $8.25

©1973, Xerox Corporation

Printed in U.S.A.

NOTICE

This publication documents the reliability and maintainability functions of the Universal Time-Sharing System (UTS) for Sigma 6/7/9 computers. All material in this manual reflects the C01 version of UTS.

RELATED PUBLICATIONS

Title	Publication No.
UTS Overview and Index Technical Manual[†]	90 19 84
UTS Basic Control and Basic I/O Technical Manual	90 19 85
UTS System and Memory Management Technical Manual	90 19 86
UTS Symbiont and Job Management Technical Manual	90 19 87
UTS Operator Communication and Monitor Services Technical Manual	90 19 88
UTS File Management Technical Manual[†]	90 19 89
UTS Interrupt Driven Tasks Technical Manual[†]	90 19 91
UTS Initialization and Recovery Technical Manual	90 19 92
UTS Command Processors Technical Manual	90 19 93
UTS System Processors Technical Manual	90 19 94
UTS Data Bases Technical Manual	90 19 95

[†]Not published as of the publication date given on the title page of this manual. Refer to the PAL Manual for current availability.

The specifications of the software system described in this publication are subject to change without notice. The availability or performance of some features may depend on a specific configuration of equipment such as additional tape units or larger memory. Customers should consult their Xerox sales representative for details.

CONTENTS

CHK – System Consistency Checks ____ 1
Purpose ____ 1
Usage ____ 1
Overview ____ 1
Input ____ 1
Output ____ 2
Subroutines ____ 2
Errors ____ 2
Description ____ 2

Diagnostic and Exerciser Interface ____ 4
Purpose ____ 4
Overview ____ 4
Interface ____ 4
Data Base ____ 4
Description ____ 5

ERRLOG – Error Logging Routine ____ 6
Purpose ____ 6
Usage ____ 6
Input ____ 6
Output ____ 6
Interaction ____ 6
Data Base ____ 6
Description ____ 6
Flowchart ____ 8

ERR:FIL – Copy Error Log ____ 10
Purpose ____ 10
Usage ____ 10
Input ____ 10
Output ____ 10
Interaction ____ 10
Data Bases ____ 10
Subroutines ____ 12
Description ____ 12
Summary Record Format ____ 15
Copy Error Record Format ____ 16
Flowchart ____ 17

ERR:SUM – Print Error Summary ____ 21
Purpose ____ 21
Usage ____ 21
Input ____ 22
Output ____ 22
Interaction ____ 22
Subroutines ____ 22
Restriction ____ 22
Description ____ 22
Error Summary Listing Format ____ 24
Flowchart ____ 25

ERR:LIST – Error Log Lister ____ 27
Purpose ____ 27
Usage ____ 27
Input ____ 27

Output 27
Interaction 27
Errors 27
Description 27

PFSR – Power Fail-Safe Recovery 29
Purpose 29
Usage 29
Input 29
Output 29
Interaction 29
Description 29

Delta 31
Introduction 31
General Description of Delta in UTS 31
Memory Map and Layout 31
Access Protection 32
Symbol Tables 32
User Execution Control 34
DCBs 35
Detailed Description of Routines 35
Introduction 35
Definitions 35
Command Structure 36
Initialization 36
Command Parse (DRSCAN1) 37
Formatting Routines 38
Input Formatting Routines 38
Output Formatting Routines 39
Evaluation and Symbol Table Search Routines 41
Flowcharts 45
Punctuation Commands 48
Semicolon Commands 51
System Control 61
Utility Routines 65
I/O Routines 70
Constant Data 71
Variable Data (User's Context) 73
Exec Delta 74
Purpose 74
Overview 74
Usage 74
Description 74
Restrictions 76
Delta Interface with Other Processors 77
Purpose 77
Usage 77

User Program Debugging 79
Overview 79

RUNNER 80
Purpose 80
Usage 80
Input 80
Output 80
Data Bases 80
Subroutines 81
Errors 82
Restrictions 83
Description 83

Debug Routines 85
SNAP – Process Debug CALs 85
Purpose 85
Overview 85
Usage 85
Error 85
MSNAP – Process !SNAP Request 86
Purpose 86
Input 86
MSNAPC – Process !SNAPC Request 87
Purpose 87
Input 87
Description 87
MIF – Process !IF Request 88
Purpose 88
Input 88
Description 89
MAND – Process !AND Request 90
Purpose 90
Input 90
Description 90
MOR – Process !OR Request 90
Purpose 90
Description 90
MCOUNT – Process !COUNT Request 91
Purpose 91
Description 92
PMD – Provide Postmortem Core Dumps 93
Purpose 93
Usage 93
Description 93

TELLUSR 94
Purpose 94
Usage 94
Input 94
Output 94
Interaction 95
Data Bases 95
Subroutines 95
Flowchart 98

DUMP 103
Purpose 103
Entry 103
Operation 103
DUMPW 103
PRINT 104
PRINTM 105

SCREECH 108
Purpose 108
Usage 108
Input 108
Output 108
Interaction 108
Description 108
SCREECH Codes 109

ANALYZE – Monitor Debug Program 113
Purpose 113
Overview 113
Interaction of ANALYZE Routines 114

Usage 115
Input 115
Output 116
Interaction 116
Data Bases 116
Subroutines 121
Error Messages 122
Commands and Routines 124
ALL 124
ALLJIT 124
COMPARE 124
DISPLAY 124
COCODE 125
IODISPLAY 125
JITS 126
PROCS 126
PSDS 126
REGS 126
SWAP 127
USERS 127
DUM 127
DLAST 127
DNEXT 127
INDR 127
EXITCL 127
INITIAL 127
INPUT 128
LP 128
MAPMODE 128
MONITOR 129
PRINT 129
REPLACEMENT 129
RUN 129
SCANNER 130
SEARCH 130
MASK 130
SYMBOLMAP 130
TRACE 131
UCLO 131
UNMAP 131
ASSOCIATEDEL 132
DISASSDEL 132
FILENAME 132
DELTAGET 132
DELTAPUT 132
Flowchart 133
Implementing a New Command 161
Adding Code to ANALYZE 162

DRSP – Dynamic Replacement of Shared Processor 172
Purpose 172
Overview 172
Proposed Enhancements 173
Description 173
Usage 175
Interaction 175
Restrictions 175
Flowchart 176
Subroutines 177
START 177
RESTART 178

INITIAL 180
DRSPMAIN 182
SYNTAX 183
LIST 188
RADNEED 191
GETRADSLOT 193
FINDSLOT 195
TELCCIONLY 197
WRITESWAP 198
PERM 200
MODRAD 202
RWRAD 204
SWITCH 206
CLOSEOUT 207
CLEANUP 208
SEARCH 210
GFID 211
WORTH 213
FINDGRAN 215
SCAN, SCANT 217
POST, POST 1 218
DAGRAN, GRANDA 219
TCTEST 221
XGRTEST 222
BUFAD 223
MASTER, SLAVE 224
S400 225
S360 226
S430 227
HEX2PRNT 228

FSAVE Processor 229
Purpose 229
Basic Philosophy 229
Module Analysis 229
INITIATE 229
Purpose 229
Entry 229
Exit 230
Operation 230
INITIATE1 230
Purpose 230
Entry 230
Exit 230
Operation 230
SPECINIT 231
Purpose 231
Entry 231
Exit 231
Operation 231
INITIATE4 232
Purpose 232
Entry 232
Exit 232
Operation 232
INITIATE5 233
Purpose 233
Entry 233
Exit 233
Operation 233
RDCARD 233
Purpose 233
Entry 234

Exit 234
Operation 234
RCD 234
Purpose 234
Entry 234
Exit 234
Operation 234
INTER 235
Purpose 235
Entry 235
Exit 235
Operation 235
NOIPRI 235
Purpose 235
Entry 236
Exit 236
Operation 236
NACN 237
Purpose 237
Entry 237
Exit 237
Operation 237
NOTLB 238
Purpose 238
Entry 238
Exit 239
Operation 239
GETFILE 239
Purpose 239
Entry 240
Exit 240
Operation 240
FITCHKS 240
Purpose 240
Entry 241
Exit 241
Operation 241
BLD:BOF 242
Purpose 242
Entry 242
Exit 242
Operation 242
GETMIX 243
Purpose 243
Entry 244
Exit 244
Operation 244
QUEREC 245
Purpose 245
Entry 245
Exit 245
Operation 245
GETKEYI 246
Purpose 246
Entry 246
Exit 246
Operation 246
MOVEKEY 246
Purpose 246
Entry 246
Exit 247
Operation 247

MOVE 247
Purpose 247
Entry 247
Exit 247
Operation 248
GETKEY 248
Purpose 248
Entry 248
Exit 248
Operation 249
GETKEYN 249
Purpose 249
Entry 249
Exit 249
Operation 249
SCHEDULE 250
Purpose 250
Entry 250
Exit 250
Operation 250
QUEMIX 250
Purpose 250
Entry 251
Exit 251
Operation 251
RANFILE 251
Purpose 251
Entry 251
Exit 252
Operation 252
GETTBUF 252
Purpose 252
Entry 252
Exit 252
Operation 253
GETIBUF 253
Purpose 253
Entry 253
Exit 253
Operation 253
GETFOUR 253
Purpose 253
Entry 253
Exit 254
BUILD 254
Purpose 254
Entry 254
Exit 254
BUILDN 254
Purpose 254
Entry 254
Exit 254
QSECTOR 255
Purpose 255
Entry 255
Exit 255
DISCIO2 255
Purpose 255
Entry 255
Exit 256
DENAC 256
Purpose 256

Entry 256
Exit 256
FITENAC 256
Purpose 256
Entry 256
Exit 256
MTIO 257
Purpose 257
Entry 257
Exit 257
WTENAC 257
Purpose 257
Entry 257
Exit 257
SENTENAC 258
Purpose 258
Entry 258
Exit 258
NEWREEL 258
Purpose 258
Entry 258
Exit 258
WRTDAT 259
Purpose 259
Entry 259
Exit 259
DCTXERR/OPNABN/OPNERR 259
Purpose 259
Entry 259
Exit 259
GOEOR 259
Purpose 259
Entry 260
Exit 260
EOFQ 260
Purpose 260
Entry 260
Exit 260
BOFQUE 260
Purpose 260
Entry 260
Exit 260
MOVEI 261
Purpose 261
Entry 261
Exit 261
WRTMARK 261
Purpose 261
Entry 261
Exit 261
NOTENUFF 261
Purpose 261
Entry 261
Exit 262
ENDUP 262
Purpose 262
Entry 262
Exit 262
ADDONE 262
Purpose 262
Entry 262
Exit 262

ADERROR 263
Purpose 263
Entry 263
Exit 263
IORUNDOWN 263
Purpose 263
DISPRUNTOTL 263
Purpose 263
Entry 263
Exit 263
DTOGRAN 263
Purpose 263
Entry 264
Exit 264
FDERR 264
Purpose 264
Entry 264
Exit 264
FITSNAP/FITERR 264
Purpose 264
Entry 264
Exit 264
LISTFD 265
Purpose 265
Entry 265
Exit 265
LISTFIT 265
Purpose 265
Entry 265
Exit 265
LISTOUTBUF 265
Purpose 265
Entry 265
Exit 265
LISTMIX 266
Purpose 266
Entry 266
Exit 266
LISTAD 266
Purpose 266
Entry 266
Exit 266
MIXSNAP/MIXERR 266
Purpose 266
Entry 266
Exit 266
DATAERR 267
Purpose 267
Entry 267
Exit 267
FDDONE 267
Purpose 267
Entry 267
Exit 267
ENDOFD 267
Purpose 267
Entry 268
Exit 268
CKTIO 268
Purpose 268
Entry 268
Exit 268

CODESCAN 268
Purpose 268
Entry 268
Exit 268
MOVENTRY 269
Purpose 269
Entry 269
Exit 269
HEXTODEC 269
Purpose 269
Entry 269
Exit 269
TACNT/TFILNME 269
Purpose 269
Entry 269
Exit 270
ACCK 270
Purpose 270
Entry 270
Exit 270
READFAIL 270
Purpose 270
Entry 270
Exit 270
READFAIL2 270
Purpose 270
Entry 270
Exit 270
READFAIL3 271
Purpose 271
Entry 271
Exit 271
READFAIL5 271
Purpose 271
Entry 271
Exit 271
LPRINT 271
Purpose 271
Entry 271
Exit 271
PLIST 272
Purpose 272
Entry 272
Exit 272
BUFSET 272
Purpose 272
Entry 272
Exit 272
PRINT 272
Purpose 272
Entry 272
Exit 272
SPACE 273
Purpose 273
Entry 273
Exit 273
TYPEIO/TYPEIO2 273
Purpose 273
Entry 273
Exit 273

FRES PROCESSOR 274
Purpose 274

Coding Conventions 274
Basic Philosophy 274

QTAP 274
Purpose 274
Usage 274
Input 275
Description 275

MTREAD 275
Purpose 275
Usage 275
Input 275
Description 275

OPNTAP 276
Purpose 276
Usage 276
Input 276
Description 276

SKPTMK 276
Purpose 276
Usage 276
Input 276
Description 277

NEXTBUF 277
Purpose 277
Usage 277
Input 277
Description 277

INITIAL 277
Purpose 277
Usage 277
Input 278
Description 278

CCI 278
Purpose 278
Usage 278
Description 278

SKP:BOF 279
Purpose 279
Usage 279
Description 279

GOT1 279
Purpose 279
Usage 279
Input 279
Description 279

CHECKIT 280
Purpose 280
Usage 280
Input 280
Description 280

ACCTOJIT 280
Purpose 280

Usage 280
Input 281
Description 281

GETVLP 281
Purpose 281
Usage 281
Input 281
Description 281

BUILD 282
Purpose 282
Usage 282
Input 282
Description 282

EXIT 282
Purpose 282
Usage 282
Input 283
Description 283

FILL Processor 284
Purpose 284
Modules 284
Description 284

BACKUP 286
Purpose 286
Overview 286
Usage 286
Input 287
Output 287
Interaction 288
Subroutines 288
Errors 289
Restrictions 289
Description 289
Flowcharts 291

FILL – Fill Restoring 295
Purpose 295
Usage 295
Input 295
Output 295
Interaction 295
Data Bases 296
Subroutines 296
Error and Messages 296
Restrictions 298
Description 298
Flowcharts 300

PURGE 302
Purpose 302
Usage 302
Input 302
Output 302
Interaction 304
Subroutines 304
DOPURLIST 305
DOPURDEL 305
PURGEINT 305

PURGELOG ____ 305
DOPURGE ____ 305
Restrictions ____ 306
Description ____ 306
Flowcharts ____ 308

FILL FORMATS ____ 314
Description ____ 314
FILL Tape Formats ____ 314
Data Formats ____ 315
FILL Disc File Formats ____ 316
SEL:FILL Table/:BREC Record ____ 318

UTS TECHNICAL MANUAL

ID

CHK - System consistency checks .

PURPOSE

CHK runs various system consistency checks in order to facilitate system debugging.

USAGE

CHK has two calling sequences.

General user checks:

```
LI, 14    n
BAL, 11   CHECK
```

where n is a code telling who is calling check. Currently n = 1 is idle and n = 2 is end of swap.

Running monitor or swapper page chains:

```
LI, 7     S:FPPH/M:FPPH
BAL, 11   T:PGCHK
```

T:PGCHK is called by the swapper at six critical points.

OVERVIEW

CHK consists of two routines; CHECK and T:PGCHK. CHECK runs all the state queues, gathering and checking various data on all users in the system, or does nothing. The presence of this routine is controlled by an assembly parameter BIGCHK. BIGCHK SET 0 causes the null routine to be assembled. BIGCHK SET 1 causes the checking software to be generated. For debugging purposes BIGCHK SET 1 might be desirable but for working systems BIGCHK must be set to 0 since the routine runs disabled and its execution is quite lengthy (unrecognized disc pack interrupts etc. can result if the number of users is high).

T:PGCHK runs any page chain to see if the head, tail, and count are consistent.

Both routines are executed only if SSW4 is set (up).

INPUT

Input for check consists of:

SNSTS	the number of states
SB:HQ	the head of each state chain
UH:FLG	the user flags

UB:JIT	the user's physical JIT page
UB:US	the user's state
SB:HIR	the list of SIR states
SB:EXU	the list of executable states
PTEL	TEL's processor number
PCCI	CCI's processor number
UB:APR UB:APO UB:ASP UB:DB UB:OV	the user's associated processor tables
UB:FL	the state queue link
S:CUIS	the current number of users in the system
S:SIR	the number of users ready to SWAP in
S:HIR	the number of users of high priority who are ready to run.

OUTPUT

Software checks 2, 3, 4, 6 or 7 from CHECK and 1 from T:PGCHK if inconsistencies are found, otherwise none.

SUBROUTINES

SHIRE in SSS is used to find out if a user is in an SIR or HIR state.

ERRORS

Software check
- 2 - user in wrong state queue
- 3 - bad JIT
- 4 - wrong number of users
- 6 - bad SIR count
- 7 - bad HIR count

DESCRIPTION

CHECK zeros internal counts for SIR, HIR and processor usage counts (UC). It then runs each state queue starting at SB:HQ and linking through UB:FL to find all users in the

system. For each user it adds one to R9, the count of users in the system (unless the user is waiting to log in), it checks his state UB:US against the index of the state queue it is running (software check 2), adds one to HIR if he is ready to run and his state is in SB:HIR, adds one to SIR if he is not ready to run and his state is in SB:EXU; and if his JIT is in memory, increments the internal usage count of all his associated shared processors and checks his JIT by verifying the temp stack pointer (software check 3).

Upon conclusion of running all users, the number of users found (register 9) is checked against S:CUIS the current number of users in the system (software check 4). SIR and HIR are compared with S:SIR and S:HIR (software check 6 and 7). CHECK then exits.

T:PGCHK begins at the head of the page chain pointed to by register 7 and follows it through MB:PPUT, counting pages as it goes, until it encounters the end of the chain (a zero in MB:PPUT). It then compares the count and tail it calculated with the count and tail pointed to by register 7. If either disagrees it causes a software check 1.

ID

Diagnostic and Exerciser Interface

PURPOSE

Direct access to card reader, card punch, line printer and other symbiont devices as well as mag tape is provided through a varient of the M:OPEN CAL. The facility is provided primarily for interfacing on-line diagnostic and exerciser programs to UTS so that they may perform ordinary PM and maintenance activities concurrently with regular system operation.

OVERVIEW

The facility is implemented via a set of changes mainly to the open routines in the module OPN. To gain direct access to the symbiont device the user program, which must have a privilege of A0 or higher, issues an Open CAL in the special format given in the UTS Reference manual. If the CAL and privilege are correct and the device is not in use (symbionts must have been suspended by the operator) then the direct I/O connection is made and the program has exclusive use of the device. Erroneous usage is reported via abnormal exit codes as specified in the UTS Reference Manual (codes 9/00 through 9/03).

INTERFACE

Code elements relating to the diagnostic open also occur in the modules COOP to pass the I/O request directly to IOQ, in IOQ end action to place status information in the DCB, and in CLS to release the device from diagnostic use.

DATA BASE

Bit 2 of DCT3 is used to flag the device as in use by diagnostic program.

Bit 2 of word 5 of the calling DCB is used to switch control directly to IOQ and to cause IOQ to place status information from the transfer in the DCB.

Words 15 through 18 of the DCB, which usually contain tab stops, are used to return status:

DCB Word			
15	0————0	AIO status
16	TDV status	TDV byte count
17	0————0	TDV cmd. addr
18	TIO status	TIO byte count

DESCRIPTION

The code in OPN to implement the diagnostic direct-device open follows a test of the device field in the FPT which indicates that a device (or free-form tape) is to be Opened. If the user's privilege (JB:PRIV) is less than A0, an abnormal code of 9/00 is returned. If the I/O address from the device field of the DCB (word 12, bits 21-31) is not a symbiont device, i.e. does not provide a DCT1 index, abnormal code 9/01 is returned. If the diagnostic-use bit (DCT3, bit 2) is already set, abnormal code 9/02 is returned. The symbiont table SNDDX is searched for the required DCT index (or tape index) and if not found, abnormal code 9/01 is returned. If found but the symbiont is active (SSTAT=1), abnormal code 9/03 is returned. Finally the diagnostic use bit is set and the direct-IOQ bit (DCB, word 5, bit 2) is set and Open continues.

In addition code at the beginning of COOP checks DCB word 5 bit 2 for the diagnostic use flag and passes control to IOQ directly if it is set.

Code in IOQ at REQCOM examines the same bit, and if set, places the device status information in the DCB.

Code in CLS near CLSDEV resets the DCT3 bit 2 flagging diagnostic use. DCB word 5, bit 2 is reset at the beginning of every open.

ID

ERRLOG - Error Logging Routine

PURPOSE

ERRLOG is called by the various parts of the system which detect errors, such as I/O errors, memory parity, file and symbiont inconsistencies, etc. When an error is detected by some routine, it builds a message of up to 10 words and calls ERRLOG. ERRLOG packs the message into a buffer, when it becomes full, writes the buffer on a special RAD file and, each time five errors have been logged, awakens the ERR:FIL ghost to copy the errors from the special file to a keyed file.

USAGE

ERRLOG is called with the address of the error message in register 6. The message may be in registers or in memory. If the message is in registers, it may wrap around, e.g., it may start in register 12 and end in register 3. The format of each message is described in Section VK.

```
LI, 6       message address
BAL, 5      ERRLOG
return
```

All registers are non-volatile

INPUT

As described in USAGE

OUTPUT

The special error log file as described in Section VK.

INTERACTION

ERRLOG calls the T:GJOBSTRT subroutine to wake up or initiate the ERR:FIL ghost. QUEUE is called to write ERRLOGs special RAD file. GSG is called to get symbiont granules for the special file.

DATA BASE

The core buffers and their controlling pointers are described in Section VK.

DESCRIPTION

When called ERRLOG saves all registers in the monitor's TSTACK and checks to see

if any new RAD granules are needed for its special file. If any are needed, GSG is called to get them.

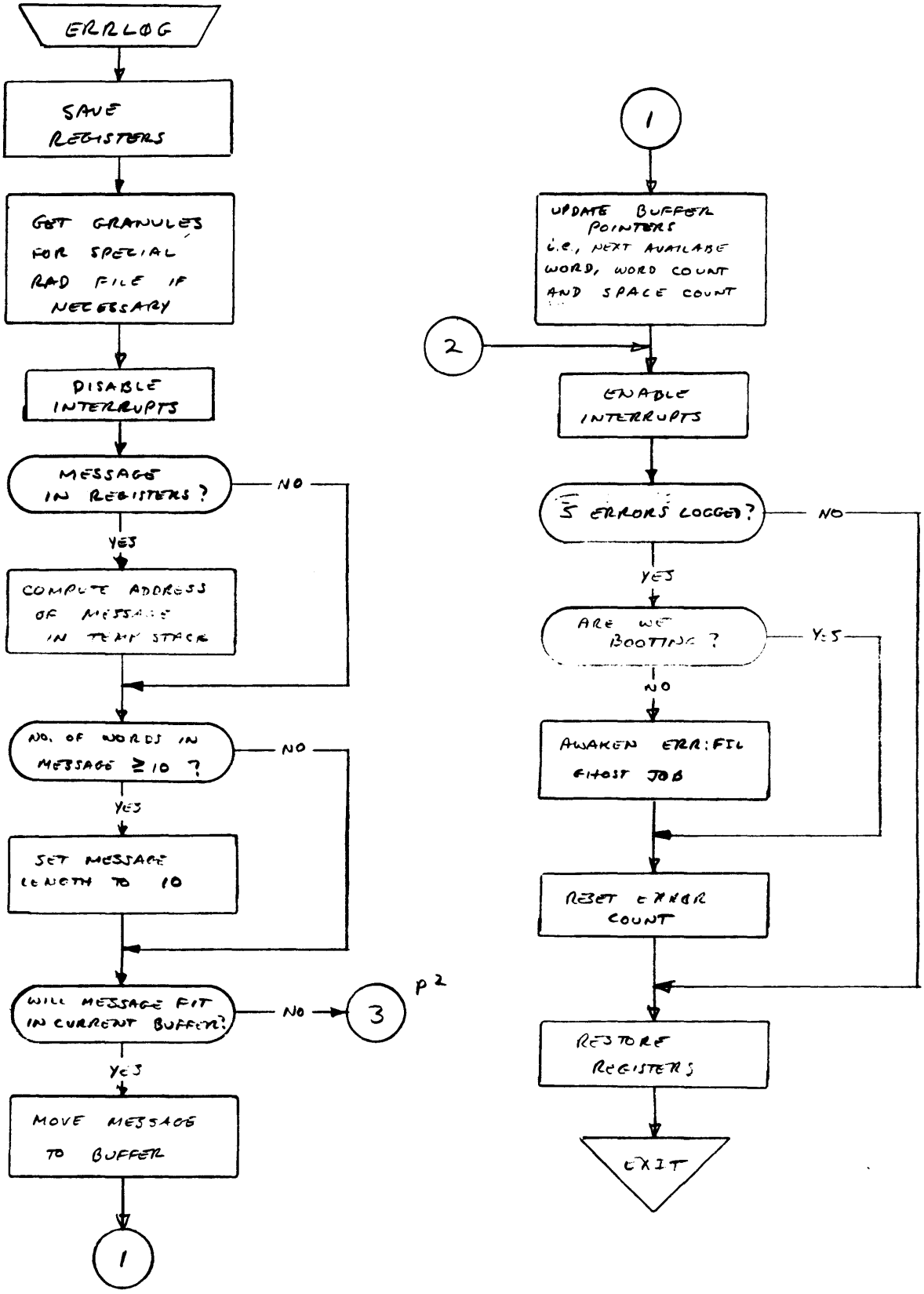
Interrupts are disabled and if the message was passed to ERRLOG in registers, the address of the message in TSTACK is computed. The length of the message is checked and if it is greater than 10 words it is set to 10 so that at least part of the message will be recorded.

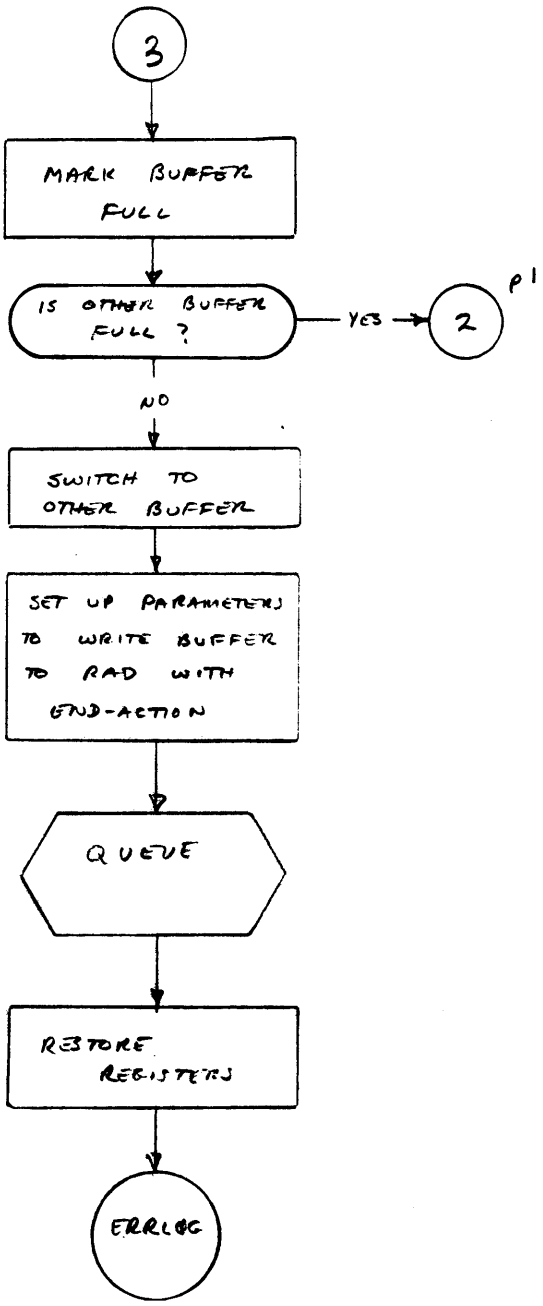
Next, the current buffer is checked to see if there is room for the message. If there is not sufficient space in the buffer, the other buffer is checked. If it is full, the message is ignored, the registers are restored and ERRLOG returns to the calling routine. If it is not full, the first buffer is marked full, the forward and backward links for the special file are put in it, the core pointers for the file are updated and QUEUE is called to write the buffer to the file. The registers are then restored and control goes back to the entry point of ERRLOG so that the message will be logged in the other buffer. The end-action routine that is specified when the full buffer is written out merely re-initializes the pointers for that buffer so that it may be used again.

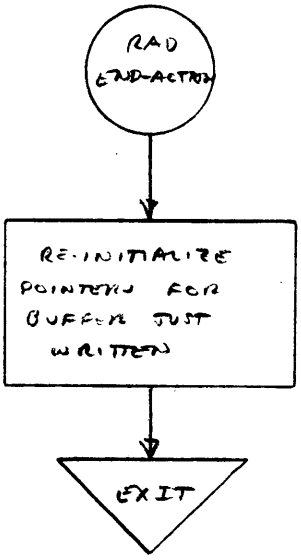
If there is sufficient space in the buffer for the message, the message is moved to the buffer and the buffer pointers are updated.

A counter is then incremented and checked to see if 5 errors have been logged since ERR:FIL was last awakened. If not, the registers are restored and ERRLOG returns to the calling routine. If 5 errors have been logged, a check is made to see if we are booting. If we are, ERR:FIL cannot be awakened so the registers are restored and ERRLOG returns to the calling program. If we are not booting, T:GJOBSTRT is called to awaken ERR:FIL.

FLOWCHART (GENERAL)




3
MARK BUFFER FULL
IS OTHER BUFFER FULL ?
YES
2
P1
NO
SWITCH TO OTHER BUFFER
SET UP PARAMETERS TO WRITE BUFFER TO RAD WITH END-ACTION
QUEUE
RESTORE REGISTERS
ERRLOG


RAD END-ACTION
RE-INITIALIZE POINTERS FOR BUFFER JUST WRITTEN
EXIT

ID

ERR:FIL - Copy Error Log

PURPOSE

ERR:FIL copies the special file created by ERRLOG onto a normal keyed file that is more readily available to diagnostic programs. While copying ERRLOG's file, ERR:FIL compiles a summary of the errors copied. A file of these summaries, which contains a summary of errors for each hour of operation and a master summary of all errors, is also maintained by ERR:FIL.

USAGE

ERR:FIL is a ghost job that is awakened by ERRLOG whenever 5 errors have been recorded. ERR:FIL may also be awakened by a program with diagnostic privilege by using the intitiate job CAL(CAL1, 6FPT) or by an operator Keyin of GJOB ERR:FIL. ERR:FIL may also be run by an on-line user running under account :SYS with diagnostic privilege by typing !ERR:FIL.

INPUT

Input to ERR:FIL is the special file written by ERRLOG. This file and the format of the entries in the file are described in Section VK of this manual. The core buffers of ERRLOG are also input and are likewise described in Section VK.

OUTPUT

ERR:FIL builds a detailed error log file (ERRFILE) and an error summary file (SUMFILE). These files are described under DATA BASE.

INTERACTION

Monitor services used by ERR:FIL are:

M:OPEN, M:READ, M:WRITE and M:CLOSE are used to construct and update ERRFILE and SUMFILE.

M:TIME	used to get ti ne and date for keys
M:EXIT	return to the monitor
T:RDERLOG	used to read the special file constructed by ERRLOG (CAL1, 6)

DATA BASES

ERRFILE is a keyed file built and updated by ERR:FIL for use by diagnostic programs. The file contains one record for each error entry in the file created by ERRLOG.

The format of each record is identical to the format of the error entries in ERRLOG's file. (The format of these entries is described in Section VK.) The keys for this file contain the Julian date in packed decimal, the time of the error in EBCDIC and a sequence number for errors with the same time tag. This sequence number is reset to zero for each entry with a new time tag. The format of the key is:

08	YY	OD	DD
H	H	M	M
N			

08 = number of bytes in key

YYODDD = Julian date in packed decimal

HHMM = Time (hours & minutes) in EBCDIC

N = Sequence number

The first record of ERRFILE is the key of the last record in ERRFILE and has a key of zero.

Several consistency checks are made while copying the error data from ERRLOG's file to ERRFILE. If an error is detected, a "copying error" record is written on ERRFILE. The key for such a record has the same date and time as the previous record and the sequence number is one greater than that of the previous record. The format of the "copying error" record and the errors that cause them to be written are described in Figure KE.02-2.

SUMFILE is a keyed file built and updated by ERR:FIL for use by ERR:SUM (described in Section KE.03). The first record of SUMFILE is the key of the last record of SUMFILE and has a key of zero. The second record of the file is the master error summary and has a key of one. Subsequent records of the file are the summaries of errors for each hour of operation. The keys of these records begin at two and are incremented for each successive record. The master error summary is the summation of the hour summaries. The format of the summary records (master and hourly) is detailed in Figure KE.02-1. The format of the keys for SUMFILE is:

01	N

01 = number of bytes in key
N = sequence number

SUBROUTINES

NEWTIME	converts the ERRFILE key to date and time in the format used by the M:TIME routine.
UPDATE	updates the master error summary, i.e., adds the error counts of the current hour summary to the error counts of the master summary.
ERRWRITE	constructs the "copying error" record and writes it on ERRFILE. Also writes the 256 word record just read from ERRLOG's file if necessary. The reading of the ERRLOG file consists of 256 word records, only the first 64 being meaningful.
SUMINIT	initializes the summary record
JULIAN	converts date and time returned by M:TIME to Julian date and time in packed decimal (described in detail in section UA).

DETAILED DESCRIPTION

When ERR:FIL is awakened, an attempt to open ERRFILE in the update mode is made. If the attempt is successful, the first record of ERRFILE is read to get the last key used. If the attempt is unsuccessful, the error code is checked. If ERRFILE exists, ERR:FIL exits. If the file does not exist, it is opened in the output mode, M:TIME and JULIAN are used to get the date to initialize the key, the key is written on ERRFILE, ERRFILE is closed and reopened in the update mode.

After ERRFILE has been opened, an attempt is made to open SUMFILE in the update mode. If the attempt is successful, the key of the last record of the file is read and then the last record, which is an incomplete hour summary, is read. If the attempt is unsuccessful, the error code is checked. If the file exists, it is considered to be in error and is marked for release. Processing continues as though the file was successfully opened and therefore nothing will be added to SUMFILE. If SUMFILE does not exist, it is opened in the output mode, the initial key is written followed by two copies of the initialized summary record. This is done to initialize the master summary and the first hour summary. SUMFILE is then closed and reopened in the update mode.

When both files are opened in the update mode, T:RDERLOG is used to read a record from ERRLOG's file. The condition codes set by T:RDERLOG are checked. If they indicate that no errors have been logged or that this job has insufficient privilege, the files are "cleaned-up" and ERR:FIL exits.

"Cleaning up" the files consists of saving the key of the last record of each file, saving the incomplete summary in SUMFILE and closing the files. If either file did

not exist when ERR:FIL was entered and if no errors were copied, the file is released when it is closed. Otherwise, it is saved.

If a record was read by T:RDERLOG, each entry in the record is copied onto ERRFILE as a keyed record. The time of each entry is checked as it is copied. If it is in the same hour as the previous entry the proper counter in the summary record is incremented and the next entry is processed. If the entry is not in the same hour, the master summary is read, updated and rewritten, the hour summary is written over the last record in SUMFILE and an initialized summary record is written as the next record in SUMFILE. Copying of the errors is then continued.

When the record read by T:RDERLOG is exhausted, the condition codes set by T:RDERLOG are checked again. If they indicate that there is no more data available, ERRFILE and SUMFILE are "cleaned-up" and ERR:FIL exits. If there is more data available, another record is read and processed. This continues until ERRLOG's file and its core buffers have been emptied.

While copying the ERRLOG file, consistency and error checks are made on the input data. If any errors or inconsistencies are found "copy error" records are written and a "copy error" counter in the summary record is incremented. The format of the "copy error" records is given in Figure KE.02-2. The error and consistency checks and the recovery actions taken are described below.

1) T:RDERLOG READ ERROR - If the condition codes set by T:RDERLOG indicate a read error, a "copy error" record is written and copying of the record is attempted. If no inconsistencies are found in the record, another "copy error" record is written after the last entry of the record to indicate the end of the questionable data.

2) ERRLOG RECORD LENGTH ERROR - If the length of the ERRLOG record is greater than 256, a "copy error" record followed by the ERRLOG record is written on ERRFILE. No attempt is made to copy this record in the detailed format.

3) ILLEGAL ENTRY TYPE - If the entry type is not one of the legal types, a "copy error" record followed by the ERRLOG record is written on ERRFILE. No attempt is made to copy the remainder of the record.

4) ILLEGAL ENTRY LENGTH - If the length byte of an entry does not correspond to the entry type, a "copy error" record is written on ERRFILE followed by the ERRLOG record. No attempt is made to copy the remainder of the record.

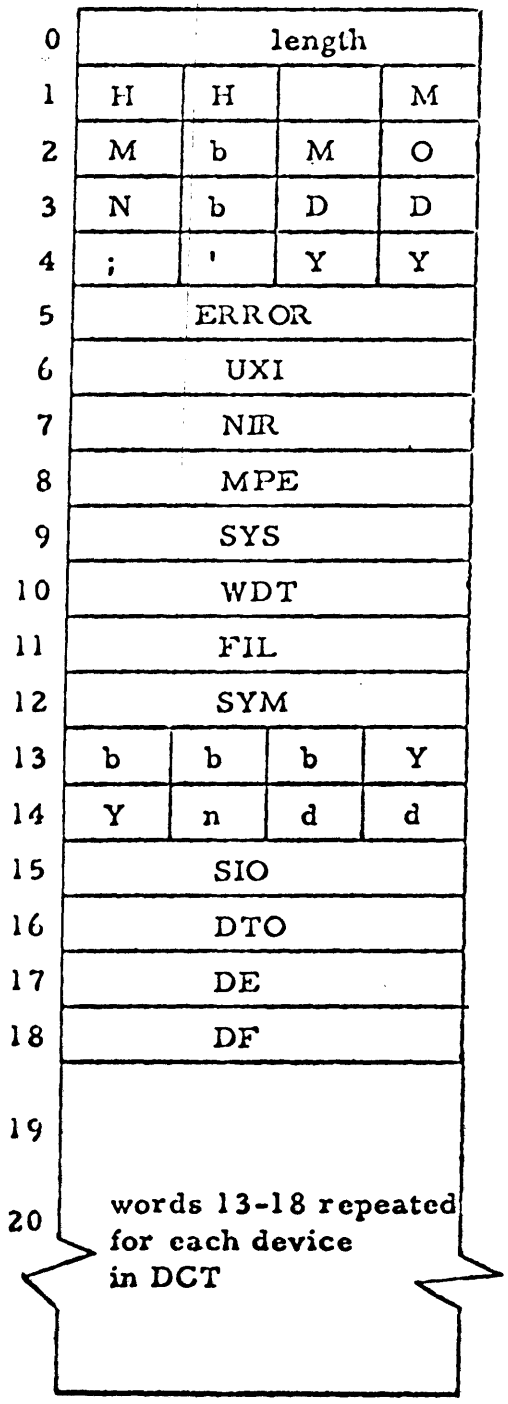
5) INCORRECT TIME - If the time of an entry is out of sequence, i.e., if it is

earlier than the time of the last record and the date has not changed, a "copy error" record is written on ERRFILE followed by the ERRLOG record. The time of this entry is then used for the key and processing continues.

NOTE: Errors that occur while booting have a time tag of 24XX but the keys of these records contain the current date and 0000 is for the time.

If read or write errors are detected while reading or writing ERRFILE and SUMFILE, they are ignored.

FIG. KE. 02-1 Summary Record Format



number of words in record

Time - the time of the first error included in this summary for the master summary and open summary records - for closed summary records, it is effectively the closing or completion time.

count of "copy error" records

count of unexpected interrupt entries

count of unrecognized interrupt entries

count of memory parity error entries

count of system start up entries

count of watch dog timer trap entries

count of file inconsistency entries

count of symbiont inconsistency entries

EBCDIC device name

count of SIO failure entries for this device

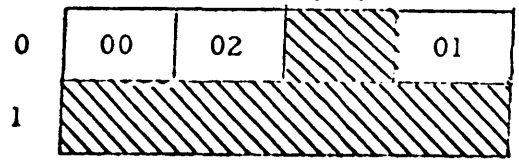
count of Device Timeout entries for this device

count of Device error entries for this device

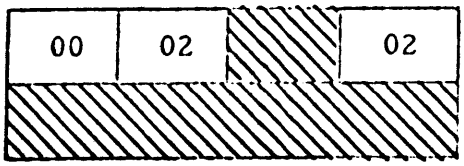
count of Device Failure entries for this device

FIG. KE.02-2 COPY ERROR RECORD FORMAT

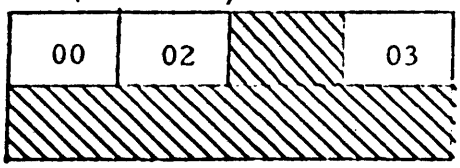
1) Read error - record precedes first entry copied from an ERRLOG record not read correctly by T:RDERLOG



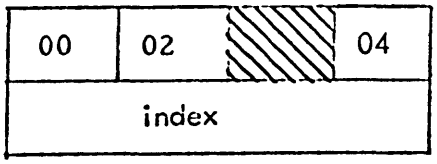
2) End read error - record follows last entry copied from an ERRLOG record not read correctly by T:RDERLOG (only if no inconsistencies appeared in the record)



3) Illegal record length - length of ERRLOG record incorrect (followed by 256 word ERRLOG record)

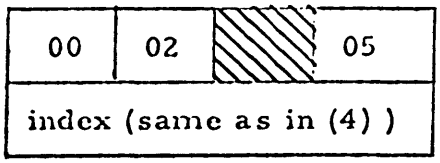


4) Incorrect entry length - length of an entry does not correspond to its entry type (followed by 256 word ERRLOG record)

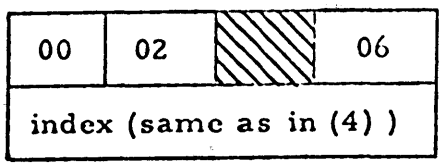


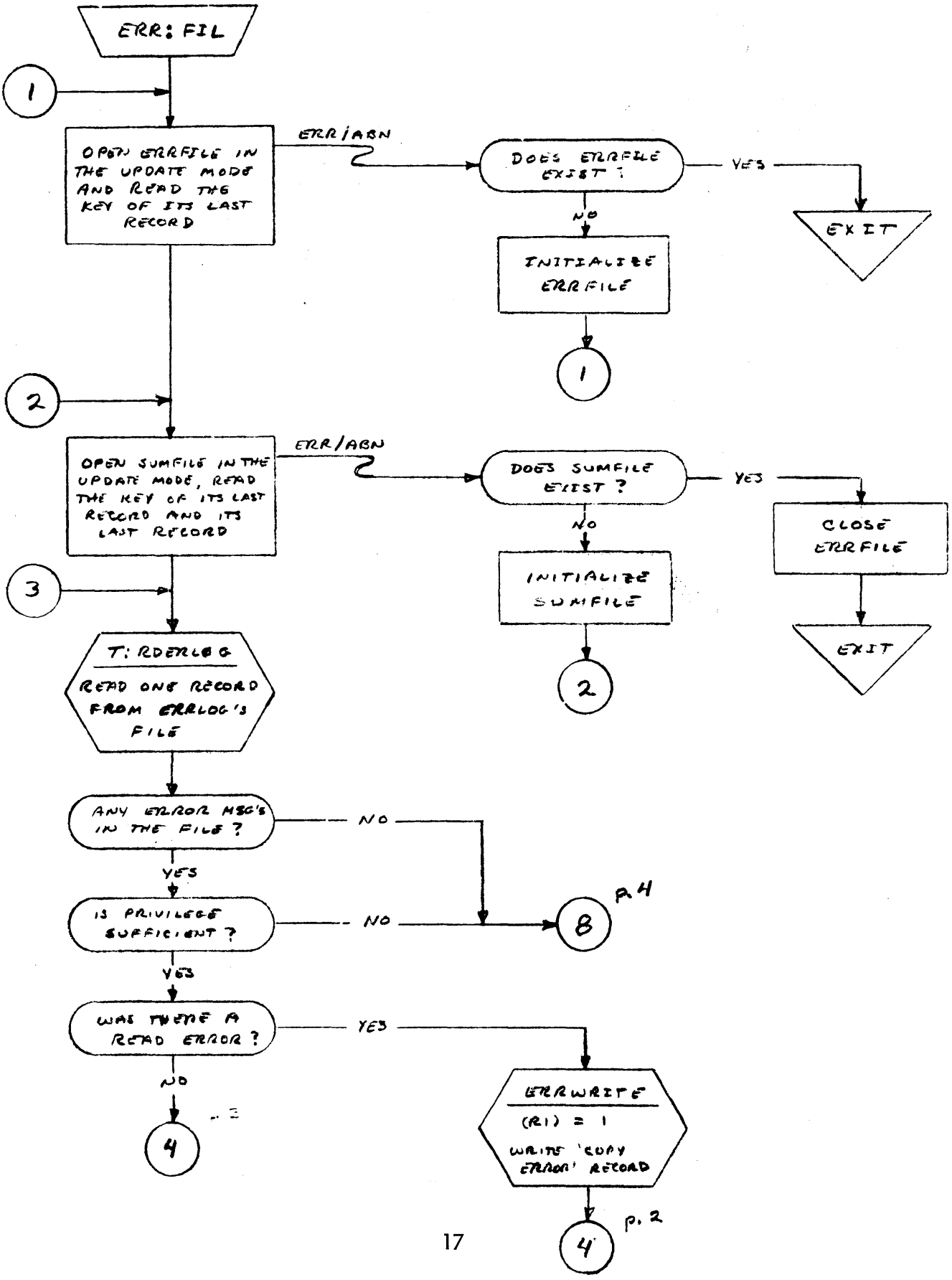
index = displacement within ERRLOG record of first word of erroneous entry

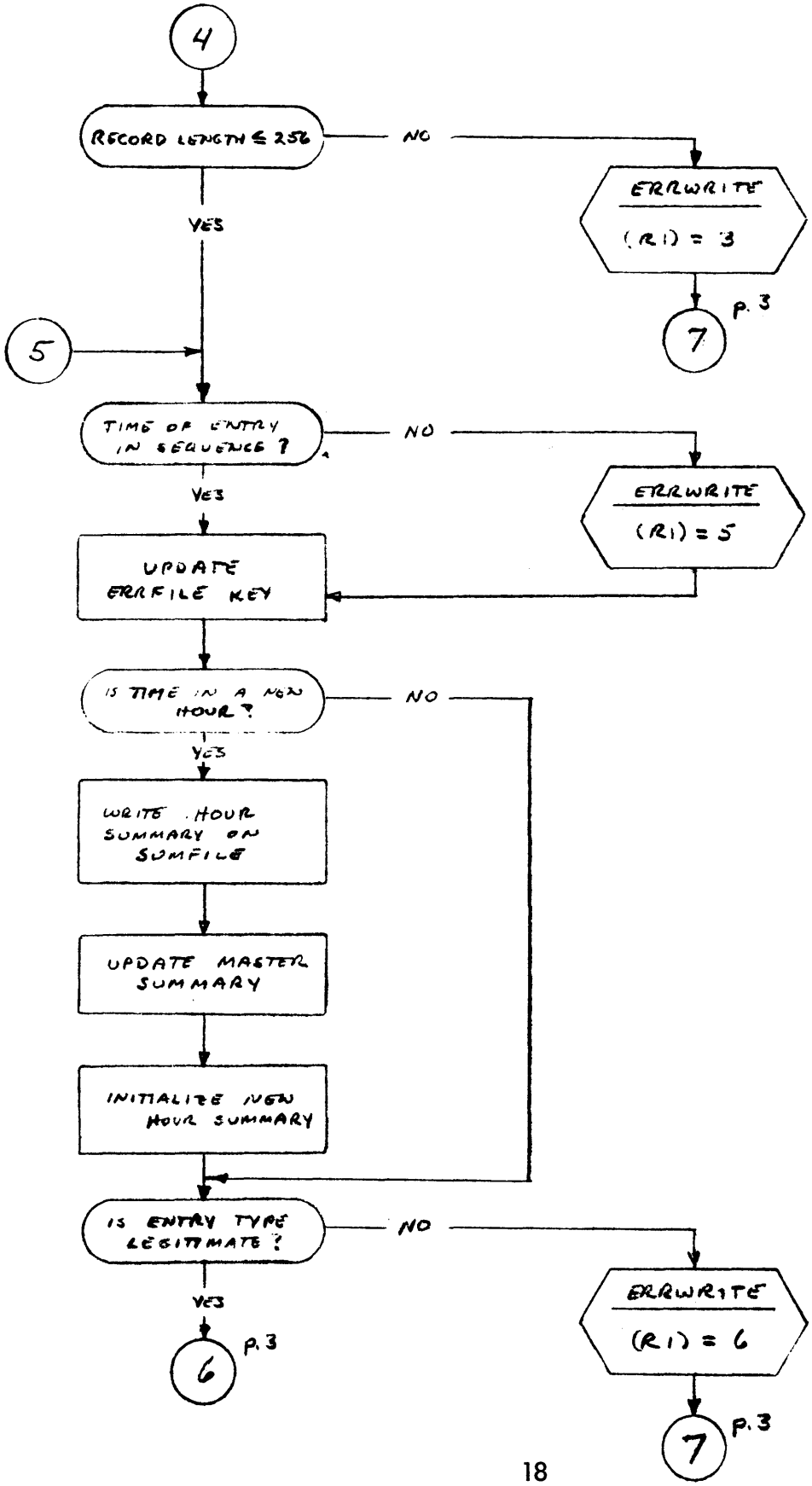
5) Incorrect time - the time of an entry is less than the previous time (followed by 256 word ERRLOG record)

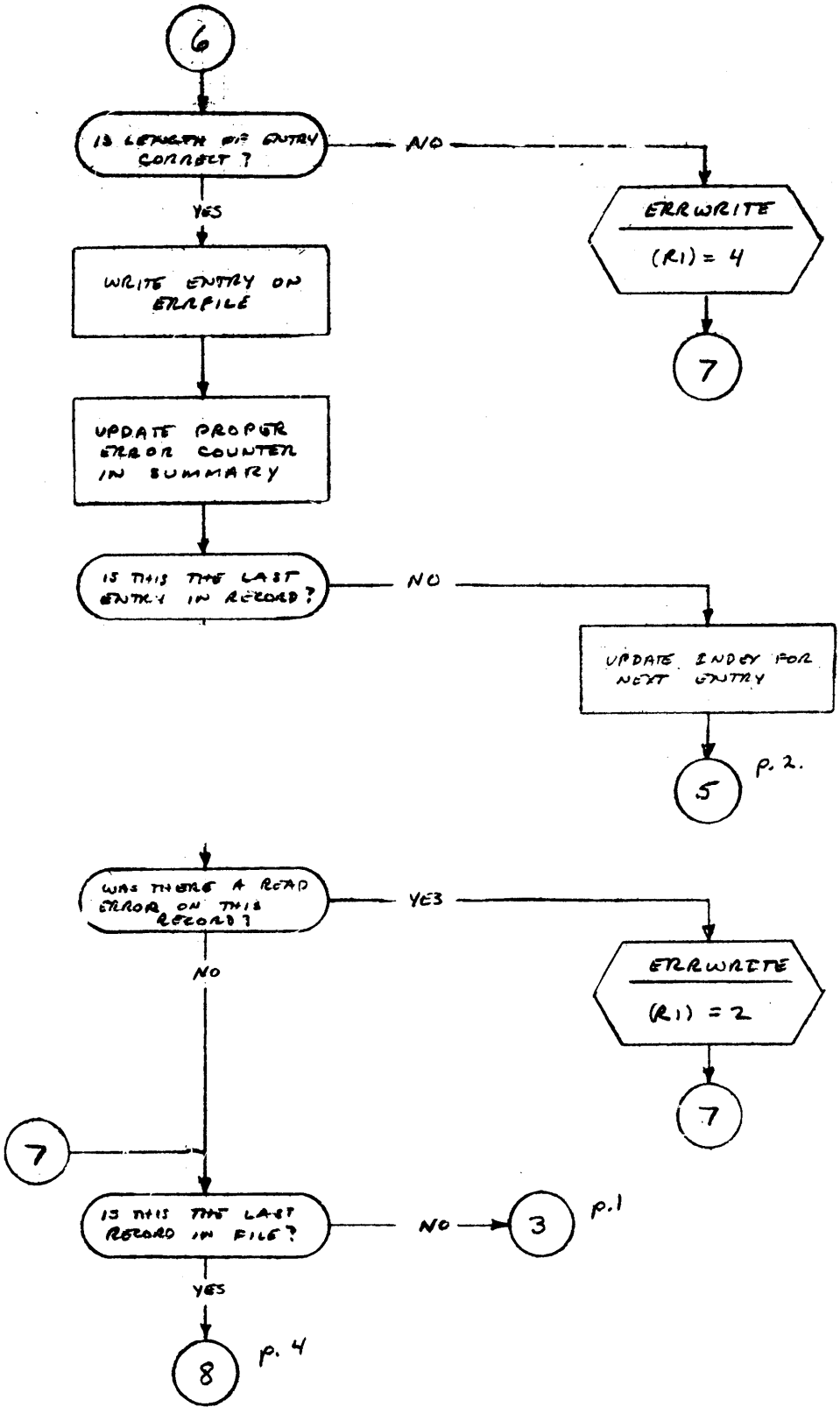


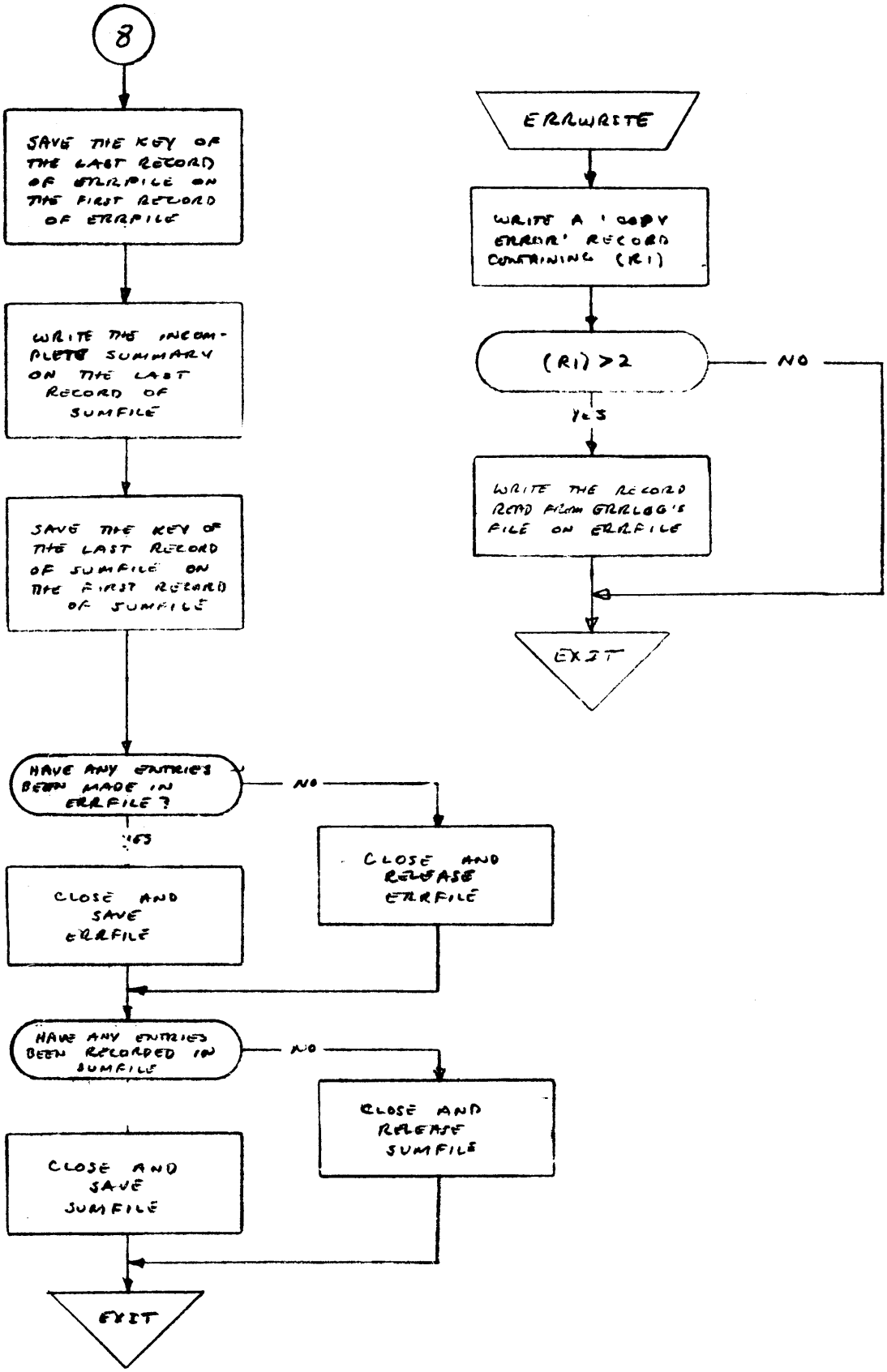
6) Illegal entry type - entry type not one of the 10 legitimate entry types (followed by 256 word ERRLOG record)




ERR: FIL
1
OPEN ERRFILE IN THE UPDATE MODE AND READ THE KEY OF ITS LAST RECORD
ERR/ABN
DOES ERRFILE EXIST ?
YES
EXIT
NO
INITIALIZE ERRFILE
1
2
OPEN SUMFILE IN THE UPDATE MODE, READ THE KEY OF ITS LAST RECORD AND ITS LAST RECORD
ERR/ABN
DOES SUMFILE EXIST ?
YES
CLOSE ERRFILE
EXIT
NO
INITIALIZE SUMFILE
2
3
T: RDERLOG
READ ONE RECORD FROM ERRLOG'S FILE
ANY ERROR MSG'S IN THE FILE ?
NO
YES
IS PRIVILEGE SUFFICIENT ?
NO
B
p. 4
YES
WAS THERE A READ ERROR ?
YES
NO
4
p. 3
ERRWRITE
(R1) = 1
WRITE 'COPY ERROR' RECORD
4
p. 2


4
RECORD LENGTH ≤ 256
NO
ERRWRITE
(R1) = 3
YES
p. 3
7
5
TIME OF ENTRY IN SEQUENCE ?
NO
ERRWRITE
(R1) = 5
YES
UPDATE ERRFILE KEY
IS TIME IN A NEW HOUR ?
NO
YES
WRITE HOUR SUMMARY ON SUMFILE
UPDATE MASTER SUMMARY
INITIALIZE NEW HOUR SUMMARY
IS ENTRY TYPE LEGITIMATE ?
NO
ERRWRITE
(R1) = 6
YES
p. 3
6
p. 3
7


6
IS LENGTH OF ENTRY CORRECT ?
NO
YES
ERRWRITE
(R1) = 4
7
WRITE ENTRY ON ERRFILE
UPDATE PROPER ERROR COUNTER IN SUMMARY
IS THIS THE LAST ENTRY IN RECORD ?
NO
UPDATE INDEX FOR NEXT ENTRY
5
p. 2.
WAS THERE A READ ERROR ON THIS RECORD ?
YES
NO
ERRWRITE
(R1) = 2
7
7
IS THIS THE LAST RECORD IN FILE ?
NO
3
p.1
YES
8
p. 4


8
SAVE THE KEY OF THE LAST RECORD OF ERRFILE ON THE FIRST RECORD OF ERRFILE
WRITE THE INCOMPLETE SUMMARY ON THE LAST RECORD OF SUMFILE
SAVE THE KEY OF THE LAST RECORD OF SUMFILE ON THE FIRST RECORD OF SUMFILE
HAVE ANY ENTRIES BEEN MADE IN ERRFILE?
NO
YES
CLOSE AND SAVE ERRFILE
CLOSE AND RELEASE ERRFILE
HAVE ANY ENTRIES BEEN RECORDED IN SUMFILE
NO
CLOSE AND SAVE SUMFILE
CLOSE AND RELEASE SUMFILE
EXIT
ERRWRITE
WRITE A 'COPY ERROR' RECORD CONTAINING (R1)
(R1) > 2
NO
YES
WRITE THE RECORD READ FROM ERRLOG'S FILE ON ERRFILE
EXIT

ID

ERR:SUM - Print Error Summary

PURPOSE

ERR:SUM reads the master error summary and the last hour summary from the error summary file (SUMFILE) constructed by ERR:FIL, prints the master summary (updated from the last hour summary) on the LO device and if requested by the user, purges SUMFILE.

USAGE

ERR:SUM is a LMN file which can be run under any account with diagnostic privilege. When executed ERR:SUM reads, and prints an updated version of error summary. ERR:SUM then asks the user if SUMFILE is to be deleted by typing the following question on the user's console.

"PURGE SUMMARY FILE?"

If the user responds:

"NO."

SUMFILE is saved and the message:

"SUMMARY FILE SAVED."

is typed on the user's console and ERR:SUM exits.

If the user responds:

"YES."

confirmation is requested by typing the following question.

"ARE YOU SURE?"

If the user responds

"YES."

SUMFILE is released, the message

"SUMMARY FILE PURGED."

is typed on the user's console and ERR:SUM exits.

If the user responds:

"NO."

he is again asked

"PURGE SUMMARY FILE?"

INPUT

The error summary file (SUMFILE) constructed by ERR:FIL. (SUMFILE is described under DATA BASE in Section KE.02 of this manual.)

OUTPUT

The format of the printed summary is given in FIG. KE.03-1

INTERACTION

Monitor services used by ERR:SUM are :

M:OPEN		
M:READ	-	used to read from SUMFILE and UC
M:WRITE	-	used to write on UC and LO
M:CLOSE		
M:DEVICE	-	used for vertical formatting on LO
SUPERCLOSE (CAL1, 9 6)	-	used to close LO symbiont file
M:EXIT		
CAL1, 6 6	-	used to awaken ERR:FIL ghost
M:WAIT	-	used to wait for ghost to run

SUBROUTINES

PRINTS	-	formats and prints error summary
DATALINE	-	format and prints the device associated
BCDBIN	-	converts a positive binary integer to an 8 character EBCDIC number with leading zeros suppressed
UPDATE	-	prepares an updated copy of the master summary, i.e., adds the error counts of an hour summary to the error counts of the master summary.
CHECK	-	examines the master summary record after it has been updated to see if any errors have been logged.

RESTRICTION

The summary file (SUMFILE) will be deleted only if ERR:SUM is being run in the :SYS account.

DESCRIPTION

ERR:SUM awakens ERR:FIL to insure that the SUMFILE is current via the awaken ghost CAL (CAL1, 6). If the account in which ERR:SUM is running does not have diagnostic privilege, the message:

"ERR:SUM EXIT: INSUFFICIENT PRIVILEGE."

is typed and ERR:SUM exits. If the account has sufficient privilege, the message:

"ERR:FIL RUNNING, PLEASE WAIT.

is typed and ERR:SUM waits for 6 seconds to allow the ghost job to run.

ERR:SUM then attempts to open SUMFILE in the input mode. If the file does not exist, ERR:SUM informs the user by typing

"SUMMARY FILE DOES NOT EXIST"

on his console and then exits. If the file cannot be opened for some other reason, ERR:SUM informs the user by typing

"SUMMARY FILE BUSY, PLEASE WAIT."

on his console, waits 6 seconds and then attempts to open the file again. This sequence will continue until the file becomes free.

When SUMFILE has been opened, the master summary record, the key of the last record and the last summary record are read from it. If an error occurs on any of these records, ERR:SUM informs the user by typing

"SUMMARY FILE READ ERROR"

on his console and exits. If no read errors occur, the subroutine UPDATE is called to add the error counts of the last summary record to the error counts of the master summary record. Subroutine check is then called to examine the updated master summary to see if any errors have been logged for this period. If no errors exist the message

"THERE ARE NO RECORDS FOR THIS PERIOD"

is typed on the user's console and exits. Otherwise, the subroutine PRINTS is then called to print the master summary.

After the master summary has been printed, ERR:SUM asks the user if SUMFILE is to be purged by typing

"PURGE SUMMARY FILE?"

on his console. The user responds by typing "YES" or "NO". If he types "YES" ERR:SUM asks him for confirmation by typing

"ARE YOU SURE?"

The user responds by typing "YES" or "NO". If he types "YES", SUMFILE is closed with the REL option, ERR:SUM informs the user that SUMFILE was purged by typing

"SUMMARY FILE PURGED"

on his console and exits. If he responds "NO", he is again asked if SUMFILE is to be purged. If the user's response to the original question was "NO", SUMFILE is closed with the SAVE option, ERR:SUM informs him that SUMFILE has been saved by typing

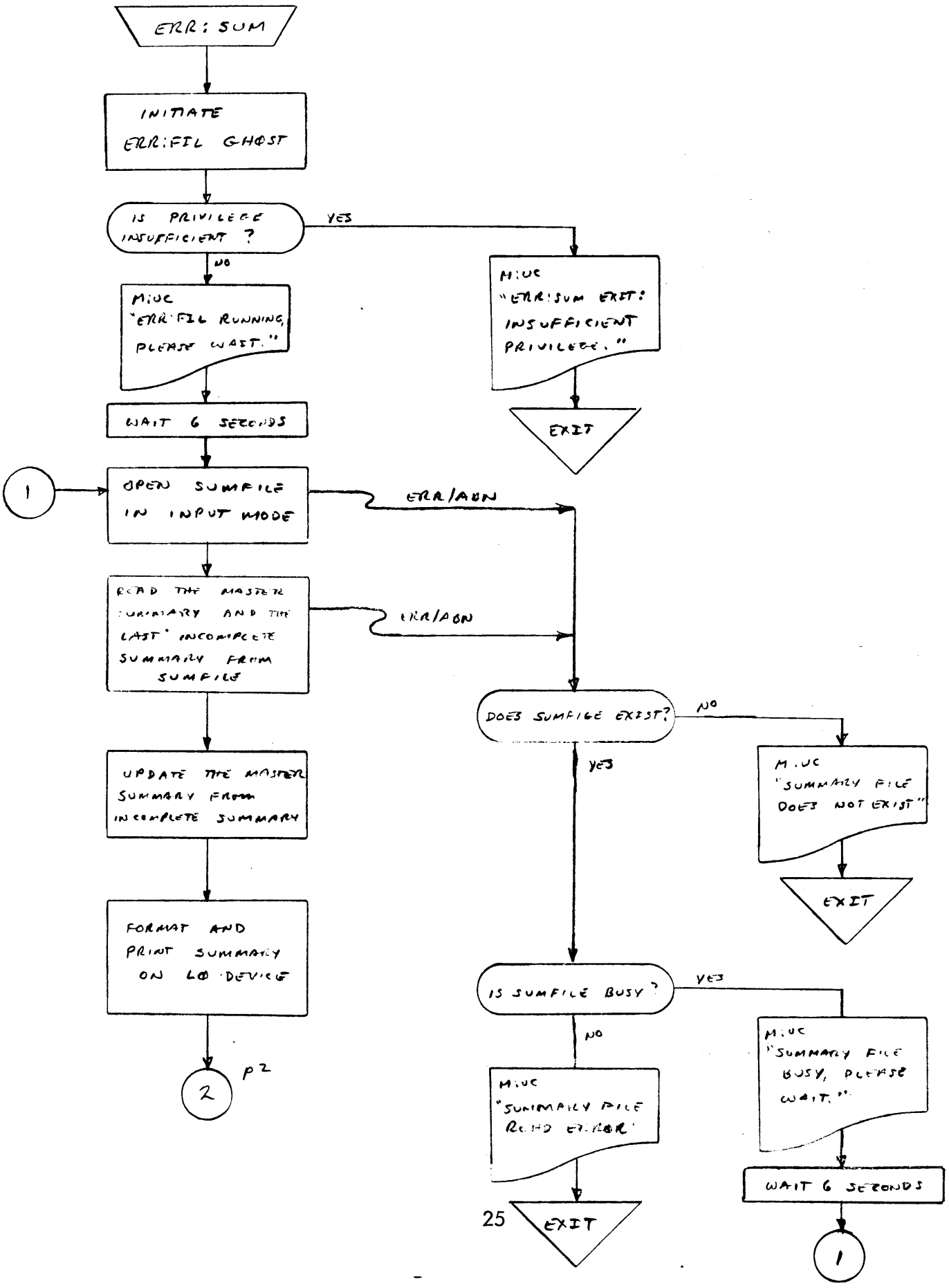
"SUMMARY FILE SAVED"

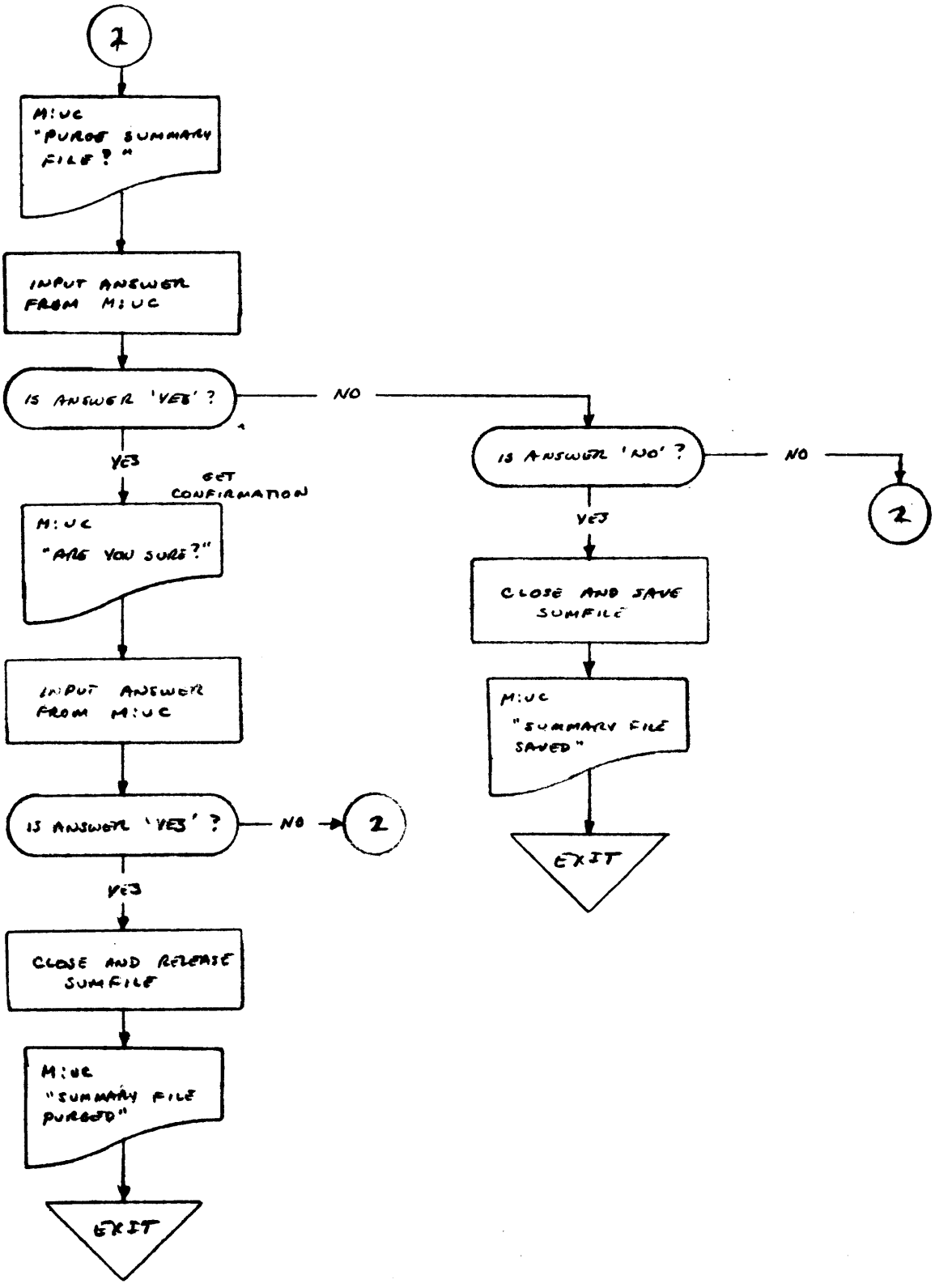
and exits.

FIGURE KE.03-1 ERROR SUMMARY LISTING FORMAT

(MASTER) ERROR LOG SUMMARY HH:MM MON DD, 'YY THROUGH HH:MM MON DD,' YY

DEVICE	SIO FAILURE	DEVICE TIMEOUT	DEVICE ERROR	DEVICE FAILURE	TOTAL
yyndd	xx	xx	xx	xx	xxx
yyndd	xx	xx	xx	xx	xxx
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
yyndd	xx	xx	xx	xx	xxx
UNEXPECTED INTERRUPTS					xxx
NO INTERRUPT RECOGNITION					xxx
MEMORY PARITY ERROR					xxx
SYSTEM STARTUP/RECOVERY					xxx
WATCHDOG TIMER TRAP					xxx
SOFTWARE DETECTED FILE INCONSISTENCY					xxx
SOFTWARE DETECTED SYMBIONT INCONSISTENCY					xxx
ERRLOG FILE COPYING ERRORS					xxx
TOTAL	xxx	xxx	xxx	xxx	xxxx


ERR:SUM
INITIATE ERR:FIL GHOST
IS PRIVILEGE INSUFFICIENT ?
YES
NO
M:UC "ERR:SUM EXIT: INSUFFICIENT PRIVILEGE."
EXIT
M:UC "ERR:FIL RUNNING, PLEASE WAIT."
WAIT 6 SECONDS
1
OPEN SUMFILE IN INPUT MODE
ERR/ABN
READ THE MASTER SUMMARY AND THE LAST INCOMPLETE SUMMARY FROM SUMFILE
ERR/ABN
DOES SUMFILE EXIST?
NO
YES
M.UC "SUMMARY FILE DOES NOT EXIST"
EXIT
UPDATE THE MASTER SUMMARY FROM INCOMPLETE SUMMARY
FORMAT AND PRINT SUMMARY ON LO DEVICE
IS SUMFILE BUSY ?
YES
NO
M:UC "SUMMARY FILE BUSY, PLEASE WAIT."
M:UC "SUMMARY FILE READ ERROR"
WAIT 6 SECONDS
EXIT
1
2
P2


2
M:UC
"PURGE SUMMARY FILE?"
INPUT ANSWER FROM M:UC
IS ANSWER 'YES'?
NO
YES
GET CONFIRMATION
M:UC
"ARE YOU SURE?"
INPUT ANSWER FROM M:UC
IS ANSWER 'YES'?
NO
2
YES
CLOSE AND RELEASE SUMFILE
M:UC
"SUMMARY FILE PURGED"
EXIT
IS ANSWER 'NO'?
NO
2
YES
CLOSE AND SAVE SUMFILE
M:UC
"SUMMARY FILE SAVED"
EXIT

ID

Error Log Lister - ERR:LIST

PURPOSE

ERR:LIST provides a way for a remote user to examine the keyed file "ERRFILE" constructed by ERR:FIL. ERR:LIST examines ERRFILE for the period specified and produces a formatted listing of the error records and/or a summary of the errors for that period. The formatted listing is complete with English headings and all format conversions intended for use by field personnel.

USAGE

ERR:LIST may be called by a terminal user with a diagnostic privilege running in the :SYS account. When running as a batch job, the card format is the same as for terminal input. The usage of ERR:LIST is described in detail in the UTS System Management Reference Manual, 90 16 74.

INPUT

File "ERRFILE" in account ":SYS".
User format specifications.

OUTPUT

As specified by the user.

INTERACTION

Monitor services used are: M:OPEN, M:READ, M:WRITE, M:CLOSE, M:DEVICE, M:EXIT, M:TIME.

ERRORS

If "ERRFILE" does not exist, "ERRFILE.:SYS DOES NOT EXIST" is output to the LO device. If no records can be found following the date and time specified, "THERE ARE NO RECORDS FOR THIS PERIOD" is output to the LO device. All other errors use the default monitor error communications.

DESCRIPTION

ERR:LIST opens "ERRFILE" and attempts to read a record with a key corresponding to the starting date and time specified. If there is no such record, the next sequential record is read.

The record is formatted according to type and is output on the selected device. Sequential records are read and output until an end-of-file is reached. At that point, a summary of all the errors for the period is output if requested.

If only a summary of errors is requested, outputting of the formatted records is inhibited and only statistics are gathered. When an end-of-file is reached, the summary is output to the selected device.

ID

PFSR - Power fail-safe recovery

PURPOSE

In the event of a power failure, some system information is saved and all I/O is halted. When power returns, the system is restored to its condition at the time of the failure.

USAGE

The power-on interrupt cell (X'50') and the power-off interrupt cell (X'51') contain XPSDs to the two routines POWRON (BEGINON) and POWROFF (BEGINOFF).

INPUT

DCTSIZ and DCT1 are used to halt all I/O and locate the system RAD.

S:CUN is used in reloading the current user's map access. OPLBT3 and DCT5 for the OC and DCT5 for the system RAD are checked for busy.

OUTPUT

During power-on, counters 3 and 4 (cells X'55' and X'54', indirect) are set to 100.

DCT13 for the system RAD is set to indicate transmission error.

INTERACTION

During power on:

LMA (Section N; INITIAL) is called to load locks, maps, and access

T:XMMC (Section GA) is called to load current users map access

COCINIT (Section DC) is called to initialize COC

INTSIM and SERDEV (Section DA) are called to service the OC and the system RAD so that they are not noted as timed-out.

DESCRIPTION

During power off: all registers are pushed into the monitor's TSTACK, all I/O is halted (for devices in DCT1) and the computer WAITs.

During power on: the message "POWER FAIL-SAFE" is typed on the OC. The locks, map and access are restored for the current user. A write-direct is issued to arm and enable counters 3, 4 count pulse and zero, memory parity, I/O, and control panel.

Counters 3 and 4 are set to 100. Next the operator's console and the system RAD are assured to be not busy and not in danger of being timed out. This is accomplished by calling INTSIM and SERDEV within module IOQ. Finally COCINIT is called to initialize COC, the registers are pulled and control is returned to the instruction which was interrupted by the power-off.

INTRODUCTION

The purpose of this document is: (1) to assist programmers maintaining Delta, and (2) to serve as a reference document for programmers seeking detailed knowledge of particular aspects of Delta code.

The tools a programmer should have available for understanding Delta are: (1) The UTS/TS Reference Manual (90 09 07), (2) this document, (3) the listing of Delta which is organized to parallel Part II of this document, and (4) an assembly concordance of the listing produced by Meta-Symbol (a concordance is an index of symbol usage in a listing). It is assumed in this document that the reader is familiar with the DELTA reference manual - or an equivalent description of the DELTA language.

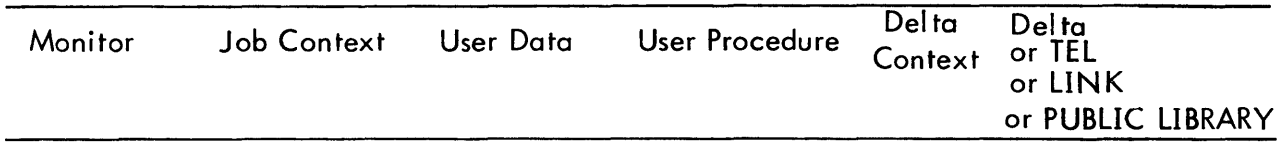
Part I of this document provides a general picture of how Delta fits into UTS and describes special features of the Delta-UTS interface. Part II, which parallels the listing in section organization, consists of prose description or flow charts (or both) of the individual routines of the Delta processor. The functional categories into which the routines are divided correspond to the section headings of Part II. Part III describes the features unique to the executive version of Delta and how it interfaces with the Monitor and the machine.

PART I

This Part explains how Delta fits into the UTS system and certain other general features of Delta-UTS operation.

Memory Map and Layout

Delta is a special shared processor in the UTS system. A shared processor is a reentrant program which can be used by any number of users without having to reload the processor for each user. In addition, Delta is one of several special shared processors which are assigned to a fixed, high virtual memory area, dedicated so that DELTA may be called at any time. The layout of virtual memory is:



As the UTS scheduler selects each user for execution, the hardware memory map is loaded with the proper correspondences between physical and virtual memory for the current user. The correspondences for the DELTA area (and library) remain the same for all users associated with DELTA. The code for DELTA is not aware of which user data (DELTA

context) the Monitor has set up for it via the map.

Access Protection

When the user's program is in execution, the access protection registers are set to 00 (unlimited access) for his data, 01 (read and execute) for his pure procedure and for Delta's data, and 11 (no access) for DELTA's procedure. When by any of several means, control passes from the user's program to Delta, the access for DELTA's procedure is changed to 01 and for DELTA's context (data) to 00. In addition, if Delta stores into the user's pure procedure area, the Monitor loads the access protection registers for user's procedure with 00 code and sets the bit which indicates that procedure has been altered (PPSWAP).

Symbol Tables

There are two types of symbol tables in Delta: (1) constant (internal to Delta), and (2) user associated or defined. The constant symbol tables are the special mnemonic table, SPECODES, the regular opcode mnemonics, OPCODES, and the special symbol table, SPECSYM. The mnemonic tables are constant and exist in DELTA's pure procedure area. The special symbols are variable in value but constant in symbol. They are included in the user's context page.

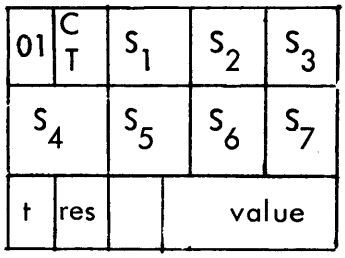
The user associated tables are the global and internal symbol tables. When the user creates a load module via LINK, the external references are gathered into a global symbol table and the internal symbols within each ROM are gathered into an internal symbol table. LINK determines the sizes of the largest internal symbol table and of the global symbol table. It then allocates virtual memory starting from DELTA's context page and moves down (higher to lower addresses) to determine the start address of the largest internal symbol table and of the global symbol table.

User Data	User Procedure	Global Symbols	Internal Symbols	Delta Context	Delta

The user can then load via Delta commands, the global symbol table and/or a single selected internal symbol table (specified by ROM name). The internal symbol table selected is always loaded at the start address of the largest internal symbol table in the load module. In addition to these symbols, the user can define symbols on-line via DELTA commands. These symbols are added to the top (low addresses) of the global symbol table. The address for the current top (low address) is in item SYMBEGIN.

The format of the symbol tables is illustrated in the following table:

Location Symbol - code = 01



where

CT is a six-bit field containing the character count of the original symbol.

S_i are the first seven (7) characters of the symbol. Symbols with fewer than seven characters are zero filled.

t is a five-bit field where the values are:

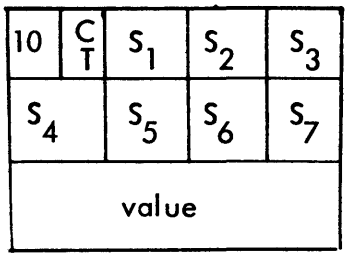
00000 - instruction
00001 - integer
00111 - EBCDIC text (also for unpacked decimal)
00010 - short floating point
00011 - long floating point
00110 - hexadecimal (also for packed decimal)
01001 - integer array
01010 - short floating point array
01011 - long floating complex array
01000 - logical array
10000 - undefined symbol

res is a three-bit field representing the internal resolution. The values are:

000 - byte
001 - halfwaord
010 - word
011 - doubleword

value Location symbols are always represented as a 19-bit byte resolution value.

Constants - Code = 10



where

CT and S_i have the same meaning as above.

value is the 32-bit value of the constant.

Delta-User Execution Control

Control comes to Delta initially through the break key entry point, T:DBRK (i.e., Delta has break key control). Delta has trap, exit, break and abort control by virtue of special case tests for Delta in the Monitor. Execution control is now in Delta. When the user wants to start execution of his program he specifies a start address with the ;G command. Delta then performs a trap return CAL (M:TRTN) with the start address in the PSD in the user's temp stack (DTSTACK). The Monitor trap return processor pulls the environment from DTSTACK and transfers control to the address in the PSD in the stack. Execution control is now in the user's program. If the user's program traps, aborts or exits, control returns to Delta. In addition, the user may have specified instruction, transfer or data breakpoints. For instruction and transfer breakpoints, XPSD instructions are planted in the user's program when the user requests execution of his program. The user's program operates in slave mode. Thus, when execution reaches a breakpoint, a privileged instruction trap results. Control goes to Delta's trap processor T:DTRAP which checks for instruction or transfer breakpoint. Data breakpoints are achieved by increasing the protection type on the data page containing the specified address from 00 (unlimited access) to 01 (read and execute). If a memory store into that page occurs, a memory protection violation trap ensues. If there is no breakpoint on the page, it is a genuine trap and Delta reports the violation and prompts (unless the user has trap control). If the page has a data breakpoint in it, Delta executes the trapping instruction and checks for a breakpoint on the cell being altered. If no breakpoint, control returns to the user's program. If there is, the action taken depends on the

breakpoint specifications.

If the user has requested trap control and the trap is not the result of one of Delta's execution (;X) or breakpoint commands, Delta will push the trap environment into the user's temp stack and transfer control to the users trap entry point as specified in JIT.

DCB's

Delta makes use of only two DCB's, M:UC and M:XX, both of which are present in the user's JIT. M:UC is used only for reading and writing the teletype. M:XX is used to send output to the printer and to read the global and internal symbol tables from file storage.

PART II

The discussion of individual routines presupposes a knowledge of the following definitions and descriptions of terms and command structure in the DELTA language.

Definitions

Constant	A decimal or hexadecimal integer, or EBCDIC value as described in the Functional Specification
Symbol	One to seven* alphanumeric characters and #, :, $, @, in which at least one character is alpha (except for % which is used only for constructing illegal opcodes, e.g., %01)
Term	A constant or a symbol
Arithmetic Operator	Plus (+) and minus (-)
Field delimiters	Comma (,) and space ( )
Punctuation commands	/, \ , Cr, lf, ↑ , tab, =, <, >, !, ., ',), (, *

* ROM names in the ;S command (symbol table load) may be up to 12 characters long.

Semicolon commands	A semicolon followed by one of the following letters: A, B, C, D, F, G, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, or by / or = (see Part III for ;E, ;V)
Field	A term or a set of terms connected by the arithmetic operators and terminated by a field delimiter or a punctuation or semicolon command. (In the special cases of fields 3 and 4, 'space' is equivalent to 'plus' with respect to the definition of 'field'. See below for discussion of 'space'.) The maximum number of fields in a DELTA command is four.
Field designator	Space ( ). This operator advances the field pointer to field 3. If the pointer is already at or beyond field 3, the 'space' is equivalent to 'plus'. (This algorithm results in a simple scan for symbolic machine instruction value with op, reg, address, and index falling into fields 1, 2, 3, and 4.
Expression	Consists of (a) terms, (b) arithmetic operators plus (+) and minus (-), and (c) field descriptors space ( ) and comma (,).

Command Structure

Each DELTA command has a unique interpretation of the input fields. Some commands such as equal (=) and ;X combine the fields into a single value expression. These commands merge the input fields via DREVALIT to form the single value. Other commands such as slash (/) and ;L expect a set of limits, and merge the fields into two values via DRGETLIM. The way this merge is performed results in field designator 'space' being equivalent to 'plus' with respect to value (see DRGETLIM). Still other commands such as ;B, ;Y, ;Z, attach individual meaning to each field position. For these routines, 'space' is equivalent to 'plus' only in fields 3 and 4.

1. INITIALIZATION

The initialization section consists of (1) command scan initialization, (2) definitions for Read, Write, Open, and Access CAL's, and (3) DEF and REF declarations.

Command scan initialization has four entry points. The first entry 'DRINIT', sends a Cr to the output buffer, dumps all waiting output, sets the message routine exit to initial entry point 'DRINITA', and branches to the routine that reads a message from the teletype, 'PMESS'. This entry is used when a fresh start on a new line is desired. The second entry, 'DRINITA', resets the switch that indicates a register is currently open (DROPENSW), and clears the flag register R8. This entry is used to start scanning a new command except when the previous command resulted in a cell being opened (/, ↑, NL, \, Tab). The third entry, 'DRINITB', resets the field counter R5, the element counter R10, the comma counter R11, the field presence flags in 'FLDREG', zeros the value accumulators for the four possible fields in an expression DREXPRS(+0-3), and sets the arithmetic operator to 'plus'. This entry sets the scanner up to parse a new expression for value or for command parameters. The fourth entry, 'DRINITC', initializes the symbol accumulator DRCHRBUF, the character counter R6, the input term type register R7, and resets the hex input mode flag. This entry sets the scanner up to examine a new term within an expression. It is used when plus, minus, space, or comma is encountered. Low entries flow into higher entries, i.e. branching to DRINITA will execute DRINITB and DRINITC also.

2. COMMAND PARSE (DRSCAN 1)

The 'scanner' referred to below consists of DRSCAN1, entered with input parameters in registers 5, 6, 7, 8, 10, and 11, where:

R5	points to the current field in the input text string.
R6	contains a negative index into the symbol buffer. Thus, the character count is the maximum allowable number of characters plus (R6).
R7	current input mode type. Initially, number type is assumed and then changed if a characteristic of another legal type is encountered.
R8	special flags are set here during the scan and evaluation. Meaning of bit if set: 18 = Indirect bit — 22 = Symbol definition mode 20 = Immediate instruction — 27 = Undefined symbol 21 = Special Shift instruction — 31 = Shift instruction
R10	count of the number of elements (terms) in the input expression.
R11	comma count.

DRSCAN1 proceeds as follows:

If the input character is a special character (EBCDIC mode, \ ), process it. If it is a letter or number, switch on input type R7 through the transfer vector, DRBUILD, to convert via one of the formatting routines (see Section 3). Check again for special characters that don't fit the general case (:, #, @, $, symbol define mode, ( ). If not one of these, it is a punctuation character. Switch on the low order six bits of the character through DRKEY1 and DRPOINT1 to enter a punctuation command processor.

3. FORMATTING ROUTINES

The formatting routines consist of both input and output routines. The input routines are entered by the scanner through a transfer vector indexed by input symbol type in R7. Integer, Hexadecimal, and EBCDIC input is accumulated in R3, while symbolic input is accumulated in the symbol buffer DRCHRBUF. All of these input routines return to DRSCAN1 to continue the scan after processing the current character.

The output formatting routines consist of Hex, EBCDIC, integer, and floating point converters and also the expression formatter DRPTEXP, the address formatter DRPRTADR, the op code and symbol output routines DRMNEMPT and DRSYMPT, and two message output routines DRPRTSML and DRPRTMSG. All these routines result in characters being stored in the output buffer, OUTBUF. They do not cause output to be sent to the teletype. In the discussion of output formatting routines that follow, "outputs" should be taken to mean "moves to the output buffer".

3.1 Input Formatting Routines

3.1.1 DRINPNUM Integer Input Conversion

This is a standard integer input conversion routine with one exception. The input characters are saved in the symbol accumulator, DRCHRBUF, in case the term is a symbol with leading numeric characters. If a non-numeric character is encountered, the input type flag in R7 is set to 'symbolic' and the scan proceeds.

3.1.2 DRINPSYM Symbolic Input Routine

This routine stores the characters of the input term in the symbol buffer, DRCHRBUF, for later evaluation.

3.1.3 DRINPHEX Hexadecimal Input Conversion

This is standard hexadecimal conversion routine.

3.1.4 DRINPEBD EBCDIC Input Routine

This routine accumulates up to four EBCDIC characters, with leading zeros, in the input value accumulator R3.

3.2 Output Formatting Routines

3.2.1 DRPRTHXA Hexadecimal Print Routine

This routine outputs a period (.) to indicate a hexadecimal value, suppresses leading zeros, and performs the conversion on the input value in R3.

3.2.2 DRPRINT Integer Print Routine

This routine checks the size of the value in R3, outputs a minus sign (-) if it is negative, and converts and outputs the absolute decimal value.

3.2.3 DRPRTEBD EBCDIC Print Routine

This routine outputs in EBCDIC mode, i.e., directly, the four bytes of the value input in R3.

3.2.4 DRPRTFL Floating Print Routine - Double Precision Entry

DRPRTFS Single Precision Entry

This is a standard floating point output converter.

3.2.5 DRPTEXP Expression Print Routine

If other than symbolic output mode is specified, DRPTEXP switches to the appropriate output formatting routine. If symbolic mode is specified, this routine outputs the value in R3 as an expression, either symbolic instruction or a less complex expression. If no bits are set in either the op code or R fields, the address field (and index register, if it is non-zero) is output in the current address output mode.

If bits are set in the op code or R fields, the symbolic op code (or %XX when XX is not a legal instruction) and integer R field are output and then the address and index fields are processed.

The mnemonic op code is obtained by using the op code value as an index into a full word table of mnemonics, OPCODES. If the op code value uniquely defines the mnemonic, the entry in OPCODES contains the mnemonic or "%XX" for nonexistent instructions. If other bits in the value must be checked to determine the mnemonic (e.g., S, SCS, SLS or LCI, LF, etc.), the entry in OPCODES contains the address of a routine to perform the special processing required by that instruction. For detail on address output, consult the next Section, 3.2.6. Further details of DRPTEXP are included in the flow chart.

3.2.6 DRPRTADR Address Print Routine

This routine outputs the value in R3 as a symbol, a symbol + hex offset (if the value is within a certain user setable offset from the nearest lower valued symbol), or as a hex constant if no symbol lies within the offset. There is an additional address output mode called csect mode which has the following characteristics:

(a) If the input value exactly matches that of a symbol, that symbol is output regardless of symbol type.
(b) In the search, the nearest symbol flagged as csect-type is retained.
(c) If no csect-type symbol is found, the value is output as a hex constant.
(d) If one is found, the hex offset limit is ignored and the output is "nearest csect-type symbol + any hex offset".

This feature allows the user to label control sections and get addresses output as a symbol plus relative line number which corresponds to the listing. Only location symbols are output by DRPRTADR if location-only mode is set (LOC$ONLY).

3.2.7 DRMNEMPT Mnemonic Op Code Output

Mnemonic op code in R6.

DRSYMPT Symbol Output

Symbol table pointer R9.

This routine has two entry points. The mnemonic entry simply outputs the symbol in R6. The symbol entry loads the symbol from the symbol table using the address in R9 as a pointer and then shifts off the leading code-byte count byte. The symbol is then output.

3.2.8 DRPRTSML Print 1 word of characters right to left

This routine outputs backward messages, i.e., right-most byte first. It results in a little saving of space in message storage.

3.2.9 DRPRTMSG Print TEXTC message

Byte address of message in R1.

DLMSG

Word address of message in R1.

This routine outputs TEXTC messages.

4. EVALUATION AND SYMBOL TABLE SEARCH ROUTINES

4.1 DREVAL Term evaluation and field value accumulation routine

This routine adds the value of the current term to the value buffer (DREXPRS) for the current field of the input expression. If the term is a symbol, this routine searches the mnemonic symbol tables and constant/location symbol tables for a matching symbol. If not found, the error exit is taken. If found but undefined, a value of zero is returned and the undefined flag is set. If found and defined, the value of the symbol is added to the value accumulator for the current field.

This routine is invoked when the arithmetic operators plus (+) and minus (-) and the field delimiters space ( ) and comma (,) are encountered. It is also called by the following punctuation and semi-commands: Cr, lf, ↑, \ , /, =, , Tab

;B, ;D, ;G, ;L, ;N, ;P, ;R, ;W, ;X, ;Y, ;Z

4.2 DREVALIT Accumulate input fields to form an instruction value

This routine evaluates the input fields as a machine language instruction. Field 1 is the op code, field 2 is the register field, field 3 is the address field, and field 4 is the index register field. A full word value for the input expression is returned in R3. Typically the value will be an instruction.

4.3 DRGETLIM Get input limits routine

This routine analyzes the input expression as a set of limits separated by a comma. If more than one comma occurs, the error exit is taken. The problem in this routine is that 'space' advances the field position to field 3, if the present field is 1 or 2, while 'plus' and 'minus' don't affect the field position in an expression. Thus, there are three basic possibilities:

( ) = space as a connector or operator
Li = term or terms connected by the arithmetic operators + or - .

	Limit 1		Limit 2
a)	L_1	,	L_2
	Field 1		Field 2
b)	L_1	,	L_2 ( ) L_3
	Field 1		Field 2 Field 3
c)	L_1 ( ) L_2	,	L_3
	Field 1 Field 3		Field 4

Thus, if field 2 is not present, field 3 is part of limit 1. Otherwise, field 3 is part of limit 2. The lower limit is returned in R2, the upper in R3. It should be noted that the item names and comments of the listing number the fields 0-3 rather than 1-4 as above.

4.4 SRCHCODE Search mnemonic table for input symbol

This routine is entered with the input symbol in R14. The table of special mnemonics, SPECODES, is searched first. This table contains (a) the immediate type mnemonics, (b) load conditions and floating mnemonics, (c) branch mnemonics, (d) byte string mnemonics, and (e) shift mnemonics. If the symbol is found in SPECODES, the op code value and appropriate special bits are correctly positioned in R2 and the flag appropriate to the special instruction type is set in R8. R0 is set non-zero to

indicate that a value for the symbol was found. If not found in SPECODES, the regular mnemonic table, OPCODES, is searched. OPCODES is ordered so that the index into the table is the value of the op code for each mnemonic. If the symbol is found, the index (i.e., op code) is correctly positioned in R2, the 'found' flag is set in R0 and the routine exits.

4.5 Symbol Table Searches (location and constant symbols)

SRCH$SYM Search special, global, and internal symbol tables

LOC$SRCH Search global and internal symbol tables

These entry points are drivers for the two routines that do the actual symbol table search; FINDSYM, which searches on symbols, and FINDVAL, which searches on value. There are three location and constant symbol tables present in DELTA: (a) the special symbol table (always present) which includes the following symbols: ;I, ;Q, $, ;M, ;C, ;F, ;1, ;2, ;3, ;4, (b) the global symbol table which includes symbols defined on-line in DELTA and which can be released, and (c) the internal symbol table which may or may not be present.

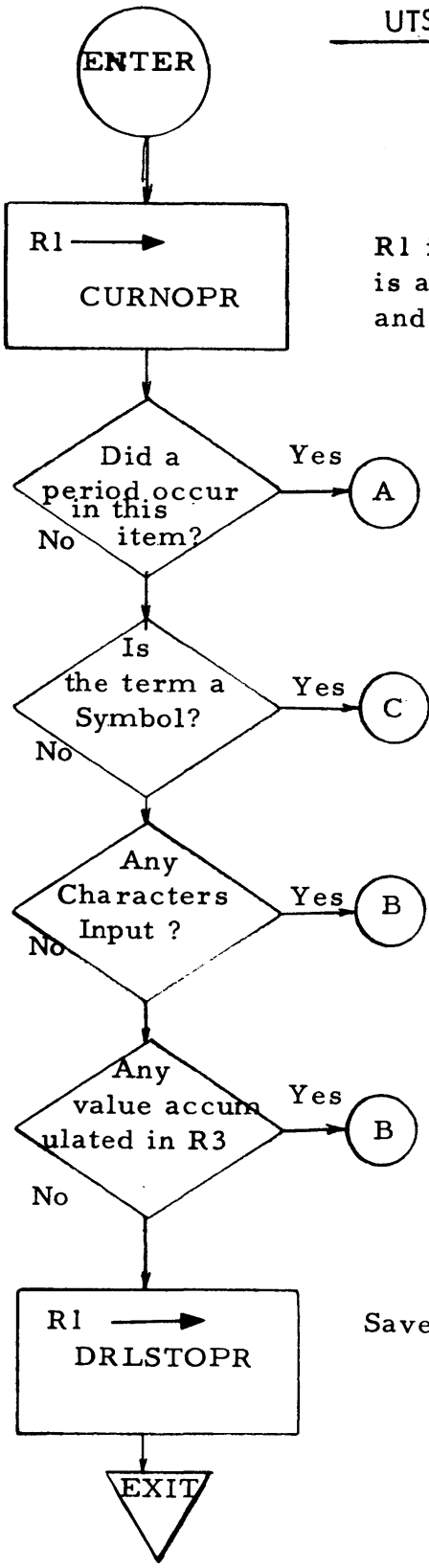
4.6 FINDSYM Search on symbol routine

This routine positions the input symbol in R14-15 to match the symbol table format and then searches the symbol table pointed to by the address in R9. The size of the current symbol table is given in R1. If found and undefined, the exit is taken through the drivers link, R4, with the undefined symbol code in R0. If found and the code indicates location symbol, the 17 low order bits of the value word are extracted to R2, the symbol type is extracted from the value word and saved in SYM$TYPE (in case symbol table control of output mode is requested) and exit is through driver's link, R4. If the symbol is a constant, the full value word is picked up in R2 and exit is through R4. If not found, exit is through R7 back to driver for searching next table or for exit.

4.7 FINDVAL Search on value routine

This routine is invoked when symbolic output of an address or constant value is desired. The routine is basically searching for the symbol whose value is closest to (and less than, or equal to), the input value. The input value is in item DRSRCHLM+1 and the nearest value, DRSRCHLM, is initially input value - hex offset -1. Symbol table values are checked within limits and if they lie between (DRSRCHLM) and (DRSRCHLM+1), DRSRCHLM is set to the new value, SYM$TYPE (see Section 5.14) is set to the symbol's type and the symbol table pointer is updated to this entry. If an exact match is found, the type is saved in SYM$TYPE

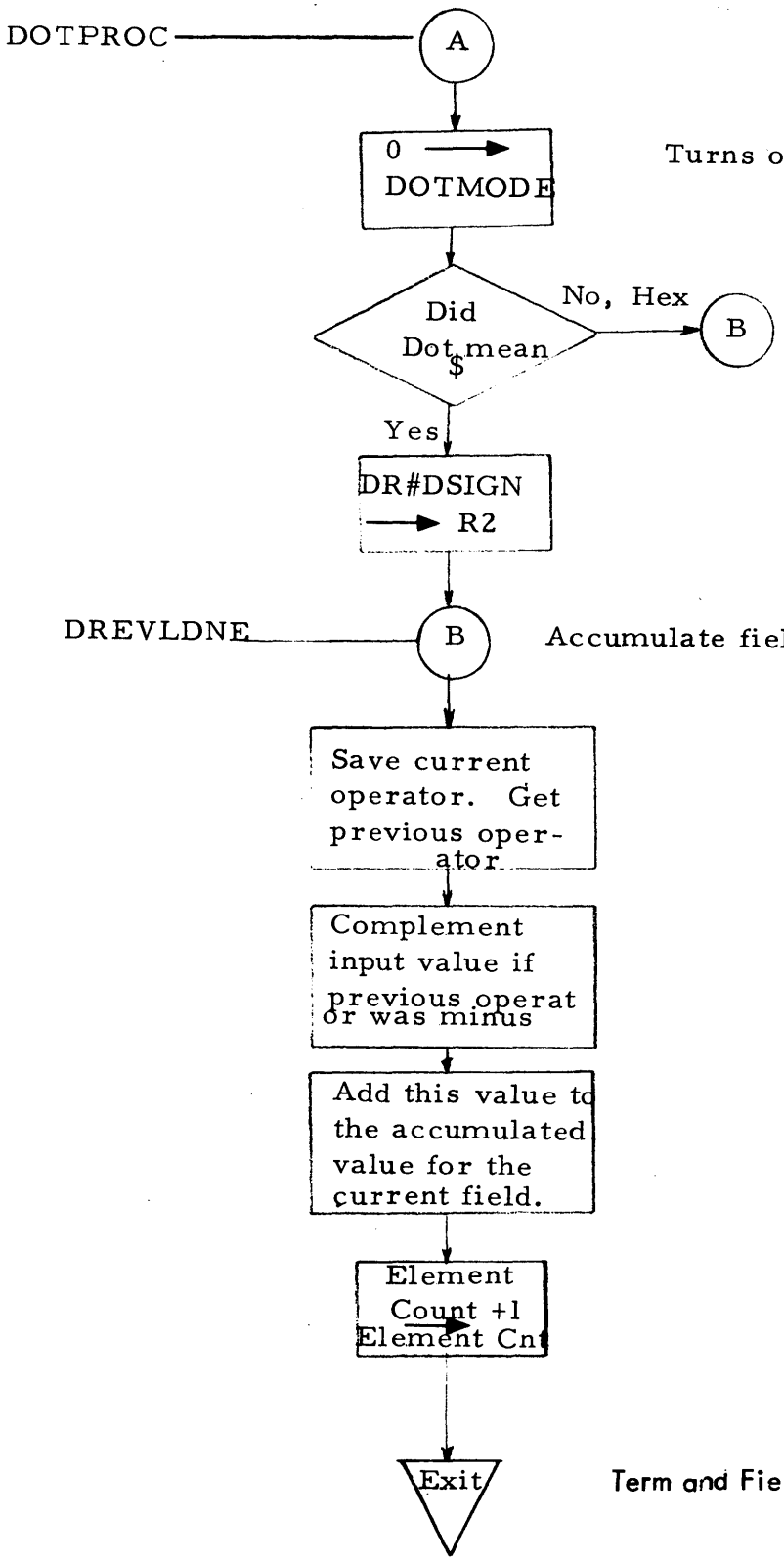
and the symbol is output. Undefined symbols are not checked for value, nor are constant type symbols if the location-only flag (LOC$ONLY) is set. In addition, if in csect mode (see Section 3.2.5 for a discussion of csect mode), only symbols flagged as csect types are considered for closest value, except that an exact match will print that symbol even if it is not csect type. In the event of an exact match, symbol type is saved in SYM$TYPE, the symbol is output and exit is through the driver's link, R4.

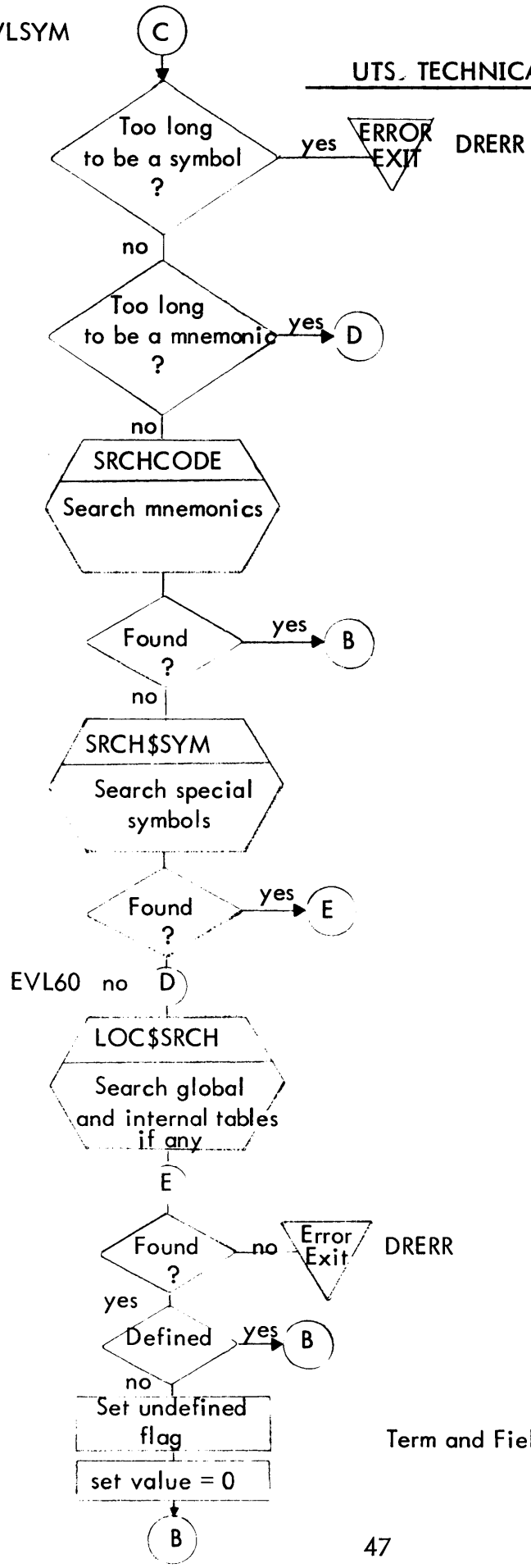


R1 is the index into DPROINT1 which is arranged so that 'space', 'plus', 'minus' and 'COMMA' come first.

Save operator for next pass through DREVAL

Term and Field Evaluation - 1


DOTPROC
A
0 → DOTMODE
Turns off Dotmode
Did Dot mean $
No, Hex
B
If any input between Dot and next operator or command, Dot meant Hex. If none, Dot meant $.
Yes
DR#DSIGN → R2
DREVLDNE
B
Accumulate field value
Save current operator. Get previous operator
Complement input value if previous operator was minus
Add this value to the accumulated value for the current field.
Element Count +1 Element Cnt
Exit
Term and Field Evaluation - 2



Term and Field Evaluation - 3

5. PUNCTUATION COMMANDS

5.1 DRASTISK Asterisk routine

This routine checks to see that the scan is currently in the address field (field 3) and that the asterisk is the first character input for this field. If no, error exit. If yes, set the asterisk flag in R8 and exit to DRSCAN1.

5.2 DRBSLASH Backslash routine

If an undefined symbol preceded the backslash, branch to error exit. Evaluation of the input expression is completed and the address field is extracted. The current location counter DR#DSIGN, is set to the input address, the backslash, get page, and open register mode flags are set, two spaces are output and the exit is to MESSAGE, with the scan for the next command starting at DRINTB.

The backslash command can be used to get a virtual page not already allocated to the user. Opening a cell in the virtual page via backslash and attempting to store into it (e.g., O Cr) causes a trap. If in get-page mode, the trap routine gets the page and continues execution of DELTA.

5.3 DRCOMMA Comma routine

This routine checks the current field number against the maximum (4). If currently in field 4, there are too many commas and the error exit is taken. Field number is checked rather than comma count because the space operator can advance the field count. The field and comma counts are advanced by one and the presence bit for the current field is set in FLDREG. The scan is continued at DRINITC.

5.4 DRCRTRN Carriage return routine

This routine simply branches to DRCLOSE and exits to MESSAGE.

5.5 DRDEFCHK Symbol definition

This routine, entered when ! or > are encountered, searches the symbol table to see if the input symbol exists. If it does and is defined, the new value word is stored in the table. If undefined, the link of unsatisfied references to this symbol is run, storing the new value. The code is set to location type and exit is to DRINITB leaving open the current location (in case of ! type symbol definition). If not found, the symbol and specified value and type are added to the top of the global symbol table by routine DRSYMAPD. The output mode to be associated with this symbol and csect type can be specified. If they are, they are included in the value word.

5.6 DREQUAL Equal routine

If there was no input, the last value typed (;Q) is output in the current or explicitly given output mode. If there was input, it is evaluated into a full word quantity by DREVALIT and output as specified. Exit is to DRINITB, leaving open the current register.

5.7 DREXCLAM Exclamation routine

This routine gets the current value of the location counter, DR#DSIGN, and saves it in the symbol definition value buffer, DRVALSAV. It then branches into routine DRDEFCHK to complete symbol definition processing.

5.8 DRLFEED Line feed routine

Line feed goes to DRCLOSE to store into and close the current location, increments current location by one, and continues in DRUPAROW.

5.9 DRLPAR (CHNGMODE) Output mode specification routine

Left parenthesis reads the next character in the input buffer via INPTCHAR and checks it against the possible mode characters. If the character is not a legal mode character, error exit. If legal, the index (which is the mode value) is saved in MODINPUT.

In addition, the pointer to the occurrence of 'Dot' in this term is advanced by two. This is a null action if 'Dot' did not occur. (See Section 5.12 for details.)

5.10 DRLSTHAN Less than routine (symbol definition)

This routine gets the full word input value from DREVALIT and saves it for use later when the 'greater than' character (>) is encountered. It then sets the symbol definition mode in R8 and exits to DRINITB (avoiding resetting R8 and leaving the current register open).

5.11 DRPERCNT Percent routine

This routine just checks to be sure the percent character is the first character of a term, resets the input symbol type to symbolic and exits to the symbolic input routine. Permits reference to non-existent instruction mnemonics (e.g., %01).

5.12 DRPERIOD Period or Dot routine

The period or dot can mean either 'hexadecimal input mode' or '$'. These meanings are distinguished by the number of characters intervening between the occurrence of the dot and the next punctuation character. If no characters intervene, dot means $ (as in './'). If one or more characters intervene, the evaluation routine will ignore the fact that dot occurred in the preceding term (i.e., dot meant 'hex mode'). In any case, Hex input type is set in R7 and the character position of the dot is saved in DOTCHAR.

5.13 DRRPAR Right parenthesis routine

This routine causes the instruction in the currently open location to be interpretively executed. Specifically, the routine checks for an open register. If no open cell, error exit. It then branches to the interpretive execution routine, TRACEXEC, with the flag STEPMODE set to true (=1).

5.14 DRSLASH Slash routine

If there were no input characters, this routine extracts the address field from the last value typed (;Q) and displays the contents of that address in the current slash output mode (determined by previous presence or absence of temporary specification and the default type). If input preceded the slash, the limits for slash display are obtained from DRGETLIM. If only one address was specified, the upper limit is zero and the routine exits after displaying the contents of the first location. If the location being displayed is one of the general registers, 0-15, the value is fetched from the user's register temp stack.

The current slash output mode is checked for 'symbol-table-control-of-output-format' mode. If in that mode, the item SYM$TYPE is checked for being set (greater than zero). If a symbol is associated with the current location, the type for that symbol is present in SYM$TYPE and will be used in DRPTEXP. If no symbol is associated with the current location, the default mode is relative output. The contents of the current location are output in the appropriate mode. If there are more locations to be displayed, the address is output on a new line in the current address mode. If that mode is symbolic, a flag is set so that only location symbols are considered in the symbol table search-output routine, FINDVAL. The contents are displayed as described above. The exit point sets a flag, DROPENSW, to show that the current location, DR#DSIGN, is open and exits indirect through SLSHEXIT.

5.15 DRSPACE Space routine

If the scan is not yet in field 3, set the current field pointer to 3. If in field

3 or 4, the routine doesn't affect the field pointer. Exit is to the code in DRCOMMA which sets the relevant field presence bit in the field register, FLDREG, and exits to DRINITC to scan the next term.

5.16 DRSQUOTE Single quote routine

If the scanner is in an initialized state, the input type in R7 is set to EBCDIC mode and exit is to DRSCAN1. If not initialized, set input type to a non-input type and exit to DRSCAN1. The assumption is that this is the terminating single quote.

5.17 DRTAB Tab routine

This routine first stores input (if any) and closes the currently open cell via DRCLOSE. It then gets the last value typed (;Q), the address field of which will be used in DRSLASHS as the current location to be opened and displayed.

5.18 DRUPAROW Up arrow routine

A branch to DRCLOSE stores any input and closes the current register. The current location is decremented by one and the new register is opened and displayed via DRSLASH, entered at DRSLASHS (no display if in backslash mode).

6. SEMICOLON COMMANDS

When the semicolon character is encountered, control is transferred to the semicolon dispatcher, DRSCOLON. The trailing character is fetched and used as an index into a byte table, DRKEY2, which contains an index into the halfword table, DRPOINT2, which, in turn, contains the relative address of the routine appropriate to the command. The absolute address is computed and control is transferred either to that routine or to DREVAL for argument evaluation, and thence, to the routine, depending on position in DRPOINT2.

There is a special class of semicolon commands which are really special symbols but can be set by typing a value before the symbol (except for ;Q). These commands are:

;C, ;F, ;I, ;M, ;Q, ;1, ;2, ;3, ;4

When one of them is encountered, DRPOINT2 points to the special semi routine, DRSPCHR, which either sets the value word in the appropriate entry of the special semi symbol table, SPECSYM= or stores the EBCDIC for this semi symbol into the symbol buffer, DRCHRBUF, and continues the scan (see flow chart).

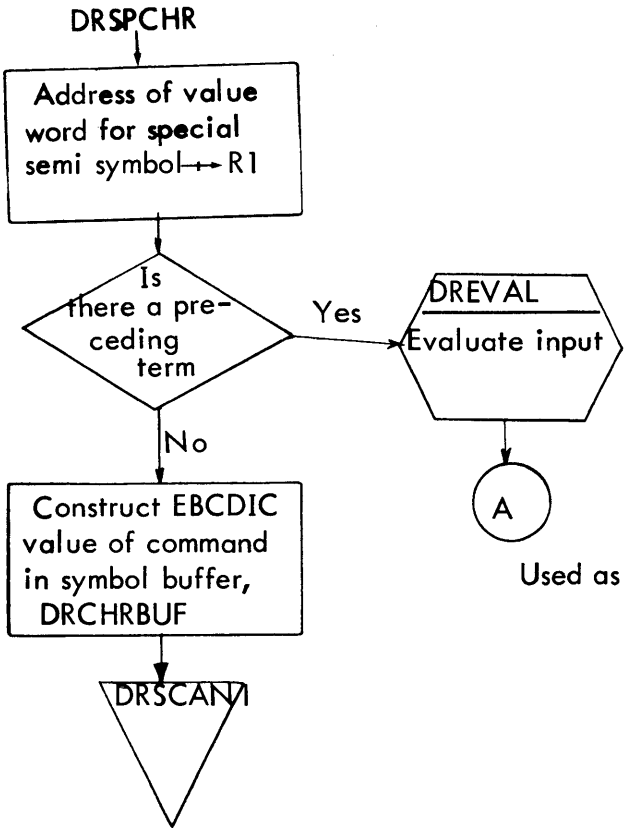
6.1 DRSEMIA Set absolute address print mode

Sets address print mode to absolute.

6.2 DRSEMIR Relative address print mode

If there is any input preceding ;R, it is evaluated and saved in item HEX$OFFSET. The trailing character is fetched. If it is not the letter 'K', the input buffer character pointer, CHARINDX, is backed up by one and the csect output mode flag, K$OUTMODE, is turned off (=0). If the trailing character is a 'K', the csect output mode flag is turned on (see Section 3.2.5 for a description of csect type output).

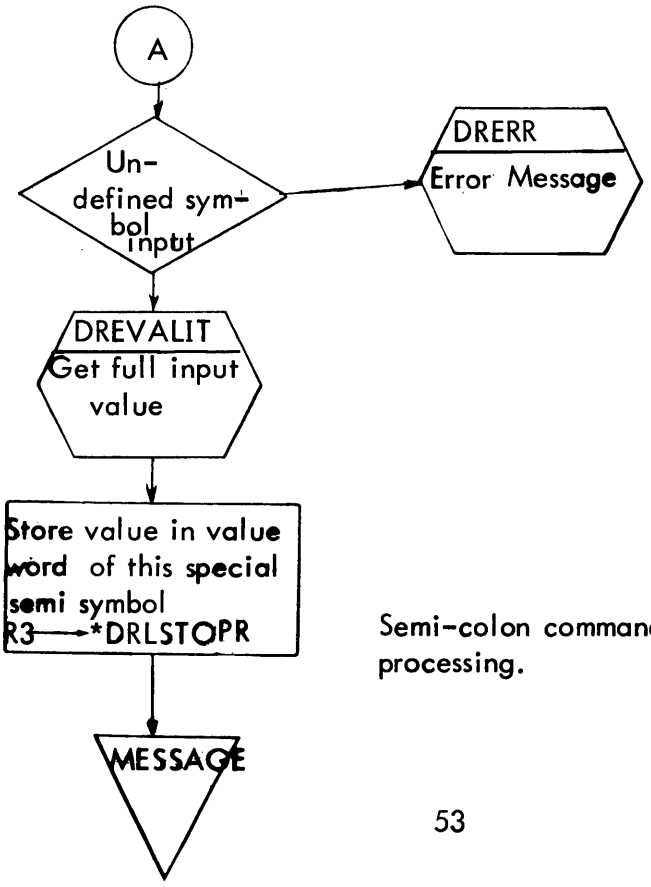
In all cases, the current slash output mode item DRSLSHMD and the address output mode item DRADRMOD are set to relative output format type (=0).



DREVAL saves R1 in DRLSTOPR. Below when the value is stored *DRLSTOPR it will go into the value word of the appropriate entry in the special symbol table, SPECSYM

Used as a command to set a given value

Used as a symbol in this case.



Semi-colon command switching and special symbol processing.

6.3 Data and Instruction Breakpoints

DRSEMIB Instruction breakpoint processor

DRSEMID Data breakpoint processor

Since the format of the commands for both breakpoint types is similar, as is the table structure maintaining information on the breakpoints, common processing routines are used which return to the calling semi command for any special processing required. These common routines appear in the Utility chapter of the listing but are discussed here.

In the constant data area are two tables of addresses, IBRKADRS and DBRKADRS. These tables contain the addresses of tables and items that are set up when breakpoint requests are made. By moving the relevant table into a similar table, BASEADRS, in the variable data area, the common routines can pick up fields from the command and store indirect through items in BASEADRS to set up the correct tables and items.

6.3.1 BRKSET

The structure of the breakpoint command is as follows:

Field 1	location or break number to be released
Field 2	breakpoint number
Field 3	I-breaks = display address D-breaks = conditional value
Field 4	D-breaks only - conditional mask

The specified table is moved into BASEADRS. The number of fields of input is then checked. If only one field was input, it could be an address with no break number specified or it could be a break release. If less than 9, control is transferred to the break release-dump driver, RELEASE. If the input was greater than 15, it is a set breakpoint request and the table is searched for an available entry (GETANUMB). If the input value was between 9 and 15, an error message is output.

If more than one field was input, field 2 - the breakpoint number field - is checked for being in the range 1-8. The current status of the specified break number is checked and if it is active the number of breakpoint entries is not incremented.

After verifying or obtaining the break number, field 1 is stored in the appropriate table of break locations. If field 3 was input, it is stored in the specified table (it will be either the display address for instruction breaks or the data compare value for data breaks). The trailing character is now checked. If it is a 'T', the trace bit for this entry is set; if not, the trace bit is reset and the input character pointer is backed up by one. Exit then goes through the return link in item CURNOPR (CURNOPR just happens to be available at the point of break processing).

DRSEMIB

On return from BRKSET, the proceed count is initialized to -1 and exit is to MESSAGE.

DRSEMID

On return from BRKSET, the mask and condition entries are initialized and a check is made to see if a data break condition was specified. If not, field 4 (the mask field) is ignored and exit is to MESSAGE. If specified, the appropriate conditional branch is stored in DBRKCOND and the mask field is checked. If no mask field, the nominal 32-bit mask is used; if specified, it is stored in DBRKMASK and exit is to MESSAGE.

6.3.2 RELEASE

If there was no input at all, ;B or ;D is a request to display the instruction or data breakpoint table and control goes to BRKDUMP. If the input value is zero, all active breaks are released. If the breaks being released are data breaks, the access protection type in the user's access image is reset to data type (00). If the input is 1-8, the specified entry is released and access reset.

6.3.3 BRKDUMP

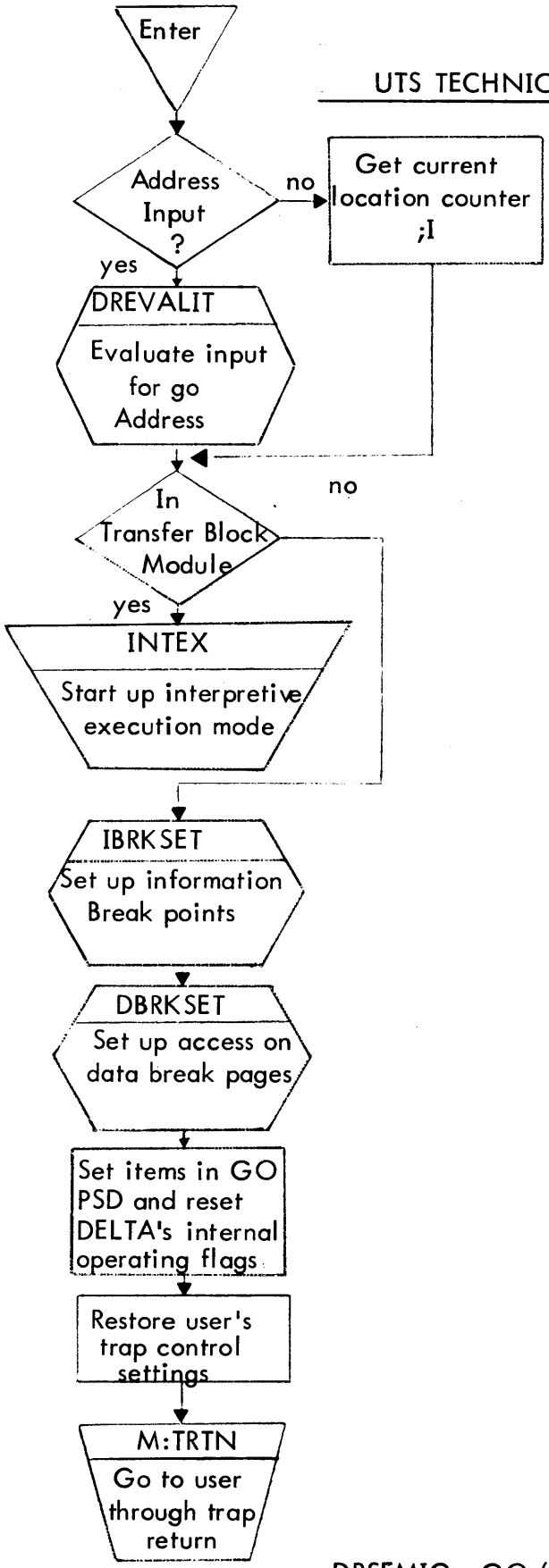
This routine outputs the break number, trace indicator and break address and then returns to link R14 plus one for completion of the line. The special processing at DRSEMIB or DRSEMID finishes the line and branches directly to TESTBD. At this point, the line is output and the test for completion is done. Exit is to MESSAGE.

6.4 DRSEMIG Start user execution

If there is no input, the start address is the current value of the location counter (;I). If the user is in interpretive execution mode (INTPMODE), control goes to INTEX. Otherwise, the start address is inserted in the PSD in the temp stack which will be loaded by the Monitor trap return processor. The conditions (;C) and floating controls (;F) are set in the PSD also and instruction and data breakpoints are set up (IBRKSET-DBRKSET). Delta's internal flags are turned off, the users trap control setting is restored and the trap return CAL is executed.

6.5 DRSEMIJ Switch output device

The output device switch is toggled. With regard to ;J, there are two classes of output from DELTA: (a) that which can go only to the teletype, and (b) that which can go to the teletype or printer depending on the setting of the switch.



DRSEMIG: GO (;G) Command Processing

6.6 DRSEMIK Delete a symbol or symbol table

If input precedes ;K, it is a symbol that exists in, and is to be "removed" from, the symbol table or the error exit is taken. If the symbol is undefined no action is taken and exit is normal. If defined, the "kill bit" is set in the entry in the symbol table to prevent use of the symbol for output and exit is to MESSAGE.

If no input, the trailing character is checked. If it is an 'I', the pages containing the internal symbol table are released, the internal symbol table size, NENTINTI, is set to zero, and the presence flag, INTLFLAG, is set to false (=0). If the letter 'G', the same thing happens for the global symbol table (which includes symbols defined on-line in DELTA). If the character is a blank or a carriage return, both internal and global symbol tables are released. Storage for the symbol tables is released to the Monitor. Any other character is an error.

6.7 DRSEMIL Set search limits

Sets the memory search limits ;1 and ;2.

6.8 DRSEMIO Hex dump on printer

The input dump limits are accumulated (DRGETLIM), if any. Any waiting output is sent to the specified device. The remainder of the command line after ;0 is positioned in the output buffer (or blanks if no characters after ;0), and a page eject-print a line is sent to the device. If no dump limits were specified (or lower limit =0), exit. Otherwise, space three lines down from the header, output the users registers, and output the specified area of memory suppressing output of duplicate lines.

6.9 DRSEMIP Proceed from a breakpoint

If the user is in transfer break mode, go to INTEX to continue execution. If in data break mode, continue at DRSGO in DRSEMIG. If in instruction break mode, input before ; P is a proceed count and is stored in IBRKPRCD. At this point, the instruction at the active breakpoint location is fetched and set up to be executed by routine BRKINTPT and the location in Delta's context where broken instructions are executed, IBRKEXEC, is placed in the PSD for user execution. Processing continues at DRSEMIGB.

6.10 DRSEMIS Load symbol table

The global and internal symbol tables exist as files on the RAD. The file name (the current load module name) and the password, if any, are in JIT. The key for the read of the files is the load module name with a trailing byte of X'09' for the global symbol table and ROM name plus trailing byte of X'0A' for the internal symbol table.

The virtual pages required for the specified symbol table are obtained as specified by JIT values (J:IST - internal, J:GST - global) the file opened and the read performed. The number of words read is obtained from the M:XX DCB and is added to the appropriate number of entry items, NENTSYM or NENTINTL.

6.11 DRSEMIT Set trace mode

The mode of the currently active breakpoint is set to 'trace'.

6.12 DRSEMIU Display undefined symbols

This routine examines the symbol tables via the driver SRCH$SYM (entry LOC$SRCH), and outputs the names of all the undefined symbols encountered.

6.13 Search Routines

DRSEMIW Word search routine

DRSEMIN Not word search routine

These routines search memory for a match (;W) or mismatch (;N) between the input value and memory using ;M as a mask. The input value can be any single expression. If, and only if, fields 1 and 2 and no others are specified, field 2 will be substituted in memory wherever field 1 is found, through the mask ;M by the utility routine STORWORD (STSWORD). When a match (or mismatch) is found, the location and contents are printed out. If substitution mode is in effect, printout occurs after the substitution has been made.

6.14 DRSEMIY Transfer break mode set up and down

The entry to ;Y sets up switches on the trailing character.

No relevant character or 'T'

If not 'T', 'R', 'S', or 'D', the input character pointer is backed up by one and the trace mode flag, TRACMODE, is reset. If 'T', TRACMODE is set. This ;Y entry is now going to set flags and start up interpretive transfer break execution.

All flags except TRACMODE are initialized to nominal values. If there is no input, interpretive execution is started at the current value of the location counter (;I). If only one field was input and the value is 0 (or <16), transfer break mode is turned off. If the value is greater than 15, execution begins at the input location. If fields 2 and 3 were specified, the values are stored in DO$DONT and TRACLOOP, respectively. If the options (fields 2 and 3) are specified, a value less than 16 for the location (field 1) is treated as null and execution proceeds from ;I. This allows setting of the options and continuing from last break without specifying an address.

Trailing 'R' Release

This routine releases entries in the special action table, TRACSPEC. Up to four addresses may be specified. If no input, release the entire table. Otherwise, each field specified contains an address which is to be deleted from the table. If an address specified does not appear in the table, error message and error exit. If it appears, zero the entry and check next input field.

Trailing 'S' Set

This routine sets entries in the special action table, TRACSPEC. If no input, error exit. TRACSPEC is then searched for a zero entry. If none, error message and error exit. Otherwise, store address in current field into the entry just found.

Trailing 'D' Display

This routine displays the addresses currently in the special action table, TRACSPEC.

6.15 DRSEMIZ Set core routine

Fields 1 and 2 are the lower and upper limits, respectively, of memory to be affected. Field 3 is optional but if not specified by the user, it will already contain the nominal value to be stored, zero. The value

in field 3 is stored in the specified range of memory by the utility routine, STORWORD.

7. SYSTEM CONTROL

Delta is entered as a result of one of the five following events:

1) Trap during user or DELTA execution.
2) User request (Monitor simulates a break entry).
3) BRK key event.
4) Error abort or exit CAL.
5) Monitor abort of the user.

These events are divided into three classes of DELTA entries:

a. Trap entry 1)
b. Break entry 2) and 3)
c. User exit 4) and 5)

The first three locations in DELTA's procedure are the corresponding entry points:

```
Relative location 0    B    T:DTRAP
Relative location 1    B    T:DBRK
Relative location 2    B    ABORTENT
```

The environment (PSD and registers) at the time of the event are pushed into DELTA's stack (DTSTACK). When the user program is in execution, DTSTACK is empty. Thus, if the user exits his program in any way, one environment will be present in DTSTACK. If the user then hits BRK or causes DELTA to trap (e.g. by attempting to display memory he doesn't have) a second environment will be pushed. In such a case, the second environment will be discarded as will any waiting output, the current operation in DELTA will be suspended and exit is to the initialization entry DRINIT. (A special two-environment case is discussed under T:DTRAP, section 7.2 below).

7.1 T:DBRK General break key entry

If this is the first time DELTA has been entered by this user, global and internal symbol table items are initialized, program start address is stored in ;I and $, and a greeting is output.

If DELTA was in control at the time of the break, a break flag, BREAKEY,

is set. DELTA's stack is adjusted and a fresh start is made after a break message.

If the user was in control, a message reports the instruction address at time time of the break and control goes to the console (DRINIT).

7.2 T:DTRAP Trap entry point

All traps enter here. After saving trap conditions, the internal DELTA-control flag, DELTCNTL, is examined. If DELTA was in control, DELTA's temp stack is decremented by 20 (the number of words pushed by the Monitor). If not a memory violation, output error message and restart. If it was a memory violation, check if in get page mode (GTPGMODE= -1). If in get page mode, get the page, reload DELTA's registers, and continue execution in DELTA at the trapping instruction. If not in get page mode or the page is not available output an error message and restart Delta. If Delta was not in control, a switch is executed on the trap location, 40 - 45.

TRAP40

Non-existent instruction and non-existent memory are error conditions. Control goes to the trap control routine, ERRCOND.

Privileged instruction trap may be an occurrence of an instruction, transfer or execute (;X) breakpoint. Control goes to the instruction break processor, IBRKRTRN.

Memory violation trap may be an occurrence of a data breakpoint. The location which caused the trap is analyzed to determine its effective address. If there are no data breakpoints or there isn't one on the page containing the address, it is a memory violation and control goes to ERRCOND.

If there is a break on the page, the instruction doing the store is executed. Check is made to see if there is a break on the altered location. If not, execution of the user continues. If there is, a check is made to see if it is a conditional break. If not, go to DATABRK to process the break-point. If it is conditional, the new value is compared as specified in the data break tables. Exit will be either to DRSGO (to to user) or to DATABRK.

7.2.1 ERRCOND (MEMVIOL) Trap control routine

This routine goes to routine TRAPCNTL to see if the user has control of the current trap. If he does, TRAPCNTL exits to the users trap control routine. If not, the appropriate error message is output and control goes to the console (DRINIT).

7.3 DATABRK

The Data breakpoint message is output. If this is a trace breakpoint, the message is dumped to the device and execution of the user continues. If not a trace break, go to routine RESTORE for some setup and leave control (in DELTA) and the console (DRINIT).

7.4 The other traps 41-45

An appropriate message is output and control goes to the console (DRINIT). If the user was in control at the time of the trap, go to RESTORE. If DELTA was in control, error exit.

7.5 ABORTENT

If entered when DELTA was in control, decrement DELTA's temp stack. If user was in control, go to RESTORE. In either case, put out the error message associated with the abort code and sub code and return control to the console (DRINIT).

7.6 IBRKRTRN Instruction breakpoint return

Instruction breaks (and transfer breaks) use XPSD instructions planted in the user's program. Since the user is operating in slave mode, attempting to execute these instructions causes a privileged instruction trap. For instruction breaks, the address field of the XPSD contains the breakpoint number. Hence, if the address causing the trap appears in the entry in the table of instruction break locations pointed to by the address of the XPSD, we have an active instruction breakpoint. If a proceed count was specified and has not yet been counted down, execution proceeds via IBRKCONT (DRSEMIP). Otherwise, the instruction break message and the display, if requested, is output. If it is a trace break, the output is sent to the specified device (Printer or Teletype) and execution continues in the user. If not a trace, routine RESTORE is invoked, various items set, and control goes to the console.

7.7 Interpretive Execution

Interpretive execution works as follows: From a given start location, search down the routine until a branch type instruction is encountered, plant a uniquely flagged return (XPSD) there and start normal execution at the start location. In the routine that gets return control, TRACRTRN, check if there is an instruction break on the location containing the branch. Continuation after instruction break processing is at INTEX where the branch is analyzed and interpretively executed, i.e., the users registers are appropriately affected, the location counters in DELTA are adjusted and TRACRTRN is entered with a flag indicating whether the branch occurred or not. If the branch falls through, go back to the search above and continue. If the branch branches, check the special action table, TRACSPEC, and the special action flag, DO$DONT, for what action to take. Also, check the BDR/BIR do-don't trace flag, TRACLOOP. If after all that, there is a break, check TRACMODE for whether to continue execution or halt in Delta, returning console control to the user via MESSAGE.

If there is an instruction break on the branch or on its effective address, that message will also be printed.

Step execution mode uses this same code except that a return to the user's console occurs after every instruction execution with display of the contents of the current instruction address.

7.7.1 INTEX

This routine starts interpretive execution. If the instruction at the current IA is a branch, exit to TRACEXEC. Otherwise, search memory for the next branch, save that instruction and replace it with an XPSD, with address field 11, then start normal execution at IA (entry DRSG3 in DRSEMIG).

7.7.2 TRACEXEC

This routine interprets the current instruction and executes it accordingly. If the instruction is not a branch or EXU, execute it in the execute area, EXCTRAC, in user's context. If it is an EXU, trace down to the object instruction and check for branch. If not, execute. In either case, if it is a branch, appropriately affect the users registers and enter TRACRTRN with flag indicating whether it branched or not.

7.7.3 TRACRTRN

Interpretive execution control return processor. This routine handles return from user execution as a result of interpretive execution. If the return was type 11 (see INTEX), i.e., to interpret and execute a branch instruction, check for an instruction break on this address and if none, exit to TRACEXEC. If there is an instruction break, the message is output and the setting of the instruction trace bit for this location determines whether execution proceeds.

If the return was type 9 (non-branching branch), check for instruction breaks as above but exit to INTEX if execution is to continue.

If the return was type 10 (branching branch), check the three output determiners - table TRACSPEC, and flags DO$DONT and TRACLOOP (see DELTA reference manual 901634A page 12) and if no output, check for an instruction break and exit as above. If output is called for, do it and proceed or not depending on the trace mode flag TRACMODE.

Step mode always outputs and halts and does not print instruction break messages.

8. UTILITY ROUTINES

The routines in this section are for the most part subroutines invoked by one or more of the routines in previous chapters.

8.1 BDMP

If a dump of the instruction break table is requested, the common dump routine, BRKDUMP, calls BDMP to output the display location for the current breakpoint. Exit is to TESTBD in BRKDUMP.

8.2 BLANKBUF

This routine stores blanks in the output buffer.

8.3 BRANCHK

This routine checks the instruction in the location pointed to by the address in R1 for being a branch type instruction. If it is, exit is

through the link, R7. If it is not, R1 is incremented by one and exit is to link + 1.

8.4 BREAK$MESSAGES

8.4.1 DBRKMSG Data Break entry

8.4.2 IBRKMSG Instruction Break entry

This routine outputs the data or instruction breakpoint message. It is invoked by DATABRK and IBRKRTRN.

8.5 BRKDUMP Dump Breakpoint table

See Section 6.3.3

8.6 BRKINTPT Instruction Break proceed setup

This routine fetches the instruction from the location of the currently active breakpoint. If the instruction is a BAL, it is changed to an unconditional branch to the effective address of the BAL, and the link register in the user's register stack is set to point to Break location plus one. In any case, the instruction is stored in the execute area, IBRKEXEC, in the context page. An unconditional branch to the break location plus one is stored at IBRKEXEC+1.

This routine is invoked by DRSEMIP, but only if the active instruction break flag is on ('active' simply means that control is in DELTA as a result of encountering an instruction break).

8.7 BRKSET

See section 6.3.1

8.8 CHKTABL

This routine checks the transfer breakpoint special action table for the presence of the address specified in R10. If found, exit through link R4. If not found, exit to link plus one, Invoked by TRACRTRN and SEMIYR and SEMIYS.

8.9 CONV

This routine converts the hexadecimal value in R7 to external EBCDIC format and stores the eight characters in the output buffer, OUTBUF. Invoked by LINESET (DRSEMIO). As each character is converted, it is compared with the character in the same position already in OUTBUF. If the characters are different, the non-duplicate indicator, SUM, is set.

8.10 DRCLOSE

This routine resets the temporary output mode flag, MODINPUT, and the step mode flag, STEPMODE, and checks if a location is open. If not, it exits. If open, it checks for current command input. If not, exit. The input is evaluated and stored in the currently open location by the utility routine, STORWORD. DRCLOSE is invoked by DRCRTRN, DRLFEED, DRUPAROW, DRTAB.

8.11 DRERR Error exit routine

This routine outputs a question mark and the current integer value of the input character pointer, CHARINDX. It exits to DRINIT and is invoked all over the place.

8.12 DRSYMAPD

This routine adds an entry to the top (low addressed memory) of the global symbol table. The entry to be added is in registers 12-14. A check is performed to see if the addition of these three words will cross a page boundary. If so, the new virtual page is obtained.

8.13 GET$HEADER

This routine blanks the output buffer, stores a page eject format character in it and moves the remaining characters in the command line (except the last charzcter) to the output buffer. Invoked by DRESMIO.

8.14 Virtual page control

8.14.1 FREE$PAGES

8.14.2 GET$PAGES

These two routines get or release pages from the page containing

the address in R6 to the page containing the address in R7. Invoked by DRSEMIK, DRSEMIS, DRSYMAPD, GETONE (TRAP40). If an error occurs, exit is to link + 1; normal exit is to link + 2.

8.15 DBRK$CHK

Depending on a mask in R7, this routine (1) checks for a data break in the page that contains the address in R6, or (2) checks for a data break at the address in R6. Invoked by TRAP40.

8.16 DUMPER

If a dump of the data break table is requested, the common dump routine, BRKDUMP, calls DUMPER to output the releation, the conditional value and the mask if they were specified. Exit is to TESTBD in BRKDUMP.

8.17 Instruction break set and restore

8.17.1 IBRKSET Set instruction breakpoints

8.17.2 IBRKRSTR Restore instruction breakpoints

Instruction breakpoints are set when going to the user (the XPSD's are planted and the instructions saved) and restored when control returns to DELTA (instructions are returned to where they belong). When setting instruction breakpoints, a check must be made to see if the location already contains a transfer breakpoint. If so, the instruction break is not planted, but the instruction break will be reported via the transfer break return routine TRACRTRN.

8.18 LINESET

This routine sets up a line of output for the printer dump routine, DRSEMIO.

8.19 Open DCB M:XX

8.19.1 OPEN$FILE

8.19.2 OPEN$PRINT

These entries open the M:XX DCB to a file using the current

load module name and password, or to the LO device, respectively. The DCB is closed by the routine which uses it.

8.20 RELEASE

See Section 6.3.2

8.21 Symbol table page control

8.21.1 RELEASE$GLOBAL

8.21.2 RELEASE$INTERNAL

These routines release the global and internal symbol table pages. If the boundary between the global and internal symbol tables is within a page and if only one of the tables is being released, the boundary page is not released.

8.22 RESTORE

This routine restores instruction breakpoints via IBRKRSTR and sets up the special symbols ;C, ;F, ;I, ;3, ;4 from the user's PSD.

8.23 SET$PRTCT

Executes a set memory protect CAL using the address in R12 and the protection type in R10.

8.24 SPACER

This routine is a tab simulator. It inserts blanks in the output buffer to the position specified in R0.

8.25 STORWORD

STSWORD

This routine stores the value in R0 into the location specified in R3 through a mask in R1. Entry at STORWORD sets a 32 bit mask in R1 or the caller inputs the mask in R1 and enters at STSWORD. If R3 is 15 or less, the value is stored in the appropriate register in the users temp stack (SAVREGS).

9. I/O ROUTINES

There are two classes of I/O routines: Internal which move a character at a time between DELTA routines and the input/output buffers, and External which move multi-character messages between DELTA's I/O buffers and the specified devices.

External Routines

9.1 PMESS

This routine sends out DELTA's prompt character, the bell, and executes a read for a maximum of 80 characters. When the Monitor returns with the completed message, the num ber of characters actually typed by the user is saved in NUMBCHAR. PMESS falls directly into the MESSAGE switching routine.

9.1.1 MESSAGE

This routine checks if the break key was hit by the user. If so, all output is ignored and it exits through DRINIT to PMESS. If no break, it checks for characters in the output buffer. If so, they are output to the device via DUMPBUF. It then checks for characters remaining to be scanned in the input buffer. If none, branch to PMESS for next command line. If there are characters, leading blanks are stripped off and the exit is indirect through MESSEXIT to one of the scan entry points.

9.2 DUMPBUF

This routine checks to see if the break key was hit recently by the user. If so, it clears the output buffer and the break key flag and branches to DRINIT. If no break, the items SEMIJ and MSG$TYPE determine which device the output is going to (LO or UC).

Internal Routines

9.3 TYPEOUT

This routine stores the low order byte of R0 in the next available position (OUTCHAR) of the output buffer. OUTCHAR is incremented by one and a check is made to see if the line has been filled. If so, it is printed via DUMPBUF. Otherwise, exit.

9.4 INPTCHAR

This routine returns the next character from the input buffer in R0. It then increments the pointer, CHARINDX, by one. If no characters remain in the buffer, error exit.

10. CONSTANT DATA

10.1 Op Code Tables

The use and structure of these tables is described in Section 3.2.5 and 4.4.

10.2 DRSPCSYM

A halfword table of addresses of the value words for the special semicolon symbols (SPECSYM). This table is used when setting new values for those symbols.

10.3 Punctuation and semicolon transfer tables

10.3.1 DRPOINT1

Halfword table of relative addresses of punctuation routines.

10.3.2 DRPOINT2

Halfword table of relative addresses of semicolon routines.

10.3.3 DRKEY1

Byte table of indexes into DRPOINT1.

10.3.4 DRKEY2

Byte table of indexes into DRPOINT2.

The use and organization of the tables for punctuation and semi commands are identical. When a punctuation character (or, after the semicolon, a valid character) is encountered, the high order bits are cleared and the remaining value is used as an index into DRKEY1 or 2, which value is then used to get the relative address of the appropriate routine from DRPOINT1 or 2.

In addition, the first N routines in both tables, have evaluation (DREVAL) automatically performed before they are entered. For DRPOINT1, N is determined by the position of item DRVAL1XX, and for DRPOINT2 by the position of DRSLIMIT.

10.4 CONDITIONS

Table of the EBCDIC form of legal conditionals that can be specified on Data breaks.

10.5 DBKCONDTN

Table of conditional branches parallel to CONDITION which are executed when a conditional data break is processed.

10.6 DBRKADRS

Table of relevant addresses for data breaks which is stored into table BASEADRS when the common break setup routines are operating.

10.7 DRBUILD

Transfer vector used for processing input terms in the scan. R7, the input type, is the index into this table.

10.8 DRPRTVEC

Transfer vector for output mode switching.

10.9 DRTYPCHAR

Byte table of legal output format modes.

10.10 IBRKADRS

Similar to DBRKADRS, Section 10.6, but for instruction breaks.

10.11 TRAPS

Transfer vector for trap control processing.

11. VARIABLE DATA (USER's CONTEXT)

This page of DELTA is flagged as 00 protection type, i.e. variable data, when DELTA is operating but is protected with an 01 (read and execute) protection type when the user has control. There is a page of this data for each user that has DELTA associated.

ID

Exec Delta

PURPOSE

The purpose of this document is to describe the usage of Exec Delta rather than its implementation.

OVERVIEW

Exec Delta, the executive version of Delta, is basically a stand-alone debugger used to debug the monitor. It is also used to process the patches to the monitor (root and overlays) at boot time. Exec Delta runs in the master mode, mapped or unmapped, and performs its own I/O. It is loaded with the monitor's ref/def stack (MONSTK) and is biased at X'EC00'. The Exec Delta command syntax is the same as the user Delta syntax and the command set is much the same.

USAGE

Exec Delta is called by GHOST1 to process monitor patches via

```
LI, 1       PACHSTRT
BAL, 11     *DLTBIAS, 1
```

For use as a debugger, Exec Delta is entered via the following steps:

1. Set the RUN switch on the control panel to IDLE
2. Display location X'4E' which contains the DELTA entry XPSD
3. Move the RUN switch to STEP
4. Move the RUN switch to RUN

DESCRIPTION

Exec Delta performs its own I/O. When it is entered it checks to see if the devices it uses are busy or have interrupts pending. If so, it waits for not busy, clears the interrupts, and remembers which devices had interrupts pending. When Delta exits, it performs a dummy I/O operation on each device which had interrupts cleared and waits until the interrupts are pending before exiting.

Patch Deck Processing

The patches to monitor are processed by a special routine in Delta. The format of the patches is as follows:

seg/address/instruction/comment

where:

seg = a hexidecimal number indicating the overlay to be patched (0 or omitted for the root)

address = a constant or a monitor label ± constant for patches to the root. For overlays the address must be a relative address in the overlay (labels may not be used).

instruction = a constant or an instruction with a constant or monitor label ± constant address field. For patches to overlays the address field must be absolute.

Some examples of monitor patches:

1. patch to root

   /IOSPIN+12/B IOSPIN+8/ COMMENT
   the specified branch instruction is stored in location IOSPIN plus decimal 12. The patch will be listed on the printer as follows:

   /IOSPIN+12/ B IOSPIN+. A B IOSPIN+8/ COMMENT
   where the underlined portion is the previous contents of IOSPIN+12

2. patches to an overlay

   1/. 3BE+. 50/. 3FFF/ CHANGE CONSTANT IN OVERLAY 1
   1/. 415/LW, 12 . 83BE+. 50, 3/

   At location . 40E(. 3BE+. 50) in overlay 1 the constant X'3FFF' is stored. At location . 415 in overlay 1 the instruction 'LW, 12 . 840E, 3' (X'32C6840E') is stored.

   Note that all overlays are biased at X'8000' and the address of the location to be changed in an overlay is relative to X'8000' but the instruction to be inserted is absolute, i.e. X'8000' + offset into overlay.

The labels used in monitor patches must be defined in the Exec Delta patch symbol table module (SYMTAB).

Patches to an overlay may not be intermixed with patches to another overlay. In fact, overlay patches must be in order by segment number in the patch deck but root patches may be placed anywhere. Thus the third patch in the following sequence would be illegal.

```
 1/. 841/LI, 1 0/
2/. 43/LS, 5 . 8541/
1/. 842/AND, 1 . 8035/
```

If Delta detects an error in a patch, it is printed on the operator's console typewriter. The operator then types logical not (¬) and new line. To ignore the patch and go on, the operator then types G. Any character other than G is interpreted as the first character of the correct patch. If a typing error is made while typing in the correct patch, the line rubout command is a question mark (?).

After the monitor root and segments have been patched and written on the RAD, the

system initializer, GHOST1, asks the operator if he wants to retain Exec Delta. If he responds yes, the pages Delta occupies are acquired from the monitor free page pool and thereafter Exec Delta is available for use. If he does not respond or responds no, the pages Delta occupies are left in the free page pool and the Exec Delta entry mechanism is disabled.

RESTRICTIONS

Exec Delta may not be entered while the computer is running in the slave mode since the XPSD will cause a privileged instruction trap.

ID

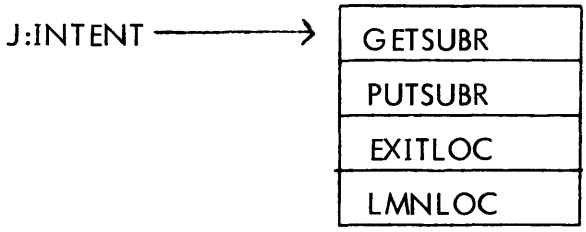
DELTA interface with processors

PURPOSE

To provide communication between user-DELTA and system processors (ANALZ and MONFIX). This interface allows a user to examine and modify a running monitor, a crashed monitor dump or a BOOTFILE using the language and tools of DELTA.

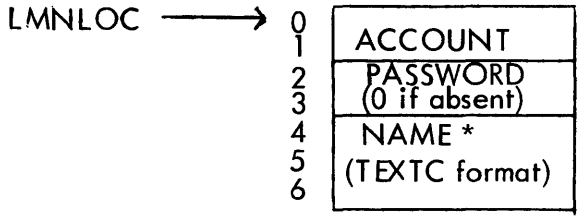
USAGE

The associate/disassociate CAL places the address of the vector taken from the first word of the FPT into J:INTENT and associates DELTA with the processor. The format of the vector is



GETSUBR is the address of the start of the GET subroutine.
PUTSUBR is the address of the start of the PUT subroutine.
EXITLOC is the address to ';G' to.
LMNLOC is the address of the LMN table.

The 7-word LMN table identifies the load module to be used by DELTA



*NAME is restricted to three words maximum.

A user-hit break causes control to be transformed to DELTA and a subsequent ';G' causes control to be returned to EXITLOC.

Each request to display a location causes DELTA to transfer control to the GET subroutine (GETSUBR) and request to store a value causes control to pass to the PUT subroutine (PUTSUBR).

GET Subroutine:

R3 = core address of word requested
BAL, R4 GETSUBR
(Value must be returned in R3. GETSUBR may use regs R0-R4, but must not clobber R5-R15).

PUT Subroutine

R0 = word to be stored.
R1 = mask by which to store. } as for 'STS, R0'
R3 = word address in which to store.
R2 = address of symbol text string corresponding to patch (for MONFIX).
BAL, R4 PUTSUBR
(PUTSUBR may use regs R0-R4, but must not clobber regs R5-R15).

EXAMPLE:

The user types 'LOC+. 1A/'. If 'LOC' has the value X'1000', GETSUBR is called with X'101A' in R3. DELTA then types out the word returned by GETSUBR in R3, and prompts for another input. The user now types 'SLS, 3 1 CR '. DELTA now calls PUTSUBR with:

R0 = X'25300001'
R1 = X'FFFFFFFF'
R3 = X'101A'
R2 = α
.
.
.
α TESTC 'LOC+. 1A/ SLS, 3 1'

Note that the symbolic string is in TEXTC form with no termination character.

UTS TECHNICAL MANUAL

ID

User Program Debugging

OVERVIEW

Batch debugging services include 1) program modification via MODIFY commands, 2) execution test and output via SNAP, SNAPC, IF, AND, OR and COUNT commands and user CALs, and 3) post mortem dumps via PMD, PMDE and PMDI commands.

These commands, which follow the RUN command, are read by CCI and each written as a record into the 'D' star file. A flag is set in J:ASSIGN (bit 14) to indicate the presence of PMD, PMDE and PMDI records. The count of the number of other debug records is set into the RUN table (consisting of RUN command information), which is left in high virtual memory for use by STEP and RUNNER.

If there is a 'D' star file, FETCH a routine of STEP, calls RUNNER before it brings the requested load module into memory. RUNNER reads the file and builds a Clobber and a Debug table in core. The Clobber table consists of 2 word entries containing 1) a location to modify or where a debug function is to take place, and 2) the modify value or the debug CAL. The Debug table consists of the parameter lists (FPTs) for the debug CALs. RUNNER rewrites, in a new format, the PMD, PMDE AND PMDI records and deletes the rest from the file. It also computes the start address if it is symbolic.

When RUNNER returns to FETCH, the load module is brought in and the debug CALs and modifies are set up using the Clobber table. As the user executes, the SNAP, COUNT, etc. CALs, inserted as a result of the above procedure and set up by the user, are honored by execution of routines in the modules SNAP and DUMP.

Upon termination of the program, if the flag in J:ASSIGN indicates PMDs or if the user was errored or aborted, STEP goes to the PMD routine of the DEBUG monitor overlay. PMD honors the PMD, PMDE and PMDI records and the TELUSER routine of DEBUG processes the error and abort codes. STEP then does the remainder of the job step shutdown.

UTS TECHNICAL MANUAL

ID

RUNNER

PURPOSE

The purpose of RUNNER is to build, just prior to execution time, two tables which become part of the user's pure procedure: 1) containing the locations to be by modified instructions or debug CALs, and 2) containing the Plists for the debug CALs. RUNNER also reformats the PMD, PMDE and PMDI records and deletes the rest of the records in the 'D' STAR file. If there is a symbolic start address specified in the Run table, RUNNER converts it to an absolute value.

USAGE

When the FETCH part of module STEP determines that RUNNER functions must be performed, it associates the special shared processor, RUNNER, and sets as the last entry in the stack the address of the beginning of the RUN table in the J:EUP (End of User's Program) page, i.e. X'1BE00'.

INPUT

The RUN table created by CCI contains in byte 1 the number of debug control commands and in byte 2, the number of modify control commands. Starting in word 10 of the RUN table, if present, is the symbolic start address in TEXTC format. DEBUG field (bit 14) in J:ASSIGN indicates the presence of PMD PMDE or PMDI records in the 'D' STAR file.

OUTPUT

The Clobber table contains two-word entries for each Debug or Modify request, specifying the location and the CAL or the modify word. If the top bit of the location word is set, it is a modify entry. All debug entries have a corresponding entry in another table, the Debug table, containing the Plist/FPT for the debug CAL.

DATA BASES

A Flag table is created in BUP (Beginning of User's Program) page and used in RUNNER to keep track of the flags used on the debug commands. Entries consist of a 2 word name followed by the address of word 7 of the Debug table entry which first referenced this flag.

SUBROUTINES

SEGSRCH finds the tree entry, in the tree pointed to by register 0, for a specified segment name pointed to by register 4. At SSH5 within the subroutine, it sets up in register 3 a pointer to the entry of the tree of the segment's back link. It then places the buffer address (the top of the buffer area input in register 5) and the REF/DEF size into an FPT to read in the segment's REF/DEF stack and sets its key into a key buffer. The pointer to the tree entry is output in register 14.

ERSUBR is used to output an error message from the error message file and return to the calling routine to continue processing. It calls upon ERRMSGE (an internal routine), to read the message from the error message file. If the high order bit of register 11, containing the return address, has been set by the calling routine, the name pointed to by location BA NAME is set into the buffer and the message is output.

PASSNAME increments register 5, which is pointing to a TEXTC name, to point byeond the name.

LOCNAM returns in register 6 the DEF value from the REF/DEF stack of the name, i.e., symbol, pointed to in register 5.

FROMTO returns in registers 8 and 9 the values from the REF/DEF stack of the "from" location and the "to" location of the debug commands.

ENTCLT enters the double word contained in registers 8 and 9 into the Clobber table.

ADFLG tries to find the flag name specified in registers 10 and 11 in the Flag table. If it is found, the address contained in the following word is placed in word 7 of the FPT in the Debug table. If the flag is not found, the name is entered into the table and the address of word 7 of the FPT currently being set up is set into the word following the name in the Flag table and a zero is set into word 7 of the FPT.

INITDB sets up words 0, 5, 6 and 7 of the FPT in the Debug table.

All of the following subroutines process some type of debug record from the 'D' STAR file and make the appropriate Flag, Clobber, and Debug table entries. Input is:

register 5 = beginning address of debug record
register 12 = address of next available entry in Debug table
register 13 = address of next available entry in Clobber table
register 14 = address of the segment's name in tree
register 15 = address of next entry in the Flag table

All of the routines check for invalid names and addresses and call on the error message routine to output appropriate error messages. Most of the previous subroutines are used to serve the following routines:

MOD	processes a Modify record.
PMD	processes a PMD, PMDE or PMDI record and is an exception. It does not create entries in the tables but creates a differently formatted PMD/PMDI record. It does not create entries in the tables but creates a differently formatted PMD/PMDI record and rewrites it to the 'D' STAR file.
SNAP	processes a SNAP or SNAPC record.
IAO	processes and IF, AND or OR record.

ERRORS

Following is a list of the RUNNER errors found in the error message file. The key is followed by the name in parentheses used in RUNNER to reference the error, followed by a description, followed by the name of the routine where error occurs.

040358 (ABB)	Name of locations where debug function to be done is not found in REF/DEF stack (INITDB)
040359 (ABC)	Name on IF, AND, or OR not found in REF/DEF stack (IAO)
04035A (ABD)	Name on SNAP not found in REF/DEF stack (SNAP)
04035B (ABE)	Name on PMD not found in REF/DEF stack (PMD)
04035C (ABF)	Name on MODIFY not found in REF/DEF stack (MOD)
04035D (AB5)	Debug flags overflowing space allowed (ADFLG)
04035E	Exceeding space allocated for Clobber table (EnTCLT)
04035F	Debug record from Debug file has an invalid byte 0 (main routine)
040360 (LMAB)	I/O error reading load module head or tree (main routine)
040361 (SGAB, (SGER)	I/O error reading the load module's REF/DEF stack (main routine)
040362 (DBER)	I/O error reading the Debug file (main routine)
040363 (ABG)	Bad symbolic name for start address (main routine)

040364	Location to Modify is not within limits (MOD)
040365 (ABH)	Start address not within legal program limits (main routine)
040366	Invalid tree size in load module (SEGSRCH)
040367 (ABI)	Can't get page following user's pure procedure for Debug and Clobber tables (main routine)
040368	Location to dump on PMD is not within legal program limits (PMD)
040369	Location where debug function to be performed is not within legal program limits (INITDB)
04036A	Location (FROM or TO) on SNAP not within legal program limits (SNAP)
04036B	Logical error in size of debug table when trying to move it to user's core (main routine)
04036C	PMD's or debugs were attempted on a LINKed LMN.

RESTRICTIONS

RUNNER is and must be a special shared processor with special JIT access.

DESCRIPTION

Since FETCH has set up M:XX for reading the load module, RUNNER saves and, before it exits, restores the error and abnormal addresses in M:XX. It moves the RUN table from the EUP page into its own data page and releases that page. It reads the LM (Load Module) head into the data page, and using that information, sets up the FPT to read in the tree at the top of the EUP page. It then computes space needed for the Debug and Clobber tables from bytes 1 and 2 of the RUN table and maximum REF/DEF stack area needed from the head. It then gets the necessary Common pages for these and reads in the tree. If there are no debug requests to process, it processes the symbolic start.

If there are debug requests, it gets virtual page, BUP, for the Flag table and opens the 'D' STAR file. RUNNER goes through a loop to process each debug record. The record is read and deleted. Its KEY consists of the name of the segment this debug applies to, plus an N indicating whether this is the 0 or the N record in the file for this segment.

The first time a segment is referenced, its entire REF/DEF stack is read in and the address of the next available entry of the Clobber table is set into word 10 of the segment's tree entry. Byte 0 of the Debug record indicates the type of debug command and the appropriate subroutine is called to process the record and set up the tables.

After all debug records are processed, the file is closed. If there is a symbolic start address, the REF/DEF stack for the root segment is read and the start address determined by adding the value of the symbolic location (determined by LOCNAM) to the displacement. This value is set into the head.

The Flag table page is released since it is used only for internal processing as are the REF/DEF stack pages. The header pure procedure doubleword size is updated to reflect the addition to it of the Clobber and Debug tables and the virtual pages for them are obtained. The Clobber table is moved and the CAL addresses relocated from the table setup area to where the Debug table will reside. It is then moved and word 0 (for chained Plists) and word 7 (for flags) relocated. The last word (word 10) of each entry of the tree is moved to the top of RUNNER's data page and they are relocated to reflect where the Clobber table is now rather than where it was built. The rest of the common pages are released. If there were any errors, RUNNER does an abort exit; otherwise, it exits normally.

STEP intercepts RUNNER's normal exit and drives directly into FETCH which reads in the load module and moves the word 11's into the tree. If there is a Clobber table, it is used to set up the Debug CALs and Modify's for the root. SEGLD sets them up for the overlays.

UTS TECHNICAL MANUAL

ID

SNAP

PURPOSE

SNAP consists of a number of routines that process the SNAP, SNAPC, IF, AND, OR and COUNT debug CALs.

OVERVIEW

Each of the debug CAL routines performs its function of either dumping portions of core or making tests capable of altering the setting of a flag for a conditional snap. All of the routines exit to DEBUGX which checks word 0 of the FPT for the link address to another debug FPT. If one is specified, the proper routine is executed and DEBUGX is re-entered. This process continues to the end of the chain. If the chain is circular or contains more than 255 links, the job will be aborted. When the end of the chain is found, the PSD in TSTACK is altered so that the instructions in words 5 and 6 of the FPT will be executed. (Word 6 is a branch instruction to the point immediately following the CAL.) Therefore, once normal trap exiting functions have been performed by TRAPEXIT (of ENTRY), control will eventually be returned to the user at the point directly after the CAL instruction.

USAGE

For all routines:
register 5 = address of JIT
register 7 = address of word 1 of FPT

ERROR

00B002 Is key to message in error message file when flag address is invalid on conditional debug command

ID

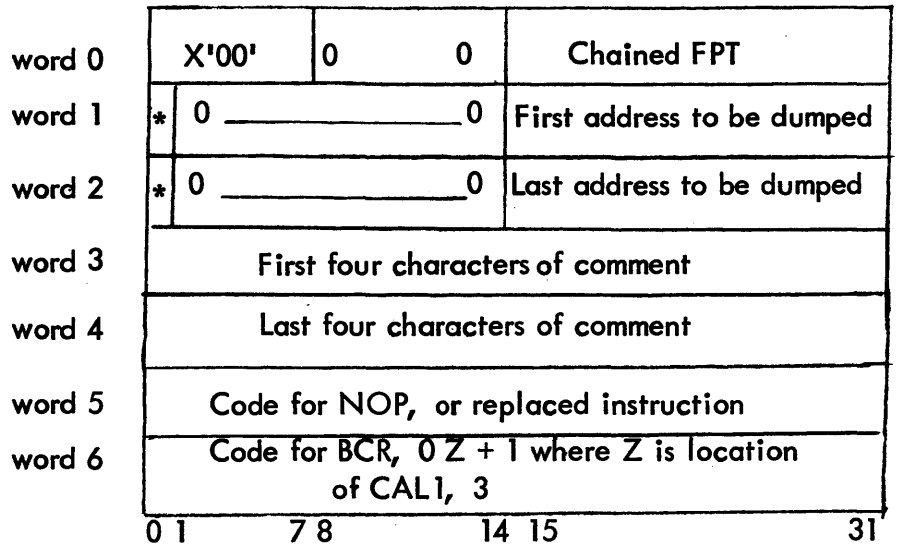
MSNAP - process !SNAP request

PURPOSE

To take an unconditional 'snapshot' of the memory locations specified by the user.

INPUT

FPT FOR M:SNAP



where

chained FPT — is the address of an FPT for a CAL1, 3 that is to be executed immediately following the current CAL1, 3. This address is used for a !SNAP control command with multiple dump locations ("from's" and "to's"). A zero address indicates that there is no chained FPT.

word 5 is the NOP instruction if the CAL1, 3 was coded by the users. Otherwise, if the CAL1, 3 is to be executed as the result of a debug control command (i.e. !SNAP), this ts the instruction from the user's program that was replaced by the CAL1, 3.

word 6 is the branch instruction to the location immediately following the CAL1, 3 instruction.

ID

MSNAPC - process !SNAPC request (conditional SNAP)

PURPOSE

To take a snapshot of memory if the flag specified by the SNAPC FPT is set (l).

INPUT

FPT same as SNAP except:

word 0, byte 0 = X'01' and
word 7 , , address of flag (1, 15, 16)

DESCRIPTION

The MSNAPC CAL checks the flag specified by the FPT. If the flag is set (flag = 1), the routine branches to MSNAP so that a dump can take place. Otherwise, it branches to DEBUGX which checks for chaining to any other DEBUG FPT. MSNAP then calls PRINT which writes on the diagnostic output device (M:DO) the comment specified in the FPT. The routine REGPRNT is then called which writes on the DØ device, l) the PSD, and 2) the registers. MSNAP verifies that the core to be dumped is in the proper range and, if it is not, aborts the job at ABORT1 of STEP. Otherwise, it enters the subroutine DUMPW, which dumps out on the M:DO device the core specified by the FPT. Finally, MSNAP exists to DEBUGX.

ID

MIF - process IIF request

PURPOSE

To make an FPT - designated test at a specified location and set the flag bit associated with the conditional snapshot if the test condition is true, and reset the flag if the test is false.

INPUT

IF FPT

word 0	X'02'	0 0	Chained FPT
word 1	* Instruction to load L1 into register 0		
word 2	* Instruction to load L2 into register 0		
word 3	Instruction to branch if specified relation (r) is true		
word 4	0		0
word 5	Code for NOP or replaced instruction		
word 6	Code for BCR, 0 Z+1 where Z is location of CAL1, 3		
word 7	*	0 0	Address of FLAG

where

chained FPT — is the address of another DEBUG FPT that is to be executed immediately following the current CAL1, 3. If address is zero, no other FPT is chained.

word 1 — is the instruction (LW, LH, LB, LD) to load the first comparand (L1) into register 0.

word 2 — is the instruction (LW, LH, LB, LD) to load the second comparand (L2) into register 0.

word 3 is the instruction to branch if the specified relation (r) between L1 and L2 is true. See below.

r	Instruction
GT	BCS, 1 0 0
LT	BCS, 2 0
EQ	BCS, 3 0
GE	BCR, 1 0
NF	BCS, 3 0

word 5 is the NOP instruction if the CAL1, 3 was coded by the user. Otherwise if the CAL1, 3 is to be executed as the result of a debug control command (i.e., !IF card), this is the instruction from the user's program that was replaced by the CAL1, 3.

word 6 is the branch instruction to the location immediately following the CAL1, 3 instruction

word 7 is the address of the flag associated with the conditional snap for the M:IF.

DESCRIPTION

The MIF routine obtains the two comparands, L1 and L2 (purpose of subroutine GETDTA). It then gets the branch instruction from word 3 of the FPT and fills its address portion with the address of SETF; L1 and L2 are compared and the conditional branching instruction, just updated with ETF in its address portion, is immediately executed. Therefore if the relation (r) is true, the MIF routine branches to SETF, where a switch is set to indicate that the flag is to be set. If r is not true, the routine falls through the branching instruction and sets a switch to indicate that the flag is to be reset.

Once the flag setting or resetting switch has been set, the flag itself is checked. If it is not within the legal limits of memory, the job is aborted at ABORT1 of STEP. Otherwise, the flag is set or reset and MIF exits to DEBUGX.

ID

MAND - process !AND request

PURPOSE

To make a specified test at a designated location if the flag bit is set. If the test condition is found to be true, the flag bit remains set; otherwise, the flag bit is reset.

INPUT

Same as IF FPT except word 0, byte 0 = X'03'

DESCRIPTION

The MAND routine checks the flag specified by word 7 of the FPT. If it is set, MAND branches to MIF. If the flag is not set, MAND branches to DEBUGX.

ID

MOR - process !OR request

PURPOSE

To make a specified test at a designated location provided the flag bit is not set (i.e. flag = 0). If the test condition is true, the flag bit is set; otherwise, the flag bit remains reset.

DESCRIPTION

The MOR routine checks the setting of the flag specified by word 7 of the FPT. If the flag is reset (i.e., 0), MOR branches to MIF to make the specified test. Otherwise, MOR branches to DEBUGX.

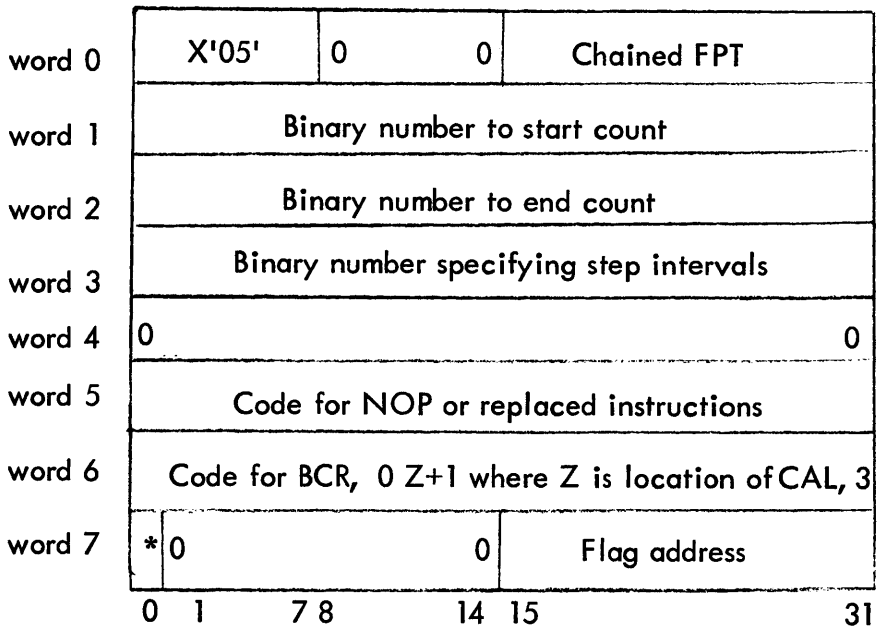
ID

MCOUNT - process !COUNT request

PURPOSE

To specify an iteration range (and steps within that range) in which a designated flag will be set. The flag will be set only if (1) the count is within the range specified by the start and end count parameters in the FPT and (2) the quotient (count-start)/ step is a non-zero integer.

COUNT FPT



where

Chained FPT — is the address of another DEBUG FPT that is to be executed immediately following the current CAL1, 3. If the address is zero, no other FPT is chained.

word 1 — specifies the starting count at which the testing of the count is to begin.

word 2 — specifies the ending count at which the incrementing of the count is to cease.

word 3 — specifies the count increment that determines the intervals (within the range designated by 'start' and 'end') at which the conditional snap flag will be set. (Flag will be set if and only if (1) the count is within the range specified by start and end, and (2) the quotient of (count-start)/step is a nonzero integer.)

word 4 — is the cell used by the Monitor for the count which initially must be zero.

word 5 — is the code for a NOP is the CAL1, 3 was included within the user's program. If the CAL1, 3 is to be executed as the result of a debug control command (i.e., a !COUNT card), it is the instruction from the user's program that was replaced by the CAL1, 3.

word 6 — is the branch instruction to the location immediately following the CAL1, 3 instruction.

word 7 — is the address of the flag associated with the conditional snap.

DESCRIPTION

Word 4 of the FPT becomes a counter which is incremented by 1 each time the FPT is used. If the count is less than the starting count or greater than the ending count, or if the quotient of (count-start)/step is not a non-zero integer, an attempt is made to reset the flag specified by word 7 of the FPT. However, if the FPT is chained to another DEBUG FPT, two checks are made: (1) is the chained FPT in core? and (2) is the FPT code (byte 0 of word 0) of the chained FPT legal? If either check is negative, the job is aborted. Otherwise, DEBUGX branches to the routine that will process the new chained FPT: MSNAP, MSNAPC, MIF, MAND, MOR, or MCOUNT. Chained FPTs are built by RUNNER when multiple dump locations have been specified on a !SNAP or !SNAPC card.

UTS TECHNICAL MANUAL

ID

PMD

PURPOSE

PMD provides dumps of the user's memory following user execution. It also calls upon TELLUSR to output error and abort messages and their associated debug information.

USAGE

STEP calls PMD with the user's exit environment in the stack.

DESCRIPTION

The run status (J:RNST) in JIT is interrogated. If a user was running and this is an abnormal termination, TELLUSR is called to list appropriate error or abort messages and dump related error information.

If PMDs are requested as indicated by field DEBUG (bit 14) in J:ASSIGN, and an M:DO DCB is found, a blocking buffer is obtained unless one remains from TELLUSR. An extra stack area is set up in the blocking buffer. The M:XX DCB is opened to the 'D' STAR file containing any PMD records defining the PMDs to accomplish. If the user exited normally, PMDIs are honored; otherwise, PMDEs PMDs and PMDIs are honored. The records are read into the buffer one at a time and the from/to pairs are pushed into a special stack in the blocking buffer. DUMPER is then called to dump the PSD, Registers and requested memory.

UTS TECHNICAL MANUAL

ID

TELLUSR

PURPOSE

TELLUSR is the routine in the DEBUG overlay segment which prints all monitor error messages to batch users who encounter an illegal trap, who are errored or aborted by the operator, or who error or abort themselves.

TELLUSR also prints I/O error messages.

TELLUSR prints the message, the location in the user program which generated it.

USAGE

TELLUSR is accessed via a OVERLAY DEBUGSEG, 7 from the exit logic in the monitor routine STEP. This transfers control to the routine PMD which uses TELLUSR as a subroutine to print the error message.

INPUT

CELLS
All table information used by TELLUSR is from the user's JIT. Specifically, TELLUSR accesses the following locations: ERO, J:ABC, J:ASSIGN, J:DCBLINK, J:FPOOL, J:JIT, J:RNST, M:XX, MDPO, MPPO, TSTACK. For a description of the contents of these cells consult Section VA.

FILES
TELLUSR reads the file ERRMSG which contains all monitor error messages. Section UB and the UTS System Management Guide, Chapter 8 detail the format of the file.

OUTPUT

TELLUSR output to the user is of the following format:

```
m m m
AT  YYY
```

where m m m is the monitor error message, YYYY the location where the error occurred.

If an I/O error is being reported, the DCB is appended to the above messages:

ON DCB d:dd

INTERACTION

GFBB to get a blocking buffer as a scratch pad

MSRRDWT to read the error message file

REGPRNT to print the PSD and registers (DUMP subroutine)

T:OVERLAY to overlay to MSROPN to open M:XX to ERRMSG

DATA BASES

TELLUSR uses a blocking buffer as a scratch area for building messages. The address of this buffer is maintained in R1.

SUBROUTINES

AT sets message 'AT XXXX' into the working buffer, where XXXX is the address at which the error was encountered

Access: BAL, 0
Uses: 5, 6, 7, 11

CHKDO routine to check the existence of the M:DO DCB

Access: BAL, 11
Uses: 0, 3, 4, 12

Exits normally if M:DO exists, exits skipping if not.

ERRMSGE the routine which reads the file ERRMSG with a specified key.

If an error is encountered on the read, the key is translated into EBCDIC and put in place of the expected message.

Access: BAL, 11
In: 1 = Buffer
12 = Key
Out: Message in buffer

FORM — sets a specified message into the working buffer

Access: BAL, 0
Uses: 4, 5, 8
In: 1 = buffer address
2 = Pointer into buffer
6 = Message address
8 = Count of characters in message

GETBUF — routine to get a blocking buffer as a work area. First attempts to get a blocking buffer via GFBB. If unsuccessful, GETBUF searches the DCB chain for an open file DCB and truncates it.

Access: BAL, 8 GETBUF
Out: R1 = Buffer address

GETWHO — routine which determines who aborted the user by analyzing the run flags in J:RNST.

Access: BAL, 0
Uses: 3, 5
Out: 5 contains index to who aborted the user:
0 = Monitor
1 = Processor
2 = User
3 = Loader

TRANS — translates specified word to EBCDIC and stores it in the output buffer.

Access: BAL, 0
Uses: 4, 5, 6, 7, 8
In: 1 = Buffer address
2 = Pointer into buffer
5 = Number to translate
7 = { 0 suppress leading zeros / 1 insert leading zeros }

WRITERR — the routine which opens M:XX to ERRMSG, forms a key from the user's error and abort code, and

03	00	ABC	ERO

calls ERRMSGE to read the error message. M:XX is then opened to the DO device and the message written to it.

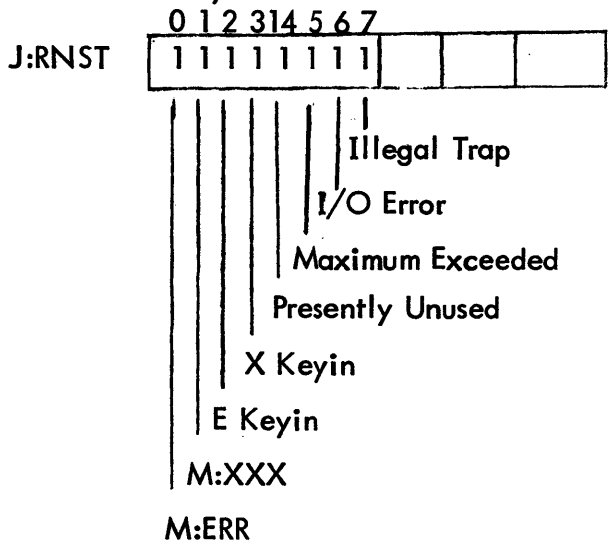
Access: BAL, 0

UTS TECHNICAL MANUAL

DESCRIPTION

TELLUSR

TELLUSR obtains a working buffer and branches to the appropriate routine as determined by J:RNST.



ILLEGAL TRAP

Routine which uses WRITERR to print the monitor error message, then prints "BY ( ) AT ( ) WHICH CONTAINS ( )" using routines GETWHO, AT, FORM, and TRANS.

IOERR

Routine to print the monitor error message and the DCB on which the error occurred. IOERR first calls WRITERR to print the error message, then attempts to locate the DCB associated with the error. If the DCB is found, and if M:DO DCB exists, the message "ON DCB d:dd" is printed. If the DCB is not located, the message "NON-EXISTANT DCB ADDRESS AT XXXX" is given.

MAXMSG

Routine which determines the exceeded maximums, stores the appropriate codes in J:ABC and M:ERO, and transfers control to ILLEGAL TRAP.

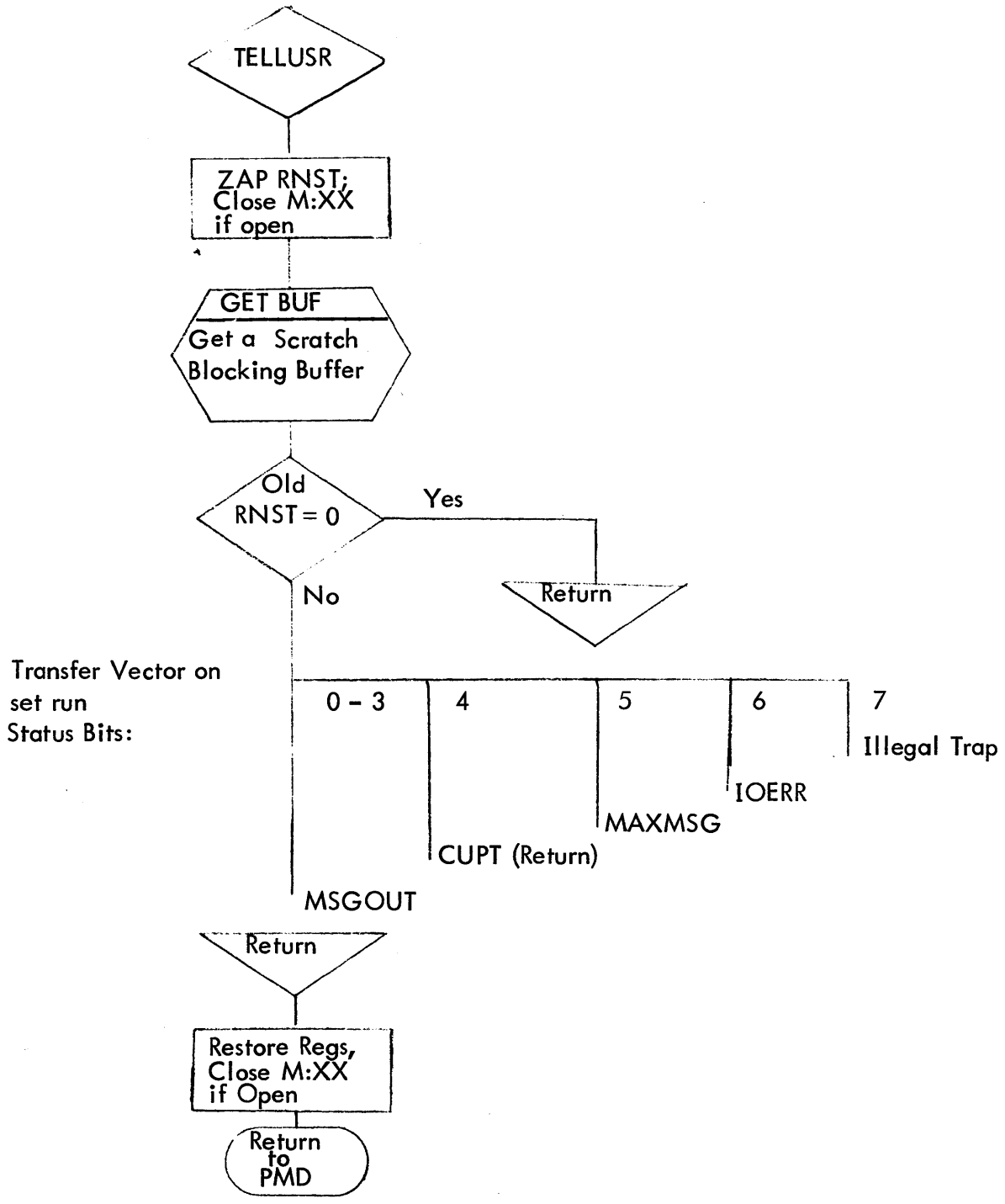
MSGOUT

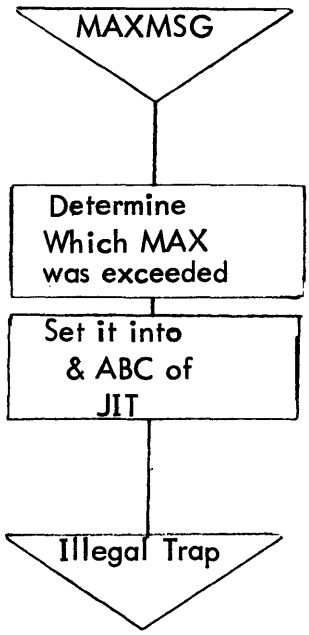
Routine which determines how the user was errored or aborted, and tells him. MSGOUT prints the message

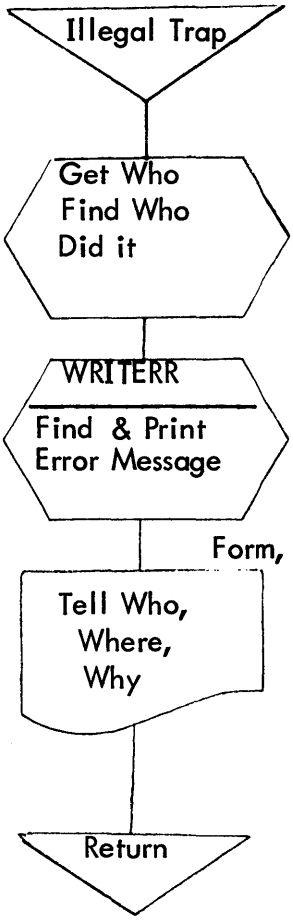
"JOB {ERRORED / ABORTED} BY {MONITOR / PROCESSOR / USER / LOADER / OPERATOR} AT XXXX"

using routines FORM, GETWHO, and AT.

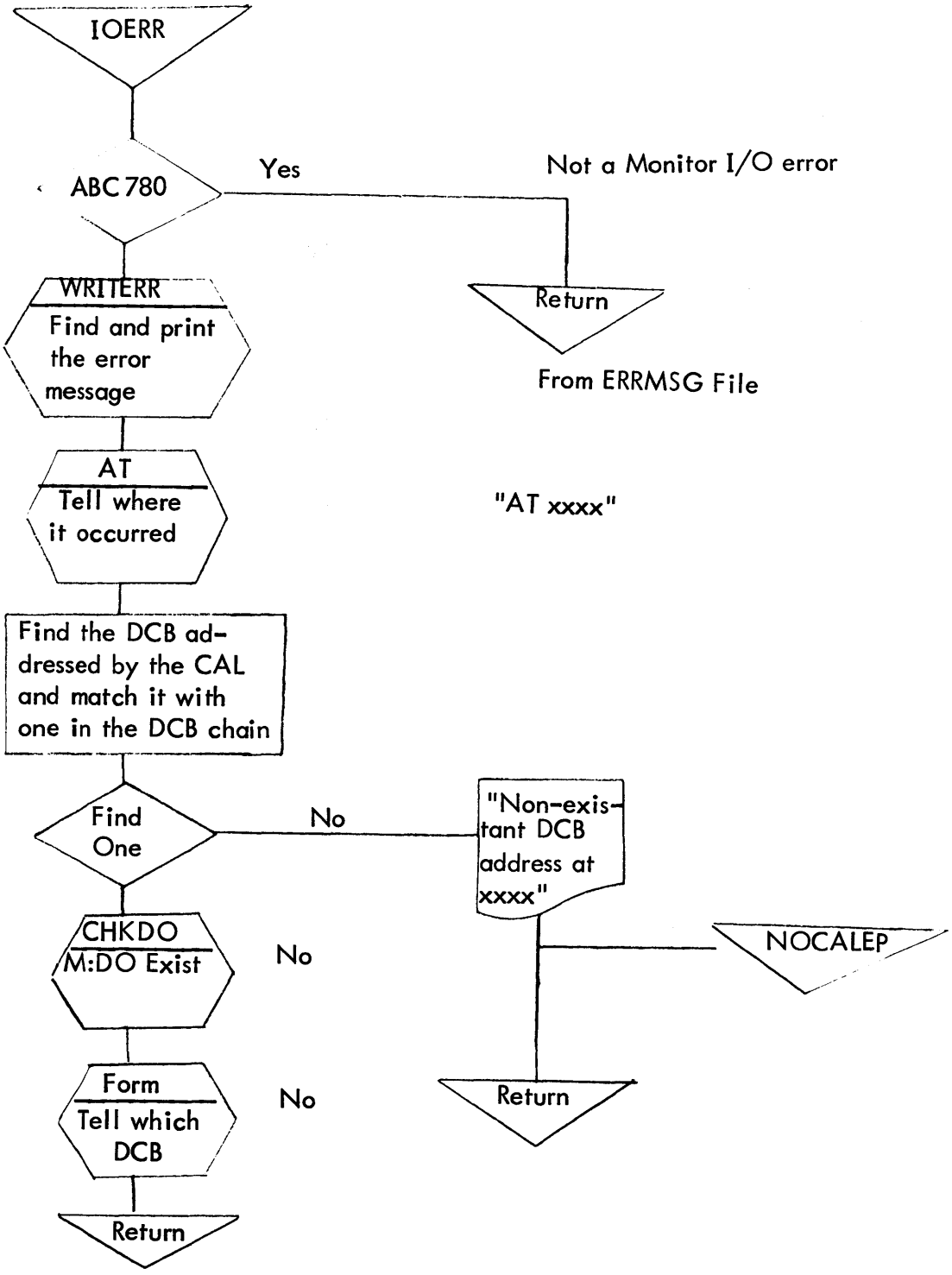
UTS TECHNICAL MANUAL

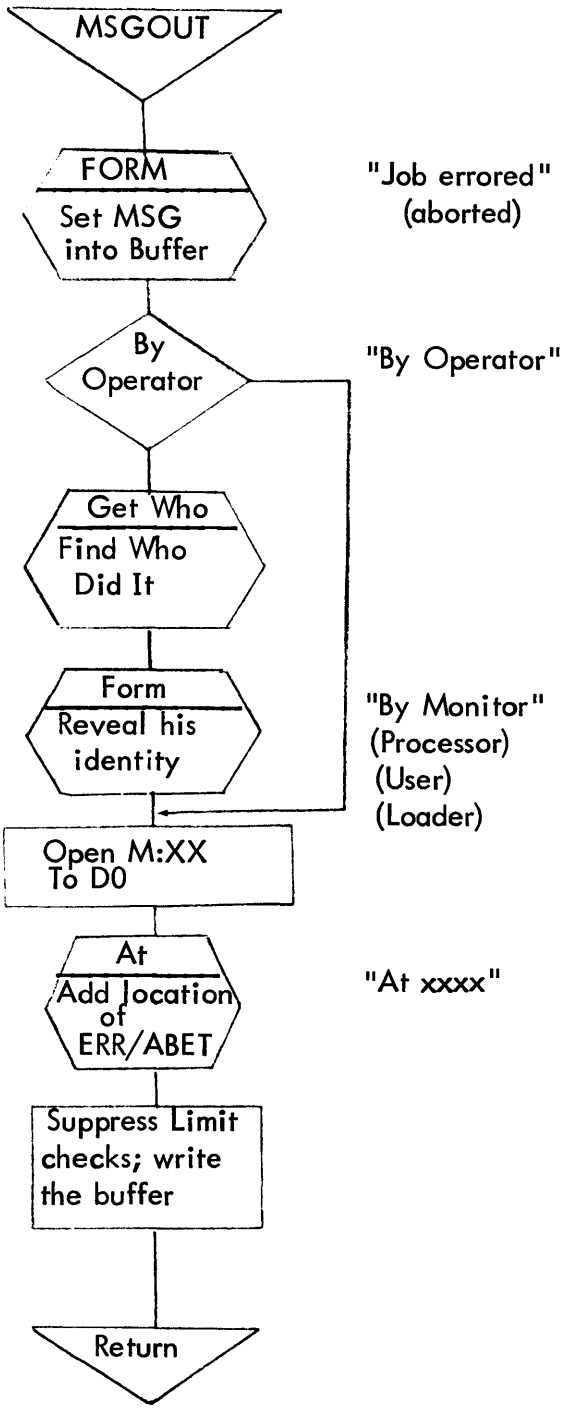



MAXMSG
Determine
Which MAX
was exceeded
Set it into
& ABC of
JIT
Illegal Trap

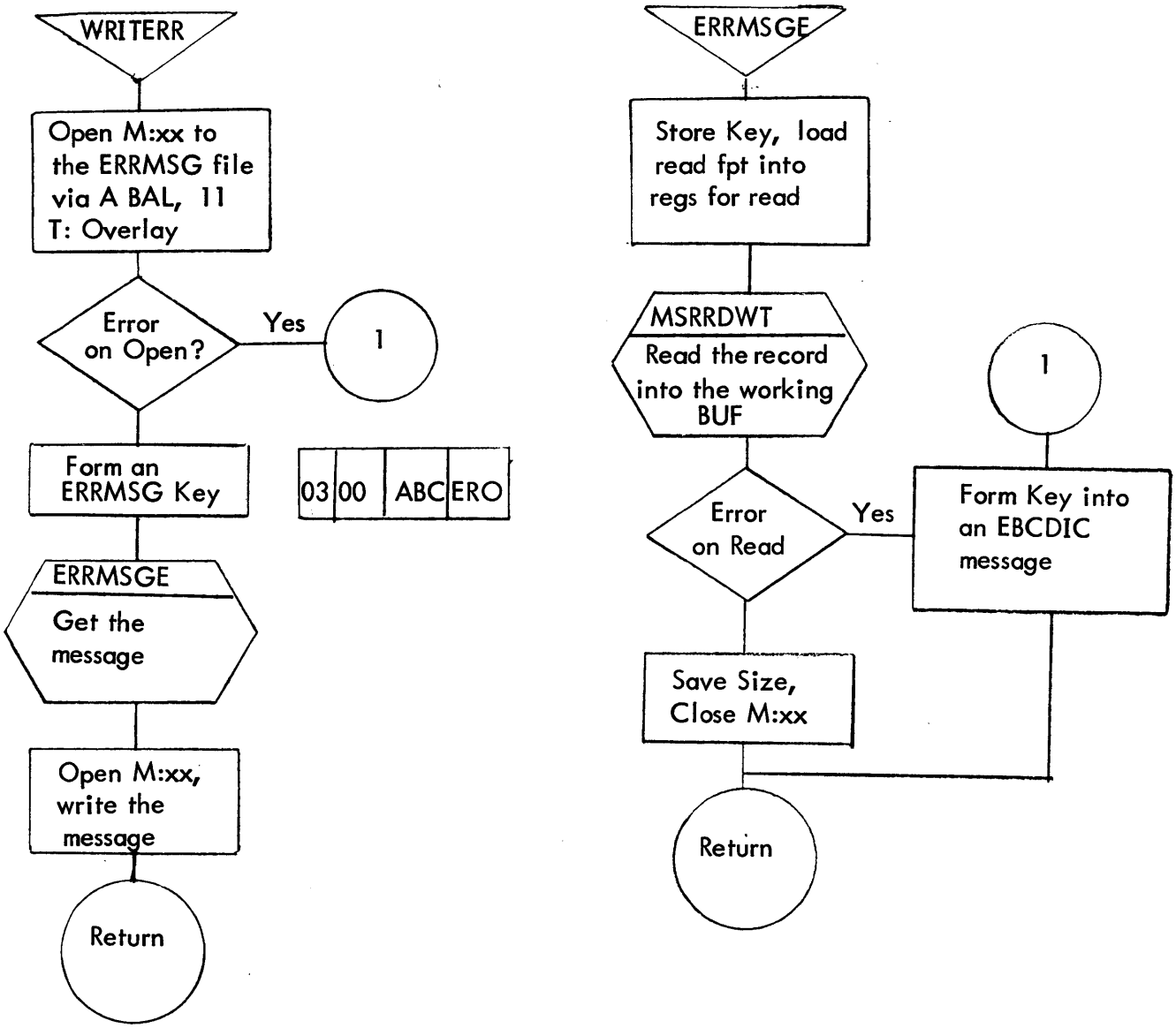

Illegal Trap
Get Who
Find Who
Did it
WRITERR
Find & Print
Error Message
Form, Trans, at
"By ( ) at ( ) which contains ( )"
Tell Who,
Where,
Why
Return

UTS TECHNICAL MANUAL




MSGOUT
FORM
Set MSG into Buffer
"Job errored" (aborted)
By Operator
"By Operator"
Get Who
Find Who Did It
Form
Reveal his identity
"By Monitor" (Processor) (User) (Loader)
Open M:XX To D0
At
Add location of ERR/ABET
"At xxxx"
Suppress Limit checks; write the buffer
Return

UTS TECHNICAL MANUAL



UTS TECHNICAL MANUAL

ID

DUMP

PURPOSE

To serve as main entry for PMD, PMDI, PMDE, dumps PSD, registers and memory for users.

ENTRY

BAL, 11 DUMPER

(R 6) = address of M:DO DCB.

(R 10) = address of user D4 in TSTACK.

(R 14) = address within blocking buffer that has a push down stack containing pairs of from-to address and a flag (must be first word pushed into stack). The flag non-zero implies an extended dump. The TS and TW bits must be set in the stack pointer doubleword (SPD).

(R 15) address of TSTACK

OPERATION

The from-to pairs of dump locations and the flag word are pulled from the stack in the Monitor buffer and pushed into TSTACK. The PSD and registers are printed via a BAL to REGPRINT. If an extended dump is specified, the JIT is printed via DUMPW. If the program is overlayed, the names of the overlays currently in core are printed. Next, all DCBs are printed via DUMPW and, if an extended dump is specified, the CFUs blocking buffers and Index buffers for these DCBs are also printed via DUMPW. Then all from-to pairs are printed via DUMPW. Next, any pages obtained by M:GP, any pages obtained by M:GCP then any pages obtained by M:GVP are printed via DUMPW. See Figure 1.

DUMPW

PURPOSE

To print contents of core locations in hexadecimal and EBCDIC format.

ENTRY

BAL, 11

(R 6) = address of output DCB.

(R 14) = address of output buffer.

(R 8) = low address to dump.

(R 9) = high address to dump.

(R 15) = TSTACK

ALTERNATE ENTRY

BAL, 11 DUMPWO (Same as DUMPW except that core addresses are not printed.)

EXIT

*11

OPERATION

Each 8 word block of core is formatted into the buffer at R14, then the buffer is written via PRINT.

PRINT

PURPOSE

To write a data record through a DCB.

ENTRY

BAL, 12

(R 6) = address of DCB.

(R 14) = buffer address.

RETURN

*12

ALTERNATE ENTRY

PRINTV

(R 15) = number of characters to print.

OPERATION

Routine pushes all the registers into TSTACK and does a BAL MSRRDWT

PRINTM

PURPOSE

To write a TEXTC message via a DCB.

ENTRY

BAL, 11

(R1) = address of TEXTC message.

(R6) = address of DCB.

EXIT

*11

OPERATION

This routine moves the message to the 34-word buffer, loads R15 with the byte count, adds one (for vertical format control), sets the first byte of the message in the buffer to an A (for double spacing), and calls PRINTV.

USERS PROGRAM STATUS DOUBLEWORD

xxxxxxxx xxxxxxxx *eeeeee*

where

x equals the hexadecimal representation of the PSD.

e equals the EBCDIC representation, if printable.

USERS GENERAL REGISTERS

xxxxxxxx xxxxxxxx ... xxxxxxxx *eeeeee*

xxxxxxxx xxxxxxxx ... xxxxxxxx *eeeeee*

where

x equals the hexadecimal representation (eight words per line) of the general registers.

e equals the EBCDIC representation, if printable.

current JOB Information Table (JIT).
if!PMDE was used the users JIT is listed

THE FOLLOWING SEGMENTS ARE PRESENTLY IN CORE

If the program is an overlaid program, list of the segments in core.

ALL USERS DCB'S FOLLOW.

List of user's DCBs.

SYSTEM CFU FOR ABOVE DCB.

If !PMDE was used and the DCB is open to a file, list of the CFU.

Figure 1. Format of a Dump Printout

SYSTEM INDEX BLOCK (IPOOL) FOR ABOVE DCB

If !PMDE was used and the DCB has an IPOOL assigned, list of the IPOOL.

SYSTEM BLOCKING BUFFER (FPOOL) FOR ABOVE DCB

If !PMDE was used and the DCB has an FPOOL assigned, list of the FPOOL.

USER SPECIFIED DUMP LIMITS FOLLOW

List any user-specified dump limits or protection types.

USER'S DYNAMIC PAGES FOLLOW

List of any presently allocated pages obtained by an M:GP procedure call.

USER'S COMMON DYNAMIC PAGES FOLLOW

List of any presently allocated pages obtained by an M:GCP procedure call.

USER'S VIRTUAL PAGES FOLLOW

List of any pages obtained by an M:GVP procedure call.

PRINTM

ID

SCREECH

PURPOSE

SCREECH provides the one "bail-out" exit from the UTS Monitor. The error code (SCREECH code) which is transmitted to SCREECH defines not only the problem but also which module discovered the problem. SCREECH loads and transfers control to Recovery.

USAGE

```
LI, 15          X'SCREECH code'
B               SCREECH
```

SCREECH codes are defined in Table LD-1.

The monitor symbol RECOVER is equated to SCREECH.

INPUT

There is no input for SCREECH except the calling sequence register 15.

OUTPUT

SCREECH pushes the registers into SAVEREGS (also equated to INITRCVR).

INTERACTION

All modules of the Monitor may call SCREECH. If switch 3 is down, the boot from RAD branches to MRECOVER so that register 15 is set to a -1 to indicate an operator call to SCREECH (MRECOVER EQ SCREECH-1).

DESCRIPTION

SCREECH pushes the registers into SAVEREGS, halts I/O on all devices in DCT1, loads recovery from the swapping RAD and transfers control RCVCTL.

TABLE OE-1, SCREECH Codes

CODE	CALLING ROUTINE	CONDITION
0	SSS	Event reported is not consistent with user's state.
1	CHECK	Either a user or processor's physical pages are not consistent (count not equal to chain) or more than one user is connected to the same physical page.
2	CHECK	State code and state queue are inconsistent.
3	CHECK	Stack pointer is self inconsistent.
4	CHECK	Number of users in the various states not equal to number of users.
5	CHECK	Name of processor not in word 2 of processor's procedure.
6	CHECK	SIR count inconsistent with sum in corresponding states.
7	CHECK	HIR count inconsistent with sum in corresponding states.
8	CHECK	Processor usage count inconsistent with number of associated users.
9	CHECK	Core page not accounted for by the totality of users.
A	CHECK	OP codes in swap I/O chain are not all read or all write orders.
B	CHECK	Seek or a TIC must occur in every I/O chain element (and doesn't).
C	CHECK	Swap attempt into monitor core area.
D	CHECK	Swap I/O command chain does not terminate where it should.
E	CHECK	I/O request with null command chain.
F	T:SIO	Input parameter function code is not read or write.
10	COC	Buffer with impossible address.
11	COC	Input interrupt on line with impossible state.
12	COC	Line state inconsistent with write operation.
13	COC	Line state inconsistent with read operation.
14	COC	Hardware entering bytes out of ring buffer.
17	IOQ	Illegal DCT index.
18	COOP	Address in sector not equal to address of sector read.
19	BUFGRAN	Attempt to return buffer with rediculous address.

CODE	CALLING ROUTINE	CONDITION
1A	OPEN/CLOSE	The world is gone (File System)
1B	SWAPPER (SSS)	At end of swap the number of pages remaining in the SWAPPER's FREE PAGE pool is non-zero
1B	AVR	
1C	Tables	Watchdog Timer Trap - PSM or PLM
1C	UCAL	Jit error, user name or account is all blanks
1D	T:OV	(Temporary) requested monitor overlay not in processor table
1E	IDLE	Entry to RECOVER at system quiescence if users have been deleted without normal shutdown – files may still be open
1F	SWAPPER	Swap scheduler has not provided enough pages for the requested swap
20	RCYL RNCUL RBG RMAT	Illegal to release specified disc adr
21	MM	Attempt to set AC on pg greater than 100
21	TYPR	Private volume allocation error.
22	RECOVER	Operator Recovery in order to revert to base monitor (A01)
23	Tables S9TRAPS	Uncorrectable Memory Parity or Map Check error
24	S9TRAPS	Malformed XPSD or MMC instruction, or invalid register designation in the monitor caused an Instruction Exception Trap
25	STEP	Unmapped
26	Tables S9TRAPS	A double PDF trap occurred
27	S9TRAPS	Loop Check or Overtemperature error
28	Tables	A Memory Parity Error occurred in a page not owned by the current user or while executing in the unmapped mode.

TABLE LD-1 SCREECH CODES (Cont'd)

29	S9TRAPS	Map Check or Data Base Check error while in Master Mode and the Register Altered bit was set
2A	SRCHF	Illegal DCT index
2B	GRAN	Granule release error
2C	ADD	Attempt to read beyond symbiont file end
2D	COOP	Buffer allocation error
49	TYPR	Tape Unit Allocation error
61		TEL/CCI trapped
62		User program too large
6A	MM	Sad Cal release without priv level
75	BUFGRAN	Attempt to release unallocated granule
79	ENTRY	Monitor stack blown
7A	COOP	No pages available for COOPerative buffers
7C	ALTCP	Got to ALTCP with a CAL1, 1 in CAL1, 2 which CAL1, 2 which CALPROC should have handles
50-5F	BPM/BTM	BPM/BTM specifics
7E	ALTCP	Monitor Trapped
FF	Disc Pack Handler	Cannot access header track

A screech code of X'80' or greater, indicates a swapper I/O error. Generally each bit or combination of bits has a meaning as follows:

0	1	2	3	4	5	6	7	
1								Swapper I/O error
1	0	1						Write Operation
1	1	0						Read Operation
1	1	1						Compare Operation
1			1					Transmission data error
1				1				Transmission memeory error
1					1			Memory address error
1						1		IOP memory error
1							1	IOP control error

Bits 3 - 7 are from I/O status bits 9 - 13.

0 0	Indicates bits 3 - 7 are not to be interpreted as status error bits.

If the byte is:

CODE	ROUTINE	CONDITION
80	TSIO	Sense ≠ seek on write
81	TSIO	Read check wasn't satisfied.
82	TSIO	Write check error - Bad order
83	TSIO	Not done with CL or write - no status
90	TSIO	Device unusual end or IOP halt occurred and no other bits were set.
91	TSIO	Write check not in system.
92	TSIO	In logging sector, IOCD bad.
93	TSIO	N write errors occurred and the offending command list can't be found.
94	TSIO	Discovered invalid order trying to continue write checking the rest of command list after N errors occurred.
95	TSIO	N read errors occurred and there is an invalid address pointing to the offending command list.
96	TSIO	N errors occurred trying to read a processor.

ID

ANALYZE - Monitor Debug Program

PURPOSE

ANALYZE is the ghost job wakened by recovery which prints a map and a summary of the crashed monitor. ANALYZE may be run on-line or in batch to summarize the monitor dump or the running monitor.

OVERVIEW

ANALYZE consists of the following general groups of routines: initialization, command scanner, main command routines, generalized subroutines. Initialization obtains working space and determines the user's identity (ghost, batch, or on-line). Control is transferred to the command scanner if the user is on-line or in batch, or a summary routine if the user is a ghost.

The command scanner reads the teletype, breaks the command into its component parts, and transfers control to the specified main routine to process further. In the case of a ghost job, the summary routine transfers control to each main routine in order. These routines return to the command scanner which will return control to the summary routine if a summary is in progress, and in batch commands through the card reader function similar to the on-line processing.

The main command routines may examine the user's command further, then perform the task requested, summarizing the COC line tables for example.

The generalized subroutines perform common operations for the main command routines. These routines translate and format data, print data, access locations, etc.

Figure LE-1 shows the interactions of the routines making up the ANALYZE processor, as well as an idealized picture of where things are done.

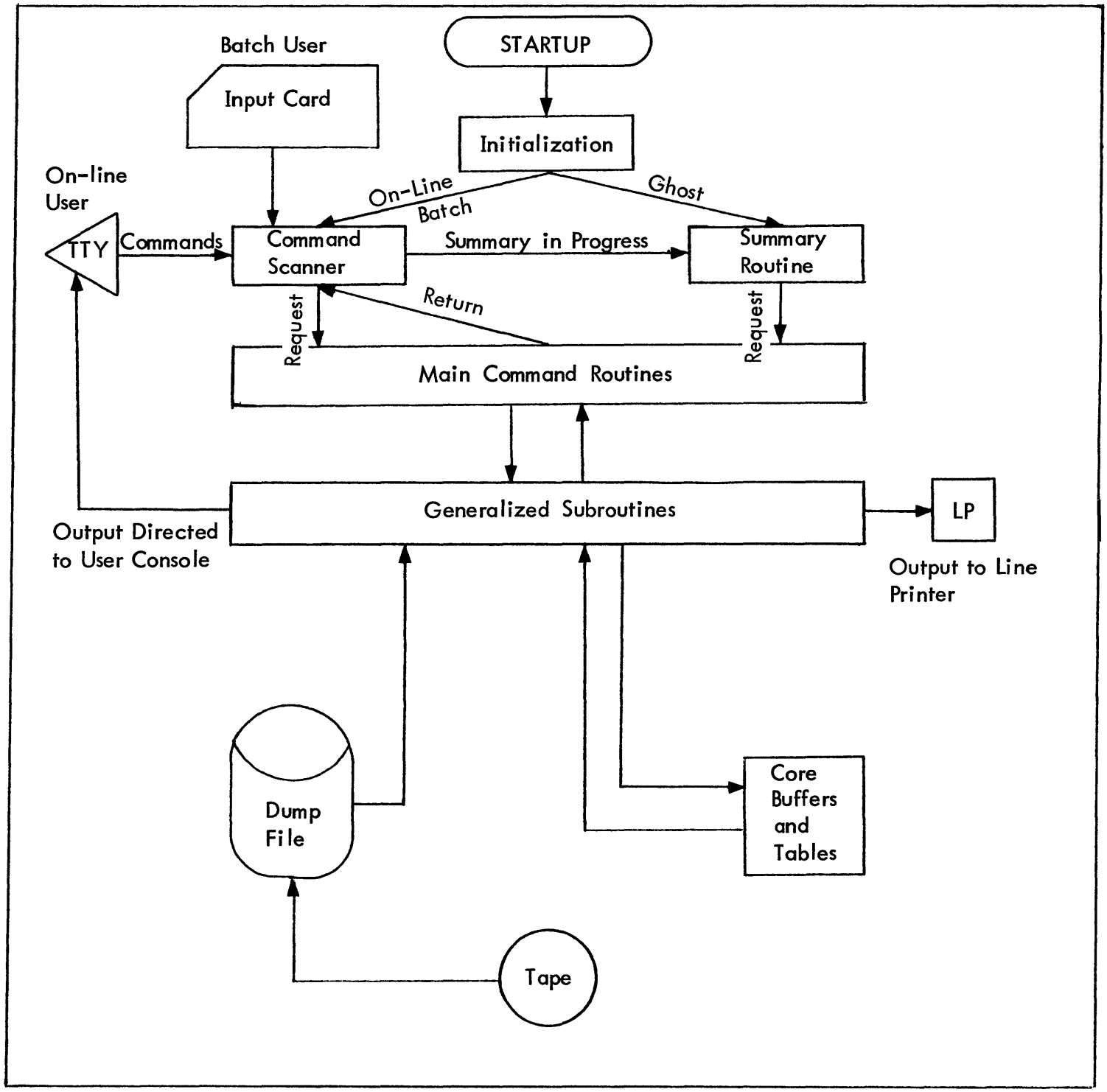


Figure LE-1. Interaction of Analyze Routines

USAGE

ANALYZE is initiated automatically during the recovery process. Following an unrecoverable crash, ANALYZE may be initiated by the operator via the keyin

!GJOB ANLZ

to dump the tape produced by the crash.

ANALYZE may be accessed on-line by typing

!ANLZ

from any account. Consult the UTS System Management Reference Manual, for details of using ANALYZE.

INPUT

Files:
ANALYZE reads the monitor dump file produced by recovery

MONDMPn

where n is the number of the crash assigned by recovery in the range 0 - 7. Each record in the file is a page (512 words) of the crashed system, with key

03	00	00	k

where k is the page number. Included in the file are the user JITS from the swapping RAD, with keys of the format

03	00	u	00

where u is the user number.

ANALYZE produces a map from the file

MONSTK

under the :SYS account. MONSTK contains the REF/DEF stack for the monitor. Consult the loader documentation for the detailed format of a REF/DEF stack.

Tapes:
ANALYZE reads the tape produced by an impossible recovery with INSN

RCVT

This tape is in label format, with LABEL

TAPDUMP

Each record a page (512 words) or the crashed monitor. From this tape ANALYZE produces the file

UTSDUMP

with exactly the same format as MONDMPn.

Buffers and Tables:
See DATA BASES for internal Buffers and Tables. ANALZ formats many of the monitor tables, for a detailed description consult the Technical Manual, Section V.

OUTPUT

For a detailed description of ANALYZE output, see the UTS System Management Reference Manual.

Buffers and Tables - See DATA BASES.

INTERACTION

ANALYZE uses the following monitor services:

I/O functions:

M:OPEN	
M:CLOSE	
M:READ	
M:WRITE	
M:SETDCB	to change ERR/ABN returns
M:DEVICE	to skip to top of form

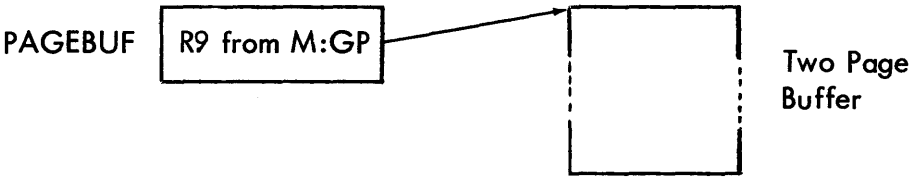
Memory Management:

M:GP	to get buffers
M:FP	to free them
M:GVP	to get a specific page
M:FVP	to free it
M:CVM	change Virtual Map to access the monitor

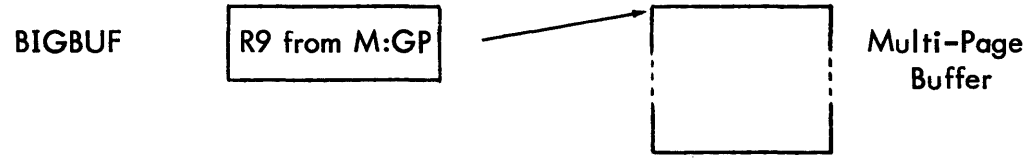
DATA BASES

Buffers:

PAGEBUF — A word containing the address of a two page buffer acquired during initialization into which the dump is read or the monitor is mapped two pages at a time.



BIGBUF A word containing the address of a large buffer acquired when a map is requested. This buffer expands until MONSTK can be read into it with no loss of data. It is then used as a scratch buffer for sorting and printing MONSTK.



Command Input Buffers and Pointers:

UCBUF Buffer into which ANALYZE commands are read (80 bytes)

FIELDS1-5 Two-word buffers into which fields from UCBUF are stored

F1-5 Two-word buffers into which subfields (field following a + or -) of corresponding main fields are stored. This arrangement allows only one level of addition or subtraction.

$FIELD_{1-5}$ Pointers into FIELDS1-5

$OPFIELD_{1-5}$ Pointers into F1-5

CHRS SP, CR, LF, comma, +, -, ↓, =, #
used as field and subfield separators

OPS Bytes to contain associated +/- operation

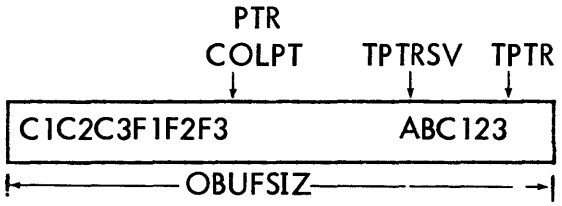
Example: UCBUF DISPLAY↓ JIT, 1, 60↓ 60 + 100

	$FIELD_1$	$OPFIELD_1$	OPS
FIELD1	DISPLAY	F1 0 - 0	0
FIELD2	JIT	F2 0 - 0	0
FIELD3	1	F3 0 - 0	0
FIELD4	60	F4 0 - 0	0
FIELD5	60	F5 100	+

Output Buffer and Pointers:

OBUF	The buffer into which ANALYZE formats its output to the user
PTR	Points to the next available position in OBUF
TPTR	Points to the next available translation position in OBUF (for translation of dumps to EBCDIC only)
TPTRSV	Original starting position for translated characters
COLPT	Position of the TTY carriage on the output line
OBUFSIZ	Size of OBUF

Example:



Dynamic Flags and Cells:

BALL	1 = summary command in progress. Checked by SCANNER for a return to ALL.
BRKCNT	1 = user hit break. Checked in MSG and BUFOUT.
BUFLIM	Doubleword set up by INITIAL containing limits of the working buffer.
PSTACK	Stack to contain DELTA request environment.
FILETEXT	Text of LMN from which DELTA obtains a symbol table.
GETFLAG	1 = DELTA get operation 0 = DELTA put operation
KEEPKEY	A blank key 4 bytes in length
KEY	Key for reading and writing (core dump file).
LASTWORD	The last word examined in the dump; to be compared with the next.
LASTLOC	Last location dumped.
LPFLAG	1 = output is directed to the line printer
MAPFLAG	1 = requested pages must be translated through the user's map
MASQ	adjustable search mask
MONFLAG	1 = examining current monitor rather than a dump.

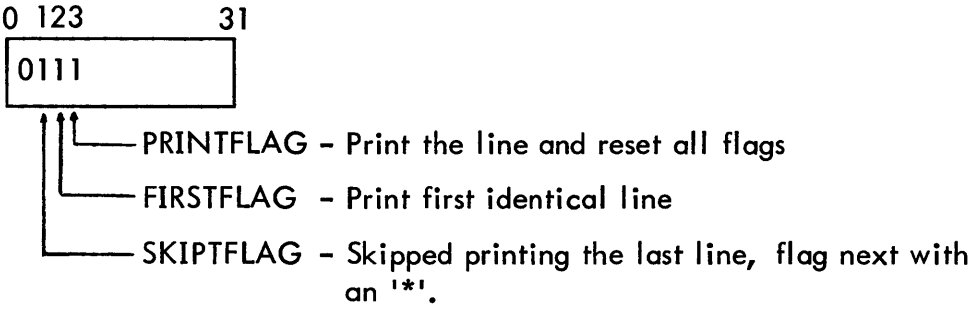
NOPS Count of operations in summary command

OLDPAGE Last page retrieved for the user

OLDPAGEM Last page requested by the user (not necessarily = OLDPAGE)

PAGEBUF Location of dynamic working buffer

PFLAGS Word containing in upper half byte flags determining the status of the line being dumped



RBUFEND Pointer to the end of the event recorder

RBUF1 Pointer to the beginning of the event recorder

READAGAINFLG 1 = changed origin of input, must read OLDPAGE again.

REPFLAG 1 = AN = sign was encountered, this operation is a replace

RDSIZE Buffer size word in RDSTK fpt used for reading MONSTK

SERVAL Cell containing value for which to search

STACK 80 word analyze tempstack; reset by the SCANNER routine

STKSIZE Cell to contain the size of MONSTK after it is read successfully

SUPPLZ 0 = suppress leading zeros on translation
1 = translate leading zeros

TRAPPAGE Address at which the monitor trapped (or zero)

TRTOTAL Count of events to trace in the event recorder

UNDERDELTA 1 = operations being performed under DELTA

VIRPAGE Virtual address word in Change Virtual Map CAL fpt

WRITELOC Buffer address word in Write fpt used for writing UTSDUMP

Tables:
Three types of tables are common in the main command routines. Type 1 determines which monitor table to dump next, by loading R14 with its address for input to GETADDR. Type 2 determines its resolution, by LB, LH, LW or LD instructions into R3 for input to TRANS or TRANSSZ. Type 3 dictates the spacing between the printed values, containing the tab stops.

The following list of tables is grouped by the main routines using them:

Routine	Table Name	Type
COCODE	GETCOC	1
	COCACT	2
	COCSP	3
IODISPLAY	CITS	1
	GETCIT	2
	CITSPACES	3
	DCTS	1
	GETDCT	2
	DCTSPACES	3
	IOQS	1
	IOLOAD	2
	IOSPACES	3
PROCS	LOADP	1
	ACTP	2
SWAP	FINDTAB	1
	LOTAB	2
	TABTAB	3
	LISWAP	1
	LWSWAP	2
USERS	UTABS	1
	ACTION	2

Other Tables:

Table	Routines	Use
CHRS	SCANNER	List of command separators
COMMANDS	SCANNER	List of the first two characters of commands
CVEC	SCANNER	Command transfer vector
EBCDIC	TRANS	Translate table: Binary - EBCDIC
EVENTS	TRACE	TEXTC table of events
JITPAG	RUN, JITS, AJOUT, MAPMODE, LOCJIT	Table of unmapped addresses of the user JITS
LIST	TRANS	TEXT string of the hex numbers for binary EBCDIC translation

MAP	MAPMODE, GETPAGE	Table loaded with a user's CMAP image
SCRMSGS	REGS	TEXTC table of Screech messages
SCRS	REGS	Table of addresses pointing to SCRMSGS
STATEX	TRACE, RUN, USERS	TEXTC table of states
SVEC	SCANNER	Transfer vector for action on separators

SUBROUTINES

Register conventions: Main routines accessed from CVEC may use all registers. The following subroutines preserve registers used in the stack, and are accessed by a

BAL, RO (routine)

BITPUT Routine to convert a byte to EBCDIC 1's and 0's in the output buffer.
IN: R3 contains byte to be translated, right-justified

BUFOUT Routine to write contents of the output buffer to M:LO DCB.

DUMPSOME Prints a formatted memory dump for on-line and batch users. Translates the dump as well as deleting identical lines.
IN: R7 = number of locations to dump
R8 = starting address in PAGEBUF

GETADDR Routine to obtain dump page containing address specified and return an equivalent pointer into the working buffer. Uses the subroutine GETPAGE.
IN: R14 contains desired location
OUT: R15 contains pointer to the desired location

GETHEX Routine to convert a location specified in an EBCDIC command to its hex value
IN: R1 = FIELDn
OUT: R2 = hex value

GETPAGE Given a page number, reads that page and the next from the dump file into the working buffer, or if MONFLAG = 1, maps the desired page of the current monitor and the next into the working buffer.
IN: R1 = desired page number

LOCJIT Subroutines to interpret the JIT address table in the monitor (or dump) and return the appropriate JIT address in R14.

LOCLOC — Routine to return in hex the starting and ending locations specified in a command, and the difference, when given the field in which to expect the starting location in EBCDIC

IN: R1 = FIELDn
OUT: R7 = difference
R8 = starting location
R9 = ending location

If REPFLAG = 1, R9 will be put into the location specified in R8.

MSG1 — Inserts TEXT message into output buffer
IN: R1 = address of message; R2 = byte count.

MSG — Inserts TEXTC message into output buffer
IN: R1 = address of the 1st word of the message in TEXTC.

TRANS and TRANSSZ — Routine to translate binary word into EBCDIC and store it into EBCDIC and store it into OBUF. If TPTR $\neq$ 0, the EBCDIC equivalent of the word will also be placed in OBUF.

TRANSSZ is the entry point to suppress leading zeros, TRANS leaves them in.

IN: R3 contains the word to be translated.

SPACES — Routine to insert spaces in OBUF to a specified column.

IN: R1 contains pointer to desired column number in OBUF.

SPACE2 — Routine to insert two spaces into OBUF.

MDSNAPY — Routine to dump area specified in the MONDUMP format.

MDIOSYM — Routine to dump the IOQ, DCT, CIT and symbiont tables in the MONDUMP format.

MDDCB — Routine to dump the active user DCBs in the MONDUMP format.

MDCORE — Entry point to set up the 0 through 5000 Monitor root dump.

ERRORS

Error Messages	Meaning and Action to Take
EH ?	ANLZ did not understand the command. Retry.
CANNOT OPEN FILE (NAME)	An unsuccessful attempt was made to open the specified file as input. Check file's existence.

Error Messages	Meaning and Action to Take
CANT ASSOCIATE DELTA	Assocaite DELTA CAL was unsuccessful. Major difficulty, SIDR it.
CANT GET THE BUFFER	Unable to get a buffer large enough to read MONSTK. Increase user page limit with SUPER.

ERR/ABN CODE = CODE | 00 | DCB	Accompanies most I/O error messages. Consult UTS Reference Manual for codes.
INCORRECT DUMP PAGE n	Data in page n fails a consistency check. Try to create the dump from tape.
LOC1 GREATER OR EQUAL LOC2	Negative range specified in a command. Retry.
NO FILE OPEN FOR INPUT	Attempt made to read a non-existent file. Type INPUT command.
NOTHING TO DISASSOCIATE	Unsuccessful disassociate DELTA CAL. Don't worry about it.
ZZ PRIVILEGE NOT HIGH ENOUGH	Message resulting from an unsuccessful Change Virtual Map CAL. Increase privilege level with SUPER. ZZ indicates your current privilege level.
SORRY, NO PAGE n	Record with key n does not exist in the dump file. Try to recreate file from tape.
TYPE INPUT COMMAND	M:EI not open and MONFLAG = 0.
YOU MUST BE LOOK AT THE MONITOR TO CHANGE IT	Replacement attempted with MONFLAG = 0. Type "MONITOR DISPLAY", and retry.
CANT READ THAT JIT # XXXX	An attempt on-line to read a user JIT from the file, and that record is not present.

ID

ANALYZE - Commands and Routines

ALL The ALL command uses most of the main routines to summarize the dump or the running monitor. As all routines return to the SCANNER, the flag BALL is set to indicate an ALL operation is in progress.

Using the cell NOPS to contain the number of routines to execute, ALL decrements it, then uses the value as a displacement into a branch table to each routine. When NOPS becomes zero, BALL is reset. If the user is ghost or batch, ALL dumps memory from zero to X'5000' and exits. Otherwise, control is passed to SCANNER to process the next command.

Entered From: SCANNER from BALL $\neq$ 0 test.

Exits To: Main command routines, or portions of them; to EXIT or SCANNER when completed.

ALLJIT The routine ALLJIT is not accessed from SCANNER, but is called upon by ALL to print the current user's JIT, AJIT, and context area, and the inswap and outswap users' JITs and AJITs.

ALLJIT first builds a list of users - current, inswap, outswap, and all others in core. Running down the list, it accesses, formats and prints each user's JIT and AJIT. If the RUNMODE is ghost or batch, the JIT page will be output in the MONDUMP format; if on-line, the LP command will enable the MONDUMP type display; otherwise, the output will be in a four-column Hex dump. To print the current user's context, the routine MAPMODE is used to load that user's map to facilitate accessing his context pages. Then JB:CMAP is consulted to determine which pages to dump. Page entries other than FPMC and NPMC are dumped within the limits of his context area.

Entered From: ALL

Exits To: SCANNER

COMPARE Command: COMPARE LOC1, LOC2

The routine COMPARE compares a monitor dump from the file UTSDUMP and the running monitor between the specified locations. A dynamic page is obtained and the page of the monitor containing LOC1 is mapped into it. The corresponding page of the dump is obtained by GETADDR.

Corresponding locations are compared and when an inequality is found, the loaction and the differing contents are printed. If the range exceeds a page, the next page of the dump and the monitor are obtained.

Entered From: SCANNER
Exits To: SCANNER

DISPLAY Command: DISPLAY op[, id][, LOC, LOC]

op = USERS, JITS, or COC
op = JITS only

In order to format and print the tables associated with the specified request, the Display command drives into a transfer vector to the routines: COCODE, IODISPLAY, JITS, PROCS, ID, REGS, SWAP, USERS, P1, P2, P3, P4, PA, SY, PR.

The routines which summarize monitor tables use a table driven technique to determine the monitor table, its resolution, and the spacing to the next column. The tables are described in DATA BASES; use is as follows:

Table type 1 is accessed by either a LOAD WORD or an EXU indexed by an internal ANALZ table number in the range from 0 to total number of tables in the display. The result of accessing table type 1 is loading R14 with the physical address of a particular table. This is followed by a BAL to the subroutine GETADDR, which returns with R15 pointing to the desired location in the working buffer. Table type 2 is now accessed via an EXU, loading a byte, halfword, word, or doubleword into R3. This is generally followed by a BAL to TRANS or TRANSSZ to translate the contents of R3 and place it into the output buffer. Table type 3 is accessed, generally by a load byte indexed by column which will load R1 with the next tab stop. A BAL to SPACES inserts the requisite spacing. Indexes are incremented, and the cycle continued until completed.

COCODE Command: DISPLAY COC

The COC routine displays the following COC tables: LM:UN, COCTERM, STATE, PROMPT, CODE, MODE, MODE2, MODE3, FLAG, COCBP, TL, LINK.

The routine first prints a header, then accesses and prints the above tables for a specified line number, or for all line numbers for which the LB:UN entry is non-zero. Two tables are special-cased, COCTERM and STATE: instead of being translated via a BAL to TRANSSZ, the terminal type and line state are placed into the output buffer by MSG.

Entered By: ALL, DISPLAY
Exits To: SCANNER

IODISPLAY Command: DISPLAY I/O

The routine IODISPLAY formats the channel information tables, selected device information tables for devices on each channel, and all the I/O queue tables for requests queued on each channel.

IODISPLAY first builds a list of channels which have devices, then using this list formats and prints CIT3, CIT4, and CIT5 for the first channel. The DCT tables are searched for devices on this channel, and for each device the tables DCT16, DCT1, DCT2, DCT3, DCT5, DCT6, DCT12, DCT13 are formatted and printed. DCTs 3 and 5 are formatted as bits, to more easily read the tables.

After all device tables are printed, CIT1 is checked to see if any requests

are queued for this channel. If so, each IOQ table is formatted and printed (IOQ1 - 15) and IOQ2 is checked for another request. All entries for requests are printed.

If no requests are queued, or when they are all printed, the channel number is bumped and IODISPLAY continues down the list. When all channels have been accessed, IODISPLAY returns to SCANNER.

Entered From: ALL, DISPLAY
Exits To: SCANNER

JITS Command: DISPLAY JITS, [M, id] [LOC1, LOC2]

The routine JITS determines which JIT to print, reads that JIT into PAGEBUF, obtains the specified limits (if any), and dumps that portion via DUMPSOME. Control is then returned to SCANNER.

Entered From: ALL, DISPLAY
Exits To: SCANNER

PROCS Command: DISPLAY PROCS

The routine PROCS formats and prints the processor tables PB:HPP, PB:TPP, PB:PSZ, PB:DSZ, PB:DCBSZ, PH:PDA, PH:DDA, PB:UC, PB:LNK, PB:PVA, PB:HVA, P:SA, P:TCB for all processors with non-zero size entry in PB:PSZ.

Entered From: ALL, DISPLAY
Exits To: SCANNER

REGS Command: DISPLAY REGS

The routine REGS determines the cause of the crash with R15, prints an appropriate message, and dumps the registers from SAVEREGS.

If the crash was caused by a monitor trap, its address from R0 is set into TRAPPAGE to allow ALL to dump that page.

Entered From: ALL, DISPLAY
Exits To: SCANNER

SWAP Command: DISPLAY SWAP

The routine SWAP formats and prints the swap tables S:SIR, S:HIR, S:SIP, #SWAP$DEV, S:CUN, SISUN, S:CUIS, S:IDLF, SB:OSN, SB:NP, SB:FPN and the SWAP lists SB:OSUL, SB:PNL, SB:FPL, S:BECL. In addition the following tables indexed by swap device are dumped: M:SWAPD, MB:SDI, MB:SFC, MB:#RTRY, M:CLBGN, MH:CLEND.

Entered From: ALL, DISPLAY
Exits To: SCANNER

USERS Command: DISPLAY USERS[,id]

The routine USERS displays the following tables for all users or a specified user: UB:US, UB:BL, UB:FL, UH:FLG, UH:FLG2, UB:JIT, UB:SWPI, UH:JIT, UH:AJIT, UB:PCT, UB:APR, UB:APO, UB:ASP, UB:DB, UB:OV, UH:TS, UH:ID.

USERS first prints a header, then accesses and prints the above tables for a specified user or for all users with their entry in UB:US $\neq 0$.

Each user's state is output in EBCDIC via a BAL to MSG. Otherwise, the tables are printed as described in DISPLAY.

Entered By: SCANNER and ALL
Exits To: SCANNER

PARTITIONS Command: DI[SPLAY] PA[RTITIONS][,ID]

The routine PARTITIONS displays the following tables for either specified partition number or all partitions: PLO=ACT, PLB:USR, PLH:FLG, PLH=QN, PLH:TOL, PLH:CVR, PLH:TL, PLH:TV, PLH:SIP, PL:MIN, PL:MAX.

PARTITIONS first prints a header and then accesses and prints the above tables for specified partitions or all.

SYMBIONTS Command: DI[SPLAY] SY[MBIONTS]

The routine SYMBIONTS reads the RBBAT recovery file and dumps the contents with a Header denoting the information being displayed.

P1 Command: DI[SPLAY] P1

The subroutine MDIOSYM is entered via this command to format all IOQ, DCT, CIT/BPLBL, and output symbiont tables.

P2 Command: DI[SPLAY] P2

The subroutine MDCORE is entered from this command to dump core locations $0 \rightarrow 5000_{16}$ in the MONDUMP format.

P3 Command: DI[SPLAY] P3

The subroutine MOTRAPS is entered from this command to dump the TRAP and INTERRUPT locations in MONDUMP format.

P4 Command: DI[SPLAY] P4

The subroutine MDDCB is entered from this command to dump the active user DCBs in MONDUMP format.

ID Command: DI[SPLAY] ID[,#]

The subroutine UID is entered from this command to list all active users or the number specified. The user number, origin, account number and user name are listed.

PROCESSORS Command: DI[SPLAY] PR[OCESSOR][,][VALUE]

The PROCESSOR command displays either the given processor number, or all processors in the system. All tables associated with the processors (PB:HPP, PB:TPP, etc. (are displayed.

ROWS Command: RO[WS] [VALUE]

This command sets the Hex column count for Hex dump displays; standard on-line Hex displays are four columns wide; batch and ghost line printer output is eight columns wide. 12 ($\leq$16) is the maximum for a line printer, and four is the maximum for the user console; any value in between is acceptable.

CL[OSE] If the batch or on-line user has been examining a crash file and now wants to examine the running Monitor, the CLOSE command will close up the open crash file and r eturn to SCANNER for the next interactive command.

DUM, DLAST, DNEXT, INDR Command: LOC1, LOC2;↑; LF; *

The routine DUM obtains the specified limits from the command, saves the last location to be dumped in LASTLOC, gets the page containing the first location into PAGEBUF, and uses DUMPSOME to dump the range.

DLAST	Specifies 1 location ((LASTLOC)-1) to DUM
DNEXT	Specifies 1 location ((LASTLOC)+1) to DUM
INDR	Specifies 1 location (*(LASTLOC)) to DUM

Entered From: SCANNER
Exits To: SCANNER

EXITCL Command: END

EXITCL closes the DCB M:EI with a release if the user is a ghost, and with save if not. The routine then executes a normal exit to the monitor.

Entered From: ALL, SCANNER

INITIAL The entry point to analyze from SCANNER, INITIAL first acquires the two-page working buffer and stores its location in PAGEBUF, its limits in BUFLIM. The type of user is checked, and if ghost, control is passed to LASTCRASH, one of the input routines to open and summarize the most recent monitor dump file. If on-line INITIAL takes break control, informs the user he is accessing ANALYZE, requests him to specify his input source via an INPUT command, and branches directly to SCANNER.

Entered From: Start location of ANLZ load module is INITIAL. Entered when ANALYZE is first accessed.

Exits To: To the SCANNER or the ALL command logic.

INPUT Command: INPUT

The input routines INPUT, SPECFIL, LABEL$TAPE, TAPEP, LASTCRASH, OPNUTSD result in the M:EI DCB open to some dump file consisting of keyed records one page long, the keys corresponding to the page numbers.

INPUT is a transfer vector to the other routines

SPECFIL opens a recovery-created file MONDMPn, where n is specified in the input command. Example: IN ↹ 6 results in opening M:EI to MONDMP6.

LABEL$TAPE reads the tape created by recovery after a recovery impossible situation, and forms a dump file called UTSDUMP, to which M:EI is opened.

TAPEP is similar to LABEL$TAPE except it reads a tape created by the EXEC DELTA ;V command

LASTCRASH opens the most recent MONDMPn.

OPNUTSD opens M:EI to the existing UTSDUMP file.

LP Command: LP [,value]

The routine LP closes M:LO and re-opens it specifying device LP, value indicates the number of Hex dump columns on the print page. The default is eight.

Entered From: SCANNER

Exits To: SCANNER

MAPMODE Command: MAP [,id]

The routine MAPMODE obtains the specified user number, reads his JIT into PAGEBUF, loads ANLZs internal MAP from the user's JB:CMAP, and sets a flag to indicate to the GETPAGE routine that mapping is in progress. Control returns to the BAL+1.

Entered From: SCANNER, ALLJIT

Exits To: BAL+1

MONITOR Command: MONITOR [DISPLAY]

The routine MONITOR sets and resets the flag MONFLAG depending on the presence of the DISPLAY part of the command. In either case, READAGAINFLG is set to require GETPAGE to renew the old page if accessed.

Entered From: SCANNER
Exits To: SCANNER

PRINT Command: PRINT

The routine PRINT issues a Superclose CAL to close symbiont files.

Entered From: SCANNER
Exits To: SCANNER

REPLACEMENT Command: LOC = VAL

The routine REPLACEMENT allows the ANLZ user to alter the running monitor. If the user is not looking at the monitor (MONFLAG = 0), he is not allowed to change it. Otherwise, the page containing the specified location is mapped into PAGEBUF, and the value is stored into it.

Entered From: SCANNER
Exits To: SCANNER

RUN Command: RUN op, id

The RUN routines UPAGES, MPAGES, PPAGES and STATES display threaded lists of pages and states. These routines pick up the head, tail, and count in the list and pass them in R4-6 to the subroutine PGSOUT, which displays the list as a chain. An exception to the above procedure is the routine STATES, which passes the head, tail, and count in R4-6 to a PGSOUT subroutine HLOOP.

Subroutines:

PGSOUT In: R4 Head of chain
R5 Tail of chain
R6 Count

Linking Register = RO

HLOOP In: Same as PGSOUT

Linking Register = R12

Entered From: CVEC or ALL command
Exits To: SCANNER

SCANNER SCANNER breaks commands into their component parts and partially interprets them. Then transfers control to the appropriate routine.

Each main routine is branched to directly from CVEC in SCANNER; upon completion of its function, the main routine returns directly to SCANNER.

SCANNER is also the "bail-out" for any routine in trouble, so the first thing done is to clear the temp stack. In the case of the summary command ALL, when each routine returns to the SCANNER, it recognizes the ALL command in progress, and returns control to the ALL routine.

Entered From: INITIAL, as normal return for all main routines, as "BAILOUT" for most error conditions.
Exits To: Command routine specified by the user.

SEARCH Command: SEARCH VAL, LOC1, LOC2

The SEARCH routine searches under the mask MSQ for the specified VAL between the specified limits of LOC1 and LOC2 in the dump file or the running monitor. SEARCH uses the GETADDR subroutine to read the pages containing LOC1 and LOC2 into PAGEBUF then compares the specified VAL with the contents of PAGEBUF starting at LOC1. Each time a match is found, the location and contents is printed in hex form using the subroutines TRANS, TRANSSZ and BUFOUT. If LOC2 - LOC1 is greater than PAGEBUF size, GETPAGE is used to get the next page.

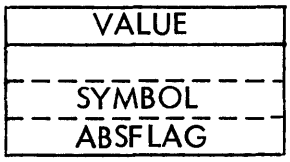
Entered From: SCANNER
Exit To: SCANNER

MASK Command: SMASK VAL

The routine MASK sets the specified VAL into MASQ via a bal to GETHEX.

Entered From: SCANNER
Exit To: SCANNER

SYMBOLMAP The routine SYMBOLMAP reads the monitor's REF/DEF stack, forms the DEFs into 4-word entries of value and symbol,



An initial 8K buffer is obtained and MONSTK is read into it. If there is lost data on the read, the buffer size is increased by one page until MONSTK will fit into it.

Each entry is tested for a length of 4 or greater and to see if it is a DEF. If not, the next entry is tested. If so, a 4-word image is created as follows: symbol value, symbol in TEXTC (7 byte maximum), and a flag indicating whether the symbol is ABS (-1) or not (Ø). The 4-word image is sorted into the symbol table by value, and the process continues until all symbols are in the table.

At this point, an additional table is build and sorted into place that indicates the ALPHA-SORTED position of each entry, and this map is then printed. The SYMBOLS are then listed in a numericaly-sorted display.

TRACE Command: TR[ACE][, VAL][, id]

The TRACE routine dumps the event recorder made by the monitor routine RECORD. The number of events to trace may be specified in the command up to the size of the buffer (32 events). An associated user may also be specified. TRACE establishes the location of the wrap-around buffer RECBUF1, and the location of the last event recorded. It then calculates the location of the first event to dump from the number specified, or if none, the size of the buffer. For each event to be displayed, the subroutine GOGET4 is called to format and output the record.

Entered By: SCANNER and ALL
Returns To: SCANNER

UCLO Command: UC[, value]

The routine UCLO closes M:LO and re-opens it to device UC. "value" is the width of the Hex columns for dump output. The default is 4 columns wide.

Entered By: SCANNER
Exits To: SCANNER

UNMAP Command: UNMAP

The routine UNMAP resets the flag to indicate mapping in progress (MAPFLAG)

Entered From: SCANNER
Exits To: SCANNER

ASSOCIATEDEL Command: DELTA

The ASSOCIATEDEL routine executes the associate CAL (documented in the UTS reference manual 90 09 07B) with an FPT which supplies the entry points of the routines DELTAGET and DELTAPUT via a transfer vector to DELTA, ASSOCIATING DELTA in the process.

Entered From: SCANNER
Exits To: SCANNER

DISASSDEL Command: NO DELTA

The DISASSDEL routine executes the disassociate CAL to disassociate DELTA from ANALYZE.

Entered From: SCANNER
Exits To: SCANNER

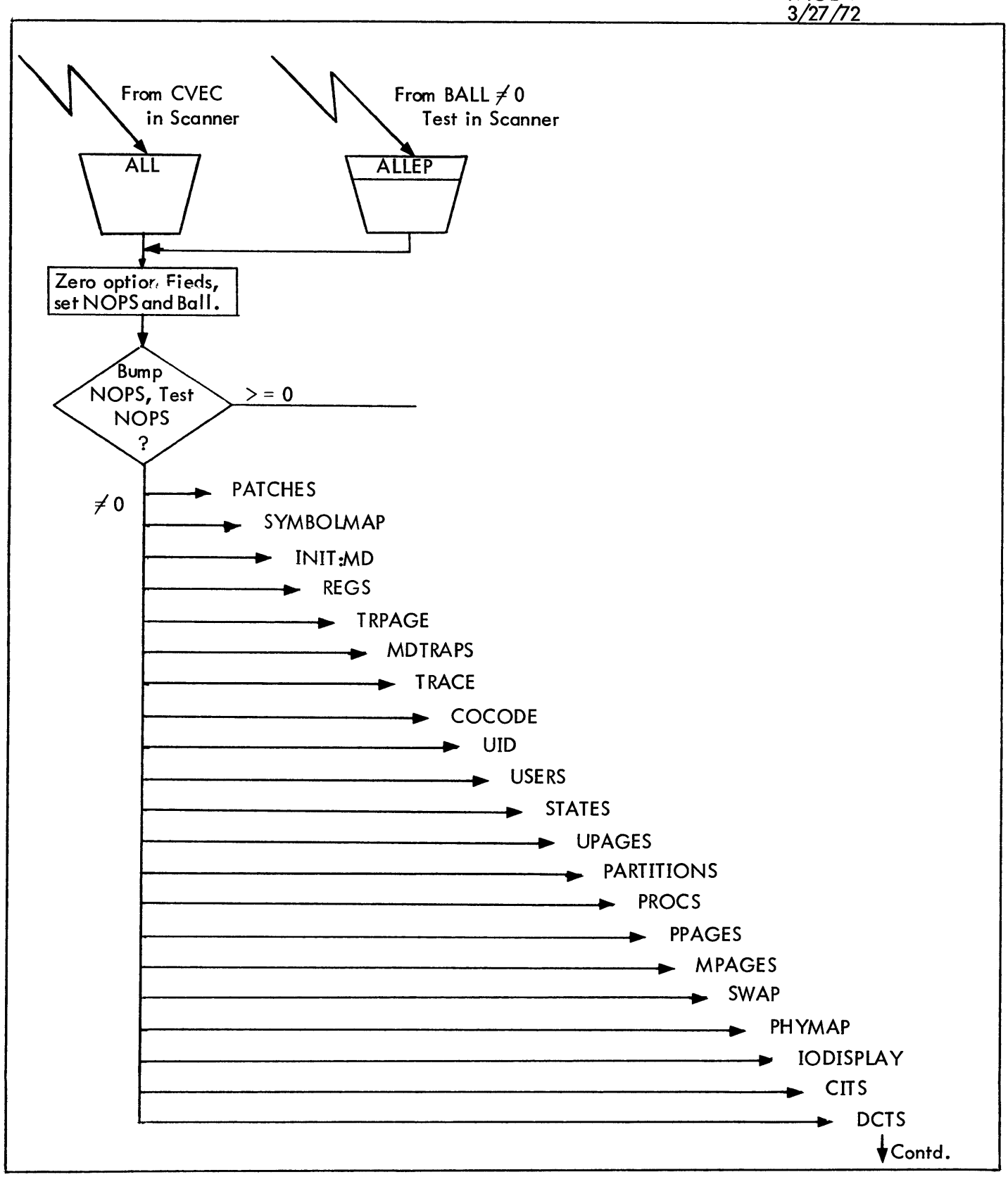
FILENAME Command: BF fid.ACN#

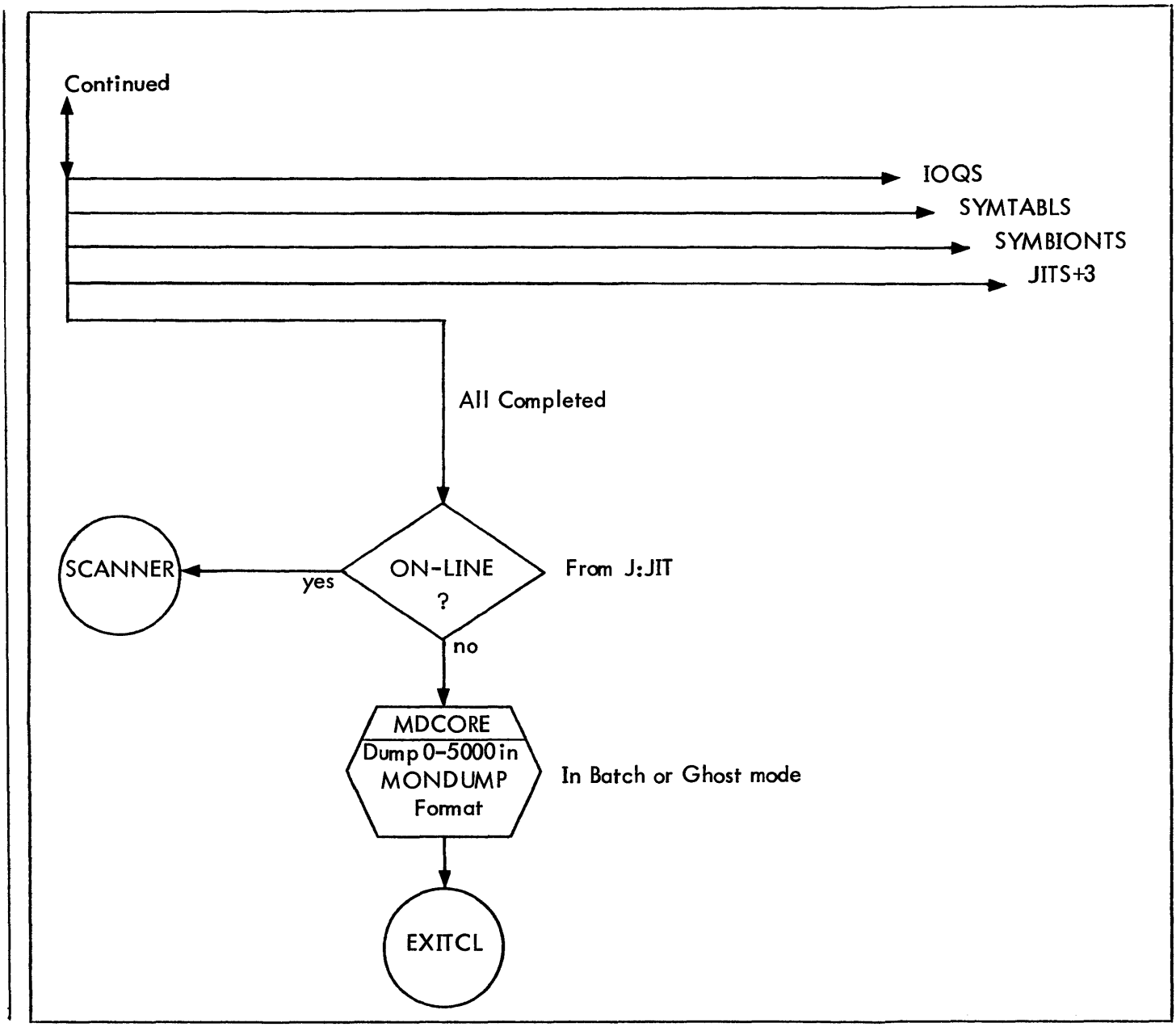
The FILENAME routine scans the FID supplied in the command, and sets it into the file name pointed to by the transfer vector of the associate CAL. The default FID is M:MON.:SYS.

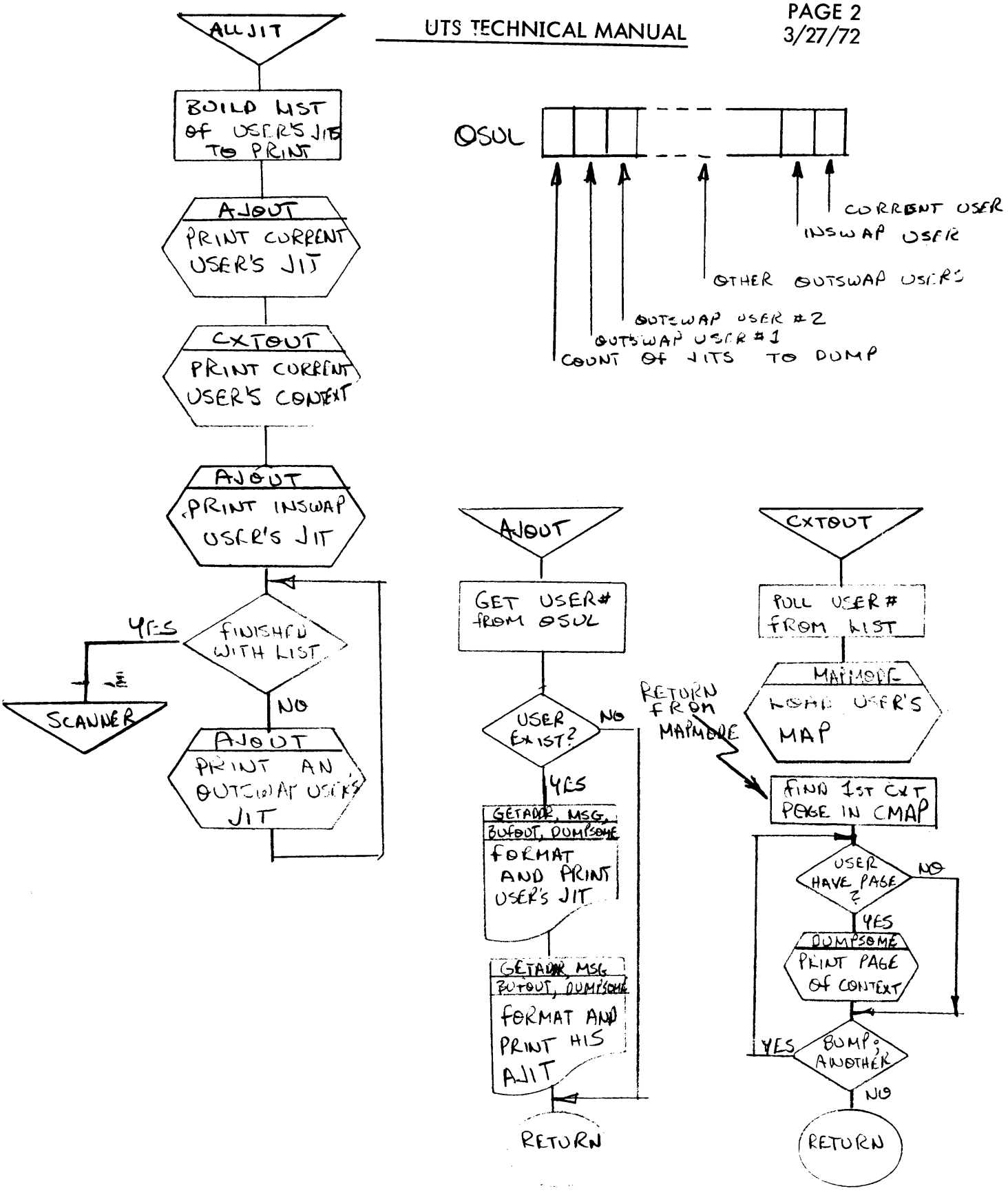
Entered From: SCANNER
Exits To: SCANNER

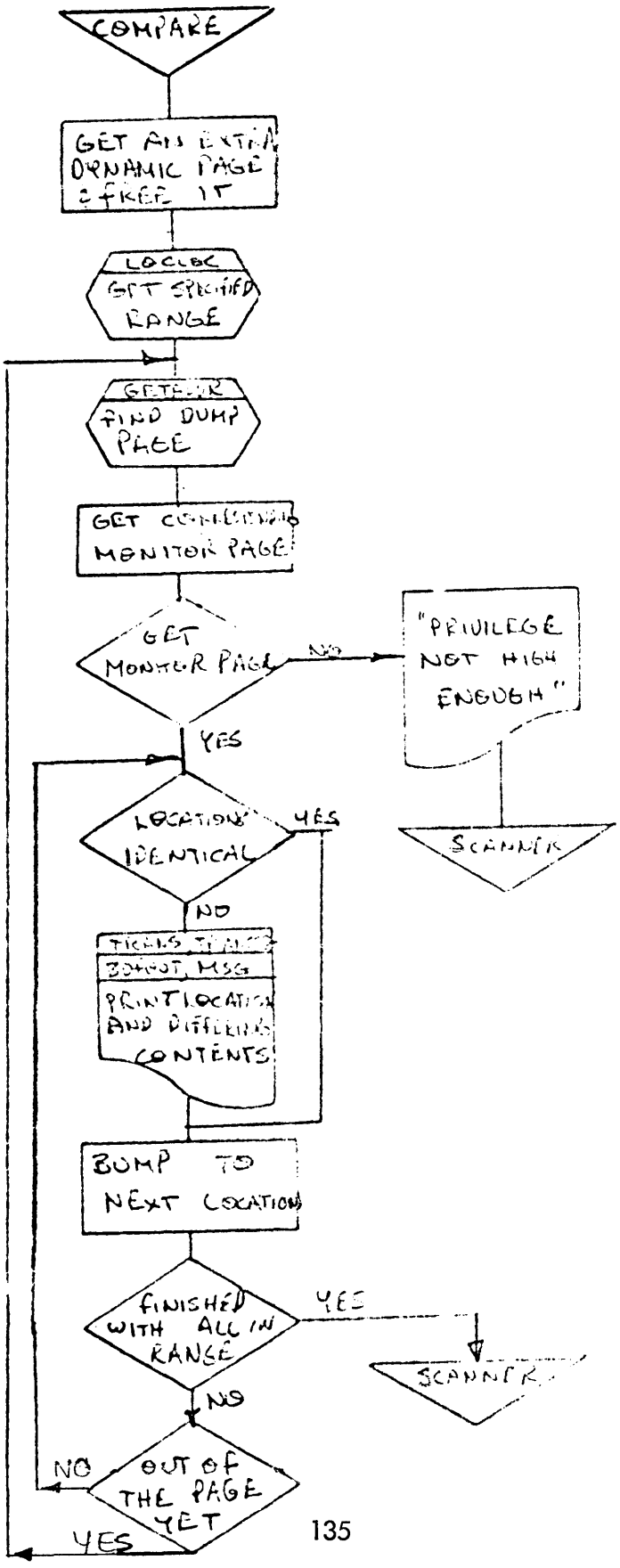
DELTAGET, DELTAPUT

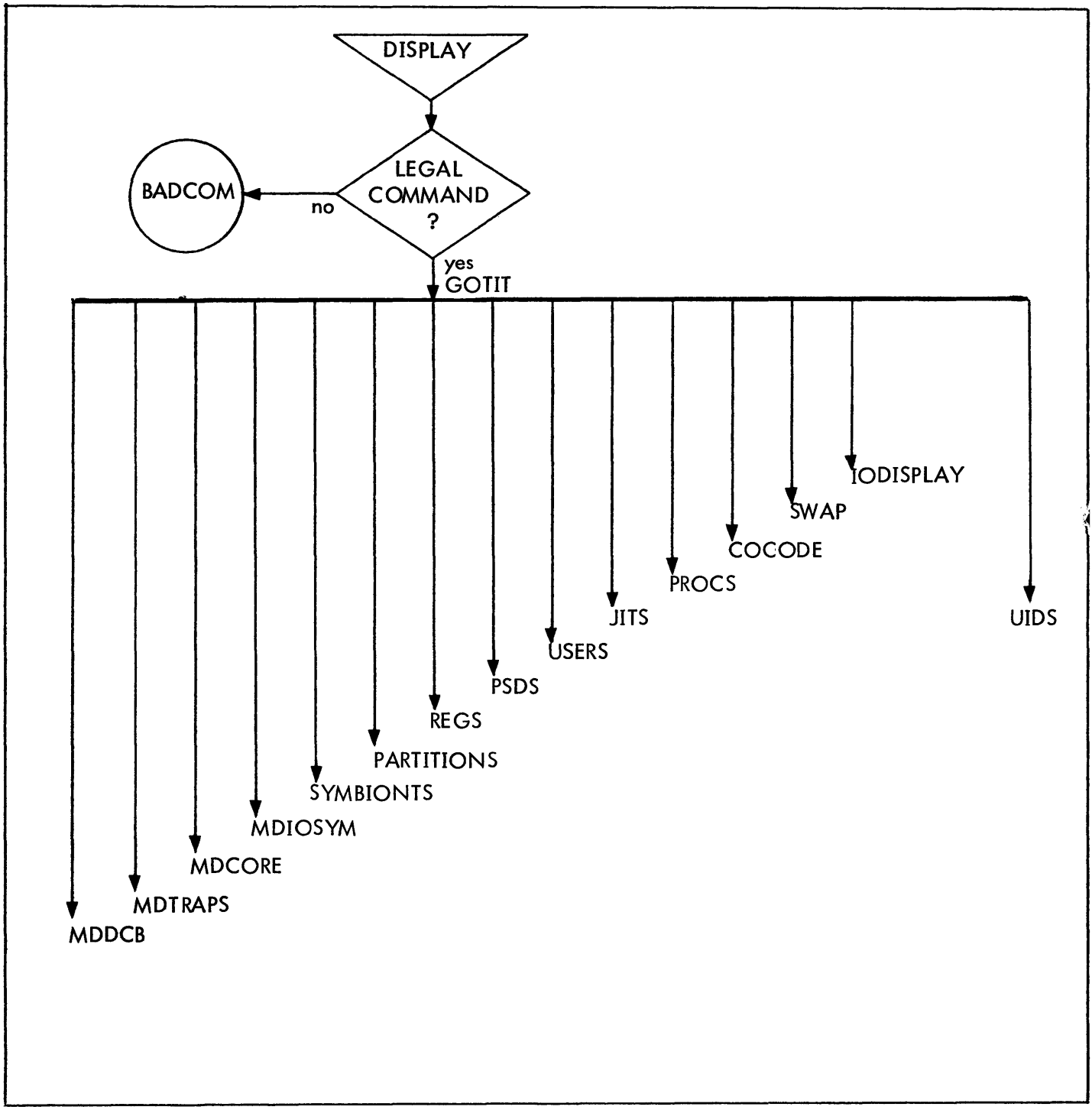
Routines which provide the ANALYZE portion of the ANALYZE/DELTA interface. This routine obtains a cell specified by DELTA in R3 using the ANALYZE routine GETADDR. The contents of that cell are returned to DELTA if DELTAGET was called, otherwise the new value specified by DELTA in R0 is stored into it. For further information, see the ANALYZE/DELTA interface documentation, section LA.

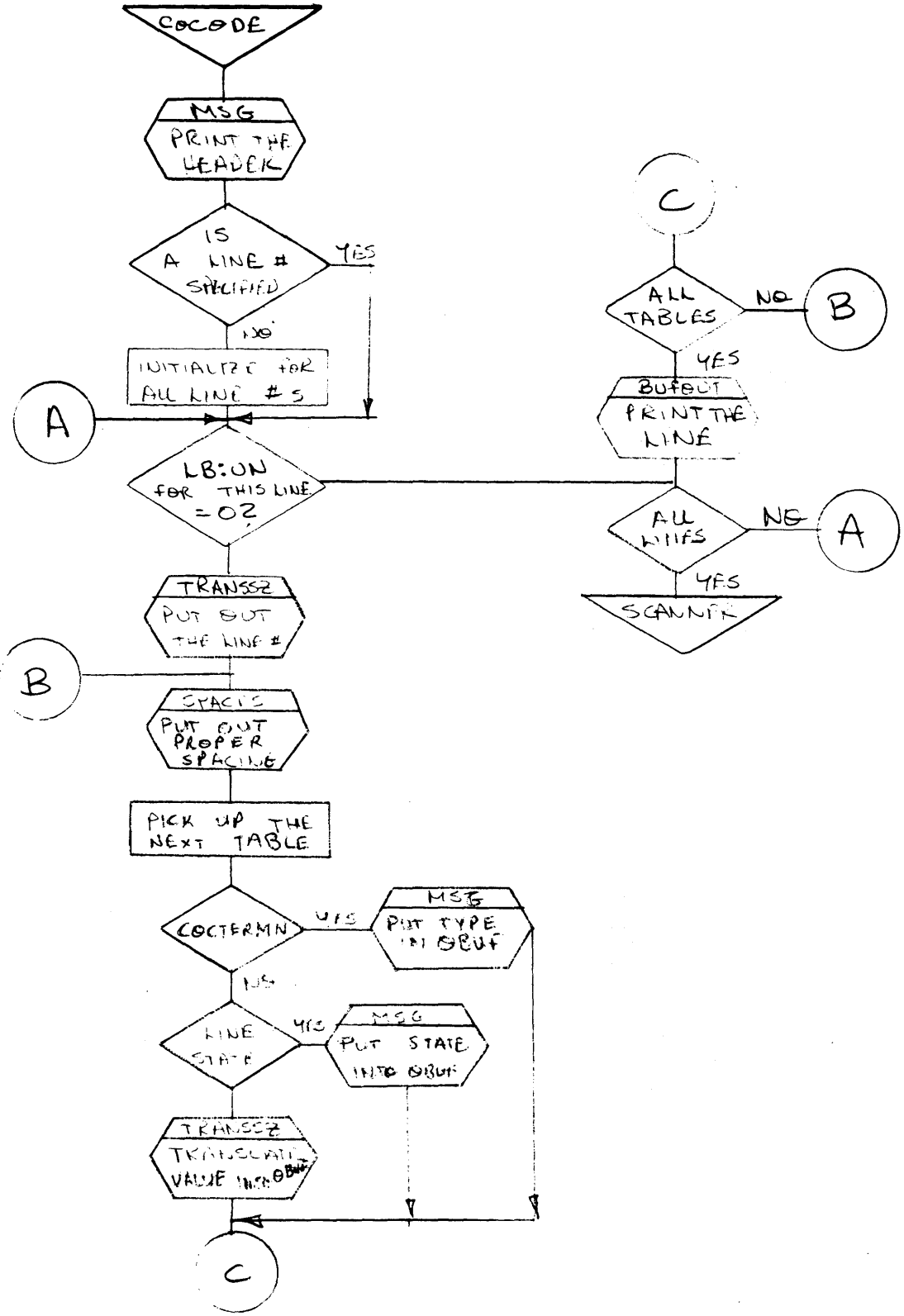

From CVEC in Scanner
From BALL ≠ 0 Test in Scanner
ALL
ALLEP
Zero option Fields, set NOPS and Ball.
Bump NOPS, Test NOPS ?
>= 0
≠ 0
PATCHES
SYMBOLMAP
INIT:MD
REGS
TRPAGE
MDTRAPS
TRACE
COCODE
UID
USERS
STATES
UPAGES
PARTITIONS
PROCS
PPAGES
MPAGES
SWAP
PHYMAP
IODISPLAY
CITS
DCTS
Contd.

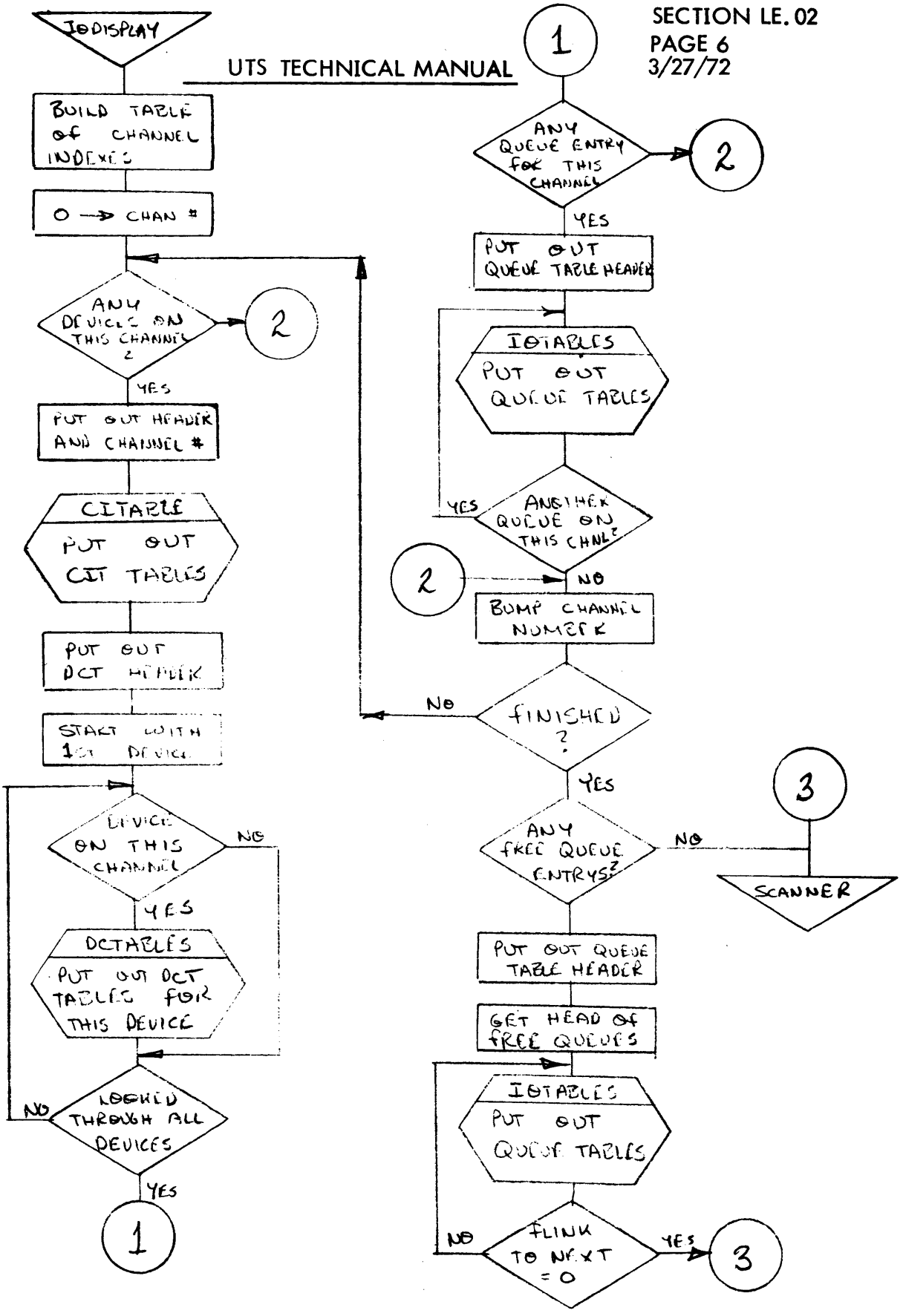

Continued
IOQS
SYMTABLS
SYMBIONTS
JITS+3
All Completed
SCANNER
yes
ON-LINE
?
From J:JIT
no
MDCORE
Dump 0-5000 in
MONDUMP
Format
In Batch or Ghost mode
EXITCL

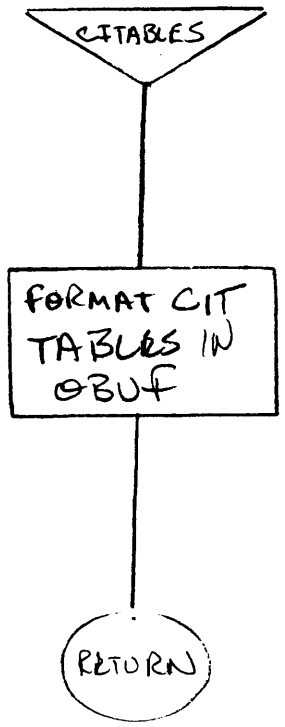

ALLJIT
BUILD LIST OF USER'S JITS TO PRINT
AJOUT
PRINT CURRENT USER'S JIT
CXTOUT
PRINT CURRENT USER'S CONTEXT
AJOUT
PRINT INSWAP USER'S JIT
FINISHED WITH LIST
YES
NO
SCANNER
AJOUT
PRINT AN OUTSWAP USER'S JIT
OSUL
CURRENT USER
INSWAP USER
OTHER OUTSWAP USERS
OUTSWAP USER #2
OUTSWAP USER #1
COUNT OF JITS TO DUMP
AJOUT
GET USER# FROM OSUL
USER EXIST?
NO
YES
GETADR, MSG, BUFOUT, DUMPSOME
FORMAT AND PRINT USER'S JIT
GETADR, MSG, BUFOUT, DUMPSOME
FORMAT AND PRINT HIS AJIT
RETURN
RETURN FROM MAPMORE
CXTOUT
PULL USER# FROM LIST
MAPMORE
MAP USER'S MAP
FIND 1ST CXT PAGE IN CMAP
USER HAVE PAGE?
NO
YES
DUMPSOME
PRINT PAGE OF CONTEXT
BUMP; ANOTHER
YES
NO
RETURN

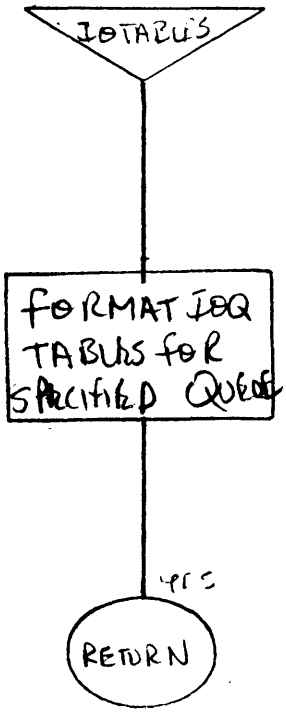

COMPARE
GET AN EXTRA DYNAMIC PAGE & FREE IT
LOCLOC
GET SPECIFIED RANGE
GETEUR
FIND DUMP PAGE
GET CORRESPONDING MONITOR PAGE
GET MONITOR PAGE
NO
"PRIVILEGE NOT HIGH ENOUGH"
SCANNER
YES
LOCATIONS IDENTICAL
YES
NO
TRANS TRANS
BUFOUT, MSG
PRINT LOCATION AND DIFFERING CONTENTS
BUMP TO NEXT LOCATION
FINISHED WITH ALL IN RANGE
YES
SCANNER
NO
OUT OF THE PAGE YET
NO
YES

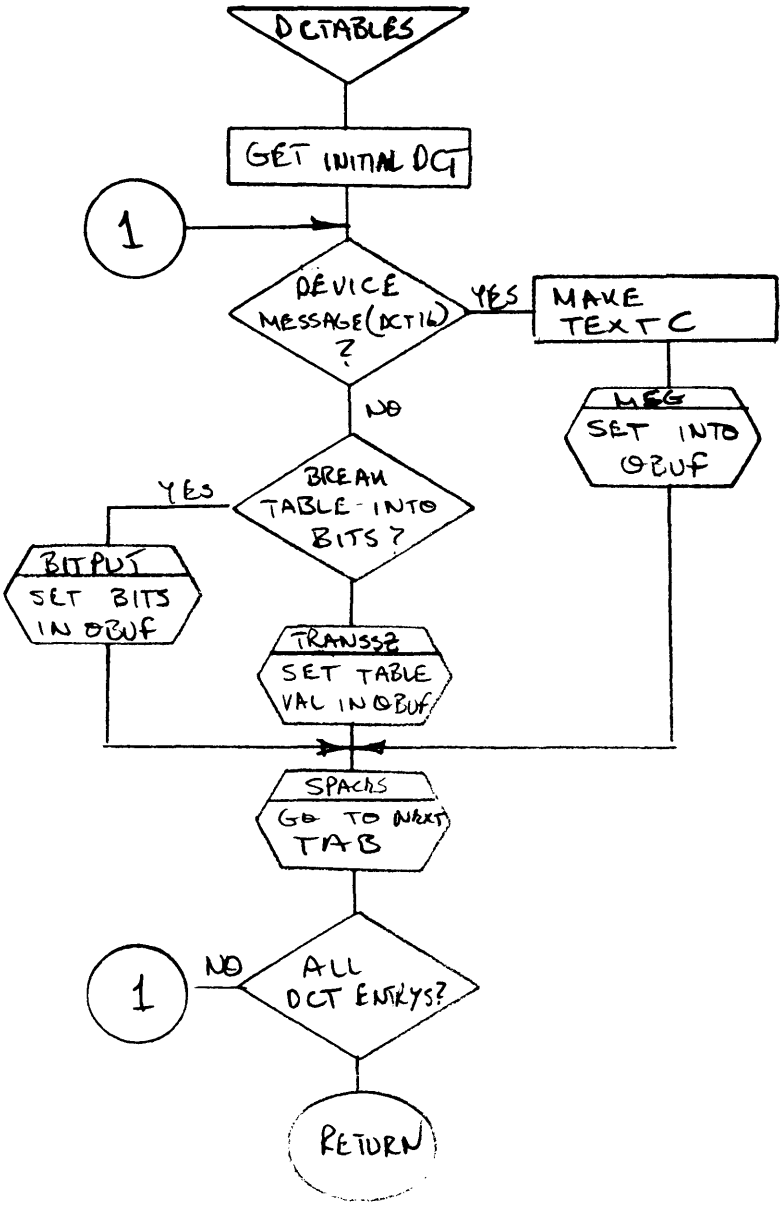

DISPLAY
LEGAL COMMAND ?
no
BADCOM
yes
GOTIT
IODISPLAY
SWAP
COCODE
PROCS
JITS
UIDS
USERS
PSDS
REGS
PARTITIONS
SYMBIONTS
MDIOSYM
MDCORE
MDTRAPS
MDDCB

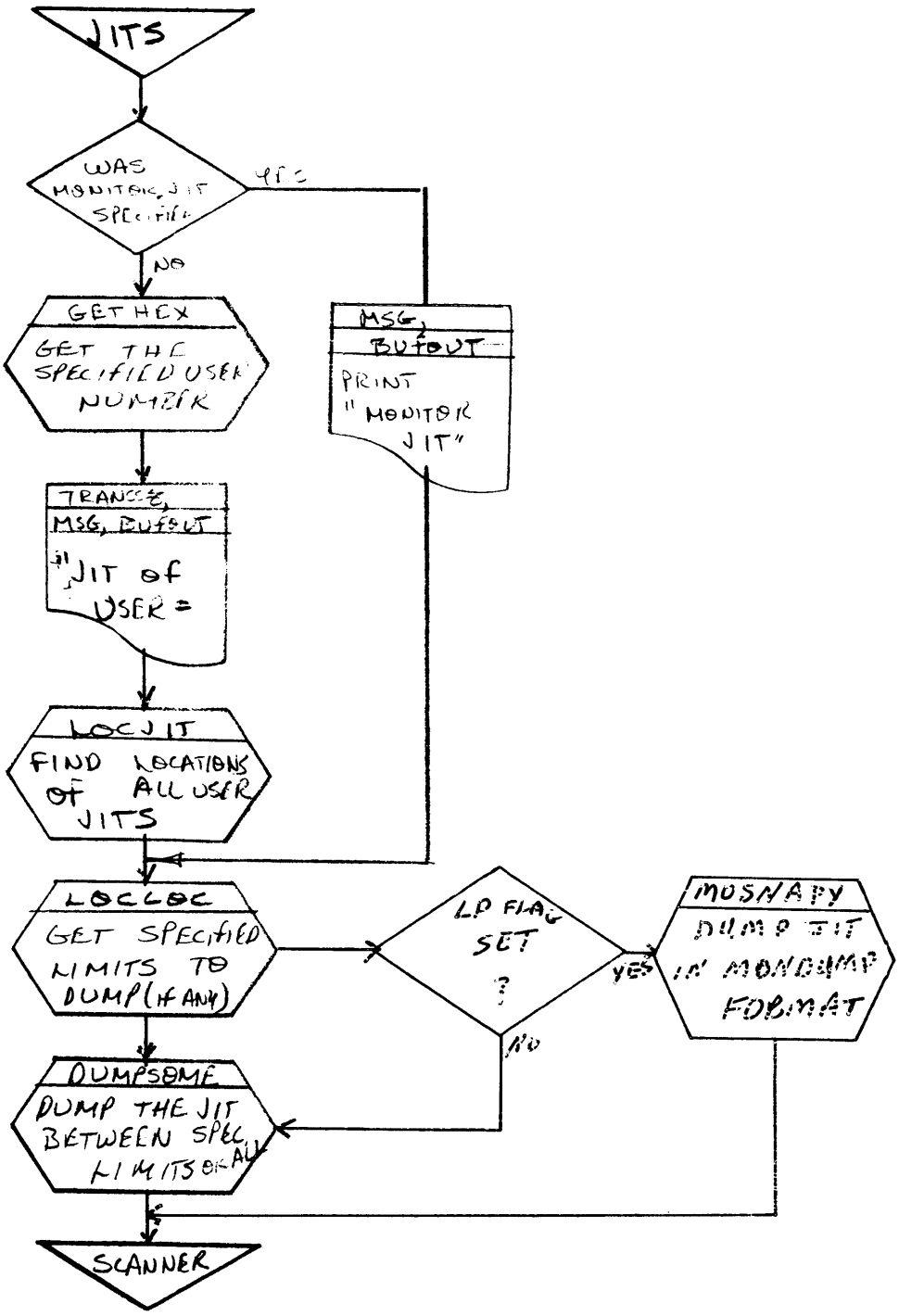

COCODE
MSG
PRINT THE HEADER
IS A LINE # SPECIFIED
YES
NO
INITIALIZE FOR ALL LINE #S
A
LB:ON FOR THIS LINE =02
TRANSSZ
PUT OUT THE LINE #
B
SPACES
PUT OUT PROPER SPACING
PICK UP THE NEXT TABLE
COCTERMN
UTS
MSG
PUT TYPE IN OBUF
NO
LINE STATE
YES
MSG
PUT STATE INTO OBUF
TRANSSZ
TRANSLATE VALUE INTO OBUF
C
C
ALL TABLES
NO
B
YES
BUFOUT
PRINT THE LINE
ALL LINES
NO
A
YES
SCANNER

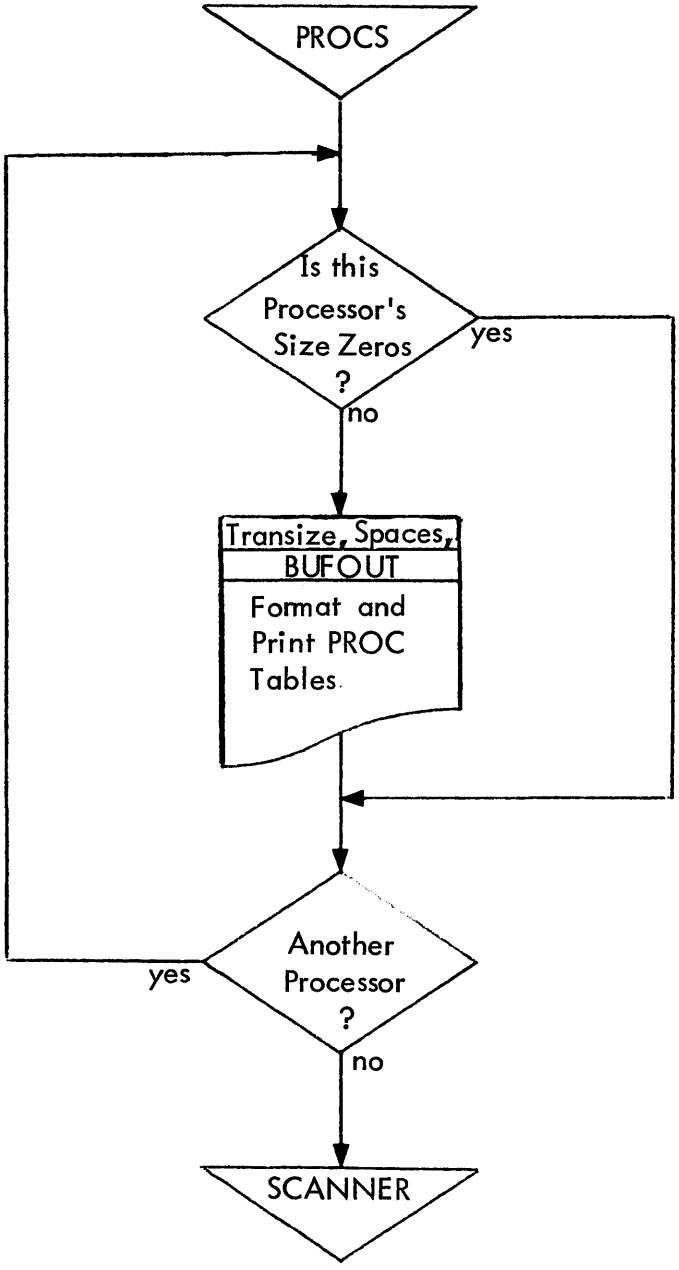

IODISPLAY
BUILD TABLE OF CHANNEL INDEXES
0 → CHAN #
ANY DEVICES ON THIS CHANNEL ?
2
YES
PUT OUT HEADER AND CHANNEL #
CITABLE
PUT OUT CIT TABLES
PUT OUT DCT HEADER
START WITH 1ST DEVICE
DEVICE ON THIS CHANNEL
NO
YES
DCTABLES
PUT OUT DCT TABLES FOR THIS DEVICE
LOOKED THROUGH ALL DEVICES
NO
YES
1
1
ANY QUEUE ENTRY FOR THIS CHANNEL
2
YES
PUT OUT QUEUE TABLE HEADER
IOTABLES
PUT OUT QUEUE TABLES
ANOTHER QUEUE ON THIS CHNL?
YES
2
NO
BUMP CHANNEL NUMBER
FINISHED ?
NO
YES
ANY FREE QUEUE ENTRYS?
NO
3
SCANNER
PUT OUT QUEUE TABLE HEADER
GET HEAD OF FREE QUEUES
IOTABLES
PUT OUT QUEUE TABLES
FLINK TO NEXT = 0
NO
YES
3

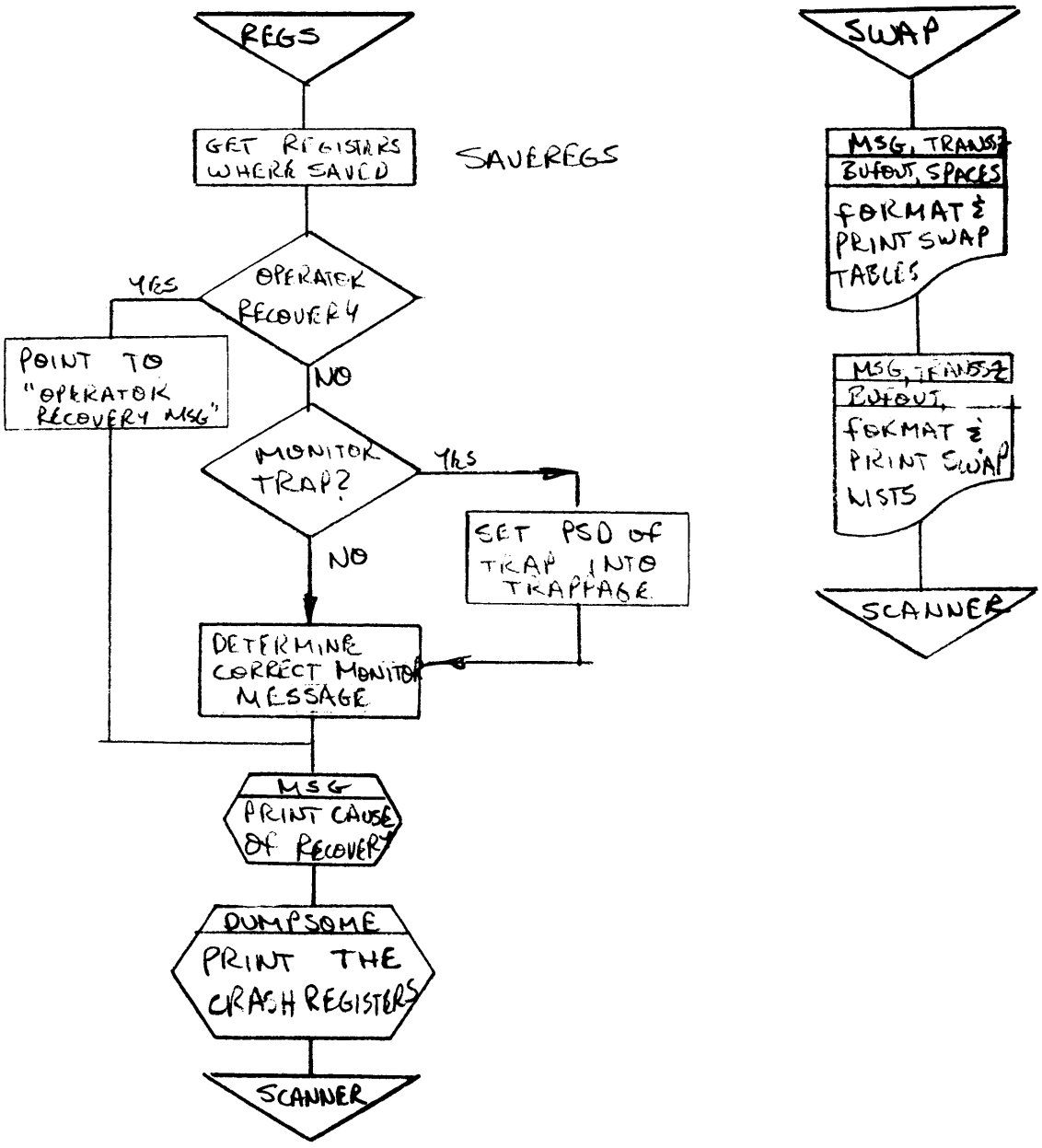

CITABLES
FORMAT CIT TABLES IN OBUF
RETURN

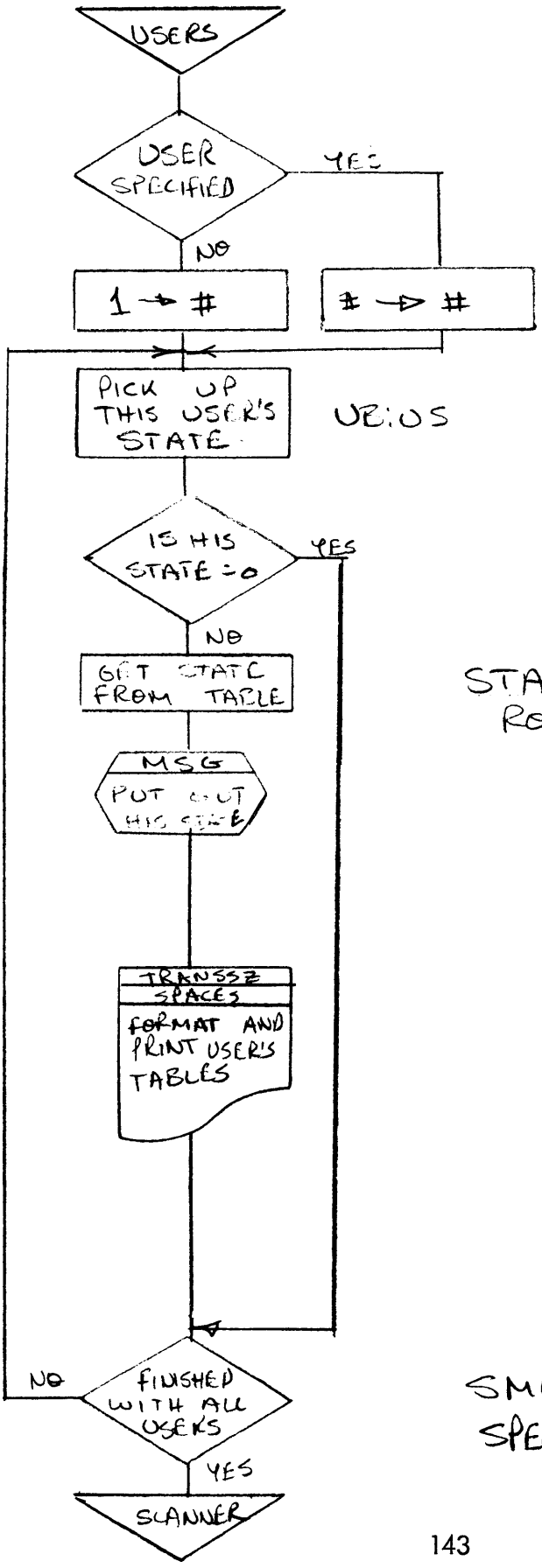

IOTABLES
FORMAT IOQ TABLES FOR SPECIFIED QUEUE
RETURN

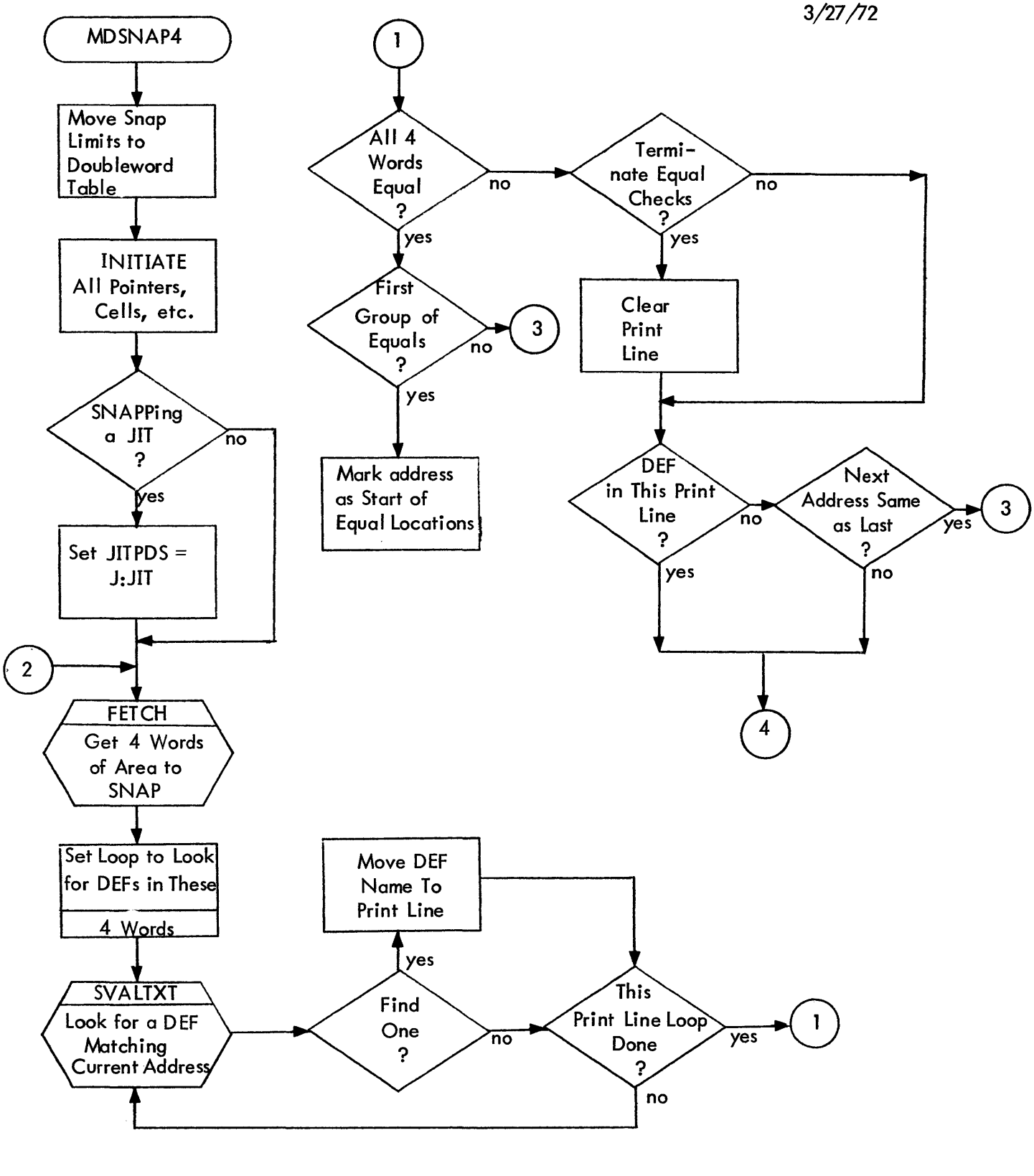

DCTABLES
GET INITIAL DCT
1
DEVICE MESSAGE(DCT16) ?
YES
MAKE TEXT C
MSG
SET INTO OBUF
NO
BREAK TABLE INTO BITS ?
YES
BITPUT
SET BITS IN OBUF
TRANSSS
SET TABLE VAL IN OBUF
SPACES
GO TO NEXT TAB
1
NO
ALL DCT ENTRYS?
RETURN

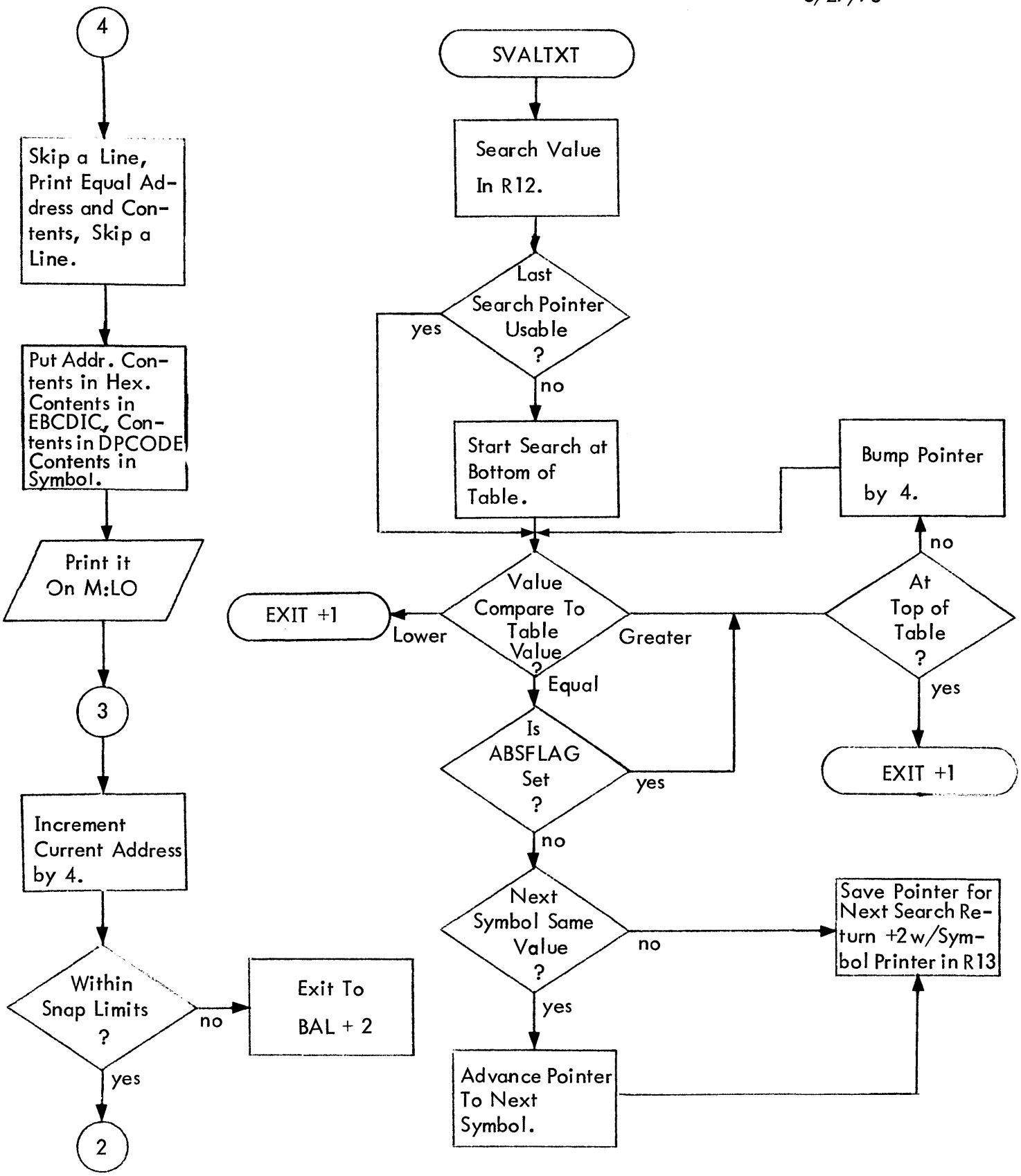

JITS
WAS MONITOR JIT SPECIFIED
YES
NO
GET HEX
GET THE SPECIFIED USER NUMBER
MSG, BUTOUT
PRINT "MONITOR JIT"
TRANSF MSG, BUTOUT
"JIT of USER =
LOCJIT
FIND LOCATIONS OF ALL USER JITS
LOCLOC
GET SPECIFIED LIMITS TO DUMP (IF ANY)
LP FLAG SET ?
YES
NO
MOSNAPY
DUMP JIT IN MONDUMP FORMAT
DUMPSOME
DUMP THE JIT BETWEEN SPEC. LIMITS OR ALL
SCANNER

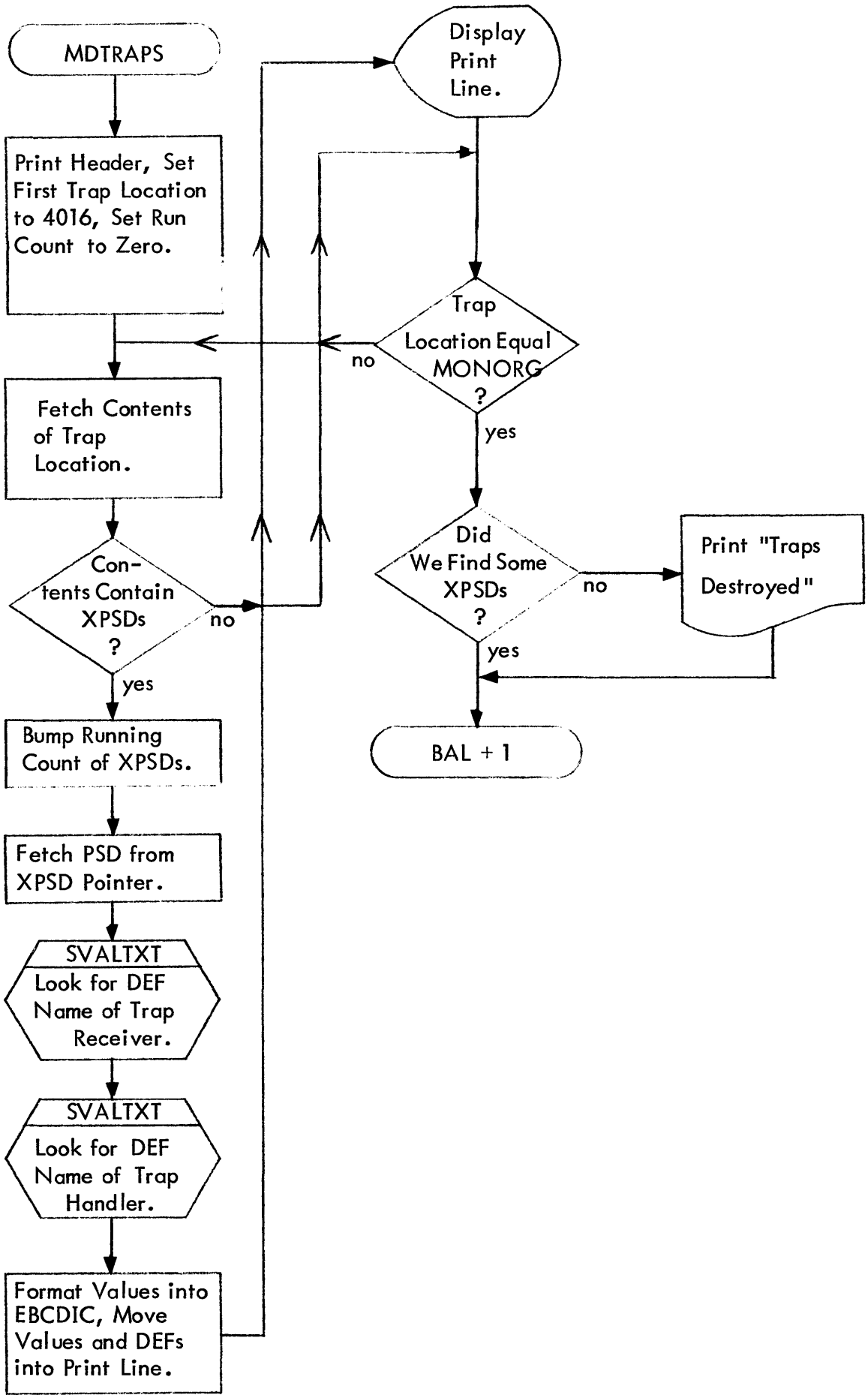

PROCS
Is this Processor's Size Zeros ?
yes
no
Transize, Spaces, BUFOUT
Format and Print PROC Tables
Another Processor ?
yes
no
SCANNER

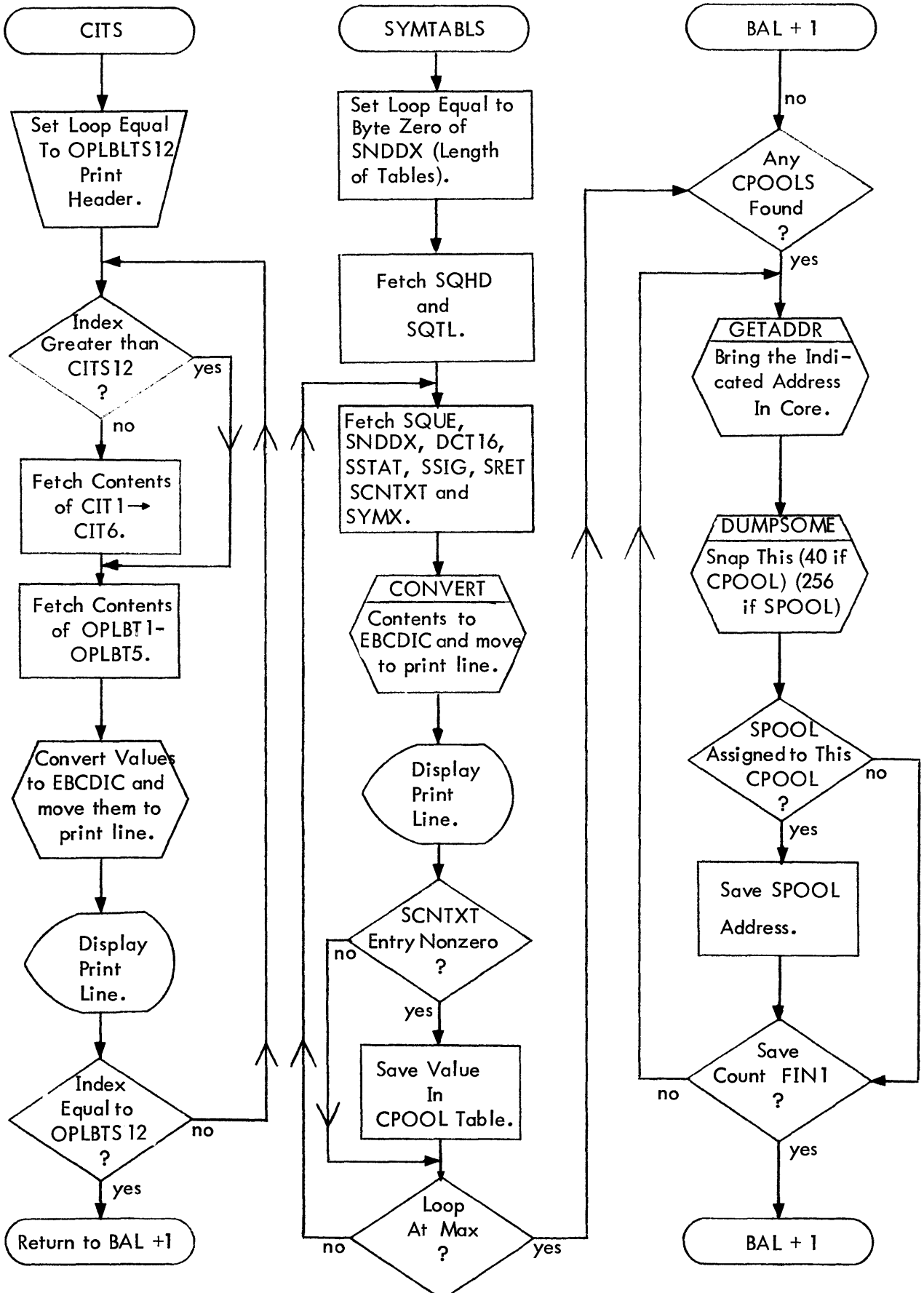

REGS
GET REGISTERS WHERE SAVED
SAVEREGS
OPERATOR RECOVERY
YES
NO
POINT TO "OPERATOR RECOVERY MSG"
MONITOR TRAP?
YES
NO
SET PSD of TRAP INTO TRAPPAGE
DETERMINE CORRECT MONITOR MESSAGE
MSG
PRINT CAUSE OF RECOVERY
DUMPSOME
PRINT THE CRASH REGISTERS
SCANNER
SWAP
MSG, TRANSF
BUFOUT, SPACES
FORMAT & PRINT SWAP TABLES
MSG, TRANSF
BUFOUT,
FORMAT & PRINT SWAP LISTS
SCANNER

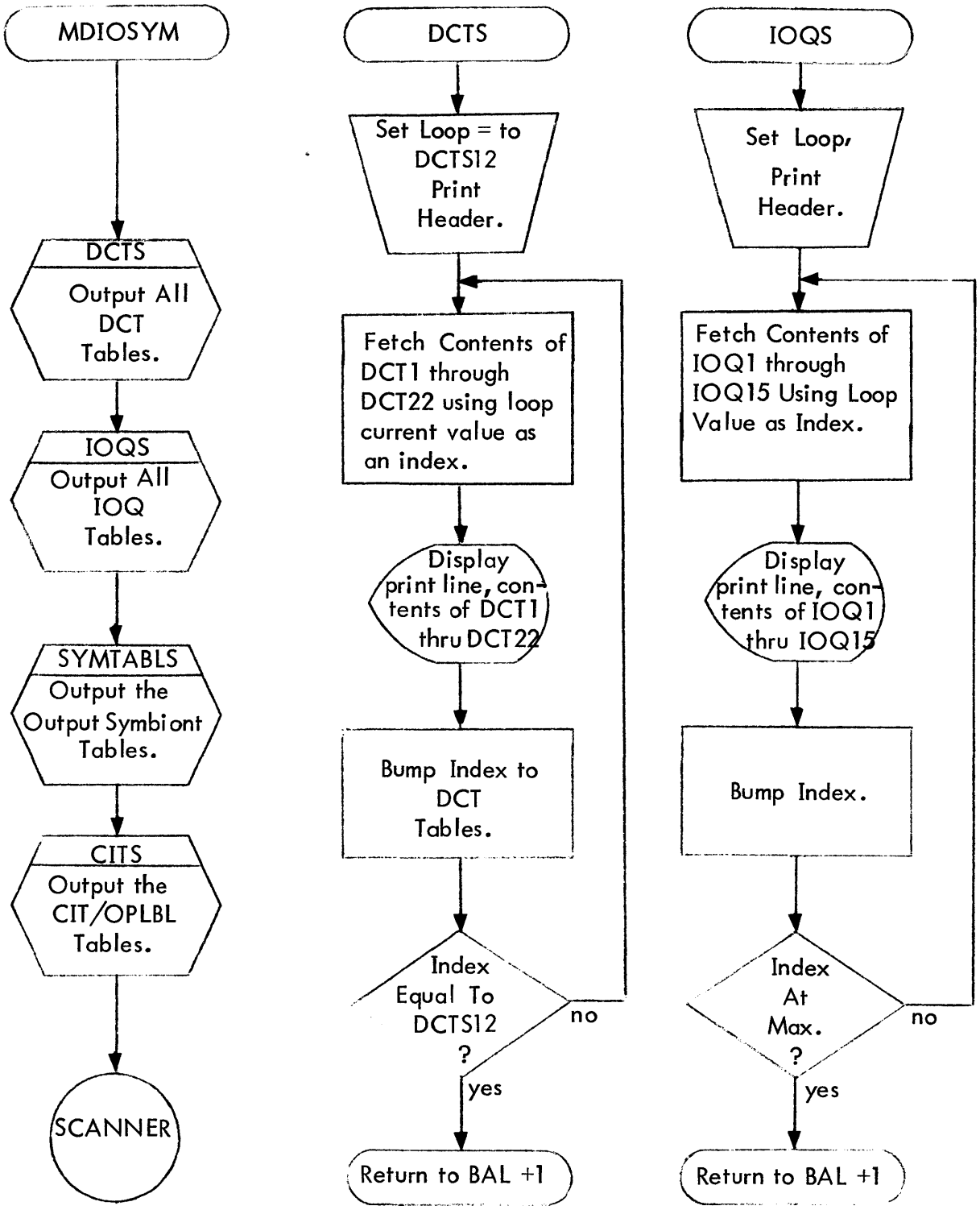

USERS
USER SPECIFIED
YES
NO
1 → #
→
TO DISPLAY FIRST
PICK UP THIS USER'S STATE
UBUS
IS HIS STATE = 0
YES
NO
GET STATE FROM TABLE
STATE TABLE IN TRACE ROUTINE
MSG
PUT OUT HIS STATE
TRANSSZ SPACES
FORMAT AND PRINT USER'S TABLES
NO
FINISHED WITH ALL USERS
SMUIS if # NOT SPECIFIED
YES
SCANNER

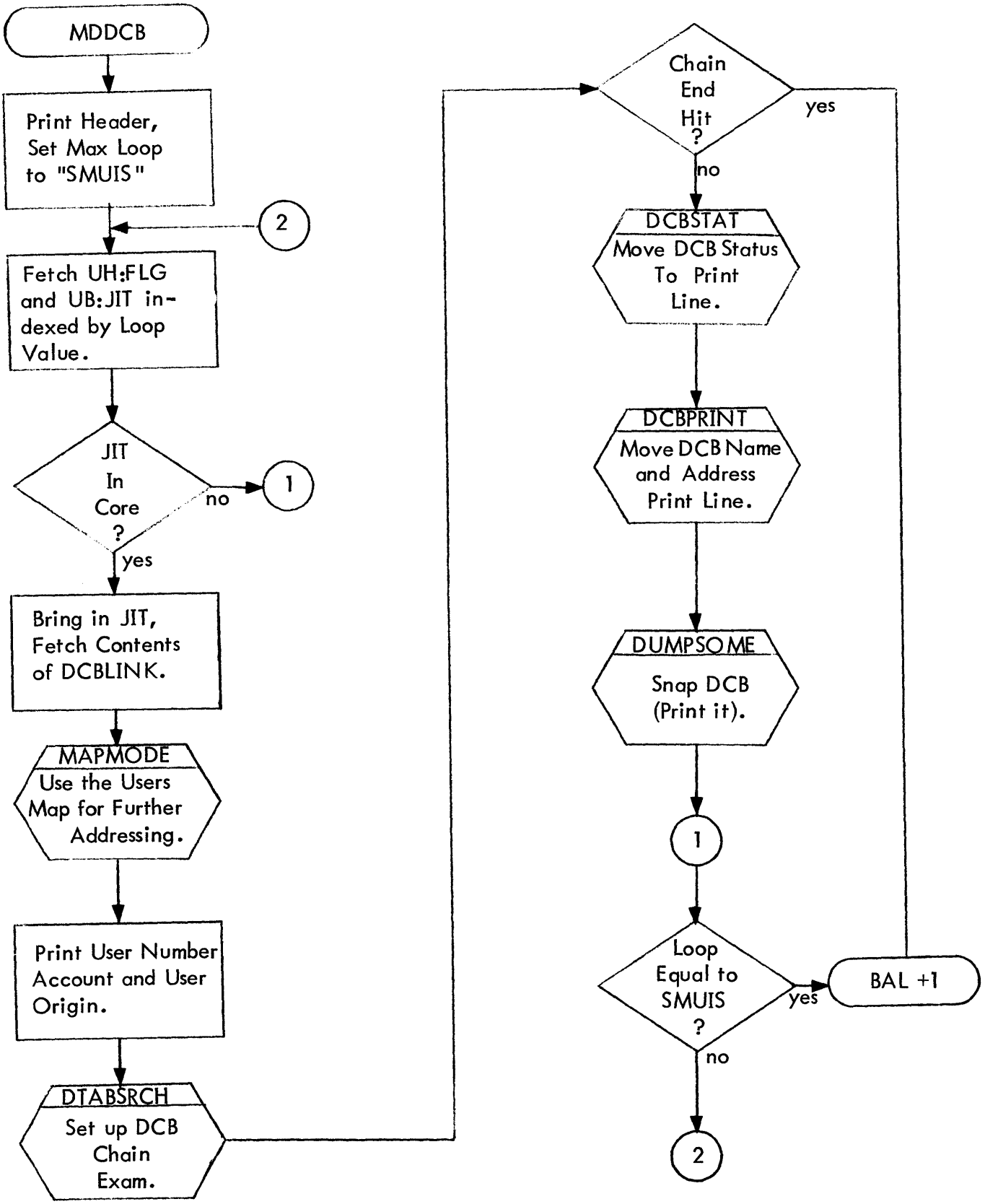

MDSNAP4
Move Snap Limits to Doubleword Table
INITIATE All Pointers, Cells, etc.
SNAPPing a JIT ?
no
yes
Set JITPDS = J:JIT
2
FETCH
Get 4 Words of Area to SNAP
Set Loop to Look for DEFs in These 4 Words
SVALTXT
Look for a DEF Matching Current Address
Find One ?
yes
no
Move DEF Name To Print Line
This Print Line Loop Done ?
yes
no
1
1
All 4 Words Equal ?
no
yes
First Group of Equals ?
no
3
yes
Mark address as Start of Equal Locations
Termi-nate Equal Checks ?
no
yes
Clear Print Line
DEF in This Print Line ?
no
yes
Next Address Same as Last ?
yes
3
no
4

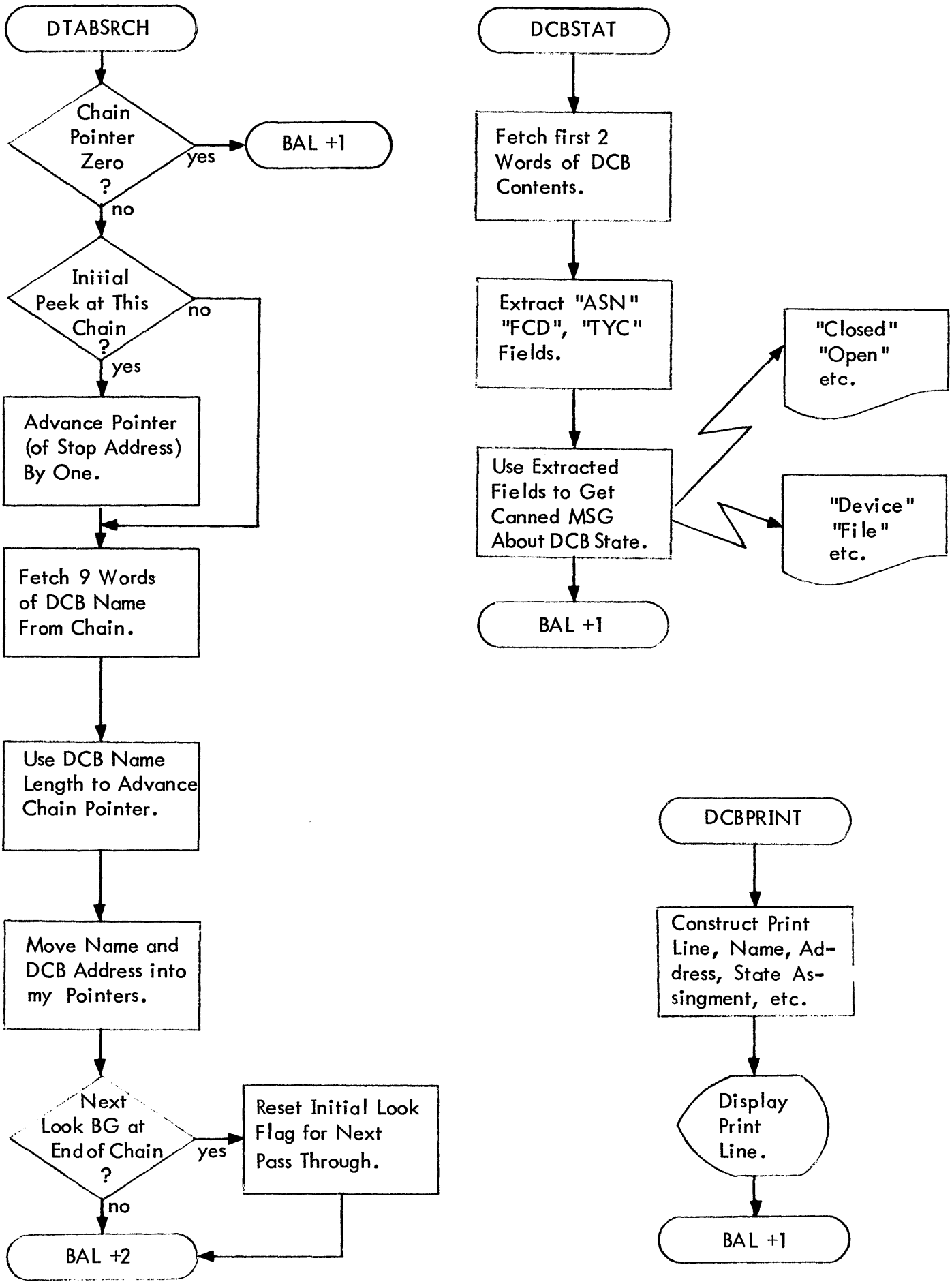

4
Skip a Line, Print Equal Address and Contents, Skip a Line.
Put Addr. Contents in Hex. Contents in EBCDIC, Contents in DPCODE Contents in Symbol.
Print it On M:LO
3
Increment Current Address by 4.
Within Snap Limits ?
no
Exit To BAL + 2
yes
2
SVALTXT
Search Value In R12.
Last Search Pointer Usable ?
yes
no
Start Search at Bottom of Table.
Value Compare To Table Value ?
Lower
EXIT +1
Greater
Equal
Is ABSFLAG Set ?
yes
no
Next Symbol Same Value ?
no
Save Pointer for Next Search Return +2 w/Symbol Printer in R13
yes
Advance Pointer To Next Symbol.
Bump Pointer by 4.
no
At Top of Table ?
yes
EXIT +1

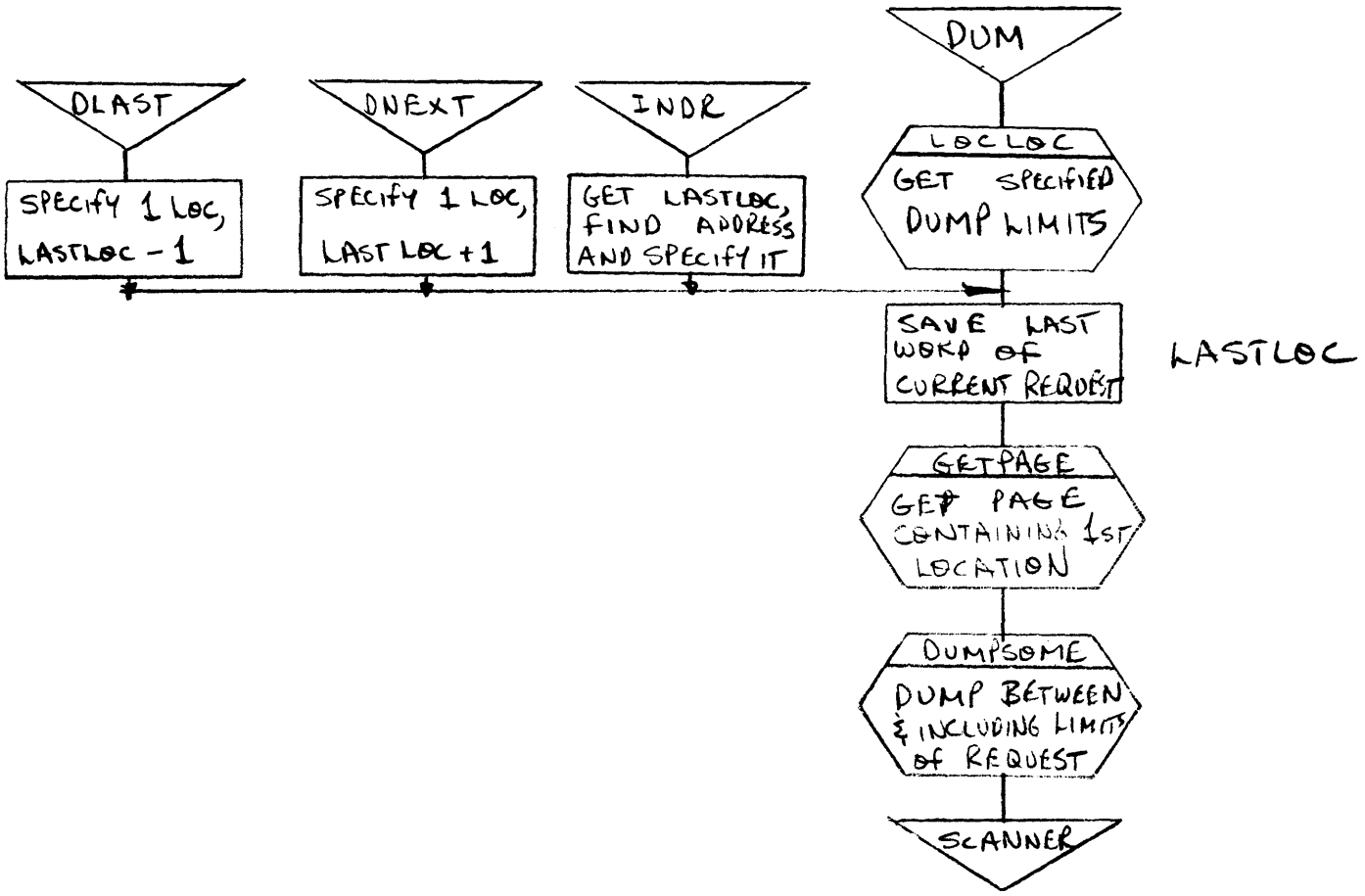

MDTRAPS
Print Header, Set First Trap Location to 4016, Set Run Count to Zero.
Fetch Contents of Trap Location.
Con-tents Contain XPSDs ?
no
yes
Bump Running Count of XPSDs.
Fetch PSD from XPSD Pointer.
SVALTXT
Look for DEF Name of Trap Receiver.
SVALTXT
Look for DEF Name of Trap Handler.
Format Values into EBCDIC, Move Values and DEFs into Print Line.
Display Print Line.
Trap Location Equal MONORG ?
no
yes
Did We Find Some XPSDs ?
no
yes
Print "Traps Destroyed"
BAL + 1

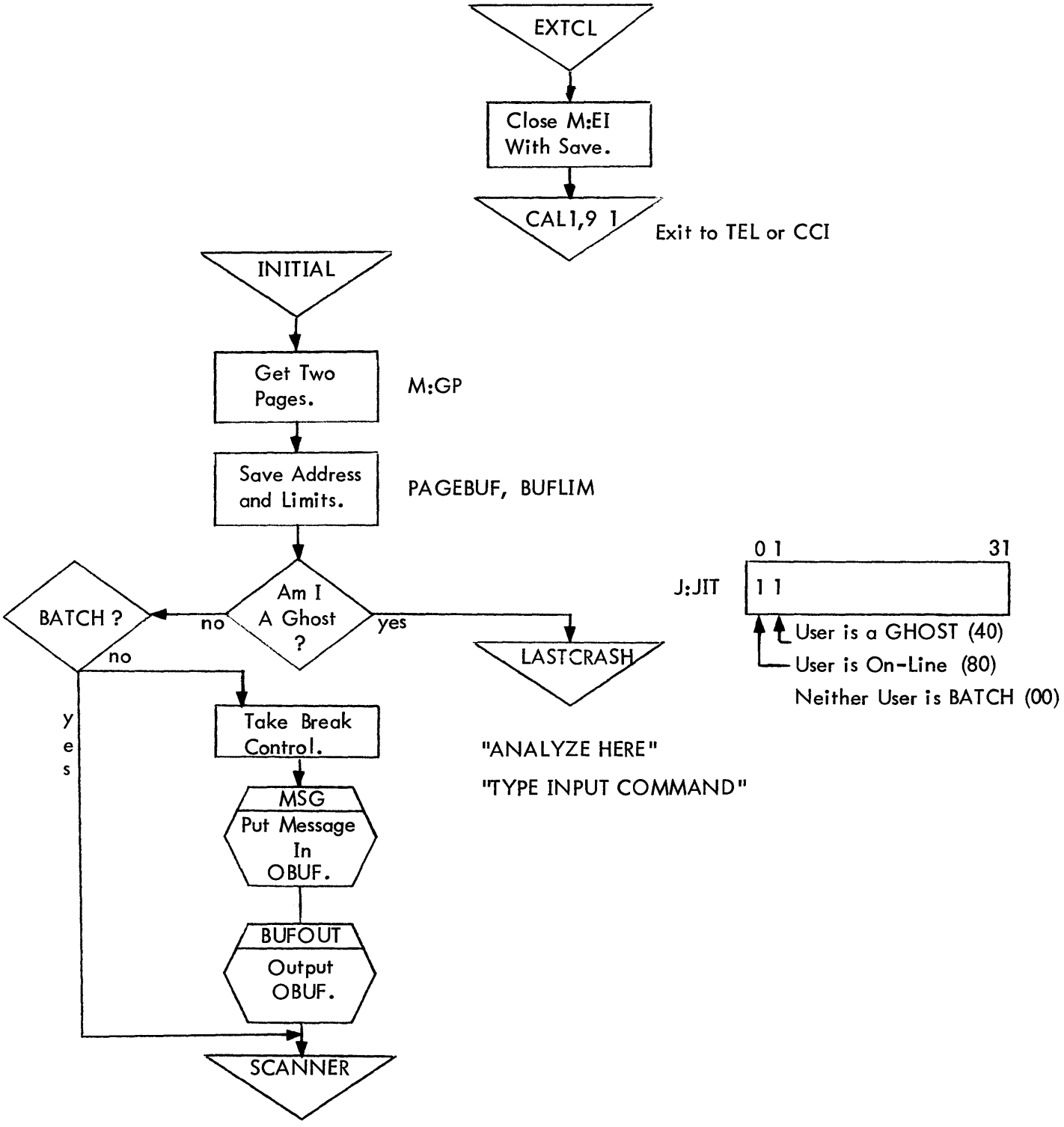

CITS
Set Loop Equal To OPLBLTS12 Print Header.
Index Greater than CITS12 ?
yes
no
Fetch Contents of CIT1→ CIT6.
Fetch Contents of OPLBT1- OPLBT5.
Convert Values to EBCDIC and move them to print line.
Display Print Line.
Index Equal to OPLBTS 12 ?
no
yes
Return to BAL +1
SYMTABLS
Set Loop Equal to Byte Zero of SNDDX (Length of Tables).
Fetch SQHD and SQTL.
Fetch SQUE, SNDDX, DCT16, SSTAT, SSIG, SRET SCNTXT and SYMX.
CONVERT
Contents to EBCDIC and move to print line.
Display Print Line.
SCNTXT Entry Nonzero ?
no
yes
Save Value In CPOOL Table.
Loop At Max ?
no
yes
BAL + 1
no
Any CPOOLS Found ?
yes
GETADDR
Bring the Indicated Address In Core.
DUMPSOME
Snap This (40 if CPOOL) (256 if SPOOL)
SPOOL Assigned to This CPOOL ?
no
yes
Save SPOOL Address.
Save Count FIN1 ?
no
yes
BAL + 1

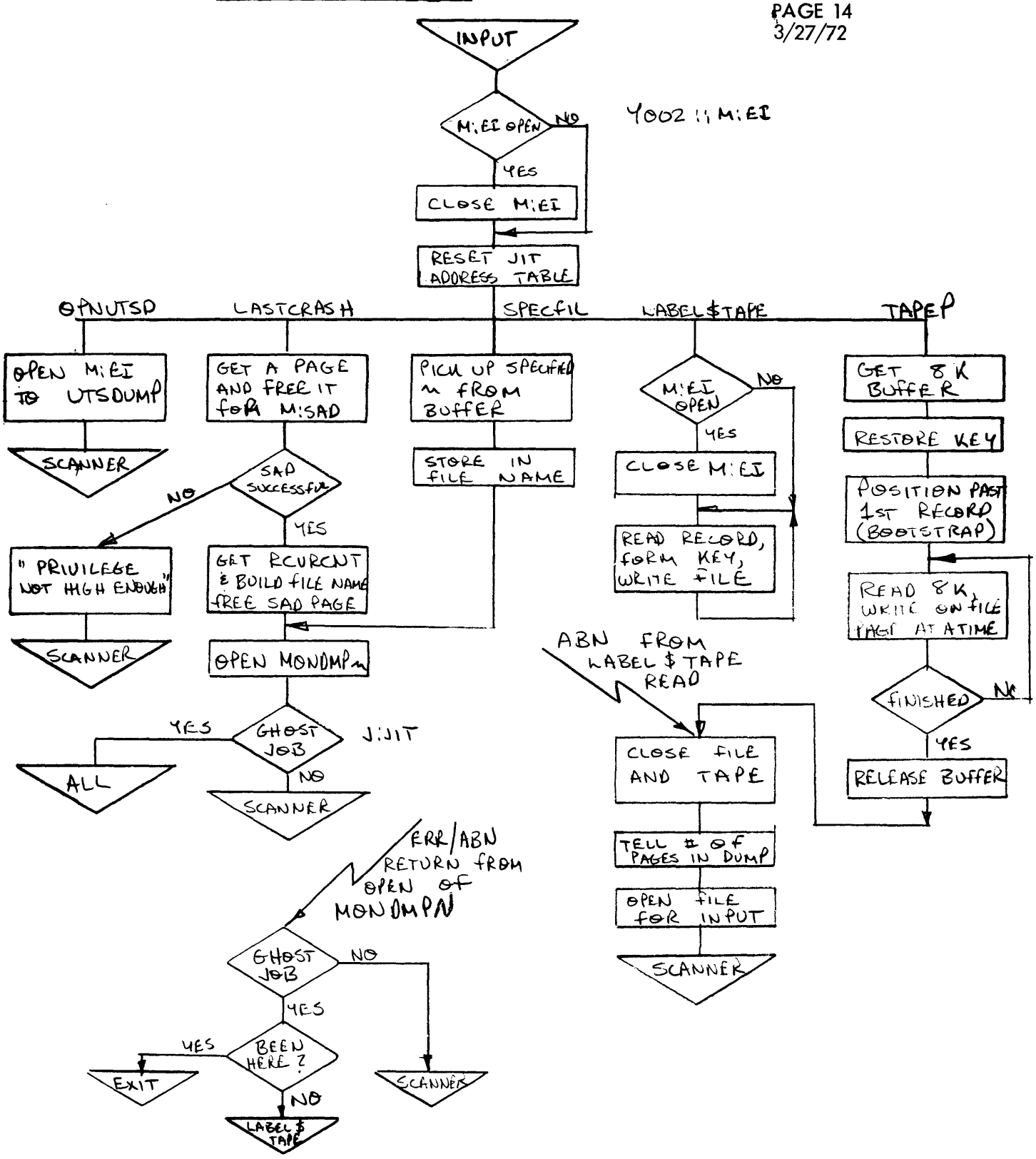

MDIOSYM
DCTS
Output All DCT Tables.
IOQS
Output All IOQ Tables.
SYMTABLS
Output the Output Symbiont Tables.
CITS
Output the CIT/OPLBL Tables.
SCANNER
DCTS
Set Loop = to DCTS12 Print Header.
Fetch Contents of DCT1 through DCT22 using loop current value as an index.
Display print line, contents of DCT1 thru DCT22
Bump Index to DCT Tables.
Index Equal To DCTS12 ?
no
yes
Return to BAL +1
IOQS
Set Loop, Print Header.
Fetch Contents of IOQ1 through IOQ15 Using Loop Value as Index.
Display print line, contents of IOQ1 thru IOQ15
Bump Index.
Index At Max. ?
no
yes
Return to BAL +1

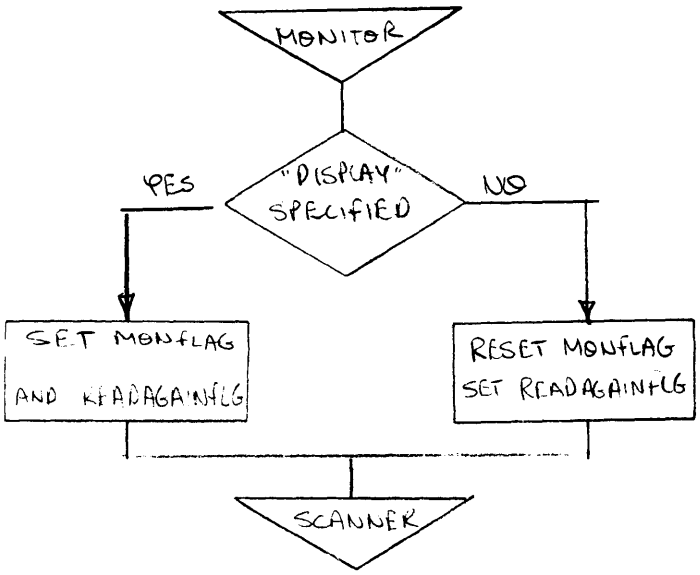

MDDCB
Print Header, Set Max Loop to "SMUIS"
2
Fetch UH:FLG and UB:JIT indexed by Loop Value.
JIT In Core ?
no
1
yes
Bring in JIT, Fetch Contents of DCBLINK.
MAPMODE
Use the Users Map for Further Addressing.
Print User Number Account and User Origin.
DTABSRCH
Set up DCB Chain Exam.
Chain End Hit ?
yes
no
DCBSTAT
Move DCB Status To Print Line.
DCBPRINT
Move DCB Name and Address Print Line.
DUMPSOME
Snap DCB (Print it).
1
Loop Equal to SMUIS ?
yes
BAL +1
no
2

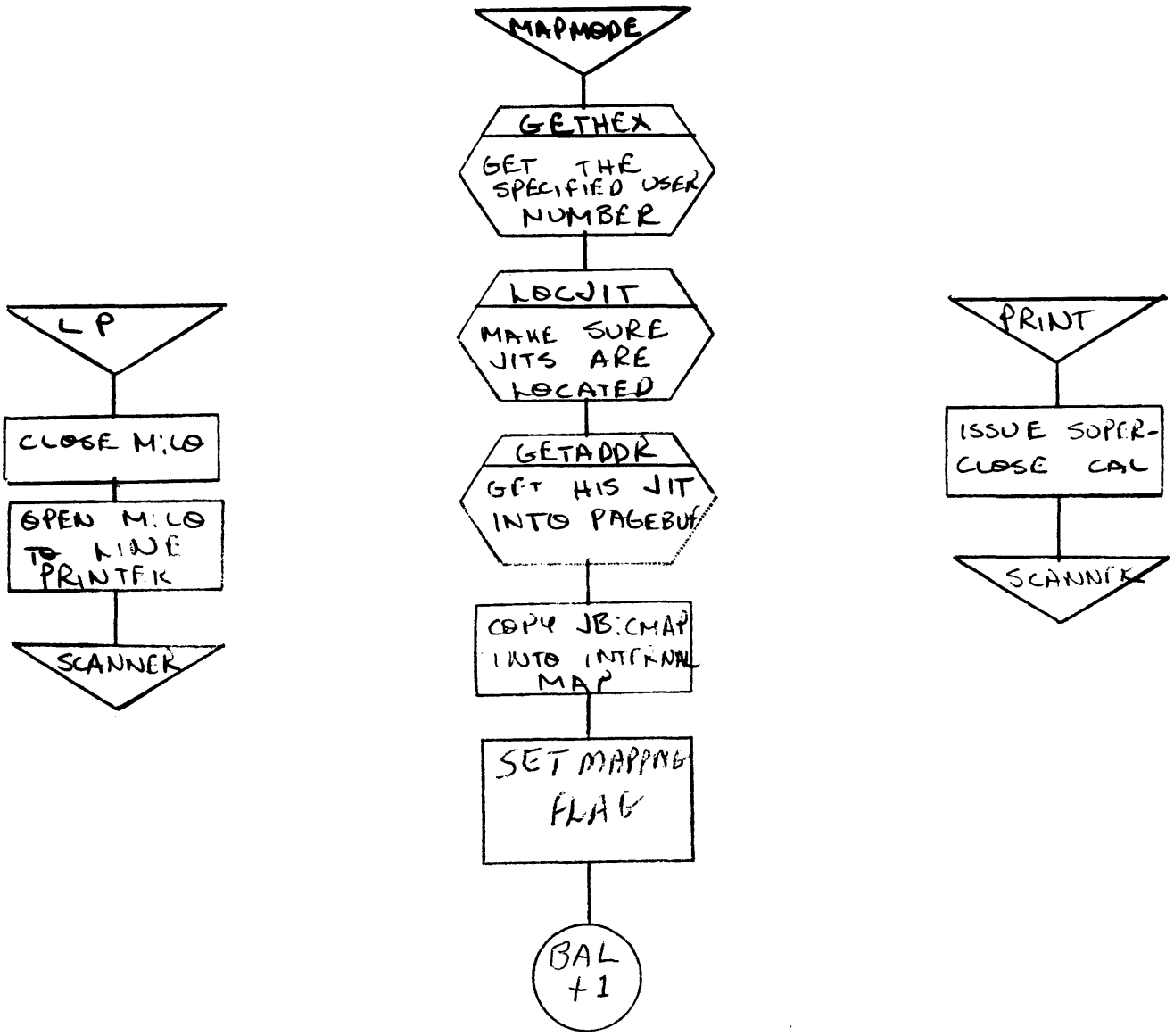

DTABSRCH
Chain Pointer Zero ?
yes
BAL +1
no
Initial Peek at This Chain ?
no
yes
Advance Pointer (of Stop Address) By One.
Fetch 9 Words of DCB Name From Chain.
Use DCB Name Length to Advance Chain Pointer.
Move Name and DCB Address into my Pointers.
Next Look BG at End of Chain ?
yes
Reset Initial Look Flag for Next Pass Through.
no
BAL +2
DCBSTAT
Fetch first 2 Words of DCB Contents.
Extract "ASN" "FCD", "TYC" Fields.
Use Extracted Fields to Get Canned MSG About DCB State.
"Closed" "Open" etc.
"Device" "File" etc.
BAL +1
DCBPRINT
Construct Print Line, Name, Address, State Assingment, etc.
Display Print Line.
BAL +1

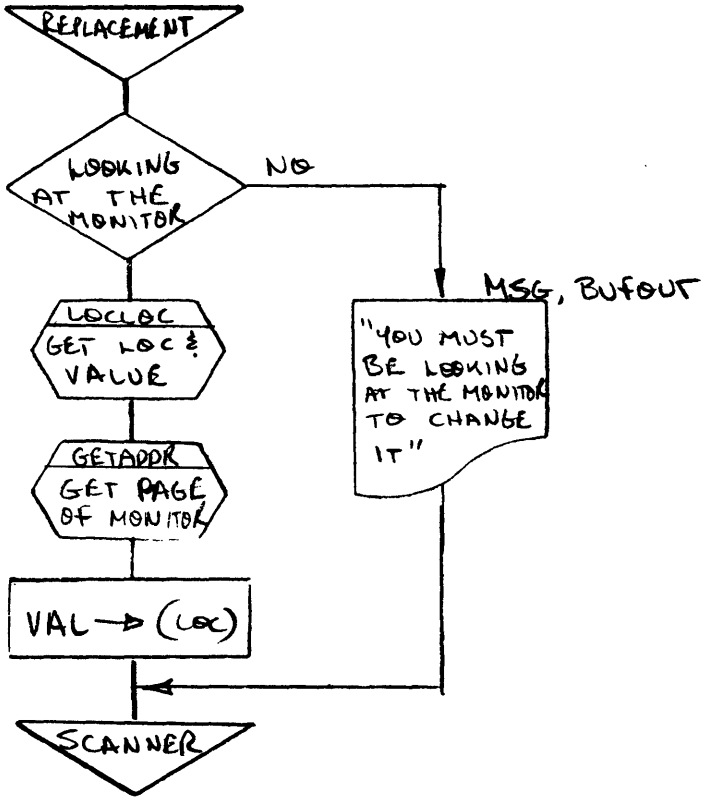

DUM
DLAST
DNEXT
INDR
LOCLOC
GET SPECIFIED DUMP LIMITS
SPECIFY 1 LOC, LASTLOC - 1
SPECIFY 1 LOC, LAST LOC + 1
GET LASTLOC, FIND ADDRESS AND SPECIFY IT
SAVE LAST WORD OF CURRENT REQUEST
LASTLOC
GETPAGE
GET PAGE CONTAINING 1ST LOCATION
DUMPSOME
DUMP BETWEEN & INCLUDING LIMITS OF REQUEST
SCANNER


EXTCL
Close M:EI With Save.
CAL1,9 1
Exit to TEL or CCI
INITIAL
Get Two Pages.
M:GP
Save Address and Limits.
PAGEBUF, BUFLIM
Am I A Ghost ?
no
yes
BATCH ?
no
yes
LASTCRASH
0 1
31
J:JIT
1 1
User is a GHOST (40)
User is On-Line (80)
Neither User is BATCH (00)
Take Break Control.
"ANALYZE HERE"
"TYPE INPUT COMMAND"
MSG
Put Message In OBUF.
BUFOUT
Output OBUF.
SCANNER


INPUT
M:EI OPEN
NO
YES
Y002 ;; M:EI
CLOSE M:EI
RESET JIT ADDRESS TABLE
OPNUTSP
LASTCRASH
SPECFIL
LABEL$TAPE
TAPEP
OPEN M:EI TO UTSDUMP
SCANNER
GET A PAGE AND FREE IT FOR M:SAD
SAD SUCCESSFUL
NO
YES
" PRIVILEGE NOT HIGH ENOUGH"
SCANNER
GET RCURCNT & BUILD FILE NAME FREE SAD PAGE
OPEN MONDMPn
PICK UP SPECIFIED n FROM BUFFER
STORE IN FILE NAME
M:EI OPEN
NO
YES
CLOSE M:EI
READ RECORD, FORM KEY, WRITE FILE
GET 8K BUFFER
RESTORE KEY
POSITION PAST 1ST RECORD (BOOTSTRAP)
READ 8K, WRITE ON FILE PAGE AT A TIME
FINISHED
NO
YES
RELEASE BUFFER
ABN FROM LABEL $ TAPE READ
CLOSE FILE AND TAPE
TELL # OF PAGES IN DUMP
OPEN FILE FOR INPUT
SCANNER
GHOST JOB
YES
NO
ALL
SCANNER
J:JIT
ERR/ABN RETURN FROM OPEN OF MONDMPN
GHOST JOB
NO
YES
BEEN HERE ?
YES
NO
EXIT
SCANNER
LABEL $ TAPE

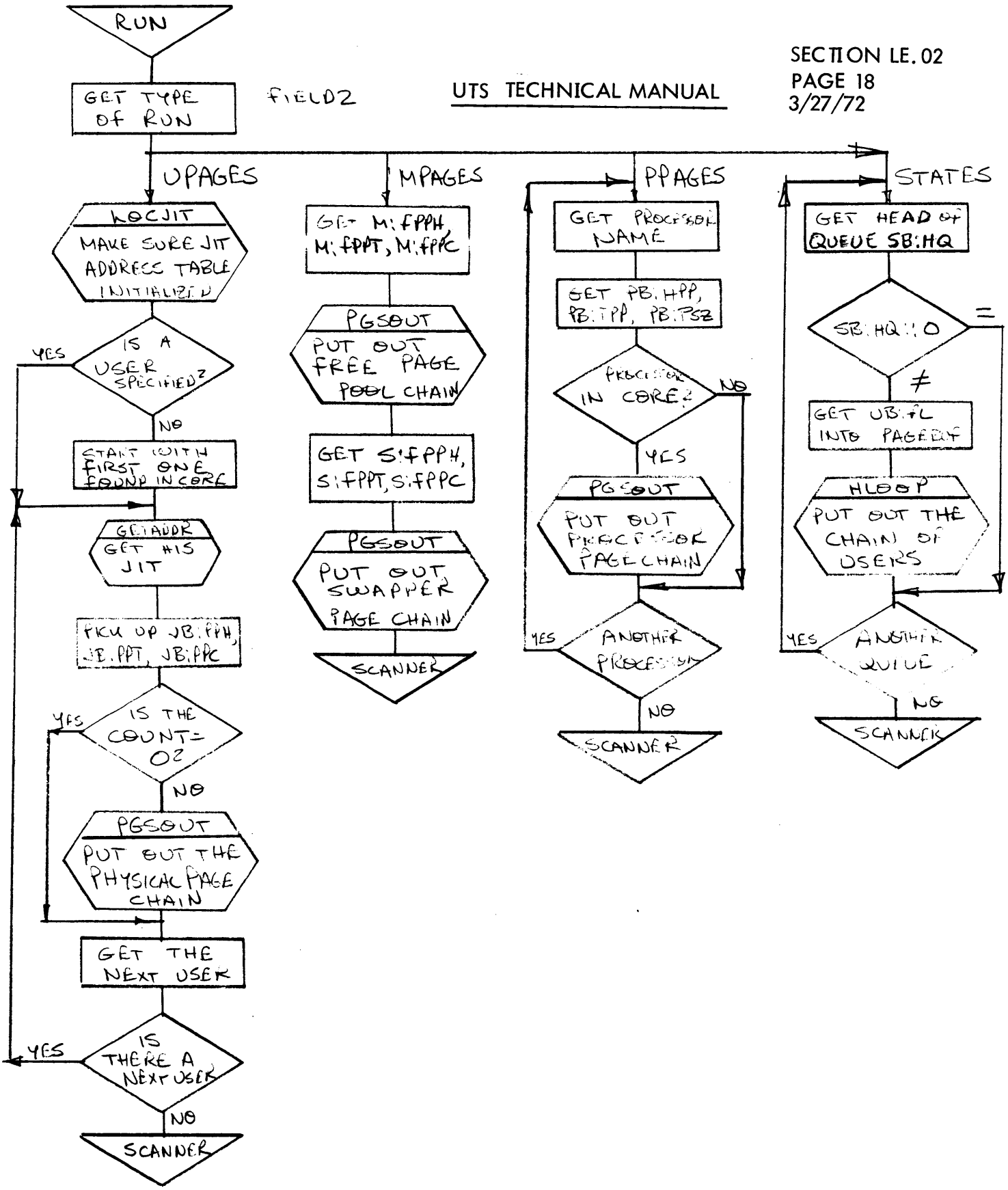

MONITOR
"DISPLAY" SPECIFIED
YES
NO
SET MONFLAG AND READAGAINFLG
RESET MONFLAG SET READAGAINFLG
SCANNER

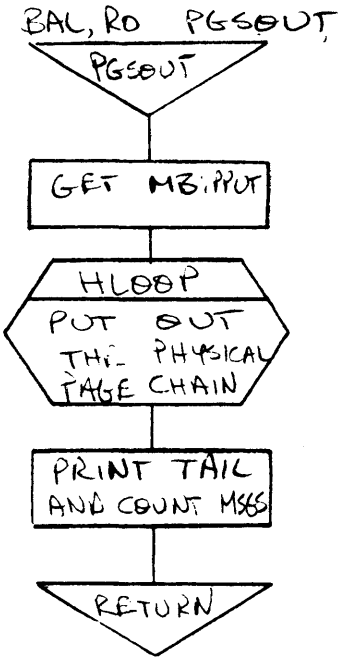

MAPMODE
GETHEX
GET THE SPECIFIED USER NUMBER
LOCJIT
MAKE SURE JITS ARE LOCATED
GETADDR
GET HIS JIT INTO PAGEBUF
COPY JB:CMAP INTO INTERNAL MAP
SET MAPPING FLAG
BAL +1
LP
CLOSE M:LO
OPEN M:LO TO LINE PRINTER
SCANNER
PRINT
ISSUE SOPER-CLOSE CAL
SCANNER

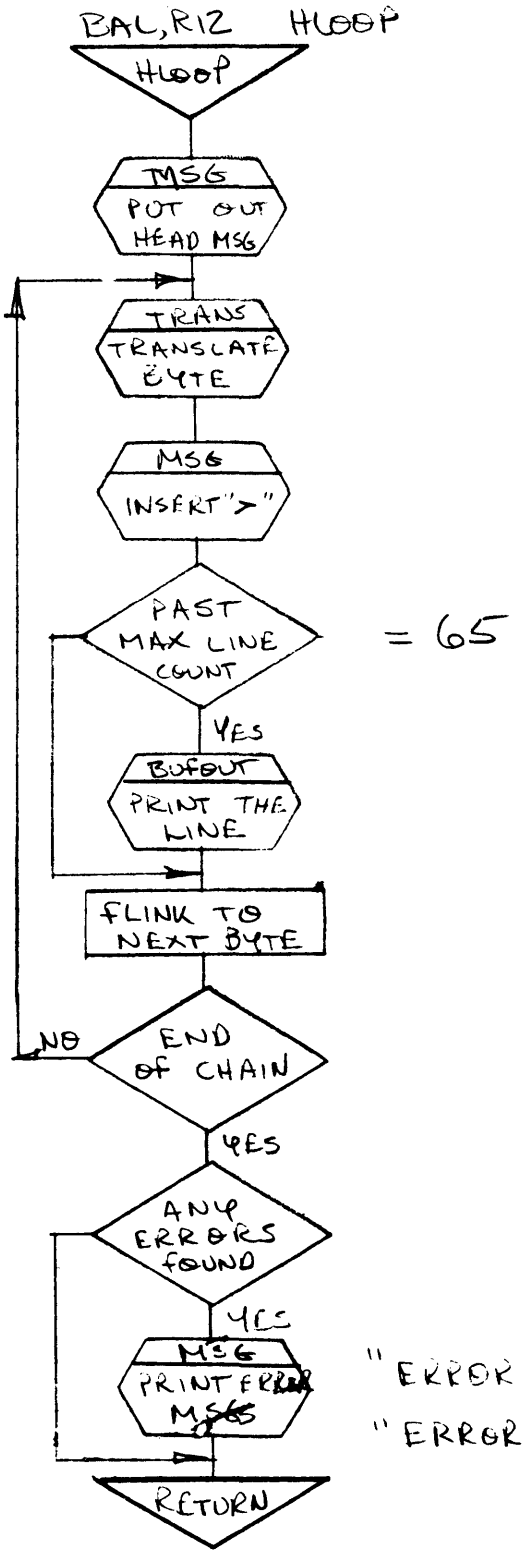

REPLACEMENT
LOOKING AT THE MONITOR
NO
MSG, BUFOUT
"YOU MUST BE LOOKING AT THE MONITOR TO CHANGE IT"
LOCLOC
GET LOC & VALUE
GETADDR
GET PAGE OF MONITOR
VAL → (LOC)
SCANNER

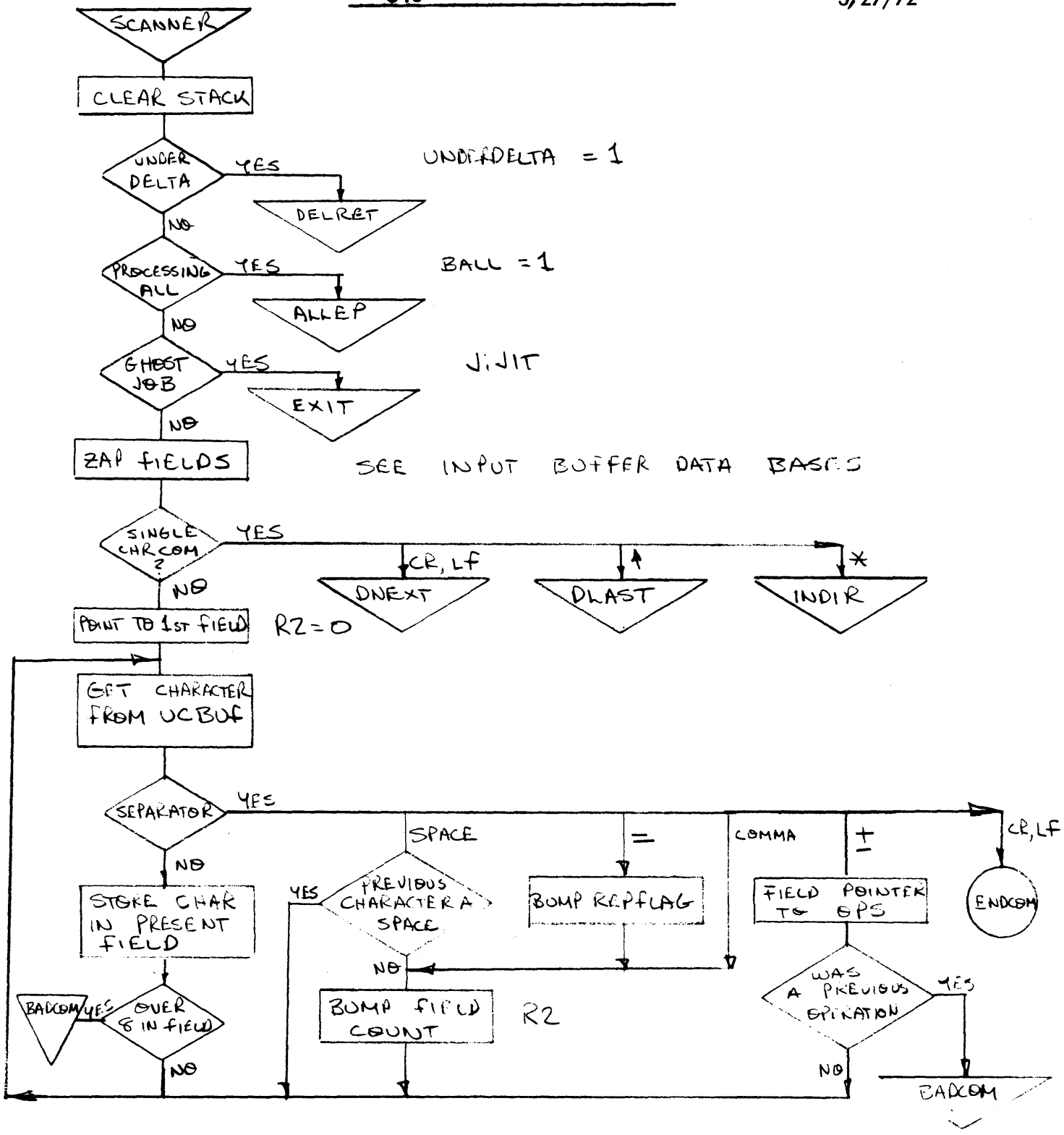
FIELD2

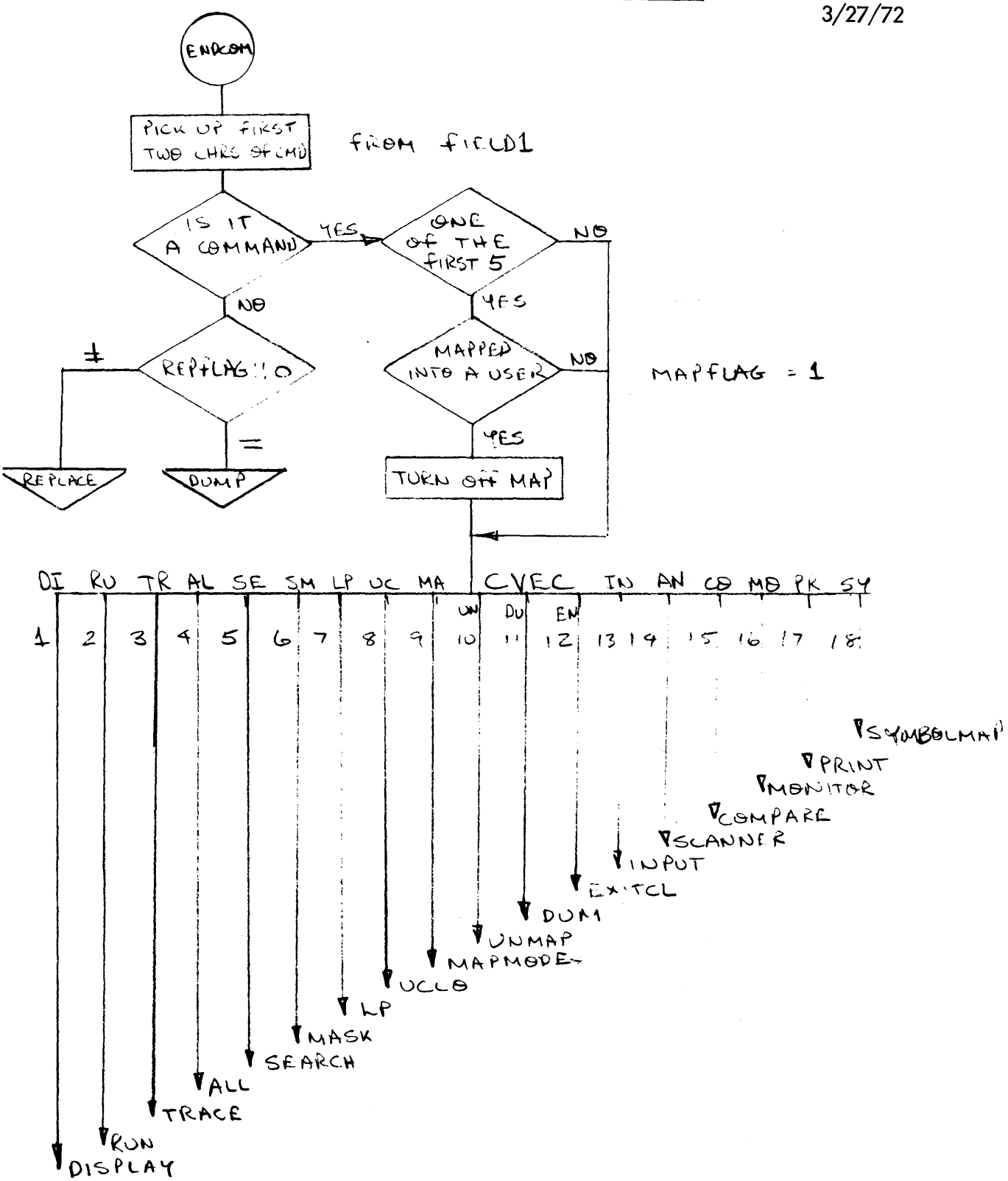
UTS TECHNICAL MANUAL

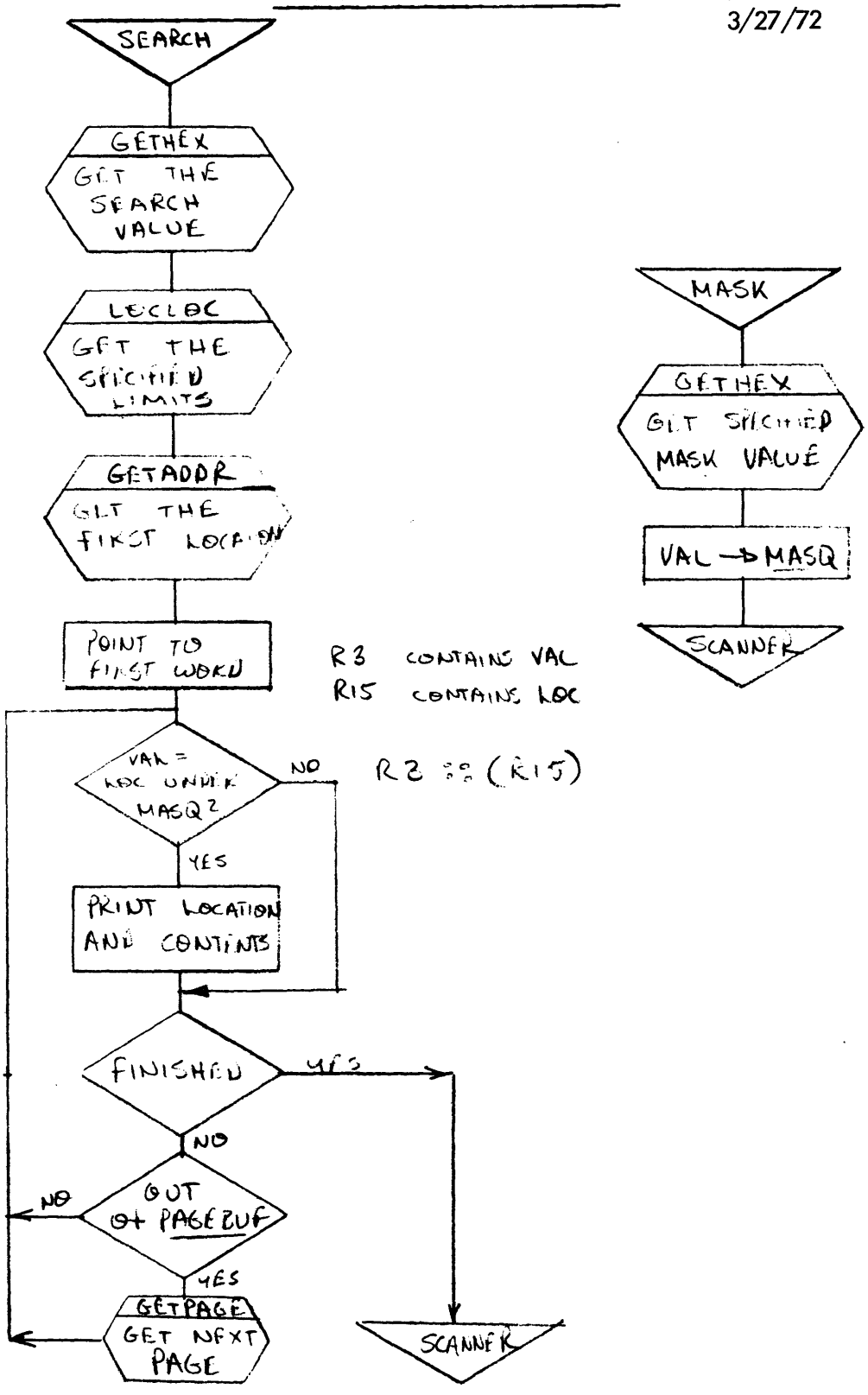


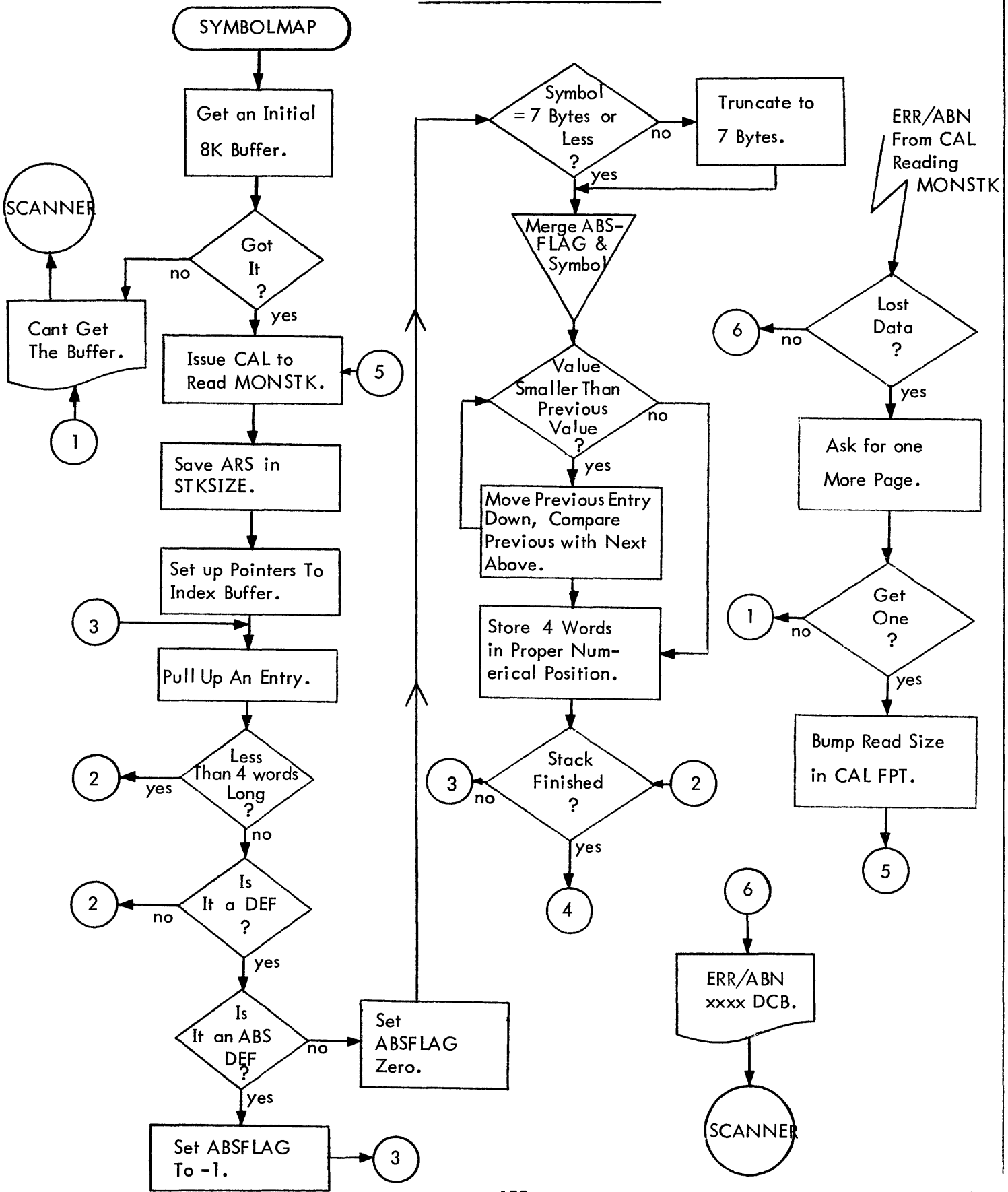

BAL, R0 PGSOUT
PGSOUT
GET MB:PPUT
HLOOP
PUT OUT THE PHYSICAL PAGE CHAIN
PRINT TAIL AND COUNT MSGS
RETURN


BAL,R12 HLOOP
HLOOP
MSG
PUT OUT HEAD MSG
TRANS
TRANSLATE BYTE
MSG
INSERT ">"
PAST MAX LINE COUNT
= 65
YES
BUFOUT
PRINT THE LINE
FLINK TO NEXT BYTE
NO
END of CHAIN
YES
ANY ERRORS FOUND
YES
MSG
PRINT ERROR MSGS
"ERROR IN TAIL"
"ERROR IN COUNT"
RETURN

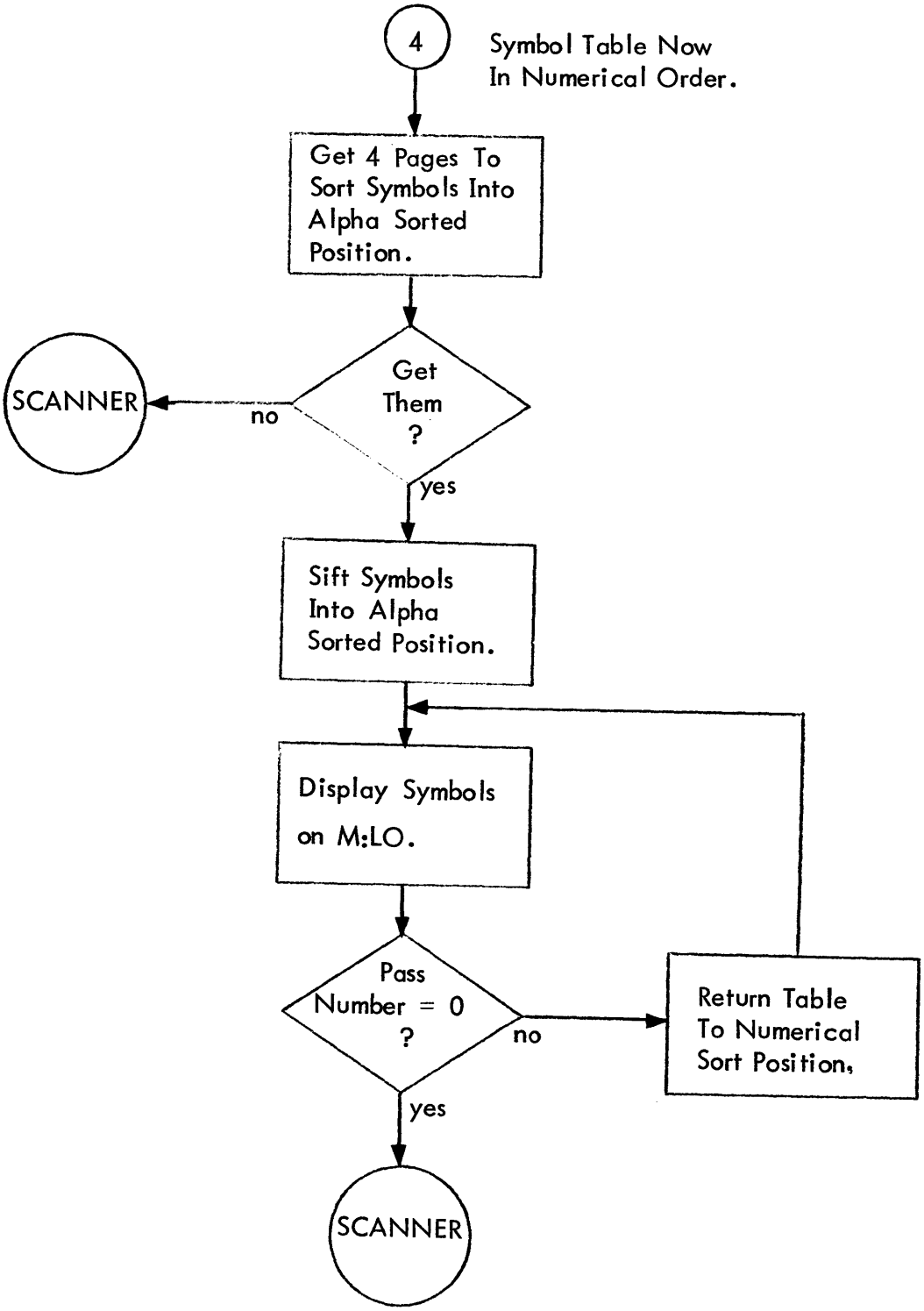

SCANNER
CLEAR STACK
UNDER DELTA
YES
UNDRADELTA = 1
DELRET
NO
PROCESSING ALL
YES
BALL = 1
ALLEP
NO
GHOST JOB
YES
JJJIT
EXIT
NO
ZAP FIELDS
SEE INPUT BUFFER DATA BASES
SINGLE CHR COM ?
YES
CR, LF
DNEXT
DLAST
*
INDIR
NO
POINT TO 1ST FIELD
R2 = 0
GET CHARACTER FROM UCBUF
SEPARATOR
YES
SPACE
=
COMMA
+
-
CR, LF
ENDCOM
NO
STORE CHAR IN PRESENT FIELD
YES
PREVIOUS CHARACTER A SPACE
BUMP REPFLAG
FIELD POINTER TO OPS
BADCOM
YES
OVER 6 IN FIELD
NO
BUMP FIELD COUNT
R2
NO
WAS A PREVIOUS OPERATION
YES
NO
BADCOM

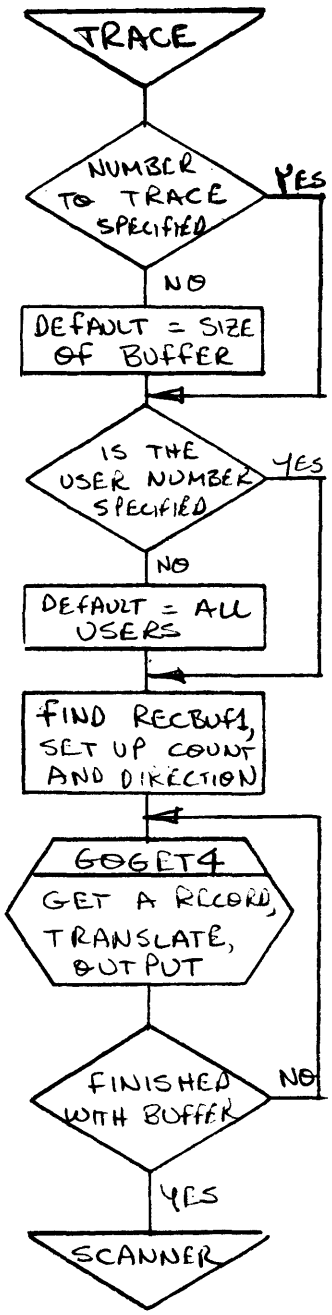

ENDCOM
PICK UP FIRST TWO CHRS OF CMD
FROM FIELD1
IS IT A COMMAND
YES
ONE OF THE FIRST 5
NO
YES
NO
≠
REPFLAG != 0
MAPPED INTO A USER
NO
MAPFLAG = 1
=
YES
REPLACE
DUMP
TURN OFF MAP
DI RU TR AL SE SM LP UC MA CVEC IN AN CO MO PK SY
UN DU EN
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
SYMBOLMAP
PRINT
MONITOR
COMPARE
SCANNER
INPUT
EXITCL
DUM
UNMAP
MAPMODE
UCLO
LP
MASK
SEARCH
ALL
TRACE
RUN
DISPLAY

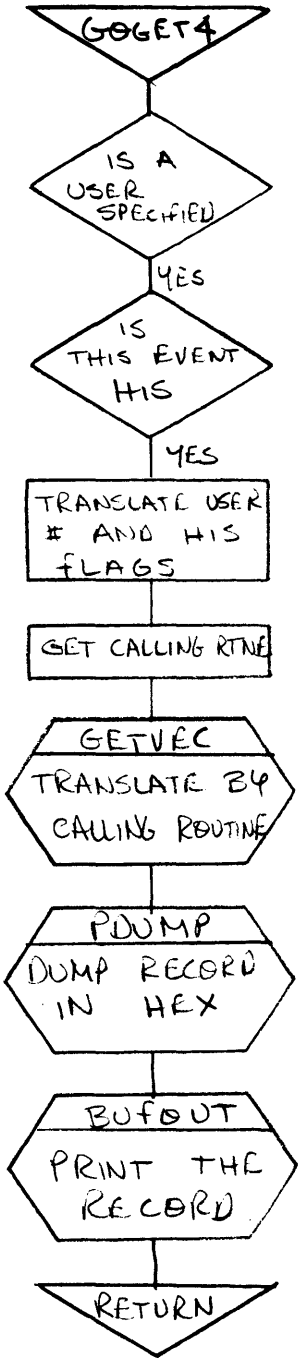

SEARCH
GETHEX
GET THE SEARCH VALUE
LOCLOC
GET THE SPECIFIED LIMITS
GETADDR
GET THE FIRST LOCATION
POINT TO FIRST WORD
R3 CONTAINS VAL
R15 CONTAINS LOC
VAL = LOC UNDER MASQ?
NO
R2 = (R15)
YES
PRINT LOCATION AND CONTENTS
FINISHED
YES
NO
OUT OF PAGEBUF
NO
YES
GETPAGE
GET NEXT PAGE
SCANNER
MASK
GETHEX
GET SPECIFIED MASK VALUE
VAL → MASQ
SCANNER


SYMBOLMAP
Get an Initial 8K Buffer.
SCANNER
Got It ?
no
yes
Cant Get The Buffer.
1
Issue CAL to Read MONSTK.
5
Save ARS in STKSIZE.
Set up Pointers To Index Buffer.
3
Pull Up An Entry.
Less Than 4 words Long ?
2
yes
no
Is It a DEF ?
2
no
yes
Is It an ABS DEF ?
no
Set ABSFLAG Zero.
yes
Set ABSFLAG To -1.
3
Symbol = 7 Bytes or Less ?
no
Truncate to 7 Bytes.
yes
Merge ABS-FLAG & Symbol
Value Smaller Than Previous Value ?
no
yes
Move Previous Entry Down, Compare Previous with Next Above.
Store 4 Words in Proper Numerical Position.
Stack Finished ?
3
no
2
yes
4
ERR/ABN From CAL Reading MONSTK
Lost Data ?
6
no
yes
Ask for one More Page.
Get One ?
1
no
yes
Bump Read Size in CAL FPT.
5
6
ERR/ABN xxxx DCB.
SCANNER

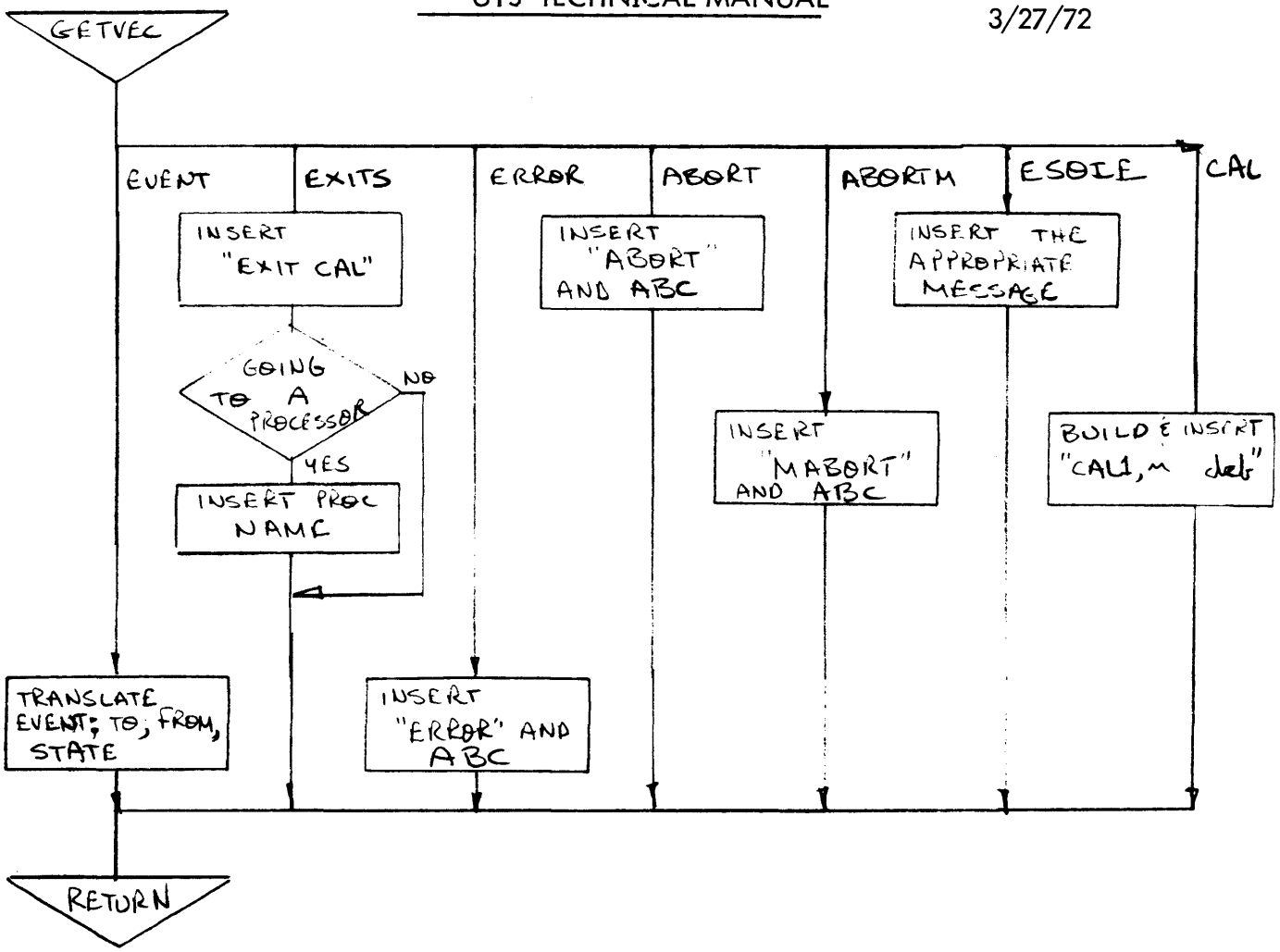
UTS TECHNICAL MANUAL




TRACE
NUMBER TO TRACE SPECIFIED
YES
NO
DEFAULT = SIZE OF BUFFER
IS THE USER NUMBER SPECIFIED
YES
NO
DEFAULT = ALL USERS
FIND RECBUF1, SET UP COUNT AND DIRECTION
GOGET4
GET A RECORD, TRANSLATE, OUTPUT
FINISHED WITH BUFFER
NO
YES
SCANNER

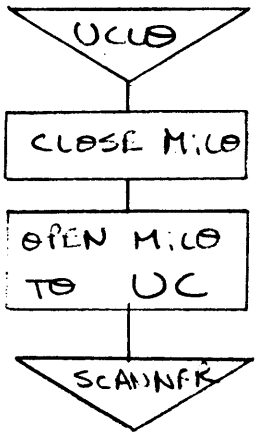

GOGET4
IS A USER SPECIFIED
YES
IS THIS EVENT HIS
YES
TRANSLATE USER # AND HIS FLAGS
GET CALLING RTNE
GETVEC
TRANSLATE BY CALLING ROUTINE
PDUMP
DUMP RECORD IN HEX
BUFOUT
PRINT THE RECORD
RETURN

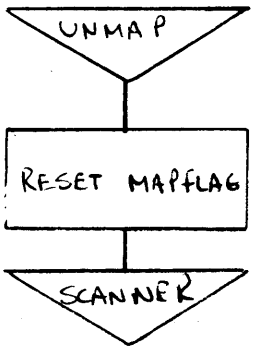
TRANSS2

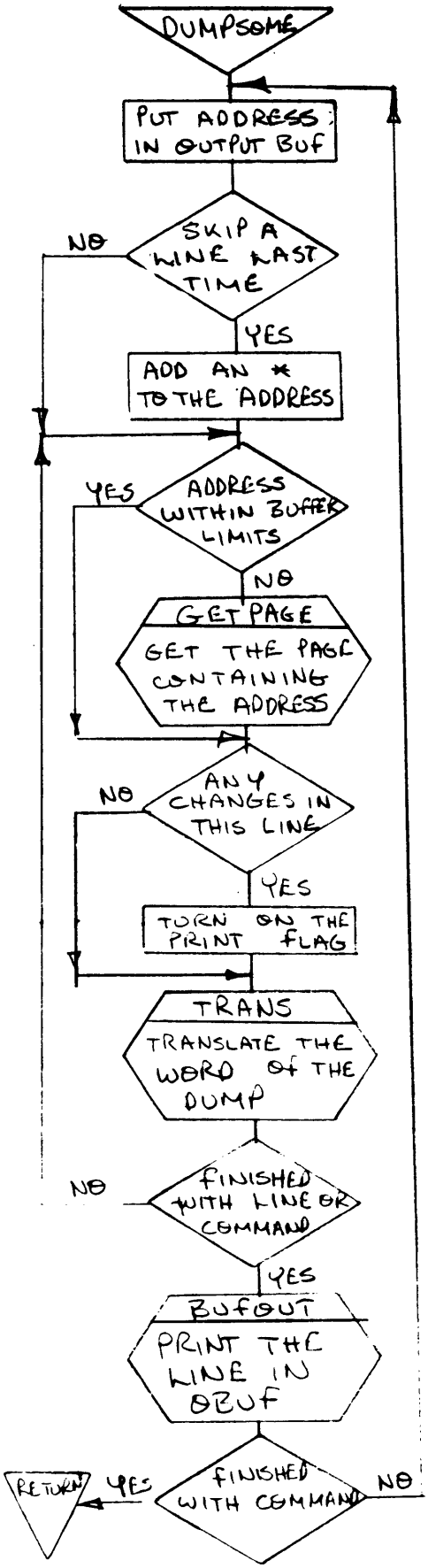


CONSULT THE DOCUMENTATION ON THE FORMAT OF THE EVENT RECORDER TO INTERPRET THIS FLOW CHART

ESOIE TRANSLATES EXECUTION, SWAP OUT, SWAP IN, AND ENDSWAP RECORDS.


UCLO
CLOSE MILO
OPEN MILO
TO UC
SCANNER


UNMAP
RESET MAPFLAG
SCANNER



OBUF

PFLAGS

BUFLIM

LASTWORD = SAME AS CURRENT

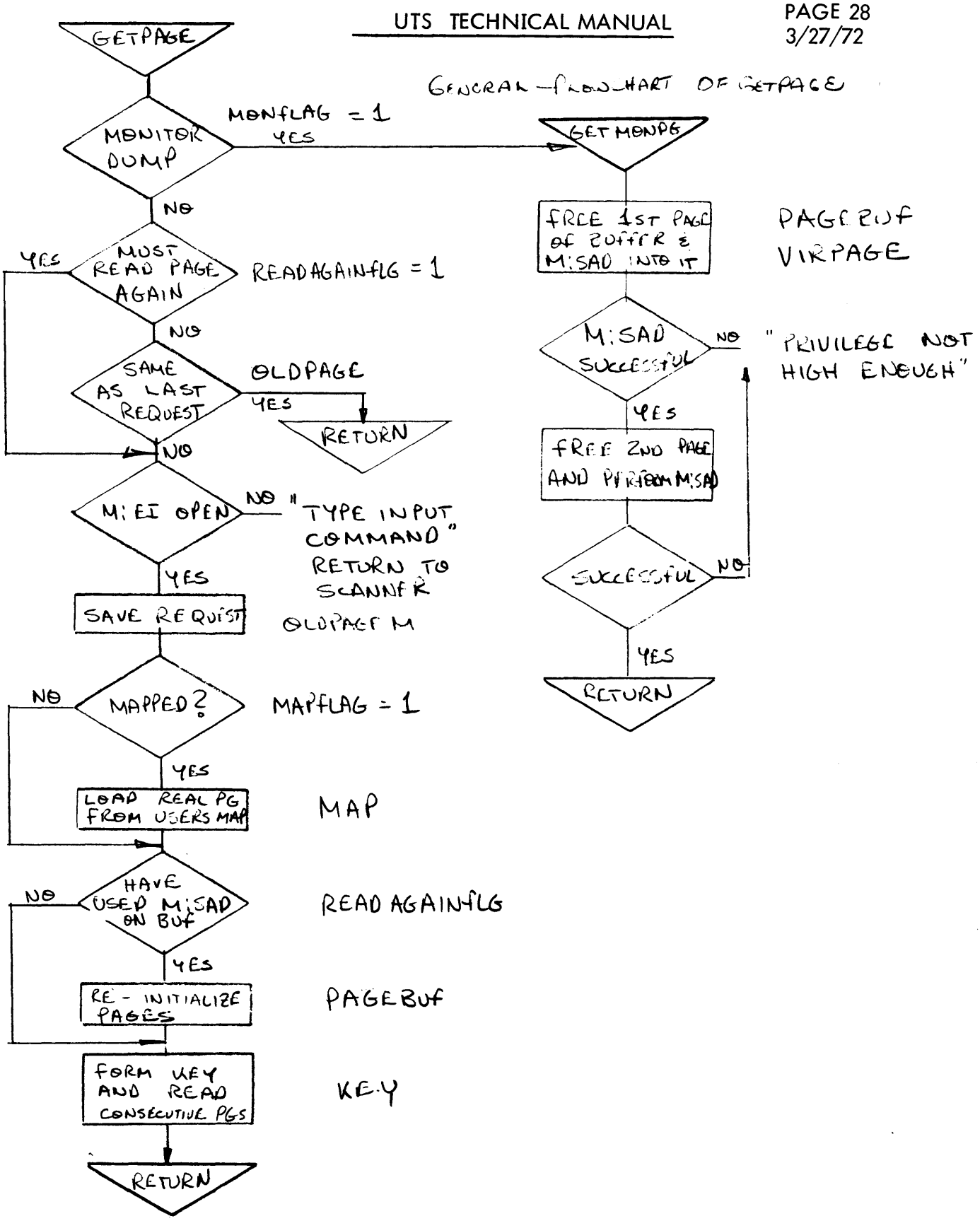
PFLAGS

R7 = 0

R7 = 0

UTS TECHNICAL MANUAL

GENERAL FLOWCHART OF GETPAGE



To implement a new display command:

1. Add to the table DCOM in ANLZ the first two characters of the second field of the display command; for the example, 'FN' is added.

2. Add to the transfer vector GOTIT in ANLZ, in parallel to the addition in 1 above, a branch to the entry label for the new display routine; for the example, B FNORTTABS is added.

3. If the routine is in an overlay, the entry label must be REF'ed.

To include the display in the ALL command (the crash dump):

1. Add to the transfer vector AOPS in ANLZ a branch to the entry label for the new display routine.

2. Same as step 3 above.

To implement as a totally new ANLZ command:

1. Add to the table COMMANDS in ANLZ the first two characters of the new command; for the example, 'FN' is added.

2. In parallel to 1 above, add to the transfer vector CVEC in ANLZ a branch to the entry label; for the example, B FNORTTABS is added.

3. REF the entry if it's in an overlay.

The areas in ANLZ referred to in the above procedures are fairly well documented and any questions may be resolved by examining the code.

ID

Adding code to ANLZ.

New display or dump routines are simple to code using existing ANLZ subroutines, and simple to add to ANLZ by making them overlays or adding them to the current overlay ANALZ01. Descriptions of pertinent subroutines have been abstracted from the Technical Manual, and are appended to this section. The easiest way to learn to write ANLZ routines is by example, so, let's take as an example the FNORT tables and data, to be added as a display to ANLZ. These tables are indexed by user number, and consist of the following:

BFNORT	byte resolution - flags
HFNORT	halfword resulution - data
WFNORT	word resulution - physical address of a 16-word context block
DFNORT	doubleword containing account of the user in text.

The desired format is the following for each user in the system:

```
FNORT TABLES:
USER # BFNORT    HFNORT   DFNORT
  N    10101010    NNNN   ACCOUNT

CONTEXT:
NNNNNNNN NNNNNNNN NNNNNNNN NNNNNNNN ................
NNNNNNNN NNNNNNNN NNNNNNNN NNNNNNNN ................
NNNNNNNN NNNNNNNN NNNNNNNN NNNNNNNN ................
NNNNNNNN NNNNNNNN NNNNNNNN NNNNNNNN................
```

Notice BFNORT has been broken into its flag bits, the account printed in EBCDIC, and the context of 16 words is in the format of a core dump. This display might be coded as shown on the attachment, although with table-building PROCs the size of the routine would be significantly reduced.

There are three existing ways the new display may be added to ANLZ: (1) as a new display command, e.g., DISPLAY FNORT, (2) as a display to be included in the crash dump (ALL command), or (3) as a totally new command, e.g., FNORT. These should be implemented as follows.

```
          SYSTEM    SIG7
          DEF       FNORTTABS
          REF       UB:US, BFNORT, HFNORT, WFNORT, DFNORT
          REF       MSG, BUFOOT, MSGI, GETADDR
          REF       SPACES, BITPUT,DUMPSOME
          REF       TRANS, TRANSSZ, SMUIS, SCANNER
```

THIS FIRST PART OF THE EXAMPLE ROUTINE PUTS OUT THE TWO FIRST MESSAGES - THE HEADERS FOR THE FNORT DISPLAY

```
FNORTTABS  EQU       $
           LI, 1     FNORTMSG
           BAL, 0    MSG
           BAL, 0    BUFOUT
           LI, 1     FNORTHDR
           BAL, 0    MSG
           BAL, 0    BUFOUT
           LI, 6     1
```

FOR THE DURATION, R6 WILL CONTAIN THE USER NUMBER, R5 THE TAB INDEX FOR THE TABLE BEING DISPLAYED FOR THAT USER. THE FIRST PART CHECKS TO SEE IF THE USER'S STATE IS NON-ZERO.

```
FNORTLOOP1 EQU       $
           LI, 5     0
           LI, 14    UB:US
           BAL, 0    GETADDR
           MTB, 0    *15, 6
           BEZ       NEXTFNORT
```

TAB TO THE LOCATION OF THE USER # ON THE LINE TO BE OUTPUT

```
           LB, 1     TABS, 5
           BAL, 0    SPACES
```

NOW OUTPUT THE USER # ITSELF

```
           LW, 3     6
           BAL, 0    TRANSSZ
```

TAB TO THE FIRST TABLE DISPLAY AND PUT OUT THE ENTRY FOR THE USER IN BIT FORMAT

```
AI, 5       1
LB, 1       TABS, 5
BAL, 0      SPACES
LI, 14      BFNORT
BAL, 0      GETADDR
LB, 3       *15, 6
BAL, 0      BITPUT
```

TAB TO THE NEXT THEN PUT OUT THE NEXT ENTRY IN HEXIDECIMAL SUPPRESSING LEADING ZEROS

```
AI, 5       1
LB, 1       TABS, 5
BAL, 0      SPACES
LI, 14      HFNORT
BAL, 0      GETADDR
LH, 3       *15, 6
```

BEING SURE TO TRIM OFF ANY SIGN BITS DUE TO THE LOAD HALF

```
AND, 3      =X'FFFF'
BAL, 0      TRANSSZ
```

ON TO THE NEXT, TO BE DISPLAYED IN EBCDIC

```
AI, 5       1
LB, 1       TABS, 5
BAL, 0      SPACES
LI, 14      DFNORT
BAL, 0      GETADDR
LD, 8       *15, 6
LI, 1       8
LI, 2       8
BAL, 0      MSGI
```

THE LINE COMPLETED, PRINT IT, THEN DUMP THE CONTEXT STARTING WITH THE MESSAGE

```
                BAL, 0          BUFOUT
                LI, 1           CONTEXT
                BAL, 0          MSG
                BAL, 0          BUFOUT
                LI, 14          WFNORT
                BAL, 0          GETADDR
                LW, 14          *15, 6
```

WITH CONTEXT ADDRESS IN HAND HEAD FOR THE DUMPSOME ROUTINE

```
                BAL, 0          GETADDR
                LW, 8           15
                LI, 7           16
                BAL, 0          DUMPSOME
NEXTFNORT       EQU             $
                AI, 6           1
```

IF FINISHED, HEAD BACK TO THE BARN OTHERWISE, GET THE NEXT USER AND DUMP HIS FNORT TABLES TOO

```
                CI, 6           SMUIS
                BLE             FNORTLOOP1
                B               SCANNER

TABS            DATA, 1         2, 8, 19, 26
FNORTMSG        TEXTC           'FNORT TABLES:'
FNORTHDR        TEXTC           'USER # BFNORT HFNORT DFNORT'
CONTEXT         TEXTC           'CONTEXT:'
                END
```


Pages 166-171 deleted

M:UC	DCB for terminal I/O
M:RCLOK2	CLOCK2 for time
S:CUN	Current user number
S:ISUN	Inswap user number
SB:OSN	Number of outswap users in
SB:OSUL	The outswap user list
TSTACK	Temp stack in the JIT
UB:APR	User table of associated processor number

OUTPUT

The format of the recording buffer is described in Section VO. 02.

INTERACTION

None

DATA BASES

None

SUBROUTINES

None

ERRORS

None

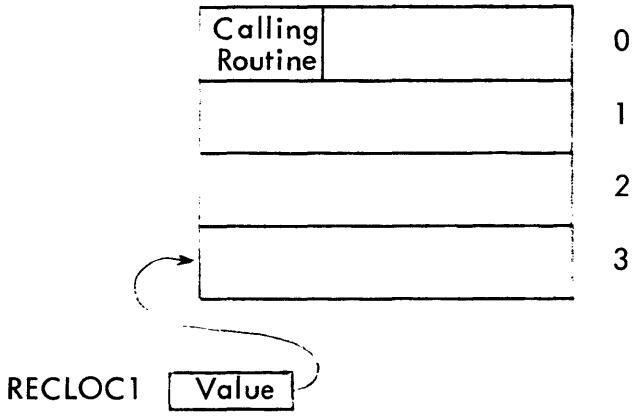
RESTRICTIONS

None

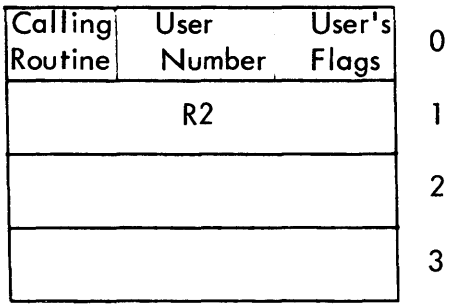
DESCRIPTION

When accessed, RECORD first decides to record or not based on the setting of the soft-switch RONOFF. If it is set, the pointer RECLOC1 is compared against the size of the buffer RECEND; and if equal, is set to the beginning of the buffer RECBUF to wrap-around properly.

RECLOC1 is bumped by 4 to make room for the information to be recorded, and the calling routine is set into the first byte of the entry.



If the number of the calling routine >10, control is given to the routine RNINER, which places the current user number and his flags in word 0, places R2 from the calling routine in word 1 of the entry, and transfers control to the common exit RECTHRU.



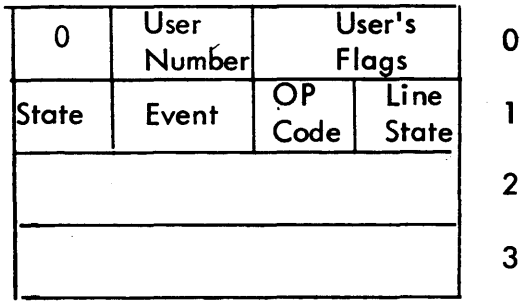
Otherwise, the vector RECVEC is used to branch to the routine handling the entry for the recognized callers:

Routine	Caller
RECEVNT	0
RECEXIT	1 - 4
RECATSE	5
RECSWAP	6 - 9
RECCAL	10

All the above routines exit to RECTHRU. RECEVNT expects the following information:

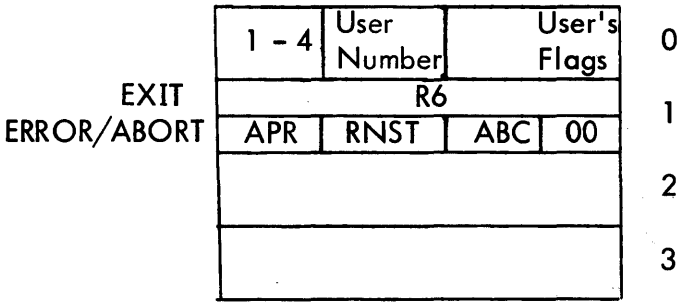
R2 = OP code (usually new state)
R3 = Current state
R4 = User number
R6 = Event number
R7 = Line number

and stores it in the entry in the following format:



The line state entry is made only if the user is on-line, and is obtained from STATE.

RECEXIT obtains current user from S:CUN, flags from UH:FLG, inserts item in the entry, then checks to see if this entry is for a normal exit. If so, R6 is set in word 1 of the entry. Otherwise the associated processor from UB:APR and ABC and RNST from the user's JIT are set into word 1.



RECATSE merely records the user number and his flags in word 0.

RECSWAP determines if a swap is being scheduled, and if so merely inserts zeros in word 1. Otherwise the inswap user number and his flags are placed in word 0, and again word 1 is filled with zeros.

SWAP SCHEDULE	6	0 ——— 0		0
SWAP	7 - 9	ISUN	User's Flags	
	0 ——— 0			1
				2
				3

RECCAL expects the following information:

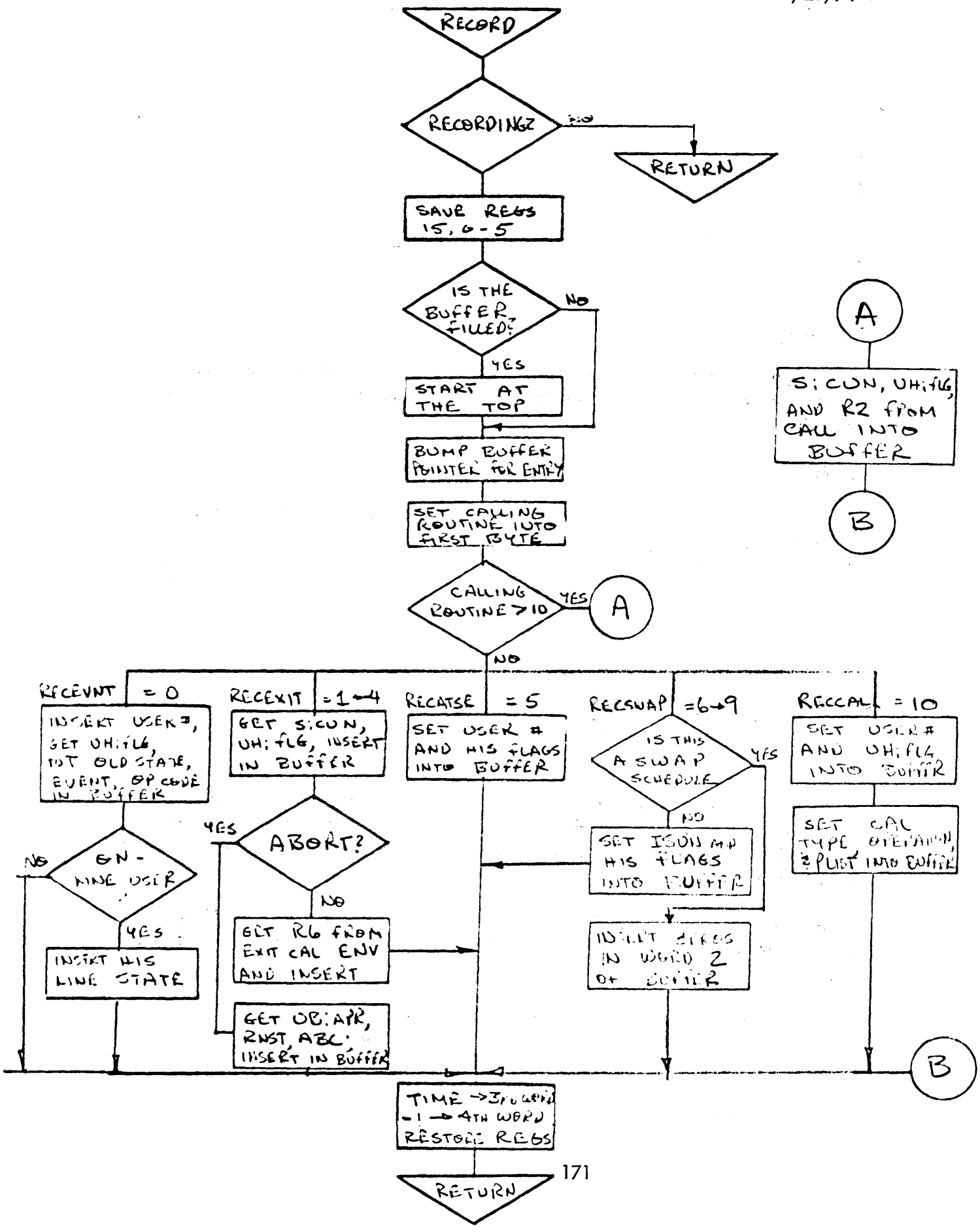
R3 = CAL1 type
R6 = 1st word of the PLIST
R8 = OP code from R6

The current user and his flags are set into word 0, and the CAL type, the OP code, and the right half of the 1st word of the PLIST (usually the DCB address) are set into word 1.

A	User Number	User's Flags	0
CAL TYPE	CAL OP	1st Word of PLIST(RH)	1
			2
			3

RECTHRO records the time of the report in word 2, and fills word 3 with F's, to facilitate reading the buffer in a dump.

Control is then returned to the calling routine.


RECORD
RECORDING?
NO
RETURN
SAVE REGS 15, 0-5
IS THE BUFFER FILLED?
NO
YES
START AT THE TOP
BUMP BUFFER POINTER FOR ENTRY
SET CALLING ROUTINE INTO FIRST BYTE
CALLING ROUTINE > 10
YES
A
NO
A
S; CUN, UH; FLG, AND R2 FROM CALL INTO BUFFER
B
RECEVNT = 0
INSERT USER #, GET UH; FLG, PUT OLD STATE, EVENT, OP CODE IN BUFFER
GEN - MINE USER
NO
YES
INSERT HIS LINE STATE
RECEXIT = 1→4
GET S; CUN, UH; FLG, INSERT IN BUFFER
YES
ABORT?
NO
GET R6 FROM EXIT CAL ENV AND INSERT
GET OB; APR, RNST, ABC; INSERT IN BUFFER
RECATSE = 5
SET USER # AND HIS FLAGS INTO BUFFER
RECSWAP = 6→9
IS THIS A SWAP SCHEDULE
YES
NO
SET ISUN AND HIS FLAGS INTO BUFFER
INSERT ERRS IN WORD 2 OF BUFFER
RECCAL = 10
SET USER # AND UH; FLG INTO BUFFER
SET CAL TYPE, OPERATION, & PLIST INTO BUFFER
B
TIME → 3RD WORD
-1 → 4TH WORD
RESTORE REGS
RETURN

UTS TECHNICAL MANUAL

ID

DRSP - Dynamic Replacement of Shared Processor

Design Specification

PURPOSE

Provides a dynamic facility for replacing, entering or deleting a shared processor or monitor overlay. Only users with a high privilege level (≥CO) will be able to use this facility.

OVERVIEW

The shared processor tables in core describe the characteristics (location and size) of the processors as they exist on the swapping RAD. The introduction of a new item (via ENTER or REPLACE) entails writing that item on the RAD and modifying the tables in core. Current users must be allowed to continue their association with the old copy. Therefore, until all current users disassociate, the old and new copies must exist simultaneously, each one occupying a separate slot in the tables. There can be no conflict between the old and new copies since the user tables retain only the index to the old copy after association, (rather than the name of the old copy). That is, current users will continue to get the old copy but a new user will associate with the new copy since the new slot is the only one which contains the name.

Extra slots will be provided as SYSGEN time, to accommodate brand new items (via ENTER) and to deal with the new copy without disabling the current one (via REPLACE). Additionally, space must be allocated on the swapping RAD to accommodate the new items.

When RAD and slot availability have been successfully established, SYSMAK writes the item to the swapping RAD and modifies the core version of the processor tables appropriately.

If the user requests a "PERM", he is asking for a copy which will be restored to the system after a crash. In the case of a processor this translates to the placement of the processor file within the :SYS account. For either a monitor overlay or a processor, the RAD version of the shared processor tables in the current monitor are changed in some selected items (e.g., flags, name).

The final phase consists of "enabling and disabling" the new and old copies respectively. With interrupts disabled, the new name is placed in the new slot and the old slot name is replaced by its index.

The program runs predominantly in slave mode with access to the monitor gained via the Change Virtual Map CAL. Master mode is used only when necessary. Usage of the program is restricted to one user to avoid a deadly embrace situation.

The DRSP commands recognize BREAK control at feasible junctures, respond with a type-out and return to the prompt situation.

PROPOSED ENHANCEMENTS

The following features are proposed as future enhancements.

1. Dynamic RAD and slot housekeeping. Fre up table slots and pool RAD resources.
2. Resolve SAVE-GET problem by including a date within the SAVE file and processor table.

DESCRIPTION

The write-up which follows is divided into

START	LH.01	Sets up the initial program conditions required by DRSP.
RESTART	LH.02	Reads DRSP commands and starts their execution
INITIAL	LH.03	Establishes access to the resident monitor
DRSPMAIN	LH.04	Serves as a driver for each major phase of processing.
SYNTAX	LH.05	Scans command line for options specified.
LIST	LH.06	Output to the user the contents of selected processor tables

RADNEED	LH.07	Reads the TREE record of the fid file
GETRADSLOT	LH.08	Finds a new slot for the proname
FINDSLOT	LH.09	Finds an available slot in P:NAME table
TELCCIONLY	LH.10	Sets up the environment necessary to replace TEL/CCI
WRITESWAP	LH.11	Calls SYSMAK1 to write fid to RAD
PERM	LH.12	Copies fid into :SYS account and modifies RAD tables
MODRAD	LH.13	Modifies shared processor tables on RAD
RWRAD	LH.14	Reads/Writes RAD tables
SWITCH	LH.15	Finalizes core copy of the processor tables
CLOSEOUT	LH.16	Restores DRSP's original conditions prior to reading the next command
	LH.17	These routines are used throughout the program
CLEANUP	LH.17.00	Release inactive processor and overlay name slots back to system
SEARCH	LH.17.01	Searches P:NAME table
GFID	LH.17.02	Decodes file, account, password from command line.
WORTH	LH.17.03	Checks if processor table slots are available
FINDGRAN	LH.17.04	Computes RAD granules associated with a shared processor
SCAN, SCANT	LH.17.05	Transfers a field from the command buffer to a specified area.
POST, POST 1	LH.17.06	Post error message
DAGRAN, GRANDA	LH.17.07	Disc address - granule conversion
TCTEST	LH.17.08	Test for TEL/CCI
XGRTEST	LH.17.09	Test for illegal pronames
BUFAD	LH.17.10	Buffer - table address conversion
MASTER, SLAVE	LH.17.11	Set MASTER/SLAVE modes
S400	LH.17.12	EBCDIC - hex conversion
S360	LH.17.13	Public library test

S430	LH. 17. 14	EBCDIC - digit conversion
HEX2PRNT	LH. 17. 15	Hex to printer conversion

USAGE

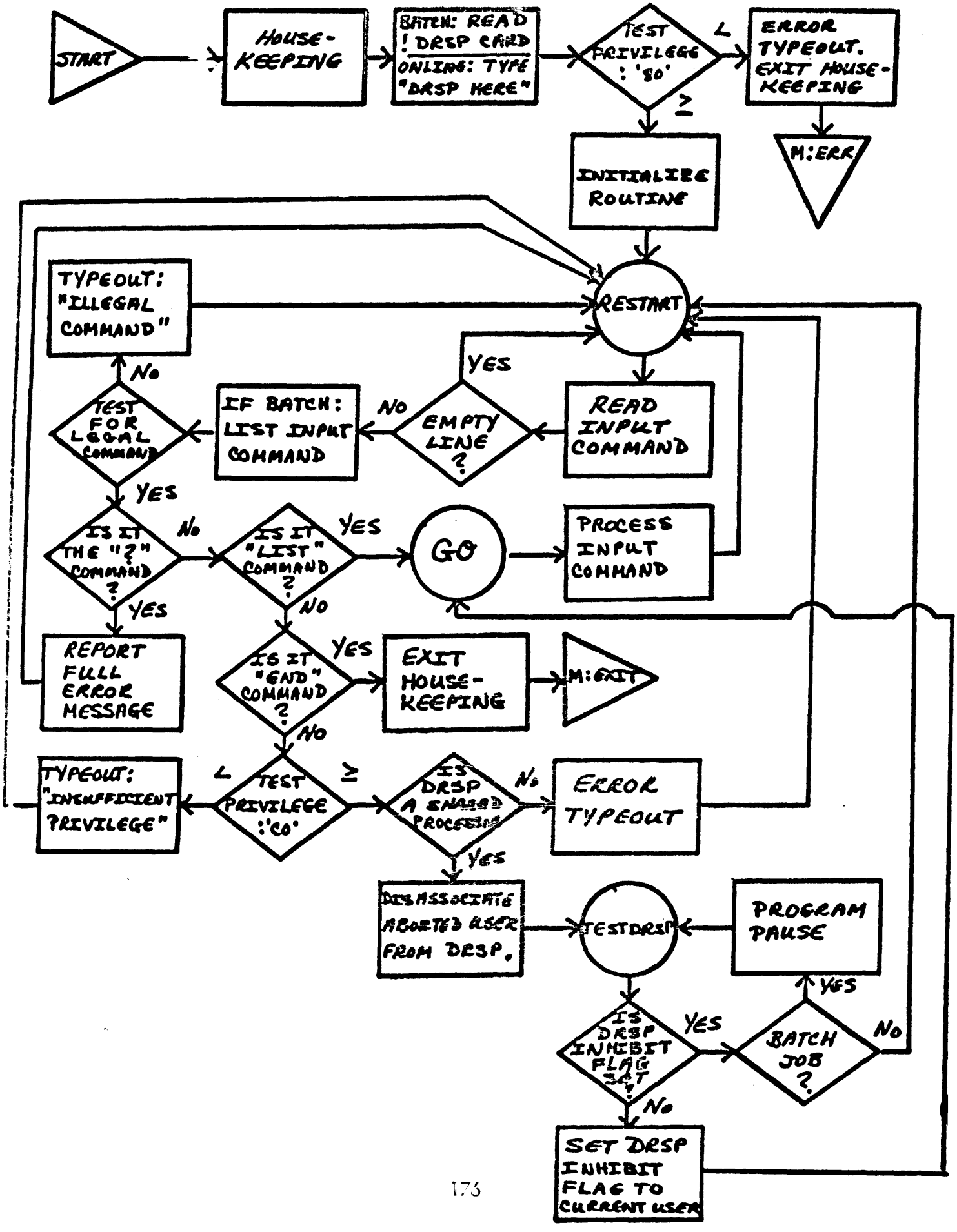
DRSP entered from TEL with the command DRSP or from CCI with a !DRSP card.

INTERACTION

SYSMAK1	Used to write the fid to the swapping RAD and to modify the core processor table.
NEWQ	The monitor I/O service used to read/write the tables under PERM.
GFB, RFB	The monitor routines used to get and release the monitor buffer to hold an end-action routine.

RESTRICTIONS

Replace or entered items are always load modules.
One level of overlay is permitted in a processor.
A monitor overlay itself has no tree structure.
DRSP occasionally runs in master mode.
Only one DRSP user at a time.
User of DRSP must have a privilege of $\geq$ C0.
Upon entry to SYSMAK, JIT access is acquired.
DRSP requires maximum core.


START
HOUSE-KEEPING
BATCH: READ ! DRSP CARD
ONLINE: TYPE "DRSP HERE"
TEST PRIVILEGE :'SO'
<
ERROR TYPEOUT. EXIT HOUSE-KEEPING
M:ERR
≥
INITIALIZE ROUTINE
RESTART
TYPEOUT: "ILLEGAL COMMAND"
READ INPUT COMMAND
EMPTY LINE ?
YES
NO
IF BATCH: LIST INPUT COMMAND
TEST FOR LEGAL COMMAND
NO
YES
IS IT THE "?" COMMAND ?
NO
YES
IS IT "LIST" COMMAND ?
YES
GO
PROCESS INPUT COMMAND
YES
NO
REPORT FULL ERROR MESSAGE
IS IT "END" COMMAND ?
YES
EXIT HOUSE-KEEPING
M:EXIT
NO
TYPEOUT: "INSUFFICIENT PRIVILEGE"
<
TEST PRIVILEGE :'CO'
≥
IS DRSP A SHARED PROCESS
NO
ERROR TYPEOUT
YES
DISASSOCIATE ABORTED USER FROM DRSP.
TESTDRSP
PROGRAM PAUSE
YES
IS DRSP INHIBIT FLAG SET ?
YES
BATCH JOB ?
NO
NO
SET DRSP INHIBIT FLAG TO CURRENT USER

UTS TECHNICAL MANUAL

ID

START

Design Specification

PURPOSE

Sets up the initial program conditions required by DRSP.

USAGE

Automatic starting point of DRSP. Executed only once during any run of DRSP. Exits to RESTART if privilege level is 80 or greater.

ERRORS

INSUFFICIENT PRIVILEGE LEVEL FOR DRSP USAGE

DESCRIPTION

START zeros out data area and sets up the stack pointer double-word. If the user is a batch job the !DRSP card is read and discarded. If the user is on-line, the message 'DRSP HERE' and the prompt character '>' are printed. The privilege is checked for >'80'; an error message is printed if it is not high enough and the routine aborts (M:ERR). If the privilege level is satisfactory, control is transferred to INITIAL to obtain memory required for execution. Then DRSP commands are processed starting at RESTART.

INTERACTION

FULLERR	Prints error message.

DATA BASE

J:JIT	On-line bit.
JB:PRIV	User's privilege level.

UTS TECHNICAL MANUAL

ID

RESTART

Design Specification

PURPOSE

To read DRSP commands and start their execution if they are legitimate and the user has the appropriate privilege level ($\geq$80 for LIST, $\geq$ CO for Replace, Enter and Delete).

USAGE

Point in DRSP to which control is returned in order to get next function to process.

ERRORS

ILLEGAL COMMAND
INSUFFICIENT PRIVILEGE LEVEL TO PROCESS THIS COMMAND
DRSP INHIBIT SET
DRSP PROGRAM ERROR (Shouldn't Happen)

INTERACTION

POST	Post error message
FULLERR	Print error message
SEARCH	Search P:NAME table (for DRSP)
MASTER	Enter Master Mode
SLAVE	Return to Slave Mode
SCAN	Scan input line for command name
M:WAIT	Suspends program execution waiting for change in the DRSP inhibit flag

DATA BASE

J:JIT	On-line bit
JB:PRIV	User's privilege level
UB:APR	Associated processor table
S:CUN	Current user number
DRSP	DRSP inhibit flag

OUTPUT

DRSP	Set to current user number (≡ inhibited)
COMD	One to eight characters of option specified; remaining characters are spaces.
CFLAG	one word code defining option specified -2 for REPLACE -1 for ENTER -0 for DELETE +1 for LIST options
EHHCNT	Character count for "EH" message set to 8.

DESCRIPTION

RESTART contains the house-keeping code required between execution of commands The next command i read (and printed if this is a batch job) and the privilege is checked to see if it can be executed ($\geq$80 for LIST, $\geq$ CO for Replace, Enter and Delete). If the command is '?', the latest error message is printed. If the command is 'R', 'E' or 'D', the DRSP inhibit flag is set to prevent another user from modifying the processor tables simultaneously. The LIST command does not set the inhibit flag. Once it is determined that the command can be executed, control goes to DRSPMAIN.

UTS TECHNICAL MANUAL

ID

INITIAL

Design Specification

PURPOSE

To establish access to the resident monitor.

USAGE

BAL, 15 INITIAL

OUTPUT

MONPGS	Number of pages in monitor required to access all pertinent tables, counts, etc.
MADDR	Beginning virtual address of monitor root.
PAGES	Contains location of first page obtained.
XTRA, W500	Contains location of next page to be accessed when reading TREE (RADNEED)
Monitor Address Table	Contains the virtual address of monitor data referenced by DRSP routines. The convention used in naming this table is to precede the monitor name with an 'X' (e.g. XP:NAME contains the virtual address of P:NAME)
2PGERR	Set to error code '2' if DRSP cannot obtain two additional pages to read the TREE record in RADNEED routine. Error message type-out is delayed until DRSP actually tries to read the TREE record in RADNEED.

INTERACTION

M:GP	Get pages to reference resident monitor.
M:FP	Free pages once it is determined they are available to reference resident monitor
M:CVM	Change Virtual Map to address monitor thru monitor address table.
M:GVP	Get Virtual Pages (2 pages) for subsequent read/write of TREE record in RADNEED routine.

SUBROUTINES

FULLERR	Print error message

ERRORS

INSUFFICIENT VIRTUAL MEMORY TO EXECUTE DRSP.
INSUFFICIENT MEMORY TO READ TREE.

DESCRIPTION

INITIAL establishes access to the resident monitor by:

1) calculating number of pages required to reference monitor counts, flags, etc.
2) obtaining the required number of pages (M:CVM), and
3) computing the addresses in the monitor address table.

The routine then gets two additional pages in order that RADNEED can read the TREE record. The next virtual address is stored in XTRA and W500, to be used to read/write the file in WRITESWAP and PERM.

UTS TECHNICAL MANUAL

ID

DRSPMAIN

Design Specification

PURPOSE

Serves as a driver for each major phase of the processing.

USAGE

Entered from RESTART in order to process command read.

DESCRIPTION

INITIAL and SYNTAX are called to establish initial conditions and read the command. If the command is LIST, subroutine LIST is called and control is passed to CLOSEOUT. If the command is DELETE, control is passed to PERM in case the PERM option is also wanted. In all other cases control is passed to RADNEED, GETRADSLOT, TELCCIONLY, WRITESWAP, PERM, SWITCH, and CLOSEOUT in that order. DRSPMAIN finally exits to RESTART.

ID

SYNTAX

Design Specification

PURPOSE

To scan command line for options specified; defines illegal combination of options; incidentally obtains P:NAME table index if proname exists and sets TEL/CCI flag.

USAGE

BAL, 15 SYNTAX

INPUT

COMD	one to eight characters of function specified; remaining characters are spaces
CFLAG	number code defining function
BYTE	current position in command line
ENDC	the terminating byte
EOI	the end of input character
BUF	the field composed by SCAN in TEXT format
LCF	the number of bytes in the field
CHAR	the command line image, one character/word
J:ACCN	the user's account number
MAXOVLY	index pointer to end of monitor overlays
XP:NAME	beginning of processor name table
XP:NAMEND	end of processor name table
XP:SA	processor flag table

OUTPUT

OINDEX	word containing address of "old" index in P:NAME table
NEWFLAGS	word containing processor flags as set by the command (in format nn0000000)
PRONAME	two word proname in TEXTC format
TEL:CCI	-1 if TEL, 1 if CCI, 0 if neither
PFLAG	flag set if PERM option requested
FILE	four words of fid file name in TEXTC format
ACCN	two words containing the users/file account number
PSWD	two words containing the file password if specified; otherwise zero
TYPE	flag (first byte of word) set to -1 for a processor and +1 for an overlay.
UFLAG	flag (first byte of word) set to "U" for the unconditional option; otherwise zero.
FFLAG	flag set non-zero if any flags set in command; otherwise, zero.
WAIT	flag set non-zero if "W" option specified for a processor' otherwise, set zero.
LTYPE	= 0 for LISTALL = 1 for LIST
LRANGE	= 0 no index range specified = 1 proname specified = 2 for both indexes specified = 3 for one index specified
LNAMES	= -1 table names are not to be printed = 0 table names are to be printed
LTITLE	= -1 "P:NAME TABLE" title to be printed = 0 no title to be printed = 1 "PROCESSOR TABLES" title to be printed
LFIRST	set to first index specified
LLAST	set to second index specified (or first if only one is specified)
EHHCNT	character count for "EH". message set to 3.

SUBROUTINES

SCAN, SCANT	composed next field from command line
POST	posts error message
SEARCH	finds proname in P:NAME table and return index or returns to "notfound" exit
GFID	scans command line for fid
TCTEST	tests proname for TEL or CCI; sets flag TEL/CCI accordingly
XGRTEST	tests proname for XDELTA, RECOVER, ALLOCAT M:DUMLM or GHOST1 and exits on found or not (two returns)
S400	convert EBCDIC index to hex
S360	test proname for :PNN format
S430	Convert EBCDIC character to hex digit

ERRORS

"PRONAME REQUIRED"	A proname was not recognized for the REPLACE, ENTER or DELETE options.
"NO SUCH OVERLAY/PROCESSOR"	The proname does not exist in the P:NAME table
"ILLEGAL COMMAND OPTION"	general error message for several illegal command options
"ILLEGAL FLAG COMBINATION"	Characters other than J, S, M, D, P set or D and P set.
"DON'T SET FLAGS WITH MONITOR OVERLAY"	flag option not applicable to the ENTER/ REPLACE of an overlay.
"PROCESSOR/OVERLAY ALREADY EXISTS"	The proname specified with the ENTER already appears in the P: NAME table.
"DON'T USE COMMAND ON TEL/CCI"	Cannot ENTER/DELETE TEL or CCI.
"PROGRAM ERROR (SHOULDN'T HAPPEN)"	Logical "dead-ends" in program; should not occur after system is debugged.

"ILLEGAL PRONAME, NOT P:NN"	Processor defined as a public library does not have a proname in the correct format
"INCORRECT FID"	File name, account or password exceeds field limits
"WAIT OPTION IGNORED FOR MONITOR OVERLAYS	Message posted but option "W" is ignored and processing of command continues

DESCRIPTION

SYNTAX checks each field of the command line for the defined options.

There is a main logical path for each of the commands except for ENTER and REPLACE which are combined because of their similar command formats. Once the requirement of the command are satisfied, no other anlaysis takes place. (The end-of-line is not required to terminate the command scan.)

Each of the command options is tested in the order indicated below:

LIST and LISTALL	Command is checked for presence of a proname or index or index range. The index option allows the user to print specific slots in the P:NAME table. The proname option allows the printout of a particular processor slot. If proname = M:DUMLM, all slots containing that dummy name will be printed. A proname/index is not required.
DELETE	Routine requires a proname which is stored in PRONAME if it is found in the P:NAME table. Proname is checked for TEL, CCI, XDELTA, RECOVER, ALLOCAT, GHOST 1 and M:DUMLM. Deleting these "processors" is not permitted. The PERM option is processed if it is specified.
REPLACE AND ENTER	Routine requires a proname which is stored in PRONAME. Proname is checked for XDELTA, RECOVER, GHOST 1, ALLOCAT and M:DUMLM which are not permitted. For ENTER, the proname TEL or CCI is not permitted. For REPLACE, proname must exist in P:NAME;

REPLACE AND ENTER (cont'd)

For ENTER, proname must not exist in P:NAME
The fid, if specified, must be preceded by "WITH" or "FROM"
If no fid is specified, PRONAME is used as the file name with the account specified as the user's account; no password is specified.
The "O" option is allowed in the ENTER command only
The "PERM" option may be specified for either REPLACE or ENTER
Flag settings are allowed for processors only
If the "P" flag is specified, the PRONAME must be of the format P:nn. If :Pnn is proname specified, "P" and "S" flags are set.
The "W" option is available for processors only.

Error messages are posted for illegal combinations or missing options. The routine exits to CLOSE.

The normal exit back to the calling routine is taken when no contradictory or illegal options are found in the command line.

UTS TECHNICAL MANUAL

ID

LIST

Design Specification

PURPOSE

To output to the user the contents of selected processor tables.

USAGE

BAL, 15 LIST	Called for by DRSPMAIN

INPUT

P:NAME	Table of processor and overlay names
PPROCS	Length of P:NAME table
PNAMEND	Index to last processor name + 1
PB:REP	Total number of users associated with processor
PB:HPP	Head of the physical page chain
PB:TPP	Tail of the physical page chain
PB:DSZ	Number of data pages
PB:DCBSZ	Number of DCB pages
PH:PDA	Disk address of first procedure page
PH:DDA	Disk address of first page of data and DCBs
PB:UC	Number of users in core using the processor
PB:LNK	Processor number of next overlay
PB:PVA	Virtual page number of first procedure page
PB:HVA	Virutal page number of first unused page
P:SA	Processor flags and start address

(The following flags are set in SYNTAX)

LTYPE	= 0 for LISTALL = -1 for LIST
LRANGE	= 0 no index range specified = 1 proname specified = 2 for both indexes specified = 3 for one index specified
LNAMES	= -1 table names are not to be printed = 0 table names are to be printed
LTITLE	= -1 "P:NAME TABLE" title to be printed = 0 no title to be printed = 1 "PROCESSOR TABLES" title to be printed
LFIRST	set to first index specified
LLAST	set to second index spec fied (or first if only one index is specified)

INTERACTION

M:PRINT	Output lines to user terminal/Printer
J:JIT	On-line/batch bit set

SUBROUTINES

FINDGRAN	Find RAD granules for slot specified
L400	Convert hex word to EBCDIC
L450	Convert byte to EBCDIC
SEARCH	Search P:NAME table for proname

DESCRIPTION

Two list options are implemented:

L [IST] [{ proname / #xx -YY }] which causes the routine to print portions/entire P:NAME table.

LISTALL [{ proname / #xx -YY }] which causes the routine to print selected processor tables for part/all of the processors listed in the P:NAME table.

Specification of a proname causes the tables for that processor to be printed. A special case is made of the proname M:DUMLM. Since slots with the name M:DUMLM are available for replacement/entering of a processor name, all such slots are printed to show the resources available to the user.

One index can be specified if the user wants to see the tables relating to a specific slot. Two indexes can be specified if the user wants to see several slots.

If neither a proname nor an index are supplied, the LIST option results in the printout of the entire P:NAME table. For LISTALL, some tables (named in section INPUT) associated with processors are printed.

The routine scans the P:NAME table for zeroed or "empty" slots (those which contain their own index). The contents of these slots are printed in hex format. Otherwise, the routine assumes that a processor name is stored in the slot and this is moved to the print line without modification. Monitor overlay, processor root and processor overlay slots are printed although only the monitor overlays and processor roots can be Replaced/Entered/Deleted. Processor overlay slots are filled/deleted along with their associated processor root.

UTS TECHNICAL MANUAL

ID

RADNEED

Design Specification

PURPOSE

To read the TREE record of the fid in order to determine the number of granules required to write the load module on the swapping RAD.

USAGE

The routine is entered with a BAL, 15 RADNEED.

INPUT

PAGES	Starting address of record read buffer
2PGERR	Set non-zero if pages are not available to read the TREE record (set in INITIAL)
TYPE	Set for processor (-1) or monitor overlay (+1)
FILE ACCN PSWD	Used to set file, account and password in open M:EI.
EIPLIST	P:LIST for open M:EI.

Procedure, data and DCB sizes read from TREE record.

OUTPUT

MAXRCD	The number of pages needed for the longest record of the fid
GRANEED	The number of granules needed
NOLAYS	The number of overlays if this is a processor

ERROR MESSAGES

CANNOT OPEN THE FID
FID IS NOT A LOAD MODULE
ONLY ONE LEVEL OF OVERLAYS FOR SHARED PROCESSORS
ONLY PROCEDURE IS ALLOWED IN A PROCESSOR OVERLAY

INSUFFICIENT MEMORY TO READ TREE
ILLEGAL PROTECTION TYPE FOR PUBLIC LIBRARY
INSUFFICIENT MEMORY TO READ MAX RECORD OF FID
MONITOR OVERLAY CANNOT HAVE OVERLAYS
OVLY DATA EXCEEDS RANGE 8000-8BFF

DESCRIPTION

The OPEN M:EIP-list is constructed out of the fid, the file is opened and the TREE record is read. The root's DATA size is bounded up to pages and the result stored in GRANEED. If this is a monitor overlay no further action is necessary since a monitor overlay (at the load module level); 1) is assumed to consist purely of DATA and 2) has not further overlays. Failure to meet these requirements results in appropriate error message.

If this is a processor, the root's PROCEDURE and DCB size are bounded up to pages. If this processor is not overlaid the three counts are totaled into GRANEED.

If the processor is overlaid, a check is made to insure that only one level of overlays exist and that each overlay consists of pure procedure (since shared processors must be of this form). The number of overlays is remembered, and the procedure of each overlay is bounded up to pages and totaled.

Special checks are made if a public library is being manipulated:

1) DCB and data size must be zero.
2) must have no overlays

Error messages are typed if the load module for a public library does not conform to this structure.

Since each granule on the RAD corresponds to a page in core, RADNEED must account for any difference between an overlay's procedure start and the next lower page boundary. (This difference actually represents the tail end of the root's procedure and is carried along on the RAD as the beginning of each overlay).

Throughout the processing, the length of the longest record is retained in MAXRCD in order to get enough pages for SYSMAK (and the file copying if PERM). The pages are obtained and RADNEED returns to DRSPMAIN.

UTS TECHNICAL MANUAL

ID

GETRADSLOT

Design Specification

PURPOSE

To find an available slot in the appropriate part of the tables and to determine whether the granules allocated to this slot are sufficient for the new item.

USAGE

GETSLOTRAD is entered from DRSPMAIN with a BAL, R15

OUTPUT

NINDEX	The new index for the replaced or entered item.

ERROR

INSUFFICIENT SPACE ON SWAP RAD

SUBROUTINES

FINDSLOT	searches P:NAME within the appropriate bounds for a 'M:DUMLM' slot.

INTERACTION

FINDSLOT	Find minimum sized dummy slot
POST	posts error messages
FINDGRAN	computes RAD granules associated with a slot

DATA BASES

TEL:CCI	TEL or CCI flag
PRONAME	proname
TYPE	monitor overlay or processor designator
GRANEED	number of granules needed

DESCRIPTION

FINDSLOT discovers an available slot and identifies if in NINDEX. Determining RAD availability consists of subtracting the pertinent granule number associated with the new slot (NINDEX) from the pertinent granule number of the next slot (for processors this is NINDEX+1; for monitor overlays it is NINDEX-1). Pertinent granule numbers are formed from the disc address in DDA or PDA for processors or monitor overlays, respectively. The subtraction is performed at the granule level, using subroutine DAGRAN to perform the conversions. If the new slot (NINDEX) happens to be the last slot in a section (monitor overlay or processor root), the "upper" disc addresses are obtained from slot 0, (DDA or PDA depending on processor or monitor overlay). Note: If the processor is CCI, the "upper" disc address is DDA of LOGON which is actually the adjacent processor on the RAD. This is not reflected in the tables. Having computed the number of granules available, GETRADSLOT makes a comparison between that number and the number needed (GRANEED). If there are not enough, an error message is posted and control passes to FIDREQ to print the size needed for the fid and exit to CLOSE.

ID

FINDSLOT

Design Specification

PURPOSE

To find an available slot in the appropriate part of the P:NAME table. If the item has overlays, FINDSLOT also determines whether there are sufficient overlay slots available.

USAGE

BAL, 11 FINDSLOT

OUTPUT

NINDEX	the index of the available slot

INTERACTION

HEX2PRNT	Convert hex to EBCDIC
CLEANUP	Release processor slots
POST	Post error message
FINDGRAN	Compute granules for a slot
WORTH	Test if possible slots are available

DATA BASE

TYPE	monitor overlay/processor designator
NOLAYS	number of overlays
P:NAME	core copy of P:NAME table

ERRORS

NO PRONAME SLOTS AVAILABLE
INSUFFICIENT OVERLAYS SLOTS
FID REQUIRES xxxx GRANULES

DESCRIPTION

FINDSLOT sets the bounds (LOW, HIGH) of the search for a new slot in P:NAME depending on whether the item is a monitor overlay or a processor. If it is a monitor overlay LOW is P:NAME+1 (slot 0 is special) and HIGH is P:NAME +2+ MAXOVLY. (P:NAME is double word table.) If this item is a processor, LOW is P:NAME +2 MAXOVLY and HIGH is P:NAMEND. The search is on an entry with the name = 'M: DUMLM'. A successful search results in the setting of NINDEX, the index to the found slot. No available slot results in an error and exit to CLOSE. A message " [FID] requires xxxx Granules" is typed to tell the USER how big a slot is required for the new load module. If an overlaid processor is being handled, the area from P:NAMEND to NXTPOVLY is searched for NOLAYS (number of overlays) slots. Not finding enough is an error. Otherwise FINDSLOT returns to GETRADSLOT.

A subroutine WORTH is used if the "W" is set in the command options. FINDSLOT determines if any slots are large enough for the writing of the new load module; if yes, the routine will wait for a slot to free up. If no slots are large enough, even when free, the error message is posted as in the non-'WAIT' case.

UTS TECHNICAL MANUAL

ID

TELCCIONLY

Design Specification

PURPOSE

Sets master mode.

USAGE

BAL, 15 TELCCIONLY

DATA BASE

UH:FLAG	Used to test for BREAK; used to test users associated
BREAK	Used to test for BREAK

SUBROUTINES

BAL, 12 MASTER	Sets Master Mode

UTS TECHNICAL MANUAL

ID

WRITESWAP

Design Specification

PURPOSE

To write the fid to the swapper and modify the core processor tables.

USAGE

BAL, 15 WRITESWAP from DRSPMAIN

INTERACTION

SYSMAK 1	writes fid and modifies tables (all entries except P:NAME and P:SA)

DATA BASES

NEWFLAGS	processor flags
NINDEX	new index
PAGES	buffer address
XS:CUN	current user number
XUH:FLG	user flag table
XP:SA	processor flags table

SUBROUTINES

BAL, 12 SLAVE	Set Slave Mode

DESCRIPTION

The routine stores the proper processor flags in the P:SA table, then sets up the environment for SYSMAK 1. The M:EI DCB is open and R7 and R6 contain the new index and buffer address. R8 contains a read buffer and address.

ERRORS

INCORRECT FID
WRITE RAD FILE I/O ERRORS
RAD OVERFLOW
SWAP I/O ERROR (QUEUE)
ILLEGAL LMN (LOAD BIAS CHECK)

UTS TECHNICAL MANUAL

ID

PERM

Design Specification

PURPOSE

To copy the fid to the :SYS account and modify the RAD versions of the shared processor tables if PERM was specified.

USAGE

PERM is entered from DRSPMAIN with a BAL, 15

ERROR MESSAGES

READ ERROR READING FID (COPY)
WRITE ERROR WRITING FID (COPY)
FILE STORAGE LIMIT IN SYSTEM ACCOUNT
DRSP M:BO ERROR (PERM)
CANT OPEN M:BO IN :SYS (PERM)
DRSP M:EI ERROR (PERM)

SUBROUTINES

POST 1	Post error message
MODRAD	reads, modifies and writes the RAD processor tables

DATA BASE

PERM	perm flag
TYPE	monitor overlay flag
PRONAME	processor name requested
EIPLIST	variable parameter list for M:EI
PAGES	buffer address
RCVRAD	disk address of monitor end

INTERACTION

M:SETDCB	Set error/abnormal exits in DCB
M:CLOSE	Close M:EI, M:BO
M:PFIL	Position file M:EI
M:READ	Read M:EI
M:WRITE	Write M:BO
M:PRECORD	Position record M:EI

DESCRIPTION

If a BREAK key, or no PERM was specified, PERM returns to DRSPMAIN. If a REPLACE or ENTER was specified, and the fid is a processor, it is copied sequentially to the :SYS account. During the copy, BREAK is checked. Regardless of the TYPE, MODRAD is called to modify the monitor's RAD tables. PERM returns to DRSPMAIN.

In case of errors during the copy into :SYS, an error message is posted but the routine returns to DRSPMAIN to allow at least a temporary ENTER/REPLACE of the proname.

UTS TECHNICAL MANUAL

ID

MODRAD

Design Specification

PURPOSE

MODRAD modifies the shared processor tables on the RAD if PERM had been specified.

USAGE

The routine is entered by a BAL, 15 MODRAD from PERM. R8 contains the disc address of the beginning of a monitor.

OUTPUT

The RAD copy of the shared processor tables is modified and rewritten to the RAD. Specifically, the proname slot in P:NAME and the new index slots for P:NAME, P:SA, PB:PSZ and PB:PVA are modified.

SUBROUTINES

SEARCH	locates the proname in the buffer
POST	posts error messages
DAGRAN	converts a disc address (in R8) to a granule number (in R9)
RWRAD	calls NEWQ to read or write a number of granules
BUFAD	find relative address in I/O buffer

DESCRIPTION

The first and last granule numbers of the RAD copy of the processor tables are computed and the tables are read via RWRAD into the I/O buffer pointed to by PAGES. SEARCH looks for the proname in the P:NAME table within the buffer. The name should be found if the operation is REPLACE or DELETE. The name should not be found if the operation is an ENTER. Error messages are posted through POST if these conditions are not satisfied but MODRAD exits normally. The word 'M:DUMLM' is placed in the proname buffer slot if the operation is REPLACE or DELETE and the proname is placed in the new index slot within P:NAME in the buffer. If a monitor overlay is being handled, P:PSZ and P:PVA are copied from the new slot core positions to the new slot positions in the buffer. If this is not a monitor overlay, only P:SA is copied. In all cases the tables are rewritten to the RAD via RWRAD and MODRAD exits.

RWRAD is entered with the appropriate function code, the first granule number and the number of granules to process.

ERRORS

PRONAME NOT FOUND ON RAD	DELETE or REPLACE had been specified
PRONAME FOUND ON RAD	ENTER had been specified
CAN'T MAKE PERM, NO RAD SLOTS	the RAD tables contain no M:DUMLM slots

DATA BASES

PAGES	I/O buffer
FC	I/O function code (0 for read, 1 for write)
GRAN1	first granule to process
NGRAN	number of granules to process

UTS TECHNICAL MANUAL

ID

RWRAD

Design Specification

PURPOSE

To read or write n granules on the system RAD.

USAGE

Preset	FC, GRAN1, NGRAN, PAGES as defined in DATA BASE
BAL, 15	RWRAD

INTERACTION

M:WAIT	Program pause
NEWQ	used to read or write the RAD
GMB	gets a monitor buffer for the end-action to NEWQ
RMB	releases the monitor buffer

SUBROUTINES

MASTER	Sets master mode
SLAVE	Sets slave mode
POST	Post error message

DATA BASE

FC	function code (=00 for read, 01 for write)
GRAN1	first granule to process
NGRAN	number of granules to process
PAGES	points to I/O buffer
MB:SDI	byte 0 contains DCT index for system RAD

ERRORS

DRSP I/O ERR/ABN (PERM) 10 attempts to read/write the RAD have failed

DESCRIPTION

RWRAD sets master mode and gets a monitor buffer to which the end action routine is moved. The relative granule and the registers are set for NEWQ. NEWQ processes 1 granule. At end action, the TYPC is stored. After NEWQ the TYPC is checked and the granule number and I/O buffer are updated. After the last granule has been processed, the monitor buffer is released, slave mode is restored and RWRAD returns.

ID

SWITCH

Design Specification

PURPOSE

To finalize the core version of the tables by entering the proname in the new slot and "erasing" the old slot.

USAGE

BAL, 15 SWITCH

DATA BASES

NINDEX	new slot index
OINDEX	old slot index
PRONAME	proname
TYPE	type of command

SUBROUTINES

MASTER	Set Master Mode
SLAVE	Set Slave Mode
CLEANUP	Release Processor slots

DESCRIPTION

If ENTER, put proname in new slot. If DELETE, put old index into old slot. If REPLACE, disable interrupts and do both. Enable interrupts and return.

RESTRICTIONS

Interrupts inhibited.

UTS TECHNICAL MANUAL

ID

CLOSEOUT

Design Specification

PURPOSE

To restore DRSP's original conditions prior to return to read next command.

USAGE

BAL, 15 CLOSEOUT from DRSPMAIN

DATA BASES

MAXRCD	max record size

INTERACTION

M:FVP	Free extra pages
M:CLOSE	Close file
M:SETDCB	Set error/abnormal exits in DCB

ERROR MESSAGE

DRSP I/O ERR/ABN (Close)

SUBROUTINES

POST1	Post error message

DESCRIPTION

Pages obtained must be freed. Close all open DCB's. Return to DRSPMAIN.

UTS TECHNICAL MANUAL

ID

CLEANUP
Design Specification

PURPOSE

Release inactive processor and overlay name slots back to the system.

USAGE

BAL, 11 CLEANUP

INPUT

P:NAME	Names of processors and overlays
PB:REP	Count of users associated with processors
PB:LNK	Table of links to associated processor overlays
PB:HPP	Head of processor page chain
PB:TPP	Tail of processor page chain

OUTPUT

P:NAME	Names of processors and overlays
PB:LNK	Table of links to associated processor overlays
M:FPPT	Monitor free page chain (tail)
M:FPPC	Monitor free page count

INTERACTION

M:PRINT	Used to print the slot index number and count of associated users.

SUBROUTINES

L450	Convert hex digits to EBCDIC
POST	Print error message
MASTER	Sets Master Mode
SLAVE	Sets Slave mode

ERRORS

# xx xx USERS	Printed if count in PB:REP has gone negative or has exceeded the maximum number of users.

DESCRIPTION

CLEANUP scans the P:NAME table for slots containing an index. If the corresponding PB:REP table entry has been counted down to zero, the name M:DUMLM is stored in the P:NAME entry and any associated overlay slots are set to zero.

CLEANUP sets master mode and inhibits interrupts in order to release any dedicated pages back to the monitor.

UTS TECHNICAL MANUAL

ID

SEARCH

Design Specification

PURPOSE

To search memory area specified for a two-word field equal to the one specified.

USAGE

R8 =	beginning address of table
R9 =	end address of table + 1
R10 =	address of two-word field to search for
BAL, 11 SEARCH	
Exit =	field not found
Exit =	field found
R8 =	table location at end of search (= end of table + 1 if not found)
R9 =	end address of table + 1
R10 =	INDEX (= 0 if none found)

DESCRIPTION

SEARCH compares a specified two-word field against a table of two-word fields (given a start and end + 1 address). The subroutine assumes that both the two-word field and the table are on a double-word boundary. The routine returns a table address and index if the two-word field is found. Returns an index of zero if the field is not found.

UTS TECHNICAL MANUAL

ID

GFID

Design Specification

PURPOSE

Decodes file name, account, password entered as part of input line and stores these fields as directed.

USAGE

R12 - Destination address of file name field
R13 - Destination address of account field
R14 - Destination address of password
BAL, 15 GFID
G505 - Flag cell set non-zero if any field exceeds the maximum number of characters allowed.

INPUT

CMDBUF - Input field stored in command line buffer.
BYTE - Character position in command line

OUTPUT

File Name - Maximum of four words stored at destination address specified.
Account - Maximum of two words stored at destination address specified.
Password - Maximum of two words stored at destination address specified.

DATA BASE

G514 Table of field delimitors

SUBROUTINES

SCAN, SCANT Pick up next field defined by given table of delimitors.

DESCRIPTION

GFID scans the command line buffer for the file name, account and password that might be entered there. If the file name is not specified in the command line, zeros are stored; if the account or password is not specified, the destination locations are ignored. If any field exceeds the maximum number of characters expected, the flag word G505 is set non-zero. The file name is stored in TEXTC format; the account and password are stored as TEXT. Scan for the account and password fields occurs only if the fields are separated by "period". Unused character positions are space filled.

UTS TECHNICAL MANUAL

ID

WORTH

Design Specification

PURPOSE

To check if processor table slots can become available.

USAGE

BAL, 15, WORTH

INPUT

TYPE	Type flag for processor/monitor overlay

OUTPUT

MAYBE	Set ≠ 0 if processor slot found
MAYBESLOTS	Set ≠ 0 if processor overlay slots found

DATA BASE

P:NAME	Processor name table
PB:LNK	Overlay link table

SUBROUTINES

FINDGRAN	Calculate granules assigned to a slot

DESCRIPTION

WORTH searches the P:NAME table for slots in the process of being released (they contain their own index number). When such a slot is found and if there are overlays associated, MAYBESLOTS count is incremented. FINDGRAN is used to determine if this slot can accommodate the size of the load module (GRANEED). If yes, the MAYBE count is incremented.

The search continues until the entire table is checked. It exits with the counts MAYBE and MAYBESLOTS set to the number of possible released slots and available processor overlay slots.

UTS TECHNICAL MANUAL

ID

FINDGRAN

Design Specification

PURPOSE

To compute the RAD granules on the swapper associated with a shared processor.

USAGE

BAL, 15 FINDGRAN

INPUT

R6 = address of P:NAME
R7 = processor slot number

OUTPUT

R8 = number of granules

DATA BASE

PH:DDA disc addresses of processors
PH:PDA disc addresses of monitor overlays

SUBROUTINES

DAGRAN converts a disc address to a granule number.

DESCRIPTION

PH:DDA or PH:PDA is selected as the table on the basis of slot number (in R7).

If R7< MAXOVLY, the lower disc address is obtained from PH:DDA in the slot pointed to by R7. The upper disc address is obtained from the next higher slot, i.e. (R7) - 1.

IF R7>MAXOVLY, the lower disc address is obtained from PH:DDA in the slot pointed to by R7. The upper disc address is obtained from the next higher slot, i.e., (R7) + 1.

In the event that R7 points to the last processor, the upper disc address is obtained from slot 0.

The routine gets the lower and upper disc addresses, converts them to granule numbers and subtracts, yielding the number of granules.

UTS TECHNICAL MANUAL

ID

SCAN, SCANT

Design Specification

PURPOSE

To transfer a field from the command buffer to input specified area.

USAGE

```
     BAL, 15   SCAN
or   BAL, 15   SCANT  (character count precedes field)
```

INPUT

```
 R8 =  deposit address
 R9    delimiter table address
R10 =  # of delimiters
R11 =  maximum number of characters

 BYTE  byte displacement into CMDBUF for next byte
CMDBUF command buffer
```

OUTPUT

R12 = 0 if field was within maximum number of characters. Otherwise, >0.

BYTE = is updated to point to next byte

ENDC contains last character of the field

LCF contains field length

DESCRIPTION

The deposit field is initialized with blanks. Leading blanks are ignored. Characters are transferred to the deposit area until a delimiter or a blank or the end of the command is encounted. Trailing blanks are ignored.

UTS TECHNICAL MANUAL

ID

POST

POST 1

Design Specification

PURPOSE

Post error message code for future reference.

USAGE

REGISTER 14: Contains error sub-code
BAL, 15 POST
BAL, 15 POST 1

SUBROUTINES

HEX2PRNT	Convert hex number for typeout
FULLERR	Print error message

DESCRIPTION

POST stores SR 1 and SR 3 and error sub-code for future printout. For the on-line user it types either the "EH @ n" or "EH" message. For the batch user it calls FULLERR to print the error message.

POST 1 sets a flag SR1SR3PO to signal FULLERR to print the SR 1 and SR 3 registers as part of the error message and goes to POST.

UTS TECHNICAL MANUAL

ID

DAGRAN, GRANDA

Design Specification

PURPOSE

To convert disc addresses to (from) RAD granule numbers.

USAGE

BAL, 15 { DAGRAN or GRANDA }

INPUT

R8 contains { disc address or RAD granule number } for { DAGRAN or GRANDA }

R9 contains { disc address or RAD granule number } for { GRANDA or DAGRAN }

DATA BASE

MB:GAM6	sector extraction mask
MB:GAM5	shift to get track
MB:GPT	number of granules per track

DESCRIPTION

DAGRAN: granule number = track x number of granule/track + $\frac{\text{sector}}{2}$

GRANDA: disc address = track number merged properly with sector number.

track number = quotient of $\frac{\text{granule number}}{\text{number of granules/track.}}$

sector number = 2x remainder of $\frac{\text{granule number}}{\text{number of granules/track.}}$

UTS TECHNICAL MANUAL

ID

TCTEST

Design Specification

PURPOSE

To test if the proname is TEL or CCI and set a flag accordingly.

USAGE

BAL, 11 TCTEST

INPUT

PRONAME	Processor/overlay name in TEXTC format.

OUTPUT

TEL:CCI	Set to non-zero if proname is either TEL or CCI: set to zero if neither.

DESCRIPTION

Tests the field PRONAME to determine whether or not it is equal to TEL or CCI. Sets the flag TEL:CCI to -1 if TEL, +1 CCI and zero if neither.

ID

XGRTEST

Design Specification

PURPOSE

To test if the proname is XDELTA, GHOST1, ALLOCAT, M:DUMLM or RECOVER and exit accordingly.

USAGE

BAL, 11 XGRTEST

INPUT

PRONAME	Processor/overlay name in TEXTC format (on a doubleword boundary)

ERROR

PRONAME IS ILLEGAL.

DESCRIPTION

Tests the field PRONAME to determine whether or not it is equal to XDELTA, GHOST1, ALLOCAT, M:DUMLM, or RECOVER. If yes, the routine exits to CLOSE; if no, the return is to the calling routine.

UTS TECHNICAL MANUAL

ID

BUFAD

Design Specification

PURPOSE

To return relative address of table location in the read-write buffer.

USAGE

R8	Monitor address
BAL, 15	BAL, 15 BUFAD
R9	Buffer address

DESCRIPTION

BUFAD computes the I/O buffer address of the Monitor address presented in register 8.

UTS TECHNICAL MANUAL

ID

MASTER

SLAVE

Design Specification

PURPOSE

To set up Master or Slave mode.

USAGE

BAL, 12 MASTER
BAL, 12 SLAVE

SUBROUTINES

POST	Post error message

DESCRIPTION

MASTER	Sets Master Mode using M:SYS. Checks to see if the user's privilege is high enough. If not, sends an error message and exits to ERREXIT.
SLAVE	Sets Slave Mode.

UTS TECHNICAL MANUAL

ID

S400

Design Specification

PURPOSE

To convert the EBCDIC index to hex representation

USAGE

R6 - Preset to user's value
R1 - Byte pointer to proname field
BAL, 15 S400
ERROR EXIT
OK EXIT
R6 - Result (maximum of last 7 digits)
R8 - Terminating byte
R1 - Resultant byte pointer to proname field

INPUT

PRONAME - Contains the index range to be converted to hex

SUBROUTINES

S430 - Convert EBCDIC character to hex digit

DESCRIPTION

Converts a series of EBCDIC characters representing an index, terminated by a "dash" or space character, to hex digits. If any of the characters are not hex the error exit is taken. When the maximum of seven digits is reached, the routine exits. In the normal case, digits are converted until a terminator is reached.

UTS TECHNICAL MANUAL

ID

S360

Design Specification

PURPOSE

To test if the proname is in the format of a public library name (:Pnn).

USAGE

BAL, 15 S360
"No" exit
"Yes" exit

INPUT

PRONAME Name of processor being modified

DESCRIPTION

S360 tests the proname for the following characteristics:

1. The name is four characters long starting with ":P".
2. The two digits nn are equal and $1 \leq n \leq 9$.

If the above characteristics are present the proname represents a public library and the routine takes the "yes" exit. If not, the "no" exit is taken.

UTS TECHNICAL MANUAL

ID

S 430

Design Specification

PURPOSE

Convert EBCDIC character to hex digit

USAGE

R8 - Input character in byte 3 position
BAL, 15 S430
R7 - Output character in digit 7 position or error flag

DESCRIPTION

S 430 converts the character presented in register 8 to its corresponding hex digit. The result (leading zeros and digit) is passed back to the calling routine in register 7. If the EBCDIC character has no hex equivalent, register 7 is set to -1 as an error flag.

UTS TECHNICAL MANUAL

ID

HEX2PRNT

Design Specification

PURPOSE

Convert hex to EBCDIC

USAGE

R4 number to be converted
BAL, 15 HEX2PRNT
R10, R11 results

DESCRIPTION

Converts 8 hex digits in R4 to 8 EBCDIC characters in R10 and R11.

ID

FSAVE Processor

PURPOSE

The FSAVE processor is designed to save files on tape at or near tape speed. FSAVE must be run under an account with CO privilege.

All tape record blocks written to tape will be 512 words or less. The label on tape cycle from PRG1 to PRG9 to be compatible with FPURGE. Tapes can be restored using either the FRES or FPURGE processors.

BASIC PHILOSOPHY

FSAVE bypasses system file and labeled tape management by branching directly to IOQ for all its I/O operations. This requires that FSAVE know the file management table structure (on disc) and that it simulate the file management and labeled tape portions of the monitor.

All the attributes of a file, including all dates, are saved in the :BOF sentinel and are restored by FRES. However, FPURGE does not preserve creation, modification, access or expiration date.

MODULE ANALYSIS

INITIATE

PURPOSE

This is the first module executed in the FSAVE processor. It gets pages from the Operating System for index buffers.

ENTRY

Entered in slave mode as first executable statement in FSAVE.

EXIT

ERROR EXIT
B SP:INIT (if not enough pages of core available)
NORMAL EXIT
B INITIATE1

OPERATION

An M:GP CAL is executed to get pages for index sector buffers. The number of pages received is saved in page total. If the requested number of pages are not readily available, the processor exits to SP:INIT.

Otherwise, the number of pages requested is saved in INDPAGES and the program exits to INITIATE1.

INITIATE1

PURPOSE

This module is responsible for computing and saving the word address of each index buffer in table IBUF.

ENTRY

B INITIATE1
(R1) = number of index buffers
(R9) = address of first page received from get page CAL.

EXIT

B GET:CC

OPERATION

Register 1 is used as an index into table IBUF for saving the Word address of each page received for use as an index buffer. Register 9 initially contains the word address of the first page

received. It is stored into the last entry of IBUF, then the next address is computed by adding IBUFSIZ to (R9), and this address is saved. This process is repeated until all index buffer addresses have been computed and saved in the IBUF table.

The routine then exits to GET:CC.

SPECINIT

PURPOSE

This module is entered if INITIATE or INITIATE4 is unable to get enough pages of core for the allocation of buffers. The routine releases all pages that it received and then requests 4 pages and allocates them for index, data, tape, and sentinel buffers.

ENTRY

B SPECINIT
PAGETOTAL = Number of pages received from Get Pages CAL.

EXIT

ERROR EXIT
B NOTENUFF (if at least 4 pages are not available)
NORMAL EXIT
B INITIATE5

OPERATION

A M:FP CAL is executed to release all pages previously obtained. An M:GP CAL is executed to ask for all pages available. If at least 4 pages are not available, the program exits to NOTENUFF.

Otherwise, a value of 1 is stored into DAPAGES and TAPAGES to set the number of Data buffer and Tape buffer pages to one. R9 contains the address of the first page received. This address is stored in DBUF to provide the address of the Data Buffer. R9 is then incremented by 512 to compute the next page address and this address is saved in TBUF for the Tape buffer address. The address in R9 is incremented again by 512 words and this address is stored in IBUF entry 1. The contents of R9 are again incremented by 256 and stored in entry 2 of IBUF

to provide the address of the second index buffer. A value of 2 is stored into INDPAGES to save the total number of 256 word index buffers available. The address is incremented again by 256 to compute the address of the next page. This address is saved in LIMIT and in CURPOS. The address is incremented by 512 and this address is saved in BUFTOP, the last buffer address.

The program then exits to INITIATE5.

INITIATE4

PURPOSE

This module is responsible for requesting pages from the Operating System for tape buffer and Data buffers and saving their addresses.

ENTRY

BAL, R7 INITIATE4
GETPAGES = Max Page request FPT (M:GP procedure).

EXIT

ERROR EXIT
B SP:INIT1 (if not enough pages are available)
NORMAL EXIT
B INITIATE5

OPERATION

Ask for one page, and if one is not available, exit to SPECINIT. Otherwise, increment PAGETOTAL and save the address of the page in LIMIT and CURPOS. The address of the top of the page is computed and saved in BUFTOP.

It then requests all possible pages. The total number received is added to PAGETOTAL. If at least 3 pages were not available it exits to SP:INIT1.

Otherwise, the address of each alternate page is stored into tables TBUF and DBUF, and the number of tape buffers, TAPAGES, and the number of data buffers, DAPAGES, are incremented until all pages are used or all entries in TBUF have been filled.

Any remaining pages are used for data buffers and their address is saved in DBUF table until that table is full. The number of data buffers is stored in DAPAGES.

The total number of tape buffers available - TAPAGES, is stored in the first entry of table TBUF. The total number of data buffers available, DAPAGES, is stored in the first entry of table DBUF.

The program then exits to INITIATE5.

INITIATE5

PURPOSE

This module is responsible for getting the date and time from the monitor.

ENTRY

B INITIATE5

EXIT

B 0,R7

OPERATION

An M:TIME CAL is executed to store the date and time from the monitor into words 3-6 of DATBUF. The date and time are converted to the appropriate format and stored in BKUPVLP+1 and BKUPVLP+2 (MMDDHHYY). The background lower limit is computed from the first address of the program and stored in word 8 of DATBUF.

RDCARD

PURPOSE

This module is responsible for reading control cards, interpreting control cards, and setting the appropriate flags. It calls on module RCD to read the cards and module INTER to interpret the cards and set the flags. It also sets up the header for the line printer according to the options specified and initializes the line printer output.

ENTRY

B RDCARD

EXIT

B NOIPRI

OPERATION

BAL on R14 to RCD read a control card. Print the card on the M:LL device. BAL on R15 to INTER to interpret the control card and set the appropriate flags. When INTER detects a +END card it returns skipping. Otherwise, RDCARD continues to call RCD to read and INTER to interpret cards.

The routine then sets up the header message, ejects a page on the M:LL device, and exits to NOPRI.

RCD

PURPOSE

This module reads a card from the M:C device, tests the card for +END in first 4 columns, sets word END to non-zero if +END card, and returns.

ENTRY

BAL, R14 RCD

EXIT

B *R14

OPERATION

Read one card through M:C device, if the card contains a +END in columns 1 through 40, RCD will set location END to non-zero.

If an error or abnormal return occurs from the READ card it sets location END to non-zero. (END will be tested in module INTER to exit the card read loop.)

The program then returns indirect on R14.

INTER

PURPOSE

The routine is responsible for interpreting control cards in INBUF and setting appropriate flags according to the options on the cards. It verifies the card commands and aborts on errors.

ENTRY

BAL,R15 INTER
INBUF contains the control card to be interpreted.

EXIT

NORMAL EXIT
B *R15 for normal control command
B *R15 + 1 if +END command

ERROR EXIT
M:XXX if invalid control command or card sequence error.

OPERATION

Checks are made on control command, and all the necessary flags are set. If an error is detected on any control commands, the message 'FASTPURGE CONTROL CARD ERROR' is put out and FSAVE aborts.

NOIPRI

PURPOSE

This module completes initialization, gets the tape mounted and initialized, and reads in the first account directory block.

ENTRY

B to NOIPRI

EXIT

B READFAIL 2 if unable to read Account Directory or if bad data is in Account Directory

NORMAL EXIT

B NACN

OPERATION

If the STATS flag is non-zero the module opens the STATS file as follows: File name = DISK-POOL, OUT mode. An abnormal or error return on the open causes a message "FILE MANAGEMENT ERROR FROM STAT FILE".

If the DUMP flag is non-zero, the program does a BAL,R10 to NEWREEL to get a tape mounted. It then does a BAL,R7 WRTDAT to write the DATE file on the tape.

The disc address of the Account Directory is found from word one of ACNCFU. The address is verified as valid by a BAL,R15 to DTOGRAN. If the address is bad the program branches to READFAIL.

The first sector of the account directory is read into ACBUF by a BAL,R15 to DISCIO. If a read failure occurs, the program branches to READFAIL2.

Otherwise, the account directory BLINK is picked up from Word 0 of ACBUF and saved in ACBLINK. The account directory FLINK is picked up from ACBUF + 1 and stored in ACFLINK. The account directory NAV is picked up from the first half-word of ACBUF+2 and stored in ACSIZE.

If the DIRLISTSW is set, the program does a BAL,R14 LISTAD to print out the account directory on the M:LL device.

The account directory BLINK from ACBUF is compared with LASTAC (in this case, it equals zero) and if they are not equal the program branches to READFAIL2.

The current account directory address from R8 is stored into LASTAC. A value of 12 is stored into NEXACN (the next index into the account directory) and into CURACN, (the current index into the account directory). The program then branches to NACN.

NACN

PURPOSE

This module fetches the next account directory entry from ACBUF, checks it for errors in format, determines if this account is to be processed, reads cards if necessary to determine the select option or skip option accounts, and branches to NOTLB after finding an account to process.

ENTRY

B NACN
ACBUF contains account directory sector
NEXACN points to next entry to be processed

EXIT

ERROR EXIT
B ADERROR if an error exists in the account directory

NORMAL EXIT
B ENDUP if all select card accounts have been processed or if end of account directory
B NOTLB if an account is found to be processed

OPERATION

The next account directory index is retrieved from NEXACN and stored into CURACN to update the current entry pointer. If all entries in this sector of the account directory have been processed, the program branches to ADDONE to read in the next sector.

The entry is verified to contain X'OB404040' in the first word and if it does not, the program branches to ADERROR.

Otherwise, if the flag "SELECT" is on and a +END card has been read, branch to ENDUP. If and END card has not been read, BAL,R15 to ACCK to determine if this account corresponds to the last SELECT account card read, and if so, read the next card. If this account number does not correspond to the last Select account card read, branch to NACN and process next entry.

If it does correspond, and the 'ALL' option has been specified BAL,R14 to READATA to read the next card, and branch to NOTLB to process the account. If 'ALL' was not specified, branch to NOTLB without reading another card.

If SELECT was not specified, determine if a +END card has been read, and if so, turn on the ALL flag to specify no SKIP, SELECT, START options and branch to NOTLB.

If an END card has not been read, check the SKIP flag. If SKIP has been specified BAL,R15 to ACCK to determine if this account is to be skipped. If it is not branch to NOTLB to process it.

If it is check the ALL flag to determine if all files in this account are to be skipped. If ALL is not specified, branch to NOTLB to process the account. If ALL is specified, VAL,R14 READATA to read the next data card, and branch to NACN to process the next entry.

If SKIP was not specified, test the STARTSET flag to determine if the +START option was specified. If it was not specified, set the ALL flag to minus one to indicate no SELECT, NO SKIP, NO STATT option and branch to NOTLB.

If +START was specified, BAL,R15 to ACCK to see if this account matches the account specified on the +START card. If no match, branch to NACN to process the next entry.

If this account does match the START account, reset the STARTSET flag, turn on the ALL flag, and branch to NOTLB.

NOTLB

PURPOSE

This routine prints the current account number on the M:LL device, reads in the next account directory sector if this is the last entry in current sector, gets the disc address of the file directory, verifies the address as valid, reads in the first file directory sector, lists it if necessary, and branches to GETFILE.

ENTRY

B NOTLB
ACN#DISP is displacement to Account Directory entry to be processed.
ACBUF contains current account directory sector.

EXIT

ERROR EXIT
B ADERROR if file directory first sector address is invalid.
B READFAIL3 if unable to read in a file directory sector or in case of a file directory BLINK failure.

NORMAL EXIT
B GETFILE

OPERATION

Move the account number from ACBUF to ACN#CURNT and space M:LL device to top of form and print "ACCOUNT#XXXXXXXX".

If this is the last entry in the current account directory sector, get the FLINK from ACFLINK and BAL,R15 to DTOGRAN to verify the address. If it is a valid address, BAL,R15 to QSECTOR to queue the next account directory block.

If this is not the last entry or if the FLINK is an invalid address, get the address of the file directory from the Account directory entry. Set LASTFD, ACNSIZE, and ACNGRAN to zero and BAL,R15 to DTOGRAN to verify the file directory address. If it is invalid, branch to ADERROR; otherwise BAL,R15 to DISCIO to read in the first sector of the file directory.

If a read error occurs, branch to READFAIL3. If the read is OK, get the BLINK from the first word of FDBUF and store in FDBLINK. Get the FLINK from the second word of FDBUF and store in FDFLINK. Get the NAV from first halfword of third word in FDBUF and store in FDSIZE.

If the list flag DIRLISTSW is set, BAL,R14 to LISTFD to list the file directory. If the BLINK is not zero for the first sector, branch to READFAIL3. Otherwise, store the current sector address in LASTFD and branch to GETFILE.

GETFILE

PURPOSE

This module gets the next file directory entry, picks up the disc address of the FIT, and reads the FIT into FITBUF.

ENTRY

B GETFILE
NEXFILE is index to next entry in FDBUF
FDBUF contains current file directory sector
FDSIZE contains the NAV for current file directory sector.

EXIT

ERROR EXIT
B FDERR if FIT address is not valid or if read error occurs when reading FIT.

NORMAL EXIT
B FILECHKS

OPERATION

Get the index to the next file directory entry and if the index equals the NAV, meaning the entire sector is processed, branch to FDDONE to get the next sector.

Get the disc address of the FIT for this entry and BAL,R15 to DTOGRAN to verify the address. If it is invalid, branch to FDERR. If it is valid, test to see if it is already in core. If not in core, is it already being read in. If so, wait until it's in.

Otherwise, BAL,R15 to DISCIO to read it into FITBUF. If a read error occurs, branch to FDERR. If the read is successful, or if the FIT is already in core, branch to FILECHKS. SYNFLAG is set if file descriptors indicate that this is a synonymous file.

FITCHKS

PURPOSE

This routine examines the FIT in FITBUF to determine if the FIT is valid. It tests to see if this file should be saved on tape, reads another SKIP or SELECT data card, if necessary, lists the FIT, and then branches to BLD:BOF. If this is the last file in the current file directory sector, it queues up to read in the next file directory sector.

ENTRY

B FITCHKS
FITBUF contains current fit to be processed.
CURFILE points to current file directory entry in FDBUF.

EXIT

ERROR EXIT
B FITSNAP if byte length of file name in FIT is zero or less.
B FITERR if file name in FIT doesn't compare to file name in file directory.

NORMAL EXIT
B BLD:BOF to process file.
B GETFILE if this file is not to be processed.

OPERATION

If the current file directory entry is the last entry in this sector and the FLINK is non-zero and valid, BAL,R15 to QSECTOR to queue the next file directory sector.

If the byte length of the name in the FIT is zero or less, branch to FITSNAP. If the byte length is valid, but the file name is not in EBCDIC branch to GETFILE to get the next file, and ignore this one.

If the file name in the FIT is valid, compare it with the file name in the current file directory entry. If there is no match branch to FITERR.

Otherwise, test the ALL flag and if it is on, meaning all files are to be saved, branch to BLD:BOF. If ALL is not set, compare the FIT file name with the file name contained in Column 14 then, BAL,R14 to READATA to read a card and follow with a BAL,R14 to READATA to read a card and follow with a BAL,R15 to INTER to interpret the card. If the card specifies a new account, set ACEQU flag to zero. If the SELECT mode is not set branch to GETFILE to process the next file.

If the SELECT mode is specified, branch to BLD:BOF to process the current file.

If the file names above do not match and the SKIP flag is not set, branch to GETFILE. Otherwise, test the FITLISTSW flag and if it is on, BAL,R14 to LISTFIT to list the FIT. Then, branch to BLD:BOF.

BLD:BOF

PURPOSE

This routine clears all index buffers, (IBUF), clears out the STACK, (DSTACK), builds a FIT record for tape, constructs a :BOF record, checks the file against the SAVE BY DATE option, writes a :BOF record on tape if necessary, writes a SYNON record if necessary, and finally branches to GETMIX.

ENTRY

B BLD:BOF
FITBUF contains FIT of file to be processed.

EXIT

ERROR EXIT
B FITERR if no file size entry is available in the FIT or if the MIX address contained in the FIT is invalid.
B FITSNAP if no '09' entry is available in the FIT.

NORMAL EXIT
B GETFILE if file is not to be saved because of SAVE BY DATE option or SAVE BY HOUR option.
B FILEDONE if no tape is being written.
B RANFILE if file is RANDOM.
B GETMIX to save the keys and records.

OPERATION

If the file is not a SYNONYMOUS file, (no 'OB' entry was found), initialize and free up all index buffers and clear the Stack.

Look for an 'OC' entry, branch to FITERR if none is found. The next word in FITBUF after the 'OC' entry is the FDA of address of the MIX. Save this in FDA. BAL,R15 to DTOGRAN to verify the address. If it is invalid, branch to FITERR.

Otherwise, test to see if a SAVE BY DATE/HOUR is being done. If not, set up to read ahead the first MIX sector so it will be in core when it is needed.

Clear the tape label and print buffers. Search for an '09' entry in the FIT by BAL,R15 to CODESCAN. If none is found, branch to FITSNAP. Otherwise, get organization type and save in ORG. Get the maximum key length and save in KEYM.

Search for an 'OD' entry by a BAL,R15 to CODESCAN to see if there is a CREATION DATE.

If this is a SYNONYMOUS file transfer the synonymous name to the :BOF record.

If it isn't SYNONYMOUS BAL,R15 to QUEMIX to queue up the first MIX read.

Construct a :BOF record. Construct a TLABEL record, include the account number, password, READ Accounts, WRITE accounts, ORGANIZATION, KEYMAX, and GRANULE count.

If a creation date exists and the SAVE BY DATE/HOUR flag is on, test to see if this file should be saved. If this file is not to be saved, branch to GETFILE.

Otherwise, store the creation date in the tape label record. Get the PBS flag (previous block count) and the size (size of last record) to zero.

Next, a read is queued up to read the FIT from the next file directory entry into core.

If a DUMP has been specified, BAL,R15 to BOFQUE to write the :BOF record and the TLABEL record to tape.

If a RANDOM file is being processed, branch to RANFILE.

If a SYNONYMOUS file is being processed, write a SYNON record and a :EOF record to tape and return to FILEDONE.

If a NULL file exists or a RANDOM file with no data, write a :EOF record via B to CKTIO. (Note: CKTIO will also branch to a routine to print the file information on the M:LL device if specified.)

If the file is neither NULL, RANDOM nor SYNONYMOUS, branch to GETMIX.

GETMIX

PURPOSE

This routine clears and releases all the data buffers and then clears the disc address table. It gets the disc address of the first MIX and reads it in printing the mix if necessary. It calls GETKEY to READ in the data records, queueing up as many Reads as possible for MIX sectors

and data granules. It calls the MOVEKEY and MOVE routines to transfer the records to the tape buffer. When the tape buffers are full, it writes them to tape. When all records have been processed, it branches to FILEDONE.

ENTRY

B GETMIX
FDA contains address of current MIX sector.

EXIT

NORMAL EXIT
B FILEDONE after entire file has been processed.

OPERATION

The program sets all data buffers free and clears the disc address table. It determines the first MIX sector address on the disc from FDA and checks to see if it is already being read in by comparing it with table INDEXDA. If it is not in the table, the program does a BAL,R8 to QUEMIX to read it in core.

If it is in the table, it tests bit zero of the correct table entry to see if it is already in core. If bit zero is a one, it loops until the bit goes to zero, meaning that it is in core. It then picks up the address of the sector in core from the parallel entry in the IBUF table and stores that in MIXBUF. It initializes the MIX displacement in CURRMIX to point to the first entry in the MIX. It picks up the NAV from word two of the MIX in MIXBUF and saves it in MISIZE.

It then BAL's on R11 to GETFOUR to read in data granules from the disc. If the INDEX flag is on, it does a BAL,R14 to LISTMIX to print the MIX sector on the LL device.

It does a BAL, R15 GETTBUF to get a tape output buffer to put the records in and saves the address of the buffer in CURBUF.

It does a BAL,R15 to GETKEYI to initialize the previous block size in the tape buffer. It does a BAL,R15 to GETKEY to get a key from the index buffer, does a BAL,15 to MOVEKEY to move the key to the buffer, and if it is writing a tape, it does a BAL,R15 to MOVE to move the data record into the buffer.

It does a BAL,R7 to 'SCHEDULE' to get the required buffers to read ahead for other data blocks. It does a BAL,R11 to GETFOUR to read in additional data blocks. It tests the flag MIXEOF to determine if all MIX for the file have been processed. If all entries have not been processed (MIXEOF is zero), it continues to move each key and data record to the tape buffer and write the tape buffer out when its full.

If all MIX entries have been processed, it branches to QUEREC to write out the tape buffer and then branches to MIXEND.

QUEREC

PURPOSE

This routine calls MTIO to write out a tape buffer.

ENTRY

BAL, R15 QUEREC
CURBUF contains address of current tape buffer.
KEYDISP has number of bytes in the buffer.

EXIT

NORMAL EXIT
B MTIO
R15 has address to branch to after tape record has been queued.

OPERATION

The program gets the current key displacement from KEYDISP and rounds it up to the nearest word. It tests a flag called 'BLOCKS' to determine if all tape records are to be dumped to the printer. If they are not, it branches to MTIO with R15 unchanged. MTIO will then return to the address in R15.

If records are to be listed, it saves R15 in R14 and loads R15 with the address of LISTOUTBUF routine before branching to MTIO. MTIO will then branch to LISTOUTBUF which will return to the original address in R15.

GETKEYI

PURPOSE

This routine sets the previous block size into a tape buffer and sets the displacement into the buffer to four.

ENTRY

BAL,R15 GETKEYI
RBS contains previous block size from last record.
CURBUF contains address of buffer to store PBS into.

EXIT

B *15

OPERATION

The program gets the previous tape block size from PBS and stores it into the buffer pointed to by CURBUF. It then sets KEYDISP to four so that the buffer index now points past the previous block size.

It returns on Register 15 to the calling program.

MOVEKEY

PURPOSE

This module moves the current key to the current blocking buffer.

ENTRY

BAL,R15 MOVEKEY
KEYDISP contains current key displacement.
CURBUF contains address of current tape buffer.
MIXBUF contains address of mex buffer.
CURMIX contains index into MIX buffer.

EXIT

NORMAL EXIT
B *15

OPERATION

The routines increment the first word of the tape buffer to update the number of keys. It gets the maximum key length for this file from KEYM and increments it by one. It gets the current key displacement from KEYDISP, rounds it up to the nearest word, and saves it in LASTKEY. It then moves the mix record from MIXBUF to CURBUF at the appropriate displacements. It saves the new MIX displacement on KEYDISP, it sets P1 flag to X'100' to indicate this is the first appearance of this key.

It then returns to the calling program.

MOVE

PURPOSE

This routine moves a data record to a tape blocking buffer.

ENTRY

BAL,R15 MOVE
R1 contains the output buffer size in bytes.
CURBUF contains the byte address of the tape blocking buffer.
CURDBLK contains the byte address of the input buffer.
KEYDISP contains the output buffer displacement.
BLDISP contains the input buffer displacement.
RWS is the number of bytes to be transferred.

EXIT

ERROR EXIT
B FAILURE if the granule pointed to by CURRB doesn't compare to the granule pointed to by the current disc address key GRANULEADR.

NORMAL EXIT
B *15 if data was not all moved.
B *15(+1) exit skipping if no bytes moved or if all data is moved successfully.

OPERATION

The program checks RWS to see how many bytes to move. If it is equal to zero, it exits skipping.

Otherwise, it compares the disc address in CURRB with the disc address in GRANULEADR. If they are not equal, it takes an error exit to FAILURE.

It increments the return address and moves as many data bytes as possible into the output buffer. If all cannot be moved, it decrements the return address, updates the granule accounting table RBHIST with the number of bytes moved, sets DATA$SW if buffer can be released, and returns. If all bytes can be moved successfully, it updates the granule accounting table and exits skipping.

GETKEY

PURPOSE

This module gets the next key from the MASTER INDEX of the file currently being processed.

ENTRY

BAL,R15 GETKEY

EXIT

ERROR EXIT
B FDERR1 if error in Read of MIX
B MIXERR if RWS in new MIX is zero.

NORMAL EXIT
B D when last record has been processed
B *R15

OPERATION

If the file organization is consecutive, control is transferred to GETKEYN. Otherwise, the routine reads in the next mix sector as required. It extracts the key from the MIX, queues up a read for the corresponding data block if it is not already in core, then attempts to queue up a read for the next MIX sector so it will be in core when needed.

It exits by a branch to D if the last mix record has been processed.

Otherwise, it returns to the calling program via R15.

If an error occurs while reading the MIX sector, it branches to FDERR1. If an error occurs in processing the MIX entry, it branches to MIXERR.

GETKEYN

PURPOSE

This module performs the function of GETKEY for consecutive files.

ENTRY

B GETKEYN

EXIT

B GETKEY5 if blocked segment.
B GETKEY01 if unblocked segment.
B CHKEND2 if at EOF.

OPERATION

The routine processes the next segment control word for a blocked or unblocked record segment. It stores the appropriate values for FAK, CSET, RWS, BLDISP, CURDBLK, GRANULEADR, etc. and increments the tape key for consecutive files, CONKEY. This is done in such a way that the MOVEKEY and MOVE routines can process consecutive files or keyed files in exactly the same manner. This routine is entered from within GETKEY and its three exit points are also within GETKEY.

SCHEDULE

PURPOSE

This module releases data buffers when they have been processed.

ENTRY

BAL,R7 SCHEDULE

EXIT

B *R7

OPERATION

The routine checks each entry of the RB1 table to see if it is currently in use. If the entry is zero, it insures that the corresponding entry in DBUF is free.

If it finds the address it releases the buffer in IBUF and checks to see if there is a FLINK.

If none exists, it branches to CKTIO.

Otherwise, it checks to see if the FLINK MIX is in INDEXDA. If it is not, it branches to MIXRATERR. If it is in the table, it waits until it is in core, makes sure the BLINK is valid, saves the address of the next sector in MIXBUF, initializes the displacement to 12, does a BAL,R15 to QUEMIX to read in the sector, and branches to NXTSECTR.

If the BLINK does not compare with the previous sector address, the program branches to READFAIL5.

QUEMIX

PURPOSE

This routine queues up a read for the next MIX sector, with END ACTION address set to MIXENAC.

ENTRY

BAL,R15 QUEMIX

EXIT

ERROR EXIT
B READFAIL5 if FLINK address is not valid.

NORMAL EXIT
B *R15

OPERATION

The routine requests an index buffer. If none is available, it exits.

If it gets a buffer, it picks up the FLINK of the MIX from NXTFLINK. If the FLINK equals zero, it exits.

It verifies the disc address of the FLINK and if it is invalid, it branches to READFAIL4. If it is valid, it branches to DISCIO2, with the return register set to the original calling routine, to queue up a read of the next mix. ENDACTION address is set to MAXANAC.

RANFILE

PURPOSE

Writes tape records for RANDOM files.

ENTRY

B RANFILE

EXIT

ERROR EXIT
B MIXERR1 if the disc address of the data granule is not valid, i.e., a matching HGP cannot be found.
B MIXSNAP if the disc address is not valid.

NORMAL EXIT
B CKTIO when entire file has been processed or if no tape is being written.

OPERATION

It checks DUMP flag to see if tape is being output, and if not, it stores the number of bytes in the record file size and branches to CKTIO.

If tape is being written, it searches the HGP to find an HGP corresponding to the data block address. If it cannot find a matching HGP, it branches to MIXERR1. Otherwise, it saves the file size in RSTORE and releases all data buffers in DBUF.

If RESTORE is zero, it branches to CKTIO. If RSTORE is non-zero, it gets a data buffer, verifies the disc address, and queues up a disc read for each 2048 character block in the RANDOM file. It then writes each record on tape. When all records have been processed, it branches to CKTIO.

GETTBUF

PURPOSE

This routine gets a tape output buffer.

ENTRY

BAL,R15 GETTBUF

EXIT

B *R15 condition codes set non-zero if buffer found. R1 is index into TBUF table for available entry.

OPERATION

The routine searches for an available tape buffer, by scanning table TBUF until an available entry is found. The index into TBUF is returned in R7. Condition codes are set non-zero if a tape buffer is found.

GETIBUF

PURPOSE

Gets a disc input buffer for MIX.

ENTRY

BAL, R15 GETIBUF

EXIT

B *R15 condition codes set non-zero if buffer is found. R1 is index into IBUF table for available entry.

OPERATION

The routine searches table IBUF for an available index buffer. The index to the available entry in IBUF is returned in R7. Condition codes are set non-zero if a buffer is found.

GETFOUR

PURPOSE

This routine performs read ahead for data granules from the disc.

ENTRY

BAL, R11 GETFOUR
DSTACK contains addresses of granules to be read.

EXIT

ERROR EXIT
B DATAERR if a bad disc address is encountered.
NORMAL EXIT
B *R11

BUILD

PURPOSE

This routine pushes the address of all data granules specified in the Just Read index sector into DSTACK. If the file organization is consecutive, control is transferred to BUILDN.

ENTRY

BAL, R11 BUILD
R7 is index into IBUF to current mix sector.

EXIT

NORMAL EXIT
B *R11

BUILDN

PURPOSE

This routine pushes the address of all unblocked data record segments for consecutive files into DSTACK.

ENTRY

B BUILDN

EXIT

B BUILD31
B BUILD7 if no unblocked data record segments in this granule.

QSECTOR

PURPOSE

This routine queues up the disc address in R8 to be Read by branching to DISCIO2. FITENAC is specified as the end action address.

ENTRY

BAL, R15 QSECTOR
R8 equals disc address to be read.
R10 = Byte address of buffer.

EXIT

B DISCIO2
R15 points to exit for DISCIO2 routine to original program.

DISCIO2

PURPOSE

This routine is the RAD and DISC Pack handler routine for FSAVE. An entry at DISCIO2 specifies no wait on I/O. End action address is in R7.

An entry at DISCIO specifies normal I/O WAIT and no end action.

It calls for I/O by branching to monitors NEWQ routine.

ENTRY

BAL, R15 DISCIO2 specifies no wait, and end action address in R7.
BAL, R15 DISCIO specifies wait, and no end action.

R1 = I/O table index (used for End Action Information)
R8 = Disc address
R9 = Byte count
R10 = Byte address of buffer
R7 = End Action Address if specified
R15 = Return address

EXIT

B *R15

DENAC

PURPOSE

This is the DISCIO end action routine; it resets the busy flag RBUSY and saves the TYC information in DSTATUS.

ENTRY

BAL, R11 DENAC from IOQ

EXIT

B *R11

FITENAC

PURPOSE

This is the end action receiver for Account directory, file directory or file information table reads. It resets the busy bit in the flag specified by R14, decrements the outstanding I/O count, and saves the Disc status (TYC) information in DSTATUS.

ENTRY

BAL, R11 FITENAC from IOQ
R14 is address of busy flag for this operation.
R12 is TYC of this operation.
R11 is return address.

EXIT

B *R11 returns to IOQ.

MTIO

PURPOSE

This is the routine which queues up tape I/O for FSAVE. It does so by branching to the monitor's NEWQNWM routine.

ENTRY

BAL, R15 MTIO
R6 = 0, this is a normal call, the command list has the end-action-address.
R6 < 0 R5 has byte count, CURBUF is the word address of the buffer.
R6 > 0 No data XFER. R6 contains function code.
R7 Points to calling sequence registers for IOQ call.
R15 Has return address.

EXIT

B *R15

WTENAC

PURPOSE

This module is the end action routine for all tape data writes. It decrements the outstanding I/O count in OPCNT. It saves the status from R12 into TPSTATUS. It clears the busy bit in the appropriate entry of TBUF tables. If the TYC indicates end of reel, it switches output reels. If the TYC indicates unrecoverable tape error, it types a message to the operator and switches the output reels. Otherwise, it exits to the address in R11.

ENTRY

BAL, R11 WTENAC from IOQ.
R14 is index into TBUF table.
R12 is TYC.
R11 is return register.

EXIT

B *R11 return to IOQ.

SENTENAC

PURPOSE

This is the end action routine for tape sentinel writes. A tape error will cause a reel change. End of Reel Status is ignored.

ENTRY

BAL, R11 SENTENAC from IOQ.
R12 is TYC.

EXIT

B *R11

NEWREEL

PURPOSE

This routine is called to open a new output reel. It executes an open CAL, verifies the DCT index returned, rewinds the tape, writes a :LBL sentinel, an :ACN sentinel, and a tape mark.

ENTRY

BAL, SR3 NEWREEL

EXIT

ERROR EXIT
B DCTXERR if bad DCT index returned from Open.

NORMAL EXIT
B *SR3

WRTDAT

PURPOSE

This routine is called to write the DAT file on reel PRG1.

ENTRY

BAL, R7 WRTDAT

EXIT

B *R7

DCTXERR/OPNABN/OPNERR

PURPOSE

This routine is an error routine from DCB opens. It types OPNFAIL and does an M:SNAP of the M:EO DCB.

ENTRY

B DCTXERR
B OPN ABN
B OPNERR

EXIT

B IORUNDWN

GOEOR

PURPOSE

This is the routine used to handle end-of-reel conditions. It writes a tape mark, an :EOV record, another tape mark, an :EOR record, 4 more tape marks, and does a M:CLOSE for that tape reel.

ENTRY

BAL, R1 GOEOR

EXIT

B *R1

EOFQ

PURPOSE

This routine creates and writes a :EOF record onto tape.

ENTRY

BAL, R14 EOFQ

EXIT

B WRTMARK (WRTMARK then returns to original routine)

BOFQUE

PURPOSE

This routine creates and writes a :BOF record on tape.

ENTRY

BAL, R15 BOFQUE

EXIT

B WRTMARK

MOVEI

PURPOSE

This subroutine is used to transfer sentinel records into background records to be written to tape.

ENTRY

BAL, R14 MOVEI

EXIT

B *R14

WRTMARK

PURPOSE

This routine queues up a write tape mark operation.

ENTRY

BAL, R15 WRTMARK

EXIT

B MTIO+2

NOTENUFF

PURPOSE

This routine prints "NOT ENOUGH CORE TO RUN, LESS THAN 5 PAGES AVAILABLE" and then does an M:XXX.

ENTRY

B NOTENUFF

EXIT

B IORUNDWN

ENDUP

PURPOSE

This routine sets the ENDOFSET flag to end processing, closes the tape, and does an M:EXIT. If no tape is being written, it displays the run totals, closes the statistics file, and runs down I/O and does an M:EXIT.

ENTRY

B ENDUP

EXIT

CAL1,9 1

ADDONE

PURPOSE

This routine is entered when the current account directory block has been processed. It gets another block of account directory if possible. Otherwise, it branches to ENDUP if processing of all accounts is completed.

ENTRY

B ADDONE
ACFLINK has FLINK for account directory.

EXIT

ERROR EXIT
B READFAIL2 if FLINK address is not valid.

NORMAL EXIT
B ENDUP if FLINK is zero.
B GETAD to queue up another read of account directory.
B GETAD3 after waiting for next block to be read in.

ADERROR

PURPOSE

This routine is entered when an error is found in an account directory key. It types a message, dumps the block, skips to the next key, and exits.

ENTRY

B ADERROR

EXIT

B ADELETE

IORUNDOWN

PURPOSE

This routine loops until all DISC and tape operations are completed, then it returns.

DISPRUNTOTL

PURPOSE

This routine displays the final run statistics.

ENTRY

BAL, R15 DISPRUNTOTL

EXIT

B *R15

DTOGRAN

PURPOSE

This module checks a disc address to verify that (1) the DCT index is correct, (2) an HGP exists for that DCT index, and (3) the sector number is within range.

ENTRY

BAL, R15 DTOGRAN
R8 is disc address.

EXIT

ERROR EXIT
B *R15 if address is bad.

NORMAL EXIT
B *R15+1 if address is valid.

FDERR

PURPOSE

This routine is entered if an error is detected in a file directory key. The key is skipped, a message is typed, and the routine branches to FDLETE.

ENTRY

B FDERR

EXIT

B GETFILE

FITSNAP/FITERR

PURPOSE

The routine is entered if there is an error in the contents of FIT. An error message is typed, the file directory and FIT are listed, and the program branches to FDERR1.

ENTRY

B FITSNAP
B FITERR

EXIT

B FDERR1

LISTFD

PURPOSE

This routine lists the current file directory block.

ENTRY

BAL, R14 LISTFD

EXIT

B *R14

LISTFIT

PURPOSE

This routine lists the current FIT.

ENTRY

BAL, R14 LISTFIT

EXIT

B *R14

LISTOUTBUF

PURPOSE

This routine lists the current tape output buffer.

ENTRY

BAL, R14 LISTOUTBUF

EXIT

B *R14

LISTMIX

PURPOSE

This routine lists the current MIX block.

ENTRY

BAL, R14 LISTMIX

EXIT

B *R14

LISTAD

PURPOSE

This routine lists the current account directory block.

ENTRY

BAL, R14 LISTAD

EXIT

B *R14

MIXSNAP/MIXERR

PURPOSE

This routine is entered when a mix error occurs. It frees all tape buffers and branches to EOFQ to close out the file on tape.

ENTRY

B MIXSNAP
B MIXERR

EXIT

B FDERR1 if no tape is being dumped.
B EOFQ if no tape is being written.

DATAERR

PURPOSE

This routine is entered if a data granule address error has occurred. It types an error message and branches to MIXSNAP.

ENTRY

B DATAERR

EXIT

B MIXSNAP

FDDONE

PURPOSE

This module is entered when a file directory block has been completed. If this is not the last link, it reads the next link in.

ENTRY

B FDDONE

EXIT

ERROR EXIT
B READFAIL3 if FLINK address is invalid.

NORMAL EXIT
B GETFD1 if next block already in core.
B GETFD to read in next block.
B ENDOFD if FLINK is zero.

ENDOFD

PURPOSE

Entered when a file directory is completed. It prints next header with new account on M:LL device after printing summaries for previous account. It then branches to NACN.

ENTRY

B ENDOFD

EXIT

B NACN

CKTIO

PURPOSE

This routine is entered when a file has been completely processed. It writes an :EOF record if necessary, prints the file name and the file information summary on M:LL device. It then branches to GETFILE to process the next file.

ENTRY

B CKTIO

EXIT

B GETFILE

CODESCAN

PURPOSE

This routine scans the FIT for the entry whose code corresponds to that in R2.

ENTRY

BAL, R15 CODESCAN
R2 contains code to search for.

EXIT

B *R15 if not found.
B *R15+1 if found.

MOVENTRY

PURPOSE

This routine moves an entry from the FIT to a :BOF record.

ENTRY

BAL, R15 MOVENTRY
RO is word address of entry in FIT.
R4 is address of :BOF record.

EXIT

B *R15

HEXTODEC

PURPOSE

This routine converts a number from hexadecimal EBCDIC.

ENTRY

BAL, R15 HEXTODEC
R3 is number to be converted.
R1 is address to store result.

EXIT

B *R15

TACNT/TFILNME

PURPOSE

These routine types "ACCOUNT" and an account name, or "FILE" and a FILENAME on the operator's console.

ENTRY

BAL, R15 TACNT to type account.
BAL, R15 TFILNME to type file name

EXIT

B *R15

ACCK

PURPOSE

Checks if current account number matches current data card. Sets ALL = -1 if all files mode, sets ACEQU = -1 if account number compares to card, sets ACEQU to zero if no compare.

ENTRY

BAL, R15 ACCK

EXIT

B *R15

READFAIL

PURPOSE

Entered because of an ACNCFU disc address error. Causes M:XXX abort.

ENTRY

B READFAIL

EXIT

B ENDUP

READFAIL2

PURPOSE

Entered because of an account directory LINK failure. Causes M:XXX abort.

ENTRY: B READFAIL2

EXIT: B ENDUP

READFAIL3

PURPOSE

Entered because of a link failure in the file directory. Causes a branch to ENDOFD to skip to next account.

ENTRY

B READFAIL3

EXIT

B ENDOFD

READFAIL5

PURPOSE

Entered because of a LINK failure in a MIX. Causes branch to MIXERR1 to skip to next file.

ENTRY

B READFAIL5

EXIT

B MIXERR1

LPRINT

PURPOSE

This routine prints one line and then clears the buffer.

ENTRY

BAL, R15 LPRINT

EXIT

B *R15

PLIST

PURPOSE

This routine is used to snap out a buffer on the M:LL device.

ENTRY

BAL, R15 PLIST
R1 contains number of bytes in buffer.
R3 contains beginning buffer address.

EXIT

B *R15

BUFSET

PURPOSE

This routine is called to move a print line from one buffer to another.

ENTRY

BAL, R15 BUFSET

EXIT

B *R15

PRINT

PURPOSE

Writes one line of print via a CAL1,1 to WRTPBUF.

ENTRY

BAL, R15 PRINT

EXIT

B *R15

SPACE

PURPOSE

Writes N lines of blanks on M:LL via CAL1,1 WRTBLNK where number of blanks is supplied by R1.

ENTRY

BAL,R15 SPACE

EXIT

B *R15

TYPEIO/TYPEIO2

PURPOSE

This routine types a message on the operators console.

ENTRY

BAL,R15 TYPEIO
R1 is byte address of TEXTC message

BAL,R15 TYPEIO2
R1 is byte address of message buffer
R3 is byte count

EXIT

B *R15

UTS TECHNICAL MANUAL

ID

FRES PROCESSOR

PURPOSE

The FRES Processor is designed to restore files from tapes created by the FSAVE processor. FRES must be run under an account with CO privilege.

FRES relies on the fact that FSAVE never writes tape blocks larger than 512 words; therefore, FRES should not be used to restore tapes written by FPURGE.

CODING CONVENTIONS

FRES was coded in a highly subroutinized manner.

Register 0 is the link register.
Registers 1 through 6 are nonvolatile.
Registers 7 through 15 are volatile.

BASIC PHILOSOPHY

FRES bypasses sytem labeled tape management by branching directly to IOQ for its tape operations. For creating files, however, FRES uses standard system services (i.e., M:OPEN, M:WRITE, M:CLOSE).

FRES restores all of the attributes of a file that were saved by FSAVE including all applicable dates.

ID

QTAP

PURPOSE

Call MTREAD to queue the next physical tape read.

USAGE

BAL,0 QTAP

INPUT

NXTBUF contains the index of the next tape buffer to be used.
TSTATUS a byte table indicating the status (or TYC) of each tape buffer.

DESCRIPTION

Check the status of the buffer to ensure that it is free (TSTATUS = 0). If it is not free, return to the calling routine. Otherwise, mark the buffer busy (TSTATUS = FF) and call MTREAD. When MTREAD returns, update NXTBUF so that it points to the next buffer and return to the calling routine.

ID

MTREAD

PURPOSE

Call NEWQNWM in IOQ which performs (queues) the actual read.

USAGE

BAL,0 MTREAD

INPUT

R7 contains the index of the tape buffer.

DESCRIPTION

Enter master mode (M:SYS) and increment UB:MF by one. Call NEWQNWM to read 2048 bytes into the appropriate buffer with the end action address equal to the physical address of EAREAD and end action information equal to the physical byte address of the STATUS entry for the appropriate buffer. When NEWQNWM returns, reenter slave mode and return to the calling routine.

ID

OPNTAP

PURPOSE

Open M:EI to device tape with the correct serial number.

USAGE

BAL,0 OPNTAP

INPUT

OPTPFPT the FPT used to open M:EI.

DESCRIPTION

Check M:EI to see if it is open. If so, allow I/O to run down (UB:MF = 0), close M:EI (M:CLOSE), change the serial number in OPTPFPT, and open M:EI to the next volume (M:OPEN). If M:EI is not open, then merely open it (M:OPEN) without changing OPTPFPT. After opening M:EI, save the DCT index in MTDCTX, rewind the tape (M:REW) and initialize CURBUF, NXTBUF and TSTATUS. Then return to the calling routine.

ID

SKPTMK

PURPOSE

Logically read physical records until encountering a tape mark.

USAGE

BAL,0 SKPTMK

INPUT

CURBUF contains the index of the current tape buffer.
TSTATUS a byte table indicating the status (or TYC) of each tape buffer.

DESCRIPTION

Check the status of the current buffer, returning to the calling program if a tape mark is indicated (TYC = 6). Otherwise, call NEXTBUF to update CURBUF to the next buffer and repeat the above test until it is satisfied.

ID

NEXTBUF

PURPOSE

Updates CURBUF to the next tape buffer.

USAGE

BAL,0 NEXTBUF

INPUT

CURBUF contains the index of the current tape buffer.
TSTATUS a byte table indicating the status (or TYC) of each tape buffer.

DESCRIPTION

Free the current tape buffer (TSTATUS = 0), update CURBUF to the next buffer in sequence, check to ensure that it is neither busy (TSTATUS = FF) nor free (TSTATUS = 0), and return to the calling routine. If the buffer is busy, spin until it is no longer busy. If it is free, force a read by calling QTAP and recycle through the above status checks.

ID

INITIAL

PURPOSE

Perform appropriate initialization of tables and data.

USAGE

BAL,0 INITIAL

INPUT

#BUF number of tape buffers.
#PAGES initial size in pages of the data buffer.

DESCRIPTION

Enter master mode (M:SYS), set JIT limit for maximum run time to unlimited (= 0), set JIT limits for temporary and permanent RAD and disc to 128K granules, and reenter the slave mode. Obtain pages (M:GP) for tape buffers and the data buffer. Save the logon account from JIT in MYACCT and return to calling routine.

ID

CCI

PURPOSE

Process control cards and data cards.

USAGE

BAL,0 CCI

DESCRIPTION

Read control cards through M:C and process as follows:

a. +END - return to calling routine.

b. +VOL - read data card and store serial number in OPTPFPT (see OPNTAP).

c. +START - read data card and store account in STACCT and file name, if present, in STFILE.

d. +SKIP - read data cards (up to #SKIP allowed) and store accounts in the doubleword table, SKIP. The data cards must be sorted so that the accounts are in ascending order.

ID

SKP:BOF

PURPOSE

Logically read physical records until encountering a :BOF, :EOV or :EOR sentinel.

USAGE

BAL,0 SKP:BOF

DESCRIPTION

Call SKP:TMK to locate the next tape mark, read the next record and check to see if it is a :BOF, in which case return to the BAL+2. If not a :BOF, check to see if it is an :EOV or :EOR, in which case return to BAL+1. If none of the above, repeat the entire process.

ID

GOT1

PURPOSE

Open the disc file corresponding to the next tape file.

USAGE

BAL,0 GOT1

INPUT

CURBUF contains the index of the current tape buffer which contains the :BOF from tape.
OPNFLFPT the FPT used to open the file.

DESCRIPTION

Reinitialize data cells. For each variable length parameter (VLP) entry that must be preserved from the original file, call GETVLP which moves the VLP entry from the :BOF to OPNFLFPT. For VLP entries that were not present in the original file (e.g., PASSWORD, READ accounts, etc.) an appropriate dummy entry is used so that the entry from the previous file which is in the DCB is not carried across to the new file. Read the user label which is the record following the :BOF and call ACCTOJIT to move the account from the user label to JIT. Check the file

organization in the user label for RANDOM and perform special setup if it is RANDOM. Next, call CHECKIT to determine whether or not to restore this file. If not, return to the BAL+1. If so, open the file (M:OPEN) using OPNFLFPT and return to the BAL+2.

ID

CHECKIT

PURPOSE

Determine whether to open on disc or skip the current tape file.

USAGE

BAL,0 CHECKIT

INPUT

CURBUF contains the index of the buffer containing the user tape label.

DESCRIPTION

If starting account/file was specified, determine whether or not we are there yet. If not, return to the BAL+1. If skip accounts were specified, see if this is one of them. If so, return to the BAL+1. If none of the above, check date in user label to ensure that tape files are in sequence. If no sequence error, return to BAL+2. Otherwise, notify user of file sequence error and exit.

ID

ACCTOJIT

PURPOSE

Move account from user tape label to JIT.

USAGE

BAL,0 ACCTOJIT

INPUT

CURBUF contains the index of the buffer containing the user tape label.
LASTACCT contains the account of the last file that was processed.

DESCRIPTION

If the account in the user label and the account in LASTACCT are equal, return to the calling program. Otherwise, move the account from the user label to LASTACCT and also to JIT. Set J:FDDA = 0 in JIT and return to the calling program.

ID

GETVLP

PURPOSE

Search a variable length parameter (VLP) list for a specified entry and, if found, move the entry from the VLP list to a specified location.

USAGE

BAL,0 GETVLP

INPUT

R4 contains the word address of the source VLP list.
R5 contains the byte address of specific entry"s destination.
R6 contains the VLP for the entry to be found and moved.

DESCRIPTION

Search the VLP list which begins at the word address in R4 for the VLP code in R6. If not found, return to the BAL+1. If found, move the entry from the VLP list to the byte address in R5 and return to the BAL+2 with R5 pointing to the first byte address following the entry's destination. In addition, if the VLP code is for a file name (VLP code = 01), move the entry to LASTFILE.

ID

BUILD

PURPOSE

Transfer logical data records from the labeled tape to the disc file.

USAGE

BAL,0 BUILD

INPUT

CURBUF contains the index of the tape buffer containing the user tape label.

DESCRIPTION

Process the tape blocks until the :EOF sentinel is encountered. If a fatal error occurs before the :EOF, snapshots are performed and we return to the BAL+1. TSTATUS is checked for each tape block to ensure that TYC = 1 for data blocks. If an :EOV is encountered, OPNTAP is called to close the current volume and open the next volume which is positioned to the beginning of the data and we continue. When processing the data blocks for KEYED or CONSEC files, write directly from the data block to the file (M:WRITE) whenever the data is blocked in a single tape block. When the data is chained between two or more tape blocks it is first moved to the data buffer, DBUF, prior to writing the file. If DBUF is not large enough, additional pages are obtained (M:GP). Inability to obtain enough pages for a record is considered a fatal error for this file. When the :EOF is encountered, return to the BAL+2. When processing the data for RANDOM files, always move the data to DBUF until it is full and then issue the M:WRITE.

ID

EXIT

PURPOSE

Terminate the run.

USAGE

B EXIT

INPUT

MYACCT contains the logon account for this job.

DESCRIPTION

Move the logon account from MYACCT to JIT. Wait for I/O to run down (UB:MF = 0) and exit (M:EXIT).

ID

FILL Processor

PURPOSE

FILL provides the ability to protect the file system and to control the amount of space available for files. File system protection is provided through scheduled operations that save files modified since the last save, or files saved through specific on-line user request. The entire file system can be restored following disasterous system failure or selected files can be restored. Available file space control is achieved through deletion of expired files and through operator initiated purges of the file system based on last reference of files.

MODULES

The FILL processor consists of three functional modules (BACKUP, FILL, and PURGE), a module of DCBs (FILLDCB); two subroutine modules that are also used by some other processors (JULIAN and MAILBOX) and the system definitions module (MONSTK). BACKUP, FILL, and PURGE are described herein; and the other modules are either described in other parts of the Technical Manual (JULIAN and MAILBOX), or do not require description (FILLDCB and MONSTK).

DESCRIPTION

When the FILL processor is initiated, whether automatically from GHOST1 at system initiation, through operator keyin of GJOB FILL, or through an on-line user using the command BACKUP, the entry is to the FILL module at location FILL. FILL determines if it is running as a ghost job (it aborts if not) and whether it is initiated by GHOST1 (permits the operator to wipe out FILL's "memory" if not). The operator is then asked whether he wants a fill, no fill, or an instant squirrel – resulting in continuation of FILL, branch to BACKUP in module BACKUP, or branch to SETSQ in module BACKUP. At SETSQ, a schedule is constructed indicating a squirrel run at the current time, then proceeds to do the backup. At BACKUP, the backup schedule is interrogated (see below). If a fill occurs, the starting tape serial number is obtained from the operator as in the "skip to" file if the operator wishes the file to start other than at the beginning of the tape. Files are then copied from tape to disk, automatically switching reels at the end of each tape. When the end of the tape set is reached, the operator is asked if there are more tapes to file. If yes, the above sequence is repeated. If no, FILL branches to BACKUP, as above. At the conclusion of restoring each file, FILL checks to see if ALLOCAT has set the flag indicating that file space is getting low, or if the operator wants to initiate a file purge. If either is true, FILL branches to PURGE in module PURGE. When PURGE completes its function, it returns (as a closed subroutine) to the FILL module and the fill completes.

At BACKUP, a check is made for selective fill commands. If there are any, FILL is entered at SELFILL, the commands are interpreted, and FILL restores the files. Upon conclusion, SELFILL returns (as a closed subroutine) to the BACKUP module. (SELFILL, like FILL, checks for purge flags and if set, calls PURGE.) BACKUP then examines the BACK:SCHED.:SYS file to determine if a backup is scheduled. If not, purge flags are checked, as above, and the FILL processor either puts itself to sleep until the next scheduled backup, or for 15 minutes, whichever is less. If a backup is scheduled, the entire file system is examined for files that have been modified since the last backup of the current type (SA, IN or SQ). Modified files are written to tape as encountered. After each file is inspected, the purge flags are checked, as above. At the conclusion of the scheduled backup, or immediately if there was no scheduled backup, the :BACKUP file (account :SYS) is checked for user or purge (see below) backup requests. Any such files are also copied to tape if not already done, and if the entry is flagged as a backup-and-delete (X'1F' VLP code) DOPURDEL in module PURGE is called to delete the file and log that fact. If the operation was a scheduled backup, or execution of a scheduled WRAPUP, the output tape, if any is dismounted, and as above, the job puts itself to sleep. When the job wakes up, whether from time elapsing, ALLOCATE waking it, a GJOB FILL, or user BACKUP command, the processor goes back to point BACKUP, etc.

Note: Normal FILL is entered only when the job is initiated.

PURGE is a closed subroutine that is called by FILL or BACKUP. A purge is initiated by the operator answering "yes" to the "expired file purge" question, or by the operator keying in a PURGE request. In either event, all files in the system are examined for expiration, and if expired they are purged (see below). For an expired file purge that is the only operation. For an operator-initiated PURGE-OLDER, files are also checked for last access, and if older than specified they are purged. For an operator-initiated PURGE-UNTIL, a table is built (currently 4 pages long) that is sorted by last access date. When all files have been examined, the table contains the oldest files in the system, based on access date, that are eligible for purge. Files are then purged until the required number of granules is available or the list is exhausted. In doing a purge, the account :SYS is not eligible, nor are files with the "no purge" descriptor set on a "NEVER" expiration. Otherwise, all expired files are purged as encountered unless the BACKUPALL SYSGEN parameter was set and the file has not been backed up since it was last modified. In this case, an entry is added to the :BACKUP.:SYS file. When the "backups" approach a page in length or at the end of the purge, the BACKUP module is entered at USR:BCK. BACKUP then copies the file to tape, encounters the X'1F' VLP and calls DOPURDEL in PURGE to delete the file, etc., until :BACKUP is exhausted and then returns to PURGE. For an operator-initiated purge, files not previously backed up are always backed up before deletion as the files are not expired. When a purge is complete, the new granule available counts are logged and PURGE returns from whence it was called. At that point in time there may be some files that have not yet been backed up but that are in :BACKUP. The number of granules of such files is logged as "in progress". Such files will be backed up and deleted before BACKUP puts the FILL job to sleep.

UTS TECHNICAL MANUAL

ID

BACKUP

PURPOSE

Backup copies files from the rapid access devices controlled by the file management system onto labeled tapes. During a system start or following a catastrophe, these tapes are used to restore the file system data base.

OVERVIEW

There is one ghost processor which includes automatic Backup, user-requested Backup, instant Squirrel, and Fill. Automatic Backup operates from a schedule established by the installation manager and it includes Saveall, Incremental, and Squirrel. The Saveall saves all files except account :SYSGEN and those files which were booted from the PO tape. Incremental saves all files which were created or modified since the last complete execution of Saveall, Incremental or Fill. Squirrel saves all files created or modified since the last execution of automatic Backup or Fill. User-requested Backup is initiated by a terminal user with a request to back up an individual file. Instant Squirrel is requested by the operator after a crash which resulted in an impossible recovery. The instant Squirrel is an attempt to save all recent files so that they can be restored after a boot from PO tape.

USAGE

The file BACK:SCHED, :SYS contains the schedule for automatic Backup. The format is TYPE = HH:MM, HH:MM. The TYPE is SA, IN, SQ, or WR. The time is a 24-hour clock. The BACK:SCHED file may be either keyed or sequential. More details of the scheduling may be found in the UTS Operations Manual - 90 16 75A.

TEL places the user Backup requests in file :BACKUP in the keyed record BACKUP. The format of the request is the file name . account . password as specified in the variable parameter list of OPEN or in the DCB. The name is code 01, the account is 02, and the password is 03. TEL then issues a wake up CAL for FILL.

At BACKUP the schedule is examined to see if automatic Backup is required. If it is, it is performed and any user requests from :BACKUP are processed and the ghost puts itself to sleep. If there is no scheduled request, any user requests are processed and the ghost puts itself to sleep. Subsequently, anytime the ghost is awakened because of a user request, sleep interval elapses, or an operator key-in of GJOBBACKUP, it first checks the schedule, then the user requests and then puts itself to sleep.

INPUT

The schedule file BACK:SCHED, :SYS is read and each request is scanned and sorted into a table by request time. Any requests which contain errors are deleted from the file.

The file :BACKUP, :SYS can contain two keyed records. The record BACKUP is written by the TEL command BACKUP. It contains the user's requests of the name, account, and password already formatted for the parameter list of OPEN. There is a one page limit set by TEL for this record. The record SAV contains the checkpoint information for automatic Backup. It contains the completion time of the Saveall, Incremental, Squirrel, the type of the last Backup, whether or not a tape is mounted, and the completion time of Fill. This record is read and written by automatic backup.

All files in all accounts are read by automatic Backup regardless of their password or read-access.

OUTPUT

The Backup tapes are created as multi-file, multi-reel, labeled tapes. Each user file is opened with FPARAM and the information thus returned is written as a user tape label when the file is opened on tape. If the amount of information thus returned is greater than the limit of 255 bytes, it is written as the first record of the file instead of as a user label. The first four words of the user label contain

word 1, byte 0 = byte count of the label
words 2 and 3 = date time returned by JULIAN (YYYY0DDD,hhmm0000) packed decimal
word 4 = null

Each file is written to tape with its original name, organization, (RANDOM files are sequential on tape), key max, etc.

Each record of each file is extended by one word of value X'56565656'. Each file is extended by one record of value X'76767676'. This provides Fill with information necessary to determine whether a file was completely written and is therefore restorable.

During an incremental Backup, as each account is processed, the names of all files in the account are added to a buffer. At the completion of the account, the file-names buffer is written as a file on the Backup tape. The name of the file is the same as the last file written but the user label contains a flag indicating it is the special file. This file is used by Fill to determine which files should be allowed to remain in each account. Files not listed in this special file are deleted.

INTERACTION

The file management system is used to perform all I/O. The services used are M:OPEN, M:CLOSE, M:READ, M:WRITE, M:PRECORD, M:PFIL, M:DELREC, M:CVOL.

M:TYPE is used to tell the operator about problems and about beginning and completion of automatic Backup.

M:TIME is used to establish the current time.

M:XXX is used in case of fatal trouble although normal termination is via the M:WAIT CAL.

Dynamic core is acquired and released by Get Common and Free Common CALs for the I/O buffer. The file names buffer is acquired and released by Get Dynamic and Free Dynamic CALs.

Super Close terminates the output to the line printer.

The DCBs for :BACKUP, the user input files (F:EI) and the Backup tape files (F:EO) are created by M:DCB PROCs. F:1 is used for reading the file BACK:SCHED. The F:EO DCB is referenced from FILL.

A 50-word table (TABLE) is used to remember user requests if the specified files are busy when Backup attempts to open them.

A 20-word table (NTRY) is used to hold the sorted times from BACK:SCHED.

SUBROUTINES

PROCs: The procedure MOVE:FLD FROM, TO is defined and used to move a variable parameter list into a specified area. The end-indicator (byte 1) terminates the list being moved.

External Subroutines: The MAILBOX subroutine is called to deliver messages to the user's MAILBOX and to the line printer. Successful Backup messages are sent to the MAILBOX only for user-requested Backup. Otherwise, only failure messages are delivered to the user.

JULIAN is called to convert the time from M:TIME into a julian format.

Internal Subroutines:

ESSENCE – converts the current time from M:TIME and puts it in CURRENT in the format of YDDD in bit positions 1 through 15 in packed decimal and the number of minutes since midnight in hex in the right-half. Essence also calls READ:SCH to read the schedule file.

FND:DATE – extracts the creation date from the FPARAM data and converts it to the format described under Essence. The creation date is stored in CREATION.

READ:SCHED – reads and scans the schedule file BACK:SCHED. The requests are sorted by time and the types are stored in SC:TYPE and the times are stored in SCHED in the same format described under Essence. The number of requests is stored in NSCHED. Any request specifying NULL causes all entries of that type to be deleted from SCHED. Entries in BACK:SCHED which are not valid are deleted from BACK:SCHED by M:DELREC.

ERRORS

Error message sent to the line printer and the user's MAILBOS:

ERR xx DID NOT BACKUP FILE filename

The xx is the error code returned by the file management system. The error may have resulted from reading the user's file or writing the Backup tape. Each message is date-time stamped by mailbox and the account of Backup (:SYS) is included in the user's MAILBOX message. The account included in the line printer message, however, is the user's account.

RESTRICTIONS

Backup/Fill ghost must be executed as a ghost with X'CO' privilege. Execution as a batch or terminal shared processor causes execution of an abort CAL.

Files with names whose first character is less than "$" are not backed up.

DESCRIPTION

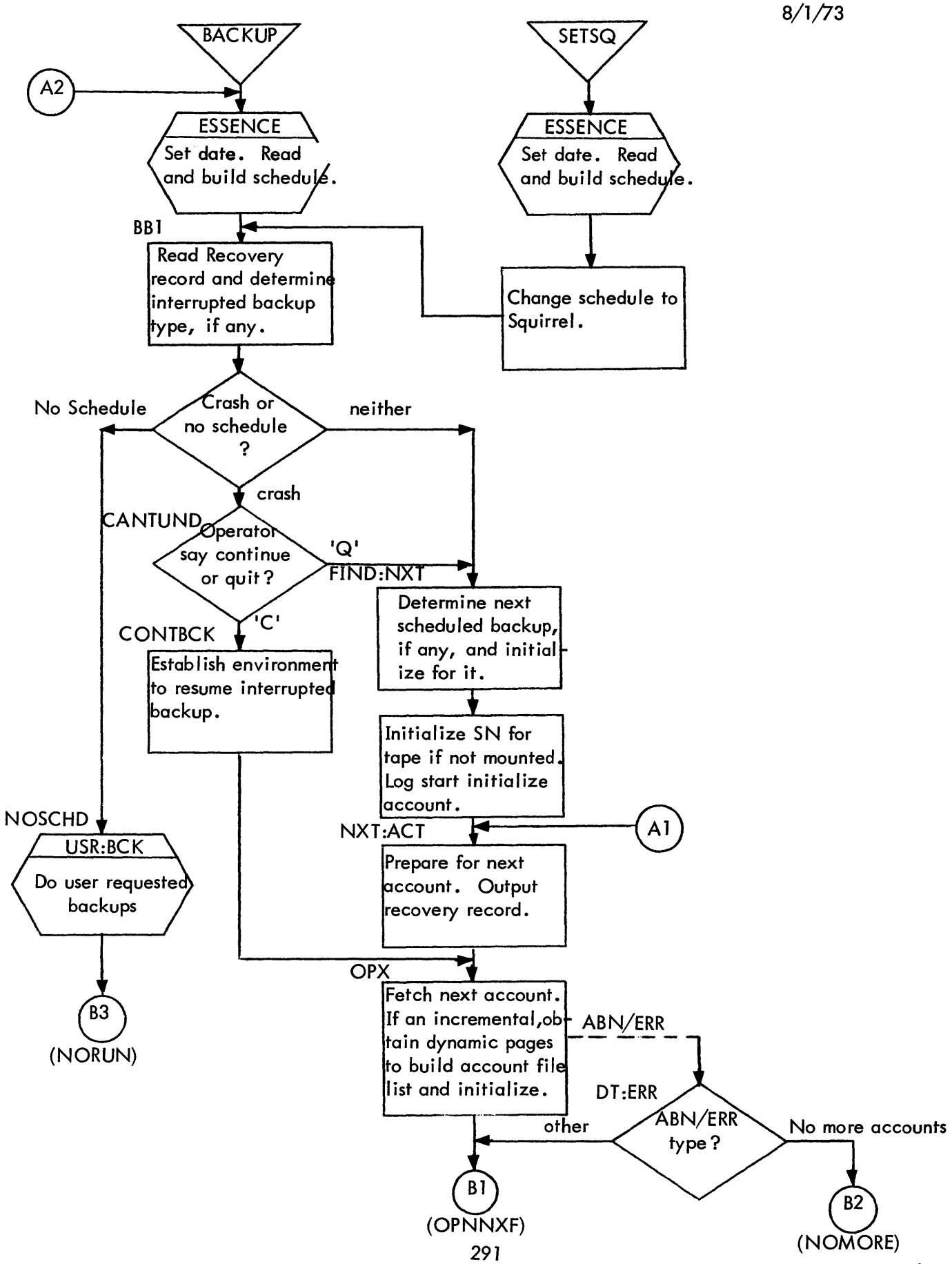
Backup first examines the backup schedule to determine whether or not there is something for automatic Backup to do. If there is, it is done. Next, it processes all user requests for backup and finally, it sets its alarm clock and goes to sleep. Any subsequent CAL to wake up causes it to run the same scenario.

The current time is established and the checkpoint information regarding last completion times is read from :BACKUP. The sorted requests from BACK:SCHED are examined to see if any request is within -1 or +15 minutes of the current time. If none is within range, the requests (if any) in :BACKUP are processed. If one is within range, it is examined to see whether or not it is too close to the completion time of its type. The reason for this is if Backup completed and the system crashed and Backup were awakened following the crash, the request in the schedule should not be honored again. The limits are :Squirrels cannot run less than 15 minutes apart. Incrementals less than 20, and Saveall less than 30.

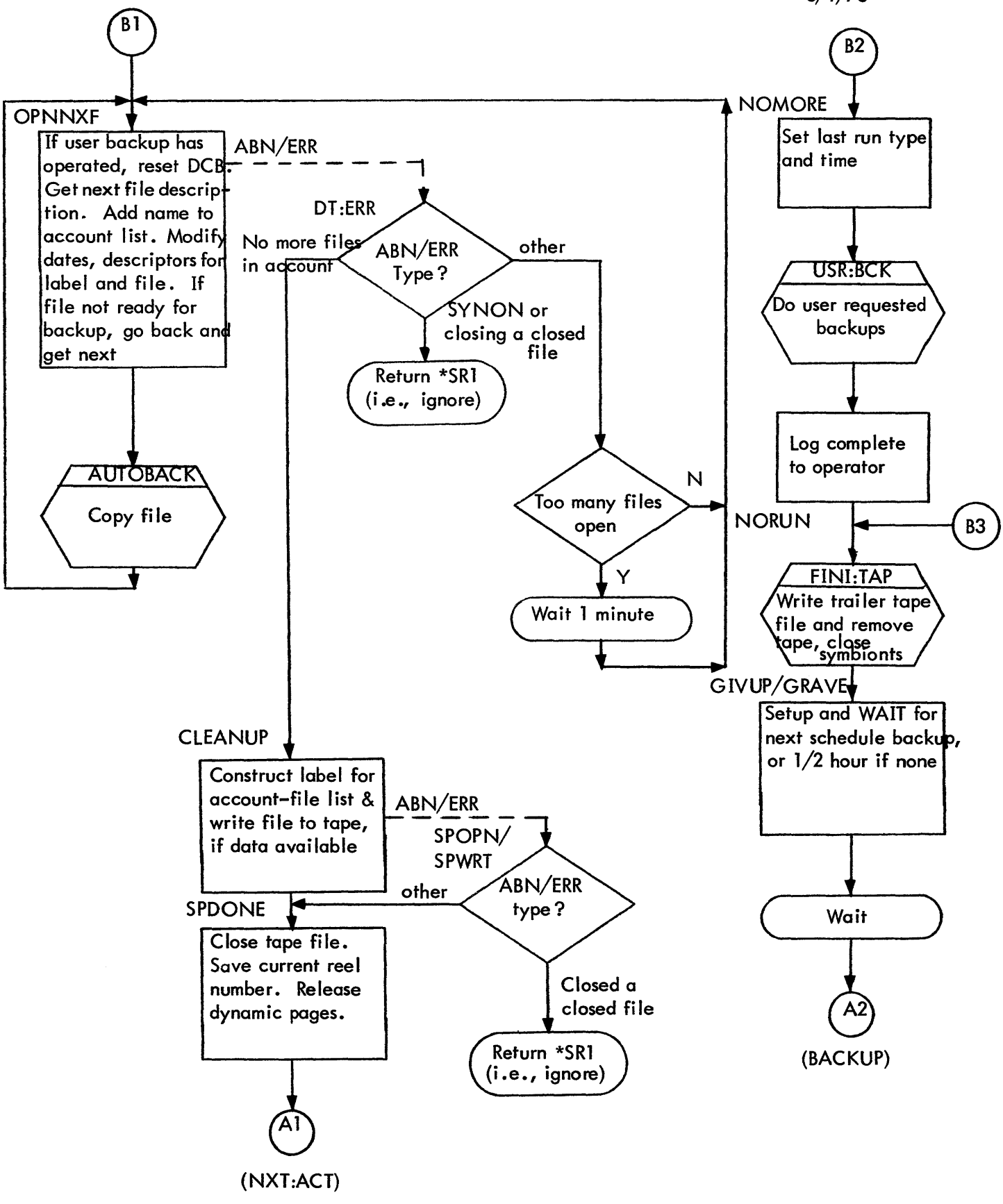
The beginning of each automatic Backup with type, date, and time is noted on the operator's console. The first account is located and the first file is opened. The file name is entered in a file-name buffer in TEXTC format. The creation date is extracted, converted, and tested. If it is time 2400, or if the NOBACKUP descriptor is set, the file is skipped because it was booted from the PO tape. If the corresponding dynamic descriptor is not set, it is skipped. Otherwise, the FPARAM information and the name and account are passed to the processing for copying the file to the Backup tape. After all files for one account have been processed, the file-name buffer is copied to the Backup tape and the file-name buffer is reset. After all accounts have been processed, any outstanding user requests are processed and the completion message is typed on the operator's console. If the run was a Saveall or Incremental, the tape is dismounted. Finally, the Backup schedule is read again and the period of wait is established for the M:WAIT CAL.

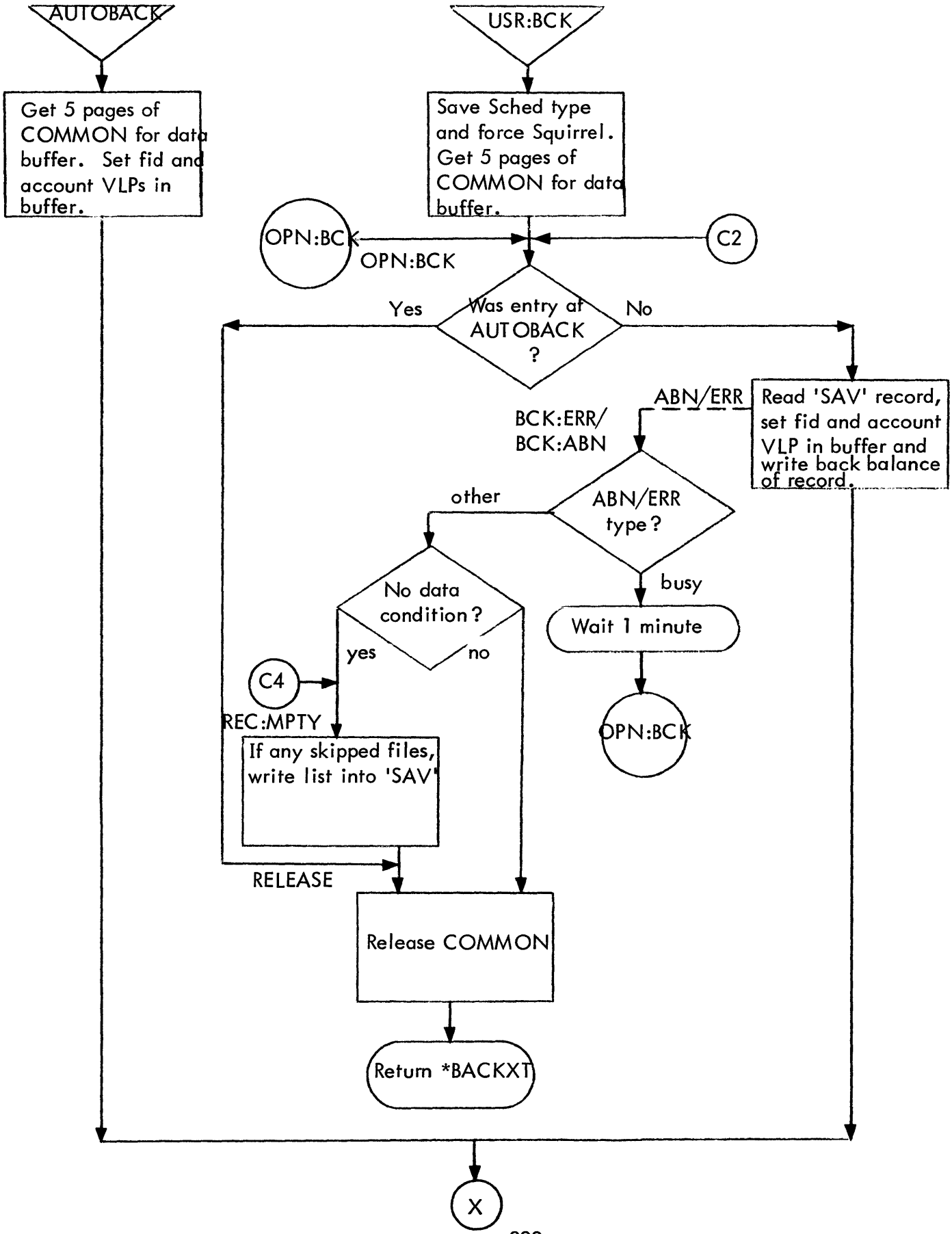
Copying a file to the Backup tape involves moving the FPARAM data to a buffer (BLABL), adding the time, opening the tape, reading the file, adding the magic word, and writing the tape.

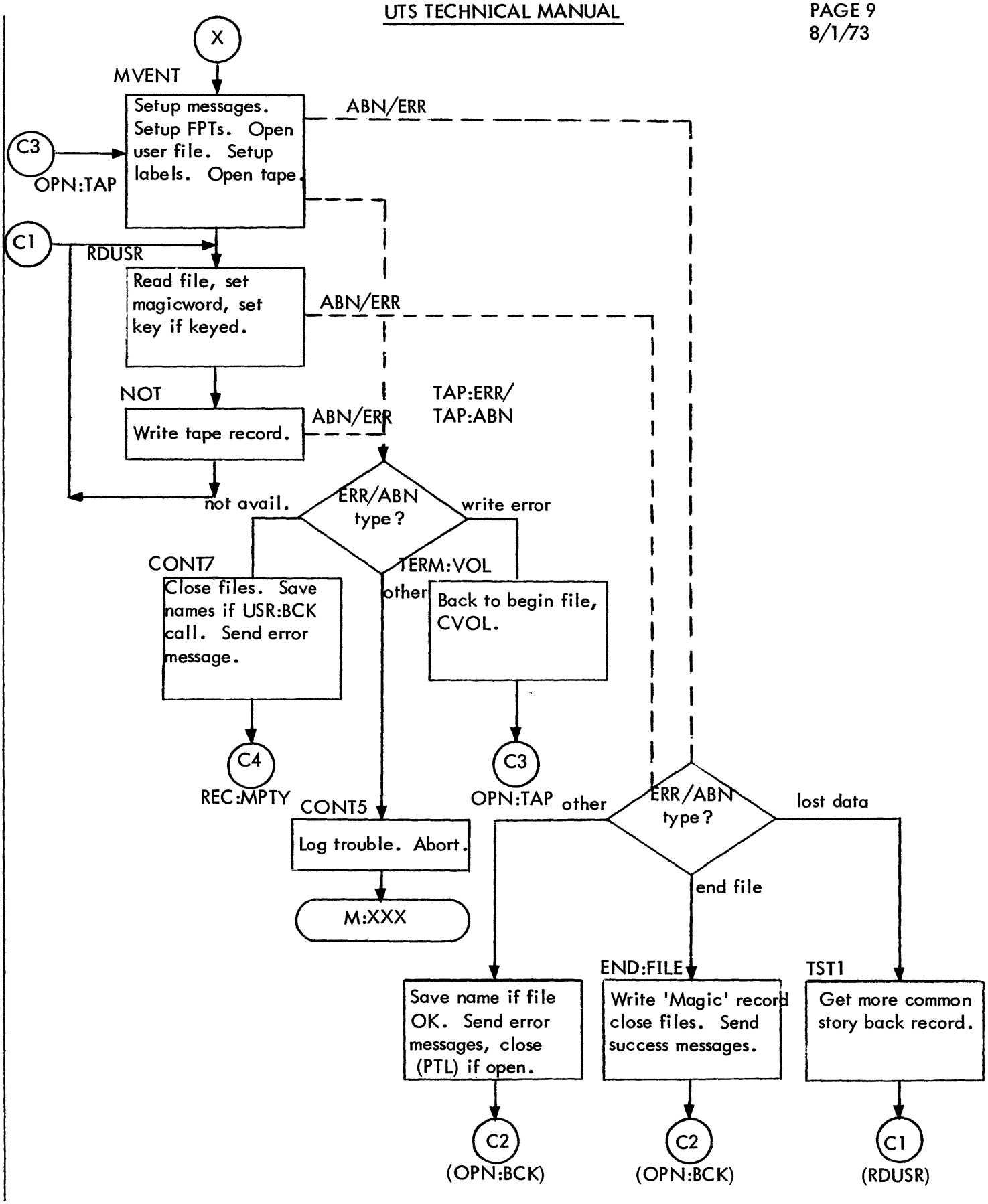
Errors in opening or reading the user's file causes a message to be sent to the user's MAILBOX and to the line printer. Successful copies are noted on the line printer. Normal volume closing of the tape is handled by the file management system, but an unrecoverable write error and some other unexpected errors (56, 57, and 1C) may cause premature volume switching. When the end-of-file is detected in the user's file, the magic record is written to the tape and the tape file is closed.


BACKUP
SETSQ
A2
ESSENCE
Set date. Read and build schedule.
ESSENCE
Set date. Read and build schedule.
BB1
Read Recovery record and determine interrupted backup type, if any.
Change schedule to Squirrel.
No Schedule
Crash or no schedule ?
neither
crash
CANTUND
Operator say continue or quit?
'Q'
FIND:NXT
'C'
CONTBCK
Establish environment to resume interrupted backup.
Determine next scheduled backup, if any, and initialize for it.
Initialize SN for tape if not mounted. Log start initialize account.
NOSCHD
USR:BCK
Do user requested backups
NXT:ACT
A1
Prepare for next account. Output recovery record.
OPX
Fetch next account. If an incremental,obtain dynamic pages to build account file list and initialize.
ABN/ERR
DT:ERR
ABN/ERR type ?
other
No more accounts
B3
(NORUN)
B1
(OPNNXF)
B2
(NOMORE)

UTS TECHNICAL MANUAL




AUTOBACK
USR:BCK
Get 5 pages of COMMON for data buffer. Set fid and account VLPs in buffer.
Save Sched type and force Squirrel. Get 5 pages of COMMON for data buffer.
OPN:BCK
OPN:BCK
C2
Was entry at AUTOBACK ?
Yes
No
ABN/ERR
BCK:ERR/
BCK:ABN
Read 'SAV' record, set fid and account VLP in buffer and write back balance of record.
ABN/ERR type?
other
busy
No data condition?
yes
no
Wait 1 minute
C4
REC:MPTY
OPN:BCK
If any skipped files, write list into 'SAV'
RELEASE
Release COMMON
Return *BACKXT
X


X
MVENT
Setup messages. Setup FPTs. Open user file. Setup labels. Open tape.
ABN/ERR
C3
OPN:TAP
C1
RDUSR
Read file, set magicword, set key if keyed.
ABN/ERR
NOT
Write tape record.
ABN/ERR
TAP:ERR/
TAP:ABN
not avail.
ERR/ABN type?
write error
CONT7
Close files. Save names if USR:BCK call. Send error message.
TERM:VOL
other
Back to begin file, CVOL.
C4
REC:MPTY
C3
OPN:TAP
CONT5
Log trouble. Abort.
M:XXX
other
ERR/ABN type?
lost data
end file
END:FILE
TST1
Save name if file OK. Send error messages, close (PTL) if open.
Write 'Magic' record close files. Send success messages.
Get more common story back record.
C2
(OPN:BCK)
C2
(OPN:BCK)
C1
(RDUSR)

ID

FILL - File Restoring

PURPOSE

FILL restores files to disk from tapes written by BACKUP. The restoration of files may be either non-selective or selective. The non-selective mode restores all files found on a set of tapes following a specified file or account. The selective mode restores the specified files or accounts from the specified tapes.

USAGE

FILL is entered each time the Backup/Fill ghost is initialized. Subsequent wake up of the ghost causes entry to Backup. If the ghost is not initialized (not sleeping nor running) either a GJOB BACKUP Keyin or a TEL command BACKUP causes FILL to be initialized. It is normally initialized by GHOST1 during the system initialization process.

Selective FILL is entered from BACKUP if the SEL:FIL file exists in the :SYS account or if the :BREC file in the :SYS account has a record keyed SEL:FIL.

INPUT

The Backup tapes described in the Data section below are input to FILL. The operator's console is read to receive decisions from the operator.

The SEL:FIL file in the :SYS account is read to receive Selective Fill commands.

OUTPUT

The user's files from the Backup tapes are written into the user's accounts. Files which were deleted between executions of Backup are not restored.

A complete transaction log is printed on the line printer.

Error messages are written in the user's MAILBOX.

Operator messages are output to the operator console.

INTERACTION

The file management system is used to perform all tape and RAD I/O. The services include M:OPEN, M:CLOSE, M:PFIL, M:READ, M:WRITE.

M:TYPE and M:KEYIN are used to communicate with the operator.

Super Close terminates the output to the line printer after each set of input tapes.

M:WAIT is used to wait a minute for a requested operator action.

M:XXX abort CAL is executed if FILL is not entered as a ghost.

The Get Common CAL gets 10 pages of core for I/O buffer.

DATA BASES

The F:TI DCB is REFed to the DCB defined in FILLDCB. It is equated to FL:TAPE. The user file DCB, F:USR, is defined by an M:DCB PROC. The :BACKUP file and SEL:FIL file DCB's are also defined by M:DCB PROC's.

The table used to drive Selective Fill is derived from the (KEYED or CONSECUTIVE) inputs in the SEL:FIL file. The table is constructed in a dynamic memory page and saved on disk in the :BACKUP file, record SEL:FIL. This table actually consists of two stacks of information. Starting with the third word of the table are entries that describe the Selective Fill requests. These entries are four words long and contain the account number (two words), pointer to the file name if specified (one word), and tape serial number to be searched. Stacked backwards from the other end of the page are the file names that have been specified, eight words per entry. The first two words of the page contain pointers to the first empty request and the first file name entry. The table formats are detailed in the Data section below.

Other data are REFed to the Backup data base such as the buffer location, the user tape label buffer, and the magic words.

SUBROUTINES

PROC: The MOVE:FLD PROC is used to move the data from the tape label to the OPEN parameter list.

MAILBOX is called to deliver error messages to the user and a complete log to the line printer. Subroutine NO:RCV formats the error message for MAILBOX.

ERROR AND MESSAGES

The messages sent by mailbox are: successful only to the printer and failures to both printer and MAILBOX in the user's account. Each message is date-time stamped by the mailbox subroutine and the user account is included in the line printer message; whereas, :SYS is included in the MAILBOX message.

Success: FILLED filename
Failure: ERR xx FAILED TO RECOVER filename

The xx is the error code returned by the file management system either because of tape failure or RAD failure. Special cases:

'E' means file out of sequence; 'F' means missing account number (can't happen); 'D' means short record, i.e., missing magic word.

Messages to the operator console:

1. REQUEST FILL, NOFILL, OR INSTANT SQUIRREL (F, N, S)

   If the operator wishes to fill from Backup tapes, he types an 'F' in response. If he wishes no action, he types 'N'. If he wishes an instant Squirrel to save files after a crash, he types 'S'.

2. FILL REEL NUMBER =

   This message is typed if 'F' is specified above or if 'YES' is the reply to 4, below. The operator then types the digit-digit-letter-digit reel number and a line feed to specify the starting reel of the set to be used. The message is repeated if the reply is of incorrect format.

3. SKIP TO FILE

   This message is typed following the reel number message to determine the starting point for the Fill operation. The operator may enter a line feed to specify the fill starts at the beginning of the tape, or either a period and account number or a filename, period, account number to specify that the Fill is to start with the specified file or account. If the file or account is not found, the Fill starts with the first file/account alpha-numerically larger than that specified.

4. ARE THERE MORE SETS OF BACKUP TAPES (YES/NO)

   This message is typed after the completion of each set of tapes. Any number of sets may be processed. The operator types NO following the last set.

5. FILL RESTARTED – REMEMBER OR IGNORE PREVIOUS RUNS (R/I)

   The FILL job has been aborted, then restarted by either a GJOB FILL keyin or a BACKUP TEL command. An "R" causes FILL to resume whatever it was doing when aborted, and "I" causes FILL to ignore previous operations and to reinitialize the BACKUP tape serial numbers.

6. FILE OUT OF SEQUENCE – QUIT OR CONTINUE (Q/C)

   Each file is checked to be in chronological sequence with the previous file. A file out of sequence means that the operator may have done a time change Keyin during the execution of Backup or, more catastrophically, the system crashed during execution of Backup and the file just read was really written on this tape in some bygone time and should not be restored. If the operator types 'Q', the Super Close will flush the line printer buffer so that he can know how much of the current set was restored and he will again be asked question 4.

7. BAD FILE ENCOUNTERED

   Each time an error message is sent to the user's MAILBOX and to the line printer, this message is typed for the operator. It is just a warning that there is trouble. Too many of these messages may mean a bad tape drive or RAD trouble.

8. BAD SEL:FIL INPUT record

   The record that prints out was in the SEL:FIL file but is of incorrect format. The record is ignored.

9. UNFILLED REQUEST fid

   The file "fid" was included in a SEL:FIL command but was not found on the specified tape set.

RESTRICTIONS

FILL must be executed as a ghost, otherwise, the abort CAL is used. The ghost must have X'CO' privilege.

DESCRIPTION

When FILL is initialized, it first determines that it is a ghost, otherwise, it aborts. It then asks the operator whether he wants FILL, no-FILL, or instant Squirrel. If he says no-FILL, control is transferred to BACKUP. If he specifies Squirrel, a dummy schedule is created specifying Squirrel now and Backup processing is initiated. If he specifies FILL, a 10 page buffer is acquired in Common and the starting reel number is requested. Each file is opened until the specified starting point is reached, then information from the user label is moved to the OPEN parameter list for the user output file and the file is copied. If the file already exists on disk with at least as late a modification date as the tape file, the file is skipped. If the user label indicates the special file-name file, each file in the specified account is compared with the list of names in the file-name list. Files which are not in the file-name and have a modification date earlier than the list's are deleted.

If a lost data condition is encountered, FILL gets another 32 pages of COMMON and re-reads the record. If the read still fails, all of available core is obtained and the record reread. If still unsuccessful, the file is skipped. After a successful read following acquisition of extra pages, all pages in excess of record size plus two pages are released. At the conclusion of copying each file, if more than 20 pages of COMMON are held, all but 10 pages are released, thus optimizing the swap size.

Files which are found to be short, i.e., missing the magic word or magic record, are not restored. Any file named MAILBOX is opened in update mode rather than output so that messages are appended rather than having each backed up MAILBOX replace the previous copy. The reason for this is that Fill may create or add to a MAILBOX and, if backed up copies deleted current copies, the Fill messages would be lost. The disadvantage is that old messages keep coming back like a song.

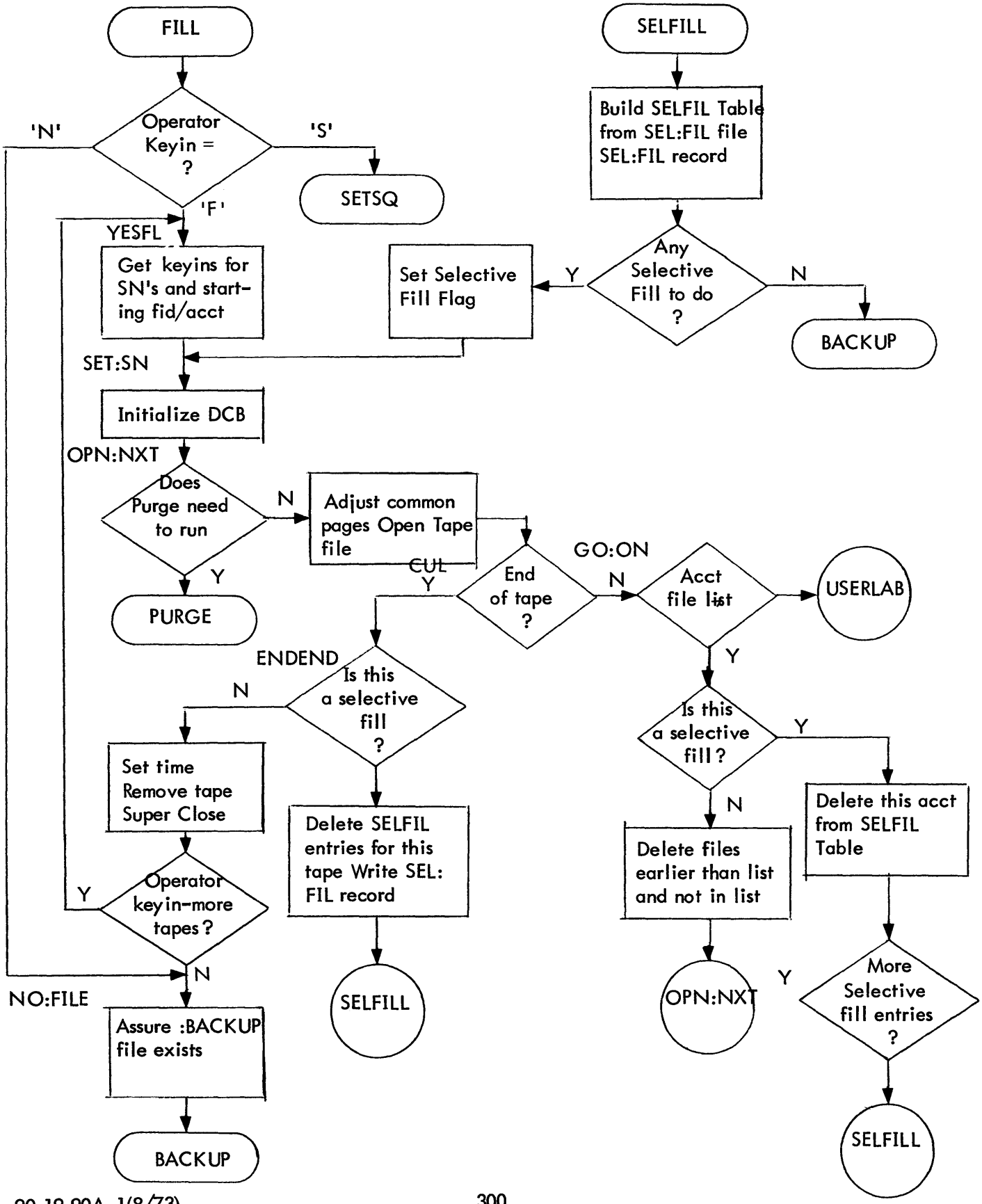
At the completion of the last reel of each set of the Backup tapes, the operator is asked if there are more sets of tapes. If there are, Fill processing continues. If there are not, Fill transfers control to Backup in order to check the Backup schedule and set the alarm clock.

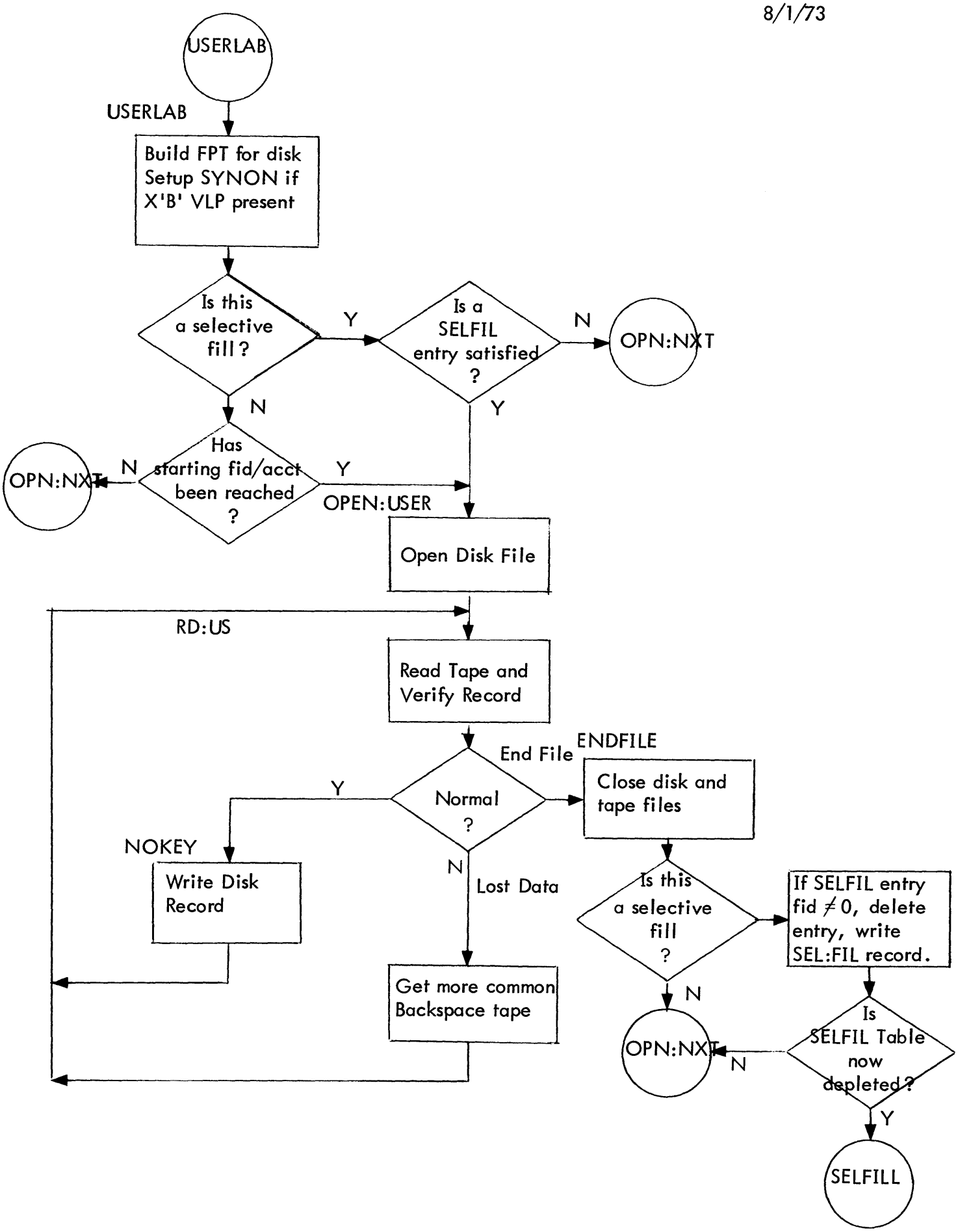
When Selective Fill (SELFILL) is entered it obtains a dynamic page and reads the SEL:FIL record of the :BREC file into the page. Then records are read sequentially from the SEL:FIL file. Each record is converted into a SEL:FIL entry and merges into the Selective Fill Table, the table is written to disk, and the SEL:FIL file record is deleted. This process terminates when either the table is full or the SEL:FIL file is empty. If an incorrect format command is found it is sent to the operator's console. When SEL:FIL file is exhausted it is deleted. The Selective Fill Table is sorted on the first three characters of the tape serial number (SN).

When the Selective Fill Table is completed, the first SN is set into the FILL SN buffer, start at beginning is indicated, the Selective Fill Flag is set, and normal FILL routines are entered. In this mode, when an account-file list record is encountered the Selective Fill Table is searched for a matching account entry. If found the entry is deleted because the end of that account has been reached. When a data file is encountered, its account and SN (and file name, if specified) is checked against the Selective Fill Table. If not found, the file is skipped. If found, the file is then copied as in FILL. After copying, if the Selective Fill Table entry was for a specific file, the entry is deleted.

If the end of a tape set is reached, all unfilled requests are logged as such and are deleted. After deleting a request due to filling or reaching end account, if there are no more entries in the Selective Fill Table for the current tape set, the tape is removed. In either event the Selective Fill Table (and SEL:FIL record) are added to from the SEL:FIL file and the process is repeated. When the SEL:FIL record and the SEL:FIL file are exhausted, SELFILL exits to BACKUP.

If the operator interrupts for Purge, or if BUFGRAN flags Purge to be operated at the conclusion of processing the current file, SELFILL transfers control to PURGE. Control returns to SELFILL when PURGE completes.


FILL
Operator Keyin = ?
'N'
'S'
'F'
SETSQ
YESFL
Get keyins for SN's and starting fid/acct
SET:SN
Initialize DCB
OPN:NXT
Does Purge need to run
N
Y
PURGE
Adjust common pages Open Tape file
CUL
End of tape ?
Y
GO:ON
N
Acct file list
USERLAB
Y
ENDEND
Is this a selective fill ?
N
Y
Set time Remove tape Super Close
Operator keyin-more tapes?
Y
N
NO:FILE
Assure :BACKUP file exists
BACKUP
Delete SELFIL entries for this tape Write SEL: FIL record
SELFILL
Is this a selective fill?
Y
N
Delete files earlier than list and not in list
OPN:NXT
Delete this acct from SELFIL Table
Y
More Selective fill entries ?
SELFILL
SELFILL
Build SELFIL Table from SEL:FIL file SEL:FIL record
Any Selective Fill to do ?
Y
N
Set Selective Fill Flag
BACKUP


USERLAB
USERLAB
Build FPT for disk
Setup SYNON if
X'B' VLP present
Is this
a selective
fill ?
Y
Is a
SELFIL
entry satisfied
?
N
OPN:NXT
N
Y
Has
starting fid/acct
been reached
?
N
OPN:NXT
Y
OPEN:USER
Open Disk File
RD:US
Read Tape and
Verify Record
End File
ENDFILE
Normal
?
Y
Close disk and
tape files
NOKEY
Write Disk
Record
N
Lost Data
Get more common
Backspace tape
Is this
a selective
fill
?
If SELFIL entry
fid ≠ 0, delete
entry, write
SEL:FIL record.
N
OPN:NXT
N
Is
SELFIL Table
now
depleted ?
Y
SELFILL

ID

PURGE

PURPOSE

The PURGE module performs the automatic and operator initiated semi-automatic file purge operations.

USAGE

PURGE can be initiated as a result of the Monitor detecting a granule shortage (automatic) or as a result of operator action (Semi-automatic). In either case the number of available granules is logged. In the automatic mode all expired files are deleted. In the semi-automatic mode, all expired files are deleted and either all files not accessed in an operator specified interval are deleted, or sufficient files are deleted, on the basis of access date, to bring the number of available granules up to the operator specified number. In both semi-automatic procedures, files are deleted only after assuring that they are backed up.

INPUT

Automatic initiation is performed by ALLOCAT. When the number of available granules reaches a threshold value, or 3/4, or 1/2, or 1/4 thereof, or after reaching such a level then increasing by 1/4 the threshold value, a WAKEUP CAL (Initiate Diagnostic Job CAL) is made by the Monitor. If FILL is already operating, the Monitor sets a flag to tell the currently operating FILL function to temporarily switch to the Purge function. Semi-automatic initiation is performed by the operator entering an interrupt Keyin for the BACKUP ghost job. The Keyin results in the job being set to indicate that BACKUP or FILL should switch to PURGE, as above.

OUTPUT

When PURGE is initiated it outputs the message:

AVAILABLE GRANULES = nnnnn

where nnnnn is the number of granules currently available for file use. If there are currently any files that are being backed up so that they can be purged (deleted), the sum of the granules involved is indicated by the following message to the operator:

GRANULES IN PROGRESS = nnnnn

When PURGE determines that the threshold has been passed, the following is output on the operator's console:

DO EXPIRED FILE PURGE (Y/N)

If N is entered, no action is taken and the message will not be repeated during that hour. If Y is entered, the message:

EXPIRED FILE PURGE INITIATED

is output on the operator's console. Then PURGE examines all files in the system, deleting those whose expiration date has passed. If the BACKUPALL SYSGEN parameter is set, files that have not been backed up will be prior to deletion.

When PURGE has been initiated by operator action (INT Keyin), the above messages are followed by:

ENTER PURGE COMMAND

The operator must then enter one of the following commands:

NONE	indicates no further action is to be taken.
MIN n	indicates the threshold value for automatic purge is to be changed to n.
PURGE UNTIL n	indicates files already backed up are to be purged, by order of time since last access, until there are at least n granules available or in progress.
PURGE ALL UNTIL n	same as preceding except files not already backed up are to be backed up and included.
PURGE OLDER t	indicates all files already backed up and not accessed in the last t days are to be purged.
PURGE ALL OLDER t	same as preceding except files not already backed up are to be backed up and purged if their access date qualifies them.

In the above commands: n = one to four digits indicating number of granules.
t = one to three digits indicating number of days

or

: and two digits indicating number of hours.

When files are purged, a description of the file is written to the M:LO DCB in the following format:

account, filename, modification date, backup date

INTERACTION

PURGE interacts with the Monitor through the following CAL's for the specified purposes

Operator – PURGE communication

M:INT
M:KEYIN
M:TYPE

Purge and delete list generation

Test File CAL

Delete Files

M:OPEN
M:CLOSE (REL)

Output Purge List

M:WRITE
M:CLOSE (SAVE)

PURGE Initiation

Initiate Diagnostic Job CAL
M:TRTN

SUBROUTINES

PURGESETCTR

Resets GRANMIN to the smallest positive value of:

GRANRAD+GRANPACK - { THRESHOLD, 3/4 * THRESHOLD, 1/2 * THRESHOLD, 1/4 * THRESHOLD, 0 }

and sets GRANRESET to a large negative value if the number reflected in GRANMIN of granules is greater than THRESHOLD, otherwise to minus 1/4 * THRESHOLD.

DOPURLIST

Writes the description of the file being purged to the M:LO DCB, and closes the file with SAVE if the file has not previously been closed. The files descriptors are set for No Backup, No Access Date Update, and No Purge.

DOPURDEL

Opens the file to be deleted and closes it with REL.

PURGEINT

Is the entry point established for operator initiated interrupts. It sets the interrupt flag (INT), wakes up the BACKUP Ghost, and returns to M:TRTN.

PURGELOG

Writes to the operator the total number of RAD and pack granules currently available and, if not zero, writes the number of granules in process of being backed up preparatory to purge. If LOGFLAG is set, no messages are output. LOGFLAG is set upon exit.

DOPURGE

Performs the file deletion or purge function based on the attributes of files, the PURGE items: PTYPE, ALLFLAG, PDATE, PTIME, INPROGRAN, and BACKUPALL and the Monitor items DATE, TIME, GRANRAD, and GRANPACK. If PTYPE is 'A' (automatic mode), only expired files are deleted. If PTYPE is 'O' the expired files are deleted and all eligible backed up files having an access date older than PDATE are released. If PTYPE is 'U' the expired files are deleted and a list of eligible, backed up files is generated and sorted according to access date. Files listed in this table are then released until the sum of GRANRAD+GRANPACK+INPROGRAN equals or exceeds PNUM. The descriptions of files released other than due to expiration is listed via the M:LO DCB. If ALLFLAG is set, files that are not backed up are included in the above processing, but are set into a list for BACKUP to backup and release and their size is added to INPROGRAN. If a file has not been backed up because its No Backup Flag is set, and the ALLFLAG is set, the file is treated the same as though it was backed up. If BACKUPALL is set, backup is assured for expired files except those flagged as 'No Backup'. BACKUPALL is assembled as a zero and must be changed to one to activate expiration backup. If a file has the 'No Purge' flag set, PURGE will delete it only for reason of expiration.

The only error condition recognized as such by PURGE is an unrecognizable input by the operator. Such input is treated as though the operator input 'NONE'.

RESTRICTIONS

PURGE operates under the same set of restrictions as BACKUP - FILL and must be loaded with MONSTK.

DESCRIPTION

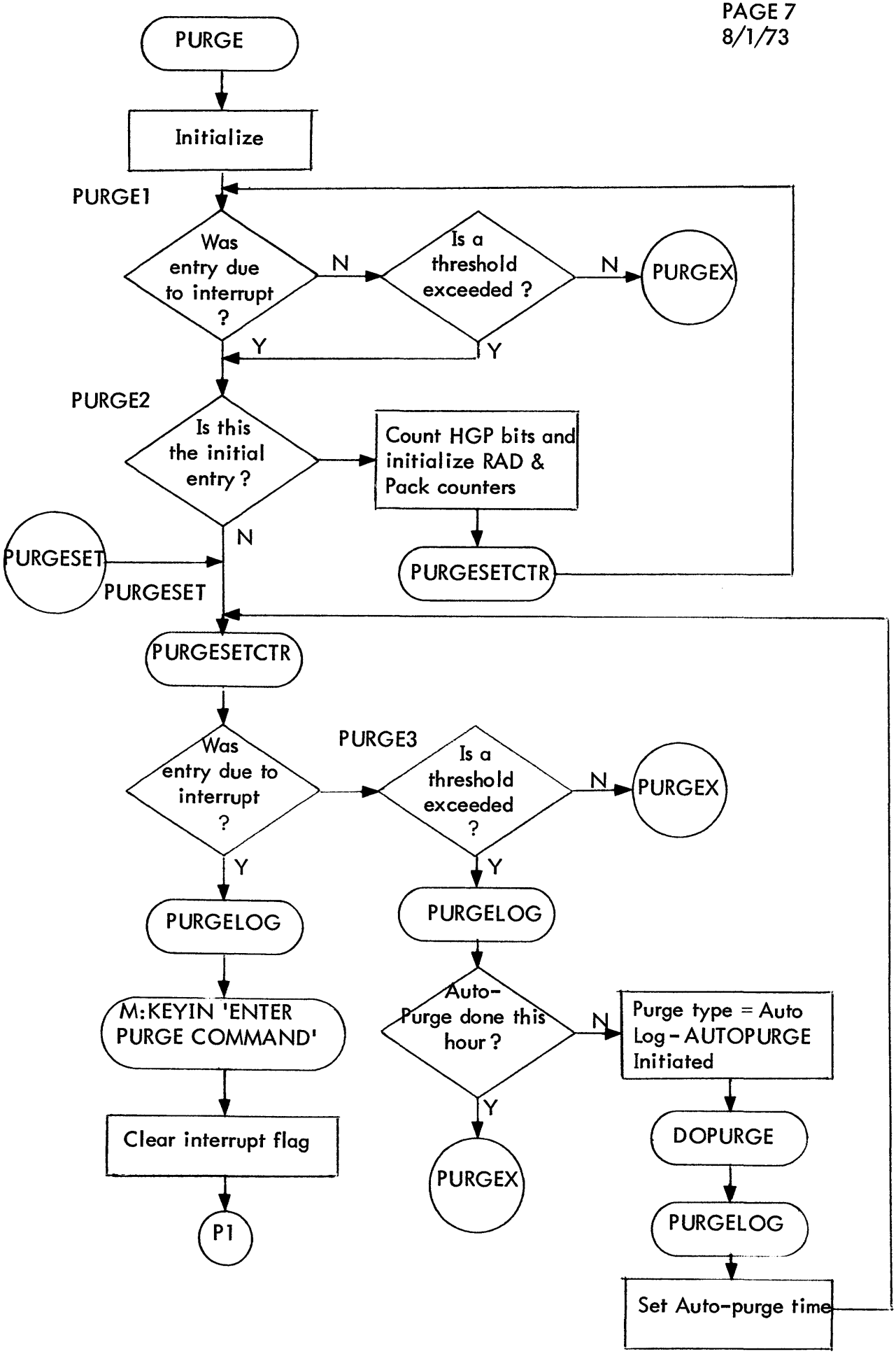
The PURGE routine checks the various granule counters and resets them where necessary. If the entry is due to operator action the number of available granules is logged and a purge command is requested. The command results in no action ('NONE'), a new threshold value ('MIN') or a purge operation ('PURGE'). If the entry is due to automatic initiation by ALLOCAT, and if a minimum granule threshold has been crossed, the operator is alerted via a number of available granules message. If an expired file purge message has not been output in the hour, a message is sent to the operator. If the operator so indicates, the expired file purge is then initiated.

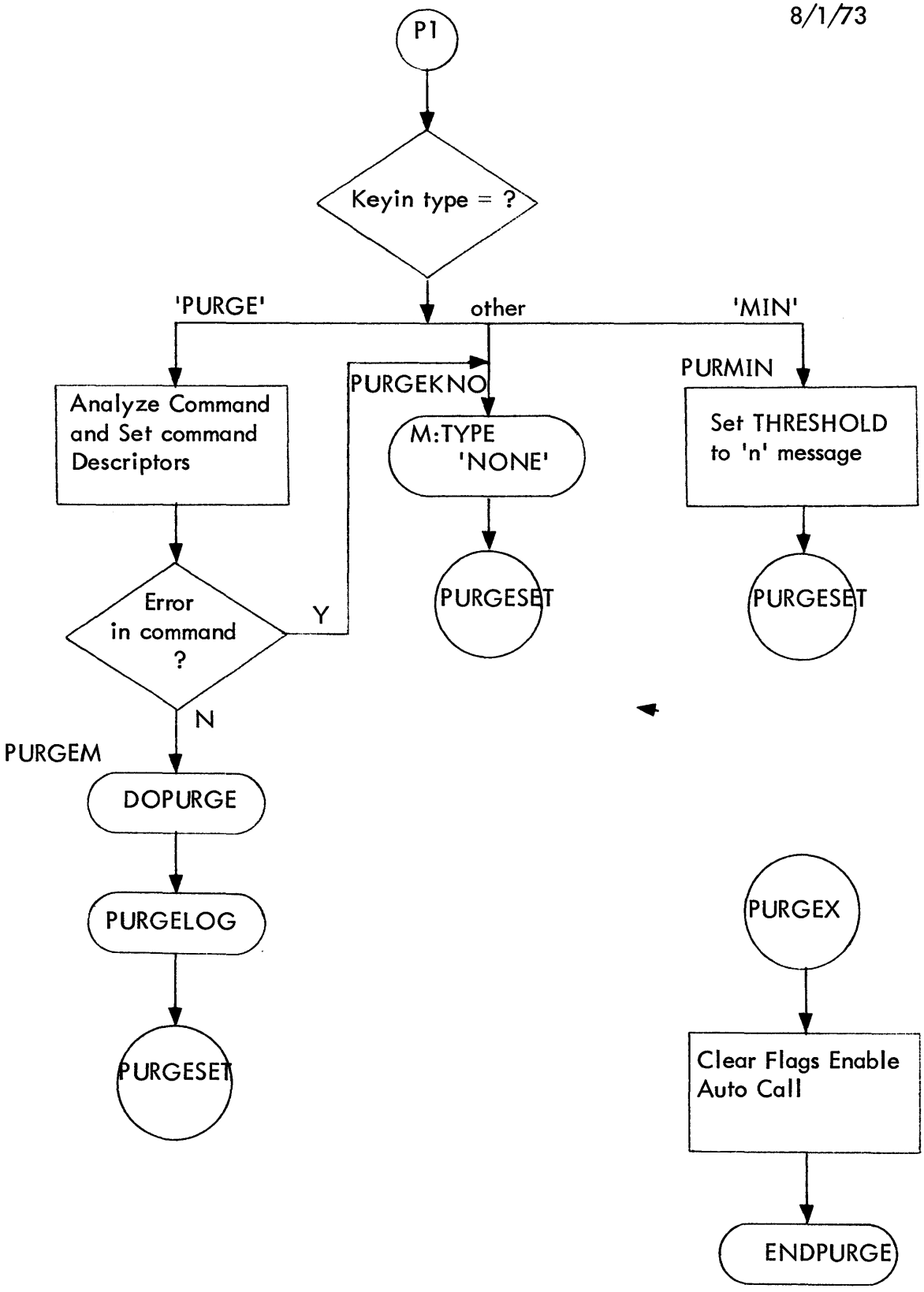
The PURGE routine is entered at location PURGE where initialization is performed. PURGE then exits unless the INT flag is on indicating an operator request, or either GRANMIN is less than zero, indicating a minimum granule threshold has passed, or GRANRESET is positive indicating granule availability has increased sufficiently (1/4*THRESHOLD) to reset GRANMIN. The PURGE triggers are reset and the automatic or semi-automatic procedure is followed depending on whether INT is 0 or 1, respectively.

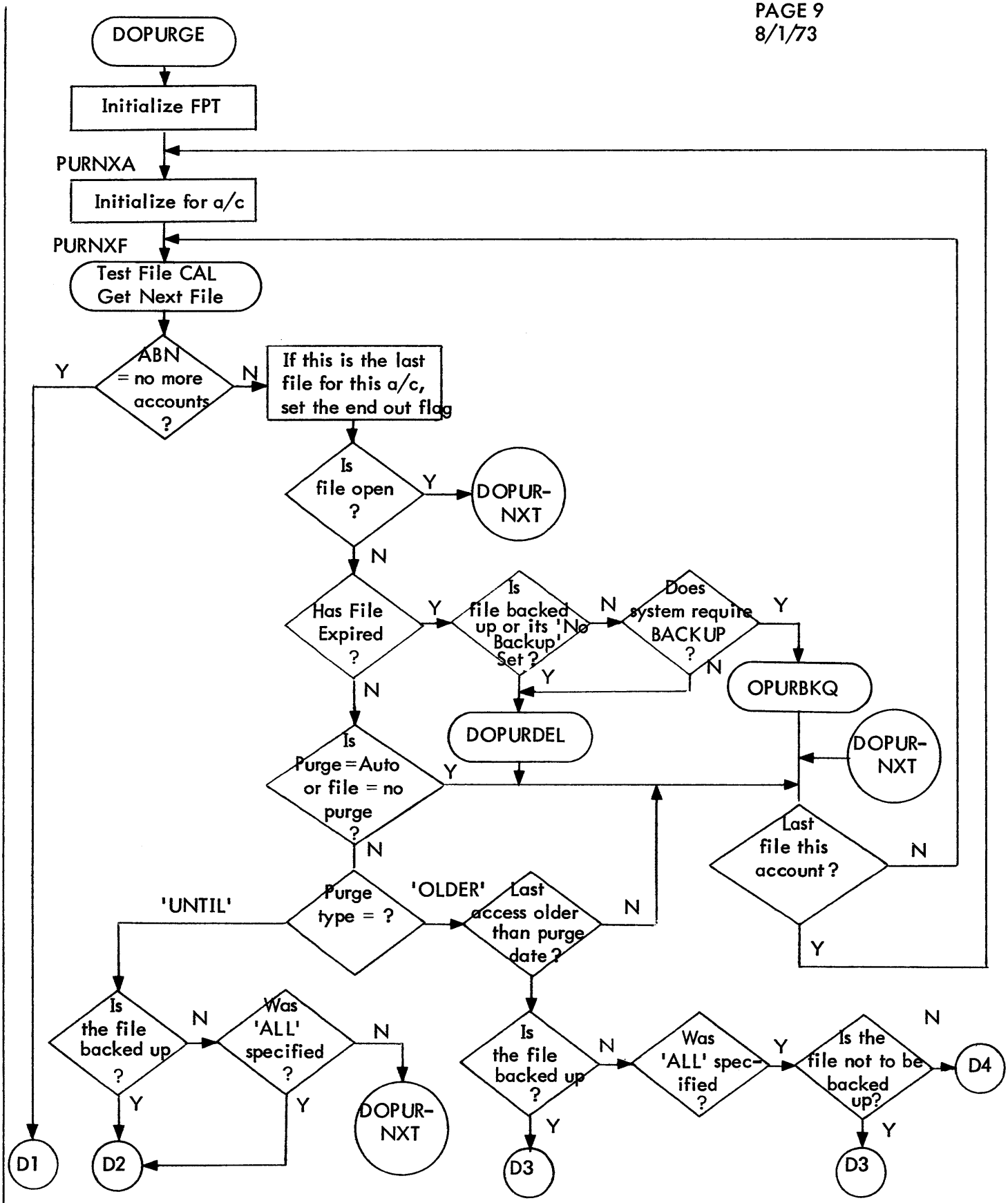
In the automatic mode, if the number of available granules plus the granules of those files in process of being backed up preliminary to purge (GRANRAD+GRANPACK+INPROGRAN) is greater than the threshold value (THRESHOLD), PURGE exits. Otherwise, messages are sent to the operator indicating the number of available granules and, if any, the number of granules "in progress" (INPROGRAN). Then PURGE exits if a purge has been completed in the hour (i.e., hh and dd of TIME and DATE are unchanged since the last purge. Otherwise, the question 'DO EXPIRED FILE PURGE (Y/N)' is asked via the operator's console. If the response is N, PURGE exits. If the response is Y, the purge type is set to automatic, an 'AUTOMATIC PURGE INITIATED' message is sent to the operator, and DOPURGE is called. DOPURGE deletes all files whose expiration date has passed. Upon completion of DOPURGE operation, the granules available messages are repeated, purge triggers are reset, and PURGE exits.

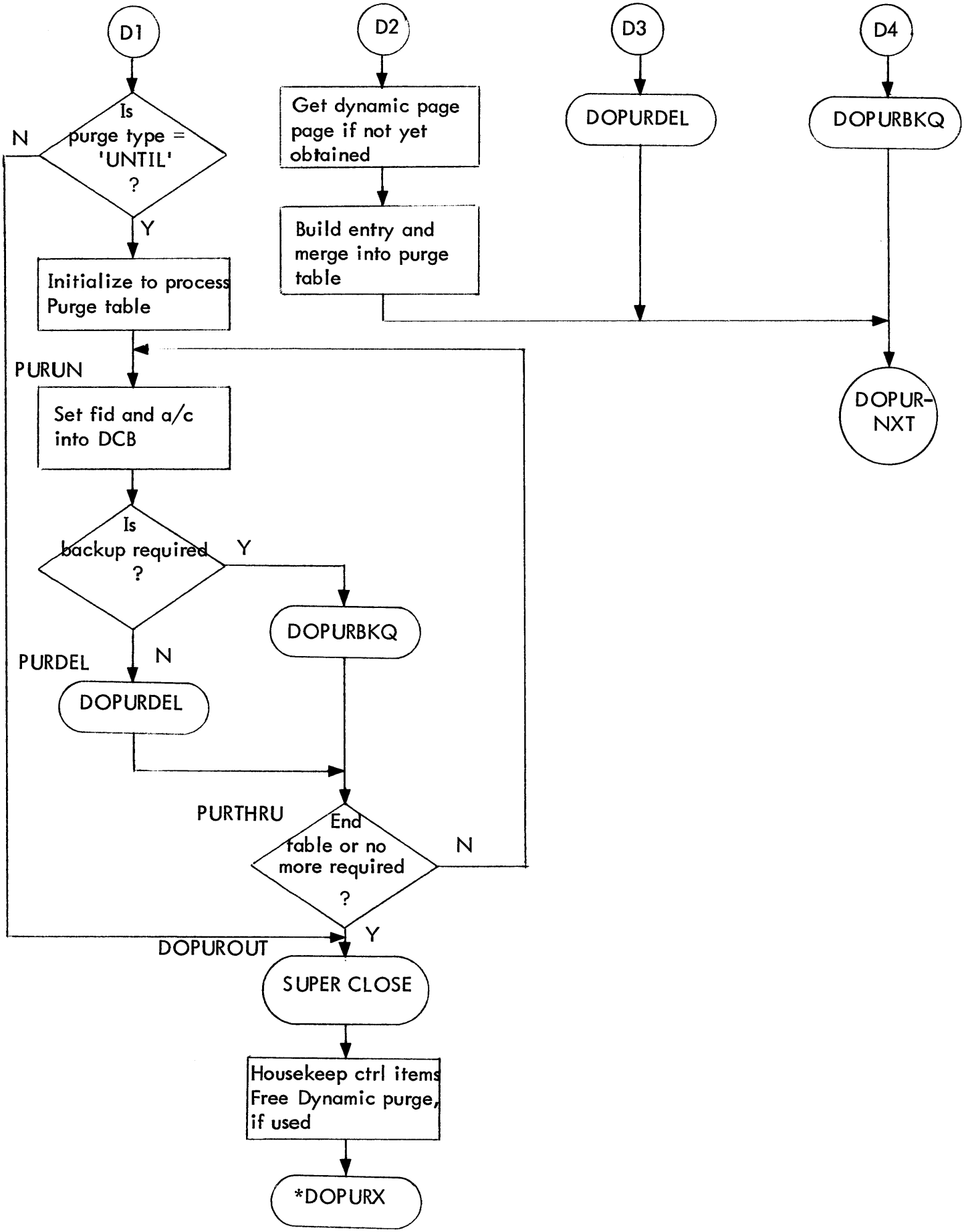
In the semi-automatic mode, granule available messages are sent to the operator, as in the automatic mode, then a keyin is requested with the message 'ENTER PURGE COMMAND'. Upon receipt of the operator's input the interrupt flag (INT) is cleared and the command is examined. If the command is unintelligible or is 'NONE', PURGE types 'NONE' and continues as for automatic mode. If the command is 'MIN', the threshold value is reset to the

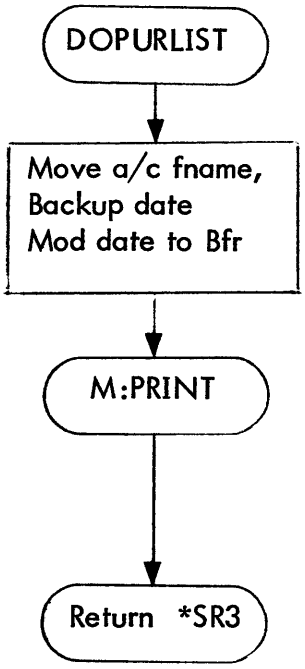
input decimal number of granules and PURGE proceeds as for automatic mode. If the command is 'PURGE', the balance of the command is analyzed, ALLFLAG is set if 'AL' is found, PTYPE is set to 'U' or 'O', depending on whether 'UNTIL' or 'OVER' is found, and PNUM is set from the decimal number in the last input field if 'UNTIL', or PDATE is set with current date minus the indicated days and hours if 'OVER'. DOPURGE is then called to perform the purge, upon return the available and in progress granule counts are logged, and PURGE proceeds as for the automatic mode.

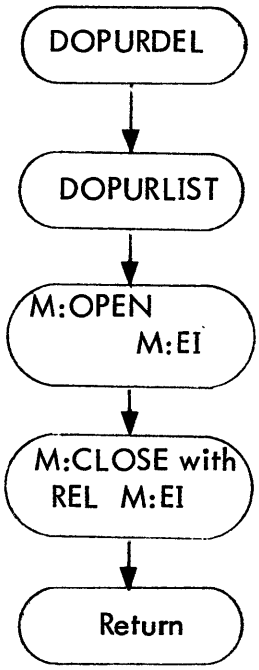

PURGE
Initialize
PURGE1
Was entry due to interrupt ?
N
Is a threshold exceeded ?
N
PURGEX
Y
Y
PURGE2
Is this the initial entry ?
Count HGP bits and initialize RAD & Pack counters
N
PURGESET
PURGESETCTR
PURGESET
PURGESETCTR
PURGE3
Was entry due to interrupt ?
Is a threshold exceeded ?
N
PURGEX
Y
Y
PURGELOG
PURGELOG
M:KEYIN 'ENTER PURGE COMMAND'
Auto-Purge done this hour ?
N
Purge type = Auto Log - AUTOPURGE Initiated
Clear interrupt flag
Y
PURGEX
DOPURGE
P1
PURGELOG
Set Auto-purge time

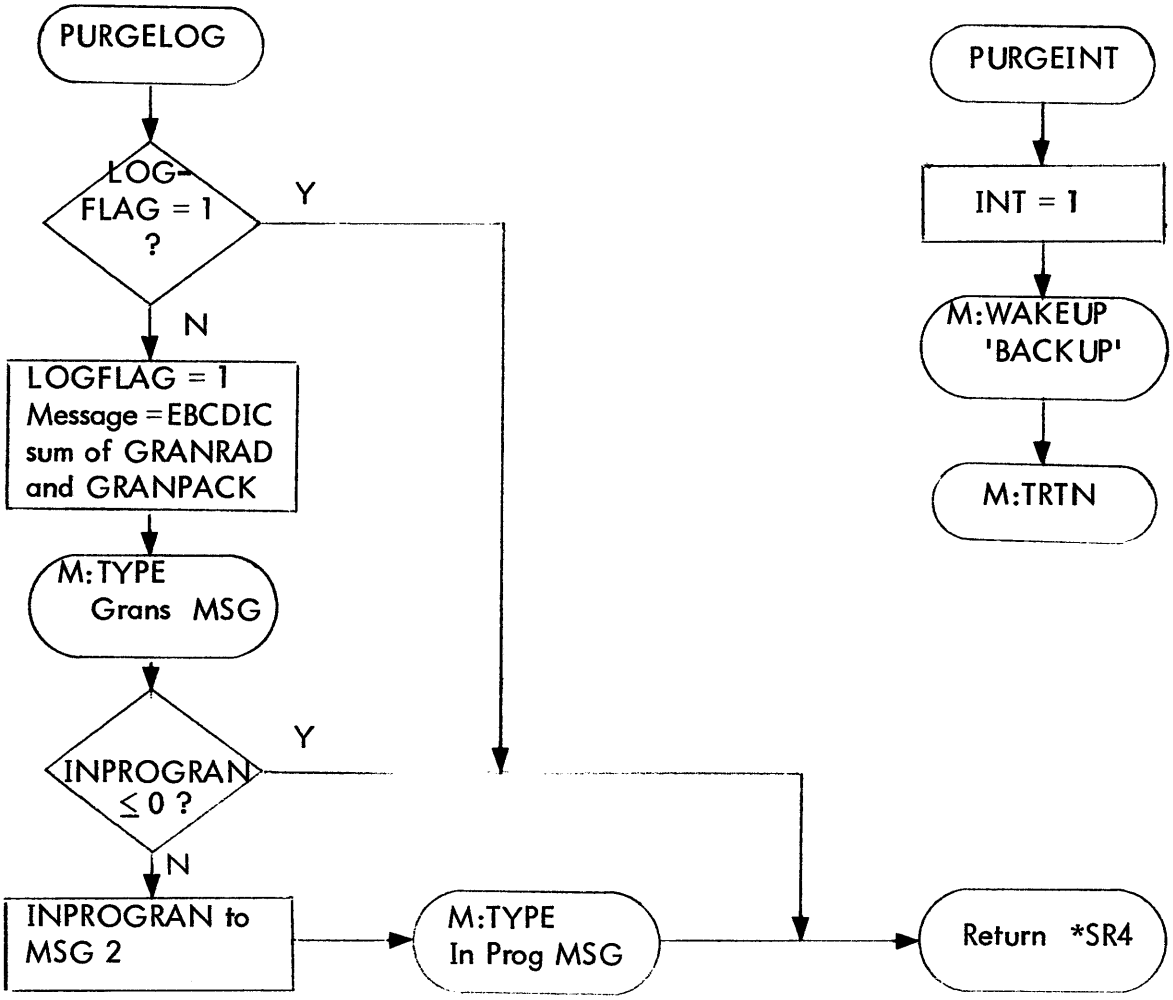

P1
Keyin type = ?
'PURGE'
other
'MIN'
PURGEKNO
PURMIN
Analyze Command and Set command Descriptors
M:TYPE 'NONE'
Set THRESHOLD to 'n' message
Error in command ?
Y
N
PURGESET
PURGESET
PURGEM
DOPURGE
PURGELOG
PURGESET
PURGEX
Clear Flags Enable Auto Call
ENDPURGE


DOPURGE
Initialize FPT
PURNXA
Initialize for a/c
PURNXF
Test File CAL
Get Next File
ABN = no more accounts ?
Y
N
If this is the last file for this a/c, set the end out flag
Is file open ?
Y
DOPUR-NXT
N
Has File Expired ?
Y
Is file backed up or its 'No Backup' Set ?
N
Does system require BACKUP ?
Y
N
Y
OPURBKQ
N
DOPURDEL
DOPUR-NXT
Is Purge = Auto or file = no purge ?
Y
N
Last file this account ?
N
Y
Purge type = ?
'UNTIL'
'OLDER'
Last access older than purge date ?
N
Is the file backed up ?
N
Was 'ALL' specified ?
N
Y
Y
DOPUR-NXT
Is the file backed up ?
N
Was 'ALL' specified ?
Y
Is the file not to be backed up?
N
D4
Y
Y
D1
D2
D3
D3

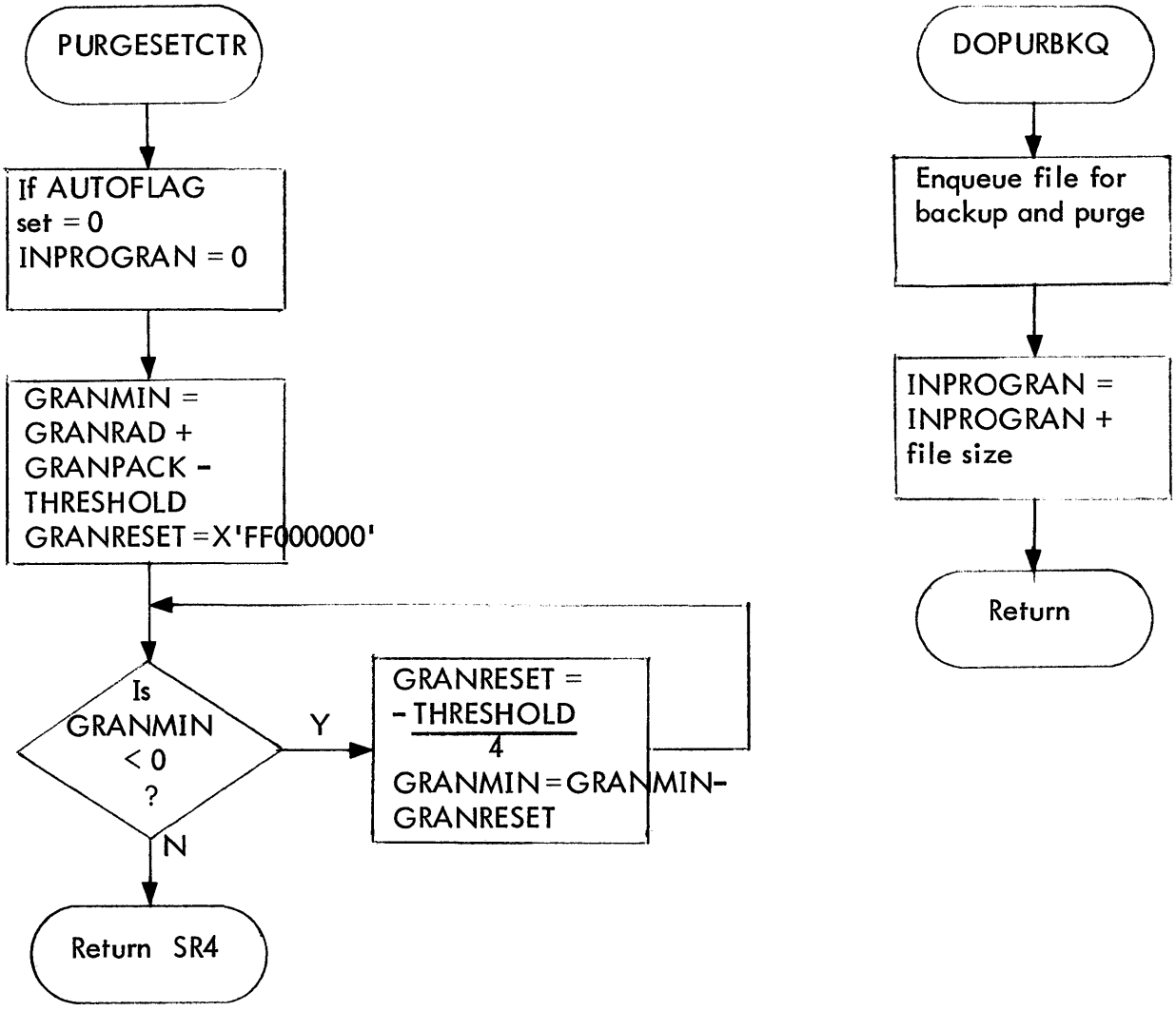

D1
D2
D3
D4
Is purge type = 'UNTIL' ?
N
Y
Get dynamic page page if not yet obtained
DOPURDEL
DOPURBKQ
Build entry and merge into purge table
Initialize to process Purge table
DOPUR-NXT
PURUN
Set fid and a/c into DCB
Is backup required ?
Y
DOPURBKQ
PURDEL
N
DOPURDEL
PURTHRU
End table or no more required ?
N
Y
DOPUROUT
SUPER CLOSE
Housekeep ctrl items Free Dynamic purge, if used
*DOPURX


DOPURLIST
Move a/c fname,
Backup date
Mod date to Bfr
M:PRINT
Return *SR3


DOPURDEL
DOPURLIST
M:OPEN
M:EI
M:CLOSE with
REL M:EI
Return


PURGELOG
LOG-
FLAG = 1
?
Y
N
LOGFLAG = 1
Message = EBCDIC
sum of GRANRAD
and GRANPACK
M:TYPE
Grans MSG
INPROGRAN
≤ 0 ?
Y
N
INPROGRAN to
MSG 2
M:TYPE
In Prog MSG
Return *SR4

PURGEINT
INT = 1
M:WAKEUP
'BACKUP'
M:TRTN


PURGESETCTR
If AUTOFLAG set = 0
INPROGRAN = 0
GRANMIN = GRANRAD + GRANPACK - THRESHOLD
GRANRESET = X'FF000000'
Is GRANMIN < 0 ?
Y
N
GRANRESET = - THRESHOLD / 4
GRANMIN = GRANMIN - GRANRESET
Return SR4
DOPURBKQ
Enqueue file for backup and purge
INPROGRAN = INPROGRAN + file size
Return

UTS TECHNICAL MANUAL

ID

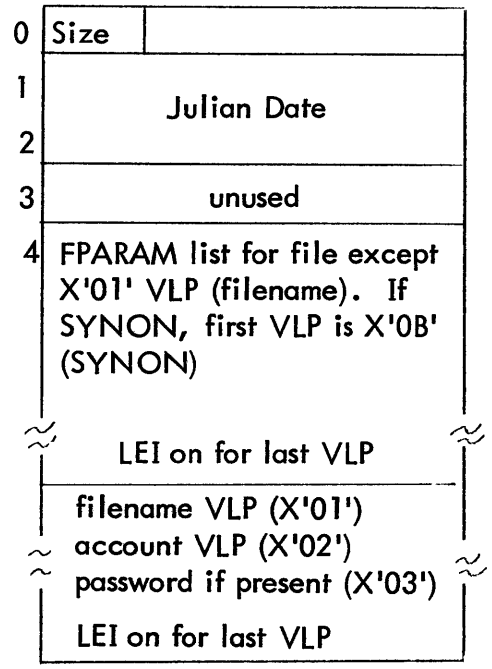
FILL FORMATS

DESCRIPTION

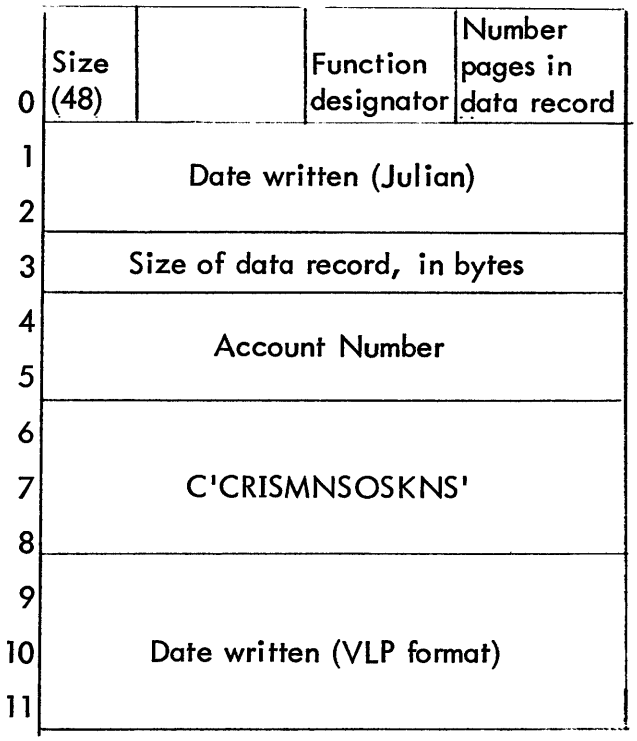
FILL Tape Formats

User Labels

Data Files

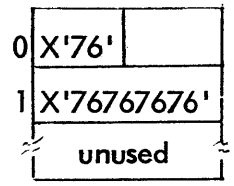


Account Filename List



End Tape Set

(BACKEND)



where

size = size of label in bytes. If zero, the record was of excessive length for a user label so was written as the first data record.

Julian date =									
0	Y	Y	Y	Y	O	D	D	D
1	H	H	M	M	O	O	O	O

YYYY = Year in packed decimal.

DDD = Day of year in packed decimal.

HH = Hour in packed decimal.

MM = Minute in packed decimal.

VLP date =				
0	M	M	D	D
1	H	H	Y	Y
2	H	H	m	m

two digit EBCDIC fields represent

MM = month (01-12)

DD = day (01-31)

HH = hour (00-24)

YY = year (00-99)

mm = minute (00-60)

function designator = 0 normal account file list.

1 truncated list-bypass checks for this account.

Data Formats

Data files are exact copies of the backed up disc files, including keys for keyed files, except:

1. Each record has appended a four byte trailer of X'56565656'.
2. Each file has appended a four byte trailer record of X'76767676'.
3. Random files are written to tape as a one granule per record (plus trailer, above) sequential file.
4. All files are written under account :SYS, no password, read and write accounts = NONE.

Account filename lists are written as a single data record, densly packed (i.e., byte aligned) as strings of TEXTC format filenames.

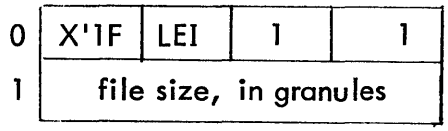
FILL Disc File Formats

Three special disc files are used by FILL. All are in account :SYS. 'SEL:FIL' is the selective Fill command input file, ':BACKUP' is the selective Backup command input file, and ':BREC' is the recovery data file.

'SEL:FIL' may be either keyed or consecutive and contains EBCDIC lines of selective Fill commands. The content of the commands are described in the FILL module writeup, section KA.02, and are of the general format:

FILL = (filename . account), (REEL = serial number)

':BACKUP' is a keyed file containing no more than one record whose key is 'BACKUP'. The record contains VLP lists for files that are to be backed up as a result of the TEL command BACKUP, or a PURGE operation. Each entry contains a filename VLP (X'01'), and an account VLP (X'02'), and may also contain a password VLP (X'03') and a purge VLP (X'1F'). The last VLP of each entry has the LEI set. The name, account, and password VLPs are of the standard form. The purge VLP is of the following form:



The purge VLP is defined only internally to FILL. ':BACKUP' should not exceed 512 words.

':BREC' is a keyed file containing up to two records. The keys are 'SAV' and 'SEL:FIL'. 'SAV' contains nine words of information necessary to resume a backup following a system recovery. The record format and the BACKUP module labels, are as follows:

LASTSAV	The Julian format time of the last completed SAVEALL.
LASTINC	The Julian format time of the last completed Incremental.
LASTSQ	The Julian format time of the last completed Squirrel.
LASTRUN	If no backup is in progress, the Julian format time of the last completed backup or fill. If a backup is in progress, the number 1, 2, or 3 to designate the type of backup as SAVEALL, Incremental, or Squirrel, respectively.
NOTAPFG	Indicates whether or not an output tape for Backup has been requested (0 = not, nonzero = mounted).

LAST ACCT — (Two words) Current, or last, account being backed up, i.e., the starting point to restart a backup. (Meaningful only if a backup is in progress.)

LASTREEL — The SN of the current or last tape used for backup.

LASTFIL — The Julian format time of the last fill, i.e., time of the most recent occurrence of a file having been restored.

The 'SEL:FIL' record, if it exists, contains a copy of the selective fill command table, derived from the 'SEL:FIL' file. The record is 512 words in length (2048 bytes). The command table has two parts, four-word per entry commands packed from the low address end (actually word 2), and eight-word per entry file names, for commands specifying a filename, built back from the high address end. The first word of the record contains the byte index of the first empty command list entry (i.e., if no entries contain an 8, which is the byte index of word 2). The second word contains the byte index of the first filename entry (i.e., if no entries, contains 2048, the byte index of the first byte beyond the table). The command list entry format and, FILL module labels, are as follows:

TACT0, TACT1 — The two-word account number in the selective file command, left-justified and blank filled.

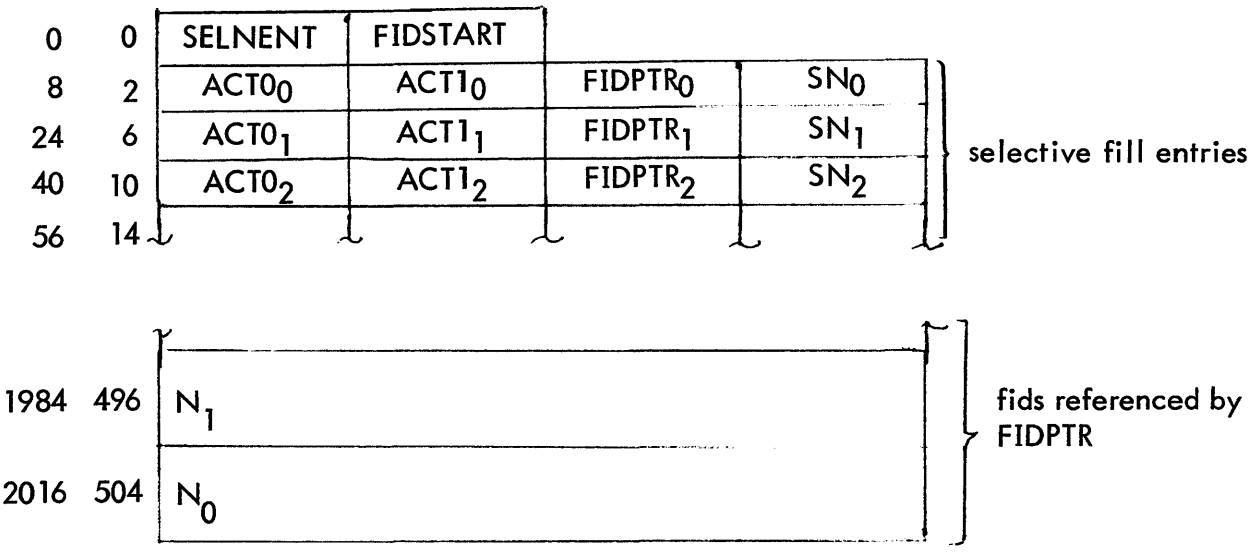
TFIDPTR — Zero if no filename specified; otherwise the byte index to the filename (in the filename portion of the record).

TSN — Before the required tape has been mounted – the SN specified. This value is used as the key to sort the command entries. After the first tape of the set has been mounted – the count of the number of files that have been restored in filling this request.

The file name entries occupy eight words each and are in TEXTC format.

UTS TECHNICAL MANUAL

SEL:FILL Table/:BREC Record



SELNENT — Byte displacement to first unused Fill entry.

FIDSTART — Byte displacement to first used fid entry.

ACT0, ACT1 — Account number associated with request.

FIDPTR — Byte displacement to fid associated with request (0 = no fid)

SN — Starting INSN for search.

N — Number of data bytes in fid (0 = no entry).

Fill entries are initially packed, ordered by SN, SN = 0 if in current tape set, ACT0 = 0 shows empty.

Fid entries are not necessarily packed, N = 0 shows empty.

SEL:FILL occupies first dynamic page in core.

Mirror table for manipulating single entry

0	TACT0	TACT1	TFIDPTM	TSN
4	TFID			
8				

FILL = ([fid].account), (REEL = ddld)

Xerox Corporation
701 South Aviation Boulevard
El Segundo, California 90245

XEROX

Reader Comment Form

We would appreciate your comments and suggestions for improving this publication.

Publication No.	Rev. Letter	Title	Current Date

How did you use this publication?

☐ Learning ☐ Installing ☐ Sales
☐ Reference ☐ Maintaining ☐ Operating

Is the material presented effectively?

☐ Fully Covered ☐ Well Illustrated ☐ Well Organized ☐ Clear

What is your overall rating of this publication?

☐ Very Good ☐ Fair ☐ Very Poor
☐ Good ☐ Poor

What is your occupation?

Your other comments may be entered here. Please be specific and give page, column, and line number references where applicable. To report errors, Please use the Xerox Software Improvement or Difficulty Report (1188) instead of this form.

Your Name & Return Address

2190(12/72)

Thank You For Your Interest. (fold & fasten as shown on back, no postage needed if mailed in U.S.A.)

Staple

Staple

Fold

First Class
Permit No. 229
El Segundo,
California

BUSINESS REPLY MAIL

No postage stamp necessary if mailed in the United States

Postage will be paid by

Xerox Corporation
701 South Aviation Boulevard
El Segundo, California 90245

Attn: Programming Publications

Fold

XEROX

701 South Aviation Boulevard
El Segundo, California 90245
213 679-4511

XEROX® is a trademark of XEROX CORPORATION