Xerox BPM/BTM

Sigma 5-8 Computers

Subsystems and Utilities Technical Manual

90 30 61A

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

XEROX

Xerox BPM/BTM

Sigma 5-8 Computers

Subsystems and Utilities

Technical Manual

90 30 61A

September 1973

Price: $11.00

Printed in U.S.A.

NOTICE

With the exception of the FAST SAVE processor description, all subsystems and utility processors described in this publication reflect operation under the H00 version of the BPM and BTM operating systems. The FAST SAVE processor described herein reflects operation under the G00 version of BPM/BTM. Other available subsystems that operate under BPM/BTM are listed under "Related Publications" below.

RELATED PUBLICATIONS

Title	Publication No.
Xerox Sigma 5 Computer/Reference Manual	90 09 59
Xerox Sigma 6 Computer/Reference Manual	90 17 13
Xerox Sigma 7 Computer/Reference Manual	90 09 50
Xerox Sigma 8 Computer/Reference Manual	90 17 49
Xerox Batch Time-Sharing Monitor (BTM)/TS Reference Manual	90 15 77
Xerox Batch Time-Sharing Monitor (BTM)/TS User's Guide	90 16 79
Xerox Batch Processing Monitor (BPM)/BP, RT Reference Manual	90 09 54
Xerox Batch Processing Monitor (BPM) and Batch Time-Sharing Monitor (BTM)/SM Reference Manual	90 17 41
Xerox Batch Processing Monitor (BPM) and Batch Time-Sharing Monitor (BTM)/OPS Reference Manual	90 11 98
Xerox Batch Processing Monitor (BPM)/System Technical Manual	90 15 28
Xerox BPM/BTM/UTS/Overlay Loader Technical Manual	90 18 03
Xerox Batch Time-Sharing Monitor (BTM)/Delta Subsystem Technical Manual	90 18 79
Xerox BPM/BTM/UTS/System Generation Technical Manual	90 18 77
Xerox BTM Edit Subsystem Technical Manual	90 19 11
Xerox BPM/BTM/UTS PCL Technical Manual	90 19 32
Xerox Volume Initialization (VOLINIT) Technical Manual	90 30 56

Manual Content Codes: BP – batch processing, LN – language, OPS – operations, RP – remote processing, RT – real-time, SM – system management, TS – time-sharing, UT – utilities.

CONTENTS

PREFACE viii

GLOSSARY ix

1.0 INTRODUCTION 1-1

1.1 Batch Processors 1-1
1.2 Subsystems 1-1
1.3 General 1-1

2.0 LOPE PROCESSOR/SUBSYSTEM 2-1

2.1 Functional Overview 2-1
2.1.1 Purpose 2-1
2.1.2 Restrictions 2-1
2.1.3 Batch Operation 2-1
2.1.4 On-Line Operation 2-3
2.1.5 Loader-Constructed DCBs 2-7
2.1.6 Map 2-7
2.1.7 Diagnostics 2-7
2.1.8 Severity Level 2-8
2.1.9 Background Memory Layout 2-8
2.2 Interfaces 2-8
2.2.1 LOPE Processor/LOAD Subsystem 2-8
2.2.2 Debugging LOPE 2-8
2.2.3 Input 2-8
2.2.4 Output 2-8
2.2.5 Loader-Generated Tables 2-18
2.2.6 LOAD Subsystem 2-21
2.2.7 Debug Program, DELTA 2-24
2.2.8 Accounting 2-24
2.2.9 Monitor 2-24
2.2.10 CCI 2-24
2.2.11 Stacks 2-24
2.2.12 Use of Memory 2-28
2.2.13 Example 2-31
2.3 Operational Overview 2-35
2.3.1 Description 2-35
2.3.2 Flowchart 2-36
2.4 Module Analysis 2-36
2.4.1 Start 2-36
2.4.2 Library 2-41
2.4.3 Loader 2-46
2.4.4 Endload 2-50
2.4.5 Execute 2-55
2.5 Subroutine Analysis 2-56
2.5.1 ASNMERG 2-56
2.5.2 BINTOHEX 2-58
2.5.3 BYTOUT, WRTREC 2-58
2.5.4 CHKDECLD 2-58
2.5.5 CKDCB 2-58
2.5.6 CRLF 2-59
2.5.7 GCBYTE 2-59
2.5.8 GCBYTEA 2-59
2.5.9 GETEFHED 2-59
2.5.10 GETULOC 2-59
2.5.11 PRESSTK 2-60
2.5.12 PRTNXT 2-60
2.5.13 PRTERR 2-60
2.5.14 RFDFSRCH 2-61
2.5.15 SETMODE 2-61
2.5.16 SETSIZE 2-61
2.5.17 STCR 2-62
2.5.18 STORFLD 2-62
2.5.19 USYMPRNT 2-62
2.5.20 WRTREC 2-62
2.5.21 WTTY, WTTY1 2-62

3.0 RUN 3-1

3.1 Functional Overview 3-1
3.1.1 General Description 3-1
3.1.2 Error Messages 3-2
3.1.3 Restriction 3-3
3.2 Interfaces 3-3
3.2.1 To the BTM Executive 3-3
3.2.2 To the User 3-4
3.3 Operational Overview 3-5
3.4 Module Analysis 3-6
3.4.1 ENTRY 3-6
3.4.2 GOTOUSER 3-6
3.4.3 TLOC 3-8
3.4.4 CHKBRK 3-8
3.4.5 OPDIS 3-9
3.4.6 OPEN 3-9
3.4.7 CLOSE 3-9
3.4.8 OPNXT 3-10
3.4.9 OPPREV 3-10
3.4.10 OPINDR 3-10
3.4.11 EHEH 3-11
3.4.12 MULTDIS 3-11
3.4.13 GOPRO 3-12
3.4.14 SEMIB 3-12
3.4.15 CLRALL 3-12
3.4.16 CLRBRK 3-13
3.4.17 DISBRK 3-13
3.5 Subroutine Analysis 3-14
3.5.1 RDVAL 3-14
3.5.2 MORECC 3-14
3.5.3 SINCK 3-15
3.5.4 SEMICK 3-15
3.5.5 SETSEG 3-15
3.5.6 GTSVAL 3-16
3.5.7 CHKSERV 3-16
3.5.8 GP 3-17
3.5.9 FP 3-17
3.5.10 SMPRT 3-17
3.5.11 GL 3-18
3.5.12 GCP 3-18
3.5.13 FCP 3-19
3.5.14 TIME 3-19
3.5.15 SEGLD 3-19
3.5.16 CHKEXU 3-20
3.5.17 TYCRLF 3-20

3.5.18 TYM ____ 3-21
3.5.19 GETTXT ____ 3-21
3.5.20 LMNAB;LMNER;LMSZER ____ 3-21
3.5.21 RDSEG ____ 3-22
3.5.22 SETSWAP ____ 3-22
3.5.23 SYMS ____ 3-22
3.5.24 RELOCR ____ 3-23
3.5.25 RDWNOB ____ 3-23
3.5.26 READWD ____ 3-23
3.5.27 STORWD ____ 3-24
3.5.28 GPAGIN ____ 3-24
3.5.29 WRPG ____ 3-25
3.5.30 RDPG ____ 3-25
3.5.31 CHKADRNX;CHKPTRNX; CHKPTR ____ 3-25
3.5.32 HEXOUT ____ 3-26
3.5.33 SYMBOUT ____ 3-26
3.5.34 SEIFS ____ 3-26

4.0 BPM SUBSYSTEM 4-1

4.1 Functional Overview ____ 4-1
4.1.1 Purpose ____ 4-1
4.1.2 Input ____ 4-1
4.1.3 Error Messages ____ 4-2
4.2 Interfaces ____ 4-3
4.2.1 Operating System ____ 4-3
4.3 Operational Overview ____ 4-3
4.3.1 Description ____ 4-3
4.3.2 DCB's ____ 4-3
4.3.3 Flowchart ____ 4-3
4.4 Module Analysis ____ 4-3
4.4.1 BEGIN ____ 4-5
4.4.2 DELETE ____ 4-5
4.4.3 STATUS ____ 4-5
4.4.4 INSERT ____ 4-9
4.4.5 FILETYPE ____ 4-11
4.4.6 EDIT ____ 4-11
4.4.7 EDITO ____ 4-11
4.5 Subroutine Analysis ____ 4-14
4.5.1 GO ____ 4-14
4.5.2 TYPE ____ 4-14
4.5.3 REPLACE ____ 4-15
4.5.4 SWAP ____ 4-15
4.5.5 DELETEE ____ 4-16
4.5.6 TRANSFER ____ 4-16
4.5.7 CRLF ____ 4-17
4.5.8 TYPEMESS ____ 4-17
4.5.9 YESNO ____ 4-17
4.5.10 INEBCHEX ____ 4-18
4.5.11 HEXDEC ____ 4-18
4.5.12 HEXEBC ____ 4-19
4.5.13 PUTLINE ____ 4-19
4.5.14 FINDLINE ____ 4-20
4.5.15 RUNDOWN ____ 4-20
4.5.16 BANGHIT ____ 4-21
4.5.17 EODCARD ____ 4-21
4.5.18 FINCARD ____ 4-22
4.5.19 JOBCARD ____ 4-22
4.5.20 NAMECOMP ____ 4-22
4.5.21 EXTACT ____ 4-23
4.5.22 PRIOR ____ 4-23
4.5.23 COMPALL ____ 4-23
4.5.24 SERCH ____ 4-24
4.5.25 CHKLIM ____ 4-24
4.5.26 SERDOWN ____ 4-25
4.5.27 GETINP ____ 4-25
4.5.28 TYPELINE ____ 4-26
4.5.29 DELLINE ____ 4-26
4.5.30 TRANSLST ____ 4-27
4.5.31 FINDEDIT ____ 4-27
4.5.32 ABNDEL ____ 4-27
4.5.33 NONO, COMMOUT ____ 4-28
4.5.34 NOROOM ____ 4-28
4.5.35 ABNINS ____ 4-29
4.5.36 NOINPT ____ 4-29
4.5.37 NOJOBC ____ 4-29
4.5.38 FINDER ____ 4-30
4.5.39 ABNINS ____ 4-30
4.5.40 NAMER; JOBCERR ____ 4-31
4.5.41 ACCTER; EXACER; PRIOER ____ 4-31
4.5.42 JOBCER ____ 4-31
4.5.43 INVL ____ 4-32

5.0 SUPER PROCESSOR/SUBSYSTEM 5-1

5.1 Functional Overview ____ 5-1
5.1.1 Purpose ____ 5-1
5.1.2 Input ____ 5-1
5.1.3 Batch Operation ____ 5-2
5.1.4 On-Line Operation ____ 5-2
5.1.5 Error Messages ____ 5-3
5.2 Interfaces ____ 5-4
5.2.1 Operating System ____ 5-4
5.2.2 CCI ____ 5-4
5.3 Operational Overview ____ 5-4
5.3.1 Description ____ 5-4
5.3.2 DCBs ____ 5-4
5.3.3 Flowchart ____ 5-4
5.4 Module Analysis ____ 5-4
5.4.1 START ____ 5-4
5.4.2 Control-Record ____ 5-6
5.4.3 USERS ____ 5-7
5.4.4 KILL ____ 5-8
5.4.5 LIST ____ 5-10
5.4.6 STATS ____ 5-11
5.4.7 DELSTATS ____ 5-13
5.4.8 PASSWORDS ____ 5-15
5.4.9 Input-Syntax-Error ____ 5-17
5.4.10 Abnormal-Exit ____ 5-17
5.4.11 Normal Exit ____ 5-18
5.5 Subroutine Analysis ____ 5-18
5.5.1 CVTDEC ____ 5-18
5.5.2 CVTNSTOR ____ 5-18
5.5.3 DATER ____ 5-19
5.5.4 FNDCOD ____ 5-19
5.5.5 GACNTN ____ 5-19
5.5.6 GDFLD and GHFLD ____ 5-20
5.5.7 GEBFLD ____ 5-21
5.5.8 GHFLD ____ 5-21
5.5.9 HELP ____ 5-21
5.5.10 INITLOG ____ 5-22
5.5.11 PAGE ____ 5-22
5.5.12 PRNT, PRNTV ____ 5-23

5.5.13 PRTERR ____ 5-23
5.5.14 PRTLIST ____ 5-23
5.5.15 PRTSTAT ____ 5-24
5.5.16 RDLOG and RDLOGS ____ 5-24
5.5.17 RDLOGACNT ____ 5-24
5.5.18 RDLOGS ____ 5-25
5.5.19 RESETLOG ____ 5-25
5.5.20 REWRLOG and REWRLOGS ____ 5-25
5.5.21 UCHKFLD ____ 5-25
5.5.22 WRLOG, REWRLOG, and REWRLOGS ____ 5-26
5.6 Authorization File, :USERLG ____ 5-26

6.0 FPURGE PROCESSOR 6-1

6.1 Functional Overview ____ 6-1
6.2 Interfaces ____ 6-1
6.2.1 Hardware Requirements ____ 6-1
6.2.2 Operational Requirements ____ 6-1
6.3 Operational Overview ____ 6-1
6.4 Module Analysis ____ 6-4
6.4.1 Control Cards ____ 6-4
6.4.2 SKPPAR ____ 6-6
6.4.3 STRPURG ____ 6-8
6.4.4 LGSUPRG ____ 6-10
6.4.5 GETC ____ 6-10
6.4.6 OPNXACCT ____ 6-10
6.4.7 CLRDCB ____ 6-10
6.4.8 PRINT ____ 6-11
6.4.9 WRTTP ____ 6-11
6.4.10 RESTORE ____ 6-11
6.4.11 MISFILS ____ 6-12
6.4.12 GETOUT ____ 6-12

7.0 FERRET SUBSYSTEM 7-1

7.1 Functional Overview ____ 7-1
7.1.1 Purpose ____ 7-1
7.1.2 Error Messages ____ 7-1
7.2 Interfaces ____ 7-2
7.3 Operational Overview ____ 7-2
7.4 Module Analysis ____ 7-12
7.4.1 FERRET ____ 7-12
7.4.2 LIST, REVIEW, GRANULES ____ 7-12
7.4.3 CHECK ____ 7-13
7.4.4 ACTIVCK ____ 7-13
7.4.5 DELETE ____ 7-14
7.4.6 COPY, KOPY ____ 7-14
7.4.7 PUNCH ____ 7-14
7.4.8 EXAMINE, INSPECT ____ 7-15
7.4.9 STATS ____ 7-15
7.4.10 MESSAGE ____ 7-16
7.4.11 NOSEP ____ 7-16
7.5 Service Subroutines ____ 7-16
7.5.1 CVDECOUT ____ 7-16
7.5.2 GETFLD ____ 7-17
7.5.3 GETNAM ____ 7-17
7.5.4 SCANOPEN ____ 7-17
7.5.5 LOCCODE ____ 7-18

8.0 EDCON 8-1

8.1 Functional Overview ____ 8-1
8.1.1 General Description ____ 8-1
8.1.2 Commands ____ 8-1
8.2 Interfaces ____ 8-2
8.3 Operational Overview ____ 8-2
8.4 Module Analysis ____ 8-2
8.4.1 MASTERPARSER ____ 8-4
8.4.2 PARSE:BUILD ____ 8-4
8.4.3 PARSE:COMPRESS, PARSE:WRITE ____ 8-4
8.4.4 PARSE:COPY ____ 8-5
8.4.5 PARSE:MERGE ____ 8-5
8.4.6 PARSE:DELETE, PARSE:LIST ____ 8-6
8.4.7 PARSE:END ____ 8-6
8.4.8 MASTEREXECUTIVE ____ 8-6
8.4.9 F:BUILD ____ 8-7
8.4.10 F:COMPRESS, F:WRITE ____ 8-7
8.4.11 F:COPY ____ 8-8
8.4.12 F:DELETE ____ 8-12
8.4.13 F:END ____ 8-12
8.4.14 F:LIST ____ 8-12
8.4.15 F:MERGE ____ 8-12
8.4.16 ADDCDTPARAM ____ 8-16
8.4.17 ADJINT ____ 8-16
8.4.18 BINTODEC ____ 8-17
8.4.19 BLANKBUF ____ 8-17
8.4.20 CLOSE ____ 8-17
8.4.21 CLOSE2 ____ 8-17
8.4.22 COMPRESSLINE ____ 8-18
8.4.23 CMPL60 ____ 8-18
8.4.24 CMPL70 ____ 8-19
8.4.25 DELETE ____ 8-19
8.4.26 DELETEFILE ____ 8-22
8.4.27 BADIO1 ____ 8-22
8.4.28 GETFILEID ____ 8-22
8.4.29 GETNEXTNAME ____ 8-23
8.4.30 GETNEXTPARAM ____ 8-23
8.4.31 MOVESEQ ____ 8-31
8.4.32 NEWCDTENTRY ____ 8-32
8.4.33 OPEN ____ 8-32
8.4.34 OPENINIT ____ 8-32
8.4.35 OPENNEW ____ 8-33
8.4.36 OPEN2, OPEN3 ____ 8-33
8.4.37 READNXTRANDOM ____ 8-34
8.4.38 READRANDOM ____ 8-34
8.4.39 READSEQUEN ____ 8-34
8.4.40 READSEQUEN2 ____ 8-35
8.4.41 READTELETYPE2 ____ 8-35
8.4.42 RECONSTRUCT$LINE ____ 8-36
8.4.43 RECL60 ____ 8-36
8.4.44 REPSEQ ____ 8-37
8.4.45 SETEOD ____ 8-37
8.4.46 SETKEY ____ 8-37
8.4.47 SETLASTKEY ____ 8-38
8.4.48 TYPECERR ____ 8-38
8.4.49 TYPEMSG ____ 8-38
8.4.50 WRITECO ____ 8-38
8.4.51 WRITENEWRANDOM ____ 8-39
8.4.52 WRITERANDOM ____ 8-39
8.4.53 WRITE2 ____ 8-40
8.4.54 WRITE$EO ____ 8-40

9.0 FAST SAVE PROCESSOR 9-1

9.1 Functional Overview 9-1
9.2 Interface 9-1
9.3 Operational Overview 9-1
9.3.1 Command Summary 9-2
9.3.2 Data Cards 9-3
9.3.3 Error Message Summary 9-4
9.3.4 Error SNAPS 9-7
9.3.5 Read Ahead Logic 9-7
9.4 Module Analysis 9-8
9.4.1 INITIATE 9-8
9.4.2 INITIATE1 9-8
9.4.3 SPECINIT 9-9
9.4.4 INITIATE4 9-9
9.4.5 INITIATE5 9-10
9.4.6 RDCARD 9-10
9.4.7 RCD 9-11
9.4.8 INTER 9-11
9.4.9 NOIPRI 9-11
9.4.10 NACN 9-12
9.4.11 NOTLB 9-14
9.4.12 GETFILE 9-14
9.4.13 FILECHKS 9-15
9.4.14 NOFCK 9-16
9.4.15 GETMIX 9-17
9.4.16 QUEREC 9-18
9.4.17 GETKEY1 9-18
9.4.18 MOVEKEY 9-18
9.4.19 MOVE 9-19
9.4.20 GETKEY 9-19
9.4.21 SCHEDULE 9-20
9.4.22 QUEMIX 9-20
9.4.23 RANFILE 9-21
9.4.24 GETTBUF 9-21
9.4.25 GETIBUF 9-22
9.4.26 GETFOUR 9-22
9.4.27 BUILD 9-22
9.4.28 STKSIZR 9-22
9.4.29 QSECTOR 9-22
9.4.30 DISCIO2 9-23
9.4.31 DENAC 9-23
9.4.32 FITENAC 9-23
9.4.33 ENCOM 9-24
9.4.34 MIXENAC 9-24
9.4.35 MTIO 9-24
9.4.36 SIMULATE 9-24
9.4.37 WTENAC 9-25
9.4.38 SENTENAC 9-25
9.4.39 NEWREEL 9-25
9.4.40 WRTDAT 9-25
9.4.41 DCTXERR/OPNABN/OPNERR 9-26
9.4.42 GOEOR 9-26
9.4.43 EOFQ 9-26
9.4.44 BOFQUE 9-26
9.4.45 MOVEI 9-26
9.4.46 WRTMARK 9-27
9.4.47 NOTENUFF 9-27
9.4.48 ENDUP 9-27
9.4.49 ADDONE 9-27
9.4.50 ADERROR 9-28
9.4.51 IORUNDOWN 9-28
9.4.52 FAILURE 9-28
9.4.53 DISPRUNTOTL 9-28
9.4.54 DTOGRAN 9-28
9.4.55 FDERR 9-29
9.4.56 FITSNAP/FITERR 9-29
9.4.57 LISTFD 9-29
9.4.58 LISTFIT 9-29
9.4.59 LISTOUTBUF 9-29
9.4.60 LISTMIX 9-30
9.4.61 LISTAD 9-30
9.4.62 MIXSNAP/MIXERR 9-30
9.4.63 MIXRATERR 9-30
9.4.64 DATAERR 9-30
9.4.65 FDDONE 9-31
9.4.66 ENDOFD 9-31
9.4.67 CKT10 9-31
9.4.68 CODESCAN 9-31
9.4.69 MOVENTRY 9-32
9.4.70 HEXTODEC 9-32
9.4.71 FDLETE 9-32
9.4.72 ADELETE 9-32
9.4.73 TACNT/TFILNME 9-32
9.4.74 ACCK 9-32
9.4.75 READFAIL 9-33
9.4.76 READFAIL2 9-33
9.4.77 READFAIL3 9-33
9.4.78 READFAIL5 9-33
9.4.79 LPRINT 9-33
9.4.80 PLIST 9-33
9.4.81 BUFSET 9-34
9.4.82 PRINT 9-34
9.4.83 SPACE 9-34
9.4.84 TYPEIO/TYPEIO2 9-34
9.5 Detailed Flowchart of Fast Save Processor 9-35
9.6 Tables 9-72

10.0 FILE ANALYZER (FANALYZE) PROCESSOR 10-1

10.1 Functional Overview 10-1
10.2 Interface 10-1
10.3 Operational Overview 10-1
10.3.1 File Analyzer Options 10-1
10.3.2 Error Processing 10-2
10.3.3 Error Messages 10-3
10.4 Module Analysis 10-5

FIGURES

2-1. Load Module Format 2-9

2-2. Format of REF/DEF Stack That Is Part of The Load Module 2-11

2-3. :BLIB Format for Each Page 2-12

2-4. Sample LOPE Map Printout 2-14

2-5. Internal Symbol Table 2-17

2-6. Background Memory Layout, LOPE Processor — 2-18
2-7. Task Control Block Format — 2-19
2-8. DCBTAB (Name Table) — 2-20
2-9. Loader-Constructed System DCBs — 2-21
2-10. Standard First X'40' Words of LOAD Subsystem — 2-22
2-11. Declaration Stack — 2-25
2-12. REF/DEF Stack — 2-26
2-13. Expression Stack — 2-27
2-14. Forward Reference Stack — 2-28
2-15. Labels used in Allocating Memory — 2-29
2-16. Layout of Memory after Initialization — 2-30
2-17. Flowchart of Overall Processing — 2-37
2-18. Flowchart of Start Module — 2-42
2-19. Flowchart for Library Modules — 2-45
2-20. Flowchart for Loader Module — 2-51
2-21. Flowchart for Endload Module — 2-54
2-22. Flowchart for Execute Module — 2-57

3-1. RUN Commands — 3-2
3-2. RUN SUBSYSTEM IN MEMORY — 3-4
3-3. USER IN MEMORY — 3-5
3-4. Operational Overview of RUN — 3-7

4-1. Operational Overview of BPM Subsystem — 4-4
4-2. Flow Chart for DELETE — 4-7
4-3. Flow Chart for STATUS — 4-8
4-4. Flow Chart of INSERT — 4-10
4-5. Flow Chart of EDIT — 4-13

5-1. Operational Overview of SUPER — 5-5
5-2. Flow Chart for USERS — 5-9
5-3. Flow Chart for LIST — 5-12
5-4. Flow Chart for STATS — 5-14
5-5. Flow Chart for DELSTATS — 5-16

6-1. Flow Chart of Major Functions — 6-3
6-2. Flow Chart of CCI — 6-5
6-3. Flow Chart of SKPPAR — 6-7
6-4. Flow Chart of STRPURG — 6-9

7-1. Flow Diagram of FERRET — 7-3

8-1. Overall Flow Diagram of EDCON — 8-3
8-1A. Flow Diagram of F:COPY — 8-9
8-2. Flow Diagram of F:MERGE — 8-13
8-3. Flow Diagram of COMPRESSLINE — 8-20
8-4. Flow Diagram of GETNEXTNAME — 8-24
8-5. Flow Diagram of GETNEXTPARAM — 8-26

9-1. Detailed Flowchart of FILE SAVE — 9-36
9-2. Formats of Paging Tables — 9-71

TABLES

2-1. Load Map Abbreviations — 2-13
6-1. FPURGE Options — 6-13
8-1. Command Description Table (CDT) — 8-40
8-2. Command Table — 8-41
9-1. Index to FAST SAVE Flowchart — 9-35
9-2. Table Key Lengths — 9-72
9-3. First Key Displacement in Sectors — 9-72
9-4. Account Directory Displacement — 9-72
9-5. File Directory Displacement — 9-72
9-6. Master Index Displacement — 9-73
9-7. I/O Queueing Function Codes — 9-73
9-8. Data Names and Definitions — 9-74

PREFACE

This document describes the purpose and architecture of the following subsystems and utility processors that operate within the environment of BPM/BTM:

LOPE	FERRET
RUN	EDCON
BPM	FAST SAVE
SUPER	FANALYZE
FPURGE	

This manual is intended for use by maintenance programmers as a guide through the listings supplied with the above processors. It is assumed that the reader is familiar with both the usage of BPM/BTM Monitor services and the Sigma Standard Object Language (see the BPM/BTM/SM Reference Manual, 90 17 41).

GLOSSARY

AJIT The on-line job information table, a Monitor table of information pertinent to the job currently in execution.

:BLIB (Binary Object Language Library). The name of the file that is the library in an account.

binary input Input from the device to which the BI (binary input) operational label is assigned.

CCI (Control Command Interpreter). A batch processor that reads the control command.

control section A relocatable section of the program into which assembled generative statements are loaded.

core image That part of a load module that is laid into core at execution time.

DCB (Device Control Block) A table in the executing program that contains the information used by the Monitor to perform an I/O operation.

DCB Name Table A loader-built table that directs the Monitor to the location of a DCB in a program.

declaration A load item of the object language that associates a symbolic name with a type of item that provides linkages between object modules.

declaration stack A loader stack that keeps track of the declarations made in a ROM.

definition A load item of the object language that equates a source language symbol with a value.

dummy section A type of program section that provides a way for more than one object module to reference the same data via an external definition used as the label for the dummy section.

expression A load item of the object language that may include sums and differences of constants, addresses, and external and forward references (which when defined will be constants or addresses) in order to represent a value.

expression stack A loader stack that contains expressions defining external forward references, and locations in the core image of the program.

external definition a load item that assigns a specific value to the symbolic name associated with the external definition. The external definition provides linkage between object modules since the external definition allows the specified symbolic name to be used as an external reference in another ROM.

external reference A reference to a declared symbolic name that is not defined within the object module in which the reference occurs. An external reference can be satisfied only if the name is defined by an external definition in another object module.

forward reference A reference to a symbolic name that will be defined later in the object module.

forward reference stack A loader stack that keeps track of the forward references made in a ROM.

GO file In batch, a temporary file of ROMs formed by a processor. Such modules may be retrieved by use of a LOPE or LOAD control command.

HEAD The key to one of the records of a load module file. The record contains basic size and source information.

idG In batch, the file name that the loader uses to access the GO file. The idG file is a temporary file.

idL In batch, the file name assigned to a load module if no name is specified through the LMN option. The idL file is a temporary file.

JIT (Job Information Table) A Monitor table of information pertinent to the job currently in execution.

library module A module that the loader may combine with ROMs to form a load module or the core image portion of the load module.

load information Control information, data, and instructions generated by a processor and contained in one or more modules capable of being linked to form an executable program.

load item The component of the object language, a load control byte followed by any additional bytes of load information pertaining to the function specified.

load map A listing of information about the storage locations used by a program.

load module A keyed file containing an executable program. Load modules are output by loaders and several other processors.

object language The standard binary language in which the output of an assembler or compiler is expressed.

object module The series of records containing the load information pertaining to a single program element.

overlay program A segmented program in which different elements (i.e., segments) occupy the same core storage area as the program is executed.

primary reference A reference to a symbolic name that is not defined within the object module in which the reference occurs. A primary reference can only be satisfied if the name is defined by an external definition in another object module. A primary reference is capable of causing a load from a library.

REF/DEF stack A loader stack with entries for the value of control sections and external names - external definitions, primary references, and secondary references. The REF/DEF stack is part of the load module.

ROM (Relocatable Object Module). An input to the loader in SIGMA object language that was generated by an assembler or a compiler.

secondary reference A reference to a symbolic name that is not defined within the object module in which the reference occurs. A secondary reference does not cause a load from a library to satisfy the reference.

system library A group of standard routines in object language format, any of which may be incorporated into a program being loaded.

TCB (Task Control Block). A table of program control information built by a loader when a load module is formed. The TCB is part of the load module. The TCB contains a temp stack and the data required to allow reentry of library routines during program execution. The TCB is program-associated, not task associated.

1.0 INTRODUCTION

The Xerox BPM and BPM/BTM capability is a total package consisting of an operating system and program packages that assist the user in taking advantage of the operating system. These program packages are known as processors.

1.1 BATCH PROCESSORS

The control or service programs which are run in the batch mode under BPM are referred to as batch processors. CCI is a control processor while FPURGE is an example of a service processor.

1.2 SUBSYSTEMS

On-line processors are referred to as subsystems. These processors are available to the BTM terminal user and may either be language or service subsystems. EDIT is an on-line service processor and SYMBOL is an example of an on-line language subsystem.

1.3 GENERAL

Program packages may be written for both the batch and on-line modes. The two major differences that govern where the programs run are: (1) how they interface with the monitor and (2) how they are loaded.

When creating a system, usually the BPM or BPM/BTM monitor is loaded first and then the batch processors are loaded. By loading the system in this order, the batch processors are automatically biased at the core page boundary just above the end of the monitor. This page boundary is known as background lower limit. Batch processors may be biased during loading such that they are above Background Lower Limit but this is an inefficient use of core and should be avoided when possible. Batch processors are usually created at SYSGEN time by PASS3 and are included on the P. O. tape by the DEF processor. The processors are then in the :SYS account after the P. O. tape is booted and the system is operational. The batch processor is invoked by its load module name on a monitor control card. Any load module in the :SYS account may be run as a processor by merely using the format on the control card as follows:

```
!NAME  (NAME is a load module name.)
```

The on-line processors or subsystems are loaded differently than the batch processors. There are two basic differences: (1) the way they are biased and (2) the way they are named.

Each on-line subsystem is given a load module name when it is loaded and this name may end in the character ":" (colon). BTM initialization knows that a copy of the load module with its name ending in a ":" (colon) in the :SYS account is a subsystem. A copy of each subsystem found in the :SYS account is moved to the swapping device and entered into the :BTM account.

The on-line processors (subsystems) must be loaded with an absolute bias. The bias used will be different each time a new system is generated and USERSIZE is varied for a given core size. The bias at which the subsystems must be located is:

```
BIAS = CORESIZE-USERSIZE.
```

Subsystems must be coded according to the rules set down in the section entitled "SUBSYSTEM INTERFACE" in the Xerox Batch Time-Sharing Monitor Reference Manual (90 15 77). These coding requirements must be observed by all BTM subsystems.

The BPM batch processors communicate with the BPM monitor via CAL1 calls. The CAL1s provide the total and complete range of monitor services available to the batch processors.

BTM subsystems use both CAL1 and CAL3 calls to provide for monitor services. A BTM subsystem is restricted to a subset of the possible CAL1s available to batch processors and some CAL1s not available to batch processors. System control and memory management functions available to the BTM subsystem are described in

the section entitled "BTM SYSTEM CALS" in the Xerox Batch Time-Sharing (BTM) Reference Manual (90 15 77). The BPM system CALs available to BTM subsystems are listed in the section entitled "BPM SYSTEM CALS" in the BTM Reference Manual (90 15 77).

The complete list and description of the BPM Monitor services available to batch processors is contained in the Xerox Batch Processing Monitor (BPM) Reference Manual (90 09 54).

2.0 LOPE PROCESSOR/SUBSYSTEM

2.1 FUNCTIONAL OVERVIEW

2.1.1 Purpose

LOPE (Load in One Pass and Execute) is a loader, either a BPM processor or a BTM subsystem, which converts object module output from a compiler or an assembler into executable format. In batch mode, the loader creates either a nonoverlaid load module in secondary storage or the core image portion of the load module, to which control is then transferred. In on-line mode, the loader creates only the core image of the program but execution is not automatic. If the user has requested execution, the LOAD subsystem monitors the execution or, if the user has requested the debug option, the LOAD subsystem transfers control to the DELTA subsystem to allow DELTA to monitor the execution. LOPE operates as a one-pass program. To build a load module or the core image portion of the load module, LOPE performs functions expected of any loader operating under the BPM Operating System.

1. Process ROMs to produce continuous sections of data, procedure, and DCBs or static data, ensuring a page boundary for these three protection types -- 00, 01, and 10 respectively.
2. Resolve references among ROMs.
3. Access libraries to satisfy primary references.
4. Build DCBs.
5. Build a DCB Name Table for Monitor use.
6. Build a Task Control Block.

2.1.2 Restrictions

1. LOPE will not form an overlaid load module.
2. LOPE will not load a ROM with a protection type other than 00 (except library routines) since LOPE builds all DCBs, DCB tables, and tree tables (control section type 01).
3. LOPE does not allow user-built DCBs. Loader-built DCBs must be used. See section 2.2.5,4. DCBs.
4. LOPE loads ROM input only. LOPE does not load load modules. The library formed when the PERM,LIB option is specified is a collection of ROMs, for which no processing has been done. The library contains no load modules.
5. LOPE replaces the library whenever the PERM,LIB option is specified. Since LOPE does not add to an existing library, each time the PERM,LIB option is specified LOPE requires that the user build the complete library desired.
6. LOPE requires that dummy sections be defined at maximum size in the initial definition.

2.1.3 Batch Operation

1. LOPE accepts object module input from one or more BI files, GO files, element files, or libraries. LOPE loads first from the BI file, then from the GO file, then from specified element files in order of appearance

in the control command, then from libraries to satisfy primary references -- from up to five libraries of specified accounts in order of appearance in the control command and then from the system library.

2. DCBs.

   a. The M:C DCB is used to read the LOPE control command.

   b. The M:BI DCB is used to read the BI device or file. The M:BI DCB may be assigned to a file in another account.

   c. The M:LI DCB is used to access libraries.

   d. The M:LL DCB is used to output the map.

   e. The M:LM DCB is used to build the load module.

   f. The M:DO DCB is used to output diagnostics.

3. The LOPE control command has the format:

   !LOPE [(option)]...[,(option)]

   where the options are:

   BI specifies that all ROMs on the BI device or file are to be loaded. If neither GO nor EF are specified, BI is assumed.

   GO specifies that all ROMs on the GO file are to be loaded.

   EF,(file name[,account[,password]])[,...] specifies that all ROMs in the designated element files are to be loaded. File name may consist of 1-8 alphanumeric characters, excluding delimiters such as, . + ( ) - and blank as imbedded characters.

   UNSAT,(account)[,...] specifies that the libraries, :BLIB, of the specified accounts are to be searched to satisfy any primary references. A maximum of five libraries may be specified.

   NOSYSLIB specifies that the system library, :BLIB in the :SYS account, is not to be searched.

   MAP specifies that a complete listing of external references and external definitions for the load module is to be output on the LL device.

   M10 specifies that each control or dummy section is to be loaded at the next greater multiple of 10_{16}.

   M100 specifies that each control or dummy section is to be loaded at the next greater multiple of 100_{16}.

   TSS,size specifies in hexadecimal the maximum word size of the Temporary Storage Stack (TSS) for the current job. The size is limited to X'7FFF' or the available core storage, whichever is less.

   PERM,LIB specifies that a library module is to be created. This option causes EXEC, LMN, MAP, and BIAS options to be ignored since the library formed is a collection of ROMs, for which no processing has been done. This option replaces, rather than updates, any existing library so the user must build the complete library desired. The library is :BLIB.

PERM specifies that the load module is to be permanent.

EXEC specifies that no load module is to be created, but the program is to be executed immediately. No RUN, DATA, or debug commands are allowed. This option causes BIAS, PERM, and LMN options to be ignored.

LMN,name specifies the name, 1-8 characters, to be given to the load module.

BIAS,value specifies the load bias of the created load module.

SL,value specifies the error severity level, hexadecimal 0-F, that will be tolerated by LOPE in forming a load module.

2.1.4 On-Line Operation

The characters typed by the LOAD sybsystem are shown in upper case for alphabetic characters and are underlined.

1. The LOAD subsystem accepts object module input from element files or through M:BI. The LOAD subsystem also loads library modules from the file :BLIB in any specified account and/or from the :BTM account.

2. DCBs.

   a. The M:BI DCB is used to read object module input. If desired, before entering the LOAD subsystem, assign M:BI to a file of object modules. See restriction in step 5 about file names. The default assignment is to the temporary file BOTEMPαC , where α is the COC line number in binary. BOTEMP.' is the default output file for on-line assemblers and compilers.

   b. The M:LI DCB is used to access libraries.

   c. The M:DO DCB is used to output diagnostics.

   d. The M:LO DCB is used to output the map.

   e. The F:X1 DCB is used to access the internal symbol table built for the debug program.

3. The command to enter the LOAD subsystem has the format:

   !LOAD

4. The element file request and response has the format:

   ELEMENT FILES: file name[(account[,password]) ... ,file name[(account[,password])]] CR

   where file name consists of 1-11 alphanumeric characters.

   See restriction in step 5. The total number of characters may not exceed the capacity of the input buffer, a system generation parameter that is typically 100 characters. To erase the element file list anytime before typing the carriage return, enter O X. If M:BI has been assigned to a file or, by default, to the BOTEMPα temporary file, enter only a carriage return.

5. Note that in executing under the debug option, element file names should be restricted to seven characters. In the symbol table built by the LOAD subsystem for the debug program, only the first seven characters of any symbol are used. If symbols, including element file names, have duplicate initial seven characters and length, only the first symbol encountered is retained.

6. The options request and response has the format:

   OPTIONS: option[,option]...[,option](CR)

   where the options are:

   $U(a_1,a_2...,a_5)$ specifies that a search of the file :BLIB (Binary Object Language Library) in each of the accounts designated (5 maximum) should be made for unsatisfied primary references before the optional search of :BLIB in the BTM system account, :BTM. Unless specified, no search of nonsystem accounts will be made.

   N specifies that no system library search is to be made for unsatisfied primary references. Unless specified, the file :BLIB (Binary Object Language Library) in the BTM system account, :BTM, will be searched for unsatisfied primary references. The file contains standard program modules for general use.

   P specifies that all programs defined by separate modules should begin on the next highest X'100' word boundary. The starting bias for a program is X'200' in the on-line memory area. The P option facilitates relating assembly listings to memory locations for debugging. Note that separate control sections within each module will be contiguous and will not begin on any particular boundary other than doubleword.

   M specifies that a load map of all external definitions and external references should be output through M:LO. Unless specified, only secondary and primary references will be output through M:DO (and M:LO if M:LO is assigned to a different file.) If the D option is specified and object module input has been assembled on-line by Symbol or in the background by Meta-Symbol with the symbolic debug (SD) option, undefined internal symbols for each element file will follow the normal map. Internal symbols are symbols not made external through the use of the DEF/REF directive nor defined under control of a LOCAL directive.

   L specifies that a library is to be created. No other option may be used. The library formed is a collection of ROMs. This option replaces, rather than updates, any existing library so the user must build the complete library desired. The library is :BLIB in the current account.

   D specifies that the program is to be executed under control of the debug program, enabling diagnostic and corrective operations on the program. The option is used mainly to check assembly language programs. However, the option may be used during the execution of any program that can be run under the LOAD subsystem. If the D option is specified and object module input has been assembled on-line by Symbol or in the background by Meta-Symbol with the symbolic debug (SD) option, undefined internal symbols for each element file will follow the normal map.

7. When the option list has been accepted, the LOAD subsystem loads the specified ROMs. If errors occur during loading, a diagnostic is output through M:DO (and M:LO if M:LO is assigned to a different file.) See section 2.2.4,5. diagnostics.

8. For any unsatisfied references to DCBs (F:alpha, where alpha is a name consisting of 1-8 alphanumeric characters), the LOAD subsystem will build a DCB with default assignment to the user terminal. The LOAD subsystem also requests specification of DCBs for which no reference occurs or for which the assignment is to the changed. Any FORTRAN DCBs for unit numbers other than 101-106 and 108 must be specified. The DCB request and response has the format:

   F: [n]$_L$=name$_1$(account [, password)] [, option]... [, option] CR

   where

   n specifies the FORTRAN unit number (for FORTRAN programs) or an alphanumeric DCB name of up to 5 characters.

   name specifies the alphanumeric name (1-11 characters) of the file to which the DCB is to be assigned. A carriage return specifies default assignment to the user terminal.

   account specifies the account of an existing file if the account is not the user's account. Account may be up to 8 characters. The default is the log-in account.

   password specifies the password of an existing file. Password is 1-8 alphanumeric characters.

   option specifies one of three options: Options will be inserted into the DCB and may be monitored by ASSIGN commands or procedure calls.

   Function option

   IN specifies that the file is to be used for input only. When assigning an input file on secondary storage, specify IN. Otherwise an attempt to read before writing will cause the creation of a new file with the same name as the existing file since the default assignment in all SIGMA loader is OUTIN, a scratch file.

   OUT specifies that the file is to be used for output only.

   INOUT specifies that the file is to be used as an update file.

   OUTIN specifies that the file is to be used as a scratch file. OUTIN is the default.

   Release option REL specifies that the file is to be released when program execution terminates. The default is that the file will be saved.

   List option L specifies that the file will eventually be listed on a listing device (FORTRAN program only).

   The Loader prompts until a carriage return only is entered.

9. The Load subsystem issues a message on the highest error severity level encountered.

   SEV. LEV = n

   where n is a hexadecimal digit.

10. If the debug option was specified, the LOAD subsystem requests satisfaction of symbols.

a. The request and response for primary references has the format:

SATISFY EXTERNALS

[< name > value]↵

where

name specifies a primary reference listed in the load map, 1-63 alphanumeric characters.

value specifies an expression of the form:

DEF name (± hexadecimal constant) or

.hexadecimal constant

Three diagnostics may be issued

-NAME ERROR. An external name is invalid.

-CONSTANT ERROR. A hexadecimal constant is invalid.

-VALUE ERROR. The value field is invalid.

The Load subsystem prompts for a response until a carriage return is entered.

b. The request and response for internal references has the format:

SATISFY INTERNALS

*EF - file name

[< symbol > value]↵

where

symbol specifies an undefined symbol listed in the load map, 1-63 alphanumeric characters.

value specifies an expression of the form:

DEF name (±. hexadecimal constant)

symbol name (±. hexadecimal constant) where symbol is an existing internal symbol in the element file. If symbol name cannot be specified, a console message will be issued. The only time symbol name is prohibited is when the combined size of the LOAD subsystem, the loaded program, the internal symbol table, and the REF/DEF stack exceeds the size of the on-line memory area.

±. hexadecimal constant

The Load subsystem prompts for a response until a carriage return is entered. The user may leave any or all undefined references unsatisfied.

11. The execute inquiry and response has the format:

XEQ? { N / Y or null / S, address } CR

where

Y or CR only specifies yes, execute the program.

N specifies no, do not execute the program. Exit to the BTM executive.

S specifies execute the program using the address specified as the starting address.

address specifies either an external definition, optionally followed by a hexadecimal addend value, or a signed absolute hexadecimal address beginning with a period.

12. The LOAD subsystem monitors the execution of the user program and responds to errors with diagnostics. See section 2.2.4,5 diagnostics. The inquiry and response after a user strikes ESC has the format:

PROCEED? {Y / any other response}

2.1.5 Loader-Constructed DCBs

See section 2.2.5,4. DCBs.

2.1.6 Map

See figure 2-4.

2.1.7 Diagnostics

1. See section 2.2.4,5. for loader diagnostics.
2. In batch mode, the following diagnostics may be output by the Monitor when bringing a program into core storage for execution, running a load module.
   a. ABOVE BUL. The program bias is greaterthan background upper limit.
   b. ABS. CANNOT REL. The Monitor cannot relocate the program because the program is absolute.
   c. BAD IF, AND, OR NAME. The name referenced on an IF, AND, or ORdebug control command is invalid.
   d. BAD LOC NAME. The name referenced as a location identifier is invalid.
   e. BAD MOD NAME. The name referenced on a MODIFY debug control command is invalid.
   f. BAD PMD NAME. The name referenced on a PMD debug control command is invalid.
   g. BAD SNAP NAME. The name referenced on a SNAP debug control command is invalid.
   h. BAD START NAME. The name on the RUN control command specified as the starting address is invalid.
   i. BELOW BLL. The program bias is less than background lower limit.
   j. IO ERROR. An input/output error or IO ABN. abnormal condition has occurred.
   k. NO LOAD MODULE. The load module named is not available.
   l. LOC NOT IN SEG. The location referenced on a debug control command is illegal.
   m. SEV. LEV. EXCEEDED. The severity level specified on the RUN control command was less than the severity level of the load module.

n. STACK OVERFLOW. The program will not fit in core.

o. TOO MANY DBUGS. The number of debug control commands specified creates an overflow of the one page of core allocated for debug tables.

2.1.8 Severity Level

See section 2.2.4,6. Severity Level.

2.1.9 Background Memory Layout

See figure 2-6.

2.2 INTERFACES

2.2.1 LOPE Processor/LOAD Subsystem

Module CN705429, BTMLOAD, is conditionally assembled to produce either the LOPE processor for background processing or the LOAD subsystem for on-line processing. The assembly parameter USER determines the version. USER is set to 1 for the LOPE processor or to 2 for the LOAD subsystem.

2.2.2 Debugging LOPE

LOPE may be run under DELTA for debugging. LOPE may be executed under the RUN subsystem.

!RUN

LOAD MODULE FID: LOPE (:SYS)

2.2.3 Input

1. ROMs.
   a. The loader converts ROMs output from a computer or an assembler into executable format. See the BPM Reference Manual, 90-09-54E, for the format of the object module.
   b. In on-line mode, when executing under the debug option, only the first seven characters of any symbol are given in the symbol table built by the LOAD subsystem for the debug program. If symbols have duplicate initial seven characters and length, only the first symbol encountered is retained. Since the same restriction applies to element file names, element file names should be restricted to seven characters when the binary input is created.
2. Library Input. See figure 2.2 for the format of the library record.
3. Assign Record. In batch mode, CCI builds a record of all assign information encountered during a job. The loader examines the record to see if any F: number DCBs have been entered. If the F: number is a primary reference in the user program, the loader generates a DCB with default entries for the name. The record is in the absolute area of secondary stroage and is accessed by the read absolute calls. See the BPM Technical Manual for the formats.

2.2.4 Output

1. Load Module.
   a. Lope procedures a load module. When a load module has been created, a !RUN control command will cause the program loader, PRGMLDR, to access the load module file, to modify and/or to relocate

the file, to lay the load module into core according to the specifications of the head and tree records, and to transfer control to the start address, whereupon the program is "in execution".

b. A load module is a keyed file with the name specified on the LOPE control command. The default name is the temporary ldL file. The keys and records of the load module are given in figure 2-1.

2. REF/DEF stack. The REF/DEF stack that is output as part of the load module has the same format as the REF/DEF stack created by the overlay loader. See figure 2-2 for the format. Note that the internal REF/DEF stack has a different format. See figure 2-12 for the format. The loader converts the REF/DEF stack to the format of the overlay loader before writing out the stack.

3. Library. The library is a collection of ROMs for which no processing has been done. See figure 2-3 for the format.

a. Record. Key = HEAD.

	0	8	16	24 31
0	80	00	FF	Number of bytes in HEAD record X'18'
1	ABS flag | L | SL -- final Severity level		Start address	
2	TCB address (doubleword address)		Module bias (doubleword address)	
3	Data (00) base (doubleword address)		Procedure (01) base (doubleword address)	
4	Static data (10) base (doubleword address)		Next available page (doubleword address)	
5	Maximum REF/DEF stack size		Tree size	

Figure 2-1. Load Module Format

b. Tree Table. Key = TREE

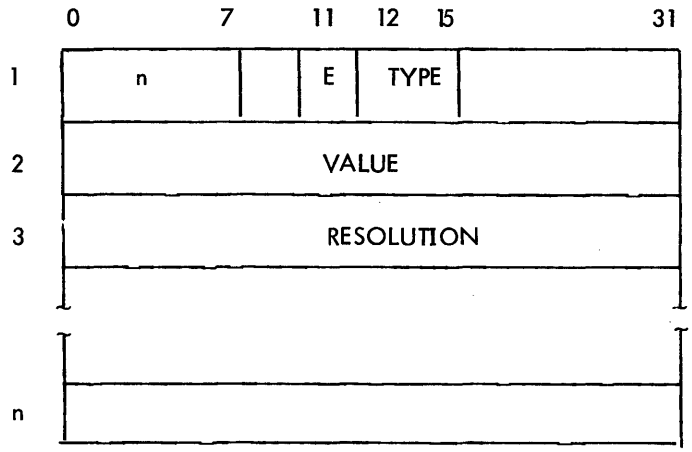
	0	16	31
-1	Size of tree table-12		
0 1 2	Segment name in TEXTC format		
3	ROM pointer	Back link (displacement from TREE)	
4	Forward link (displacement from TREE)	Overlay link (displacement from TREE)	
5	00 size (number of doublewords)	00 location (doubleword address)	
6	REF/DEF stack size	REF/DEF location (doubleword address)	
7	01 size (number of doublewords)	01 location (doubleword address)	
8	Expression size	Expression location (doubleword address)	
9	10 size (number of doublewords)	10 location (doubleword address)	
10	Used to monitor size of REF tables	0	

c. The following records are built.

Key	Record
00	REF/DEF stack
03	00 Control sections
05	01 Control sections

Note that the expression stack is not part of the load module since the stack is only needed to load a load module with a ROM, a feature absent in LOPE.

Figure 2-1. Load Module Format (cont.)



where:

n = number of words in this entry.

E = 1, if the entry has a VALUE

TYPE = 0 or 8 DEF

1 SREF

2 PREF

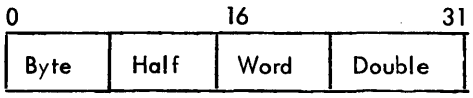
3 or B Dummy Section

4 or 6 Control Section

5 or 7 Forward Reference

VALUE = constant or address not a library

RESOLUTION = the resolution in which the VALUE is expressed. Resolution is of the form:

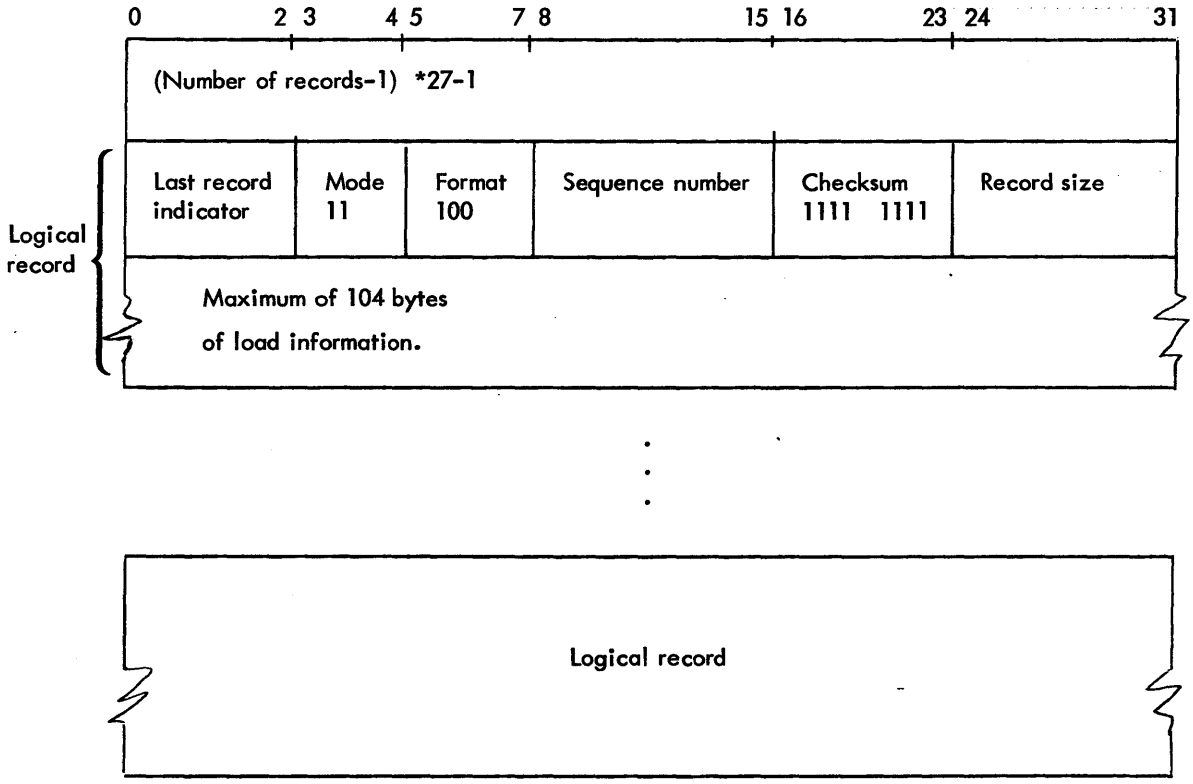


If the VALUE is a constant, the RESOLUTION word is 0.

If the VALUE is an address, one and only one byte of the RESOLUTION word is nonzero (viz., the appropriate byte = X'01').

If the RESOLUTION assumes a form different from either of the above, the value is of mixed resolution. (In this case the load module cannot be relocated and is forced ABS.)

Figure 2-2. Format of REF/DEF Stack That Is Part of The Load Module.



1. Logical record. Each record consists of 4 bytes of control information followed by a maximum of 104 bytes of load information, 108 bytes, the equivalent of 72 card columns.

2. Control information.

   a. Last record indicator is 000 for last record or 001 for not last record.

   b. Mode is always 11, specifying that the loader is to read binary information.

   c. Format is always 100, specifying object language format.

   d. Sequence number is zero for the first record of the module and is incremented by one for each subsequent record until the number recycles to zero after reaching 255.

   e. Checksum is always X'FF'.

   f. Record size specifies the number of bytes in the logical record. Record size is usually 108 bytes, X'6C', 27 words.

3. General description of :BLIB

   a. For each ROM, declare extenral definitions, then declare external references and define origin load items, then a special 02 load item for the library preceed the rest of the load items.

   b. The special 02 load item is followed by the declaration numbers of the external definitions in halfword entries.

Figure 2-3. :BLIB Format for Each Page

c. The physical records are written a page at a time.

d. If a record does not fit into the last few words of a page, these words contain garbage and the record begins on the next page.

e. If intermediate records are less than 108 bytes in size, LOPE writes 108 bytes anyway but reflects the logical size in the byte count.

Figure 2-3. :BLIB Format for Each Page (cont.)

4. Load Map. If the map option was specified by the user, a load map is produced at the completion of loading. The load map lists external definitions with the address at which each external definition has been loaded. The map shows the location of the beginning of all global symbols (DEFs), as well as DCBs. The map also lists unsatisfied external references and unused or double definitions. See Table 2-1 for the abbreviations used. See figure 2.4 for a sample load map.

Table 2-1. Load Map Abbreviations

Abbreviation	Meaning
SREF	Secondary reference.
PREF	Unsatisfied primary reference (REF).
DEF	External definition. The external definition was referenced and found.
UDEF	Unused definition. No external reference was made to this external definition.
DDEF	Double definition. More than one definition has been found for this name. The first definition found has been used.

```
 01:27 JAN 01,'01 ID=0013-F00
 JOB JOB,JOB
LOPE (MAP),(BI),(NOSYSLIB)

PREF              COMLIST
PREF              USECOM
PREF              REPCOM
PREF              IOSCU
PREF              IOSERCK
PREF              IOSEREC
PREF              RE:ENT
PREF              DCT1
PREF              DCT4
PREF              DCT7
PREF              DCT17
PREF              Y8
PREF              IOQ4
PREF              IOQ5
PREF              IOQ8
PREF              IOQ10
PREF              IOQ11
PREF              IOQ12
PREF              ROOTSYM
PREF              ROOTCNT
UDEF    5A00 0    LOWEST LOC
UDEF    5A64 0    START
UDEF    63AA 0    PATCH
UDEF    6498 0    DPAK
UDEF    6498 0    CMIO
UDEF    64C1 0    DPAKCU
UDEF    64C1 0    CMCU
UDEF    6532 0    NENTSYM
UDEF    6533 0    SYMBEGIN
UDEF    653A 0    DRINIT
UDEF    66CC 0    OFFSET
UDEF    6A7D 0    DSNAP
UDEF    6DA2 0    PACHSTRT
UDEF    6EEC 0    BEGIN
UDEF    6EED 0    DELINIT
UDEF    6F70 0    DL1
UDEF    6F7A 0    DL2
UDEF    6F84 0    DL3
UDEF    6F8E 0    DL4
UDEF    6F98 0    DL5
UDEF    6FFE 0    SNAPTBL
UDEF    7056 0    OLDPSD
UDEF    76AB 0    HIGHEST LOC
DEF     77D0 0    M:DO
DEF     77A0 0    M:OC
DEF     7770 0    M:LO
DEF     7740 0    M:EI
```

Figure 2-4. Sample LOPE Map Printout

5. Diagnostics.

a. General Diagnostics.

(1) *** WARNING *** name NOT A USABLE DCB. The DCB named cannot be used as built because LOPE must build all DCBs.

(2) ERROR IN ABOVE DCB DESCRIPTION. In on-line mode, an error exists in F: input from the user.

(3) ILLEGAL LOAD OPTION. In on-line mode, an option specified by the user is invalid.

(4) ILLEGAL LOPE OPTION. In batch mode, the LOPE control command submitted by the user contains an error.

(5) TOO MANY DCBs. The number of DCBs referenced by the user exceeds the number of DCBs that can be fit into two pages (1024 words) of core storage.

(6) TOO MANY EFS. The number of element files specified by the user exceeds the number of element files that can be fit into an internal element file list built by the loader.

b. Load Diagnostics. Each load diagnostic is followed by an explanatory message from section c to indicate at what point in the load process the error occurred.

(1) CHECKSUM ERROR. A bit (or bits) was dropped when the specified record was punched or read.

(2) CONFLICTING DSECT. All declared COMMON areas must be the same size. (common) SIZE

(3) ILLEGAL ORIGIN. The ROM data contains an illegal ORG directive that caused an attempt to load outside the available area.

(4) ILLEGAL ROM DATA. The Relocatable Object Module being loaded contains illegal object language.

(5) NO ELEMENT FILE. The specified element file cannot be found.

(6) NO LIB FILE. The specified library cannot be found.

(7) SEQUENCE ERROR. The sequence of the record following the specified record is not equal to the current sequence plus one and the current record is not the last record in a ROM or the first record in a ROM is missing.

(8) STACK OVERFLOW. There is not sufficient room in core for the loader, the program, and the loader stacks.

c. Explanatory Messages. One of these messages follows each of the load diagnostics in section b to indicate at what point in the load process the error occurred. In the messages XX and YY are hexadecimal numbers.

(1) LOADING FROM BI
SEQ NO XX
OVERALL ROM NUMBER YY

The loader was processing ROMs through M:BI when an error occurred in the XXth record of the YYth ROM ecountered in the entire load process.

(2) LOADING ELEMENT FILE OR GO — The loader was processing
EF – name SEQ NO XX — ROMs from the file "name"
OVERALL ROM NUMBER YY — when an error occurred in the XX th record of the YYth ROM encountered in the entire load process.

(3) PROCESSING LIBRARY — The loader had processed
account SEQ NO XX — all ROMs in the specified
OVERALL ROM NUMBER YY — element file and was processing in the account listed when an error occurred in record XX of the YYth ROM encountered in the entire load operation.

d. On-line Diagnostics for Executing without the Debug Option:

(1) DECIMAL FAULT
(2) FIXED OVERFLOW
(3) FLOATING FAULT
(4) ILLEGAL ARGUMENTS TO CALL
(5) ILLEGAL CALL
(6) MEMORY PROTECTION VIOLATION
(7) NONEXISTENT ADDRESS
(8) NONEXISTENT INSTRUCTION
(9) PRIVILEGED INSTRUCTION
(10) PROCEED
(11) UNIMPLEMENTED INSTRUCTION
(12) USER EXIT
(13) STACK VIOLATION

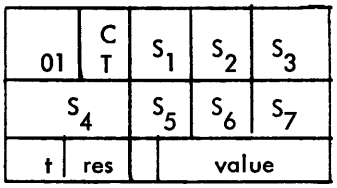
6. Severity Level. A non-zero severity level is printed at the end of the load process. The severity level is the maximum of any severity levels inherited from the ROMs and of any severity levels generated by the loader. After printing the severity level, the loader compares the value with the maximum severity level specified by the user and aborts processing if the value exceeds the maximum.

Loader-Generated Severity Levels

Type of Error	Severity
PREF	7
DDEF	4
REF load table exceeded	F

7. Internal Symbol Table. In on-line mode if the debug option is specified, the Internal Symbol Table is created for the debug program, DELTA. See figure 2-5 for the format. Only the first seven characters of any symbol are used. If symbols have duplicate initial seven characters and length only the first symbol encountered is retained.

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:

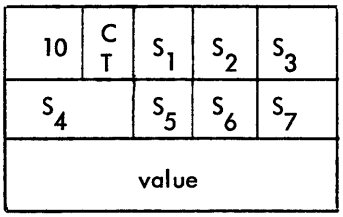
00000 - instruction
00001 - integer
00111 - EBCDIC text (also for unpacked decimal)
00010 - short floating point
00011 - long floating point
00110 - hexadecimal (also for packed decimal)
01001 - integer array
01010 - short floating point array
01011 - long floating complex array
01000 - logical array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

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

000 - byte
001 - halfword
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.

Figure 2-5. Internal Symbol Table

2.2.5 Loader-Generated Tables

See Figure 2-6 for memory allocation, including the position of the tables, at execution time.

1. Task Control Block. See Figure 2-7 for the format. The TCB resides in the 00 area.
2. DCB Name Table. See Figure 2-8 for the format. The DCB Name Table resides in the 01 area.
3. Tree. A copy of the tree table is placed at the beginning of the 01 area as well as being recorded in the tree record. See figure 2-1 for the format.
4. DCBs.
   a. The loader builds a DCB for any F: or M: primary reference. In batch mode, LOPE always builds on M:DO DCB.
   b. All loader-generated DCBs are 48 words long. The first 22 words (0-21) are standard and allocated for the fixed portion of the DCB. Each variable-length parameter (words 22-40) is preceded by a control word and allocated to allow, in addition to the control word, three words for file name, two words for account, two words for password, three words for INSN (SN), and three words for OUTSN (SN). Words 41-48 are reserved. Default information is placed into recognized DCB names. See Figure 2-9 for the recognized DCBs and defaults.
   c. If a file is opened in the output mode through any of the M:DCBs in Figure 2.8 except M:C, a flag is set in the Job Information Table (JIT). If the DCB is not reassigned before the DCB is opened again in output mode during the same job, all records output through the DCB are appended to the end of the file -- file extension occurs.

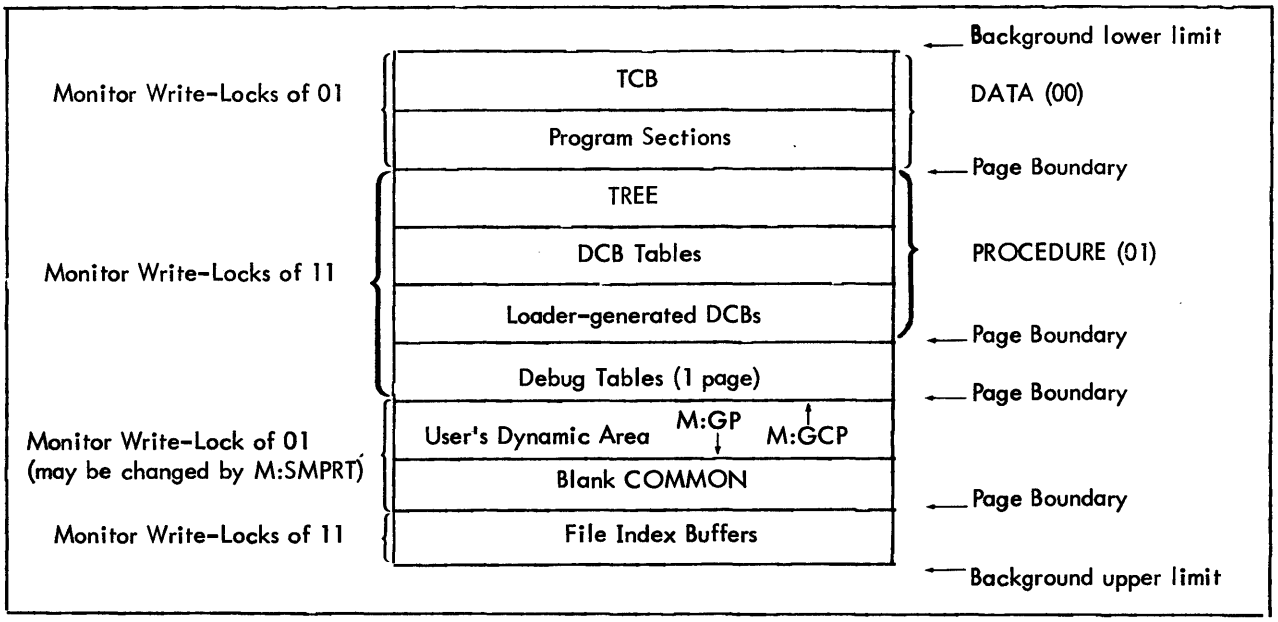


Figure 2-6. Background Memory Layout, LOPE Processor

The fields of the TCB are as follows:

TSTACK is the address of the current top of the user's temp stack.

TSS indicates the size, in words, of the user's temp stack.

TSA is the address of the temp stack used by the library error package.

TSASIZ indicates the size, in words, of the temp stack used by the library error package.

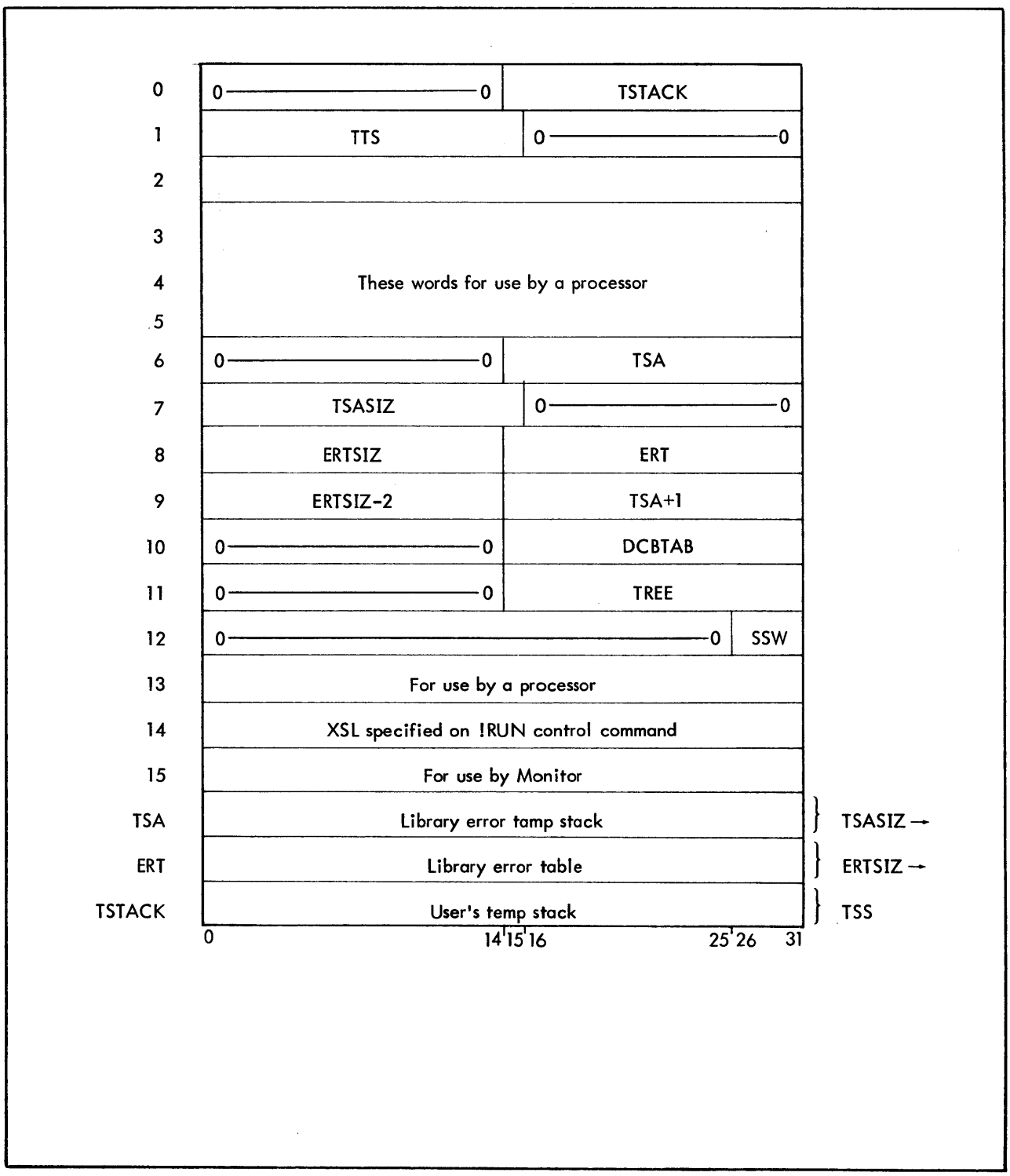


Figure 2-7. Task Control Block Format

ERSIZ indicates the size, in words, of the error table used by the library error package.

ERT is the address of the error table used by the library error package.

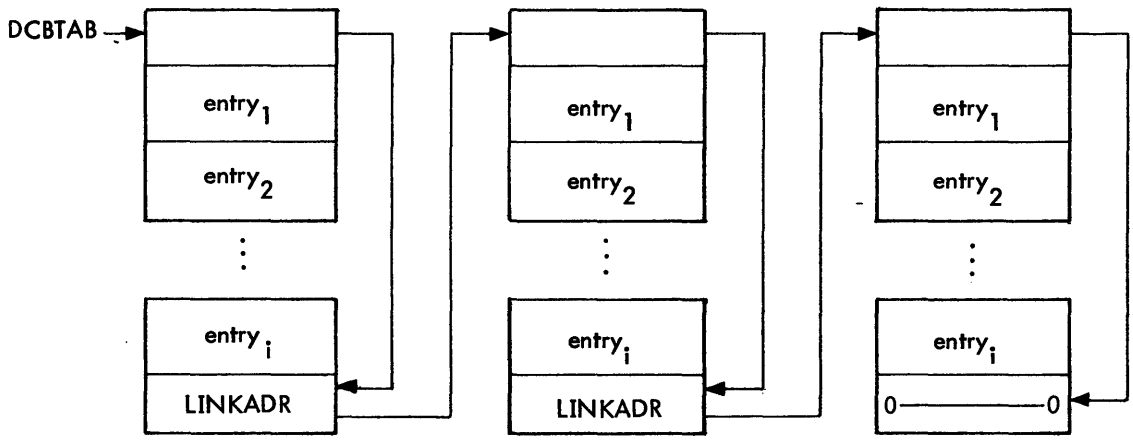
DCBTAB is the address of a table of names and addresses of all of the user's DCBs. This table has the form shown in Figure 2-8.

TREE is a pointer to the location of the user's overlay structure.

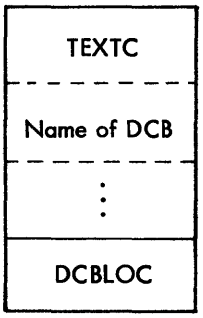
SSW contains the user's sense switch settings (bit 26 contains the setting of switch 1, etc.).

On transferring control to a user's program or to a processor, the Monitor communicates the TCB address through general register 0.

Figure 2-7. Task Control Block Format (continued)



Notes: 1. Each entry contains a variable length DCB name and DCBLOC word as follows:



2. LINKADDR is the location of another block of the DCB table. If LINKADDR contains zero, the current block is the last one of the DCB table.

3. DCBLOC is the address of the first word of the DCB.

Figure 2-8. DCBTAB (Name Table)

DCB Name	Function	Record Byte Size	Operational Label
M:C	Input	120	C
M:OC	Input/output	85	OC
M:LO	Output	132	LO
M:LL	Output	132	LL
M:DO	Output	132	DO
M:PO	Output	80	PO
M:BO	Output	120	BO
M:LI	Input	120	LI
M:SI	Input	80	SI
M:BI	Input	120	BI
M:SL	Output	132	SL
M:SO	Output	80	SO
M:CI	Input	120	CI
M:CO	Output	120	CO
M:AL	Output	80	AL
M:EI	Input	120	EI
M:EO	Output	120	EO
M:GO	Output	120	NO
F:101	Input	0	OC
F102	Output	0	OC
F:103	Input	0	PR
F:104	Output	0	PP
F:105	Input	80	SI
F:106	Output	120	BO
F:108	Output	132	LO

Figure 2-9. Loader-Constructed System DCBs

2.2.6 LOAD Subsystem

1. The LOAD subsystem is a two-level subsystem. Both levels use the same memory area so both do not run concurrently. For each terminal in the system an area of secondary swap storage is allocated for each level. The LOAD subsystem is maintained in subsystem storage as the LOAD subsystem is time-sliced into memory until the job step is complete. After the program is loaded by the LOAD subsystem, an image of the terminal user's program is maintained in user storage. After the LOAD subsystem transfers execution to the user program, the LOAD subsystem maintains effective control by monitoring all abnormal conditions such as a trap. The BTM Executive keeps track of the level applicable to each job. The Executive always interrupts a user program and gives execution to the LOAD subsystem if an abnormal condition occurs.

2. The LOAD subsystem follows the rules for a BTM subsystem.

   a. The first 40_{16} locations contain specified data. See Figure 2-10 for the format.

   b. The Load Subsystem has a Task Control Block (TCB). See Figure 2-7 for the format.

      (1) The TCB specifies no temp stack for the LOAD subsystem.

         (a) The address of the current top of the stack is STACK, a label that reserves the number of words specified by the value SZSTACK, zero.

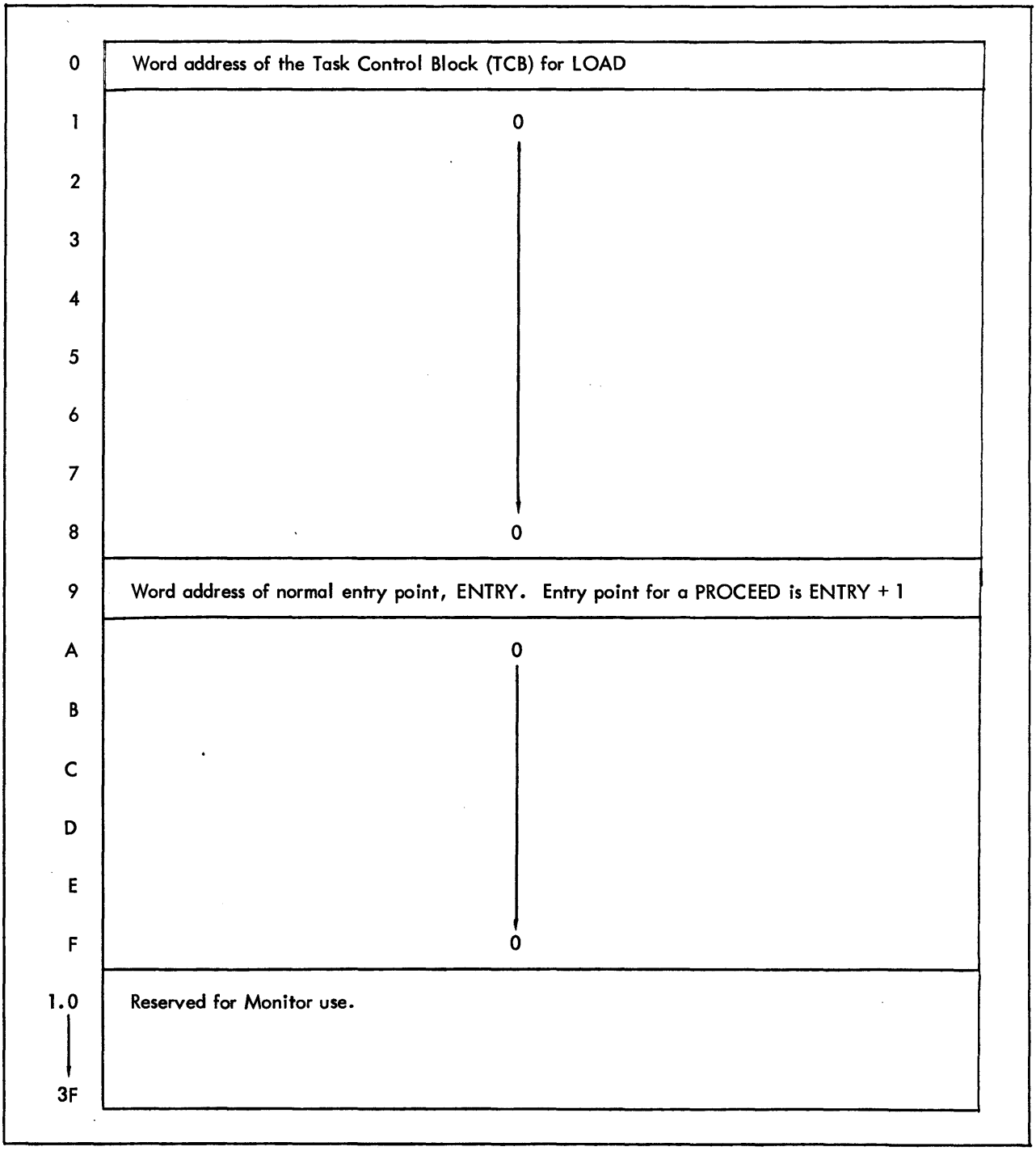


Figure 2-10. Standard First X'40' Words of LOAD Subsystem

(b) The stack size, in words, is the value SZSTACK, zero.

(c) The stack word count is zero.

(2) The TCB specifies no temp stack for the library error package.

(a) The address of the stack is zero.

(b) The stack size, in words, is zero.

(c) The stack word count is zero.

(3) The TCB specifies no error table for the libarary error package.

(a) The table size, in words, is zero.

(b) The table address is zero.

(4) The TCB has the address of the DCB Name Table, MYDCBTAB.

c. The LOAD subsystem has a DCB Name Table with entries for all DCB used. See Figure 2-8 for the format.

(1) The first link address points to the last word of the table. The last word of the table, also a link address, is zero to indicate the end of the table.

(2) Entries are given for M:BI, M:LI, M:DO, M:LO, and F:X1.

d. The LOAD subsystem has all DCBs used by the subsystem assembled with protection type 00.

e. Upon entry, the LOAD subsystem has access to data.

(1) Register 1 contains the COC line number in binary.

(2) Registers 4 and 5 contain the EBCDIC login account designation, left justified and blank filled.

(3) Register 2 contains the terminal job entry flag. 0 means that the console is excluded from the system and 1-F indicates the maximum priority.

(4) Register 3, byte 0, contains the batch authorization flags from the BTM job information table, AJIT.

(5) Registers 13-15 contain the EBCDIC login name designation, left-justified and blank filled.

f. The LOAD subsystem uses BTM system calls to perform system control and memory management functions.

(1) CAL3,14 returns the maximum number of pages that can be allocated. This value is used in allocating the dynamic data area.

(2) CAL3,11 informs the BTM Executive of the new configuration whenever the amount of space used in the dynamic data area changes.

(3) CAL3,4 loads the debug program, DELTA, and transfers control to DELTA.

(4) CAL3,7 swaps a page between the user area and the subsystem area to allow LOPE to monitor execution of the user program (to determine why control was returned).

(5) CAL3,6 returns control to the BTM Executive.

2.2.7 Debug Program, DELTA

The LOAD subsystem issues a call to load DELTA and to transfer control to DELTA if the user has specified execution under the debug option. DELTA accesses data indirectly through the first word of the last page of memory which LOPE has set to point to the global symbol table location and size, user program starting address, internal symbol table and size, and last word of user program (following DCBs). See Figure 2-5 for the format of the Internal Symbol Table and section 2.2.4,7 for the description of the table. See the DELTA documentation for a diagram of memory following DELTA initilization after an on-line load.

2.2.8 Accounting

The loader is charged until the loader gives control to the user program. The instruction issued just before control goes to the user program causes the user to be charged.

2.2.9 Monitor

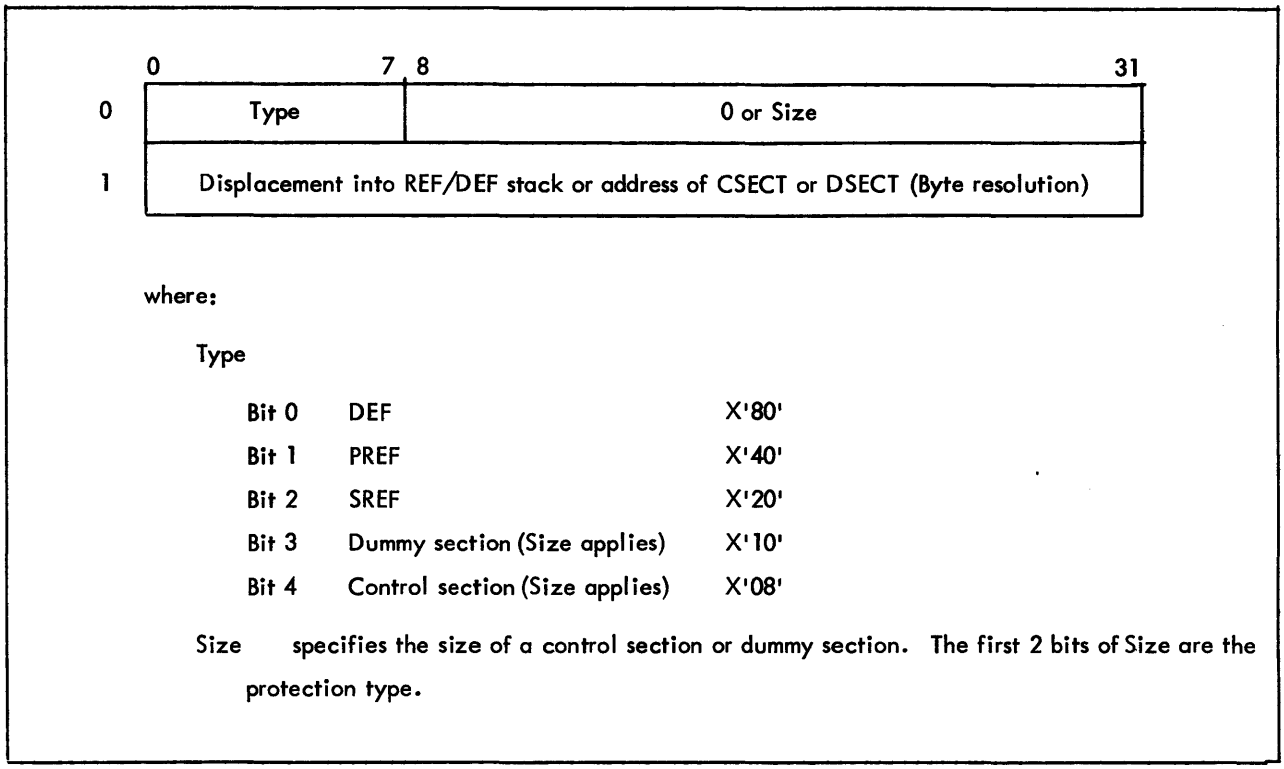
1. The loader access the job information table, JIT in batch mode, or AJIT, in on-line mode, to obtain information about the current job. For example, in on-line mode, the loader obtains the log-in account from AJIT if a password with no account is specified in a DCB assignment.
2. The loader uses a CAL1,9 9 to have accounting properly allocated. To change accounting from processor to user, the loader issues the call immediately before transferring control to the user program.

2.2.10 CCI

CCI builds the assign record which the loader reads. See section 2.2.3,3 for the description.

2.2.11 Stacks

1. To translate object modules into an executable form, the loader processes load items, which constitute the object language. To obtain all information about control sections, dummy sections, external definitions, external references, and forward references, the loader combines data from different types of load items by retaining data in stacks. The loader forms five stacks: the declaration stack; the REF/DEF stack; the expression stack, the forward reference stack, and the temporary storage stack. Note that LOPE has different stacks and a different format for the stacks than the overlay loader.
2. Declaration Stack. See Figure 2-11.
   a. When processing a declare load item, the loader builds an entry for the item in the declaration stack. The ROM has assigned declaration numbers, created declare load items, to all control sections, dummy sections, external definitions, and external references. At the end of the module, the loader clears the stack since the declaration numbers apply only within a module.



where:

Type

Bit 0	DEF	X'80'
Bit 1	PREF	X'40'
Bit 2	SREF	X'20'
Bit 3	Dummy section (Size applies)	X'10'
Bit 4	Control section (Size applies)	X'08'

Size specifies the size of a control section or dummy section. The first 2 bits of Size are the protection type.

Figure 2-11. Declaration Stack

b. Given a declaration number, the loader can locate the corresponding stack entry by position in the stack since the ROM has assigned declaration number consecutively. The loader must be able to locate the item by declaration number in order to evaluate expression load items, which refer to declared load items by number.

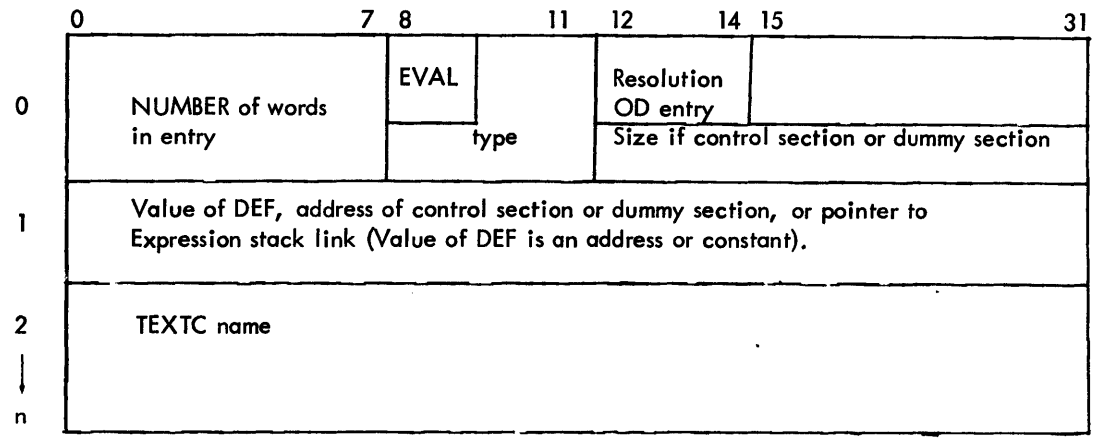
c. The two word stack entry contains the type and a pointer to the corresponding entry in the REF/DEF stack that is being built to contain information about the item. For a control section, after the base address and size are available, these values are added to the declaration stack entry, with base address replacing the pointer to the REF/DEF stack.

3. REF/DEF Stack. See Figure 2-12.

a. When processing a declare load item, the loader searches the REF/DEF stack for a corresponding entry and builds an entry if the entry does not exist. In this way the loader creates an entry for each control section, dummy section, external definition, and external reference.

b. The REF/DEF stack contains information about an entry. In batch mode, LOPE converts the REF/DEF stack format to the REF/DEF stack format of the overlay loader and outputs the stack as part of the load module so that symbolic debugs may be used during execution.

c. The variable-length entry contains the number of words in the entry, type, resolution, value of DEF or a pointer to the expression stack link, and the name of the item in TEXTC format.



where:

type

Bit 0	Evaluated flag	1 = yes, 0 = no or DEF X'80'
Bit 1	External reference	SREF X'40'
Bit 2	PREF	PREF X'60'
Bit 3	Dummy section	X'10'
		DDEF X'90'

More than one indicator may be on. For example a DEF that satisfies a primary reference is a X'EO'.

Figure 2-12. REF/DEF Stack

4. Expression Stack. See Figure 2-13.

   a. To process a define load item, the loader obtains the expression control bytes following the item (which refer to constants, declaration numbers, and forward reference numbers), builds an expression (using the declaration stack if necessary), places the expression in the expression stack entry for the item, and attempts to evaluate the expression. If the loader cannot evaluate the expression, the loader chains the entry to an entry in the REF/DEF stack if the expression needs the value of an external reference or to an entry in the forward reference stack if the entry needs the value of a forward reference.

   b. If the loader can evaluate the expression, the loader places the value of the expression in the destination, a REF/DEF stack entry if the expression defines an external definition, a forward reference stack entry if the expression defines a forward reference, or the core image if the expression is a core expression. Note that the expression stack is not part of the load module since the stack is only needed to load a load module with a ROM, a feature absent in LOPE.

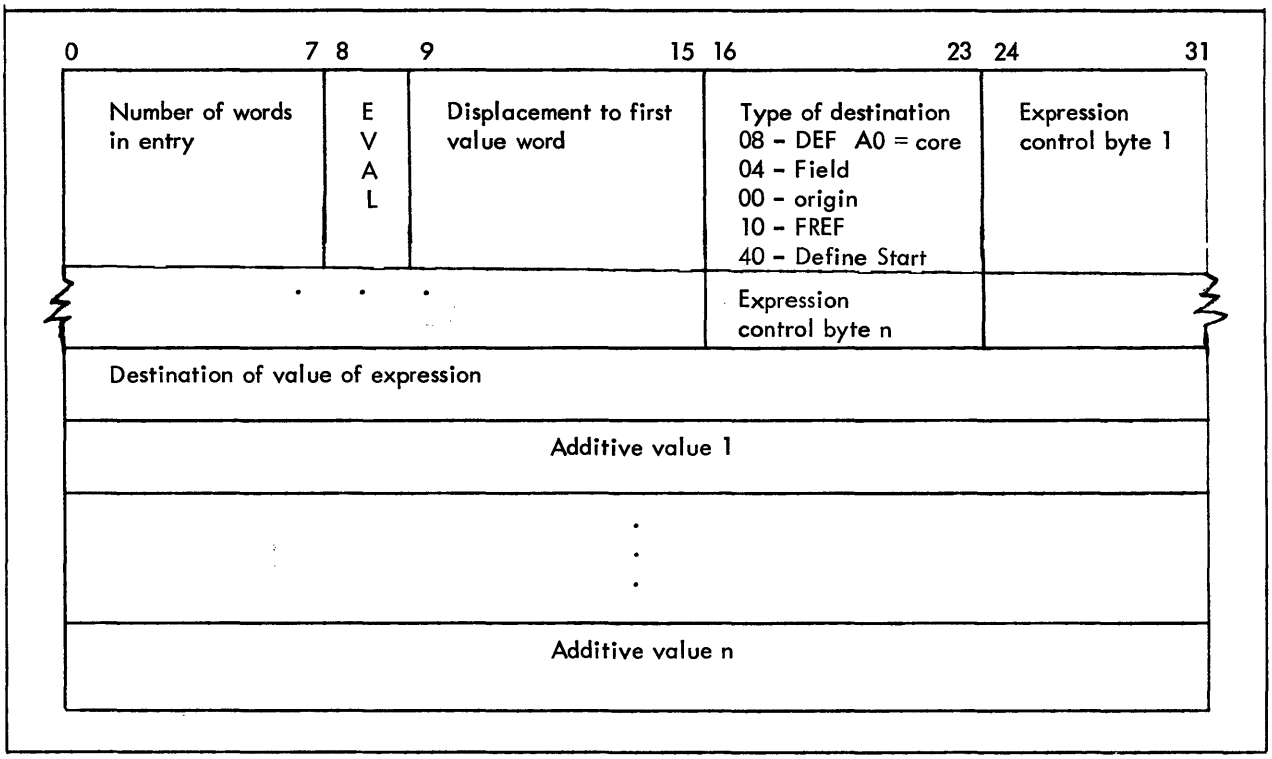


Figure 2-13. Expression Stack

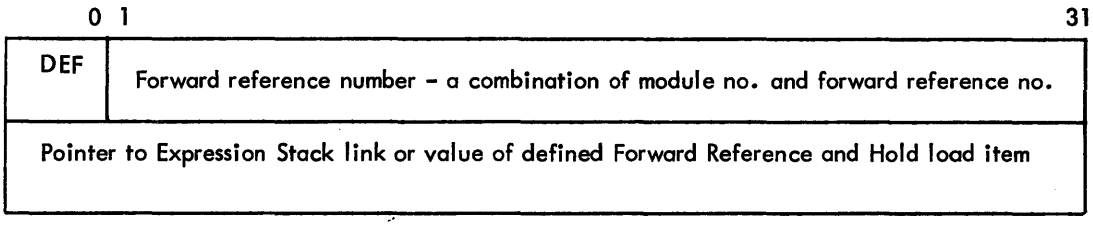
c. The variable length entry contains the number of words in the entry, the evaluated flag, the displacement to the first value word, the type of destination, the expression control bytes, and the destination of the value of the expression.

5. Forward Reference Stack. See Figure 2-14.

a. In processing expressions, the loader finds expressions involving forward references. The expressions refer to forward references by random numbers. The loader searches the forward reference stack for a corresponding entry and builds an entry if the entry does not exist. If the load item is a DFREF, define forward reference, the loader removes the entry from the stack as soon as the forward reference is defined because the forward reference number is closed when defined. A new expression involving the number refers to a new forward reference. If the load item is a DFREFH, define forward reference and hold, the loader keeps the entry in the stack until module end. At module end, the loader clears the stack even if the forward reference has not been resolved. The defining expression refers to a primary reference or a dummy section.

b. The loader uses the forward reference stack to keep track of the forward reference number until the entry is defined. When the entry is defined, the value is placed in the expression stack entry that needs the value.

c. The two word entry contains the forward reference number and a pointer to the expression stack link. If the entry is a forward reference and hold, the entry contains a defined flag. When the flag is set, the value of the forward reference replaces the pointer to the expression stack.



where:

DEF is 1 if a forward reference and hold load item is defined.

Figure 2-14. Forward Reference Stack

6. The temporary storage stack contains temporary entries. A minimum fixed size is kept to allow for non-recursive use without checking for stack overflow.

7. The stacks are allocated at the top of the common dynamic area and grow downward. See Figure 2-15. The sum of the initial sizes is the value SUMSIZES, 3840 bytes. The size of the stacks vary depending on the usage. The stacks expand as entries are added. Since the entries are referenced indirectly through the stack pointer, the actual position of the stack in core may vary. The stacks may be moved around to make more stack space available from the dynamic data area. The stacks may also be compressed to redistribute unused space in one stack to another stack. The subroutine PRESSTK collapses the stack space if possible and the code at STKOVF expands the appropriate stack if an overflow occurs. Code using the stacks checks for overflow after pushes and call STKOVF with the amount of space needed as an argument. A certain amount of space is left in the temporary stack for non-recursive use.

8. See Figure 2-15 and 2-16 for the labels associated with the stacks and the position of the stacks in core.

2.2.12 Use of Memory

1. In batch mode, LOPE issues a get-page call to request the maximum number of pages available. In on-line mode, the FOAD subsystem issues a CAL3,4 to find out the maximum number of pages that can be activated. The loader begins at the low end of the dynamic data area when processing the library option or building the core image and allows the library buffers or core image to extend upwards. The loader begins at the high end of the dynamic data area for the stacks and allows the stacks to extend downward. See Figure 2-16. Whenever the amount of dynamic data area used increases, the loader issues a CAL3,11 to inform the BTM Executive of the new configuration so that swapping will be correct and efficient. If the library buffer or the core image overlaps the temp stack and space cannot be removed from the stacks, processing is aborted for lack of core.

2. When the execute option is specified, the loader uses the set memory protect call to change the protection type of the 01 area to 00 so that the loader can store into this area when creating the core image of the user program.

Stack	Stack pointer Doubleword	Base	Initial Size (Bytes)
Declaration	DECLSTK	DECLBAS	DECLSIZ = 128
REF/DEF	RFDFSTK	RFDFBAS	RFDFSIZ = 1024
Expression	EXPRSTK	EXPRBAS	EXPRSIZ = 2048
Forward reference	FREFSTK	FREFBAS	FREFSIZ = 312
Temporary storage	TSTACK	TEMPBAS	TMPSIZ = 128
			SUMSIZES = 3840

Library Processing

BLKBUF contains the address of a one page buffer used to hold the physical library record.

OBUF contains the address of the buffer for the logical library record.

DYNDATA contains the address of a one page buffer used to store declaration numbers of external definitions in halfword entries.

The page after the address in DYNDATA is a buffer in which declare reference and define origin load items are stored until declare external definition load items have been placed in the library record.

Memory Allocation

LDDATA+1 is the lower limit of the program.

LDDATA+2 is a word containing the upper limit of the dynamic data area.

MAXLOC is the address of the highest location loaded.

BACORE is the byte address of the next section to be allocated.

LOC is the current value of the location counter.

On-line Mode

USERSIZE is the number of available pages.

00SIZE is the number of program data pages in the LOAD subsystem.

01SIZE is the number of program procedure pages in the LOAD subsystem.

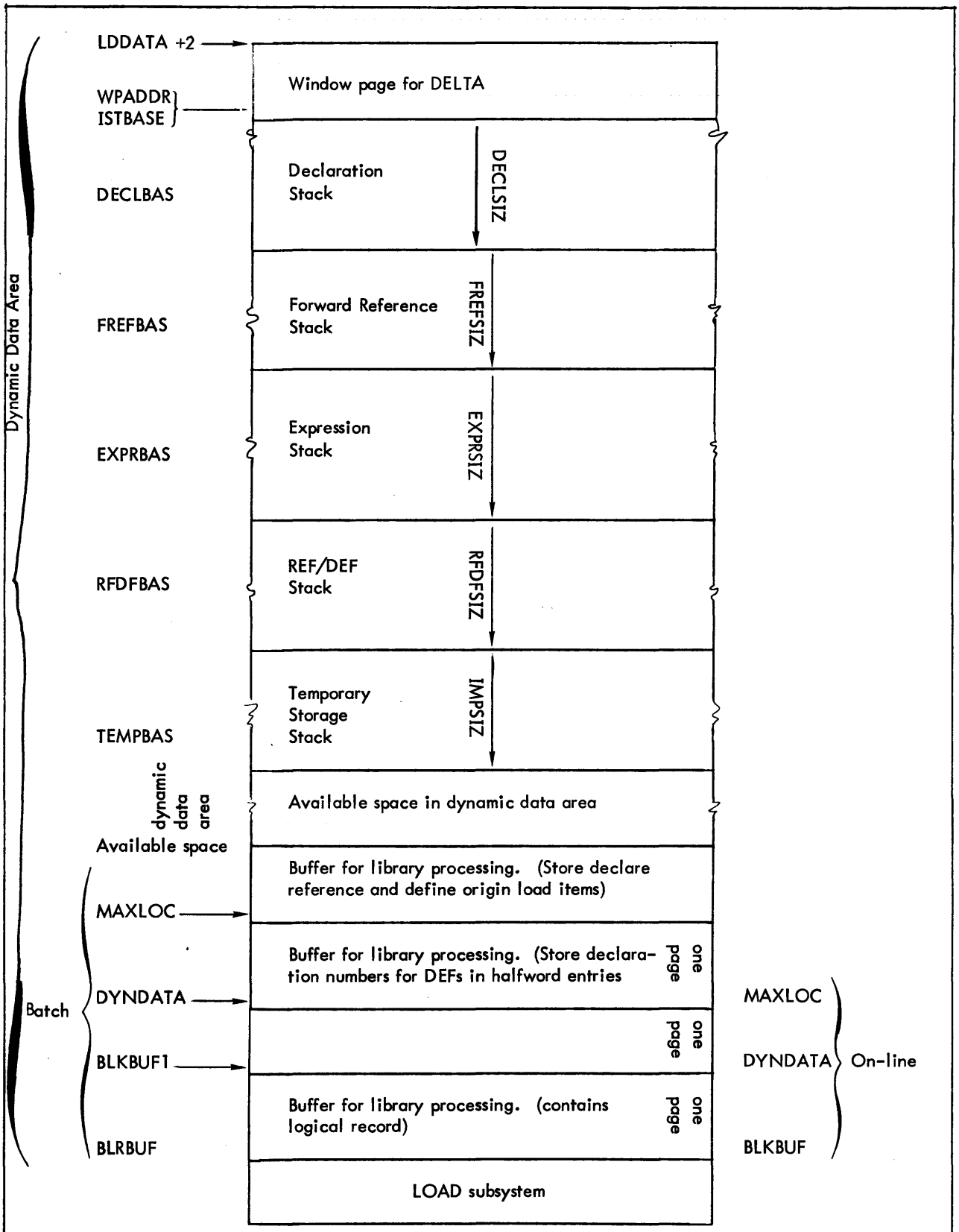
On-line Mode-Debug Option

WPADDR is the address of the window page.

WPNUM is the page number of the window page.

ISTBASE is the address of the Internal Symbol Table.

Figure 2-15. Labels used in Allocating Memory



As the core image is built upward in the dynamic data area, MAXLOC increases. LOC, the current value of the load location counter is checked to make sure LOC remains below TEMPBAS. If space cannot be made available, processing is aborted.

Figure 2-16. Layout of Memory after Initialization

2.2.13 Example

1. Sample Program

A sample program was assembled under the METASYM processor.

```
 1                                                  SYSTEM    SIG7FDP
 2                                                  DEF       AB1
 3                                                  REF       AB2
 4   01 00000    6A900000 X   START                 BAL,9     AB2
 5   01 00001    00000008 02                        DATA      ZAP+2
 6   02 00000                                       CSECT     0
 7   02 00000                                       RES       5
 8   02 00005    000000FF A   AB1                   DATA      X'FF'
 9         02 00006           ZAP                   EQU       $
10         01 00000                                 END       START

CONTROL SECTION SUMMARY: 01 00002   PT 0     02 00006   PT 0
```

2. ROM

A load-item-by-load-item interpretation (a ROMBUST) of the ROM for the sample program was made. The interpretation of the load items is in the order in which the items were output by the METASYM processors. Note that each load item is listed, in hexadecimal, on the line above the description.

```
ROMBUST of Sample Program
RECORD ---      RECORD NUMBER: 0


  RECORD TYPE: LAST, MODE: BINARY, FORMAT: OBJECT LANGUAGE.
  SEQUENCE NUMBER 0
  CHECKSUM: 200
  RECORD SIZE: 66


  0303C1C2F1
  DECLARE EXTERNAL DEFINITION NAME (3 BYTES) NAME: AB1    DECLARATION NUMBER: 1


  0503C1C2F2
  DECLARE PRIMARY REFERENCE NAME (3 BYTES) NAME: AB2    DECLARATION NUMBER: 2


  0C000008
  DECLARE NONSTANDARD CONTROL SECTION     DECLARATION NUMBER: 3
  ACCESS CODE: FULL ACCESS. SIZE 8    X'8'


  0C000018
  DECLARE NONSTANDARD CONTROL SECTION     DECLARATION NUMBER: 4
  ACCESS CODE: FULL ACCESS. SIZE 24    X'18'



0A0101000000142004O2
DEFINE EXTERNAL DEFINITION
NUMBER 1
ADD CONSTANT: 20    X'14'
ADD VALUE OF DECLARATION (BYTE RESOLUTION)
NUMBER 4
EXPRESSION END
```

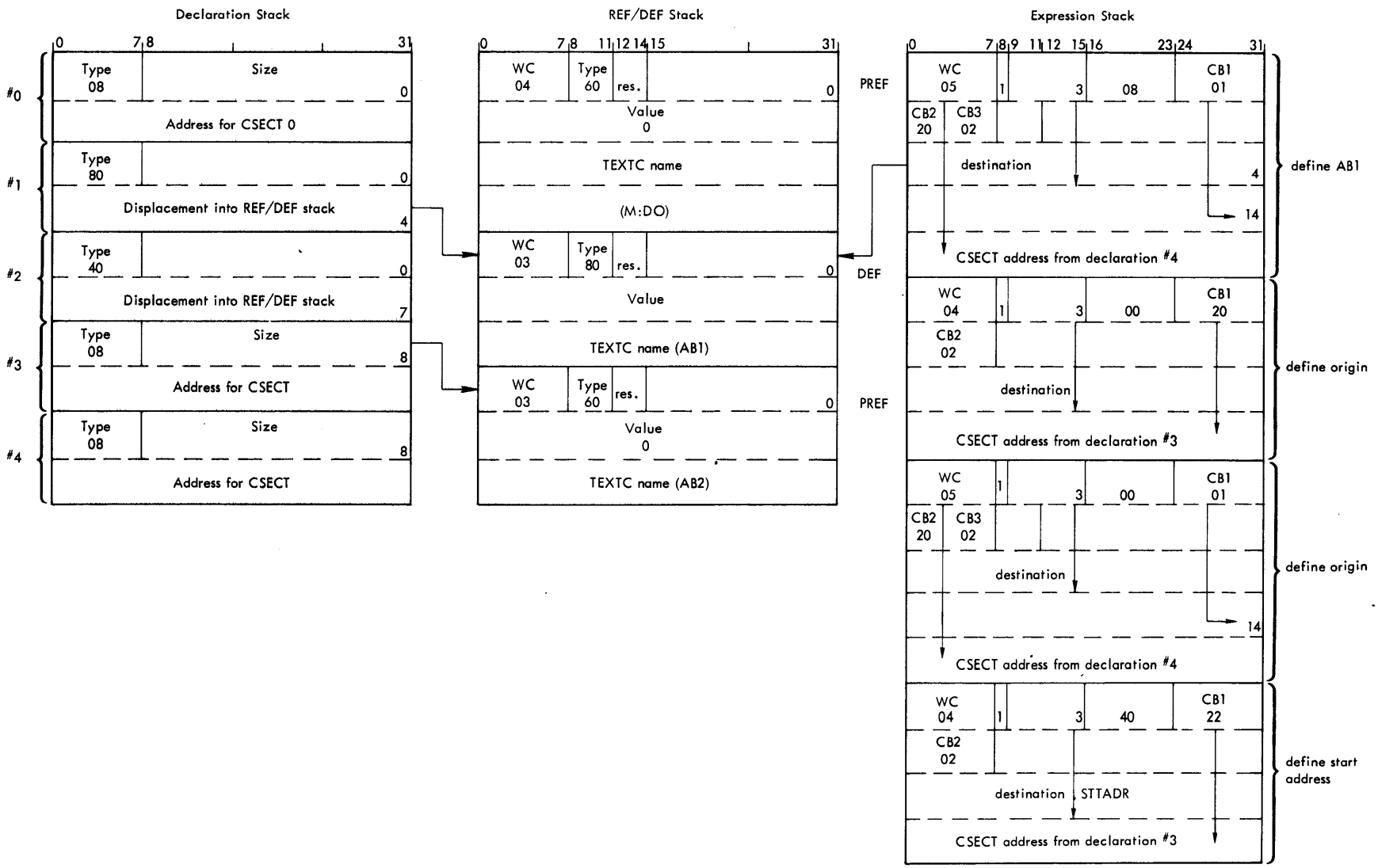
```
04200302
ORIGIN
ADD VALUE OF DECLARATION (BYTE RESOLUTION)
NUMBER 3
EXPRESSION END


826A900000
LOAD RELOCATABLE (SHORT FORM), RELOCATE ADDRESS FIELD (WORD RESOLUTION)
RELATIVE TO DECLARATION NUMBER 2
THE FOLLOWING 4 BYTES: X'6A900000'


8400000008
LOAD RELOCATABLE (SHORT FORM), RELOCATE ADDRESS FIELD (WORD RESOLUTION)
RELATIVE TO DECLARATION NUMBER 4
THE FOLLOWING 4 BYTES: X'8'


0401000000014200402
ORIGIN
ADD CONSTANT: 20    X'14'

ADD VALUE OF DECLARATION (BYTE RESOLUTION)
NUMBER 4
EXPRESSION END


44000000FF
LOAD ABSOLUTE THE FOLLOWING 4 BYTES: X'000000FF'


0D220302
DEFINE START
ADD VALUE OF DECLARATION (WORD RESOLUTION)
NUMBER 3
EXPRESSION END


0E00
MODULE END
```

3. Load Module

   Using LOPE to load the sample ROM, a load module called TARGET was created.

Target Load Module

```
HEAD
00000    80000FF18  80006664  33003300  33003400  36003600  0019000C

07E3C1D9C7C5E300
00000    05000000  00C1AF40  01000000  04D47AC4  D6404040  04000000  000199AC  01000000
00008    03C1C2F1  04020000  00007A64  01C00000  03C1C2F2  06000000  0001AFFC  01000000
00010    CBC8C9C7  C8C5E2F3  40D3D6C3  06000000  00019800  01000000  CAD3D6E6  C5E2E340
00018    D3D6C340

07E3C1D9C7C5E303
00000    00C06623  00400000  00000000  00000000  00000000  00000000  0000660F  000A0000
00008    0014661A  00106610  0000680E  00000000  00000000  00000000  00000000  00000000
00010  THRU  005F
         00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000
00060    00000000  00000000  00000000  00000000  6A900000  0000666E  00000000  00000000
00068    00000000  00000000  00000000  000000FF  00000000  00000000  00000000  00000000
00070  THRU  01FF
         00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000

07E3C1D9C7C5E305
00000    00000000  00000000  0000000C  07E3C1D9  C7C5E305  40404040  00000000  00000000
00008    01003300  00030000  02003400  00000000  00003600  00000000  00006812  04D47AC4
00010    D6404040  00006BD0  00000000  00000000  00000000  00000000  00000000  00000000
00018  THRU  03CF
         00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000
03D0     30000003  00080005  0A000000  00000000  00000000  80000011  00006BE6  00000000
03D8     00000000  00000000  00006BF8  00000000  00000000  00000000  00000000  00000000
03E0     00000000  00000000  00000000  00000000  00000000  00000000  01000003  00000000

03E8     00000000  00000000  02000002  00000000  00000000  03000002  00000000  00000000
03F0     07000003  00000000  00000000  00000000  08010003  00000000  00000000  00000000
03F8     00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000

TREE
00000    0000000C  06E3C1D9  C7C5E300  40404040  00000000  00000000  01003300  0019000C
00008    02003400  00000000  00003600  00000000
```


Declaration Stack
REF/DEF Stack
Expression Stack
#0
Type 08
Size
Address for CSECT 0
#1
Type 80
Displacement into REF/DEF stack
#2
Type 40
Displacement into REF/DEF stack
#3
Type 08
Size
Address for CSECT
#4
Type 08
Size
Address for CSECT
WC 04
Type 60
res.
Value 0
TEXTC name
(M:DO)
WC 03
Type 80
res.
Value
TEXTC name (AB1)
WC 03
Type 60
res.
Value 0
TEXTC name (AB2)
PREF
DEF
PREF
WC 05
CB1 01
CB2 20
CB3 02
destination
CSECT address from declaration #4
define AB1
WC 04
CB1 20
CB2 02
destination
CSECT address from declaration #3
define origin
WC 05
CB1 01
CB2 20
CB3 02
destination
CSECT address from declaration #4
define origin
WC 04
CB1 22
CB2 02
destination
STTADR
CSECT address from declaration #3
define start address

2.3 OPERATIONAL OVERVIEW

2.3.1 Description

1. Entry.

   a. The starting address for the loader in ENTRY, which branches to SIGSALS.

   b. The instruction after ENTRY is the entry point for a PROCEED command.

   c. The instruction after USERGO is the entry point for return from execution of a user program.

2. Exit.

   a. The code after COMPROM in the Library Module, PERMLIB, issues a CAL1,9 1 to exit from the job step.

   b. In batch mode, after building a load module, the code after EXPNDLOP in the Execute Module issues a CAL1,9 1 to exit from the job step.

   c. In on-line mode, after setting up to save files, the code after SMALLER in the Execute Module issues a CAL3,6 0 to return to the BTM Executive.

   d. In batch mode, if executing the program, the coce after $STTADR in the Execute Module transfers control to the user program by branching to the address in STTADR.

   e. In on-line mode, if executing the program with the debug option, the code after DELTA in the Execute module issues a CAL3,4 0 to load DELTA and to transfer control to DELTA.

   f. In on-line mode, after executing the program without the debug option, NORMRT in the Execute module issues a CAL3,6 0 to return to the BTM Executive.

   g. If the job step cannot be continued, the code after ABORTIT issues, in batch mode, a CAL1,9 3 to abort the job or, in on-line mode, a CAL3,6 0 to return to the BTM Executive.

3. In the start module at SIGSALS, do initial processing: read the control input supplied by the user and save data about the options specified. If the library option was specified by the user, go to the Library Module, PERMLIB. Otherwise go to the Loader Module, LDR. In the Library Module, create the user library, :BLIB, in the current account and issue a CAL1,9 1 to exit from the job step. In the Loader Module, LDR, load all ROMs. In the Endload Module, ENDLOAD, print a map, build the DCBs, build a DCB Name Table and a Task Control Block, TCB. If in any of these modules an error occurs, branch to one of the print error entry points, TOMNYDCB, TOOMANYEF, or PA__, and, in batch mode, abort the job or, in on-line mode, return to the BTM Executive. In the Execute Module, in batch mode, if the load module option was specified by the user, build a load module and issue a CAL1,9 1 to exit from the job step. In batch mode, if the execute option was specified by the user, transfer control to the user program. In on-line mode, if an execute option was not specified by the user, set up to save the files and issue a CAL3, 6 0 to return to the BTM Executive. In on-line mode, if the debug option was specified by the user, load and transfer control to the DELTA subsystem. In on-line mode, if the execute option was specified by the user, monitor execution of the user program and issue a CAL3,6 0 to return to the BTM Executive.

2.3.2 Flowchart

See Figure 2-17.

2.4 MODULE ANALYSIS

2.4.1 Start

1. Module Name. SIGSALS.
2. Purpose. Perform initial processing. Read the control input supplied by the user and save data on the options specified.
3. Entry.
   a. ENTRY is the initial entry point for the loader. When the LOPE processor or the LOAD subsystem is invoked, the processor/subsystem is loaded into core and control is transferred to ENTRY. ENTRY branches to SIGSALS.
   b. In batch mode, the LOPE control command must be read. In on-line mode, control input must be entered.
   c. In on-line mode, data is available to the subsystem.
      (1) Register 1 contains the COC line number in binary.
      (2) Registers 4 and 5 contain the EBCDIC login account designation, left-justified and blank filled.
      (3) Register 2 contains the terminal job entry flag. 0 means that the console is excluded from the system and 1-F indicates the maximum priority.
      (4) Register 3, byte 0, contains the batch authorization flags from the BTM job information table, AJIT.
      (5) Registers 13-15 contain the EBCDIC login name designation, left-justified and blank-filled.
4. Exit.
   a. The normal exit is to the Library Module, PERMLIB, if the library option has specified by the user or to the Loader Module, LDR, if the library option was not specified by the user.
      (1) The exit is taken after initial processing is complete.
      (2) Certain fields have been initialized. See Figure 2-16 for the layout of memory.
         (a) DCBs.
            (1) In on-line mode, the default assignment of M:BI to the temporary file BOTEMPα, where α is the COC line number in binary, has been made.
            (2) In on-line mode, if the debug option was specified, an intermediate file, the name of which is based on the line number, has been opened. The buffer for the intermediate file will be used as the window page by the debug program. DFLAG, the debug flag, has been set.

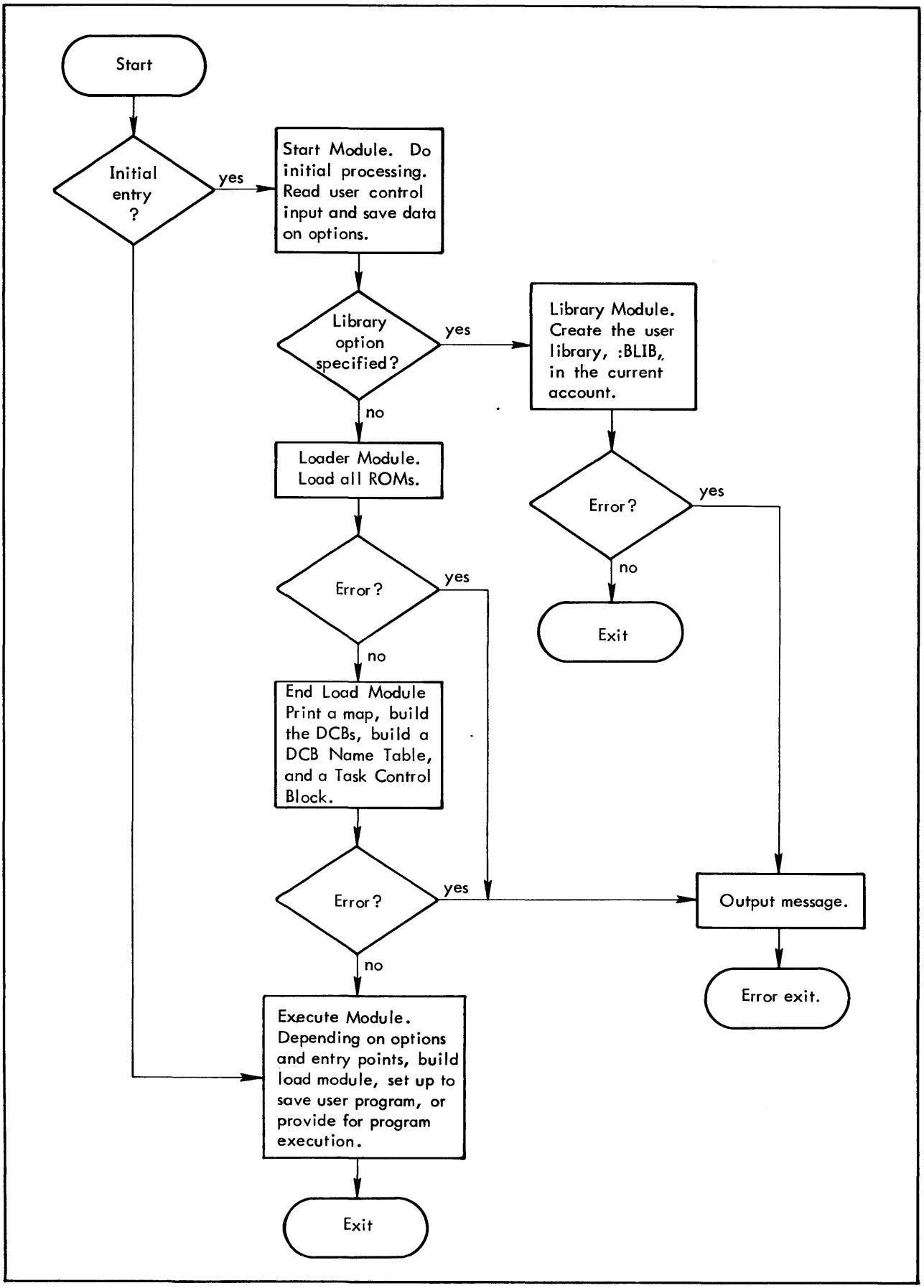


Figure 2-17. Flowchart of Overall Processing

(3) In batch mode, the job id from the Job Information Table, JIT, has been used to create the default file name for the M:LM DCB (the temporary load module file), the GO file name, and the name used to access the temporary assign file.

(4) In batch mode, if the load module option was specified by the user, the load module name has been placed in the M:LM DCB.

(b) EFLIST, the element file list, contains, in sequence, the identification of each element file specified by the user and/or, in batch mode, the GO file if specified by the user.

(c) LIBLIST, the library list, contains, in sequence, the account numbers of libraries from which primary references are to be satisfied.

(d) Stack pointers and bases have been set up. See Figure 2-16 for the names.

(e) For library processing, BCDBUF, OBUF, and DYNDATA contain the address of buffers.

(f) Register D2 has bits set for options.

Bit 0	map
Bit 1	no-system-library
Bit 2	100_{16}-control-section
Bit 5	10_{16}-control-section (batch)
Bit 6	execute (batch)
Bit 7	binary (batch)
Bit 8	permanent-load-module (batch)

(g) In batch mode, register D1 contains the converted value of the severity level option if specified by the user.

(h) In batch mode, register D1 contains the bias.

(i) In batch mode, register SR3 contains the converted value of the temporary storage option, if specified by the user.

b. The error exit is, in batch mode, to abort the job step or, in on-line mode, to return to the BTM Executive.

(1) The exit, to MNYDCB, is taken if the number of DCBs referenced exceeds the number that can be fit into two pages of core storage. The error message printed is: "TOO MANY DCBS".

(2) The exit is taken if the number of element files specified exceeds the number that can be fit into an internal element file list, EFLIST, built by the loader. The error message printed is: "TOO MANY EFS".

(3) The exit is taken if the LOPE control command or the LOAD control input contains an error. For LOPE, the error message printed is: "ILLEGAL LOPE CARD". For LOAD, the error message printed is: "ILLEGAL LOAD OPTION".

5. Operation.

   a. In on-line mode, set the default assignment for the M:BI DCB to the BOTEMPα temporary file where α is the COC line number in binary and save the binary COC line number in LINENUM.

   b. Save the lower limit in the second of the six loader control words, LDDATA+1. In batch mode, set the lower limit at TSTACK+1536. In on-line mode, set the lower limit at ORIGIN+512.

   c. Allocate memory. See Figure 2-15. Obtain all available pages. In on-line mode, use the CAL3,14 0 call to obtain the number of available pages and store the value in USERSIZE, calculate and save the number of program data pages in the LOAD subsystem in 00SIZE, calculate and save the number of program procedure pages in the LOAD subsystem in 01SIZE, and calculate the number of words available past the end of the LOAD subsystem. From this point register SR1 contains the number of words available and register SR2 contains the first word address of the available space. From the lower end of the dynamic data area, reserve a page for the buffer used to write the library record by storing the address of the first available word in BLKBUF. In batch mode, store the address of the page after the address in BLKBUF, from register SR2, in BLKBUF1 and increment register SR2 to the next page. Store the address of the next available page in DYNDATA. In on-line mode, add X'1FF' to the address in DYNDATA; store the result in MAXLOC the address of the maximum location loaded; building down from the top of the dynamic data area, reserve room for the five stacks, the value of SUMSIZES; store the address of the temporary storage stack, the lowest stack, in TEMPBAS; and describe memory to the BTM Executive for efficient swapping (SETSIZE subroutine). Clear the DYNDATA area up to the end of the available space. Store the last word address of available memory in the third of the six loader control words, upper limit, LDDATA+2.

   d. In batch mode, save the job id from the Job Information Table, JIT, in the parameter list used to open the temporary load module file, in the word used to build the GO file name, and in the parameter list used to open the temporary assign file.

   e. Set up the stack pointers and bases in the high end of the dynamic data area. Put the last-word-address-of-available-memory+1 into the word after the last base address.

   f. Obtain the control input from the user. In batch mode, print the control command through the M:LO DCB and read the control command through the M:C DCB. In on-line mode, save registers 0-3, set the prompt character to activate on carriage return, issue a new line (CRLF subroutine), and request element file names (WTTY subroutine).

   g. Obtain a byte from the control command (GCBYTE subroutine). In on-line mode, if processing element files, go to step i. In batch mode, if the control command is continued to the next record, read the next record from the C device, print the record through the M:LO DCB, and go to the beginning of step g.

   h. Obtain the keyword from the control input (NAMFLD subroutine) and use a jump table to determine where to branch to process the options. When all options have been processed, to to step t.

i. To process the element file option, obtain the element file name (NAMFLD subroutine, and GCBYTE subroutine in batch mode) and store the name in an element file list, EFLIST. If the list becomes full, print an error message (PRTERR subroutine if on-line) and, in batch mode, abort the job step or, in on-line mode, return to the BTM Executive. Obtain and save account and password, if specified, using the GCBYTE and the NAMFLD subroutines. Repeat step i until all element files have been processed. In on-line mode, request LOAD options (WTTY subroutine). Return to step g.

j. To process the map option, set bit 0 in register D2. Return to step g.

k. To process the no-system-library option, set bit 1 in register D2. Return to step g.

l. To process the 100_{16}-control-section option, set bit 2 in register D2. Return to step g.

m. To process the 10_{16}-control-section option, set bit 5 in register D2. Return to step g.

n. To process the unsatisfied reference option, obtain and save the account numbers specified by the user in the library list, LIBLIST, using the GCBYTE and the NAMFLD subroutines. Return to step g.

o. To process the library-module option, go to the Library Module, PERMLIB.

p. In on-line mode, to process the debug option, set up the intermediate file name based on the COC line number and open the file. Set the debug flag, DFLAG. Move the stack pointers to make room for the intermediate file buffer. The buffer space will be used as the window page by the debug program, DELTA. The last page in the dynamic data area, top address less 511, is the address of the buffer. Store the address in ISTBASE and WPADDR. Store the page number in WPNUM. Inform the BTM Executive of the new swap size (SETSIZE subroutine) and return to step g.

q. In batch mode, to process the GO option, build the name of the GO file using the job id saved from the Job Information Table, JIT, and store the name in the element file list, EFLIST. Return to step g.

r. In batch mode, to process the binary option, set bit 7 in register D2. Return to step g.

s. In batch mode, to process the execute option, set bit 6 in register D2. Return to step g.

t. In batch mode, to process the permanent-loud-module option, set bit 8 in register D2. Return to step g.

u. In batch mode, to process the severity level option, obtain and convert the hexadecimal value, using the GCBYTE subroutine. Save the value in D1. Return to step g.

v. In batch mode, to process the bias option, obtain and convert the hexadecimal value, using the GCBYTE subroutine. Save the value in register D1. Return to step g.

w. In batch mode, to process the temporary-storage-stack option, obtain and convert the hexadecimal value, using the GCBYTE subroutine. Save the value in SR3 and return to step g.

x. In batch mode, to process the load module option, obtain the load module name (NAMFLD subroutine) and save the name in the parameter list of the open call for the M:LM DCB. Return to step g.

y. If the LOPE control command or the LOAD control input contains an error, print an error message (PRTERR subroutine in on-line mode), close the M:LO DCB, and, in batch mode, about the job or, in on-line mode, issue a CAL3,6 to return to the BTM Executive.

z. In batch mode, if the execute option was not specified, open the load module file (as a temporary file if the permanent option was not specified). If the element file list, EFLIST, is empty, set the binary flag, bit 7 in register D2. Add the system library to the library list, LIBLIST, unless the no-system-library option was specified. Save the address of TSTACK in job lower limit, JOBLL, and the address of the upper limit of the dynamic data area, from LDDATA+2, in job upper limit, JOBUL. Set a default bias in D1 if the bias option was not specified. In on-line mode, restore registers 0-3. Go to the Loader Module, LDR.

6. Flowchart. See Figure 2-18.

2.4.2 Library

1. Module Name. PERMLIB.

2. Purpose. Create user library, :BLIB, in the current account.

3. Entry.

a. Library is entered from the Start Module.

b. Library requires input. See Figure 2-16 for the layout of memory.

(1) In on-line mode, the default assignment of M:BI to the temporary file, BOTEMPα, where α is the COC line number in binary, has been made.

(2) EFLIST, the element file list, contains, in sequence, the identification of each element file specified by the user and/or, in batch mode, the GO file if specified by the user.

(3) LIBLIST, the library list, contains, in sequence, the account numbers of libraries from which primary references are to be satisfied.

(4) BLKBUF contains the address of the buffer for the physical library record.

(5) OBUF contains the address of the buffer for the logical library record.

(6) DYNDATA contains the address of the page for the declaration numbers of external definitions, in halfword entries, and allows the next page, for declare external reference and define origin load items, to be referenced. These buffers are used until all declare external definition, declare external reference, and define origin load items have been processed.

4. Exit.

a. The normal exit is to issue a CAL1,9 1 to terminate the job step.

(1) The exit is taken after user library processing is complete.

(2) A user library, :BLIB, has been created in the current account.

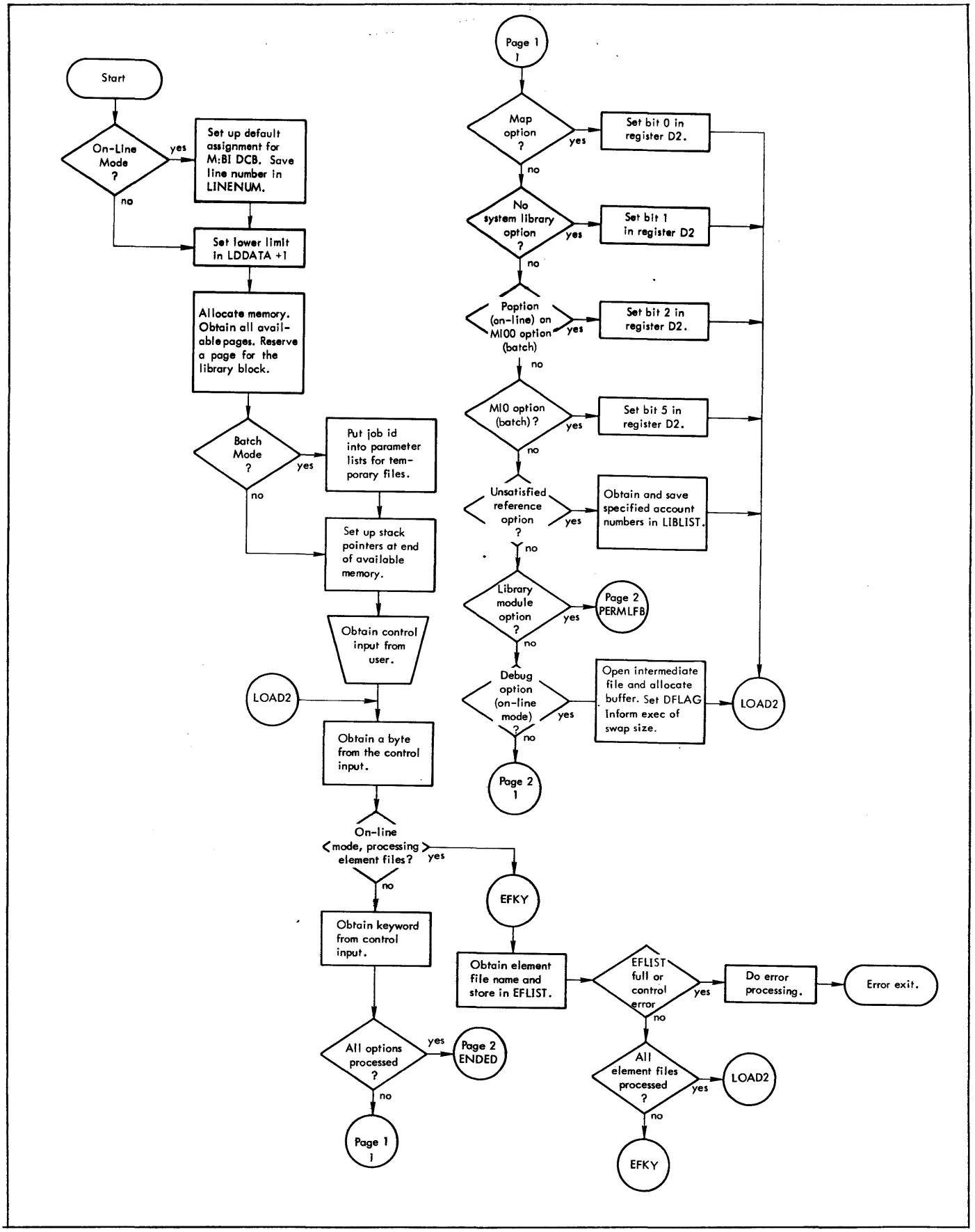


Figure 2-18. Flowchart of Start Module

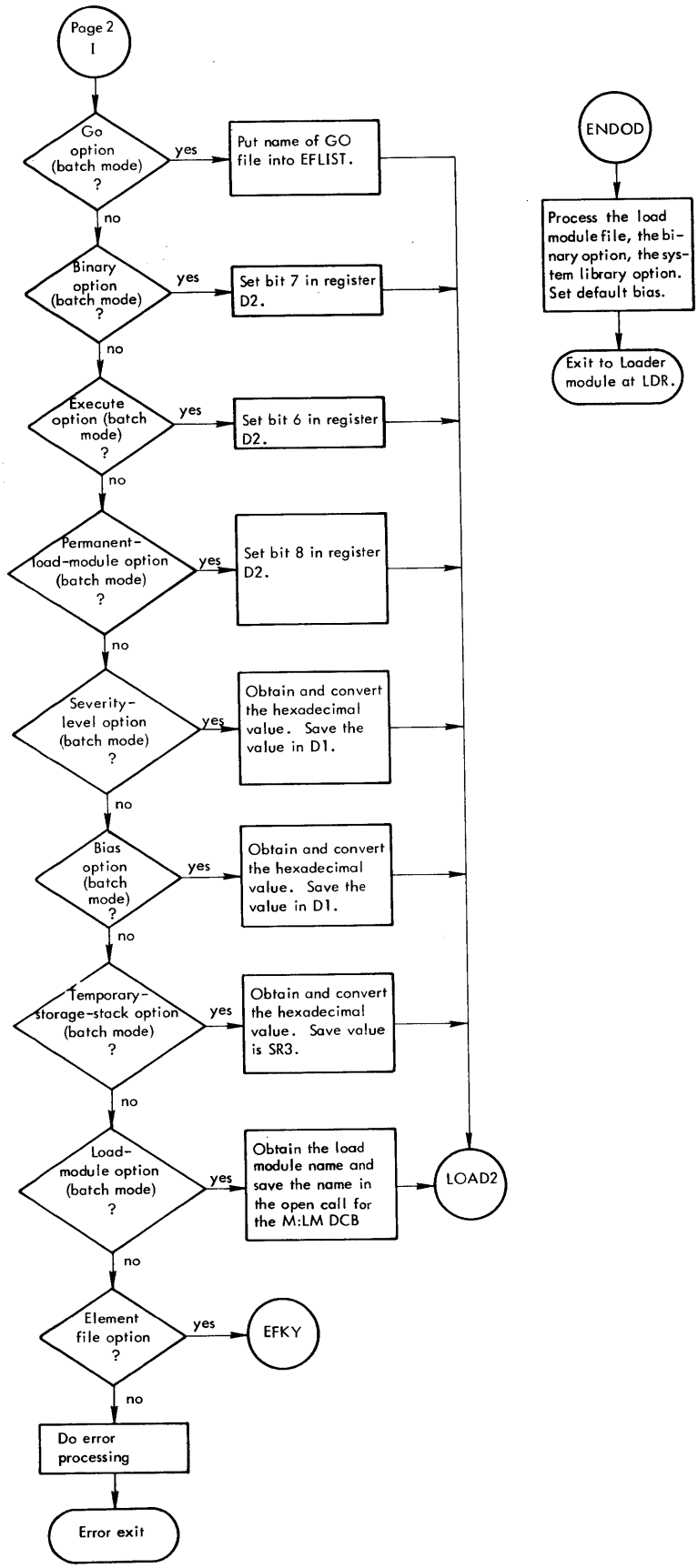


Figure 2-18. Flowchart of Start Module (cont.)

b. The error exit is in batch mode, to abort the job or, in on-line mode, to return to the BTM Executive.

(1) The exit is taken if the number of element files specified by the user exceeds the number of files that can be fit into the internal element file list, EFLIST, built by the loader the error message printed is: "TOO MANY EFS".

(2) The exit is taken if the read of the ROM input fails. The error message printed is: "ILLEGAL ROM DATA", "SEQUENCE ERROR", or "CHECKSUM ERROR".

5. Operation.

a. Prepare to create the user library. See Figure 2-2 for the format of the library. See Figure 2-7 for the layout of memory. Open the file :BLIB in the current account through the M:LM DCB. If the element file list, EFLIST, is empty, set the binary flag, but 7, in register D2. Store register D2 in LDDATA+5, the last of the 6 loader control words. If the number of element files specified by the user exceeds the number of element files that can be fit into the element file list, EFLIST, print an error message (PRTERR subroutine, if on-line) and, in batch mode, abort the job step or, in on-line mode, return to the BTM Executive.

b. Create the user library by copying ROM input into records for the :BLIB file. While processing declare external definition, declare external reference, and define origin load items, calculate and store the declaration numbers of the external definitions in halfword entries in the page at DYNDATA. First put each declare external definition load item into the buffer for the logical library record, OBUF (BYTOUT subroutine). Store all declare external reference and define origin load items in the page after the address is DYNDATA (PUT subroutine). When another type of load item is found, put the declare external reference and define origin load items from the page after DYNDATA into the buffer for the logical library record, OBUF, (BYTOUT subroutine). Create an 02 expression end load item especially for the logical library record in OBUF. Follow the 02 control byte with the halfword declaration numbers of the external definitions saved at the address in DYNDATA (BYTOUT subroutine). Copy bytes into the buffer for the logical library record, OBUF, until the end of the module is reached (BYTOUT subroutine).

c. Step b forms logical records of 108 bytes in OBUF. Each record contains 4 bytes of control information followed by a maximum of 104 bytes of load information. Initialize the record control information with the SETMODE subroutine. Write one page at a time. If a record does not fit into the last words of a page, these words contain garbage and the record begins on the next page. If intermediate logical records contain less than 108 bytes, write 108 bytes anyway, but put the correct logical size in the byte count. Return to step b until all ROM's have been processed.

d. Close the M:LM DCB and issue a CAL1, 9 1 to terminate the job step.

e. If an error in the ROM data is found, print an error message (PRTERR subroutine if on-line) and, in batch mode, abort the job step or, in on-line mode, return to the BTM Executive.

6. Flowchart. See Figure 2-19.

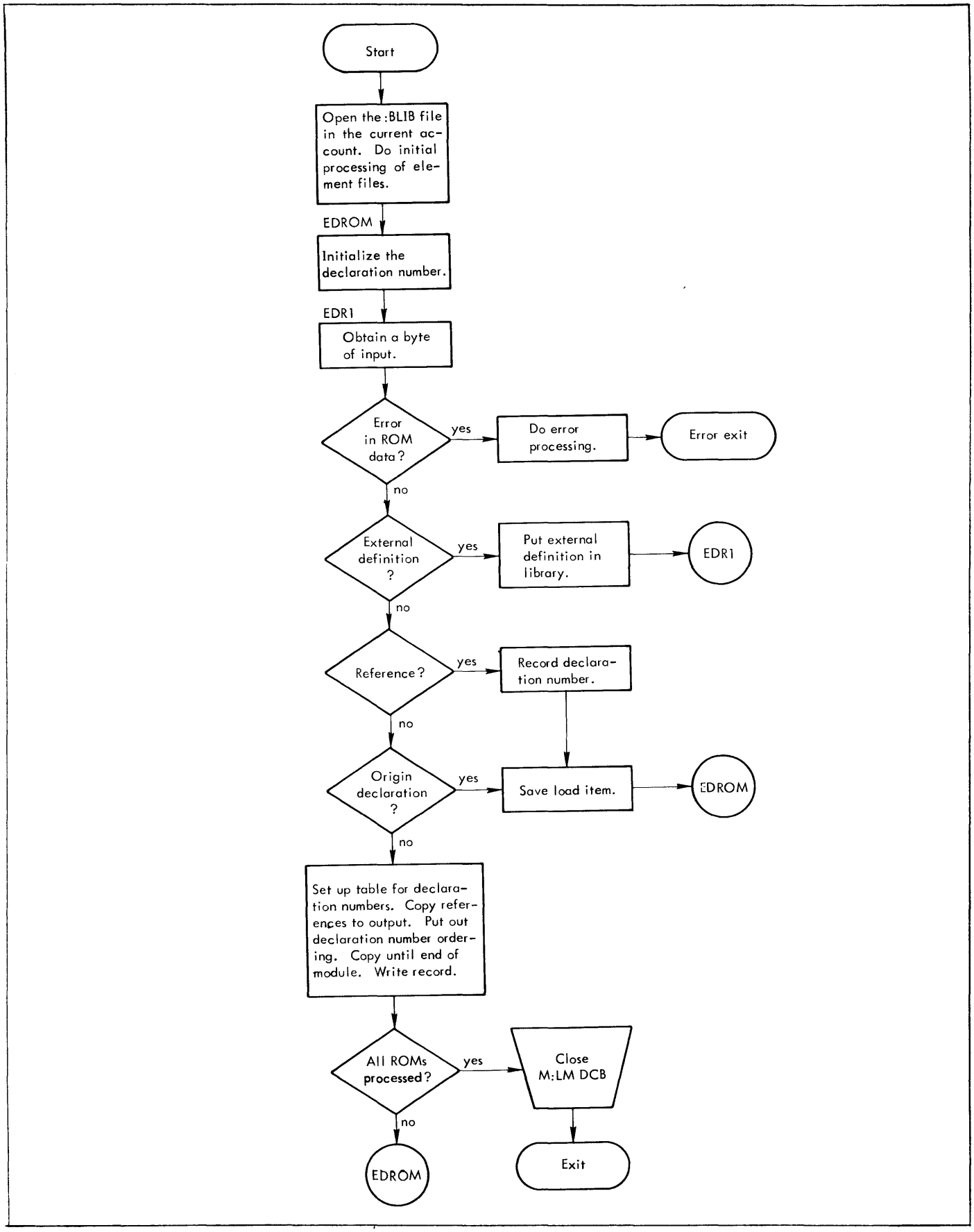


Figure 2-19. Flowchart for Library Modules

2.4.3 Loader

1. Module Name. LDR.

2. Purpose. Load all ROMs.

3. Entry.

   a. Loader is entered from the Start Module.

   b. Loader requires input. See the fields passed from the Start Module.

4. Exit.

   a. The normal exit is to the ENDLOAD Module.

      (1) The exit is taken after all ROMs have been loaded.

      (2) Entries have been made in the expression stack, and the REF/DEF stack.

   b. The error exit is, in batch mode, to abort the job step or, is on-line mode, to return to the BTM Executive.

      (1) The exit is taken if a load item defining an origin results in an expression that cannot be evaluated or if a load item is invalid. The diagnostic output is "ILLEGAL ROM DATA".

      (2) The exit is taken if space is not available in a stack even after all stacks have been pressed to a minimum. The error message printed is : "STACK OVERFLOW".

5. Operation.

   a. Prepare to process load items.

      (1) Save registers SR1-D2, the information obtained from processing the control input, in LDDATA, the six loader control words. Calculate ABSBIAS, the difference between the bias and available memory.

      (2) In on-line mode, set TCB, the address of the Task Control Block, to ORIGIN+X'40'. Use the bias saved in LDDATA+4 to set BACORE, the starting byte address of the next section to be allocated.

      (3) In batch mode, set TCB, the address of the TCB, to the bias saved in LDDATA+4. TCBSIZ determines the present size of the Task Control Block; where CMPWDS is the number of words reserved for use by a processor, 4; TSASIZ is the size of the library error temp stack, 10; ERTSIZ is the size of the library error table, 10. Set BACORE, the starting byte address of the next section to be allocated to point to the location past the Task Control Block, the address of the Task Control Block plus TCBSIZ plus the size specified by the user for the temp stack in the Task Control Block (from LDDATA+2). Using the ENNAM subroutine, enter M:DO as a primary reference in the REF/DEF stack so that the M:DO DCB will be built -- to allow the loaded program to produce snapshot debugs during execution.

   b. Initialize control section 0 and the declaration stack (ENDECL subroutine). If not a library load, process the option that the control section or dummy section is to begin at the next greater multiple of 100_{16} or, in batch, 10_{16} by incrementing BACORE.

c. Obtain a byte of input. Read a card if necessary.

d. If loading from a library and the SATIS flag is not set (the ROM has not satisfied a primary reference), only 03, declare external definition name, and OC, declare nonstandard control section, are allowable load item codes. Anything else causes the stacks to be rolled back because the object module is to be skipped. The remaining records of the module are read and ignored. Return to step b.

e. Use a jump table to determine how to process the load item.

f. If the item is padding, return to step c.

g. If the item declares an external definition name or an external reference name, make sure that the REF/DEF stack has an entry for the name and put an entry into the declaration stack for the item.

   (1) Set up an entry in BUF2.

      (a) Set up words 2-n by transferring each byte of the TEXTC name. Read a record if necessary to obtain the name.

      (b) Set word 1 to zero.

      (c) In word 0, set up only the word count field, byte 0.

   (2) Search the REF/DEF stack to determine whether the name is already in the stack. If the name is not in the stack, add the entry. Research the spare needed for the entry and move the entry into the reserved area.

   (3) If an external definition name is doubly defined, set on both bits 0 and 3 in the type field of the entry in the REF/DEF stack. For a primary reference name, set on bits 1 and 2. For a secondary reference name, set on bit 2.

   (4) Add an entry to the declaration stack for the item.

   (5) Return to step c.

h. If the item is a definition, build an entry in the expression accumulator for the item and move the entry to the expression stack.

   (1) If the item defines an external definition, the name must have been previously declared, i.e., the REF/DEF stack and the declaration stack must have an entry for the name. Obtain the destination of the value of the expression, a displacement into the REF/DEF stack, from the declaration stack.

   (2) If the item defines a forward reference, the two byte forward reference number is the destination of the value of the expression.

   (3) If the item defines the starting address, STTADR is the destination of the value of the expression STTADR has been zeroed. The starting address will be placed in the head record of the load module if a load module is built.

   (4) If the item defines a field, inform the expression routine that a field is to be loaded and that the field is defined by terminal bits (8-14), number of bits (0-7), and location (15-31).

(5) To process an add or subtract declaration, determine whether the declaration is defined by using the displacement obtained from the declaration stack to access the REF/DEF stack. If the declaration is defined, use the value to build the entry for the expression stack. If the expression is not defined, add this expression to the chain for the reference by placing a pointer to the expression stack entry, with byte 0 of the word containing the displacement to the value word in the expression stack entry into word 1 of the REF/DEF stack entry. Save the address of word 1 of the REF/DEF stack entry in the temporary stack so that the REF/DEF stack entry can be modified if the growth of the expression changes the displacement of the value word. Go to step 4.

(6) To process an add or subtract forward reference, determine whether the forward reference number is in the forward reference stack. If not, create an entry for the number. Place a word that points to the expression stack (with byte 0 containing the displacement to the value word in the expression stack) into word 0 of the entry. If an already existing entry is a forward reference and hold type and is defined, use the value from the forward reference stack to build the entry for the expression stack. Otherwise, add the item to the chain for the forward reference by placing a word that points to the expression stack (with byte 0 containing the displacement to the value word in the expression stack entry) into word 1 of the forward reference stack entry. Save the address of word 1 of the forward reference stack in the temporary stack so that the forward reference stack entry may be modified if the growth of the expression stack changes the displacement to the value word.

(7) If the words containing the expression control bytes increase, modify the entries in the REF/DEF stack and the forward reference stack that point to the value words of the expression stack entry.

(8) Evaluate the expression. If a value has been found, flag the entry in the expression stack as defined. If an expression is not used, remove the expression from the stack. If the destination of the expression is an external definition or a forward reference and the value is not a constant, convert to the value to an address. If the destination of the expression is an external definition, use the displacement into the REF/DEF stack from the expression stack entry to access the REF/DEF stack. If the REF/DEF stack entry is an external definition or a dummy section, return to step c. Otherwise set the external definition type flag, word 0, bit 0, and the resolution field, word 0, bits 4-6, in the REF/DEF stack entry. Obtain the link value from word 1 of the REF/DEF stack entry. Replace the link value with the value of the external definition, an address or a constant.

(9) If the item defines a forward reference and hold item, satisfy any existing references and save the value for future references until the end of the module is reached.

(10) If the item defines an origin and the expression is defined, check that the value is above the bias and the background lower limit. If so, subtract the absolute bias, store the resulting value in LOC, the value of the load location counter, and check that LOC is within bounds. If LOC is valid, return to step c. Origin load items enable the loader to determine where elements are to be loaded within a control section. The ROM supplies at least one origin load item for each control section. If the expression defining an origin cannot be evaluated or if the value is out of bounds, output a diagnostics and, in batch mode, abort the job or, in on-line mode, return to the BTM Executive.

(11) To satisfy any chain of expressions, use the link value as a displacement into the expression stack to obtain the chained entry. Replace the expression control byte in the entry with an add constant with resolution. Obtain the link value from the value word of the entry. Replace the link value with the value of the load item (use the complement of the value if the expression control byte specified subtraction.) Attempt to evaluate the expression. Return to the beggining of step (11) until all chained entries have been processed.

(12) Then return to step c.

i. If the item declares a standard control section, put the size and memory protection class into a declaration stack entry. Increment BACORE, the byte address of the next section to the allocated. Return to step c.

j. If the item declares a nonstandard control section, put the type, the size, memory protection class, and base into a declaration stack entry. Increment BACORE, the byte address of the next section to be allocated. If the standard control section has been referenced, put in the size. Otherwise, put the new BACORE into the base of the standard control section. Return to step c.

k. If the item declares a dummy section, the item must have been preceded by a declare external definition name for the label associated with the first location. Use the entry in the declaration stack created by the declare external definition name in order to access the entry for the name in the REF/DEF stack. If the entry shows that the dummy section has not been allocated, allocate the dummy section. In on-line mode or in batch mode with the execute option specified, put blank common into memory. See Figure 2-6 for the layout of memory.
Increment BACORE, the byte address of the next section to be allocated. Store the size and address in the REF/DEF stack entry. Store the address in other chained expressions using the procedure described in step h (11). Add an entry for the load item to the declaration stack. If the entry in the declaration stack for control section 0, the first entry, has a zero size, store the new BACORE, the byte address of the next section to be allocated, into the base address field. Return to step c. If the REF/DEF stack entry shows that the dummy section has been allocated, check that the new size is not greater than the original size. If the new size is greater, abort the processing. Otherwise return to step c.

l. If the item is a load absolute item, load the specified number of bytes. R5, bits 28-31, contains the number with 0 implying 16. Process as fast as possible. Advance the load location counter. Return to step c.

m. If the item is a load relocatable long form, increment the load location counter, LOC. Depending on the control bit, bit 4, read either a one-byte or a 2 byte name number. Put the control bit that determines whether relocation is to be relative to a forward reference or to a declaration into SR1. Put the address relocation code into register 6. Go to step o.

n. If the item is a load elocatable short from, increment the load location counter, LOC. Put the control bit that determines whether relocation is to be relative to a forward reference or to a declaration into register SR1

o. Read the word immediately following this load item. Ensure that the load location counter, LOC, is a word address. Form an expression, put the expression into the expression stack, and evaluate the expression if possible. Increment the load location counter, LOC, by 4 and check that the new value is within bounds. If the value is within bounds, return to step c.

p. Whenever the load location counter, LOC, is incremented, the counter is checked for being within bounds. Since the core image is being built upward in the dynamic data area, LOC must remain below TEMPBAS, the address of the lowest stack. See Figure 2-16 for the layout of memory. If necessary, press the stacks to remove unneeded space (PRESSTK subroutine) and leave more space in which to build the core image. If the configuration changes, declare the new configuration to the BTM Executive (SETSIZE subroutine). Return to step b. If space cannot be made available, abort processing.

q. If the item is a repeat load, only one of three load items to be repeated is legal. If a load absolute item follows the repeat load item, get the absolute data and load the data the specified number of times. Otherwise set up a chain and load normally. Return to step c.

r. If the item is the special O2 expression end load item created for the user library, use the declaration numbers (of the external definitions) following the control byte to rearrange the external definitions, chaining the items so that the first is the first source, the second is the destination of the first and the next source. Continue until the first is a destination so that the loop is closed. Return to step c.

s. If the item is invalid, print an error message (PRTERR subroutine if on-line) and, in batch mode, abort the job or, in on-line mode, return to the BTM Executive.

t. If space is not available in a stack, press all stacks to a minimum (PRESSTK subroutine). If space is still not available, print a diagnostic (PRTERR subroutine, if on-line) and, in batch mode, abort the job or, in on-line mode, return to the BTM Executive. Otherwise, inform the BTM Executive of the change in swap size (SETSIZE subroutine).

u. If the item identifies the end of the object module, read the severity level. Replace the severity level from the ROM if the current severity level is greater. If more ROMs are to be loaded, return to step b. Otherwise, load from libraries to satisfy all primary references in the REF/DEF stack except those having names that begin with F: or M:, for which the loader will build DCBs. If a library module satisfies a primary reference, load the module. When all loading is complete, go to the Endload Module.

6. Flowchart. See Figure 2-20.

2.4.4 ENDLOAD

1. Module Name. ENDLOAD.

2. Purpose. Print map. Build DCBs. Build a DCB Name Table for Monitor use. Build a TCB.

3. Entry.

   a. ENDLOAD is entered from Load.

   b. ENDLOAD requires input, the entries in the loader stacks.

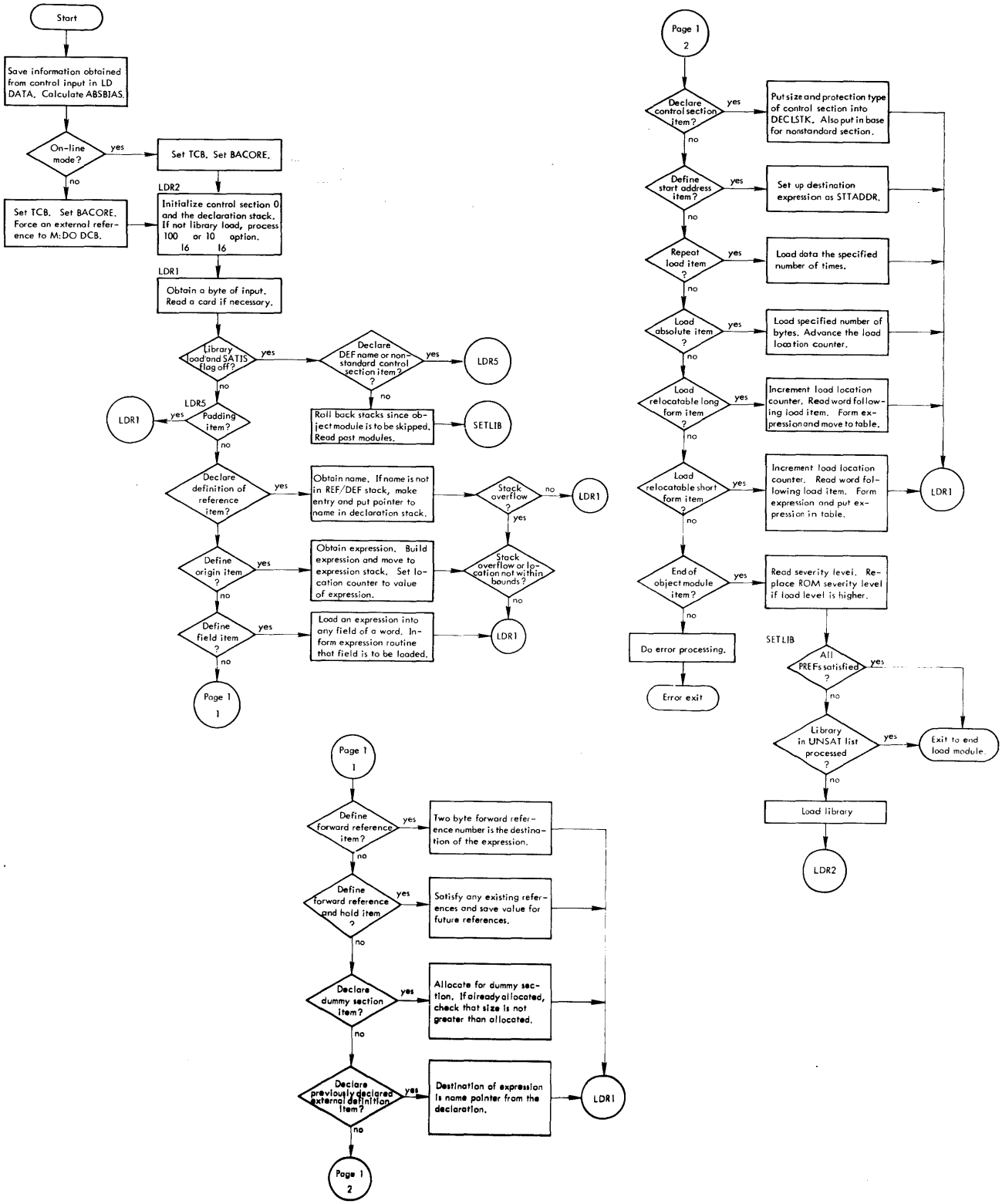


Figure 2-20. Flowchart for Loader Module

4. Exit.

   a. The normal exit is to the Execute module.

      (1) The entry point for batch mode or for on-line mode with the execute option specified is GOEXEC. For on-line mode, when the execute option is not specified, the entry point is SMALLER.

      (2) Data is passed to the other modules.

         (a) Register 0 contains the address of the TCB.

         (b) SR1 contains the highest severity level encountered.

         (c) SR2 contains the start address for execution.

         (d) SR3 contains the bias.

         (e) SR4 contains the address of the end of the TCB.

   b. The error exit is, in batch mode, to abort the job step or, in on-line mode, to return to the BTM Executive. The exit is taken if there is not enough room to continue processing. The error message printed is: "STACK OVERFLOW" or "TOO MANY DCBS."

5. Operation.

   a. Close the M:LI DCB if the M:LI DCB is open. Allocate the unallocated ASECT code. In on-line mode, if the program is to be run in debug mode, finish the intermediate file by inserting the last element file header. Calculate the ABSBIAS word displacement, the difference between the load and the execute bias. In batch mode, define the highest location and the lowest location.

   b. Print a load map. If map is specified, print a load map of external definitions and external references. Do not map element file head items or internal symbols that are not external definitions. If map is not specified, print only secondary and primary references. In either case, skip primary and secondary references for F: and M: names since such names refer to DCBs.

   c. Form the Task Control Block (TCB) for the user. See section 2.2.5.,1. Task Control Block, and Figure 2-7.

   d. Build tree and DCB Name Table. D1 contains the highest location used. Build the tree and the DCB Name Table at the next available page. Put the address of the DCB Name Table into the Task Control Block (TCB). If there is not sufficient room to continue processing, print an error message and, in batch mode, abort the job step or, in on-line mode, return to the BTM Executive.

   e. Build DCBs for referenced Fi names. Also add DEF'd F: names and M: names to the DCB Name Table. If a DCB is DEF'd, print a warning message. If a DCB is a secondary reference, ignore the DCB.

   f. Build a DCB Name Table entry and a DCB for assigned F: names.

      (1) Put standard assignments into F:101-F:108 for assignment type, function, and operational label. For batch mode, set assignment type to device. For on-line mode, set assignment type to user terminal. Set function to OUTIN.

(2) Override standard assignments if the assign for the DCB specifies different values.

(a) In batch mode, check for assigns. If assignments were given, go through the ABS records. Set up the IDA temporary file for unusual assigns.

(b) In on-line mode, do on-line processing. If the debug option was specified, print the map of undefined internal symbols for each element file. Request and read assign options for the DCB -- file name, optional account number, and optional password -- and set up parameter words for each option. If file name is greater than 11 characters or if account number is greater than 8 characters, print an error message and repeat the prompt to request assignment information for a DCB. If password is specified with no account, default to the login account in the on-line Job Information Table (AJIT). Read the DCB options -- function, release, list. Store the appropriate values and set flags for the options.

g. If the DCB name is not an F: name and the DCB is referenced, incorporate the assign information using the ASNMERG subroutine. If the DCB name is not in the DCB Name Table, build the DCB and add the DCB name to the DCB Name Table.

h. Set up standard assignments for Monitor DCBs.

i. Set up registers to transfer information.

(1) Put the address of the TCB into register 0.

(2) Put the highest severity level encountered into register SR1.

(3) Put the start address for execution into register SR2.

(4) Put the bias into register SR3.

(5) Put the address of the end of the TCB into register SR4.

j. Close the M:LO DCB.

k. If F4:COM is overlayed by DCBs, print an error message and, in batch mode, abort the job step or, in on-line mode, return to the BTM Executive.

l. In on-line mode, output the severity level message.

m. In on-line mode, if the debug option was specified, request and read values to satisfy any primary references and/or internal references. Go to the execute module at only point GOEXEC.

n. In on-line mode, if the debug option has not specified, output the execute inquiry and read the reply. If the reply is invalid, print an error message and return to the BTM Executive. If the reply is "N", go to the Execute Module at entry point SMALLER. If the reply is "S", obtain the start address specified by the user, store the start address, and go to the Execute Module at entry point GOEXEC. If the reply is "Y", go to the Execute Module at entry point GOEXEC.

o. In batch mode, go to the Execute Module at entry point GOEXEC.

6. Flowchart. See Figure 2-21.

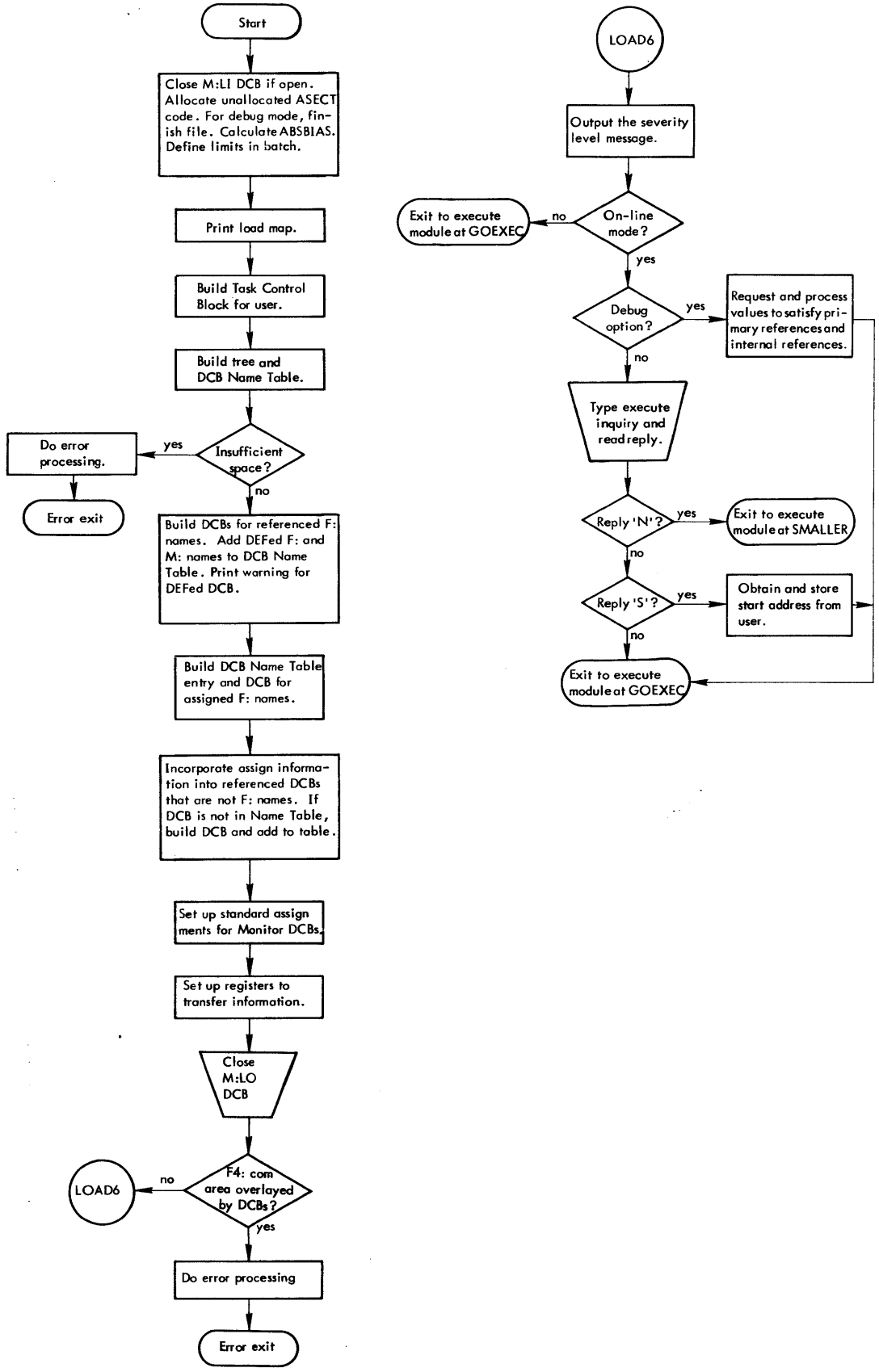


Figure 2-21. Flowchart for Endload Module

2.4.5 Execute.

1. Module Name. GOEXEC.

2. Purpose. In batch mode, if the load module option is specified, build a load module. In on-line mode, if an execute option is not specified, set up to save the user program. In either mode, if an execute option is specified, provide for execution of the user program.

3. Entry.

   a. Execute is entered from Endload at entry point GOEXEC for batch mode and for on-line mode with the execute option specified. For on-line mode when the execute option is not specified, the entry point is SMALLER.

   b. Execute requires input.

      (1) Register 0 contains the address of the TCB.

      (2) SR1 contains the highest severity level encountered.

      (3) SR2 contains the start address for execution.

      (4) SR3 contains the bias.

      (5) SR4 contains the address of the end of the TCB.

4. Exit.

   a. In batch mode, if the execute option is not specified, issue a CAL1,9 1 to exit from the job step. A load module has been built.

   b. In on-line mode, if the execute option is not specified, return to the BTM Executive. The user program is set up to be saved if desired.

   c. In batch mode, if the execute option is specified, transfer control to the user program. The user program has been set up for execution.

   d. In on-line modes if the debug option is specified, transfer control to the DELTA subsystem. The user program is set up for execution.

   e. In on-line mode, if the execute option is specified, return control to the BTM Executive. The user program has been executed and messages output regarding the execution.

5. Operation.

   a. At entry point GOEXEC in batch mode, if the execute option is not specified go to step b. Otherwise link the user DCB Name Table to loader DCBTAB. Set the 01 protection type and free the unused pages. If the program is smaller than the loader, free only to the top of the loader. Open all pages. Transfer control to the start address of the user program.

   b. In batch mode, when the execute option is not specified, build a load module. Fill in values in the head and tree. Change the format of the REF/DEF stack, RFDFSTIK, to conform to the format built by the overlay loader so that symbolic debugs can be used. Convert the type and resolution to that of the overlay loader. Issue a CAL1,9 1 to exit from the job step.

c. At entry point GOEXEC in on-line mode, set the address in register D4 so that the execute option may be processed.

d. From both entry point GOEXEC and entry point SMALLER, do on-line processing. Put the address of the Task Control Block into register 0. If blank common is at the top, declare the exact configuration. If the debug option was specified, go to step f. If the execute option was specified, go to step g.

e. Set a special flag for saved files and issue a CAL3,6 to exit to the BTM Executive.

f. Process the debug option. If the symbol tables are not loaded, attempt to load the tables. Put the parameters in the window page; the address and size of the global symbol table, the starting address of the user program, the address and size of the internal symbol table, and the last word of the user program, following M: and F: DCBs. Place DE, the first two characters of the debug subsystem name into register 0. Issue a CAL3,4 to load the debug subsystem and to transfer control to the subsystem DELTA.

g. Start the user program at the user level. Register 0 contains the first half of the program status doubleword, PSD, to be used when the new process is initialized. Bits 0-3 are the condition code (CC); bits 4-7 are the floating controls (FC); and bits 15-16 are the instruction address (IA).

h. When a return from the user is made, determine whether a CAL3,6 was issued to return to the next higher level. If a CAL3,6 was issued, go to step i. If not, do the break processing. Output the inquiry and read the reply on whether to continue processing. If the reply is anything other than "Y", go to step j. Otherwise issue a new line. Load register 0 so that the user program may be continued and go to step g.

i. Output the user-exit diagnostic.

j. Issue a CAL3,6 0 to return to the BTM Executive.

6. Flowchart. See Figure 2-22.

2.5 SUBROUTINE ANALYSIS

2.5.1 ASNMERG

1. Purpose. Merge assignments (record in BUF) into the DCB pointed to by register 6.

2. Entry. ASNMERG is called by the Endload Module. Register 0 is the link register. Upon entry, BUF contains the record with the assignments to be merged. Register 6 contains the address of the DCB to which the assignments apply.

3. Exit. Return to the calling routine.

4. Operation. Handle the first-length parameters first, then the variable-length parameters.

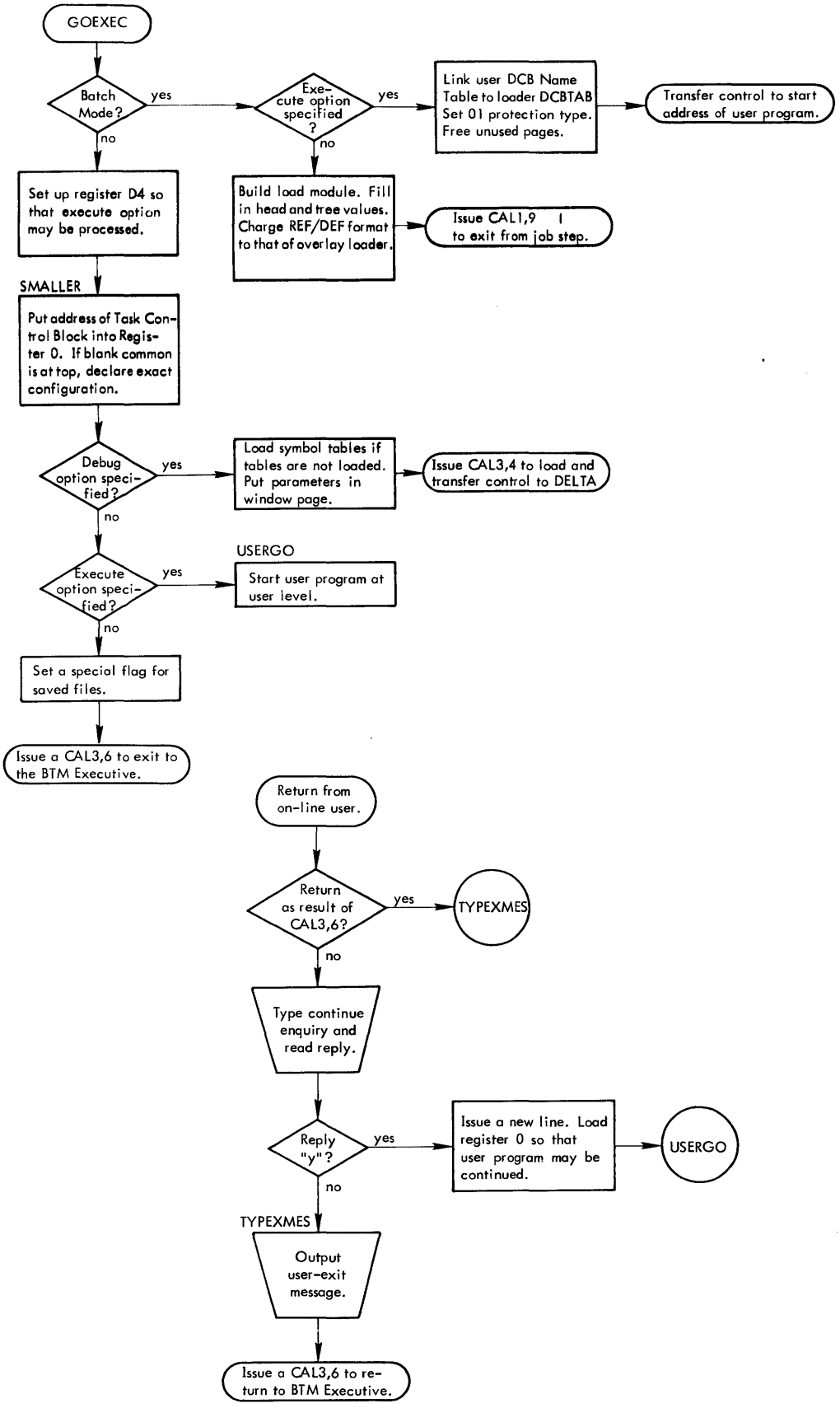


Figure 2-22. Flowchart for Execute Module

2.5.2 BINTOHEX

1. Purpose. Convert the binary number in register 7 to hexadecimal (EBCDIC) in registers D3 and D4.
2. Entry. BINTOHEX is called by the Library Module, the Loader Module, and the Endload Module. SR4 is the link register. Upon entry, register 7 contains the binary number to be converted.
3. Exit. Return to the calling routine.
4. Operation. Convert the binary number in register 7 to an EBCDIC number in registers D3 and D4. Suppress leading zeros.

2.5.3 BYTOUT, WRTREC

1. Purpose. Put a byte into the library.
2. Entry. BYTOUT is called by the Library Module. WRTREC is a second entry point. SR4 is the link register. Register 5 contains the byte to be stored. OBUF is the buffer for the logical library records.
3. Exit. Return to the calling routine.
4. Operation.
   (1) BYTOUT. Put a byte into OBUF at the next available location. Use byte 3 of OBUF, record size, to determine the next available location.
   (2) WRTREC. Update record size. When record size is 108 bytes, move the record from the logical library record buffer, OBUF, to the physical library record, BLKBUF. When BLKBUF, which is one page long, will not hold any more logical records, write out the page.

2.5.4 CHKDECLD

1. Purpose. Check a declaration for definition.
2. Entry. CHKDECLD is called by the Loader Module. Upon entry, register 5 contains the expression control byte.
3. Exit. Return to the calling routine.
4. Operation. If the declaration is defined, put in the value with proper resolution and change register 5 to add constant. Just right shift the value if the destination of the value is a core destination. If the declaration is not defined, add the declaration to the chain for the reference. Within the routine, register 7 contains the declaration number.

2.5.5 CKDCB

1. Purpose. See whether the DCB name pointed to by SR1 is in the DCB Name Table.
2. Entry. CKDCB is called by the Endload Module. Upon entry, SR1 contains the address of a DCB name.
3. Exit. If the DCB name is in the DCB Name Table, return to the calling location plus 2. Otherwise, return to the calling location plus 1. Register 6 contains the address of the DCB.

4. Operation. Check the name pointed to by SR1 against the entries in the DCB Name Table. If the name does not match an entry, exit to the calling location plus 1. If the name does match, put the address of the DCB into register 6 and exit to the calling location plus 2.

2.5.6 CRLF

1. Purpose. In on-line mode, issue a new line on the user terminal.
2. Entry. CRLF is called in on-line mode by all the modules. Register 1 is the link register.
3. Exit. Return to the calling routine. A new line has been issued to the user terminal
4. Operation. Issue a carriage return, then a line feed.

2.5.7 GCBYTE

1. Purpose. Obtain a byte from the control command.
2. Entry. GCBYTE is called by the Start and the Endload Modules. SR4 is the link register.
3. Exit. Return to the calling routine. Register 5 contains the byte from the control command.
4. Operation. In batch mode, obtain a byte from BUF, the buffer for the control command. In on-line mode, issue a call to return in register 0 (in EBCDIC format) the next character in the Teletype input buffer. If there is no activation character in the buffer, the LOAD subsystem is dismissed until an activation character is typed. In either mode, put the byte obtained into register 5.

2.5.8 GCBYTEA

1. Purpose. In on-line mode, read a character from the Teletype input buffer.
2. Entry. In on-line mode, GCBYTEA is called by the Endload Module. SR4 is the link register.
3. Exit. Return to the calling routine. Register 5 contains the character obtained.
4. Operation. Issue a call to return in register 0 (in EBCDIC format) the next character in the Teletype input buffer. If there is no activation character in the buffer, the LOAD subsystem is dismissed until an activation character is typed. Put the byte obtained into register 5.

2.5.9 GETEFHED

1. Purpose. Search for next element file header. Is in the REF/DEF stack (RFDFSRCH subroutine). If the element file header is not found, return to the calling address+1. If the element file header is found, save the new start index, print the header and the element file name. Compute the address of the value word and set up the symbol table base offset and local length. Compute the byte address of the name. Reset the USYM start index. Make USYMHEAD correspond to this element file.
2. Entry. GETEFHED is called by the ENDLOAD module. SR4 is the link register.
3. Exit. Return to the calling location plus one if the next element file header is not in the REF/DEF stack. Otherwise, return to the calling location plus two.
4. Operation. Determine whether the next element file header.

2.5.10 GETULOC

1. Purpose. In on-line mode, obtain the desired location from the user.
2. Entry. In on-line mode, GETULOC is called by the Execute Module. Register 3 is the link register.

3. Exit. Return to the calling routine.
4. Operation. Issue a CAL3,7 0 to swap the desired page from the user level swap storage area to the subsystem area of memory in order to examine the page to determine why control has been returned to the LOAD subsystem.

2.5.11 PRESSTK

1. Purpose. Remove space in all stacks except TSTACK.
2. Entry. PRESSTK is called by the Loader. The Endland, and the Execute Modules. Register 0 is the link register.
3. Exit. In batch mode, return to the calling routine. In on-line mode, branch to the SETSIZE subroutine.
4. Operation. Remove space in all stacks except TSTACK. Leave a minimum of space in TSTACK to allow for non-recursive use which does not check for stack overflow. In batch mode, return to the calling routine. In on-line mode, branch to the SETSIZE subroutine.

2.5.12 PRTNXT

1. Purpose. Print next map entry.
2. Entry. PRTNXT is called by Endload Module. SR2 is the link register.
3. Exit. Return to the calling routine.
4. Operation.
   (1) Obtain the correct map text code. If the entry is a reference, go to step (3). If the entry is a constant, go to step (2). Otherwise put the word value plus a byte offset into the output buffer.
   (2) Convert the binary value to EBCDIC (BINTOHEX subroutine) and store the result in the output buffer.
   (3) Store the map code and name in the output buffer.

2.5.13 PRTERR

1. Purpose. In on-line mode, output a message.
2. Entry. PRTERR is to print all error messages. Register 6 is the link register. Register 1 contains address of error messages.
3. Exit. Return to the calling routine.
4. Operation. Output a message using The WTTY and The CRLF subroutines.

2.5.14 RFDFSRCH

1. Purpose. To determine whether an item is in the REF/DEF stack.
2. Entry. RRDFSRCH is called by the Loader and the Endload modules. SR4 is the link register. Register 4 contains the item to be protected. Register 5 contains the mask job used in compare. Register 6 contains the index.
3. Exit. Return to the calling address + 1 if the item is not in the REF/DEF stack. Otherwise, return to the calling address + 2.
4. Operation. Search the REF/DEF stack for the item in register 4. Use the mask in register 5 for the compare. Search the REF/DEF stack until the index in register 6 is equal to or greater than the index of the stack. If the item is not found, exit to the calling address + 1. If the item is found, exit to the calling address + 2.

2.5.15 SETMODE

1. Purpose. Set indicators at module and when processing a module.
2. Entry. SETMODE is called by the library, the Loader, and the Endload Modules. Register 2 is the link register.
3. Exit. Return to the calling routine. MODNUM, the current module number, has been incremented. SEQNUM has been set to a-1. SATIS, the flag for a module that DEF'ed a REF, has been zeroed. LASTCARD, the flag for end-of-module has been zeroed. Register 1 contains the previous contents of LASTCARD.
4. Operation. Increment MODNUM, the current module number. Set SEQNUM to a minus one. Zero SATIS, the flag for a module that DEF's a REF, and LASTCARD, the flag for end-of-module. Save the previous contents of LASTCARD in register 1.

2.5.16 SETSIZE.

1. Purpose. In on-line mode, inform the BTM Monitor of the configuration of memory to allow correct and efficient swapping.
2. Entry. In on-line mode, SETSIZE is called by the Start, Loader, Endload, and Execute modules. Register 0 is the link register.
3. Exit. Return to the calling routine. The BTM Executive has been informed of the new configuration.
4. Operation. Compute the current dynamic data size and the common dynamic data pages, the number of pages in the stack area. Issue a CAL3, 11 0 to inform the BTM Monitor of the configuration. In register 0, bits 0-7 are non zero; bits 8-23 are zero; and bits 24-31 are the number of program data pages swapped in and out. In register 1, bits 0-7 are the pure procedure pages, unmodified during execution and swapped in only; bits 8-15 are the dynamic data pages swapped in and out; bits 16-23 are the inactive pages not swapped, and bits 24-31 are the common data pages, growing down from the top of available memory and swapped in and out.

2.5.17 STCR

1. Purpose. To skip to the end of the control card in BUF.
2. Entry. STCR is called by the Endload Module. SR4 is the link register.
3. Exit. Return to the calling routine.
4. Operation. Obtain bytes from the control command in BUF until the ENDLOAD indicator is set.

2.5.18 STORFLD.

1. Purpose. Add a field value to an expression.
2. Entry. STORFLD is called by the Loader Module. SR4 is the link register.
3. Exit. Return to the calling routine.
4. Operation. Add the field value to the expression.

2.5.19 USYMPRNT

1. Purpose. In on-line mode, print the undefined symbol map.
2. Entry. USYMPRNT is entered from the Endload Module. SR4 is the link register.
3. Exit. Return to the calling routine.
4. Operation.
   a. Print the undefined internals heading. Issue a new line (WTTY1 and CRLF subroutines). Zero the element file header index.
   b. Search for the next element file header, (GETEFHED subroutine). If none are found, exit.
   c. Otherwise determine whether an item is in the REF/DEF stack. If not, return to step b.
   d. Save the new start index. Print the USYM Message (WTTY1 subroutine). Compute the byte address and print the symbol name (WTTY1 and CRLF subroutines.) Return to step b.

2.5.20 WRTREC. See BYTOUT.

2.5.21 WTTY, WTTY1.

1. Purpose. Type a message in TEXTC format (WTTY1) or in TEXT format (WTTY).
2. Entry. WTTY1 is called by the Endload and the Execute Modules. WTTY is called by the Start, the Library, the Loader, and the Endload Modules. Register 1 is the link register.
3. Exit. Return to the calling routine.
4. Operation. Issue a CAL3,1 0 to output the message.

3.0 RUN

3.1 FUNCTIONAL OVERVIEW

3.1.1 General Description

RUN is a BTM subsystem which allows the on-line user to execute a previously formed load module (NON-PAGED) under control of the RUN subsystem. Several BPM services which otherwise would not be available to an on-line user are simulated by RUN, allowing overlayed load modules (M:SEGLD) to be executed. Thus, most load modules capable of batch execution will also execute on-line.

Its capabilities include the re-biasing of a load module into core images within the BTM user area, the application to the load module of any "MODS" (!MODIFY CARDS), modifying instructions or data, displaying symbol address values and memory cell contents, and the insertion or deletion of instruction breakpoints.

The RUN subsystem is entered by the Executive command !RUN, and upon entry, requests the load module identification as follows:

LOAD MODULE FID:

where upon the user supplies the file identification in the form of

NAME [(ACCT[, PASS])]

RUN is s two-level subsystem which means that RUN can relinquish execution to a user program upon recieving a ";G" (GO) or ";P" (PROCEED) command. RUN maintains control over the user program in the event of traps, exits, breakpoints or aborts.

Symbol tables for the users program (root or overlay segments) are made available to RUN. These include all REFS and DEFS of the specified segment and its backward path. The symbol tables are part of the load module and are loaded with it. (See OVERLAY LOADER TECHNICAL MANUAL 90 18 03).

The various RUN commands are summarized in Figure 3-1.

COMMAND	DESCRIPTION
[s];S	Selects a symbol table by overlay segment name, or root if S is omitted
[e]/	Open cell and print contents (of cell e)
e1, e2/	Displays the contents of cell e1 through e2 and open cell e2
(RET)	Closes currently open cell
e (RET)	The expression e is stored into the currently open cell
(LF)	Opens the next higher cell
e (LF)	Modifies the currently open cell and opens the next higher cell
↑	Opens the next lower cell
e ↑	Modifies the currently open cell and opens the next lower cell
(TAB)	Displays and opens the cell addressed by the last quantity typed
e (TAB)	Modifies the currently open cell and opens the cell addressed by the last quantity typed
;G	Begin execution
e; G	Begin execution at location e
;P	Proceed from breakpoint
;B	Display all breakpoints
e;B	Set a breakpoint at location e
N;B	Clear breakpoint N ($0 < N \leq 8$)
e;I	Set execution counter to location e
e;C	Set condition codes to that of expression e
;A	Display locations in hexadecimal
;R	Display locations as symbols plus a hexadecimal offset
;I	Display the execution counter
;C	Display the current condition codes
;Q	Display most recently displayed value

Figure 3-1. RUN Commands

3.1.2 Error Messages

Error messages displayed to the user fall into three catagories:

1. Those that are command syntax errors.
2. Those that are machine trap diagnostics from an executing load module.
3. Those that are associated with the loading of the users load module.

The command syntax error is "?" and a re-prompt (BELL) by the subsystem is initialized.

Trap messages with the location of the offending instruction in symbolic (;R) or hexadecimal (;A) are:

1] NONEXIST INST AT ____
2] NONEXIST MEM REF AT ____
3] PRIV INST AT ____
4] MEM PROT VIOL AT ____
5] UNIMP INST AT ____
6] STACK LIMIT AT ____
7] FIXED OVERFLOW AT ____
8] FLOAT FAULT AT ____
9] DECIMAL FAULT AT ____
10] BAD CAL AT ____

Messages associated with loading of the users load module are:

1] NAME TOO LONG
2] CANNOT OPEN LOAD MODULE
3] FILE NOT PROPER LOAD MODULE
4] LOAD MODULE DOES NOT FIT

3.1.3 Restriction

RUN subsystem is capable of loading only those load modules which are "NON PAGED", and whose size will fit into the BTM user area.

ABSed load modules are permissable only if they are biased at a page beyond the TCB (REGPAGES+256), since a relocation can not be done.

A rough estimate can be made to determine whether or not a load module may or may not fit into the BTM user area. The formula is

USERSIZE -2.5K (SIZE OF RUN SUBSYSTEM) -TREE (SIZE OF TREE) = CORE AVAILABLE

Non overlaid load modules may have up to four files open at the same time. Overlaid may have only three at a time.

3.2 INTERFACES

3.2.1 To the BTM Executive

When the BTM user invokes RUN via a !RUN command, it is given control by the BTM Executive. Figure 3-2 shows the memory layout of RUN after it has obtained control, and has loaded the user load module.

RUN operates as a two-level subsystem which has control over execution, traps, aborts and breakpoints of a user program. RUN uses standard BTM subsystem cals to start execution of the users program, set swap size, and inform the user of trap faults. The reading and writing of a users terminal is accomplished via the normal BTM CAL3's (CAL3,0 for reading; CAL3,1 for writing). The activation type (3) is set to activate on (LF), (RET), (ESC), ↑ , /, or =. It is set when the user is "popped up" to the RUN subsystem level at location DEBUG. Prior to returning to the user (CAL3,5) it is set to activate on (CR) or (LF) only (4). This is done at GOTOUSER.

3.2.2 To the User

RUN interfaces to the user via the RUN command verbs (see Figure 3-1) at subsystem level and via breakpoints at execution level. RUN prompts for input with a "BELL" for input commands through the users terminal at initialization time and also for any "POP UP" reason (breakpoints, trap conditions,

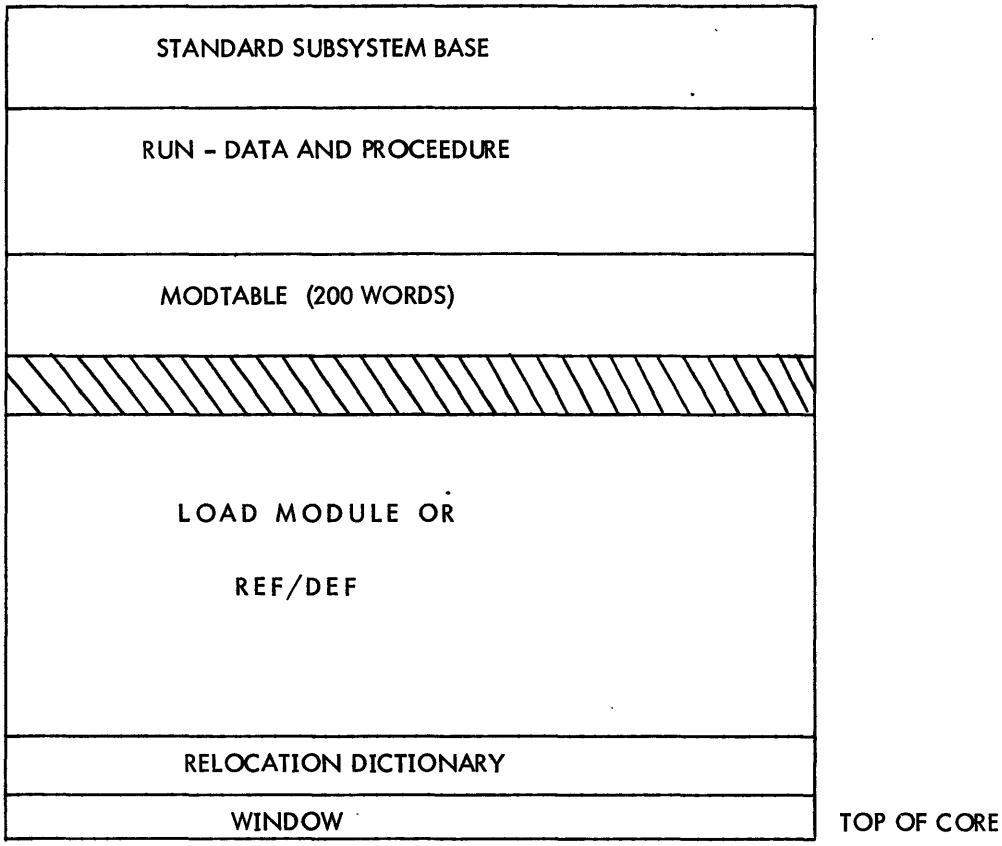


Figure 3-2. RUN SUBSYSTEM IN MEMORY

(ESC) - (ESC)) encountered during execution at user level. The BTM user at executive time has all the standard BTM system CAL's which apply to user level and the BPM system CALs available to the BTM user. (See BTM Reference Manual, 90 15 77, Appendix A). In addition, the RUN subsystem simulates certain BPM CALs not available to the BTM user. These include M:GP, M:FP, M:SMPRT, M:GL, M:GCP, M:FCP, and M:TIME.

Since RUN subsystem is a debugging aid for user execution, RUN subsystem swaps portions of users process into its window page for either examination and/or modification. This is done when control is returned to RUN from a user level, and certain debugging commands are invoked. (i.e./,;B,;Q)

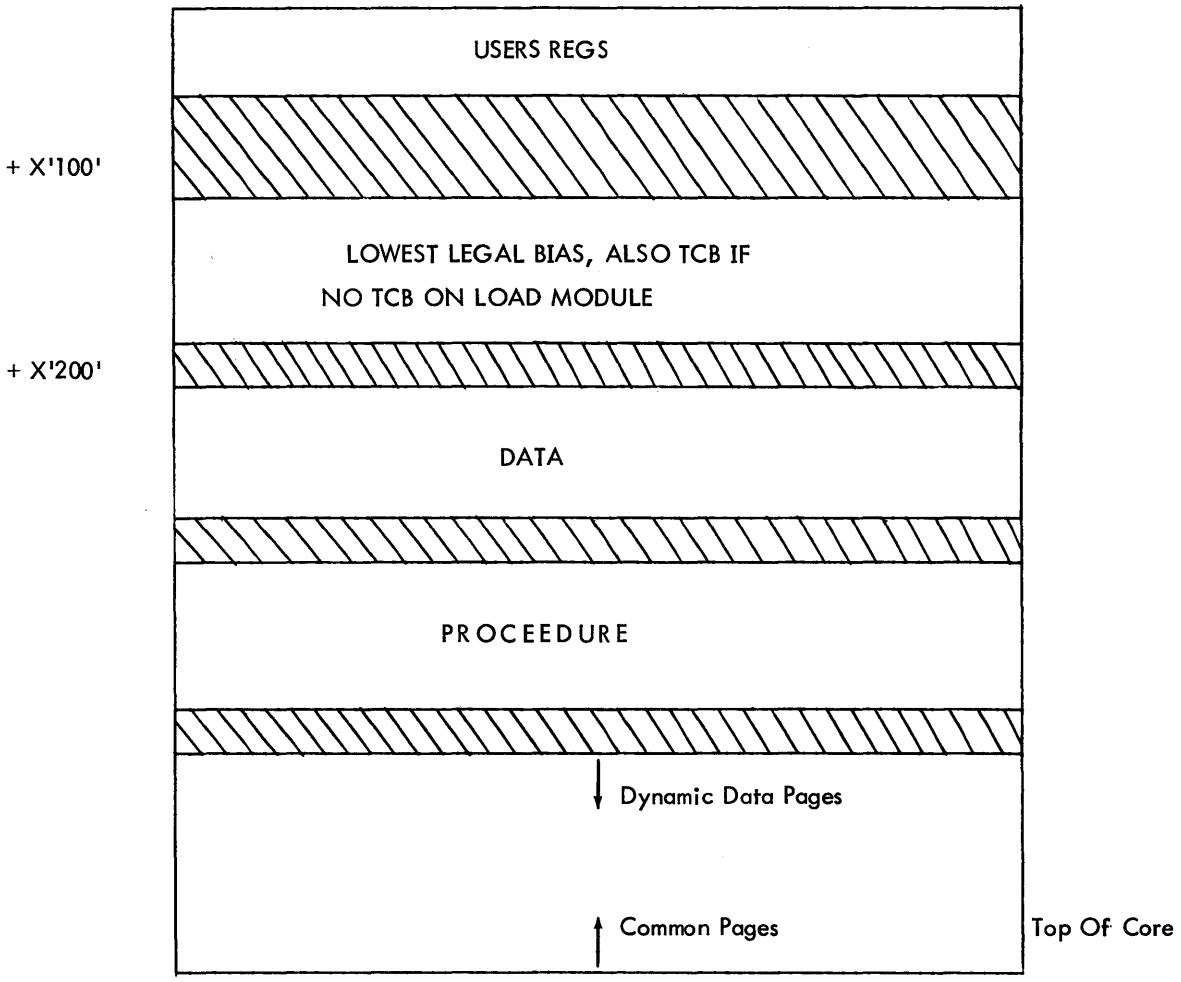


Figure 3-3. USER IN MEMORY

3.3 OPERATIONAL OVERVIEW

The RUN subsystem is entered via a specific request from the user at BTM Executive level (!RUN), trap during user execution under the RUN subsystem, a specific user request (BREAKPOINTS), and monitor aborts during user execution.

Upon initial entry (from the BTM Executive) the RUN subsystem prompts the user with, "LOAD MODULE FID:", whereupon the user supplies the name of the load module to be run. RUN then, using the load module named, reads the "HEAD" and "TREE" from the load module, re-biases the load module, applies any !MODIFY commands to the load module(MODTABLE) and describes memory according to Figure 3-3.

After initialization, RUN prompts the user for commands from the terminal. If the user wants to execute his program, he merely types ;P or ;G. RUN the issues a CAL3,5 and user execution takes place. For any instruction breakpoints, CAL3,6's are inserted into the users program and when executed by the running program, control is returned to RUN.

Figure 3-4 depicts the overall execution flow of RUN.

3.4 MODULE ANALYSIS

3.4.1 ENTRY

1. Module Name:
   ENTRY
2. Purpose:
   a. Obtain load module name.
   b. Re-bias the load module into the BTM user area.
   c. Build a TCB if none exits.
   d. Set swap size.
3. Entry:
   Entered from the BTM Executive.
4. Exit:
   a. Exits to DEBUG if the load module loaded.
   b. Exits to LMNER if the load module is not a proper load module.
   c. Exits to LMNAB if the load module does not exist.
   d. Exits to LMSZER if the load module is too big or biased improperly.
5. Operation:
   a. Request load module FID.
   b. Read users terminal (GETTXT) for load module name, account, and password.
   c. Obtain all of core and set swap size to all of core.
   d. Read the "HEAD" and "TREE" records from the load module.
   e. Re-bias the load module and set OOSIZ and OISIZ.
   f. Read the root (RDSEG) and REFDEF stack (SYMS).
   g. Set up the "GO" address (HEAD+1) and the TCB (CAL3,8).
   h. Describes memory and swap size.
   i. Go to "DEBUG".

3.4.2 GOTOUSER

1. Module Name:
   GOTOUSER
2. Purpose:
   Transfer control from "RUN" (level 1) to the users program (level 2).
3. Entry:
   a. Entered from GOPRO from a ;G or ; P.

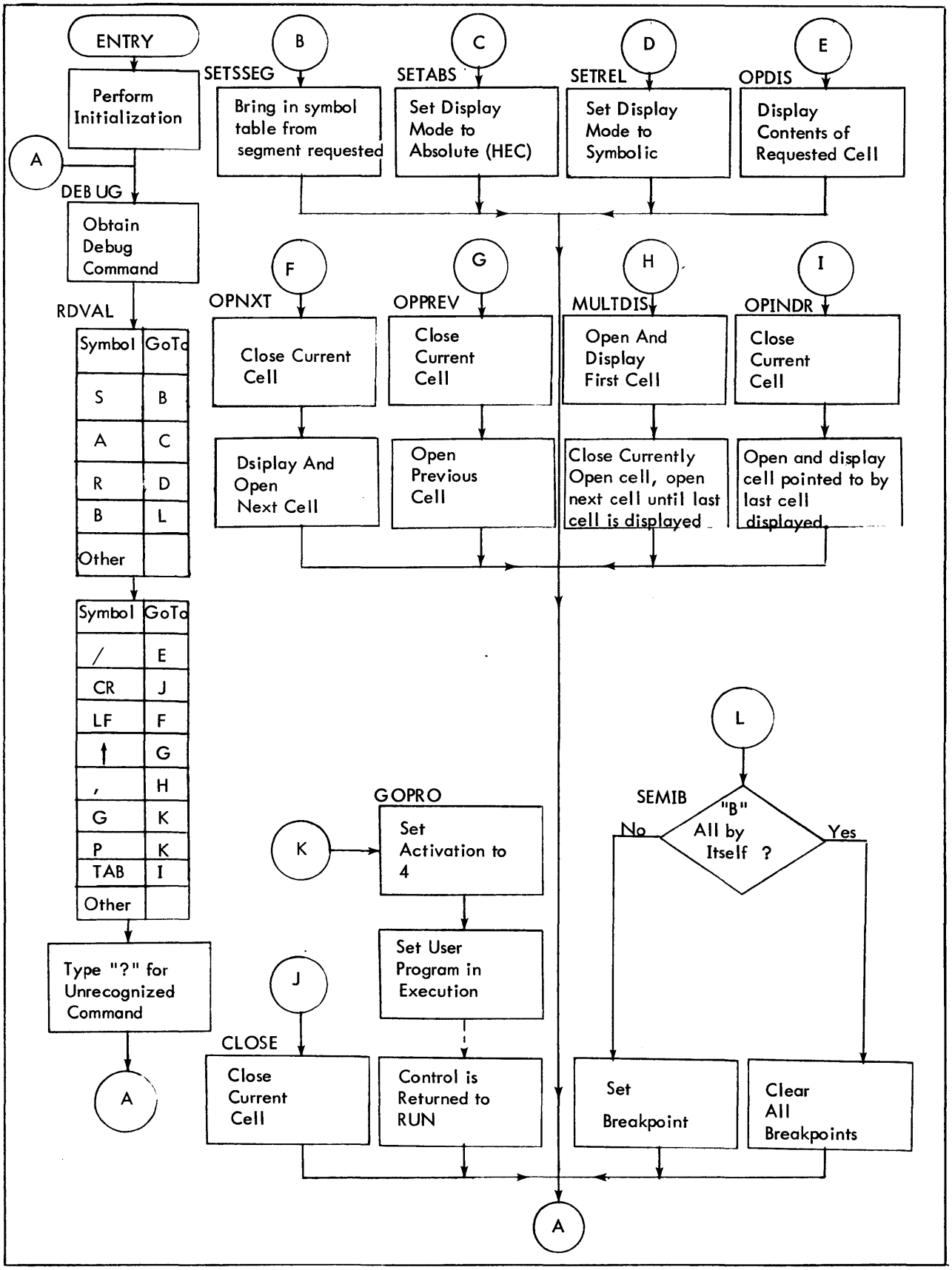


Figure 3-4. Operational Overview of RUN

b. Entered from CALRET after CAL1 simulation.

c. Entered from SETGOAD upon an overlay call.

4. Exit:
Transfers control to users program via GOADR.

5. Operation:

a. Set the swap size (SETUSZ) for the users environment.

b. Write out the current window page (WRPG) in case there were modifications.

c. Set activation to four (activate on CR/LF only).

d. Enter the user at the address specified by GOADR (CAL3,5).

3.4.3 TLOC

1. Module Name:

TLOC

2. Purpose:
Analyze and either give control to the user after typing a message or provide service, for any "POP UP" level.

3. Entry:
Entered from the user upon

1. A break in his program or (esc) - (esc).

2. Trap fault.

4. Exit:
Exits to appropriate control routine applicable to the debug mode.

5. Operation:

a. Obtain the PSD from the "POP UP" (CAL3,9) and store as GOADR.

b. Type message to user (TYM) informing him of the reason for the "POP UP".

c. Read the current symbol table for the segment in (SYMS) in case of debugging.

d. Prompt the user for input with a "BELL" and wait for a debug command.

3.4.4 CHKBRK

1. Module Name:
CHKBRK

2. Purpose:
Type the break number associated with the break which caused the "POP UP".

3. Entry:
Entered from TLOC ("POP UP") upon detection that the "POP UP" was due to a breakpoint.

4. Exit:
Exits to TLOC to enter the debug mode.

5. Operation:

a. Change all CAL3,6's (breakpoints) back to the original contents (RDWNOB).

b. Type the location that the break occurred.

c. Enter the debug mode (TLOC).

3.4.5 OPDIS

1. Module Name:
   OPDIS
2. Purpose:
   Display the contents of a location (/).
3. Entry:
   Entered from DEBUG upon detection of a "/".
4. Exit:
   To PROMPT for another debugging command.
5. Operation:
   a. Set REG2 with the value (VALUE) of the cell desired.
   b. BAL to OPEN to open the cell and display the contents.
   c. Return to PROMPT for another debug command.

3.4.6 OPEN

1. Module Name:
   OPEN
2. Purpose:
   Open and display the contents of a cell specified as "e /".
3. Entry:
   Entered from
   a. OPDISI to display one cell.
   b. OPNXT upon detection of a line feed.
   c. MULTDIS for multiple cell displays.
4. Exit:
   B 0,4
5. Operation:
   a. Check for address error or register display (CHKPTRNX).
   b. Type the address (HEXOUT) being displayed if requested (REG 7 greater or equal to zero).
   c. Read the memory address into WINDOW (READWD).
   d. Type the contents (HEXOUT) of the desired memory location.
   e. Return - B 0,4.

3.4.7 CLOSE

1. Module Name:
   CLOSE
2. Purpose:
   Alter the specified memory location.
3. Entry:
   Entered from DEBUG upon detection of a (CR).
4. Exit:
   Back to PROMPT for another debug command.
5. Operation:
   a. Check for a value typed by the user (CCHAR).

b. If nothing typed prior to the (CR), do not alter the location (CLOPRM).

c. Store the new value in the memory cell requested (STORWD).

d. Return to PROMPT for another debug command.

3.4.8 OPNXT

1. Module Name:
   OPNXT
2. Purpose:
   Store the value in the currently open cell and display the contents of the next one.
3. Entry:
   Entered from DEBUG upon detection of a (LF).
4. Exit:
   Back to PROMPT for another debug command.
5. Operation:
   a. Check if modification is being made to the currently open cell (CCHAR).
   b. If so, store the new value (STORWD) and; if not:
   c. Open the next cell (OPEN).
   d. Return to PROMPT for another debug command.

3.4.9 OPPREV

1. Module Name:
   OPPREV
2. Purpose:
   Display and open the memory location previous to the currently open cell.
3. Entry:
   Entered from DEBUG upon detection or a " ↑ ".
4. Exit:
   Returns to PROMPT for another debug command.
5. Operation:
   a. Check if modification is being made to the currently open cell (CCHAR).
   b. If so, store the new value (STORWD) and; if not;
   c. Get the address of the currently open cell (LOC) and decrement it.
   d. BAL to OPEN
   e. Return to PROMPT.

3.4.10 OPINDR

1. Module Name:
   OPINDR
2. Purpose:
   Display and open the memory cell pointed to by the address field of the currently open cell.
3. Entry:
   Entered from DEBUG upon detection of a "ESCI" (TAB)
4. Exit:
   Returns to PROMPT for another debug command.

5. Operation:
   a. Check if modification is being made to the currently open cell (CCHAR).
   b. If so, store the new value (STORWD) and; if not;
   c. Obtain in REG 2 the address field of the currently open cell (QUANT).
   d. Open the cell pointed to by QUANT (BAL,4 OPEN).
   e. Return to PROMPT for another debug command.

3.4.11 EHEH

1. Module Name:
   EHEH
2. Purpose:
   Type a "?" to the user, and run down his input strean in case of an error.
3. Entry:
   Entered from CLOSE, OPNXT, OPPREV, OPINDR, MULTDIS, SEM:B, TRYC, CMPSEG when either the core location being displayed or the new value typed by the user is in error.
4. Exit:
   To PROMPT for another debug command.
5. Operation:
   a. Type a carriage return-line feed to the user (TYCRLF).
   b. Type a question mark.
   c. Scan down input looking for an activation character.
   d. Return to PROMPT.

3.4.12 MULTDIS

1. Module Name:
   MULTDIS
2. Purpose:
   Display contents "from" a starting location " to" an ending location.
3. Entry:
   Entered from DEBUG upon detection of a ",".
4. Exit:
   Back to PROMPT for another debug command.
5. Operation:
   a. Get "from" address (VALUE) and store in DOT.
   b. Read "to" address (RDVAL) and set in TOLOC.
   c. Open and type (OPEN) "from" address (DOT).
   d. Bump from address (DOT) and type contents (OPEN) until equal to TOLOC.
   e. Return to PROMPT for another debug command.

3.4.13 GOPRO

1. Module Name:
   GOPRO
2. Purpose:
   a. Go back to user at location specified by ;G or ;P.
   b. Go back to user, executing the replaced breakpoint instruction, specified by ;G or ;P.
3. Entry:
   Entered from DEBUG upon detection of a ;G or ;P.
4. Exit:
   Exits to user via GOTOUSER.
5. Operation:
   a. Run down input until a carriage return is detected.
   b. If user specified an address (VALUE), store into GOPOR and return to GOTOUSER.
   c. If no VALUE to start execution at, and no breakpoints are set, then return to GOTOUSER.
   d. Set breakpoint number in REG 8, and transfer to CHKEXU (CHKEXU will replace the break point with the actual instruction and start execution at that point).

3.4.14 SEMIB

1. Module Name:
   SEMIB
2. Purpose:
   To set and remove breakpoints in a users program and type the breakpoints.
3. Entry:
   Entered from SEMICK when a "B" is detected.
4. Exit:
   Returns to PROMPT
5. Operation:
   a. Type all breakpoints to user if only ;B (DISBRK).
   b. If VALUE equals zero, then clear all breaks (CLRALL).
   c. Find a free break entry (BREAKS).
   d. Get contents of location to store breakpoint in (READWD).
   e. Store break into location (STORWD).
   f. Store address of breakpoint is break table (BREAKS).
   g. Store replaced instruction (STORWD).
   h. Return to PROMPT.

3.4.15 CLRALL

1. Module Name:
   CLRALL
2. Purpose:
   Clear all breakpoints.
3. Entry:
   Entered from SEMIB if VALUE equals O (O;B).

4. Exit:
   Returns to PROMPT

5. Operation:

   a. Set REG 7 to indicate all breaks to be cleared (REG 7 = 8).

   b. BAL to CLRBRK (Clears breakpoint and restores actual instruction).

   c. Loop on REG 7.

   d. Return to PROMPT.

3.4.16 CLRBRK

1. Module Name:
   CLRBRK

2. Purpose:
   Remove breakpoint from users program. REG 7 equals the breakpoint number.

3. Entry:

   a. Entered from SEMIB if only one breakpoint is to be cleared.

   b. Entered from CLRALL if all breakpoints are to be cleared.

4. Exits:
   B 0,4

5. Operation:

   a. Check break table (BREAKS) to be sure break exists.

      1. If not return (B 0,4).

      2. Continue

   b. Clear break from modtable (MOD BUF) if it exists within the table.

   c. Before removing break, check to be sure that the break is in memory (RDWNOB) and not in any overlay.

   d. Get the location of the word to be replaced (READWD).

   e. Zap break entry in break table (BREAKS).

   f. Replace breakpoint (CAL3,6) with actual instruction (STORW1).

   g. Decrement all breakpoint numbers to reflect that one has been removed.

   h. Zap the break point number in BREAKS if the break cleared was the breakpoint which caused the "POP UP".

   i. RETURN B 0,4

3.4.17 DISBRK

1. Module Name:
   DISBRK

2. Purpose:
   Display all breakpoints to the user.

3. Entry:
   Entered from SEMIB if only ";B "typed by user.

4. Exit:
   Returns to PROMPT.

5. Operation:

   a. Get entry out of break table (BREAKS).

   b. If zero, loop on the number of breaks (8).

   c. Type break number (HXNOD) to user in REG 7.

   d. Type the location of the breakpoint (HEXOUT).

   e. Loop on the number of breaks (8).

   f. Return to PROMPT.

3.5 SUBROUTINE ANALYSIS

3.5.1 RDVAL

1. Subroutine Name:
   RDVAL

2. Purpose:
   Read debug commands.

3. Entry:
   Entered by DEBUG

4. Exit:
   To MORECC

5. Operation:

   a. Zap CCHAR (CHARACTER COUNT), VALUE (will contain value typed if not symbolic). SIGN (positive or negative number), NAMTMP (will contain TEXTC name if symbolic name typed), SYMVAL (will contain value of symbolic name (NAMTMP)).

   b. Get a character (CAL3,0).

   c. Check if special (CCR) and go to MORECC if it is.

   d. Pack it for TEXTC name (NAMTMP).

   e. Convert to Hex and loop to "b".

3.5.2 MORECC

1. Subroutine Name:
   MORECC

2. Purpose:
   Set and transfer to appropriate control function for users debug commands.

3. Entry:
   Entered from RDVDL.

4. Exit:
   To appropriate control functions

5. Operation:

   a. Convert number (REG 3) to negative if SIGN is set.

   b. Go to SEMICK if ";" hit.

   c. Get absolute value from a symbolic names (GTSVAL) if SYMVAL set.

   d. Type (HEXOUT) if " = " encountered.

   e. Set debug mode to Hexadecimal (RADIX=16) if "A".

   f. Get next character.

3.5.3 SINCK

1. Subroutine Name:
   SINCK
2. Purpose:
   Set sign of number positive or negative depending upon whether the user requested a negative number or not.
3. Entry:
   Entered from MORECC.
4. Exit:
   B RDVAL2 (in RDVAL).
5. Operation:
   a. Set SIGN equal to a "t" if positive.
   b. Set SIGN to negative "-" if negative.
   c. Return

3.5.4 SEMICK

1. Subroutine Name:
   SEMICK
2. Purpose:
   Transfer to appropriate routines upon detection of a ";" and a command verb (See figure 3-1).
3. Entry:
   Entered from MORECC upon detection of a ";".
4. Exit:
   To appropriate routine requested by the command verb.
5. Operation:
   a. Transfer to SETSSEG if user request S.
   b. Transfer to GTSVAL if a symbolic name was entered.
   c. SETABS if A requests.
   d. SETREL if R requests.
   e. SEMIB if B requests.
   f. Set GOADR if G requested.
   g. Make VALUE positive or negative (REG 1), if SIGN set.
   h. RETURN for next character (RDVAL1).

3.5.5 SETSEG

1. Subroutine Name:
   SETSEG
2. Purpose:
   Select the symbol table from an overlay named (S).
3. Entry:
   Entered from SEMICK upon detection of an S.
4. Exit:
   Return to PROMPT for next debug command.

5. Operation:
   a. Check CCHAR. Zero means root of load module.
   b. Set REG5 to point to the segment name in the tree (TREE).
   c. Set SSEG equal to the segment number.
   d. BAL to SYMS (Brings in symbol table).
   e. Return to PROMPT.

3.5.6 GTSVAL

1. Subroutine Name:
   GTSVAL
2. Purpose:
   Determine the value of a symbolic symbol.
3. Entry:
   a. BAL, 10 GTSVAL; NAMTMP equals none of symbol.
   b. Entered from MORECC, SEMICK
4. Exit:
   B *10; VALUE equals absolute value of symbol.
5. Operation:
   a. Set REG 2 equal to the start of the symbol table (DDAT).
   b. Compare a byte at a time, NAMTMP against the symbols in DDAT.
   c. When found, load value and store in VALUE (-2 words from the name in REF/DEF stack equals the value of the constant).
   d. Return (B *10).

3.5.7 CHKSERV

1. Subroutine Name:
   CHKSERV
2. Purpose:
   Interrnpt and transfer to simulator routines for M:GP, M:FP, M:SMPRT, M:GL, M:GCP, M:FCP, M:TIME, M:SEGLD.
3. Entry:
   Entered from "POP UP" level for non-allowed CAL.
4. Exit:
   Exits to appropriate simulation routine.
5. Operation:
   a. Read the page into WINDOW where the CAL was executed (READWD).
   b. If the CAL was done via an EXU, go to CHKPTR to chain thru the EXU's.
   c. Allow only CAL1's to go thru (PECAL).
   d. And only CAL1, 8.
   e. Determine the fpt address (CHKPTR) and read in the page of users code containing the fpt address (READWD) and set FPTADR to the address of the fpt.
   f. Branch to appropriate simulation routine using byte zero (FPT code) of the fpt.

3.5.8 GP

1. Subroutine Name:
   GP
2. Purpose:
   Simulate a M:GP CAL (Get page).
3. Entry:
   Entered from CHKSERV with an fpt code of X'08'.
4. Exit:
   Returns to GO TO USER.
5. Operation:
   a. Find the number of pages requested (R0).
   b. If the user request more than are available (NUSIZ) give him all that are possible.
   c. Decrements the number of pages available (NUSIZ) and increse the number the user has (DYPG).
   d. Set SR1 (REG 8) of the users (STORWD) to the number of pages obtained.
   e. Compute the address of the start of the area given and store in SR2 (REG 9) (STORWD).
   f. Bump the return address (CALRET) by 1 (GOADR).
   g. Set the condition codes (CCSET).
   h. Return to user.

3.5.9 FP

1. Subroutine Name:
   FP
2. Purpose:
   Simulate a M:FP CAL (Free page).
3. Entry:
   Entered from CHKSERV with an FPT code of X'09'.
4. Exit:
   Returns to CCSET.
5. Operation:
   a. Find the number of pages requested (R0).
   b. If the user tries to release more than he has (DYPG), release none.
   c. Decrement the number of pages the user has (DYPG). and increment the number available (NUSIZ).
   d. Go to CCSET.

3.5.10 SMPRT

1. Subroutine Name:
   SMPRT
2. Purpose:
   Simulate a M:SMPRT CAL (set memory protection).
3. Entry:
   Entered from CHKSERV with an FPT code of X'0A'.
4. Exit:
   Returns to CCSET.

5. Operation:
   a. Set all protection types to 00 (00SIZE + 01SIZE) and zap 01 size.
   b. Read the second word of the FPT (contains protection value and ending address) into WINDOW (READWD).
   c. Convert start address (REG 7) to pages and end address (REG 0) to pages.
   d. Compute all the pages currently in use (CMPG).
   e. If the starting page is in the area and the ending page is also; set CC1 to zero (CCSET).
   f. If the above is not true, set CC1 to a one (CCSET).
   g. NOTE: No memory protection is actually set, since BTM users always run as protection zero.

3.5.11 GL

1. Subroutine Name:
   GL
2. Purpose:
   Simulate a M:GL CAL (GET LIMITS).
3. Entry:
   Entered from CHKSERV with an FPT code of X'0B'.
4. Exit:
   Returns via CCSET.
5. Operation:
   a. Set SR2 (REG 9) equal to the upper memory limit (TOPMEM).
   b. Convert the number of common pages (CMPG) to words.
   c. Compute the difference between top of memory (TOPMEM) and REG 13 (total common pages).
   d. Store in SR1 (REG 8) (STORWD).
   e. Return to CCSET

3.5.12 GCP

1. Subroutine Name:
   GCP
2. Purpose:
   Simulate a M:GCP CAL (Get common pages).
3. Entry:
   Entered from CHKSERV with an FPT code of X'0C'.
4. Exit:
   Returns via CCSET
5. Operation:
   a. Find number of pages requested (R0).
   b. If the user requests more than are available (NUSIZ) give him all that are possible.
   c. Increment the number of common pages the user has (CMPG) and decrease the number of pages available (NUSIZ).
   d. Set SR1 (REG 8) of the user (STORWD), to the number of pages obtained.
   e. Compute the start address of the lowest common page and store in the users SR2 (REG 9) (STORWD).

3.5.13 FCP

1. Subroutine Name:
   FCP
2. Purpose:
   Simulate a M:FCP CAL (Free Common Pages).
3. Entry:
   Entered from CHKSERV with an FPT code of X'OD'.
4. Exit:
   Return via CCSET.
5. Operation:
   a. Obtain the number of pages requested (R0).
   b. If the user tries to release more than he has, release none.
   c. Increment the number of pages the user has (NUSIZ) and decrement the number of common pages (CMPG).
   d. Return to CCSET.

3.5.14 TIME

1. Subroutine Name:
   TIME
2. Purpose:
   Simulate a M:TIME CAL (Dated Time).
3. Entry:
   Entered from CHKSERV with a FPT code of X'10'.
4. Exit:
   To CALRET
5. Operation:
   a. Get address to put date and time (CHKADRNX).
   b. Issue a BTM date and time CAL (CAL3, 15).
   c. Store hours and minutes (STORWD).
   d. Store Month (STORWD).
   e. Store Day (STORWD).
   f. Store Year (STORWD).
   g. Return to CALRET

3.5.15 SEGLD

1. Subroutine Name:
   SEGLD
2. Purpose:
   Simulate an M:SEGLD (Segload).
3. Entry:
   Entered from CHKSERV with an FPT Code of X'01'.
4. Exit:
   a. Transfer to GOTOUSER with the new segment in.
   b. Transfers to PECAL if any error conditions occur.

5. Operation:
   a. Check for indirect or indexed address (CHKADRNX).
   b. Get second word of FPT (READWD) which contains the address of the TEXTC segment name.
   c. Get the segment name (READWD).
   d. Look through TREE for segment name (SEGLOOP).
   e. Set all segments in TREE not in except for the segment requested (SEGLP1) (Backward Path).
   f. Set "Forward Path" still in (SEGLP2).
   g. Read the requested segment in (RDSEG).
   h. Return to FPT+2 (CALRET) or transfer to CHKEXU if BAL.

3.5.16 CHKEXU

1. Subroutine Name:
   CHKEXU
2. Purpose:
   Run down chain of "EXU's" from an "Executed CAL".
3. Entry:
   Entered from SEMIB, and SEGLD.
4. Exit:
   Transfer to GOTOUSER.
5. Operation:
   a. Read into WINDOW the requested page (READWD).
   b. If not an "EXU" (X'67') check for a BAL (CHKBAL).
   c. Skip down "EXU's (CHKPTR) until non "EXU" encountered.
   d. If not BAL, return to user (SETGOAD).
   e. Set replaced instruction into HEAD.
   f. Get new transfer address (CHKPTR).
   g. Swap old GOADR for new GOADR.
   h. Store the new GOADR in users memory.
   i. Return via GOTOUSER.

3.5.17 TYCRLF

1. Subroutine Name:
   TYCRLF
2. Purpose:
   Type a carriage return and line feed to users terminal.
3. Entry:
   BAL,4 TYCRLF
4. Exit:
   B 0,4
5. Operation:
   a. Send a carriage return (X'15') to user (CAL3, 1).
   b. Send a line feed (X'25') to user (CAL3, 1)
   c. Return

3.5.18 TYM

1. Subroutine Name:
   TYM
2. Purpose:
   Type a TEXTC message to the users terminal.
3. Entry:
   BAL,4 TYM; R2 = BYTE ADDRESS OF TEXTC MESSAGE.
4. Exit:
   B 0,4
5. Operation:
   a. Place byte count from message into REG 3.
   b. Send a character at a time (CAL3, 1) from the message until REG 3 (byte count) goes to zero.
   c. Return.

3.5.19 GETTXT

1. Subroutine Name:
   GETTXT
2. Purpose:
   Obtain TEXTC format from user type in.
3. Entry:
   a. Entered from ENTRY for request of "LOAD MODULE FID".
   b. BAL,5 GETTXT; REG 8 equals maximum character length
4. Exit:
   a. B O,5 if TEXTC O.K.
   b. ENTRY1 if an error.
5. Operation:
   a. Set REG 12-14 with blanks (will contain TEXTC name).
   b. Look for delimiters (",", "(",")", "X'15") until either hit or REG8 is exhausted.
   c. Store characters into REGS 12-14 bumping,byte count (byte 0 of REG12) to reflect number of character in REGS 12 - 14.
   d. Type a error message (NERM) if input from user exceeds maximum character length (REG 8).

3.5.20 LMNAB; LMNER; LMSZER

1. Subroutine Name:
   LMNAB; LMNER; LMSZER
2. Purpose:
   Type an appropriate error message to the user if there is a problem with his Load Module.
3. Entry:
   B LMNAB; LMNER; LMSZER
4. Exit:
   Returns to ENTRY for another Load Module Name from user.
5. Operation:
   a. Type appropriate error message (TYM).
   b. Return for another request (ENTRY).

3.5.21 RDSEG

1. Subroutine Name:
   RDSEG
2. Purpose:
   Obtain an overlay segment by segment number.
3. Entry:
   BAL,9 RDSEG; REG 7 equals segment number.
4. Exit:
   B *9
5. Operation:
   a. Check to see if segment is currently in (X'40000000 set in TREE).
   b. Set swap size equal to users size (USIZ).
   c. Set symbol table not in (SSEG to zero).
   d. Bring the users memory into the subsystem memory (CAL3,7).
   e. Read the overlay in (RDRCD1).
   f. Set all DCB's to HERE (5) and reallocate them.
   g. Apply any !MODIFY's to the overlay (MODBUF).
   h. Set the new swap size (NOROT1).
   i. Re-align usersize (USIZ).
   j. Return B*9

3.5.22 SETSWAP

1. Subroutine Name:
   SETSWAP
2. Purpose:
   Set up swap arguments for Swap description.
3. Entry:
   BAL,11 SETSWAP
4. Exit:
   B *11
5. Operation:
   a. Set right half of REG 1 to the start address of swap.
   b. Set into the left half of REG 1, the number of pages (00SIZ+)01SIZ).
   c. Set into REG 0, the page number of user level swap storage.
   d. Return B *11

3.5.23 SYMS

1. Subroutine Name:
   SYMS
2. Purpose:
   Routine to bring in the Symbol Table of the current segment in core.
3. Entry:
   BAL,9 SYMS
4. Exit:
   B *9

5. Operation:
   a. Set segment names in (SSEG).
   b. Get the segment requested (RDRCD).
   c. Convert all names to word resolution.
   d. Add appropriate bias (BIAS).
   e. Return B *9.

3.5.24 RELOCR

1. Subroutine Name:
   RELOCR
2. Purpose:
   Relocate a segment to fit current bias.
3. Entry:
   B RELOCR
4. Exit:
   B *10
5. Operation:
   a. Read the relocation dictionary (RDRCD).
   b. Put it in memory below WINDOW.
   c. Convert to word resolution and add appropriate bias (BIAS).
   d. Store it back in relocation dictionary.
   e. Return.

3.5.25 RDWNOB

1. Subroutine Name:
   RDWNOB
2. Purpose:
   Set break table inactive to avoid breaks while obtaining a page of user memory.
3. Entry:
   BAL,9 RDWNOB
4. Exit:
   To READWD
5. Operation:
   a. Set all break locations (BREAKS), negative.
   b. Fall through to READWD.

3.5.26 READWD

1. Subroutine Name:
   READWD
2. Purpose:
   Obtain a word from users memory.
3. Entry:
   BAL,9 READWD; REG 8 = address

4. Exit:
   B *9; REG 0 = Word.
5. Operation:
   a. Get the page of memory (GPAGIN) which contains the requested word.
   b. Set the requested word in REG 0.
   c. Set all breaks active (Zap MINUS sign of BREAKS).
   d. Return

3.5.27 STORWD

1. Subroutine Name:
   STORWD
2. Purpose:
   Put a word into users memory.
3. Entry:
   BAL,9 STORWD REG 8 = Address to put word; REG 0 = Word.
4. Exit:
   B *9
5. Operation:
   a. Check if ROOT in (if root, merely store).
   b. Find size and address of segment (STORWL).
   c. Get the page needed (GPAGIN).
   d. Store into the page.
   e. Set the write flag (WRIND) and return.

3.5.28 GPAGIN

1. Subroutine Name:
   GPAGIN
2. Purpose:
   Obtain the page of core, to which a memory address pertains.
3. Entry:
   BAL,12 GPAGIN; REG 8 = address
4. Exit:
   B *12 REG 5 equals address of location in WINDOW.
5. Operation:
   a. Compute address of page needed for memory location desired.
   b. If changing a breakpoint (BREAKS), set REG5 to point to break table.
   c. Check if the page is currently in window (PGIN) and if so return (B *12).
   d. Write out current page (in case of modification) and read in page containing requested address.
   e. Return.

3.5.29 WRPG

1. Subroutine Name:
   WRPG
2. Purpose:
   Write a page of subsystem memory to user level swap storage.
3. Entry:
   BAL,11 WRPG
4. Exit:
   B *11
5. Operation:
   a. Set REG 0 to page number of user level swap storage (PGIN).
   b. Set REG 1 pointing to WINDOW (WINDPG).
   c. Set REG 2 positive (1).
   d. Issue a swap (CAL3,7).
   e. Return B *11

3.5.30 RDPG

1. Subroutine Name:
   RDPG
2. Purpose:
   Read a page of user memory into WINDOW.
3. Entry:
   BAL,11 RDPG; REG 4 equals page number.
4. Exit:
   B *11
5. Operation:
   a. Set REG 0 with page number (REG 4).
   b. Set subsystem page to read into (WINDPG) REG 1.
   c. Set REG 2 negative.
   d. Issue swap (CAL3,7).
   e. Return (B *11)

3.5.31 CHKADRNX; CHKPTRNX; CHKPTR

1. Subroutine Name:
   CHKADRNX; CHKPTRNX; CHKPTR
2. Purpose:
   Decode indirect address or registers from a user FPT.
3. Entry:
   BAL,1 CHKADRNX; CHKPTRNX, CHKPTR; REG 2 = address of FPT.
4. Exit:
   B *1
5. Operation:
   a. Check for indirect bit in REG 2 and transfer to NOAST if not.

b. Check if address is out of core (NOMLIM) and transfer indirect to ADER if it is.

c. Read the address specified by the FPT address into the window page (READWD).

d. Transfer to NOINDX if not indexed (NOAST).

e. Read the register requested (READWD) and add to the address of the FPT.

f. Set in REG 3 the effective address (NOINDX).

g. Return

3.5.32 HEXOUT

1. Subroutine Name:
   HEXOUT
2. Purpose:
   Convert a number to hexadecimal and type it to the user.
3. Entry:
   BAL,1 HEXOUT; REG 3 = Number
4. Exit:
   B *1
5. Operation:
   a. Transfer to SYMBOUT if ABSFLAG not set.
   b. Output a "." (CAL3,1) (NOSYMB)
   c. Convert number to hex (HEXCHR) and store in LMN until REG 3 exhausted.
   d. Type from LMN to users terminal (CAL3,1)
   e. Return

3.5.33 SYMBOUT

1. Subroutine Name:
   SYMBOUT
2. Purpose:
   Convert a number to its symbolic name plus a hexadecimal displacement.
3. Entry:
   B SYMBOUT
4. Exit:
   B SEIFS
5. Operation:
   a. Look for symbol in dynamic data (DDAT).
   b. Branch to SEIFS to print on the user terminal the symbolic name.

3.5.34 SEIFS

1. Subroutine Name:
   SEIFS
2. Purpose:
   Type a symbolic name to the users terminal
3. Entry:
   B SEIFS; REG 6 = address of name.

4. Exit:
   B NOSYMB
5. Operation:
   a. Get byte address of name into REG6.
   b. Get byte count of name into REG 2.
   c. Type name to user (CAL3,1) until byte count (REG2) is exhausted.
   d. Type a "+" (CAL3, 1) to user for hexadecimal offset from symbol.
   e. Branch to NOSYMB to type hexadecimal offset.

4.0 BPM SUBSYSTEM

4.1 FUNCTIONAL OVERVIEW

4.1.1 Purpose

BPM (Terminal Batch Entry) is a BTM subsystem which provide the ability for an on-line user to insert a job file, status check a previously inserted job file, or delete a previously inserted job file. This subsystem is available only when BTM is operating in a symbiont BPM environment.

4.1.2 Input

The BPM Subsystem prompts the BTM user immediately after entry with

1. INSERT Job? $\{^{Y}_{N}\}$. Prepares the subsystem to enter a job file.
   a. An affirmative reply (Y) will cause the subsystem to read the job file from the users terminal or from a previously ASSIGNed file built, for example, by EDIT. The subsystem then prompts with the command "EDIT?"
   b. A negative reply will cause the subsystem to prompt with the next control command. (STATUS CHECK) .
2. STATUS CHECK? $\{^{Y}_{N}\}$. Allows the user to check the status of any previously inserted job file.
   a. An affirmative reply (Y) will cause the subsystem to prompt with "ID=" at which time, the user responds with four hexadecinal digits of a previously inserted job id. If only a carriage return is entered, the subsystem returns to the Executive level (!).

      After receiving a valid job id the subsystem will return one of the following five status indicators.

      RUNNING -The job requested is currently execution in the Batch Partition.
      COMPLETED -The job requested has been processed.
      NONEXISTENT -The job requested has never been entered into the Batch Queue.
      WAITING ON OUTPUT -The job requested has been processed but the output has not been completed.
      WAITING: N AHEAD CURRENT ID: X -The job requested is still waiting to be processed. N is the number ahead of the job requested and X is the ID of the job that is currently executing in the Batch Partition.

      The subsystem then prompts again with "ID=".

   b. A negative reply will cause the subsystem to prompt with the next control command.(Delete Job)
3. DELETE JOB? $\{^{Y}_{N}\}$. Allows the user to delete from the Batch Queue, a previously inserted job file.

   An affirmative reply (Y), will cause the subsystem to prompt with "ID=" at which time, the user responds with four hexadecimal digits of a previously inserted job ID. The subsystem will respond with one of the five following messages from a delete request.

   NONEXISTENT - The job requested has never been entered into the batch queue.

   ...DELETED - The job file was deleted from the batch queue.

   UNABLE TO DELETE JOB. AUTHORIZATION REQUIRED.
   - User is trying to delete a job other than his own.

UNABLE TO DELETE JOB. TOO LATE. - The requested job has already been run in the batch partition.

UNABLE TO DELETE JOB. NON-SYMBIONT SYSTEM. - There is no Batch Job queue or a non-symbiont syst.

The subsystem then prompts again with "ID=". If only a carriage return is entered, the subsystem returns to the Executive level (!).

b. A negative response will cause the subsystem to return to the Executive level (!).

4. EDIT? {Y / N}. Allows the user to edit his job file prior to the job insertion.

a. If the user reply is negative (N), the subsystem enters the job file in the Batch Job queue and types the following to the user

JOB INSERTED. ID = XXXX

where XXXX is the hexadecimal characters of his job ID. Then the subsystem responds with the next control command (STATUS CHECK).

b. If the user reply is affirmative (Y), the subsystem prompts with a question mark (?). At this point the user responds as follows, where XX and YY are record numbers as displayed on the console.

T (RET)	Type file
TXX[,YY](RET)	Type lines XX [Thru,YY]
X (RET)	Erase the job file.
G (RET)	Go (insert the job file)
S XX,YY (RET)	Swap lines XX and YY
D XX [,YY] (RET)	Delete lines XX [Thru,YY]
A XX (RET)	Add the following lines after line XX
R XX [,YY]	Replace XX [Thru,YY] with the following lines.

4.1.3 Error Messages

1. INVALID ID - the response to the "ID=" request was either greater than four characters, or, an illegal configuration of hexadecimal characters.

2. BAD I/O. ABNORMAL CODE - YY - An abnormal was returned from the BPM subsystem reading its input from M:SI DCB. YY equals the error code.

3. File too large. Task ABORTED - The job file to be inserted must fit into the BTM partition. This allows in a 16K user area, approximately 700 records.

4. NONEXISTENT LINE - An edit command refers to a non-existent line.

5. ERRONEOUS COMMAND IGNORED - An edit command is either illegal or contains improper parameters.

6. NO INPUT DATA. BYE - There is no job file is send to the Batch job queue.

7. FIN CARD IGNORED - A !FIN card was found in the Job file. It is deleted from the job file.

8. JOB CARD MISSING OR RECORDS OUT OF ORDER - A !JOB command must be the first record of a Job file.

9. IMPROPER JOB ACCOUNT - The user account on the !JOB command does not match his log-in account.

10. IMPROPER JOB NAME - the users name on his !JOB command does not match his log-in name.

11. IMPROPER JOB PRIORITY - Either the user has an illegal hexadecimal character on his !JOB command, or, he has specified a priority greater than his authorized maximum.

12. IMPROPER EXTENDED ACCOUNTING - The extended accounting field from the !JOB command does not conform to the rules.

13. IMPROPER JOB CARD - the !JOB command does not conform to the rules of a proper job card.

4.2 INTERFACES

4.2.1 Operating System

The BPM subsystem only interfaces with the BTM Executive. It cannot be run as a Batch processor.

4.3 OPERATIONAL OVERVIEW

4.3.1 Description

Processing starts at location BEGIN. The users log-in account and name, terminal number, authorization flags, and maximum priority are stored at this time. If the user is not authorized, he is returned to the BTM Executive. Options are processed and control is given to the appropriate routine; INSERT, STATUS, DELETE. Any abnormal returns from any I/O commands, cause the subsystem to abort through the Abnormal-Exit routines. Any exit condition found within a control routine is exited thru the normal exit routine .

4.3.2 DCB's

1. M:SI - Reads the job file
2. M:EO - Send the job file to the Symbiont

4.3.3 Flowchart

See Figure 4-1

4.4 MODULE ANALYSIS

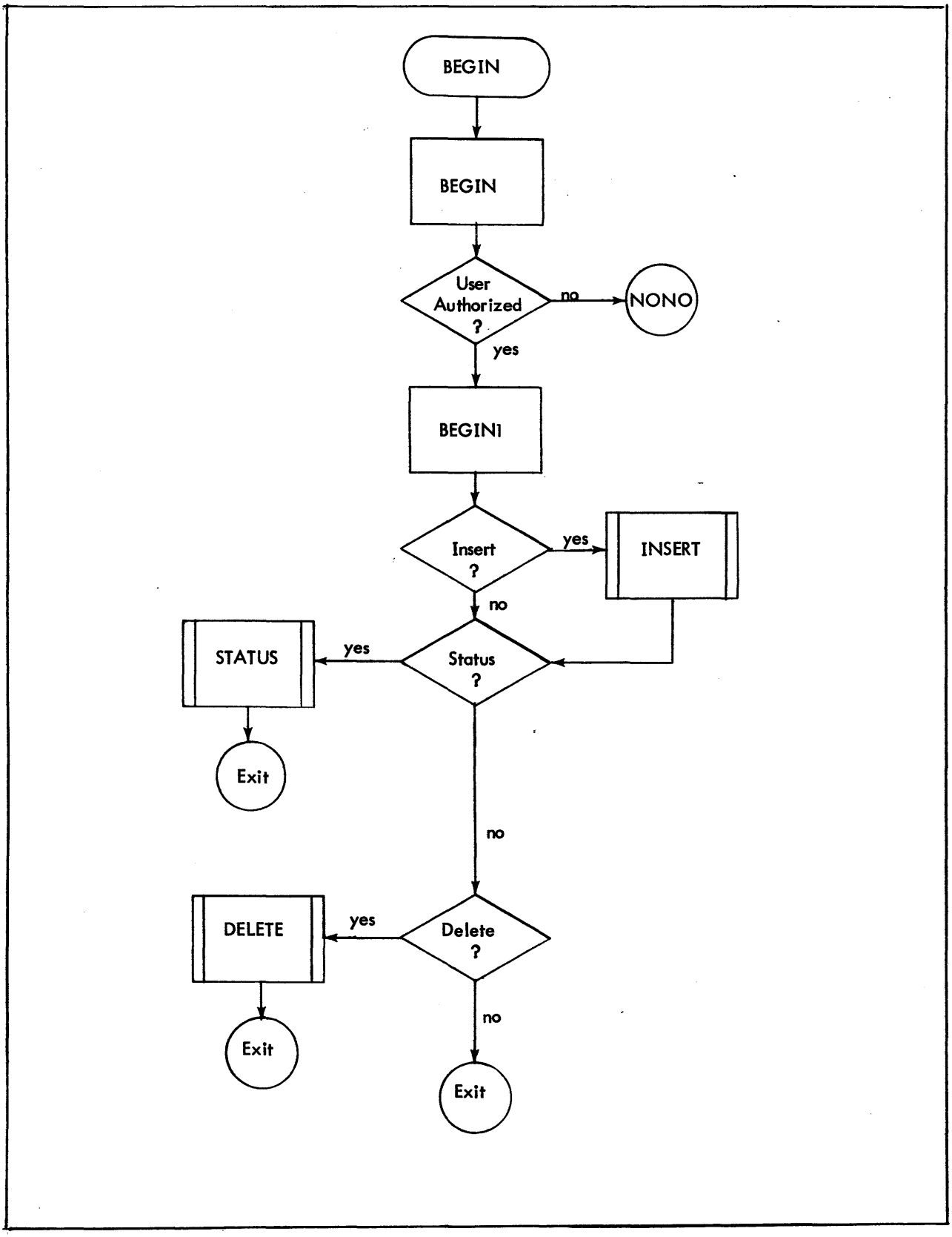


Figure 4-1. Operational Overview of BPM Subsystem

4.4.1 BEGIN

1. Module Name
   BEGIN
2. Purpose:
   Request option from user and transfer to appropriate routine.
3. Entry:
   BEGIN is the entry point into the subsystem, whenever BPM is invoked.
4. Exit:
   a. Transfers control to the appropriate routine requested by the user.
   b. Exits to the time-sharing executive when all options have been exhausted.
5. Operation:
   a. Store the users log-in account (REGS 4,5), name (REGS 13,14,15) and maximum priority (REG 2).
   b. Abort the user (NONO) if batch flag not set (REG 3 greater or equal to zero).
   c. Invoke options to user and request a "Y" or "N" (YESNO). If "Y" transfer to option; if "N" invoke next option.
   d. Return to the time-sharing executive when all options have been invoked.

4.4.2 DELETE

1. Module Name:
   DELETE
2. Purpose:
   Allow the user to delete from the batch job queue, a previously inserted job file.
3. Entry:
   Entered from BEGIN upon a positive reply (Y) to "DELETE JOB?"
4. Exit:
   The exit is to the time-sharing executive upon a carriage return only to an ID request.
5. Operation:
   a. Request the ID of the job file to be deleted (INEBCHEX)
   b. Store the ID in DELID and issue an M:JOB delete cal.
   c. Type "...DELETED" if the delete was successful and issue another ID request.
   d. If the delete was unsucessful, control is transferred to ABNDEL.
      a. An appropriate error message will be typed (see ABNDEL).
      b. Issue another ID request.
6. Flowchart:
   See figure 4-2.

4.4.3 STATUS

1. Module Name:
   STATUS
2. Purpose:
   Allow the user to status check a previously inserted job file.
3. Entry:
   Entered from BEGIN upon a positive reply (Y) to "STATUS CHECK?"

4. Exit:
   The exit is to the time-sharing executive upon a carriage return only, to an ID request.

5. Operation:

   a. Request the ID of the job file to be status checked (INEBCHEX).

   b. Issue an M:JOB status request using the ID in REG 8.

   c. Using the number returned in REG 8 (0 = completed; 1 = running; 2 = waiting to run; 3 = non-existent; 4 = Waiting to output) index into STATMS for the appropriate status message which is then typed to the user (TYPEMESS).

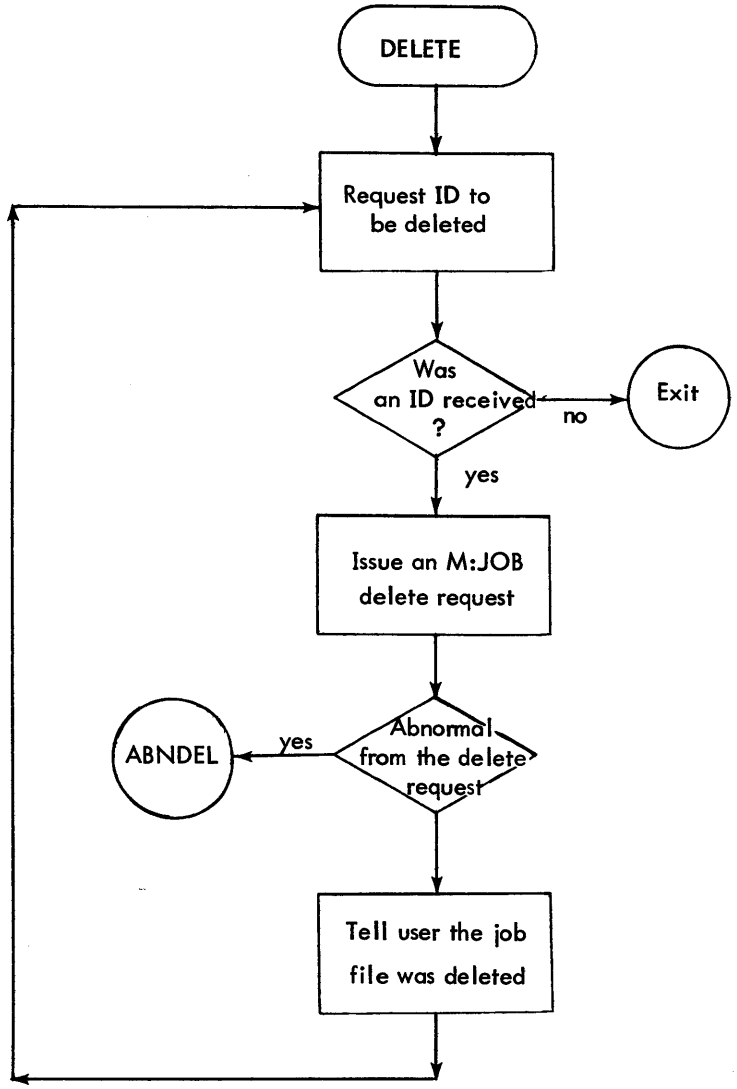


Figure 4-2. Flow Chart for DELETE

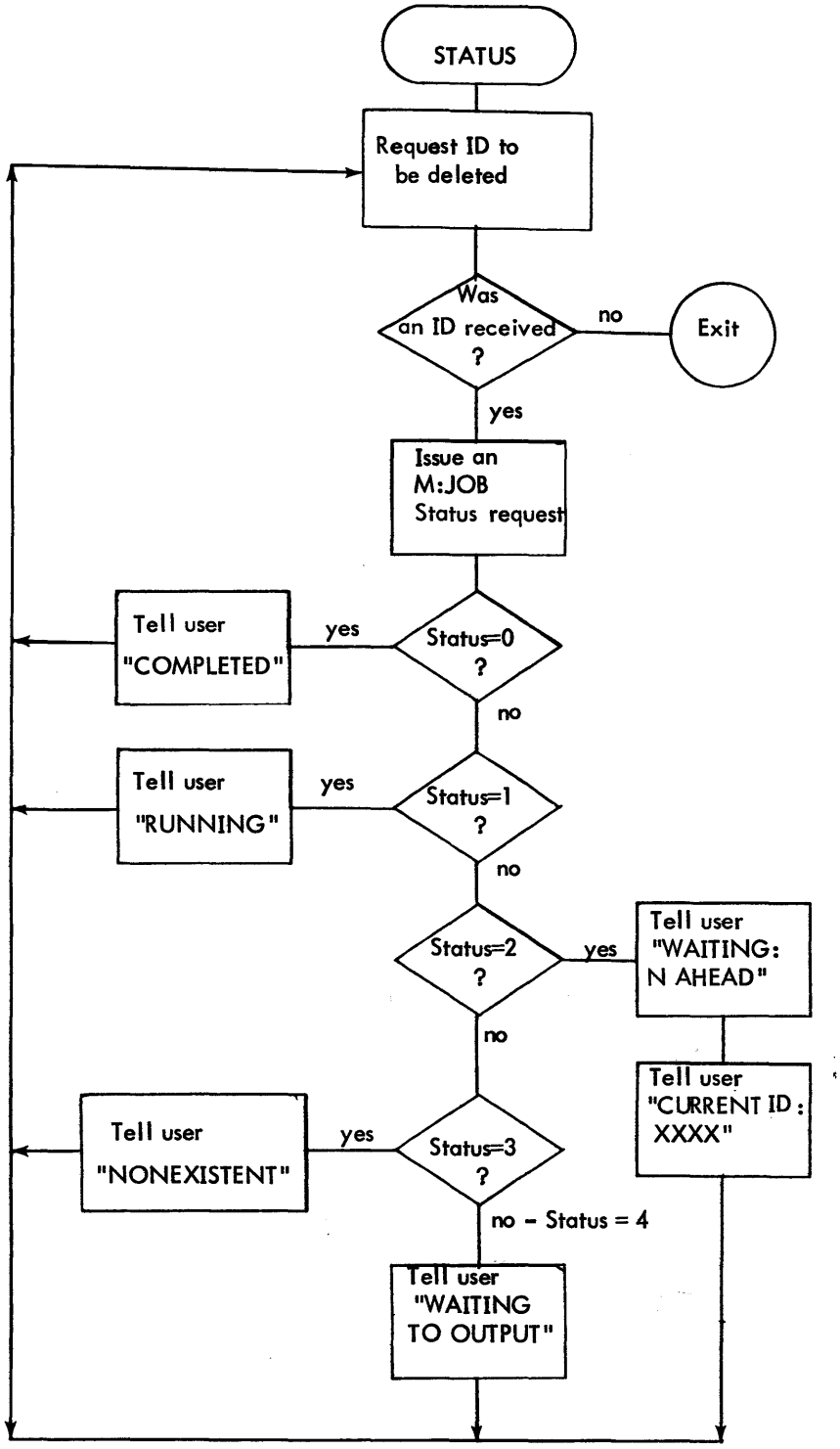


Figure 4-3. Flow Chart for STATUS

4.4.4 INSERT

1. Module Name:
   INSERT
2. Purpose:
   Allow a user to enter a job file into the batch job queue.
3. Entry:
   Entered from BEGIN upon a positive reply (Y) to "INSERT JOB?"
4. Exit:
   Control is transferred to EDIT upon detection of a end of file (from file input) or a null line to an input request (carriage return only).
5. Operation:
   a. Type maximum priority allowed (MAXPRI) to the user (TYPEMESS).
   b. Find size of user area (CAL3,14) and store in MAXPAGES.
   c. Start data buffer at a page boundary behind the subsystem (STRTDATA, NEXTPAGE, FIRSPAGE).
   d. Open M:SI and determine if it is from a users console or file .
   e. If file, read data records until end of file (ABNSI).
   f. If console, type the number (LINENO), and read input until a null line (carriage return only).
   g. Place data records into the buffer pointed to by STRTDATA (PUTLINE).
6. Flowchart
   See Figure 4-5.

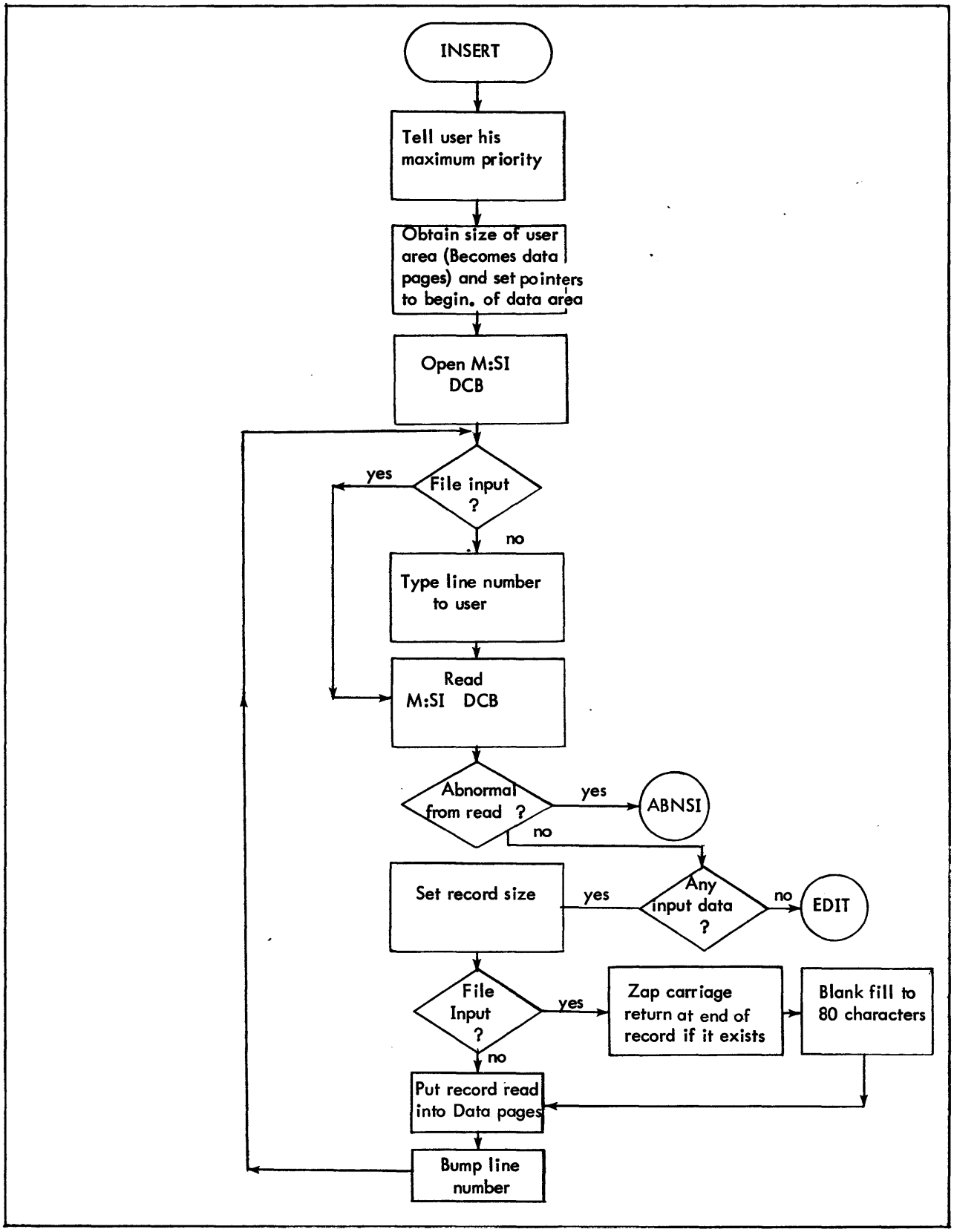


Figure 4-4. Flow Chart of INSERT

4.4.5 FILETYPE

1. Module Name:
   FILETYPE
2. Purpose:
   Delete carriage returns from lines of data read from a file and blank fill records to 80 characters.
3. Entry:
   Entered from INSERT whenever a record is read from file input with REG 1 equally the number of characters with the record.
4. Exit:
   Transfers to INSERT to read the next control record.
5. Operation:
   a. Check REG 1 for greater than 80 characters and truncate if so.
   b. Zap last character of record to a blank, if the last character is a carriage return (X'15').
   c. Blank fill record to 80 characters.

4.4.6 EDIT

1. Module Name:
   EDIT
2. Purpose:
   Allow the user to modify his job file, prior to insertion into the batch job queue.
3. Entry:
   a. Entered from INSERT, when a null record is received as input
   b. Entered from ABNSI upon detection of an End of File from file input.
   c. Entered from TRANSFER if no job card was found in the job file.
4. Exit:
   a. Exits to TRANSFER upon a negative response (N) to "EDIT?"
   b. Exit to EDITO upon a positive reply (Y)
5. Operation:
   a. Ask the user if editing is desired (TYPEMESS)
   b. Transfer to EDITO if desired (YESNO).
6. Flowchart
   See Figure 4-5.

4.4.7 EDITO

1. Module Name:
   EDITO
2. Purpose:
   Request edit commands from user
3. Entry:
   a. Entered from EDIT upon a positive response (Y) to "EDIT?"

b. Entered from APPEND, SCRATCH, TYPE, REPLACE, SWAP, DELETEE from:
   1. An erroneous command format.
   2. Normal return for more edit commands.

4. Exit:
   a. To the time-sharing executive if no input exists in the job file (NOINPT).
   b. To TRANSFER upon the edit command "GO".
5. Operation:
   a. Check for any Job file (LINENO) in the data area and exit if none (NOINPT).
   b. Request edit function.
   c. Branch to function desired (FINDEDIT).
   d. Run down input look for a carriage return if function is erroneous.
   e. Type to user (TYPEMESS) that edit function was erroneous.
   f. Request another edit command.
6. Flowchart

   See Figure 4-5.

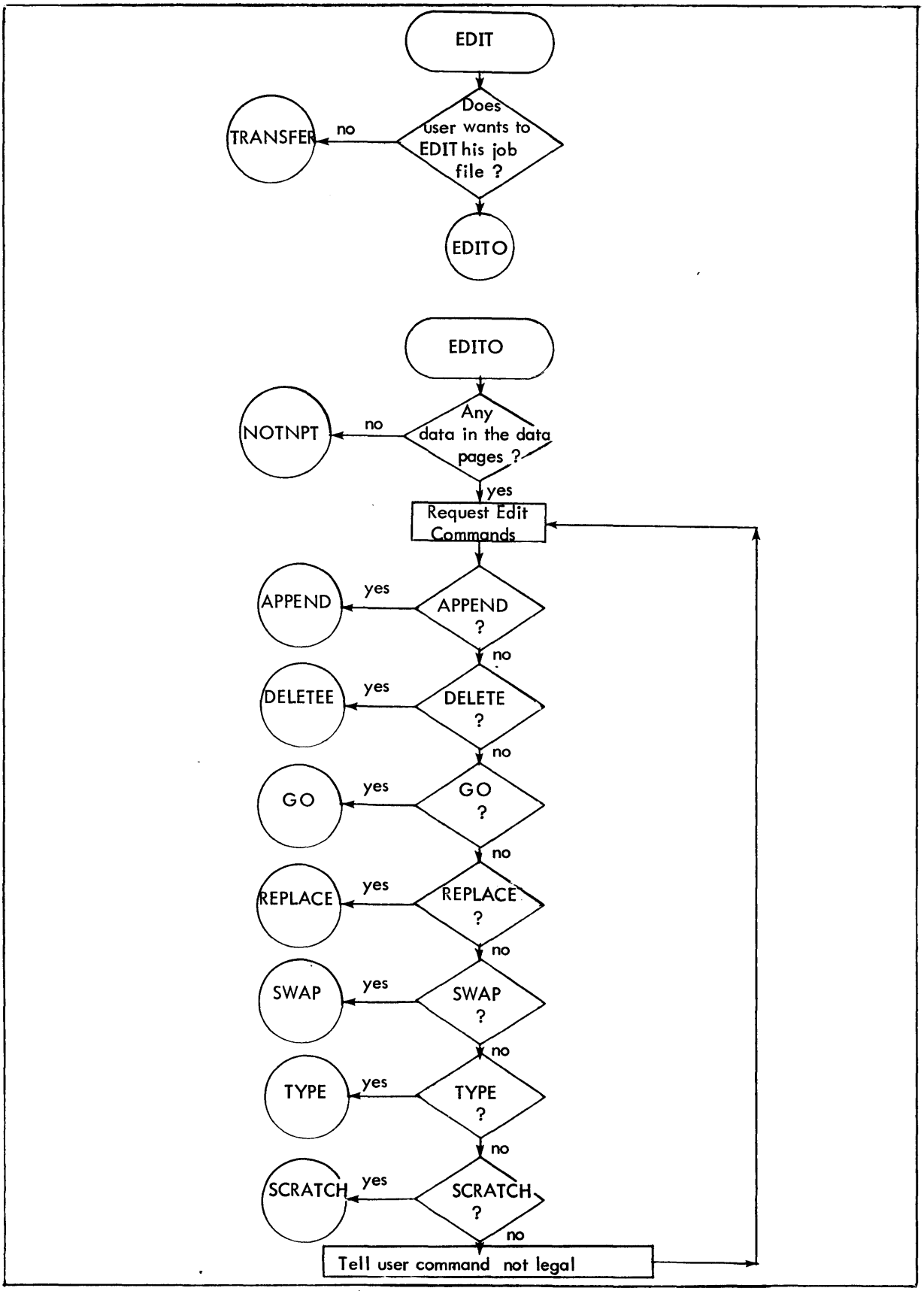


Figure 4-5. Flow Chart of EDIT

4.5 SUBROUTINE ANALYSIS

4.5.1 GO

1. Subroutine Name:
   GO
2. Purpose:
   Send the job file to the symbiont batch queue.
3. Entry:
   Entered from FINDEDIT upon detection of a "G" (go).
4. Exit:
   a. To EDIT2 if more than one line number.
   b. To EDIT3 if no line number.
   c. To EDIT1 for more edit commands (normal exit).
5. Operation:
   a. Get append line number (GETINP).
   b. If no lines exist (LINENO less then or equal to zero), then allow append to zero only.
   c. Find the line to be appended to (FINDLINE).
   d. Type line (HEXDEC) number + 1 to user and await input.
   e. Bump all line numbers in data area past the appended line number by one (1).
   f. Loop back to APPEND 1 for another input record.

4.5.2 TYPE

1. Subroutine Name:
   TYPE
2. Purpose:
   Allowes the user to type all or portions of his job file.
3. Entry:
   Entered from FINDEDIT upon detection of a "T" (TYPE).
4. Exit:
   a. Exits to EDIT 2 if no lines exist to type or all lines typed.
   b. Exits to EDIT3 if no line number was given or the starting line number was greater than the ending line number.
   c. Exits to EDIT2 if more than TWO line numbers were specified.
5. Operation:
   a. Check for the existence of any lines to type (LINENO).
   b. Find out if the user wants all lines types, one line typed or more than one line typed (GETINP).
   c. Type the line number to the user (HEXDEC).
   d. Type the line to the user (TYPELINE).
   e. Loop on the number of lines to type (REG 11) or until all are typed (LINENO).
   f. Return to EDIT1.

4.5.3 REPLACE

1. Subroutine Name:
   REPLACE
2. Purpose:
   Allows the user to replace various lines within his job file.
3. Entry:
   Entered from FINDEDIT upon detection of a "R" (REPLACE).
4. Exit:
   a. Exits to EDIT2 if no input exists to replace or if the line number itself is invalid.
   b. Exits to EDIT3 if no line number is supplied or the starting line number is greater than the ending line number.
   c. Exits to EDIT1 if the line to replace can't be found.
   d. Exits to APPENDO (Normal EXIT).
5. Operation:
   a. Check to see if any job file lines exist (LINENO).
   b. Findout if the user wants one line or more than one line replaced (GETINP).
      1. Transfer to REPLACE1 if more than one.
      2. Continue normally if only one.
   c. Delete all lines (DELLINE) requested from the line numbers as input to REPLACE.
   d. Exit to APPENDO to insert the new lines in.

4.5.4 SWAP

1. Subroutine Name:
   SWAP
2. Purpose:
   Allows the user to exchange lines within his job file.
3. Entry:
   Entered from FINDEDIT upon detection of an "S" (SWAP).
4. Exit:
   a. Exits to EDIT3 if no lines requested or only one line requested.
   b. Exits to EDIT2 if more than two lines typed.
   c. Exits to EDIT1 (Normal Exit).
5. Operation:
   a. Check to see if any job file lines exist (LINENO).
   b. Get the lines to be swapped (GETINP).
   c. Store the first line number (TEMP) over the second and the second (TEMP1) over the first.
   d. Exit to EDIT1 for next edit command.

4.5.5 DELETEE

1. Subroutine Name:
   DELETEE
2. Purpose:
   Allows the user to delete portions of his job file.
3. Entry:
   Entered from FINDEDIT upon detection of a "D" (DELETE).
4. Exit:
   a. Exit to EDIT2 if no data lines exist or invalid line number.
   b. Exit to EDIT3 if no lines were requested or starting line number is greater then the ending line number.
   c. Exit to EDIT1 (Normal) after lines have been deleted.
5. Operation:
   a. Check for any job file lines existing (LINENO).
   b. Get the lines to be deleted (GETTINP).
      1. DELETEE1 if more than one.
      2. Continue normally if only one.
   c. Delete the line (DELLINE) and return to EDIT1.

4.5.6 TRANSFER

1. Subroutine Name:
   TRANSFER
2. Purpose:
   Send the built symbiont blocks of job files to the batch job queue.
3. Entry:
   a. From EDIT when no editing is desired.
   b. From GO when the user wants the job file to be sent.
4. Exit:
   a. To NOINPT if nothing exists in the job file to send (LINENO).
   b. TRANSLST to send last block.
   c. NOJOBC if no job card was found in the job file.
5. Operation:
   a. Check for data to send (LINENO).
   b. Find each line (FINDLINE) of data.
   c. Check each line for the first byte being a "!" (control card).
   d. Move each line of job file to the symbiont buffer (SUMBBUFF) being built for the M:JOB insert cal.
   e. When full send buffer.
   f. Keep filling and sending SYMBBUFF until all records of the job file have been sent.

4.5.7 CRLF

1. Subroutine Name:
   CRLF
2. Purpose:
   Send a carriage return (X'15') and lines feed (X'25') to the users console.
3. Entry:
   a. Entry is accomplished via a BAL,R15 to CRLF.
   b. It is entered from STRTFIL, BEGIN, DELETE, STATUS, INSERT, EDITO, TYPE, YESNO, FINCARD, TRANSLST, ABNDEL, NONO, NOJOBC, FINDER, NAMER, JOBCER, INVL.
4. Exit:
   Exits through TYPEMESS by a B *R15.
5. Operation:
   a. Sets into REG14, the address of the carriage return - line feed message (CRLFMS).
   b. Falls through to TYPEMESS.

4.5.8 TYPEMESS

1. Subroutine Name:
   TYPEMESS
2. Purpose:
   Send a textc type message to the users console
3. Entry:
   a. BAL,R15 TYPEMESS.
   b. R14 = address of textc message.
   c. Entered from BEGIN, CRLF, DELETE, STATUS, INSERT, EDIT, EDITO, HEXEBC, FINCARD, TRANSLST, ABNDEL, NONO, ABNSI, NOJOBC, FINDER, ABNINS, NAMER, INVL.
4. Exit:
   Exits via B *R15
5. Operation:
   a. Obtain byte count from message pointed to by REG 14.
   b. Get bytes from message into REG 0.
   c. Issue a CAL3,1 (PRINT).
   d. Loop on byte count (REG 1).

4.5.9 YESNO

1. Subroutine Name:
   YESNO
2. Purpose:
   Recieve a "Y" or "N" response from user
3. Entry:
   (B)
   a. BAL,R15 YESNO
   b. Entered from BEGIN, EDIT, ABNINS

4. Exit:
   a. Returns to BAL + 1 if "Y".
   b. Returns to BAL + 2 if "N".
5. Operation:
   a. Set activation to "ALL" (CAL3,2 with REG 0 = 1).
   b. Recieve character (CAL3,0)
   c. Check for "Y" or "N".
   d. If not
      1. Send user a question mark.
      2. Re-request input.
   e. Bump R15 by 1 if no.
   f. Set activation to carriage return-line feed. (CAL3,2 REG = 4).
   g. Return through CRLF.

4.5.10 INEBCHEX

1. Subroutine Name:
   INEBCHEX
2. Purpose:
   Recieve ebcidic characters as input and convert them to a hex number.
3. Entry:
   a. BAL,R15 INEBCHEX
   b. Entered from DELETE, STATUS
4. Exit:
   a. Returns *R15+1 with hexadecimal number in REG 8.
   b. Returns to the time-sharing executive if a null record is supplied as input (carriage return only).
   c. Transfers to INVL if the input is not valid Hexadecinal digits.
5. Operation:
   a. Zero REG 8 (will contain resultant hex number).
   b. Get input characters and convert them to hexadecimal.
   c. Check if valid hex characters (O-F) and transfer to INVL if not.
   d. Bump return address
   e. If input came in, return *R15; if not, exit to the time-sharing executive (CAL3,6).

4.5.11 HEXDEC

1. Subroutine Name:
   HEXDEC
2. Purpose:
   Convert hexadecinal number to decimal and type on the users console, the result.
3. Entry:
   a. BAL,R15 HEXDEC
   b. R10 = hexadecimal number.

c. Entered from STATUS, INSERT, APPEND, TYPE

4. Exit:
Transfers to HEXEBC

5. Operation:

a. Set REG 0 to zero (will contain result).

b. Divide REG 10 by X'A' storing remainders in REG 0.

c. Load REG 9 with result in REG 0.

d. Full through to HEXEBC to type result.

4.5.12 HEXEBC

1. Subroutine Name:
HEXEBC

2. Purpose:
Convert hexadecimal numbers to ebcdic and type to the user, the result.

3. Entry:

a. BAL,R15 HEXEBC

b. R9 = Hexadecimal number.

c. Entered from HEXDEC, STATUS, TRANSLST, ABNSI, JOBCER.

4. Exit:
To TYPEMESS to print the number.

5. Operation:

a. Find size of number (For textc format).

b. Store size (byte count) in EBCDICNO.

c. Convert numbers to the ebcdic.

d. Store numbers in EBCDICNO.

e. Loop on byte count.

f. Set REG 14 pointing to EBCDICNO and transfer to TYPEMESS.

4.5.13 PUTLINE

1. Subroutine Name:
PUTLINE

2. Purpose:
Put a line of input data, with its line number (LINENO) into the data pages.

3. Entry:

a. BAL,R15 PUTLINE

b. R10 = line number (LINENO).

c. Entered by INSERT, APPEND

4. Exit:

a. B *R15

b. NOROOM if no more data pages available.

5. Operation:
   a. Check for room available in current page (STRTDATA) for another 21 word data record.
   b. If not:
      1. Check if any pages left and transfer to NOROOM if not.
      2. Bump NEXTPAGE by 1 to reflect next page limit.
      3. Set swap size (CAL3, 11 0) for new page obtained.
   c. Store line number (REG 10).
   d. Move record
   e. Bump STRTDATA by 21 words.
   f. Return (B *R15).

4.5.14 FINDLINE

1. Subroutine Name:
   FINDLINE
2. Purpose:
   Find a line of input in the data area by its line number.
3. Entry:
   a. BAL,R15 FINDLINE
   b. R10 = line number.
   c. Entered from APPEND, TYPE, REPLACE, SWAP, DELETEE
4. Exit:
   a. B *R15
   b. REG7 - address of desired line.
5. Operation:
   a. Set address pointer (REG 7) to beginning of data area (FIRSPAGE).
   b. Loop through data area until -
      1. Line number (REG 10) is found in data area.
      2. The data area is exhausted (STRTDATA)
   c. Return *R15 with condition codes = 0 (line found) or condition codes = 1 (set by FINDERS indicates line not found.)

4.5.15 RUNDOWN

1. Subroutine Name:
   RUNDOWN
2. Purpose:
   Run down input buffer looking for a carriage return (X'15).
3. Entry:
   a. BAL, R15 RUNDOWN
   b. Entered from EDITO, GO, SCRATCH, INVL
4. Exit:
   B *R15

5. Operation:
   a. Read input character (CAL3,0).
   b. Loop until carriage return found.
   c. Return *R15

4.5.16 BANGHIT

1. Subroutine Name:
   BANGHIT
2. Purpose:
   Analyze a "bang" card (!) hit during a transfer to the symbiont batch queue, for a !FIN, !EOD, !JOB.
3. Entry:
   a. B BANGHIT
   b. R7 = address of data line in data buffer.
   c. Entered from TRANSFER
4. Exit:
   a. Return to TRANSIO if "Bang" card was not special.
   b. Transfer to EODCARD, FINCARD, JOBCARD depending on the type of card.
5. Operation:
   a. Loop thru card, ignoring, blanks storing the first 4 characters in REG 15 (including the "BANG" (!)).
   b. Compare REG 15 with SPECBANG.
   c. If equal, transfer to appropriate routine (BANGBRUS)
   d. Return to TRANSIO if not equal.

4.5.17 EODCARD

1. Subroutine Name:
   EODCARD
2. Purpose:
   Set special control word in the symbiont buffer (SYMBBUFF) for an !EOD card.
3. Entry:
   Entered from BANGHIT
4. B TRANS2
5. Operation:
   a. Set REG 12 = EODCONS
   b. Return (B TRANS2)

4.5.18 FINCARD

1. Subroutine Name:
   FINCARD
2. Purpose"
   Delete a !FIN from the input job file
3. Entry:
   Entered from BANGHIT
4. Exit:
   B TRANS1
5. Operation:
   a. Inform user that the !FIN card is being ignored (TYPEMESS)
   b. Return (B TRANS1)

4.5.19 JOBCARD

1. Subroutine Name:
   JOBCARD
2. Purpose:
   Process the !JOB control card from the input job files
3. Entry:
   a. Entered from BANGHIT
   b. R10 = line number (LINENO); R7 = address of job card in buffer.
4. Exit:
   a. Exit to NAMECOMP after getting accounts.
   b. Exit to COMPALL if all fields of the job card have been exhausted.
   c. Exit to NOJOBC if job card is not the first line number.
5. Operation:
   a. Check if there is a "JOB" currently waiting to be sent (JPR greater or equal to zero)
   b. If not transfer to JOBCARDO;
      1. Set EOBCONS (X'40') into SYMBBUFF
      2. Send the current block (SYMBBUFF)
      3. Delete all lines from the data area (DELLINE) which have been previous sent (M:JOB INSERT) to the batch job queue.
      4. Re-initialize the line number (set it to 1); find the job card (FINDLINE).
   c. Blank out USERACNT, USERNAME
   d. Search !JOB card for the account, storing it into USERACNT.

4.5.20 NAMECOMP

1. Subroutine Name:
   NAMECOMP
2. Purpose:
   Obtain the name field from the !JOB card.
3. Entry:
   entered from SEARCH, when a comma (,) is detected on the !JOB card.

4. Exit:
   Exits to PRIOR after storing the name field from the !JOB card.
5. Operation:
   a. Search !JOB card for the name field.
   b. Store it a byte at a time into USERNAME.

4.5.21 EXTACT

1. Subroutine Name:
   EXTACT
2. Purpose:
   Check extended accounting field on the !JOB card for extended accounting conformity.
3. Entry:
   Entered from CHKLIM upon detection of a left parenthesis (( ).
4. Exit:
   a. To JOBCER if the extended accounting format is in error.
   b. To EXACER if illegal delimiters exist within the extended accounting record.
5. Operation:
   a. Check to see if the routine has been re-entered (EXTFLG greater than zero).
   b. Search down !JOB card until a right parenthesis is encountered ( )).

4.5.22 PRIOR

1. Subroutine Name:
   PRIOR
2. Purpose:
   Obtain the priority field from the !JOB card.
3. Entry:
   a. Entered from NAMECOMP upon detection of a comma (,).
   b. Entered from EXTACT upon detection of a right parenthesis ( )).
4. Exit:
   a. Exit to PRIOER if priority field is not a hexadecimal number.
   b. Exit to JOBCER if the priority field is not followed by a space or a comma.
   c. Exit to COMPALL if priority field is legitimate.
5. Operation:
   a. Get priority from Job card (SERCH).
   b. Check if legal hexadecimal character.

4.5.23 COMPALL

1. Subroutine Name:
   COMPALL
2. Purpose:
   Compare fields from !JOB card to authorization from AJIT.
3. Entry:
   a. Entered when all fields of the !JOB card have been processed.
   b. Entered from PRIOR, CHKLIM.

4. Exit:
   a. ACCTER if account numbers is error.
   b. NAMER if name field in error.
   c. PRIOER if priority is greater than MAXPRI.
5. Operation:
   a. Compare USERACNT (from !JOB card) to LOGACNT (log-in account).
   b. Compare USERNAME (from !JOB card) to LOGNAME (log-in name).
   c. Compare JPR (priority from !JOB card) to MAXPRI.
   d. Return to TRANSIO.

4.5.24 SERCH

1. Subroutine Name:
   SERCH
2. Purpose:
   Get bytes of fields from !JOB card.
3. Entry:
   a. BAL,R15 SERCH
   b. R7 = address of !JOB card in data area; R3 = byte displacement on !JOB card; REG 0 = character.
   c. Entered from JOBCARD, NAMECOMP, EXTACT, PRIOR, SERDOWN.
4. Exit:
   a. B *R9 if serched more than 80 character.
   b. B *R15 normal return.
5. Operation:
   a. Obtain byte from !JOB card.
   b. Check for greater than 80 characters processed.
   c. Compare desired character REG 0 with that from card REG 1.
   d. Return

4.5.25 CHKLIM

1. Subroutine Name:
   CHKLIM
2. Purpose:
   Check for any delimiters hit on !JOB card.
3. Entry:
   a. BAL,R14 CHKLIM
   b. R11 = address of desired subroutine of a comma (,) is found; R1 = character to be checked.
   c. Entered from JOBCARD, NAMECOMP, SERDOWN.
4. Exit:
   a. Exits to SERDOWN if a blank is detected.
   b. Exits to COMPALL if a period is detected.

5. Operation:
    a. Compare current character REG 1 to various delimiters.
    b. Branch to appropriate routine if it compares; otherwise.
    c. Return B *R14.

4.5.26 SERDOWN

1. Subroutine Name:
   SERDOWN
2. Purpose:
   Rundown !JOB card, looking for a comma (,).
3. Entry:
   B SERDOWN
4. Exit:
    a. Branch to JOBCER if comma can't be found.
    b. BAL,R14 CHKLIM
5. Operation:
    a. Set character to be searched for to a blank REG = X'40').
    b. Serch card for it.
    c. Check limiter on card at first non-blank character.

4.5.27 GETINP

1. Subroutine Name:
   GETINP
2. Purpose:
   Input decimal characters and convert them to a hexadecimal number.
3. Entry:
    a. BAL,R15 GETINP
    b. Entered from APPEND, TYPE, REPLACE, SWAP, DELETEE
4. Exit:
    a. Transfers to EDIT1 if input error.
    b. B *R15 Normal exit; R10 = number.
5. Operation:
    a. Issue a read request (CAL3,0) from the users console.
    b. Convert number to hexadecimal.
    c. Pack number into REG 11.
    d. Loop till carriage return or comma.
    e. Load REG 10 from REG 11.
    f. Return *R15 with
        1. Condition codes set to zero if no numbers were obtained.
        2. Condition codes set to one if one number was obtained.

3. Condition codes set to two if more than one number was obtained.

4.5.28 TYPELINE

1. Subroutine Name:
   TYPELINE
2. Purpose:
   Type a line of the job file on the users console.
3. Entry:
   a. BAL,R15 TYPELINE
   b. R10 = desired line number.
   c. Entered from TYPE, JOBCER
4. Exit:
   a. Transfers to EDIT1 if the line does not exist.
   b. B *R13 (REG 15 is saved by the routine in REG 13).
5. Operation:
   a. Find the requested line (FINDLINE)
   b. Type a character at a time, of the record (CAL3,1).
   c. Loop until 80 characters typed.

4.5.29 DELLINE

1. Subroutine Name:
   DELLINE
2. Purpose:
   Delete records of the job file from the data area.
3. Entry:
   a. BAL,R13 DELLINE
   b. R10 = starting line number; R11 = number of records to delete.
   c. Entered from REPLACE, DELETEE, JOBCARD.
4. Exit:
   a. Transfers to EDIT1 if line number can't be found (REG 10).
   b. B *R13 normal exit.
5. Operation:
   a. Find the line to be deleted (FINDLINE)
   b. Zap the line number.
   c. Loop until the specified count (REG 11) has been deleted or until past the last line number (LINENO).
   d. Down date remaining line numbers.
   e. Return (B *R13).

4.5.30 TRANSLST

1. Subroutine Name:
   TRANSLST
2. Purpose:
   Close out the job file, send it to the symbiont batch queue and inform user of his job ID.
3. Entry:
   a. BAL,R13 TRANSLST
   b. Entered from TRANDONE, JOBCARD
4. Exit:
   B *R13
5. Operation:
   a. Set last flag (EODCONS) into SYMBBUFF.
   b. Send last block (M:JOB insert, with priority field on).
   c. Tell user job was inserted (TYPEMESS).
   d. Tell user this ID (HEXEBC).
   e. Return (B *R13).

4.5.31 FINDEDIT

1. Subroutine Name:
   FINDEDIT
2. Purpose:
   Branch to appropriate edit option, specified by the user.
3. Entry:
   a. BAL,R15 FINDEDIT
   b. R0 = users specified edit command.
   c. Entered from EDITO.
4. Exit:
   a. B *R15 if not found.
   b. Branch to desired option.
5. Operation:
   a. Obtain legal edit commands (EDITCM)
   b. Compare legal to desired.
   c. Loop on the number of edit commands allowed (NOEDITCM)
   d. Branch to option if found.
   e. B *R15 if not found.

4.5.32 ABNDEL

1. Subroutine Name:
   ABNDEL
2. Purpose:
   Handle any abnormal returns from an M:JOB delete request.

3. Entry:

   a. Entered by an abnormal from M:JOB delete request.

   b. R 10 = abnormal code; R8 = address of the M:JOB CAL+1.

4. Exit:

   Return to DELETE for another ID request.

5. Operation:

   a. Obtain abnormal code from REG 10 (Byte 1 of REG 10).

   b. Type to the user (TYPEMESS) an error message corresponding to one of the following abnormal codes.

      1. Abnormal code= X'39' - ID nonexistent.
      2. Abnormal code = X'3A' - too late.
      3. Abnormal code = X'3C' - user not authorized to delete job.
      4. Abnormal code = X'3D' - nonsymbiont system

   c. Return to DELETE to issue another ID request.

4.5.33 NONO, COMMOUT

1. Subroutine Names:
   NONO; COMMOUT

2. Purpose:
   Abort the user with an appropriate message.

3. Entry:

   a. B NONO; B COMMOUT

   b. R14 = address of message.

   c. Entered from BEGIN, NOROOM, NOINPT, ABNSI, ABNINS.

4. Exit:
   To the time-sharing executive.

5. Operation:

   a. Type message (in REG 14) to the user (TYPEMESS).

   b. Abort the user (CAL3,6).

4.5.34 NOROOM

1. Subroutine Name:
   NOROOM

2. Purpose:
   Abort the user when his data buffer is exhausted.

3. Entry:

   a. B NOROOM

   b. Entered from PUTLINE

4. Exit:
   To COMMOUT

5. Operation:

   a. Set REG 14 to message address (NOROOMS).

   b. Transfer to COMMOUT

4.5.35 ABNINS

1. Subroutine Name:
   ABNINS
2. Purpose:
   Handle abnormal returns from a M:READ of the M:SI DCB.
3. Entry:
   a. Entered by an abnormal form as M:READ request of the M:SI DCB.
   b. R10 = abnormal code; R8 = address of CAL=1.
4. Exit:
   a. To INSERT1 if console input.
   b. To EDIT if end of file.
   c. To COMMOUT if not end of file.
5. Operation:
   a. Return to INSERT1 if console input.
   b. Obtain abnormal code from REG 10 (Byte 0).
   c. If end of file (X'06'), transfer to EDIT.
   d. Tell user he had an abnormal from M:SI (TYPEMESS).
   e. Type abnormal code to him (HEXDEC).
   f. Abort (COMMOUT).

4.5.36 NOINPT

1. Subroutine Name:
   NOINPT
2. Purpose:
   Abort user if no job file exists and he tries to send one.
3. Entry:
   a. B NOINPT
   b. Entered from EDITO, TRANSFER
4. Exit:
   B COMMOUT
5. Operation:
   a. Set into REG 14 the address of the message (NOINPTMS).
   b. Transfer to COMMOUT.

4.5.37 NOJOBC

1. Subroutine Name:
   NOJOBC
2. Purpose:
   Inform user that his job file cannot be sent to the batch job queue because it does not contain a !JOB card as the first record.
3. Entry:
   a. B NOJOBC

b. Entered from TRANSFER, JOBCARD, TRANSLST.

4. Exit:
   B EDITRTY

5. Operation:

   a. Tell user (TYPEMESS) that the job card is messing; (NOJOBCMS).

   b. Transfer to EDITRTY

4.5.38 FINDER

1. Subroutine Name:
   FINDER

2. Purpose:
   Inform the user that the desired line number cannot be found.

3. Entry:

   a. BAL,R15 FINDER

   b. B *R13 (REG 13 is loaded from REG 15 upon entry).

4. Operation:

   a. Save REG 15 into REG 13.

   b. Tell user (TYPEMESS) line could not be found (FINDERMS).

   c. Set condition codes to a line (line not found).

   d. Return (B *R13)

4.5.39 ABNINS

1. Subroutine Name:
   ABNINS

2. Purpose:
   Handle any abnormal returns from an M:JOB insert cal.

3. Entry:

   a. Entered from an abnormal return caused by an M:JOB insert request.

   b. R10 = abnormal code; R8 = address of CAL+1.

4. Exit:

   a. To COMMOUT1 if the user does not want to retry the M:JOB insert.

   b. To the M:JOB cal to retry the insert.

5. Operation:

   a. Save address of cal + 1 in REG 7 (REG 8 = address of cal+1).

   b. If abnormal not because of symbiont full (X'3B' in byte 1 of REG 10), then abort user (COMMOUT1).

   c. Ask user for retry (YESNO).

   d. B -1, R7 if retry is desired; otherwise, abort him (COMMOUT1).

4.5.40 NAMER; JOBCERR

1. Subroutine Name:
   NAMER; JOBCERR
2. Purpose:
   Inform the user his name field on the job card was not correct.
3. Entry:
   a. B NAMER; B JOBCERR
   b. Entered from COMPALL, ACCTER, EXACER, PRIOER, JOBCER.
4. Exit:
   Transfers to EDITRTY
5. Operation:
   a. Type error message (TYPEMESS).
   b. Zap priority (JPR= -1).
   c. Transfer to EDITRTY

4.5.41 ACCTER; EXACER; PRIOER

1. Subroutine Names:
   ACCTER, EXACER, PRIOER
2. Purpose:
   Inform the user his job card in in error.
3. Entry:
   a. B ACCTER; B EXACER; B PRIOER.
   b. Entered from JOBCARD, COMPALL, SERCH, EXTACT, PRIOR
4. Exit:
   B JOBCERR
5. Operation:
   a. Set proper error message into REG 14.
   b. Transfer to JOBCERR.

4.5.42 JOBCER

1. Subroutine Name:
   JOBCER
2. Purpose:
   Inform the user of an error on his job card.
3. Entry:
   a. B JOBCER
   b. Entered from EXTACT, PRIOR, CHKLIM, SERDOWN.
4. Exit:
   B JOBCERR
5. Operation:
   a. Type the line number of the !JOB card.
   b. Type the !JOB card. (TYPELINE).
   c. Type a "$" under the error on the ! JOB card
   d. Transfer to JOBCERR

4.5.43 INVL

1. Subroutine Name:
   INVL
2. Purpose:
   Inform user that he typed in an invalid ID.
3. Entry:
   a. BAL,R15 INVL
   b. Entered from INEBCHEX
4. Exit:
   B *R13 (REG 13 is loaded from REG 15 upon entry).
5. Operation:
   a. Run down input looking for a carriage return (RUNDOWN).
   b. Type message (TYPEMESS) to user (INVLMS).
   c. B *R13

5.0 SUPER PROCESSOR/SUBSYSTEM

5.1 FUNCTIONAL OVERVIEW.

5.1.1 Purpose.

SUPER (Supervisor Control) is a BPM processor or a BTM subsystem that provides the ability to create, to update, to list, and to summarize the authorization file, :USERLG, which is used by the operating system to control and to record user activity. In order to access the authorization file, SUPER must be run under the :SYS account.

5.1.2 Input.

The following control records and associated specification records constitute input. Control records designate the function desired. Specification records for the control record follow certain control records to give detailed information. Input records contain data beginning in the first record position.

1. USERS. USERS authorizes users for various Monitor services by indicating how records in the authorization file are to be created or updated.
   a. The control record indicates the Monitor services for which the user(s) are to be authorized. The record has the format:

      U[SERS][, BCH][, BTM][, FGD][, RBT]

      where:

      BCH specifies batch-job.

      BTM specifies time-sharing.

      FGD specifies real-time job.

      RBT specifies remote-batch-job.

   b. The specification record gives details on the users to be authorized. The record has the format:

      account, name [(extended accounting), password, batch priority, RAD granules, disk granules]

      where:

      account specifies the user account number, a maximum of 8 alphanumeric characters.

      name specifies the user name, a maximum of 12 alphanumeric characters.

      extended accounting specifies installation-specific accounting information, a maximum of 24 characters (excluding the parentheses). The default is no extended accounting, blanks.

      password specifies the user password, a maximum of 8 characters. The default is no password, blanks.

      batch priority specifies in hexadecimal the maximum job priority allowed the user, 0-F. The default is 1.

      RAD / DISK granules specifies in decimal the maximum amount of secondary storage allowed the user. The maximum is 65,535. The default is 0.

   c. On the specification record, if any of the optional fields are blank and no record for the user exists in the authorization file, blank fields are set to default values. If any of the optional fields are blank, but a record for the user already exists in the authorization file, blank fields remain the same as on the previous record.

2. KILL. KILL cancels user authorization for Monitor services by designating a record to be deleted from the authorization file.
   a. The control record has the format:

      K[ILL]

b. The specification record has the format:

account, name

3. LIST. LIST lists authorized users by reading the authorization file and printing information from the file.

a. The control record has the format:

L[IST][, account]

b. If an account is specified, all users authorized for the account are listed. If no account is specified, all users authorized for the system are listed.

4. STATS. STATS summarizes user statistics by reading the authorization file and processing data from the file.

a. The control record has the format:

S[TATS]

b. The optional specification record has the format:

account[, name]

c. If the specification record contains an account and a name, statistics for that user are summarized. If the specification record contains an account only, statistics for all users in the account are summarized. If the specification record is omitted, statistics for all users in the system are summarized.

5. DELSTATS. DELSTATS deletes (resets) user statistics by designating records to be updated in the authorization file. All statistics are reset to zero, except amount of secondary storage used.

a. The control record has the format:

D[ELSTATS]

b. The optional specification record has the format:

account[, name]

c. If the specification record contains an account and a name, statistics for that user are reset. If the specification record contains an account only, statistics for all users in the account are reset. If the specification record is omitted, statistics for all users in the system are reset.

6. PASSWORDS. PASSWORDS lists passwords of user files. If printable, passwords are listed in EBCDIC. If non-printable, passwords are listed in hexadecimal. If no password exists, "***NONE***" is printed.

a. The control record has the format:

P[ASSWORDS], account

b. Passwords for all files in the account are listed.

5.1.3 Batch Operation.

1. Job Account. The account on the job card must be :SYS.
2. DCBs. The M:SI DCB is used to read user input and the M:LO DCB is used to produce listing output.
3. Control Commands. A !SUPER control command is followed by control records with specification records. A !EOD record is required between each set of a control record and the specification records, if any, for the control record.
4. Input Syntax Error. A dollar sign ($) is printed beneath the erroneous field and the error message - "SYNTAX ERROR, RECORD IGNORED"- is printed.

5.1.4 On-Line Operation.

The characters typed by SUPER are shown in upper-case for alphabetic characters and underlined.

1. Log-on Account. The user must log-on under the :SYS account.
2. Input Assignment. SUPER reads input from a file or from the terminal. SUPER types:

:

a. For file input, enter the ID in the format:

File name[(account [, password])]

b. For terminal input, respond with a carriage return (CR).

3. Options. SUPER types:

ENTER OPTION:

:

a. After the prompt :, respond with one of the control records or an "X" to exit to the executive.

b. After a control record has been entered, SUPER prompts with the character > for each specification record for the control record. After the prompt >, enter a specification record if desired. To indicate the last specification record for the control record and to prepare to enter the next control record, respond with a carriage return, a line feed (LF), or an "X".

4. Input Syntax Error. SUPER types the format of the record required.

5.1.5 Error Messages.

1. ...ILLEGAL OPTION, OPTIONS ARE: In processing a control or specification record in on-line mode, an invalid option has been found.
2. ...SYNTAX ERROR, FORMAT IS: In processing a control or specification record in on-line mode, a syntax error has been found.
3. **WARNING** ABOVE USER NOT FOUND. In processing the KILL or DELSTATS option, no authorization-file record for the specified user exists.
4. ABNORMAL I/O ON OPEN NEXT. In processing the PASSWORDS option, the open of the next user file failed.
5. ABORT: :USERLG FILE DOESN'T EXIST. In processing the USER, KILL, LIST, STATS, or DELSTATS option, the authorization file does not exist.
6. ABORT: ABNORMAL I/O ON :USERLG DELREC. In processing the KILL option, a delete to the authorization file failed.
7. ABORT: ABNORMAL I/O ON :USERLG WRITE. In processing the USERS option, a write to the authorization file failed.
8. ABORT: ABNORMAL I/O ON LOG READ DIRECT. In processing the USERS, STATS, or DELSTATS option, a read to the authorization file failed.
9. ABORT: ABNORMAL I/O ON LOG REWRITE SEQUEN. In processing the DELSTATS option, a write to the authorization file failed.
10. ABORT: ABNORMAL I/O ON READ LOG SEQUEN. In processing the LIST, STATS, or DELSTATS option, a read to the authorization file failed.
11. ABORT: CAN'T OPEN :USERLG. In processing the USERS, KILL, LIST, STATS, or DELSTATS option, the authorization file cannot be opened.
12. ABORT: READ M:SI ERROR. In processing an option, the open or a read of the M:SI DCB failed.
13. ABOVE USER NOT VALIDATED. In processing the STATS option, no authorization-file record for the specified user exists.
14. BAD FID, TRY AGAIN. In on-line mode, a bad file identification has been entered for the input assignment.
15. CANNOT ACCESS :USERLG FILE. The job account in batch mode or the log-on account in on-line mode is not :SYS so SUPER cannot access the authorization file.
16. SYNTAX ERROR, RECORD IGNORED. A control or specification record contains an error in the field above the dollar sign.

5.2 INTERFACES.

5.2.1 Operating System.

In batch mode, when processing the PASSWORDS option, SUPER uses the M:SYS call to run in master mode in order to store into the DCB used to access the user files. In on-line mode, SUPER stores directly into an unprotected dummy DCB, not the actual DCB used by the BTM executive.

5.2.2 CCI.

After SUPER creates the authorization file, :USERLG, CCI places job usage times into the file.

5.3 OPERATIONAL OVERVIEW.

5.3.1 Description.

Begin processing in the START module. If on-line SUPER is not running under the :SYS account, branch to the Normal-Exit module to terminate the job step. Otherwise, open the M:SI DCB through which control and specification records are to be read and, in batch mode, read the !SUPER record. If the open or the read fails, branch to the Abnormal-Exit module to process abnormal termination of the job step. Begin the main processing loop in the Control-Record module. Read the control record. If the read fails, branch to the Abnormal-Exit module. Otherwise, determine which module should process the record and branch to that module. Process the option in the USERS, the KILL, the LIST, the STATS, the DELSTATS, the PASSWORDS, or the Input-Syntax-Error module and branch back to the Control-Record module. When processing the option, if I/O operations fail when accessing the control or specification records or the authorization file, branch to the Abnormal-Exit module. When an exit condition is found in the Control-Record module, branch to the Normal-Exit module.

5.3.2 DCBs.

1. M:SI. Control and Specification Input Records.
2. M:LO. Listing Output.
3. M:DO. Authorization File, :USERLG.
4. M:EI. User Files.

5.3.3 Flowchart.

See Figure 5-1.

5.4 MODULE ANALYSIS.

5.4.1 START.

1. Module Name - START.
2. Purpose. Perform processing preceding the main processing loop. Open the M:SI DCB through which control and specification records are to be read and, in batch mode, read the !SUPER record.
3. Entry.
   a. START is the only entry point for the SUPER processor/subsystem. Whenever SUPER is invoked, the processor/subsystem is loaded into core and control is transferred to START.

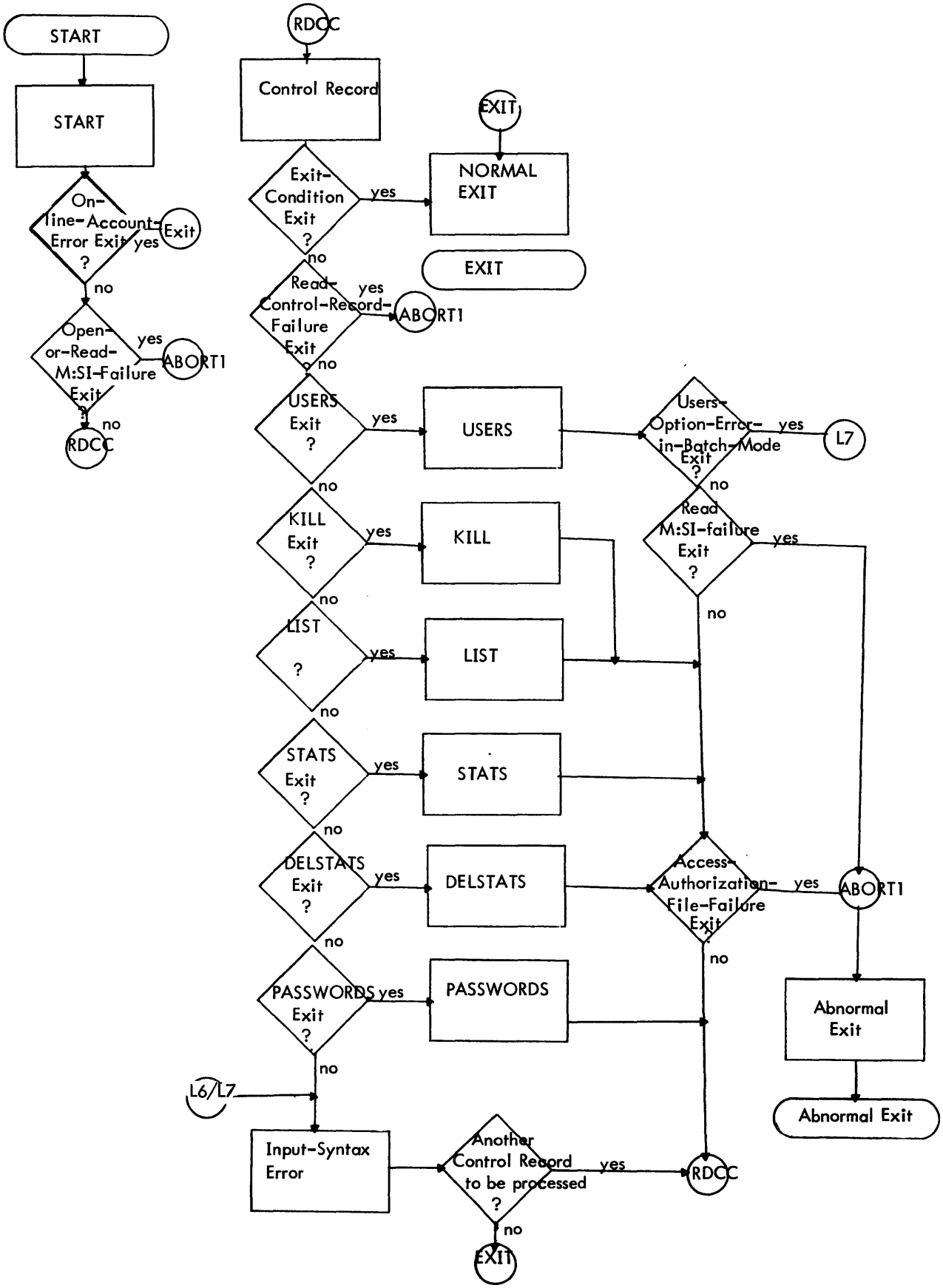


Figure 5-1. Operational Overview of SUPER

b. In batch mode, the !SUPER record must be input.

4. Exit.

a. The norma exit is to the Control-Record module, RDCC.

(1) The exit is taken after initial processing is completed.

(2) The ABORT indicator is initialized off (zero).

b. The exit if on-line SUPER is not running under the :SYS account is to the Normal-Exit module, EXIT. The error message printed is: "CANNOT ACCESS :USERLG FILE."

c. The exit if the open or a read of the M:SI DCB fails is to the Abnormal-Exit module, ABORT1. The error message printed is: "ABORT: READ M:SI ERROR."

5. Operation.

a. Do initial form alignment for output listing (PAGE subroutine).

b. Do on-line processing. Change DCBs for running on-line. If the log-on account is not :SYS, print an error message (PRNT subroutine) and branch to the Normal-Exit module, EXIT. Otherwise, request the assignment for input (PRNT subroutine). Read the response and assign the M:SI DCB to a file or to the user terminal. If a bad file identification is specified, print an error message (PRNT subroutine) and go to step a. If input is assigned to a file, open the M:SI DCB and branch to the Control-Record module, RDCC. If assigned to the user terminal, go to step c.

c. Open the M:SI DCB and, in batch mode, read the !SUPER record.

d. If opening the M:SI DCB returns an error code or if reading the M:SI DCB returns an abnormal or error code, print an error message (PRNT subroutine) and branch to the Abnormal-Exit module, ABORT1.

5.4.2 Control-Record.

1. Module Name - RDCC.

2. Purpose. Read the next control record to be processed and determine which module will process the record.

3. Entry.

a. Control-Record is entered from START, USERS, KILL, LIST, STATS, DELSTATS, PASSWORDS, or Input-Syntax-Error.

b. Control-Record has input.

(1) Control records are read.

(2) The ABORT indicator is checked.

4. Exit.

a. The normal exit is to the module that processes the control record: USERS, KILL, LIST, STATS, DELSTATS, PASSWORDS, or Input-Syntax-Error.

(1) The exit is taken after the option has been found in a jump table.

(2) A control record has been read and, in on-line mode, the prompt character has been set to a caret (>).

b. The exit if all control records have been processed or if the ABORT indicator has been set is to the Normal-Exit module, EXIT.

c. The exit if a read of a control record fails is to the Abnormal-Exit module, ABORT1. The error message printed is: "ABORT: READ M:SI ERROR."

5. Operation.

a. Branch to the Normal-Exit module, EXIT, if the ABORT indicator is set (non-zero). Otherwise, in on-line mode, request the option (PRNT subroutine) and set the prompt character to a colon (:).

b. Read a control record. If an EOD is read, read another record. When the end-of-file is reached, branch to the Normal-Exit module, EXIT. For any other read failure, print an error message (PRNT subroutine) and branch to the Abnormal-Exit module, ABORT1.

c. Analyze the option. Obtain the first EBCDIC field from the control record (GEBFLD subroutine). If a field cannot be found, branch to the Input-Syntax-Error module, L6. Otherwise, use a jump table to determine which module processes the control record and branch to that module: USERS, KILL, LIST, STATS, DELSTATS, PASSWORDS, or to Input-Syntax Error, L6. In on-line mode, first set the prompt character to a caret (>).

5.4.3 USERS.

1. Module Name - USERS.
2. Purpose. Process USERS option to authorize users for various Monitor services by creating and/or updating records in the authorization file.
3. Entry.
   a. USERS is entered from Control-Record.
   b. USERS has input.
      (1) Specification records are read. In on-line mode, the prompt character has been set to a caret (>).
      (2) The authorization file, :USERLG, is accessed.
4. Exit.
   a. The normal exit is to the Control-Record module, RDCC.
      (1) The exit is taken after all specification records for the USERS option have been processed.
      (2) The authorization file, :USERLG, has been created or updated according to the options on the control and specification records.
      (3) If a specification record contains a syntax error, the error message printed is: "SYNTAX ERROR, RECORD IGNORED."
   b. In batch mode or in on-line mode with file input, if the USERS control record contains an invalid Monitor-service option, the exit is to the Input-Syntax-Error module, L7. A dollar sign ($) is printed under the erroneous field.
   c. In on-line mode with terminal input, if the USERS control record contains an invalid Monitor-service option, the exit is to the Control-Record module, RDCC. A dollar sign is printed under the invalid field and the correct form of the USERS control record is printed.
   d. The exit if a read of the M:SI DCB fails is to the Abnormal-Exit module, ABORT1. The error message printed is: "ABORT: READ M:SI ERROR."
   e. The exit if the open of the authorization file fails is to the Abnormal-Exit module, ABORT1. For an abnormal return, the error message printed is: "ABORT: :USERLG FILE DOESN'T EXIST." For an error return, the error message printed is: "ABORT: CAN'T OPEN :USERLG."
   f. The exit if any other access of the authorization file fails is to the Control-Record module, RDCC.
      (1) The exit is taken if a write fails. The error message printed is: "ABORT: ABNORMAL I/O ON :USERLG WRITE."
      (2) The exit is taken if a read fails. The error message printed is: "ABORT: ABNORMAL I/O ON LOG READ DIRECT."
      (3) The ABORT indicator is set.
5. Operation.
   a. Analyze the Monitor-service option. From the USERS control record, obtain the next EBCDIC field (GEBFLD subroutine). Set a flag for the Monitor service specified (UCHKFLD subroutine). If the Monitor-service option is invalid, do error processing using the PRTERR and the HELP subroutines and, if in batch mode in on-line mode with file input, branch to the Input-Syntax-Error module, L7, or, if in on-line mode with terminal input, branch to the Control-Record module, RDCC.
   b. Open the authorization file in update mode if the file exists or in out mode if the file does not exist. If the open fails, print an error message (PRNT subroutine) and branch to the Abnormal-Exit module, ABORT1.

c. Set the Monitor-service flags for the authorization-file record.

d. Read a specification record. When an EOD or an EOF is read, go to step g. For any other read failure, print an error message (PRNT subroutine) and branch to the Abnormal-Exit module, ABORT1.

e. Format on authorization-file record according to the options on the control and specification record (INITLOG, GEBFLD, GHFLD, GDFLD subroutines). If a specification-record option is invalid, do error processing using the PRTERR and the HELP subroutines and go to step c.

f. Write the authorization-file record created (WRLOG subroutine) and go to step c. For any write failure other than the record-already-exists abnormal condition, print an error message (PRNT subroutine), set the ABORT indicator, and go to step g. If the record already exists, read and rewrite the record using the RDLOG and the REWRLOG subroutines and go to step c. If the read fails, print an error message (PRNT subroutine) and set the ABORT indicator.

g. Close the authorization file with SAVE specified and branch to the Control-Record module, RDCC.

6. Flowchart.

See Figure 5-2.

5.4.4 KILL.

1. Module Name - KILL.
2. Purpose. Process KILL option to cancel user authorization for Monitor services by deleting records from the authorization file.
3. Entry.
   a. KILL is entered from Control-Record.
   b. KILL has input.
      (1) Specification records are read. In on-line mode, the prompt character has been set to a caret (>).
      (2) The authorization file, :USERLG, is accessed.
4. Exit.
   a. The normal exit is to the Control-Record module, RDCC.
      (1) The exit is taken after all specification records for the KILL option have been processed.
      (2) The authorization file, :USERLG, has been updated according to the options on the specification records.
      (3) Error messages may have been printed.
         (a) In batch mode, if a specification record contains a syntax error, the message printed is: "SYNTAX ERROR, RECORD IGNORED."
         (b) If no authorization-file record for a specified user exists, the message printed is: "**WARNING** ABOVE USER NOT FOUND".
   b. The exit if a read of the M:SI DCB fails is to the Abnormal-Exit module, ABORT1. The error message printed is: "ABORT: READ M:SI ERROR".
   c. The exit if the open of the authorization file fails is to the Abnormal-Exit module, ABORT1. For an abnormal return, the error message printed is: "ABORT: :USERLG FILE DOESN'T EXIST". For an error return, the error message printed is: "ABORT: CAN'T OPEN :USERLG".
   d. The exit if a delete to the authorization file fails is to the Control-Record module, RDCC.
      (1) The exit is taken immediately after attempting the operation.
      (2) Error processing has been done.
         (a) The ABORT indicator is set.
         (b) The error message printed is: "ABORT: ABNORMAL I/O ON :USERLG DELREC".

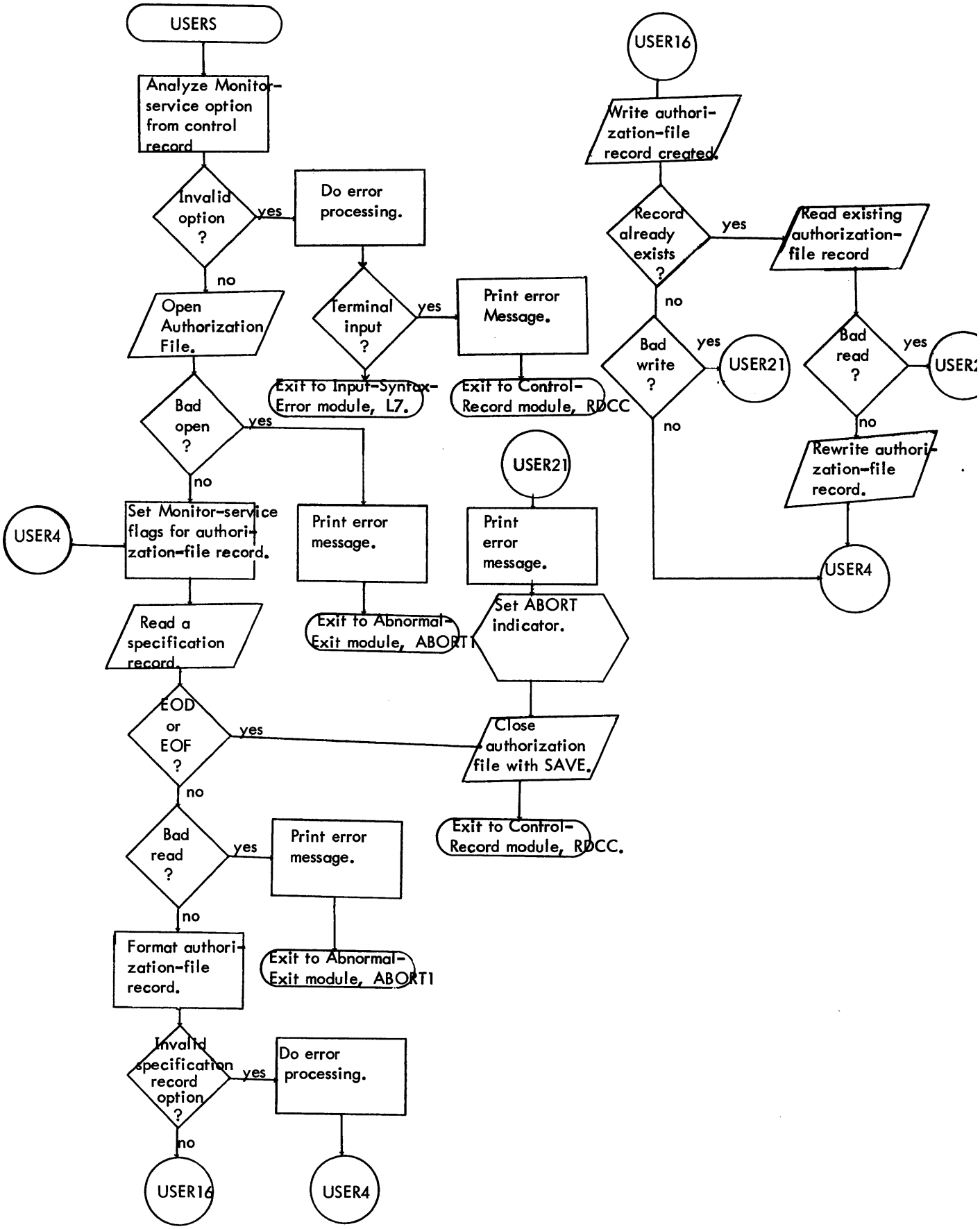


Figure 5-2. Flow Chart for USERS

5. Operation.

   a. Open the authorization file in update mode. If the open fails, print an error message (PRNT subroutine) and branch to the Abnormal-Exit module, ABORT1.

   b. Read a specification record. When an EOD or an EOF is read, go to step e. For any other read failure, print an error message (PRNT subroutine) and branch to the Abnormal-Exit module, ABORT1.

   c. Format a key to access the authorization file using the account and the name options from the specification record (GACNTN subroutine). If an option is invalid, do error processing using the PRTERR and the HELP subroutines and go to step b. In batch mode, first print an error message (PRNT subroutine).

   d. Delete the specified record from the authorization file. If the record cannot be found, print an error message (PRNT subroutine) and go to step b. For any other delete failure, print an error message (PRNT subroutine) and set the ABORT indicator.

   e. Close the authorization file with SAVE specified and branch to the Control-Record module, RDCC.

5.4.5 LIST.

1. Module Name - LIST.

2. Purpose. Process LIST option to list authorized users by reading the authorization file and printing information from the file.

3. Entry.

   a. LIST is entered from Control-Record.

   b. LIST has input.

      (1) The LIST control record must have been read. In on-line mode, the prompt character has been set to a caret (>).

      (2) The authorization file, :USERLG, is accessed.

4. Exit.

   a. The normal exit is to the Control-Record module, RDCC.

      (1) The exit is taken after the LIST control record has been processed.

      (2) A listing of authorized users is produced.

   b. The exit if the LIST control record contains an invalid account option is to the Control-Record module, RDCC. The error message printed is: "SYNTAX ERROR, RECORD IGNORED".

   c. The exit if an access of the authorization file fails is to the Abnormal-Exit module, ABORT1.

      (1) The exit is taken if the open fails. For an abnormal return, the error message printed is: "ABORT: :USERLG FILE DOESN'T EXIST". For an error return, the error message printed is: "ABORT: CAN'T OPEN :USERLG".

      (2) The exit is taken if a read fails. The error message printed is: "ABORT: ABNORMAL I/O ON READ LOG SEQUEN".

5. Operation.

   a. Obtain the account option from the LIST control record (GEBFLD subroutine). If the option is invalid, do error processing using the PRTERR and the HELP subroutines and go to step f. In batch mode, first print an error message (PRNT subroutine).

   b. Print a heading for the listing of authorized users, using the PAGE and the PRNT subroutines.

   c. Process the account option if specified. Open the authorization file with direct access. Read a record for the account (RDLOGACNT subroutine) and print the user data (PRTLIST subroutine). Continue reading and printing until all records for the account have been processed. Go to step f.

   d. Process the file option if no account is specified. Open the authorization file with sequential access. Read a record (RDLOGS subroutine) and print the user data (PRTLIST subroutine). Continue reading and printing until the end-of-file is reached. Go to step f.

e. If the open of the authorization file fails or if a read to the file fails for other than the end-of-file condition described in step d, print an error message (PRNT subroutine) and branch to the Abnormal-Exit module, ABORT1.

f. Close the authorization file with SAVE specified and branch to the Control-Record module, RDCC.

6. Flowchart.

See Figure 5-3.

5.4.6 STATS.

1. Module Name - STATS.
2. Purpose. Process STATS option to summarize user statistics by reading the authorization file and printing data based on the file.
3. Entry.
   a. STATS is entered from Control Record.
   b. STATS has input.
      (1) Specification records may be read. In on-line mode, the prompt character has been set to a caret (>).
      (2) The authorization file, :USERLG, is accessed.
4. Exit.
   a. The normal exit is to the Control-Record module, RDCC.
      (1) The exit is taken after any specification records have been processed.
      (2) Listing output has been produced.
         (a) A summary of user statistics is produced.
         (b) In batch mode, if a specification record contained a syntax error, the error message printed is: "SYNTAX ERROR, RECORD IGNORED."
         (c) If no authorization-file record for a specified user existed, the error message printed is: "ABOVE USER NOT VALIDATED."
   b. The exit if a read of the M:SI DCB fails is to the Abnormal-Exit module, ABORT1. The error message printed is: "ABORT: READ M:SI ERROR."
   c. The exit if an access of the authorization file fails is to the Abnormal-Exit module, ABORT1.
      (1) The exit is taken if the open fails. For an abnormal return, the error message printed is: "ABORT: :USERLG FILE DOESN'T EXIST." For an error return, the error message printed is: "ABORT: CAN'T OPEN :USERLG."
      (2) The exit is taken if a read fails. If processing the account option, the error message printed is: "ABORT: ABNORMAL I/O ON LOG READ DIRECT." If processing the file option, the error message printed is: "ABORT: ABNORMAL I/O ON READ LOG SEQUEN."
5. Operation.
   a. Print a heading for the summary of user statistics, using the PAGE and PRNT subroutines.
   b. Read a specification record. If no specification record exists, go to step i.
   c. Format a key to access the authorization file using the account and the name options from the specification record (GACNTN subroutine). If an option is invalid, do error processing using the PRTERR and the HELP subroutines and go to step g. In batch mode, first print an error message (PRNT subroutine).
   d. Process the user option if specified. Open the authorization file with direct access. Read the record for the specified user (RDLOG subroutine) and print the user statistics (PRTSTAT subroutine). Go to step g.

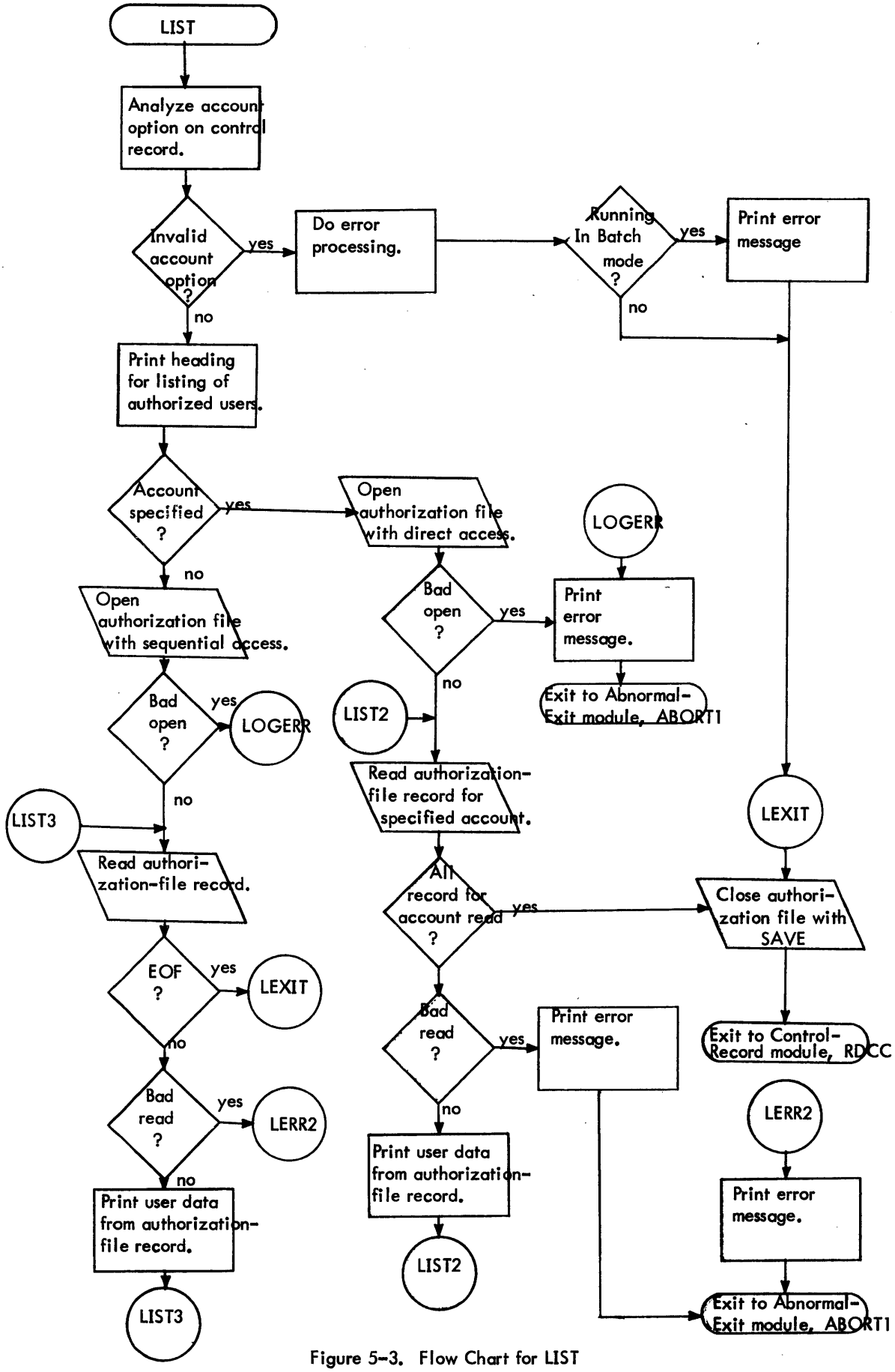


Figure 5-3. Flow Chart for LIST

e. Process the account option if specified. Read an authorization-file record for the specified account (RDLOGACNT subroutine) and print the user statistics (PRTSTAT subroutine). Continue reading and printing until all records for the account have been processed. Go to step g.

f. When an option is specified, if a read to the authorization file does not find the record specified or if a read fails for any other reason, print an error message (PRNT subroutine). If the read fails for any reason except the non-existent record condition, branch to the Abnormal-Exit module, ABORT1.

g. Read the next specification record and go to step c. When and EOD or an EOF is read, go to step k.

h. If a read of a specification record fails, print an error message (PRNT subroutine) and branch to the Abnormal-Exit module, ABORT1.

i. Process the file option. Open the authorization file with sequential access. Read a record (RDLOGS subroutine) and print the user statistics (PRTSTAT subroutine). Continue reading and printing until the end-of-file is reached. Go to step k. If a read fails for any other reason, print an error message (PRNT subroutine) and branch to the Abnormal-Exit module, ABORT1.

j. If the open of the authorization file fails, print an error message and branch to the Abnormal-Exit module, ABORT1.

k. Close the authorization file with SAVE specified and branch to the Control-Record module, RDCC.

6. Flowchart.

See Figure 5-4.

5.4.7 DELSTATS.

1. Module Name - DELET.

2. Purpose. Process DELSTATS option to delete (reset) user statistics by updating records in the authorization file.

3. Entry.

   a. DELSTATS is entered from Control-Record.

   b. DELSTATS has input.

      (1) Specification records may be read. In on-line mode, the prompt character has been set to a caret (>).

      (2) The authorization file, :USERLG, is accessed.

4. Exit.

   a. The normal exit is to the Control-Record module, RDCC.

      (1) The exit is taken after any specification records have been processed.

      (2) The authorization file, :USERLG, has been updated according to the options on the specification records.

      (3) Error messages may have been printed.

         (a) In batch mode, if a specification record contained a syntax error, the message printed is: "SYNTAX ERROR, RECORD IGNORED."

         (b) If no authorization-file record for a specified user existed, the message printed is: "**WARNING** ABOVE USER NOT FOUND."

   b. The exit if a read of the M:SI DCB fails is to the Abnormal-Exit module, ABORT1. The error message printed is: "ABORT: READ M:SI ERROR."

   c. The exit if an access of the authorization file fails is to the Abnormal-Exit module, ABORT1.

      (1) The exit is taken if the open fails. For an abnormal return, the error message printed is: "ABORT: :USERLG FILE DOESN'T EXIST." For an error return, the error message printed is: "ABORT: CAN'T OPEN :USERLG."

      (2) The exit is taken if a read fails. If processing the user or the account option, the error message printed is: "ABORT: ABNORMAL I/O ON LOG READ DIRECT." If processing the file option, the error message printed is: "ABORT: ABNORMAL I/O ON READ LOG SEQUEN."

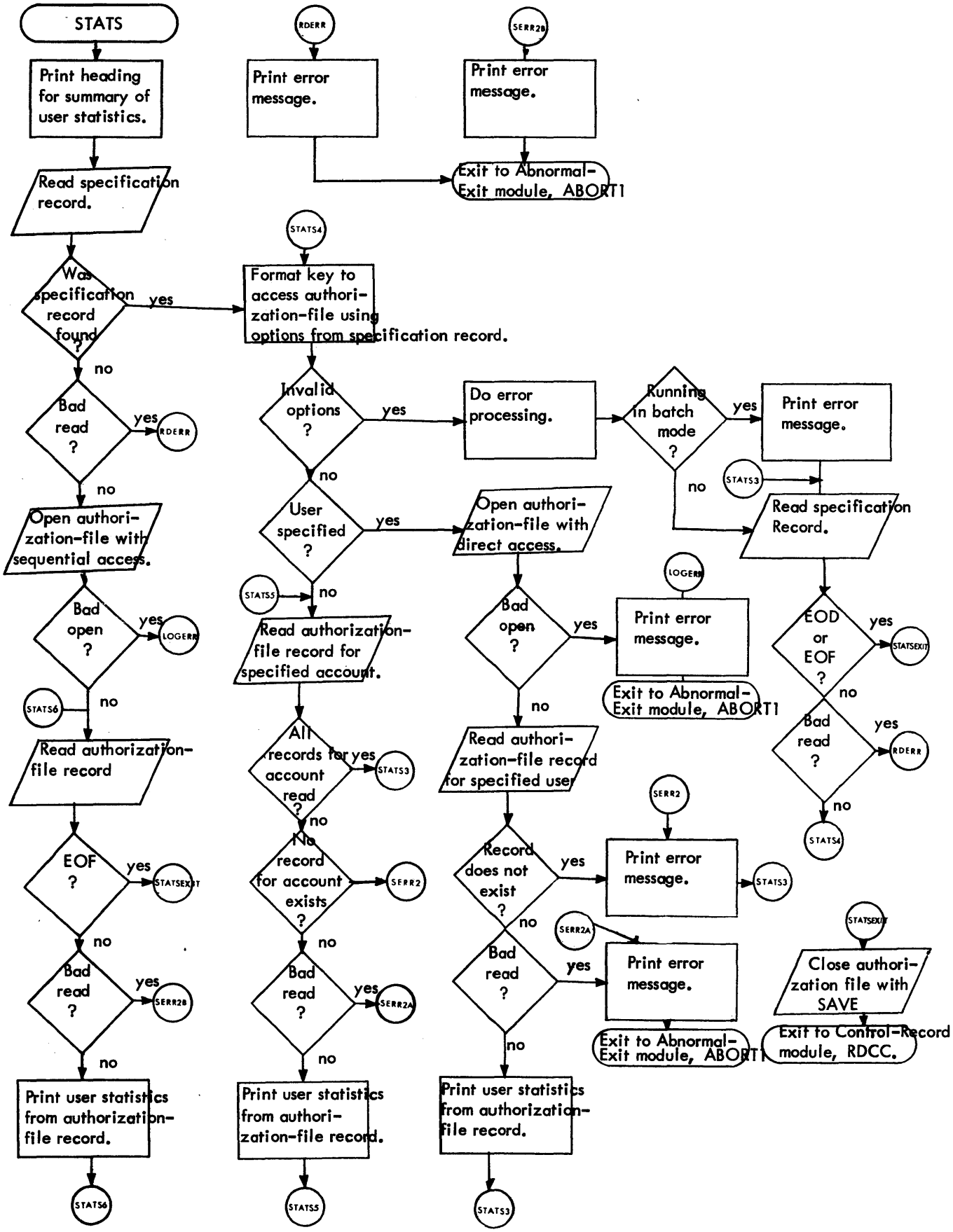


Figure 5-4. Flow Chart for STATS

(3) The exit is taken if a write fails. The error message printed is: "ABORT: ABNORMAL I/O ON LOG REWRITE SEQUEN."

5. Operation.

a. Read a specification record. If a specification record exists, go to step c.

b. Process the file option. Open the authorization file with sequential access. Read a record (RDLOGS subroutine), reset user statistics in the record (RESETLOG subroutine), and write the updated record (REWRLOGS subroutine). Continue reading and updating until the end-of-file is reached. Go to step j. If a read fails, print an error message (PRNT subroutine) and branch to the Abnormal-Exit module, ABORT1.

c. Open the authorization file with direct access.

d. Format a key to access the authorization file using the account and the name options from the specification record (GACNTN subroutine). If an option is invalid, do error processing using the PRTERR and the HELP subroutines and go to step h. In batch mode, first print an error message (PRNT subroutine).

e. Process the user option if specified. Read the authorization-file record for the specified user (RDLOG subroutine), reset user statistics in the record (RESETLOG subroutine), and write the updated record (REWRLOG subroutine). Go to step h.

f. Process the account option if specified. Read an authorization-file record for the specified account (RDLOGACNT subroutine), reset user statistics in the record (RESETLOG subroutine), and write the updated record (REWRLOG subroutine). Continue reading and updating until all records for the account have been processed. Go to step h.

g. When an option is specified, if a read to the authorization file does not find the record specified or if a read fails for any other reason, print an error message (PRNT subroutine). If the read fails for any reason except the non-existent-record condition, branch to the Abnormal-Exit module, ABORT1.

h. Read the next specification record and go to step d. When an EOD or an EOF is read, go to step j.

i. If a read of a specification record fails, if the open of the authorization file fails, or if a write to the authorization file fails, print an error message (PRNT subroutine) and branch to the Abnormal-Exit module, ABORT1.

j. Close the authorization file with SAVE specified and branch to the Control-Record module, RDCC.

6. Flowchart.

See Figure 5-5.

5.4.8 PASSWORDS.

1. Module Name - PASSWORD.

2. Purpose. Process PASSWORDS option to list passwords of user files in a specified account.

3. Entry.

a. The PASSWORDS module is branched to from the Control-Record module.

b. The PASSWORDS module requires input.

(1) The PASSWORDS control record must have been read. In on-line mode, the prompt character has been set to a caret (>).

(2) The file directory for the specified account is accessed.

4. Exit.

a. The exit is to the Control-Record module, RDCC.

b. A listing of passwords for files in the specified account is output unless an error occurred in processing the option.

c. Error messages may have been printed.

(1) If the PASSWORDS control record contained an invalid account option, the error message printed is: "SYNTAX ERROR, RECORD IGNORED."

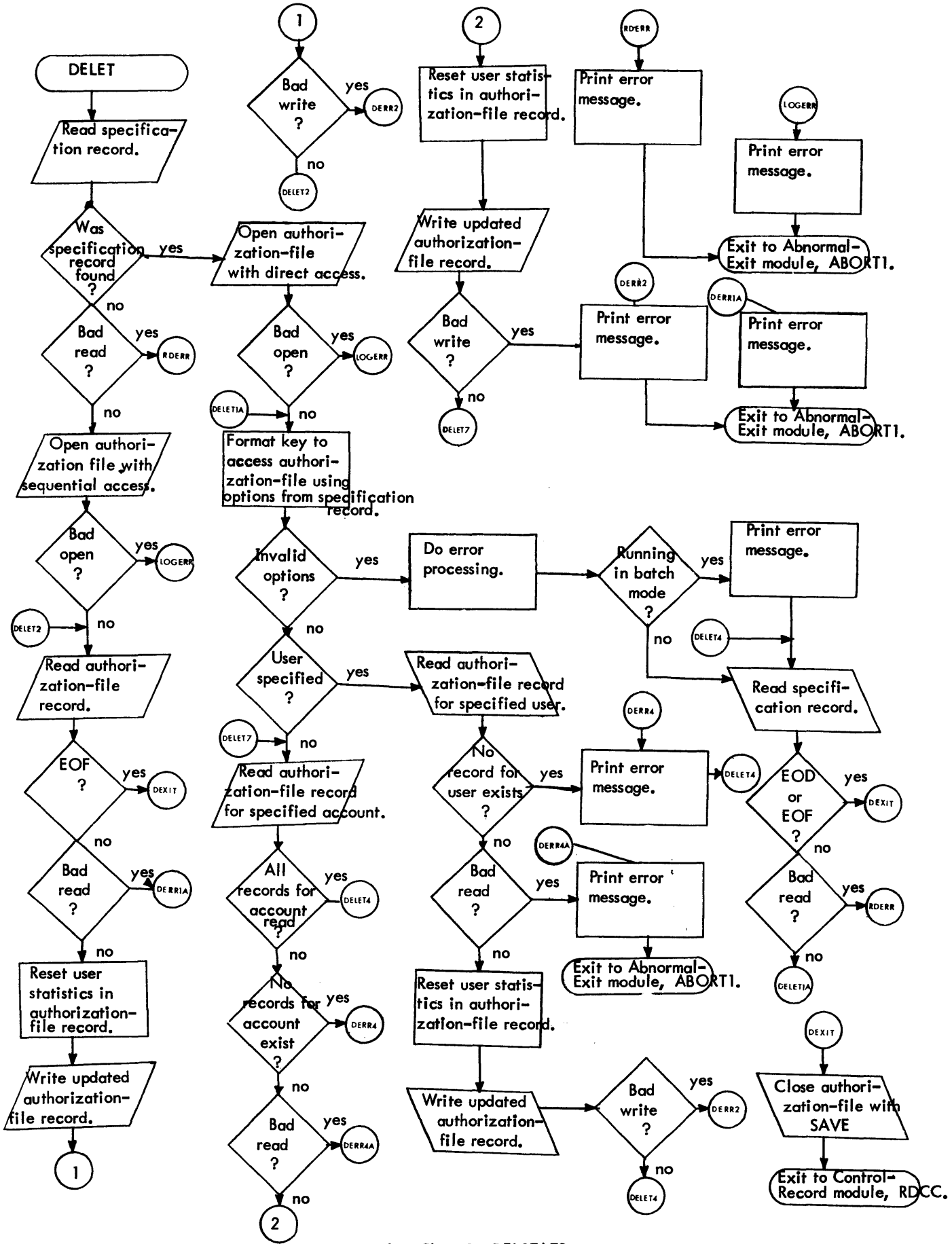


Figure 5-5. Flow Chart for DELSTATS

(2) If the open for the specified account, with next file option, fails, the error message printed is: "ABNORMAL I/O ON OPEN NEXT."

5. Operation.

a. Find the file name entry in the M:EI DCB (FNDCOD subroutine) and zero the number of significant entries in the file name parameter. In on-line mode, the file-name parameter may be stored into directly since the DCB changed is an unprotected dummy DCB, not the actual DCB used by the BTM executive. In batch mode, first issue the M:SYS call, save the run flags from the job information table, and reset the run flags to Monitor before storing into the file name parameter. After the store, load a program status doubleword to return to slave mode.

b. Obtain the next EBCDIC field from the control record (GEBFLD subroutine). If no valid field is found, do error processing using the PRTERR and HELP subroutine and go to step f. In batch mode, first print an error message (PRNT subroutine).

c. Do form alignment on the output listing (PAGE subroutine).

d. Issue an open for the specified account, with the next-file option. If the end-of-files is encountered, go to step f. For any other failure, print an error message (PRNT subroutine) and go to step f.

e. Find the file name in the variable-length parameters of the M:EI DCB (FNDCOD subroutine), move the name to the output line, find the file password (FNDCOD subroutine), move the description, ***NONE*** or the password, in EBCDIC if printable or in hexadecimal if non-printable, to the output line. Print the line (PRNTV subroutine). Close the M:EI DCB with SAVE specified and go to step d.

f. Close the M:EO DCB with SAVE specified and branch to the Control-Record module. In batch mode, first issue the M:SYS call, restore the run flags saved in step a, and load a program status doubleword to return to slave mode.

5.4.9 Input-Syntax-Error.

1. Module Name - L6 and L7.

2. Purpose. Process invalid control-record option.

3. Entry.

a. Input-Syntax-Error is entered from Control-Record at L6 and from USERS at L7.

b. An invalid control record must have been read.

4. Exit.

a. The normal exit is to the Control-Record module, RDCC. In batch mode, a set of a control record and the specification records for the control record have been read.

b. The exit if no more control records exist is to the Normal-Exit module, EXIT.

c. A dollar sign ($) is printed under the erroneous field. In on-line mode, the correct form of the required control command is printed. In batch mode, the error message printed is: "SYNTAX ERROR, RECORD IGNORED."

5. Operation.

a. At entry point L6, print a dollar sign ($) under the erroneous field (PRTERR subroutine). Print the record first if in on-line mode with file input.

b. In on-line mode, print the correct form of the required command (HELP subroutine) and branch to the Control-Record module, RDCC.

c. At entry point L7 for batch mode, print an error message (PRNT subroutine), read records from M:SI until an EOD is read, then branch to the Control-Record module, RDCC. If an EOF is read, branch to the Normal-Exit module, EXIT. If a read fails, print an error message (PRNT subroutine) and branch to the Abnormal-Exit module, ABORT1.

5.4.10 Abnormal-Exit.

1. Module Name - ABORT1.

2. Purpose. Process abnormal termination of job step.
3. Entry. Abnormal-Exit is entered from START, Control-Record, USERS, KILL, LIST, STATS, and DELSTATS.
4. Exit. Abnormal-Exit errors the job after closing files to be saved.
5. Operation. Close, with SAVE specified, the M:LO DCB, the M:SI DCB, and the authorization file. Error the job.

5.4.11 Normal Exit.

1. Module Name - EXIT.
2. Purpose. Process normal termination of job step.
3. Entry. Normal-Exit is entered from START, Control-Record, and Input-Syntax-Error.
4. Exit. Normal-Exit terminates the job step after closing files to be saved.
5. Operation. Close, with SAVE specified, the M:LO DCB, the M:SI DCB, and the authorization file. Issue the M:EXIT call to terminate the job.

5.5 SUBROUTINE ANALYSIS.

Subroutines are in alphabetic order. Register 10 is the link register for all subroutines.

5.5.1 CVTDEC.

1. Subroutine Name - CVTDEC.
2. Calling Modules - LIST, STATS.
3. Purpose. Convert a binary value to EBCDIC decimal.
4. Entry.
   a. Register 10 contains the return address.
   b. Register 1 contains the binary value to be converted.
5. Exit.
   a. CVTDEC returns to the calling routine after converting the value.
   b. Registers 12 and 13 contain the EBCDIC decimal result.
6. Operation. Save registers 5-11. Convert the binary value in register 1 to EBCDIC decimal. Precede the EBCDIC decimal result in registers 12 and 13 by a minus sign if negative. Restore registers 5-11 and branch to the return address in register 10.

5.5.2 CVTNSTOR.

1. Subroutine Name - CVTNSTOR.
2. Calling Module - STATS.
3. Purpose. Convert binary value to EBCDIC decimal and store result in proper format in specified output buffer.
4. Entry.
   a. Register 10 contains the return address.
   b. Register 1 contains the binary value to be converted.
   c. Register 7 contains the address of the output buffer.
   d. Register 6 contains the number of bytes in the output field less one.
   e. Register 8 contains the number of characters to the right of the decimal point. Zero indicates no decimal point.

5. Exit.

   a. CVTNSTOR returns to the calling routine after storing the result in the specified buffer.

   b. The specified buffer contains an EBCDIC decimal value in the specified format.

6. Operation. Save all registers. Convert binary value to EBCDIC decimal (CVTDEC subroutine). Store the result in the specified format in the specified buffer. Restore all registers and branch to the return address in register 10.

5.5.3 DATER.

1. Subroutine Name - DATER.
2. Calling Modules - USERS, DELSTATS.
3. Purpose. Obtain binary date.
4. Entry - Register 10 contains the return address.
5. Exit.

   a. DATER returns to the calling routine after calculating date.

   b. Register 9 contains binary date.

6. Operation. Calculate binary date the first time the subroutine is entered. Save registers 4-7. Obtain time. Save register 10. Save all registers. Convert time to binary by branching into GHFLD subroutine. Restore all registers in GHFLD subroutine and return. Restore register 10. Restore registers 4-7. Load binary date into register 9. Store binary date in DBUF so date need not be recalculated. Branch to return address in register 10.

5.5.4 FNDCOD.

1. Subroutine Name - FNDCOD.
2. Calling Module - PASSWORDS.
3. Purpose. Find a code in the variable-length parameter of a DCB.
4. Entry.

   a. Register 10 contains the return address.

   b. Register 1 contains the code to be found.

   c. Register 2 points to the variable-length parameters of a DCB.

5. Exit.

   a. FNDCOD returns to the calling routine after searching the variable-length parameters of a DCB for the specified code.

   b. Register 7 points to the first data word of the specified variable-length parameter or, if the specified code was not found, to blanks.

6. Operation. Save register 3 so that register 3 may be used as a work register. Search the variable-length parameters for the specified code. If found, increment register 7 to point to the first data word of the parameter. If not found, load the address of blanks into register 7. Restore register 3 and branch to the return address in register 10.

5.5.5 GACNTN.

1. Subroutine Name - GACNTN.
2. Calling Modules - KILL, STATS, DELSTATS.
3. Purpose. Format a key to access the authorization file using the account and the name options from the specification record.
4. Entry.

   a. Register 10 contains the return address.

b. Specification record must have been read.

5. Exit.

a. GACNTN returns to the calling routine after processing the account and the name options.

b. Data is returned.

(1) The condition codes are set:

CC1 on means no option was found.

CC2 on means an invalid option was found. (See CC4 for an additional error indication.)

CC3 on means a valid option was found.

CC4 on means an option field was too long.

(2) A key to access the authorization file is formatted unless no valid option was found.

c. Exit to the return address if valid options are found or if an invalid account option is found. Exit to the return address plus one if an invalid name option is found.

6. Operation.

a. Save all registers. Obtain the account option from the specification record (GEBFLD subroutine) and add the option to the key to be used to access the authorization file. If the account option is invalid, go to step b. Otherwise, obtain the name option, if any, from the specification record (GEBFLD subroutine) and add the option to the key. If the name option is invalid, increment the return address saved by one and go to step b.

b. Restore all registers, set the condition codes, and branch to the address in register 10.

5.5.6 GDFLD and GHFLD.

1. Subroutine Names - GDFLD and GHFLD.

2. Calling Modules. USERS calls both subroutines. DELSTATS calls GHFLD.

3. Purpose. GDFLD obtains the value of the next decimal field on the USERS specification record. GHFLD obtains the value of the next hexadecimal field on the USERS specification record.

4. Entry.

a. Register 10 contains the return address.

b. USERS specification record must have been read.

c. FLDPTR contains the record position at which to begin searching.

d. BUFLEN contains the record size.

5. Exit.

a. GDFLD and GHFLD return to the calling routine after attempting to determine the value of the desired type of field on the USERS specification record.

b. Data is returned.

(1) The condition codes are set:

CC1 on means no field found.

CC2 on means invalid field found.

CC3 on means valid field found.

(2) NFLDBUF contains the value of the valid field found.

6. Operation.

a. Save all registers. Set up an execute instruction to convert a field on the USERS specification record to decimal, for GDFLD, or to hexadecimal, for GHFLD. Obtain the next EBCDIC field from the record. If no valid field is found, go to step b. Otherwise, convert the field to a binary value. Convert the binary value to a decimal value for GDFLD ot to a hexadecimal value for GHFLD.

b. Restore all registers, set the condition codes, and branch to the return address in register 10.

5.5.7 GEBFLD.

1. Subroutine Name - GEBFLD.
2. Calling Modules - Control-Record, USERS, KILL, LIST, STATS, DELSTATS, PASSWORDS.
3. Purpose. Obtain the next EBCDIC field from a control or a specification record.
4. Entry.
   a. Register 10 contains the return address.
   b. Control or specification record must have been read.
   c. FLDPTR contains the record position at which to begin analysis.
   d. BUFLEN contains the record size.
5. Exit.
   a. GEBFLD returns to the calling routine after attempting to obtain the next EBCDIC field from the control or the specification record.
   b. Data is returned.
      (1) The condition codes are set:

      CC1 on means no field found.

      CC2 on means invalid field found.

      CC3 on means valid field found.

      (2) FLDBUF contains the EBCDIC field found.
      (3) FLDBUF1 contains an additional field in parentheses, if any such field existed. The field is stored without the parentheses.
      (4) FLDPTR is incremented to the next record position to be analyzed.
      (5) Registers.
         (a) Register 13 contains the length of the field in FLDBUF1.
         (b) Register 14 contains the length of the field in FLDBUF.
6. Operation.
   a. Save registers 1-4 so that the registers may be used as work registers. Store the next EBCDIC field from the control or the specification record into FLDBUF. If no valid field is found, go to step b. Otherwise, store the next field in parentheses, if any such field exists, into FLDBUF1, without the parentheses. Load register 13 with the length of the field in FLDBUF1.
   b. Load register 14 with the length of the field in FLDBUF. Increment FLDPTR to the next record position to be analyzed. Restore registers 1-4, set the condition codes, and branch to the return address in register 10.

5.5.8 GHFLD.

See GDFLD.

5.5.9 HELP.

1. Subroutine Name - HELP.
2. Calling Modules - USERS, KILL, LIST, STATS, DELSTATS, PASSWORDS, Input-Syntax-Error.
3. Purpose. Print the correct form of the required control or specification record if in on-line mode.
4. Entry.
   a. Register 10 contains the return address.

b. Register 1 contains the level of help needed.

1 means reading a control record.

2 means reading a USERS record.

3 means reading a USERS record.

4 means reading a DELSTATS record.

5 means reading a KILL record.

6 means reading a STATS record.

7 means reading a LIST record.

8 means reading a PASSWORD record.

5. Exit.

a. HELP returns to the calling routine after printing the correct form of the required control or specification record if in on-line mode.

b. Condition code 1 is set if no message was printed. Otherwise, the correct form of the required control or specification record is printed.

6. Operation.

a. Save registers 1-10. If in batch mode or if the record format has already been printed, go to step b. Otherwise, print the correct form of the required control or specification record.

b. Restore registers 1-10, set the condition codes, and branch to the return address in register 10.

5.5.10 INITLOG.

1. Subroutine Name - INITLOG.
2. Calling Module - USERS.
3. Purpose. Format field defaults for a new authorization-file record.
4. Entry - Register 10 contains the return address.
5. Exit.

a. INITLOG returns to the calling routine after setting field defaults for the authorization-file record.

b. All optional fields of the authorization-file record being formatted are set to defaults.

6. Operation. Save registers 1-3 so that the registers may be used as work registers. Initialize all optional fields of the authorization-file record being formatted to defaults. Restore registers 1-3 and branch to the return address in register 10.

5.5.11 PAGE.

1. Subroutine Name - PAGE.
2. Calling Modules - START, LIST, STATS, PASSWORDS.
3. Purpose. Do form alignment on output listing.
4. Entry - Register 10 contains the return address.
5. Exit.

a. PAGE returns to the calling routine after doing form alignment.

b. In batch mode, the output listing is advanced to top of page. In on-line mode, 3 lines are skipped on the terminal listing.

6. Operation. In batch mode, advance output listing to top-of-page. In on-line mode, save registers 0 and 1, skip 3 lines on the terminal listing, restore registers 0 and 1. Branch to the return address in register 10.

5.5.12 PRNT, PRNTV.

1. Subroutine Names - PRNT, PRNTV.
2. Calling Modules - START, Control-Record, USERS, KILL, LIST, STATS, DELSTATS, PASSWORDS, Input-Syntax-Error.
3. Purpose. PRNT issues a TEXTC message to the M:LO DCB. PRNTV issues a TEXT message to the M:LO DCB.
4. Entry.
   a. Register 10 contains the return address.
   b. For PRNT, register 1 contains the address of a TEXTC message on a word boundary. For PRNTV, register 1 contains the address of a TEXT message.
   c. For PRNTV, register 2 contains the length in bytes of the message.
5. Exit.
   a. PRNT and PRNTV return to the calling routine after writing a message.
   b. A message has been printed through the M:LO DCB.
6. Operation. Save registers 2 and 3 so that the registers may be used as work registers. For PRNT, load register 2 with the byte count of the message and register 3 with a byte displacement of 1. For PRNTV, load register 3 with a byte displacement of zero. Write the message through the M:LO DCB. Restore registers 2 and 3 and branch to the return address in register 10.

5.5.13 PRTERR.

1. Subroutine Name - PRTERR.
2. Calling Modules - USERS, KILL, LIST, STATS, DELSTATS, PASSWORDS, Input-Syntax-Error.
3. Purpose. Print a dollar sign ($) under an invalid field on a control or specification record.
4. Entry - Register 10 contains the return address.
5. Exit.
   a. PRTERR returns to the calling routine after printing the dollar sign.
   b. A dollar sign is printed under the invalid field on a control or specification record. In on-line mode with file input, the record was printed first.
6. Operation. Save register 10 so that register 10 may be used as a work register. Print a dollar sign under the invalid field on the control or specification record (PRNTV subroutine). In on-line mode with file input, first print the record (PRNTV subroutine). Restore register 10 and branch to the return address in register 10.

5.5.14 PRTLIST.

1. Subroutine Name - PRTLIST.
2. Calling Module - LIST.
3. Purpose. List the parameters from an authorization-file record.
4. Entry.
   a. Register 10 contains the return address.
   b. LOGBUF1 contains an authorization-file record.
5. Exit.
   a. PRTLIST returns to the calling routine after listing the parameters from an authorization-file record.
   b. Account, name, password, secondary-storage limits, priority, monitor-service authorization, date and time of last change, and extended accounting are listed.
6. Operation. Save all registers. Move account, name, extended accounting, and password to the ouput line. Format the date and time of the last change, using the CVTDEC subroutine.

Move the priority and monitor-service authorization indicators to the output line. Format the secondary-storage limits, using the CVTDEC subroutine. Print the parameters (PRNTV subroutine). Restore all registers and branch to the return address in register 10.

5.5.15 PRTSTAT.

1. Subroutine Name - PRTSTAT.
2. Calling Module - STATS.
3. Purpose. Print statistics from an authorization-file record.
4. Entry.
   a. Register 10 contains the return address.
   b. LOGBUF1 contains an authorization-file record.
5. Exit.
   a. PRTSTAT returns to the calling routine after printing the statistics from an authorization-file record.
   b. Account, name, total batch and on-line job sessions, job connect time, time breakdown, and secondary storage used and authorized are printed.
6. Operation. Save all registers. Move account and name to the output line. Format numeric fields, using the CVTNSTOR subroutine. Print statistics (PRNTV subroutine). Restore all registers and branch to the return address in register 10.

5.5.16 RDLOG and RDLOGS.

1. Subroutine Names - RDLOG, RDLOGS.
2. Calling Modules. USERS, LIST, STATS, DELSTATS call RDLOG. LIST, STATS, and DELSTATS call RDLOGS.
3. Purpose. Read an authorization-file record. RDLOG reads a specified key. RDLOGS reads sequentially.
4. Entry.
   a. Register 10 contains the return address.
   b. For RDLOG, register 1 contains the address of the key to be read.
5. Exit.
   a. RDLOG and RDLOGS exit to the calling routine after issuing a read to the authorization file.
   b. Data is returned.
      (1) LOGBUF1 contains the authorization-file record read.
      (2) Condition code 1 is set if the read failed. In this case register 13, byte 3, contains the abnormal or error code.
6. Operation. Save registers 4-11. Set the abnormal and error addresses in the M:EO DCB to RDLOG1. Issue the read to the authorization file. If the read fails, load the abnormal or error code from register 10 into register 13 and set condition code 1. Restore registers 4-11 and branch to the return address in register 10.

5.5.17 RDLOGACNT.

1. Subroutine Name - RDLOGACNT.
2. Calling Modules - LIST, STATS, DELSTATS.
3. Purpose. Read an authorization-file record for a specified account.
4. Entry.
   a. Register 10 contains the return address.
   b. ACCNTMP contains the account to be read.

5. Exit.

   a. RDLOGACNT returns to the calling routine after issuing a read to the authorization file.

   b. Data is returned.

      (1) LOGBUF1 contains the authorization-file record read.

      ACCNT1 contains the key read.

      (2) Condition code 1 is set if the read failed. In this case, register 13, byte 3, contains the abnormal or error code.

      (3) Condition code 2 is set if no more records for the account exist.

6. Operation. Save all registers. For the first read for the account, read specifying a key (RDLOG subroutine). If the read returns the no-key-exists abnormal code but the key accessed is for the specified account number and the key specified contains no name, read specifying the last key accessed (RDLOG subroutine). Once a record for the account has been processed, read the next record sequentially (RDLOGS subroutine). If no more records for the account exist, set condition code 2. If a read fails, set condition code 1 and load the abnormal or error code into register 13. Restore all registers and branch to the return address in register 10.

5.5.18 RDLOGS.

See RDLOG.

5.5.19 RESETLOG.

1. Subroutine Name - RESETLOG.
2. Calling Module - DELSTATS.
3. Purpose. Reset statistics for an authorization-file record.
4. Entry.
   a. Register 10 contains the return address.
   b. Register 1 contains the address of the buffer in which the statistics are to be reset.
5. Exit.
   a. RESETLOG returns to the calling routine after resetting statistics for an authorization-file record.
   b. The authorization-file record formatted in the buffer pointed to by register 1 has all statistic fields reset except secondary storage used.
6. Operation. Save registers 3-10. Obtain the binary date (DATER subroutine) and move the date to the authorization-file record being formatted. Reset all statistics in the record buffer except secondary storage used. Restore registers 3-10 and branch to the return address in register 10.

5.5.20 REWRLOG and REWRLOGS.

See WRLOG.

5.5.21 UCHKFLD.

1. Subroutine Name - UCHKFLD.
2. Calling Module - USERS.
3. Purpose. Set flag for Monitor-service specified on a USERS control record.
4. Entry.
   a. Register 10 contains the return address.
   b. FLDBUF contains the field to be analyzed.
   c. Register 14 contains the length of the field in FLDBUF.

d. Condition code 2 is set if the field is invalid.

5. Exit.

   a. UCHKFLD returns to the calling routine after analyzing an option on the USERS control record.

   b. Data is returned.

      (1) FLGS has a bit set for a Monitor-service specified on the USERS control record.

      (2) Condition code 2 is set if the field on the USERS control record is invalid.

6. Operation. If condition code 2 is set, branch immediately to the return address in register 10. Otherwise find the option in a jump table, set the bit in FLGS for the option, and reset the condition codes. If the option cannot be found in the jump table, set condition code 2. Branch to the return address in register 10.

5.5.22 WRLOG, REWRLOG, and REWRLOGS.

1. Subroutine Names - WRLOG, REWRLOG, REWRLOGS.

2. Calling Modules. USERS calls WRLOG and REWRLOG. DELSTATS calls REWRLOG and REWRLOGS.

3. Purpose. WRLOG writes an authorization-file record. REWRLOG rewrites an authorization-file record with a key. REWRLOGS rewrites an authorization-file record without a key.

4. Entry.

   a. Register 10 contains the return address.

   b. For WRLOG and REWRLOG, register 1 contains the address of the key, a word address.

   c. LOGBUF contains the record to be written.

5. Entry.

   a. WRLOG, REWRLOG, and REWRLOGS return to the calling routine after issuing a write to the authorization file.

   b. Condition code 1 is set if the read fails. In this case, register 13, byte 3, contains the abnormal or error code.

6. Operation. Save registers 4-11. Obtain the binary date (DATER subroutine) and store the date in the authorization-file record buffer. Set the abnormal and error addresses in the M:EO DCB to WRABN. Issue the write to the authorization file. For WRLOG, write with a key and the NEWKEY option specified. For WRLOG, write with a key and the NEWKEY option specified. For REWRLOG, write with a key and the ONEWKEY option specified. For REWRLOGS, write with no key specified. If the write fails, load the abnormal or error code from register 10 into register 13. Restore registers 4-11. Reset the condition codes or, if the write failed, set condition code 1. Branch to the return address in register 10.

5.6 AUTHORIZATION FILE, :USERLG.

See Chapter 20 of the BPM Technical Manual (Publication Number 901528).

6.0 FPURGE PROCESSOR

6.1 FUNCTIONAL OVERVIEW.

The File Purge processor (FPURGE) primary function is to maintain and control user's RAD and/or disk pack files, and to protect the integrity of these files by periodically creating back-up files on tape in the event of an unrecoverable system failure. (Hardware or Software).

6.2 INTERFACES

6.2.1 Hardware Requirements

The file purge processor operates with a basic BPM configuration of one magnetic tape unit, a card reader, a typewriter, a line printer, and at least one rad or disk pack.

6.2.2 Operational Requirements.

For FPURGE to gain access to the account directory and open all files, it must run in the master mode. FPURGE utilizes the M:SYS CAL which is legal only if the job is running under the :SYS account.

6.3 OPERATIONAL OVERVIEW

The FPURGE control command is read through the C device. It is then decoded, and the necessary flags are set. FPURGE will access the RAD(s) and/or disk packs, or tape(s) by accounts in ascending order. The files to be accessed are set up by the options selected.

To protect the integrity of the operating system, FPURGE will ignore all requests to process files under the accounts :SYS, :BTM, and COBLIB. These accounts will remain as they are, unless the SKIP option us used. (See Table 6-1.)

File structure will remain the same throughout all functions. (e.g. mode, access).

Tapes are written in Monitor labeled format. A new file will be opened and closed for each user's file copied to tape. Tapes will default to 9T, unless they are assigned through M:EI, or M:EO, to DEVICE 7T.

The format of the control command is

```
!FPURGE function,option
```

where function is

SAVE will take user's files from Rad and/or disk packs, and copy them to tape.

RES user's files that were previously copied to tape, will now be restored to Rad or disk pack.

LOG the names of the user's files are to be logged by their account number on the line printer.

PURGE user's files are to be deleted.

On the above four functions, if neither the ALL nor SELECT option is specified, then ALL will be assumed by default.

The options for each function are defined in Table 6-1.

Data cards are used to define the files to be processed. Each function requires its own set of data cards. Their format is as follows:

Card Columns

1 - 8	Account number (left-justified)
9	Must be blank
10 - 12	All (optional). If ALL is specified, all files within the names account number are processed and the file name field (card columns 14-44) must be blank.
13	Must be blank
14 - 44	Filename (left-justified)
45 - 80	not used

Each set of data cards must be terminated by a !EOD control command.

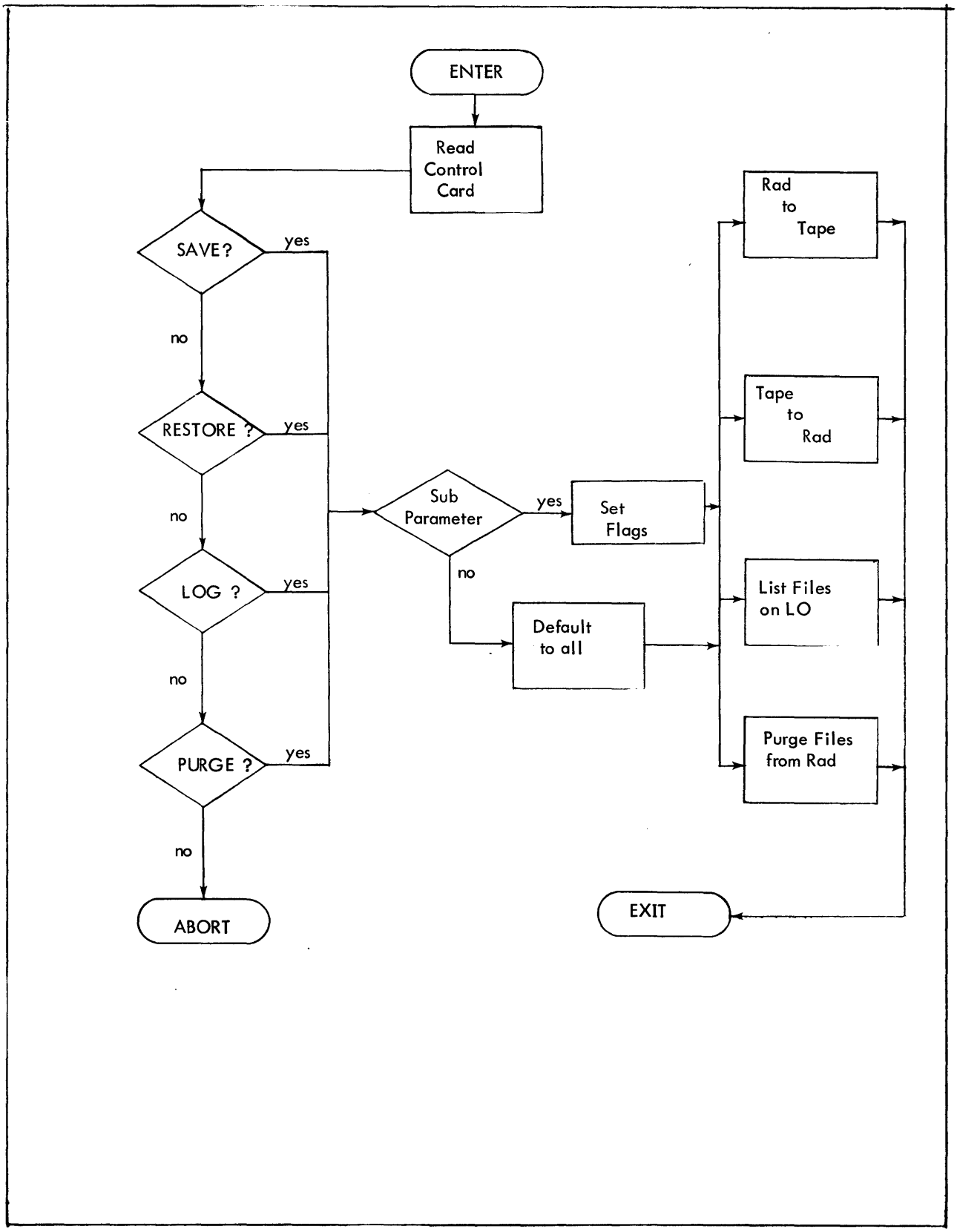


Figure 6-1. Flow Chart of Major Functions

6.4 MODULE ANALYSIS

6.4.1 CONTROL CARDS

1. Routine name:
   CCI

2. Purpose:
   To decode control cards.

3. Entry:
   Start

4. Exit:
   Exits to STRPURG.

5. Operation:
   This routine decodes all control cards and sets the flags needed.

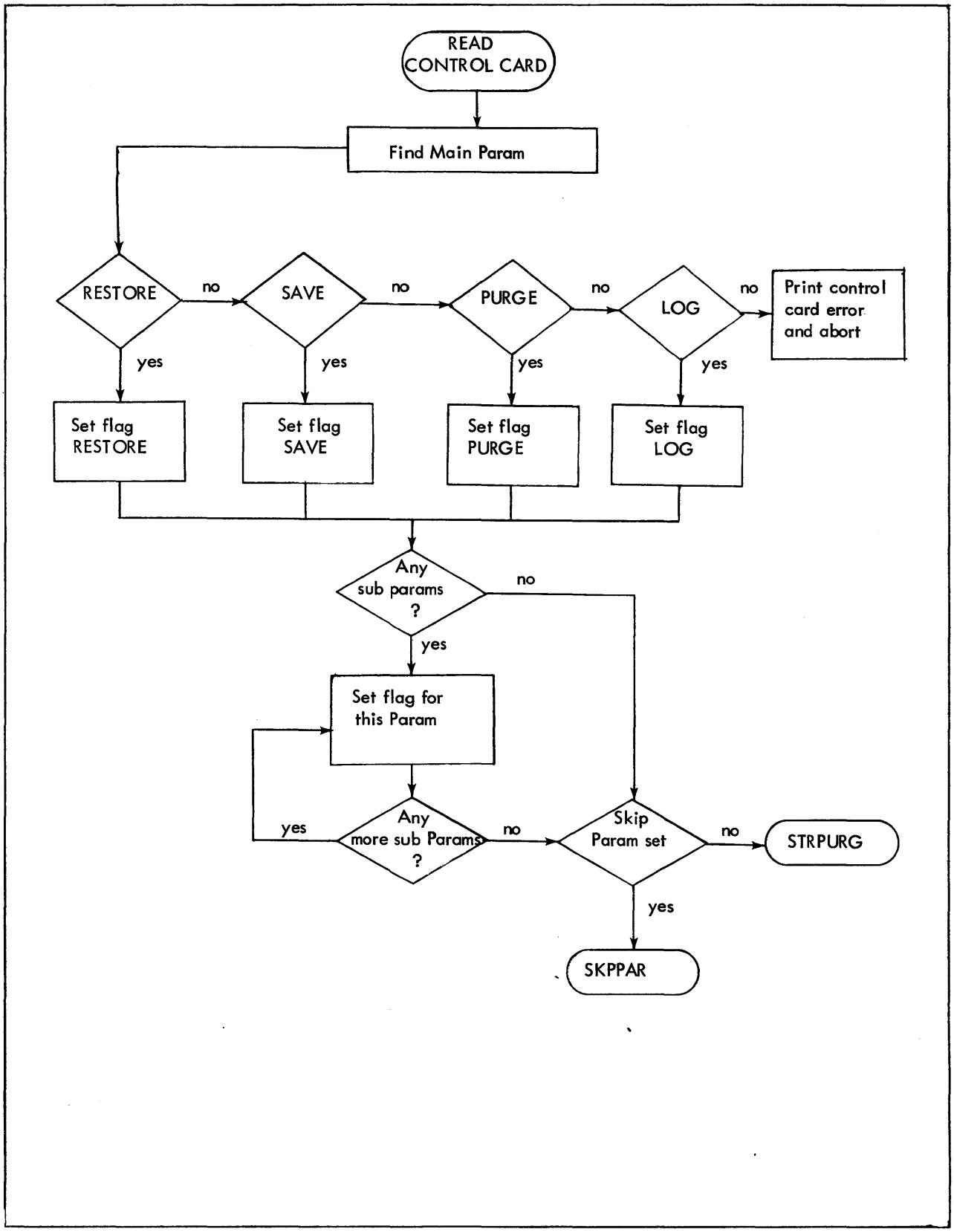


Figure 6-2. Flow Chart of CCI

6.4.2 SKPPAR.

1. Purpose:
   to determine how many accounts to skip.

2. Entry:
   Entered from Control Card Routine.

3. Exit:
   Exits to STRPURG

4. Operation:
   All accounts named will be skipped, and if the skip option is used, accounts :SYS, :BTM, and COBLIB will be included in the specified option. Up to 110 accounts can be used, but SKIP must be the first entry on every card and these cards must be terminated by a .END card.

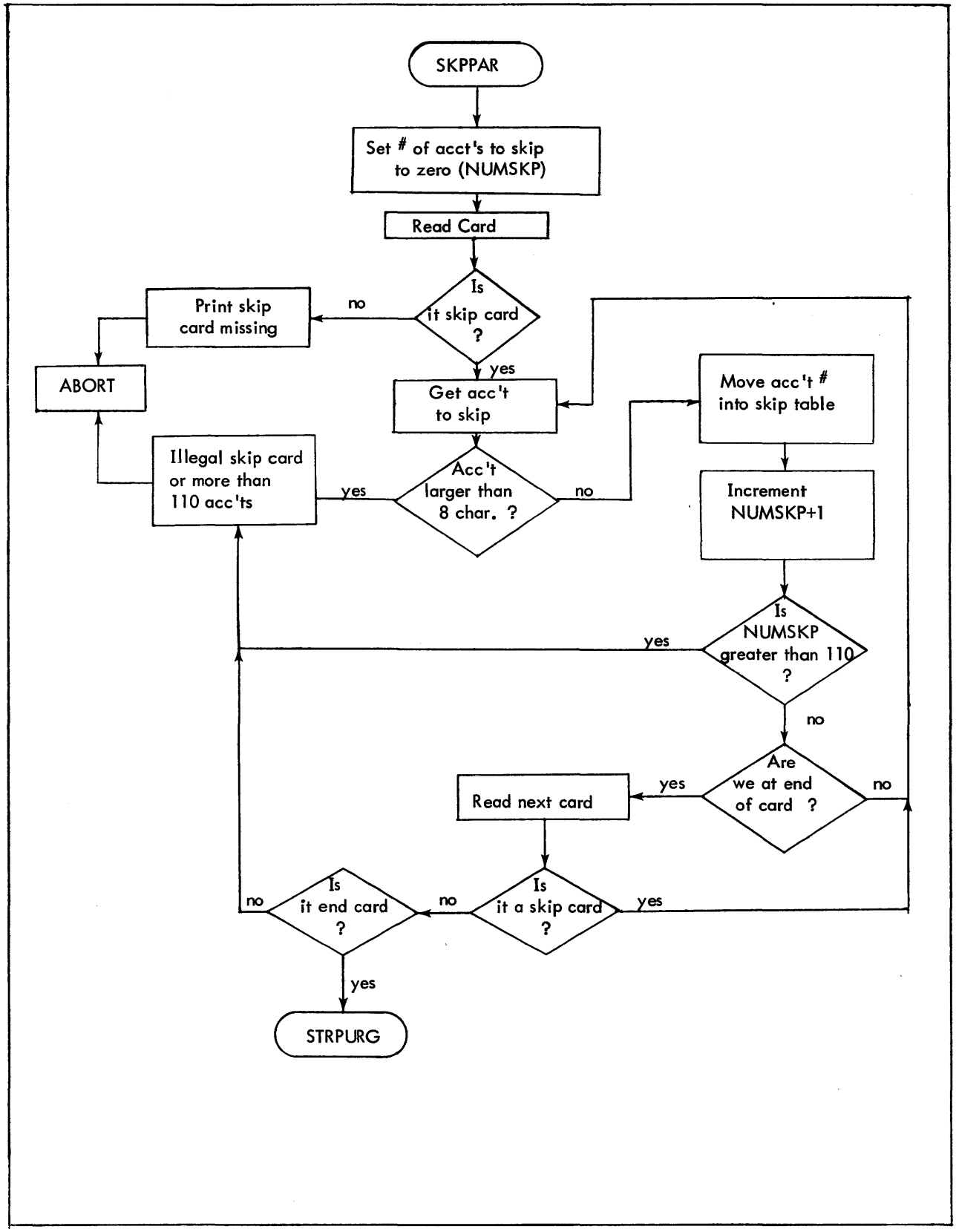


Figure 6-3. Flow Chart of SKPPAR

6.4.3 STRPURG.

1. Purpose:
   Check whether it is a Save or Restore, and select or all.

2. Entry:
   Entered from Control Card Routine.

3. Exit:
   Exits to LGSVPG or Restore.

4. Operation:
   This routine determines whether or not a select option or all option is used. If select is used, a link to sore is made, to make sure the accounts and files are sorted and put into a dummy file.

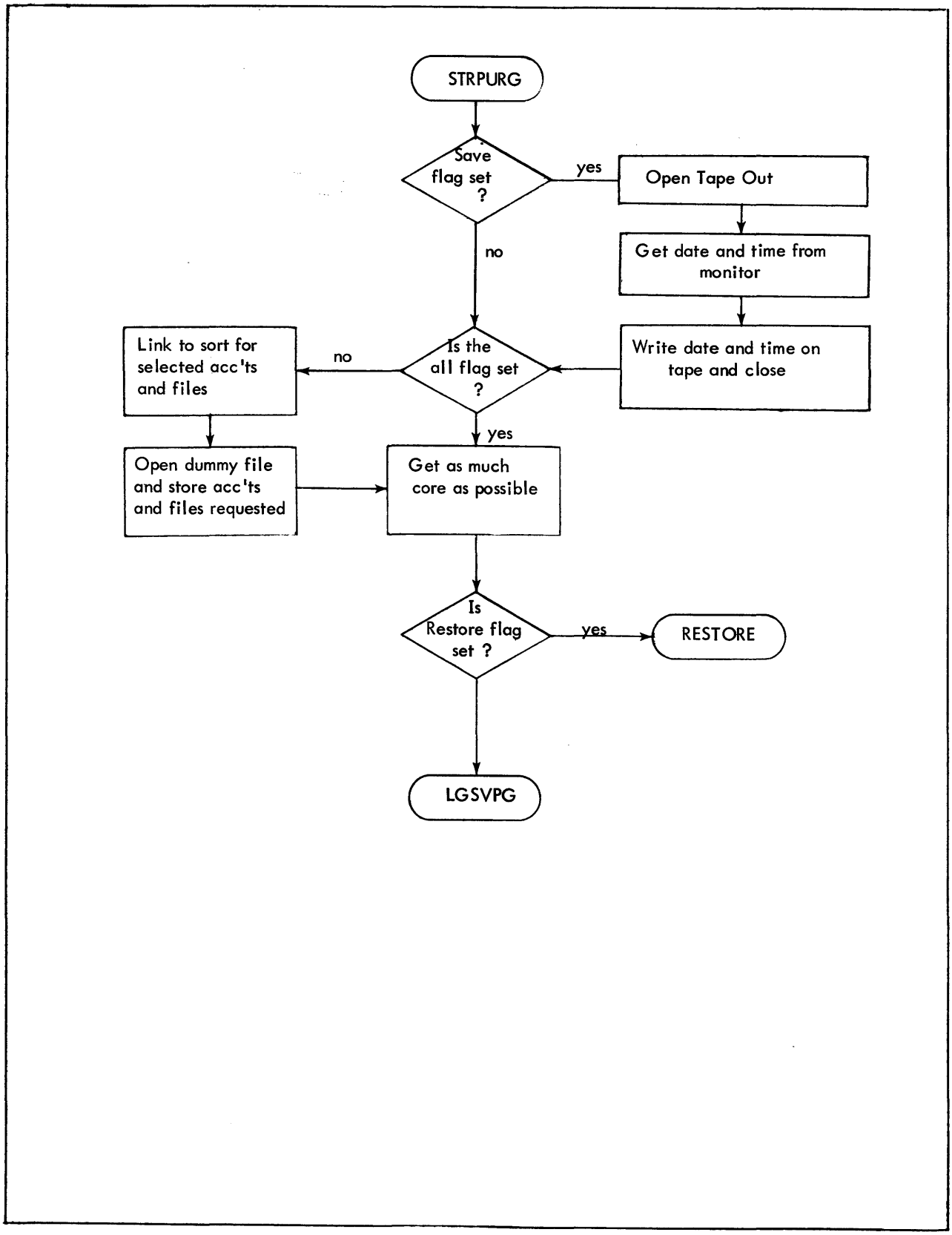


Figure 6-4. Flow Chart of STRPURG.

6.4.4 LGSUPRG.

1. Purpose:
   to Log, Save, or Purge accounts or files in an account.

2. Entry:
   Entered from STRPURG.

3. Exits:
   B MISFILS

4. Operation:
   This subroutine checks if a special start was requested (SPCSTR). If it was this routine will start with the account that was specified through the OC device. If no special start was requested, this routine will Log, Save, or Purge either all the accounts or just the accounts and files requested by the select option. This subroutine Bal's on R11 to GETC if the select option was used. This routine also Bal's on R11 to the following routines: OPNXACCT, CLRDCB, and PRINT.

6.4.5 GETC.

1. Purpose:
   to get data cards if select option was used.

2. Entry:
   BAL R11

3. Exit:
   B *R11

4. Operation:
   Opens a dummy file to read data cards to process specific accounts and/or files. A flag is set, (KDONE) when all data cards are read.

6.4.6 OPNXACCT.

1. Purpose:
   to get next account.

2. Entry:
   R2 and R3 contain current account. BAL R11

3. Exit: B *R11 with R2 and R3 containing next account. If next account is found, R11 returns with a +2. If all accounts are processed as need be, R11 returns with a +1.

4. Operation:
   This routine does an open next on accounts specified.

6.4.7 CLRDCB.

1. Purpose:
   to open all files.

2. Entry:
   BAL R11

3. Exit:
   B *R11

4. Operation:
   Zeroes out Password and Read Account, because FPURGE processor has to be able to access all files.

6.4.8 PRINT.

1. Purpose:
   Print file name on LO.

2. Entry:
   BAL R11

3. Exit:
   B *R11

4. Operation:
   This routine will print the account each time it changes. All file names, and size in granules will be printed on the LO, and if the count option was used on the control card, each file will have a byte count printed also. When accounts change a total of granules and bytes will be printed for each account.

6.4.9 WRTTP.

1. Purpose:
   Write tape.

2. Entry:
   BAL R11

3. Exit:
   B *R11

4. Operation:
   Writes tape in labeled format.

6.4.10 RESTORE.

1. Purpose:
   Restore tape to Rad and/or disk

2. Entry:
   Entered from STRPURG.

3. Exit:
   B MISFILS

4. Operation:
   A check is made to see if volume 1 is requested. If not, the tape is opened unlabeled and positioned to the :ACN sentinel. If the :ACN sentinel is correct the tape is opened as labeled and is restored to Rad and/or disk. If it is Vol 1, the tape is opened labeled and processed per the requirements on the control card.

6.4.11 MISFILS.

1. Purpose:
   to print on LO ane files not found.

2. Entry:
   Entered from LGSUPRG and Restore

3. Exit:
   B GETOUT

4. Operation:
   A check is made to see if the count option was used; if it was, print the total number of granules and bytes that were processed. We then check, if any files that were requested by the select option, were not processed. If not, these files are printed on the LO as not found.

6.4.12 GETOUT.

1. Purpose:
   Terminate Processor

2. Entry:
   Entered from MISFILS

3. Exit:
   M:EXIT

4. Operation:
   Restores original account number and then exits.

Table 6-1. FPURGE Options

OPTION	MEANING	OPTION ALLOWED FOR			
		SAVE	RES	LOG	PURGE
ALL	All user's files will be processed. All cannot be used if SELECT is also specified.	X	X	X	X
COUNT	FPURGE will print the file names, number of granules, and number of bytes.	X	X	X	X
DATE	FPURGE will ask the operator to type in a desired date and time. Only files that have been created or updated on or after this date will be processed.	X	X	X	X
DP	When DP is used, FPURGE will make two passes through the file directory. The first pass will save file(s) that are RANDOM and have been specified to be on a disk pack. The second pass will process all other files.	X			
SELECT	Specifies that files to be accessed are defined by data cards. SELECT cannot be used if ALL is specified.	X	X	X	X
SKIP	This option gives the user the ability to name accounts to safeguard. When SKIP is used, the default accounts (:SYS, :BTM, and COBLIB) will be replaced be the accounts named on the skip cards. The format of the SKIP card is SKIP (ACCNT,),(ACCNT$_2$),(ACCNT$_n$) SKIP must be in columns 1-4 and must be on every card used. A max. of 110 accounts can be named. SKIP cards must follow directly after the FPURGE control command and must be terminated by a .END card in columns 1-4.	X	X	X	X
START	FPURGE will request the operator to key in the account and file at which processing is to start. This is provided for error recovery.	X	X	X	X

Table 6-1. FPURGE Options (Cont.)

OPTION	MEANING	OPTION ALLOWED FOR			
		SAVE	RES	LOG	PURGE
START (AUTO)	If the system crashes during the save function, START (AUTO) will find (from previous tape) what the last file was and continue from the next file.	X			
VOL	FPURGE will ask the operator to type in the desired tape volume number with which to start restore operations.		X		

7.0 FERRET SUBSYSTEM

7.1 FUNCTIONAL OVERVIEW

7.1.1 Purpose

FERRET is a utility subsystem which provides a general capability for obtaining information about entries in the file management system. The subsystem also provides for limited file manipulation. Its capabilities include listing and reviewing all files in a specific account; copying, deleting, and testing the status of specific files; and displaying records within a file.

7.1.2 Error Messages

1. ILLEGAL SYNTAX. A syntax error has been found in processing the COPY or KOPY command.
2. COMMAND NOT LEGAL. The command entered by the user is not a legal FERRET option.
3. CANNOT DELETE FILE file name. In processing the DELETE command, a specified file cannot be deleted; the most probable reason for this is that the file is currently in use by another user.
4. CANNOT ACCESS FILE file name. An abnormal condition was encountered when opening the specified input file.
5. FILE DOES NOT EXIST - file name. The specified input file does not exist in the Log-in account.
6. CANNOT ACCESS NEXT FILE. In processing the LIST, REVIEW or GRANULES command, the next file in the account could not be opened.
7. CANNOT COPY -- RECORDS TOO LARGE. In processing the PUNCH, COPY or KOPY command, data was lost because the buffer is smaller than the record read.
8. FIRST RECORD NON-EXISTENT. In processing the EXAMINE or INSPECT command, the first record specified does not exist in the file.
9. RECORD EXCEEDS BUFFER SIZE. In processing the EXAMINE or INSPECT command, data was lost because the buffer is smaller than the record read.
10. M>N IN M, N SPECIFICATION. In processing the EXAMINE or INSPECT command, the number of the first record to be examined is greater than the last record in the range.
11. CANNOT CREATE FILE. In processing the COPY or KOPY command, the specified file could not be opened in the output mode.
12. NO OUTPUT SPECIFICATION. In processing the COPY or KOPY command, the output file specification was not present.
13. NO SPECIFICATION FIELD. The command given requires a specification field and none is present.
14. ILLEGAL DECIMAL CHARACTER. In processing the EXAMINE or INSPECT command, the input to specify the record(s) to be examined is not a decimal character.
15. FIRST REC G-T RANDSIZE. In processing the EXAMINE or INSPECT command, the input file is random and does not contain the first record specified.
16. LAST REC G-T RANDSIZE--RANDSIZE SUBSTITUTED. In processing the EXAMINE or INSPECT command, the input file is random and does not contain the last record specified.
17. NO KEY LENGTH FOR KEYED FILE--ERROR. An error exists in the FPARAMS returned when a Keyed file was opened.

7.2 INTERFACES

FERRET is given control by BTM:EXEC upon user request, and is entered at STARTFIL. The address of its normal entry point (STARTFIL) is contained in word 9 of FERRET and the entry point following IPROCEED is at STARTFIL+1, in accordance with coding conventions for BTM subsystems.

CALs to the monitor are used to access user commands and input text (CAL3,0) and to write to the teletype (CAL3,1). CAL3,11 is used to describe memory so that the maximum number of available memory pages are swapped in and out. The activation type is set to 4 via a CAL3,2, causing activation only on carriage return or line feed. CAL1,8 is used to obtain accounting data on the current session.

7.3 OPERATIONAL OVERVIEW

FERRET is composed of two types of routines: command routines which perform the functions specified by the input command; and service sub-routines which perform tasks common to several command routines and are called by a command routine when needed.

Upon entry from the BTM Executive the FERRET routine performs subsystem initialization, identifies the command which was input by the user, and gives control to the routine which will process that command.

Commands which pertain to an account and all files contained therein are LIST, REVIEW and GRANULES. When the routines REVIEW and GRANULES receive control from FERRET they essentially just set a flag to identify the function to be performed, then give control to the LIST routine which processes all three commands.

Commands which pertain to a specific file (and the routines which process them if the name of the routine is different) are TEST (CHECK), ACTIVITY (ACTIVCK), DELETE, COPY (COPYFILE), KOPY(KOPYFIL), and PUNCH. KOPYFILE merely sets a flag to indicate that keys are to be retained, then gives control to COPYFILE which processes both COPY and KOPY commands.

Commands which pertain to a file and its individual records are EXAMINE and INSPECT. At location INSPECT a flag is set to indicate that keys are to be displayed, then control is given to the routine EXAMINE which processes both commands.

Input files are read thru the M:EI DCB, output files are written (COPY or KOPY commands) thru the M:EO DCB, and records are displayed on the user's terminal (EXAMINE and INSPECT commands) thru the M:LO DCB.

See Figure 7-1 for the FERRET Subsystem flow diagram.

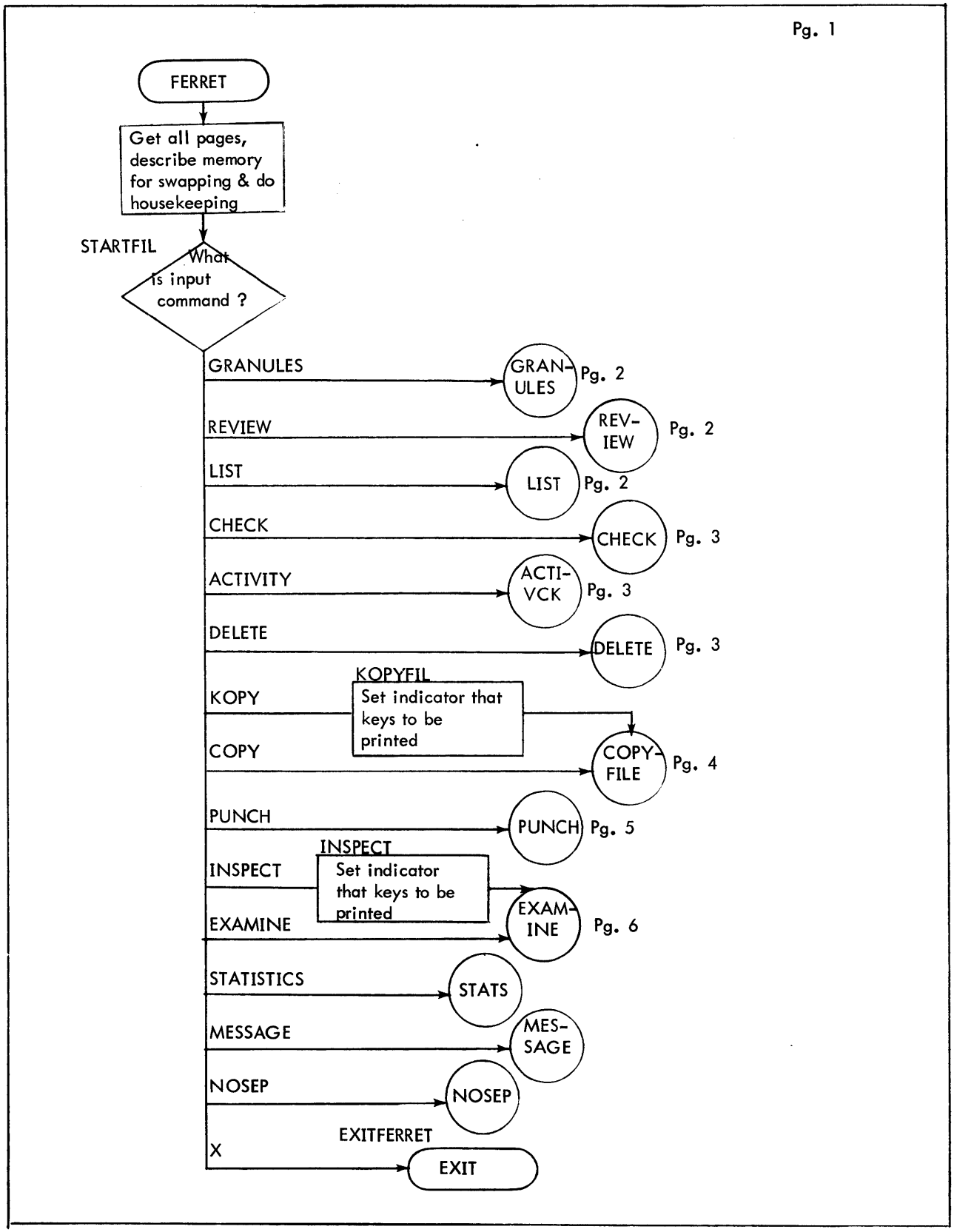


Figure 7-1. Flow Diagram of FERRET

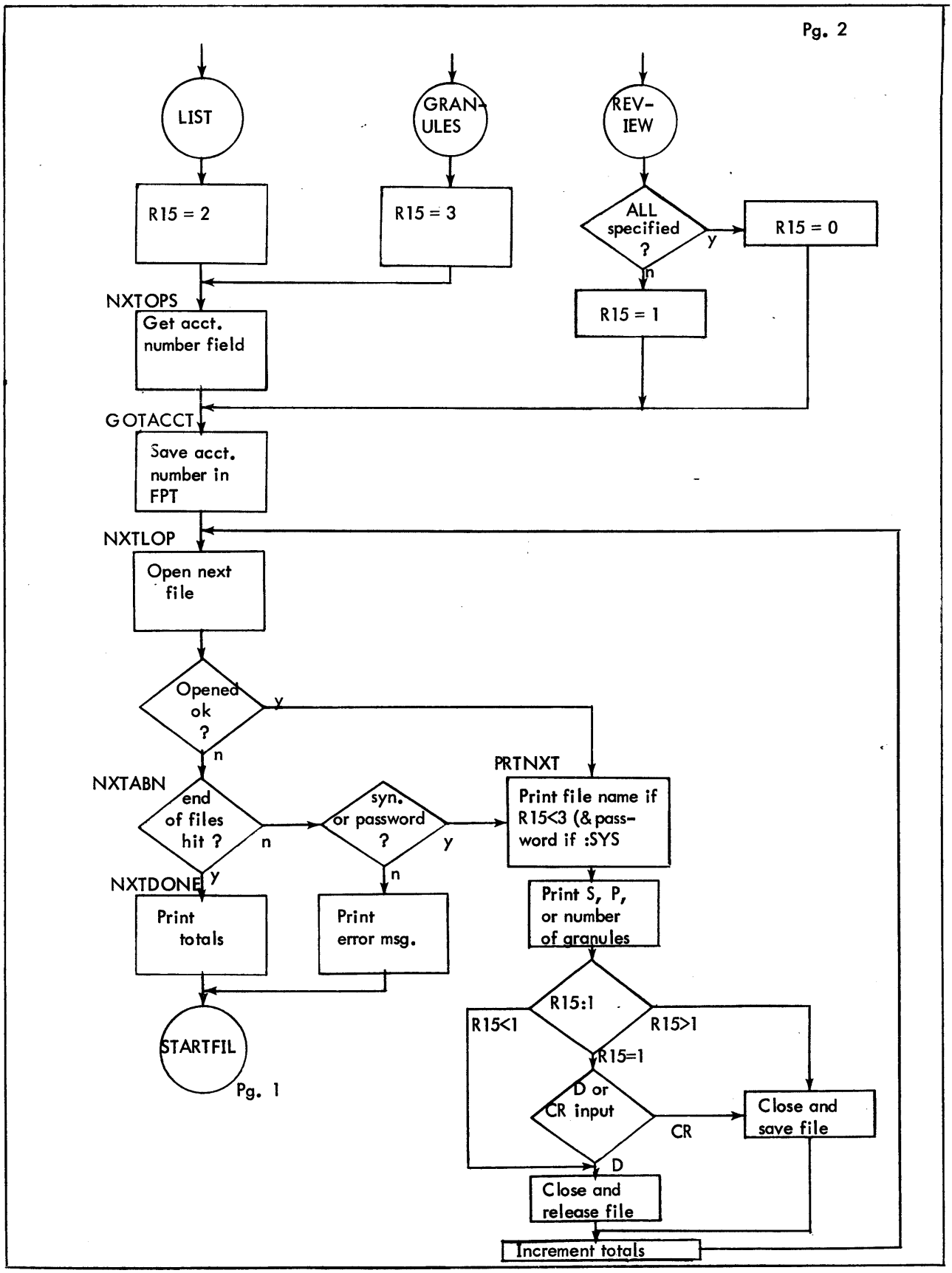


Figure 7-1. Flow Diagram of FERRET (Cont.)

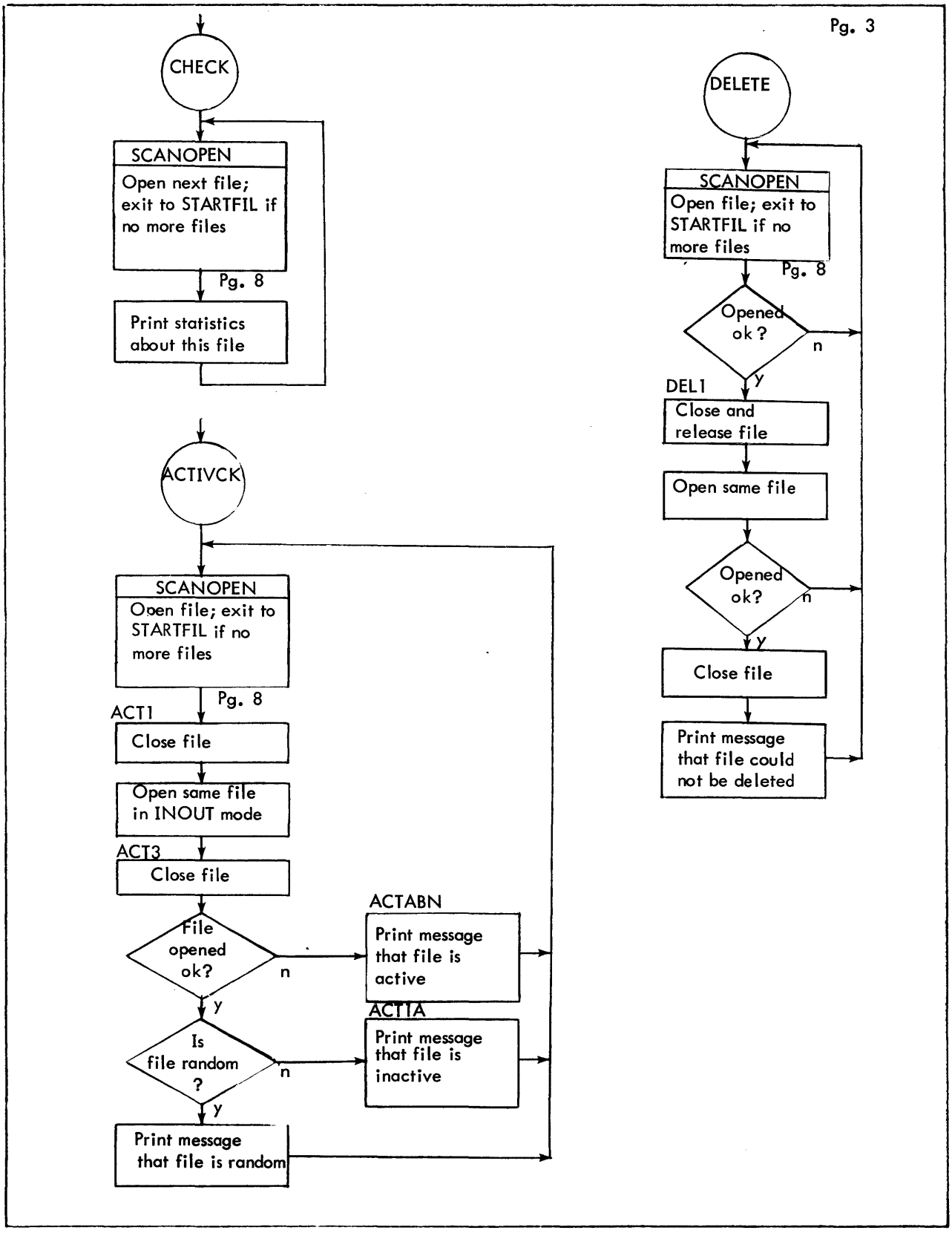


Figure 7-1. Flow Diagram of FERRET (Cont.)

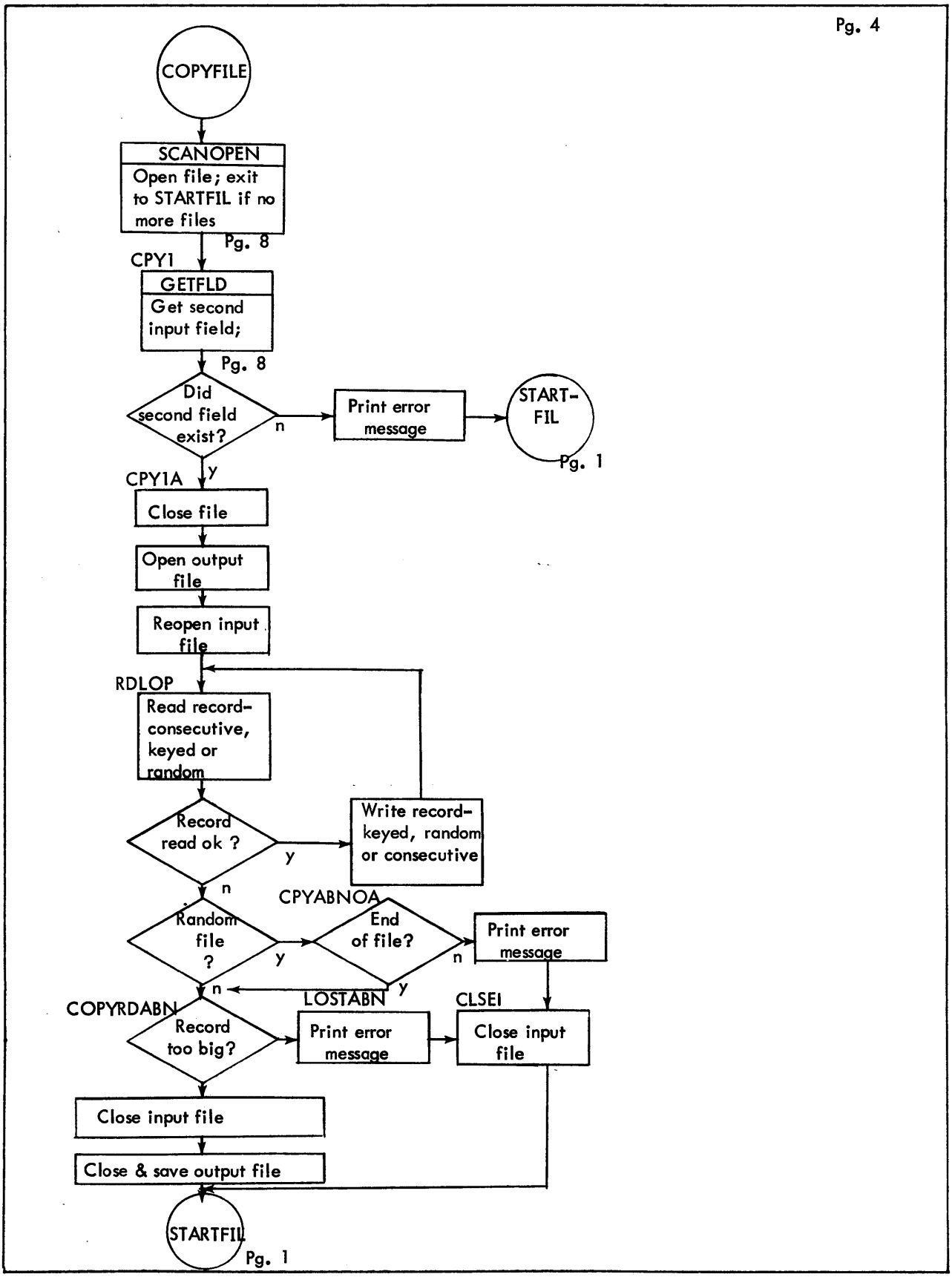


Figure 7-1. Flow Diagram of FERRET (Cont.)

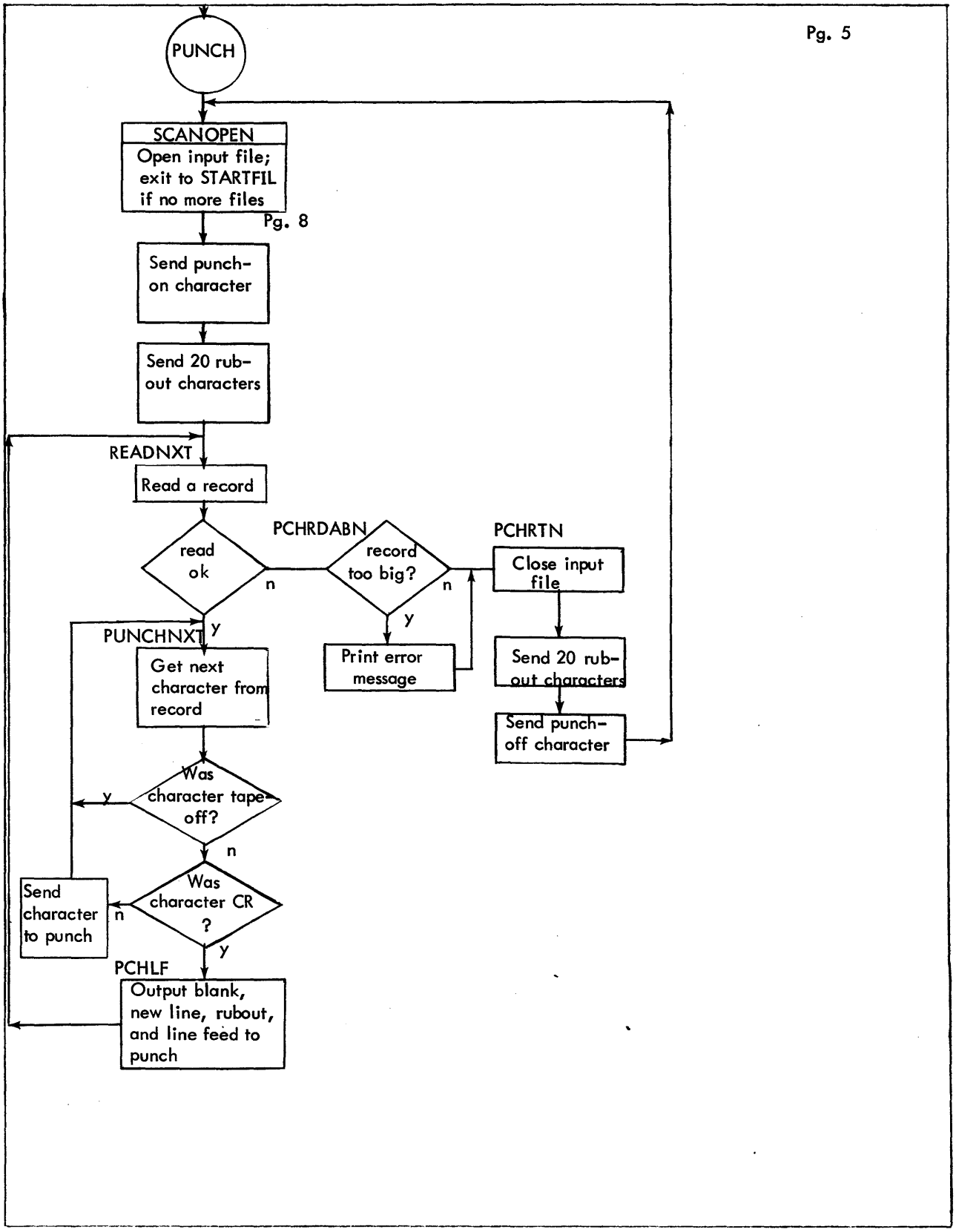


Figure 7-1. Flow Diagram of FERRET (Cont.)

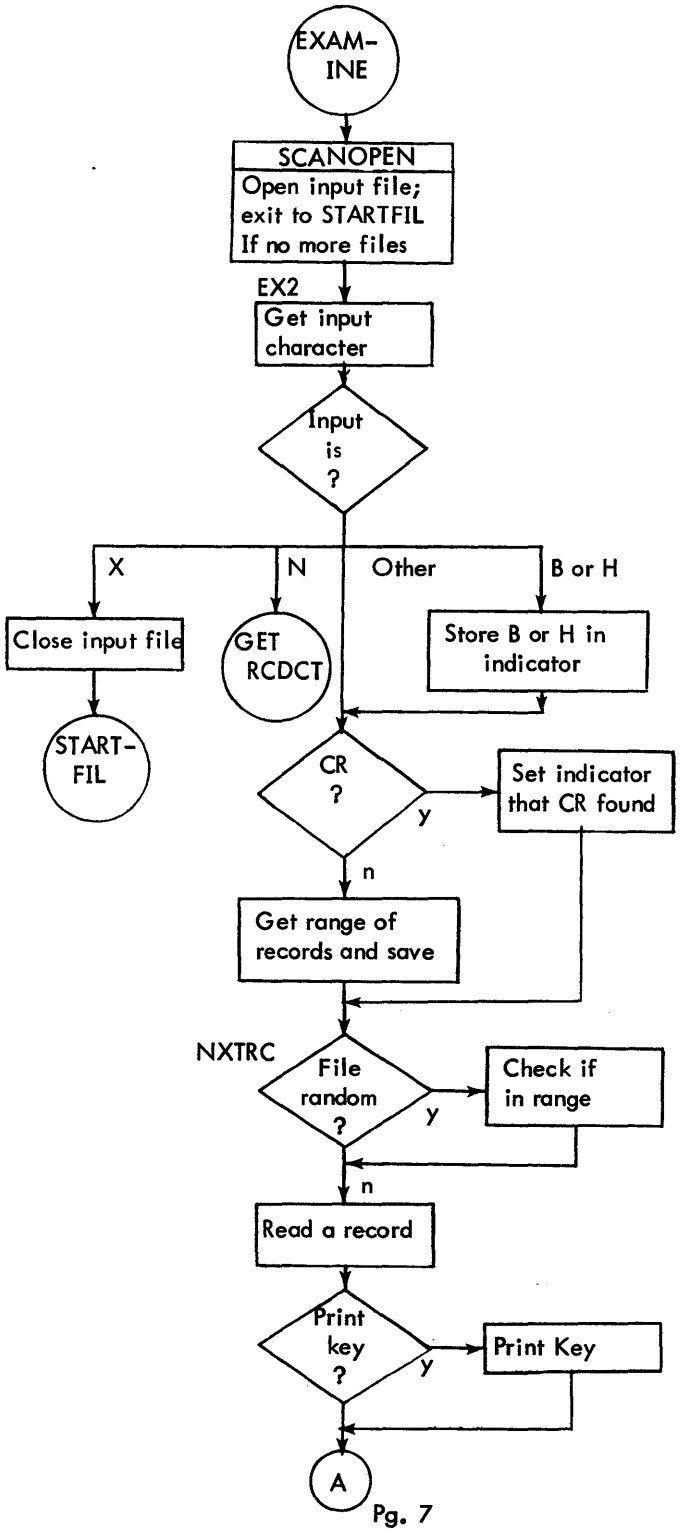


Figure 7-1. Flow Diagram of FERRET (Cont.)

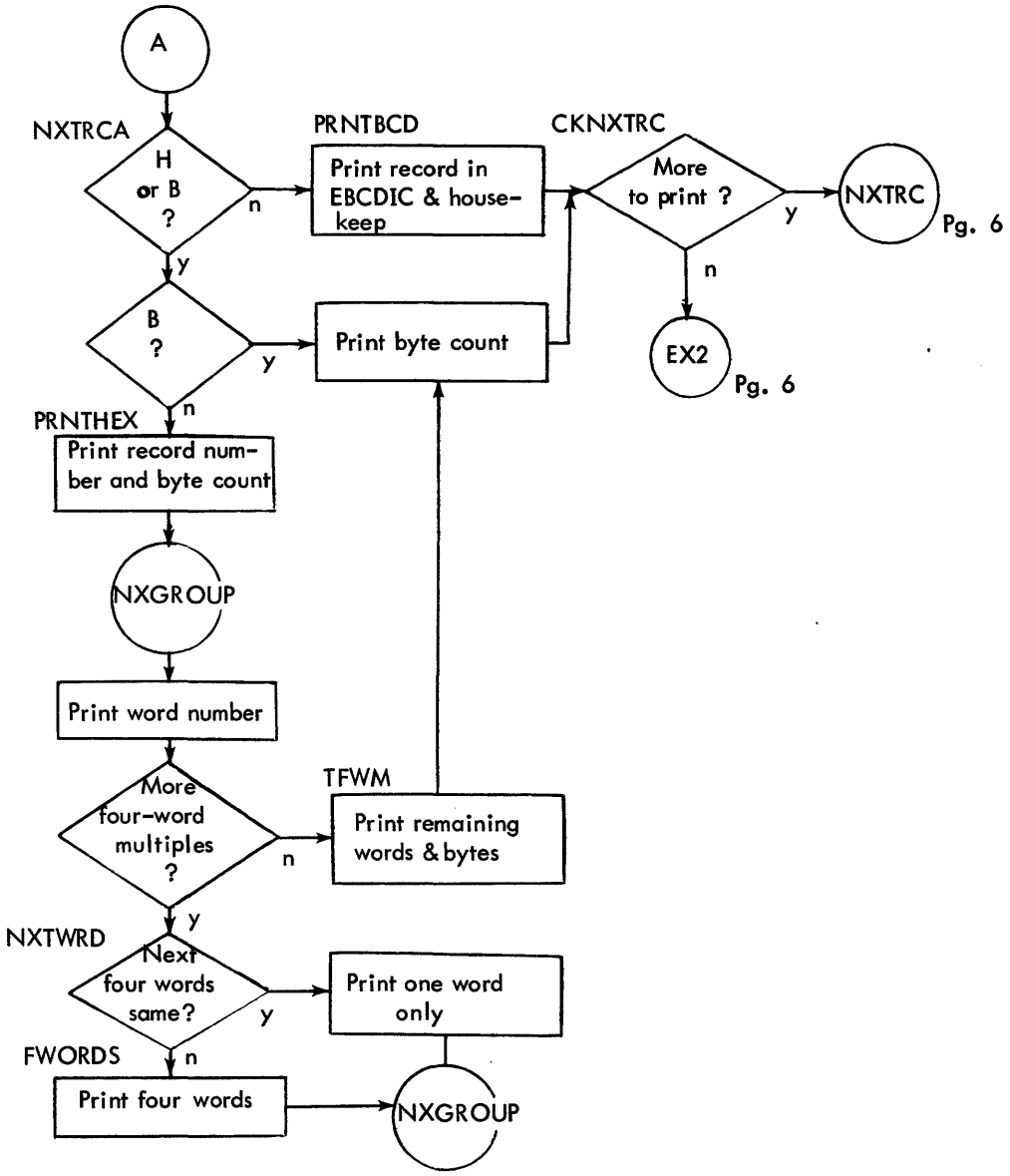


Figure 7-1. Flow Diagram of FERRET (Cont.)

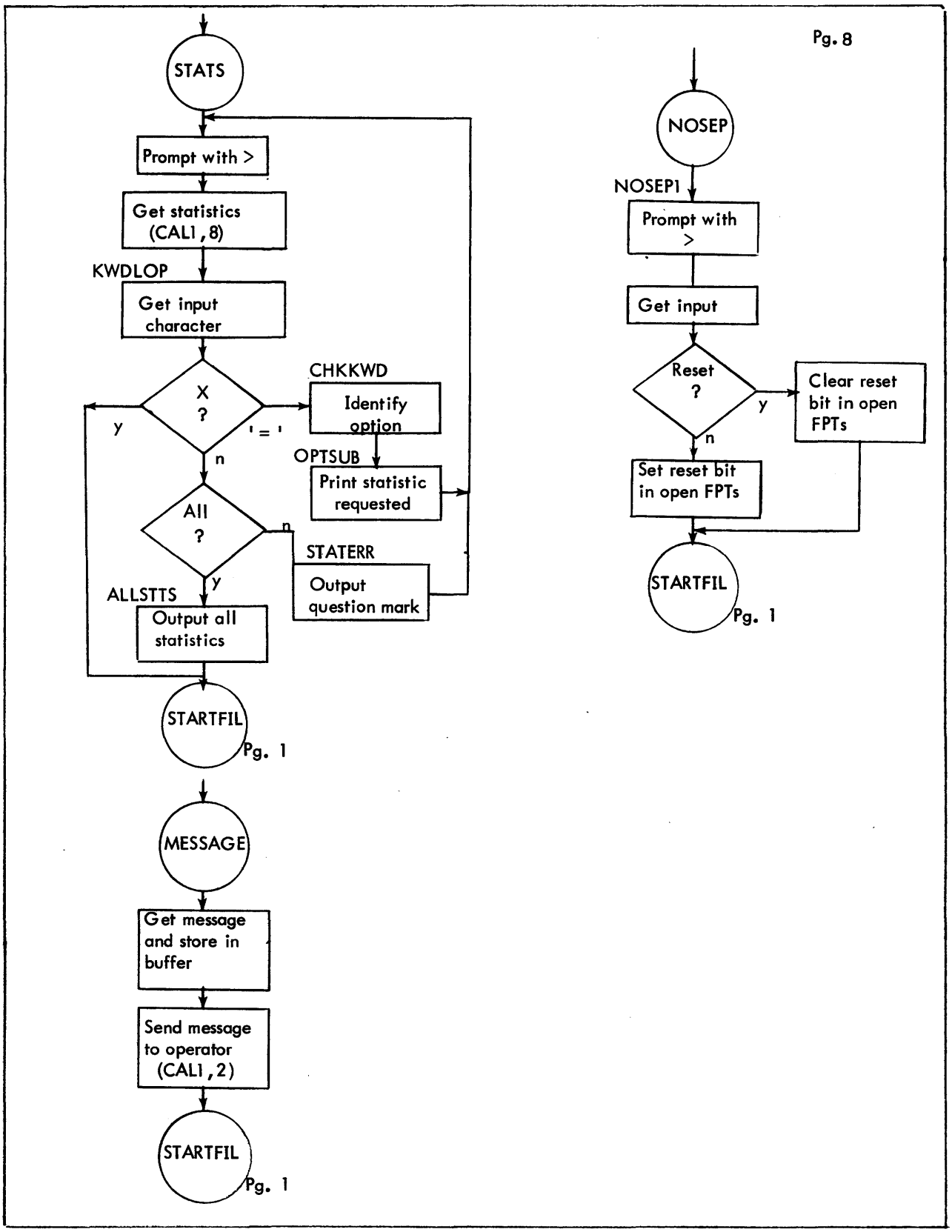


Figure 7-1. Flow Diagram of FERRET (Cont.)

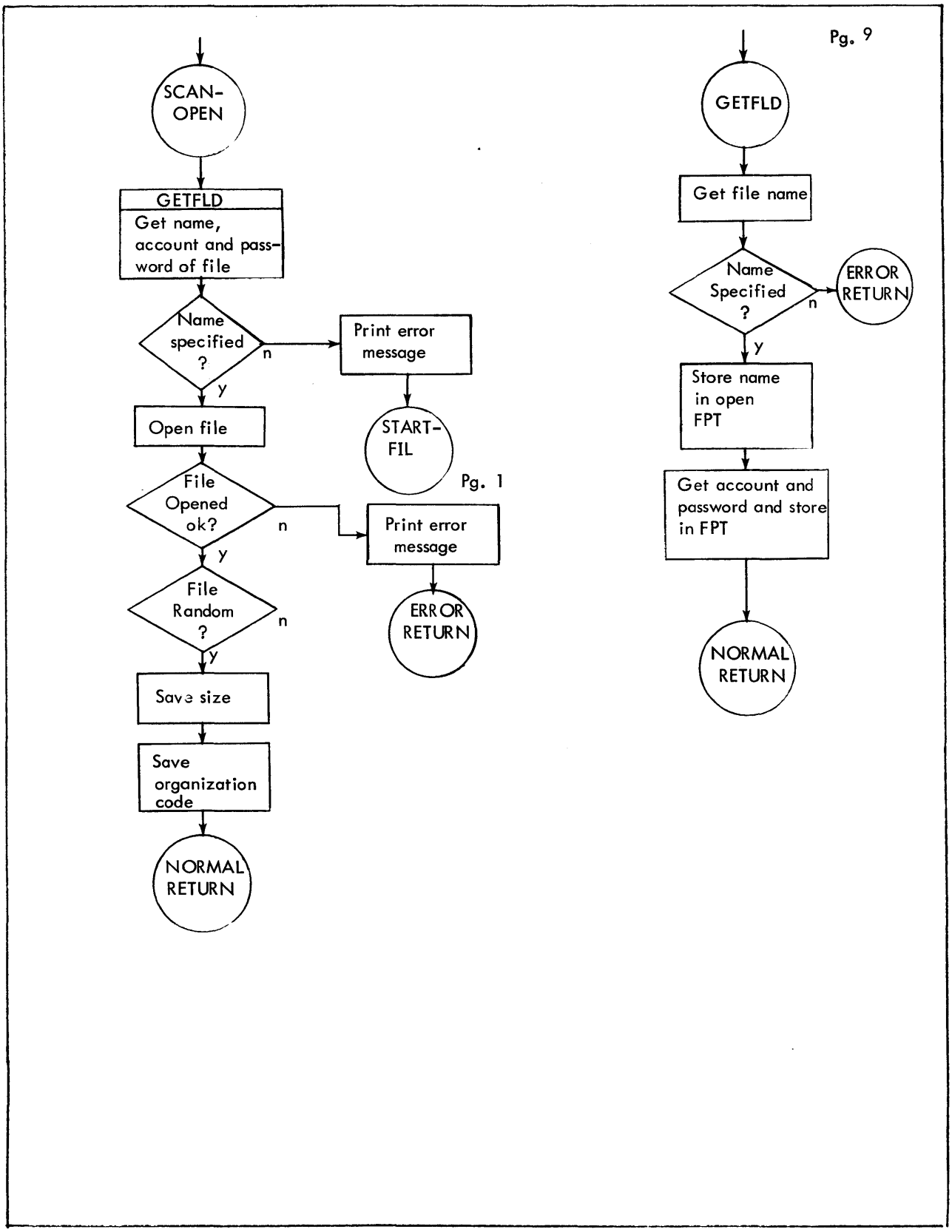


Figure 7-1. Flow Diagram of FERRET (Cont.)

7.4 MODULE ANALYSIS

Flow charts of the individual modules described in this section appear in Figure 7-1 as part of the FERRET flow diagram.

7.4.1 FERRET

1. Purpose:
   To perform initialization and housekeeping functions upon entry into the FERRET subsystem, and to identify the command to be processed.
2. Entry:
   Entered at STARTFIL from BTM Executive. In accordance with subsystem coding conventions, registers at entry contain:

(R1)	=	COC line number (in binary)
(R2)	=	terminal job entry flag; 0 indicates that the console is excluded from the system, and a value of 1-F indicates the maximum priority.
(R3)	=	batch authorization flags from AJIT in byte 0.
(R4 & R5)	=	log-in account designation (in EBCDIC, left-justified and blank filled).
(R13 - R15)	=	log-in name designation (in EBCDIC, left-justified and blank filled).

3. Exit:
   Exits to the routine which will process the current command (the branch table is CMNDVEC).
4. Operation:
   The COC line number is converted to EBCDIC and placed in the print image for the MESSAGE command (OPRMES). The number of available pages is requested and a CAL3,11 describes memory so that all available memory is swapped in and out. The activation type is set to 4--activation on carriage return or line feed only. After the input command has been identified, exit is to the routine which will process that command.

7.4.2 LIST, REVIEW, GRANULES

1. Purpose:
   -- To print a list of all files in an account (with the password, if the user is in the :SYS account), and the size of each file in granules if the LIST command is invoked;
   -- To print the total number of granules used by all unprotected files in an account if the GRANULES command is invoked;
   -- To print the name of each file in the log-in account and halt after each file (except passworded or synonymous files) for a decision from the user whether to delete or save each file if the REVIEW command is invoked;
   -- To delete all unprotected files in the log-in account if the REVIEW ALL command is invoked.
2. Entry:

Command	Initial Entry point after identification of command	Entry point in LIST routine	Contents of R15 at entry to LIST routine
LIST	LIST	LIST	2
GRANULES	GRANULES	NXTOPS	3
REVIEW	REVIEW	GOTACCT	1
REVIEW ALL	REVIEW	GOTACCT	0

3. Exit:
   Exits to STARTFIL in FERRET routine after processing all files in the account.

4. Operation:
   Although all three commands are processed by the LIST routine, the REVIEW and GRANULES routines require some initialization prior to entering the LIST routine. This is performed at locations REVIEW and GRANULES, respectively, and includes setting register 15 to indicate which function is to be performed.

   LIST opens the first file in the account and prints, for REVIEW or LIST commands, the name of that file. If the file was opened successfully, the size of the file in granules is printed following the name; if it was not opened and the command was REVIEW, P or S is printed instead of the granule count if the file is protected or synonymous. If the command was GRANULES, the number of granules in the file is added to the total number of granules for the account. If the command was REVIEW, LIST checks input from the terminal to see if the user wants the file deleted. If the file is to be deleted or if the command was REVIEW ALL, the file is closed and released; if the file is not to be deleted or the command was LIST or GRANULES, the file is closed and saved.

   This operation is continued for each file in the account until all files have been processed, at which time totals are printed and control is given to STARTFIL.

7.4.3 CHECK

1. Purpose:
   To determine whether the user may read one or more specified files.

2. Entry:
   Entered at CHECK after identification of the TEST command by the FERRET routine.

3. Exit:
   Exits (from SCANOPEN subroutine) to STARTFIL in FERRET routine after testing the file(s) specified.

4. Operation:
   BALs to the subroutine SCANOPEN, which opens the specified file, if possible. If the file was opened, CHECK1 outputs some statistics about the file on the appropriate terminal, then branches to CHECK to test the next file specified.

7.4.4 ACTIVCK

1. Purpose:
   To determine whether the specified file(s) are currently inactive (accessible).

2. Entry:
   Entered at ACTIVCK after identification of the ACTIVITY command by the FERRET routine.

3. Exit:
   Exits from SCANOPEN subroutine to STARTFIL in FERRET routine after processing the file(s) specified.

4. Operation:
   BALs to the subroutine SCANOPEN, which opens the specified file in the IN mode. If the file was opened (meaning the file does exist), it is closed and then reopened in the INOUT mode. If it was successfully opened in the INOUT mode it is either inactive or has random organization and a message is output indicating which case is true. If the file could not be opened in the INOUT mode, a message to this effect is output before returning to ACTIVCK to process the next file specified.

7.4.5 DELETE

1. Purpose:
   To delete one or more specified files from the log-in account.
2. Entry:
   Entered at DELETE after identification of the DELETE command by the FERRET routine.
3. Exit:
   Exits (from SCANOPEN subroutine) to STARTFIL in FERRET routine after processing specified file(s).
4. Operation:
   After setting the DCB to the update mode, DELETE BALs to the subroutine SCANOPEN, which opens the specified file, if possible. If the file was successfully opened, upon return from SCANOPEN the file is closed with REL specified. A check is made to insure the file was actually deleted and if so, the next specified file is processed. If the file was not deleted, an error message is output before the next file is processed.

7.4.6 COPY, KOPY

1. Purpose:
   To create a new copy, in the log-in account, of an existing file (and to retain keys if the command is KOPY).
2. Entry:
   Entered at COPYFILE after identification of the command by the FERRET routine. If the command was KOPY, a flag is set at KOPYFIL before control is given to COPYFILE.
3. Exit:
   Exits to STARTFIL in FERRET routine (via CKFLSH) after performing functions specified.
4. Operation:
   COPYFILE BALs to the SCANOPEN subroutine to get the name of the input file and open it. The name of the output file is then obtained, using the GETFLD subroutine. If the input file is keyed, a check is made to see if a key entry is present in the FPARAMS and whether keys are to be retained when the file is copied. The input DCB is closed so that the output DCB can be opened in case the input and output files have identical names. (For further information on the protocol of opening multiple DCBs to the same file name refer to the File Attributes appendix of the BPM Reference Manual, 90-09-54E). The output DCB is opened and the input DCB is reopened. Successive input records are read and written to the output file until the end of the file is encountered. The input and output DCBs are then closed in that order and control is given to STARTFIL. If either DCB could not be opened, if data was lost, or if the input command was not in the proper format, an error message is printed before control is returned to STARTFIL.

7.4.7 PUNCH

1. Purpose:
   To output (punch) one or more files to paper tape on the user's console.
2. Entry:
   Entered at PUNCH after identification of the PUNCH command by the FERRET routine.
3. Exit:
   Exits from SCANOPEN subroutine to STARTFIL in FERRET routine after processing the file(s) specified.
4. Operation:
   BALs to the subroutine SCANOPEN which opens the specified file, if possible. If the file could not be opened, control goes to PUNCH to process the next file. If the file was opened successfully, a character (X'12'), is output to turn on the punch at the user's terminal, and outputs 20 rubout

(X'00') characters to prepare the paper tape. A record is read from the input file and each character in that record is obtained from the input buffer and output to the punch until a carriage return (X'15') is encountered. A carriage return terminates that record regardless of the number of characters that were punched. Several characters are output as a divider between records and the routine branches back to read the next record. When all records have been output and the file is complete, the input file is closed, 20 rubout characters are output to terminate the file, and the punch is turned off. Control returns to PUNCH to process the next file specified. If no more files are specified, the SCANOPEN subroutine returns control to STARTFIL.

7.4.8 EXAMINE, INSPECT

1. Purpose:
   To examine a file and display any or all of the following information, based on requests input by the user.
   -- print the number of records in the file

   -- print the number of bytes in each record or in specific records

   -- print, in EBCDIC or hexadecimal, the contents of all records in the file or of specific records.

   If the command is INSPECT, the same action takes place except the key is also displayed for each record selected.

3. Entry:
   Entered at EXAMINE after identification of the EXAMINE command if the command was INSPECT, a flag is set at INSPECT to signify that keys are to be retained, then control is given to EXAMINE.

3. Exit:
   Exits to STARTFIL after the user inputs an X in response to a prompt.

4. Operation:
   BALs to the subroutine SCANOPEN, which opens the specified file, then issues a prompt. The user's response is checked to determine what action is to be taken.

   If N was specified, the number of records, in the file is determined and printed. If the sub-command was not N, PRNTRCDS positions to the first record to be processed (the first record, if no range was specified) and that record is read. If the command was INSPECT, the key is printed at this time. If B was specified, the record number and the number of bytes in that record are printed, and the next record is read. If neither H nor B was specified, the record is written thru M:LO to the user's terminal and the next record is read. If H was specified, the record number and byte count are printed in hex before the record is dumped and the record is examined in 4-word blocks. If all four words in a block are identical, the word is printed only once on that line and the next block is examined. If the four words are not identical, all four words are printed on one line and the next block is examined. Each line is preceded by the number of the first word in that line, starting with word 0 on line 1. When the entire record has been printed, the next record is read. When all records have been processed, another prompt is issued. If the user's response is other than X, the above sequence of actions is repeated. If the response is X, control is given to STARTFIL.

7.4.9 STATS

1. Purpose:
   To display any or all of the following statistics for the current session, as requested by the user.

   -- number of users currently logged in

   -- amount of time current user has been logged in

   -- number of RAD granules remaining and/or used in current session

   -- number of disk granules remaining and/or used in current session

   -- amount of CPU execution time, I/O wait time, and/or Monitor service time used in current session.

2. Entry:
   Entered at STATS after identification of the STATISTICS command by the FERRET routine.

3. Exit:
   Exits to STARTFIL in the FERRET routine after the user inputs an X in response to a prompt.

4. Operation:
   STATS does a CAL1,8 to obtain the accounting statistics, saves these statistics, and issues a prompt for subcommands. After the subcommand is identified by indexing thru a table of subcommands (KWDS), the index value when the command was identified is used to access the storage area for the particular statistic desired. This statistic is converted and output with the appropriate message. If the subcommand ALL was input, the index is set to the end of table KWDS and decremented until all statistics have been printed. When the subcommand entered in response to a prompt is identified as X, control is given to STARTFIL.

7.4.10 MESSAGE

1. Purpose:
   To cause a specified message to be printed on the system operator's console, prefaced by an identification of the originating console.

2. Entry:
   Entered at MESSAGE after identification of the MESSAGE command by the FERRET routine.

3. Exit:
   Exits to STARTFIL in the FERRET routine.

4. Operation:
   Obtains each character in the message from the terminal and places it in the proper psoition in the TEXTC statement, OPRMES. When all characters have been stored in the print image, OPRMES is output to the system operator's console via a CAL1,2 0, and control is given to STARTFIL. If the size of the message will exceed 100 characters or if a carriage return is encountered immediately following the MESSAGE command, an error message is output before control is given to STARTFIL.

7.4.11 NOSEP

1. Purpose:
   To set (or reset) the NOSEP bit in the open FPT for M:EO.

2. Entry:
   Entered at NOSEP after identification of the NOSEP command.

3. Exit:
   Exits to FERRET1 in the FERRET routine.

4. Operation:
   Upon entry, NOSEP asks the user whether the NOSEP bit is to be set or reset and prompts for an answer. If the user answers either SET or RESET, a 1 or 0, respectively, is merged into bit 9 of the CPYOPN FPTs and control is given to FERRET1. If the user responds to the prompt with other then SET or RESET, the routine outputs two question marks and another prompt character. This continues until the user responds with either SET or RESET.

7.5 SERVICE SUBROUTINES

The following service subroutines are used by several different command routines andperform such tasks as converting numbers, examining input fields, opening the input DCB, and searching tables.

7.5.1 CVDECOUT

1. Purpose:
   To convert a hexadecimal number to decimal and output it to the terminal.

2. Entry:
Entered from CHECK, LIST, EXAMINE and STATS.

```
BAL,3  CVDECOUT
(1) = number to be printed
```

3. Exit:
B *3

4. Operation:
The value in R1 is tested and a minus is output if it is negative. The number is then converted to decimal digits and these digits are placed in a buffer. Each digit is converted to EBCDIC, then output to the terminal.

7.5.2 GETFLD

1. Purpose:
To obtain an entire field from the terminal -- name (account number, password) -- and put it in the proper entries in the FPT.

2. Entry:
Entered from LIST, REVIEW, COPYFILE and SCANOPEN.

```
BAL,9      GETFLD
```

3. Exit:
Error return B *9
Normal return B *9 (+1)

4. Operation:
GETFLD BALs to GETNAM subroutine to obtain the file name,then stores the names in the FPT at OPNNAM. If an account number and/or password is present, it is obtained and stored at OPNACT or OPNPAS, respectively. If a carriage return was encountered before the file name, the error return is taken.

7.5.3 GETNAM

1. Purpose:
To obtain a field from the terminal and move it into a buffer.

2. Entry:
Entered from NOSEP, GETFLD and GETNAM1.

```
BAL,7      GETNAM
(0) = current character
```

3. Exit:
B *7

4. Operation:
GETNAM gets one character at a time from the terminal and moves it into a buffer in TEXTC format until a terminating character is found. If called by GETNAM1, the TEXTC format is changed to TEXT upon return.

7.5.4 SCANOPEN

1. Purpose:
To open the specified input file.

2. Entry:
Entered from DELETE, CHECK, PUNCH, ACTIVCK, EXAMINE and COPYFILE.

```
BAL,8  SCANOPEN
```

3. Exit:
   Normal return B *8
   Error return B *8 (+1)

4. Operation:
   SCANOPEN BALs to GETFLD to obtain the name, account number and password of the file to be opened. If no name is returned, an error message is printed and control is given to STARTFIL. An attempt is made to open the file. If the file is opened successfully, the organization code and, if the file is random, size are saved. If the file could not be opened an error message is printed and the error return is taken.

7.5.5 LOCCODE

1. Purpose:
   To scan the entries returned by FPARAM for a specific code.

2. Entry:
   Entered from CHECK, LIST, EXAMINE, COPYFILE and SCANOPEN

   ```
   BAL,3      LOCCODE
   (2) = param code
   ```

3. Exit:
   B *3
   (2) = 0 if entry not found
   (2) = address of entry, if found

4. Operation:
   LOCCODE compares the code for each entry with the code in register 2. If the code is found, the address of that entry is returned in register 2. If the code has not been found after all entries have been checked, a zero is returned in register 2.

8.0 EDCON

8.1 FUNCTIONAL OVERVIEW

8.1.1 GENERAL DESCRIPTION

EDCON is a BPM processor which provides the services of creating, deleting, manipulating, compressing and decompressing files. It makes available to a batch user commands which resemble some of the BTM EDIT Subsystem's file commands. Input media acceptable include cards and keyed files in either EBCDIC or compressed format. Output may be in any format acceptable as input with the added feature of sequencing in characters 73-80 of a record. EDCON may be used on-line under the BTM RUN Subsystem. More than one command may be input each time EDCON is invoked, if desired. Command syntax resembles that of EDIT.

8.1.2 COMMANDS

1. BUILD fid [,n [,i]]
   a. M:EI defaults to the card reader.
   b. M:EI may be in either EBCDIC or compressed format.
   c. If M:EI is the card reader, it must be terminated by an !EOD.
   d. If M:EI is a file, it must be keyed having records of 140 bytes or less in length.
   e. M:EO is fid which must not already exist.
   f. M:EO will be a keyed file with records in EBCDIC format. Thus if M:EI is compressed, M:EO will be decompressed.
   g. Sequencing refers to record keys and does not become part of the record text. Default values for n and i are 1.000, where the decimal point is assumed.

2. COPY fid_1 {ON / OVER} fid_2 [,n[,i]]
   a. fid_1 becomes M:EI; fid_2 becomes M:EO.
   b. If fid_1 is not keyed, n must be specified.
   c. If n is omitted, the sequence numbers (record keys) of fid_1 will be retained in fid_2.
   d. Default values for n and i are 1.000.
   e. If $fid_1 \neq fid_2$, a new file is created and password is not allowed.
   f. M:EO will be keyed.
   g. A file may be copied over itself.

3. DELETE fid
   fid must be keyed.

4. END
   This command is not mandatory. Execution is terminated by this command or if omitted by encountering end-of-file on the C device.

5. LIST fid
   a. fid must exist in keyed format.
   b. M:EO defaults to the LO device, and record keys appear with records.

6. MERGE fid_1 [,n_1 [-n_2]] INTO fid_2, n_3 [-n_4][,i]
   a. fid_2 is M:EI and must be keyed.
   b. If [,n_1 [-n_2]] is not specified, the entire file will be transferred.
   c. fid_2 is M:EO and either may or may not exist.
   d. Record(s) n_3 [-n_4] is(are) deleted from fid_2 if existing and is(are) replaced at n_3 by the specified record(s) from fid_1.

e. Default value for i is 1.000.
f. The merge is stopped at n_4 if further transfer would generate records with sequence numbers greater than existing ones.

7. WRITE fid [,LIST]
   a. M:EI is fid and must be keyed.
   b. M:EO defaults to the card punch.
   c. M:EO is written in EBCDIC format having no trailing carriage returns.
   d. When M:EO is written via the card punch sequencing is provided in characters 73-80 in format YYYYXXXX, where YYYY is a maximum of four bytes of fid name and XXXX is sequence number.
   e. If [,LIST] is specified, M:LO defaults to the line printer, and record keys appear with records.
   f. M:EI must be in EBCDIC format.

8. COMPRESS fid [,LIST]
   This command is identical to WRITE except that M:EO is written in compressed format.

8.2 INTERFACES

EDCON operates in a similar manner to other BPM processors and is in addition executable under the RUN Subsystem of BTM.

8.3 OPERATIONAL OVERVIEW

Execution begins in BEGINEDITOR with reading of the first command via the C device and proceeds to MASTERPARSER, which serves as driver for command reading and scanning. As a command is identified, execution is directed to its corresponding PARSE:(Command) subroutine where the command and parameters are entered into the Command Description Table (CDT). See Tables 8-1 and 8-2. When no more commands remain or END command is encountered, execution proceeds to MASTEREXECUTIVE, the driver for command execution which occurs via F:(Command) routines. After all commands have been honored, return is made to the monitor. EDCON is coded in a highly modular fashion. See Figure 8-1, EDCON Overview Flow Diagram.

8.4 MODULE ANALYSIS

In the descriptions which follow register notations have the following meaning:

symbol	is	register
X3		1
X4		2
X1		3
X2		4
P1		5
P2		6
LNK		7
T1		8
T2		9
P3		10
R1		11
R2		12
F:LNK		13

The routine and subroutine descriptions which follow are organized such that MASTERPARSER appears first with its associated PARSE: subroutines, followed by MASTEREXECUTIVE with its associated F: subroutines, and lastly by the general subroutines.

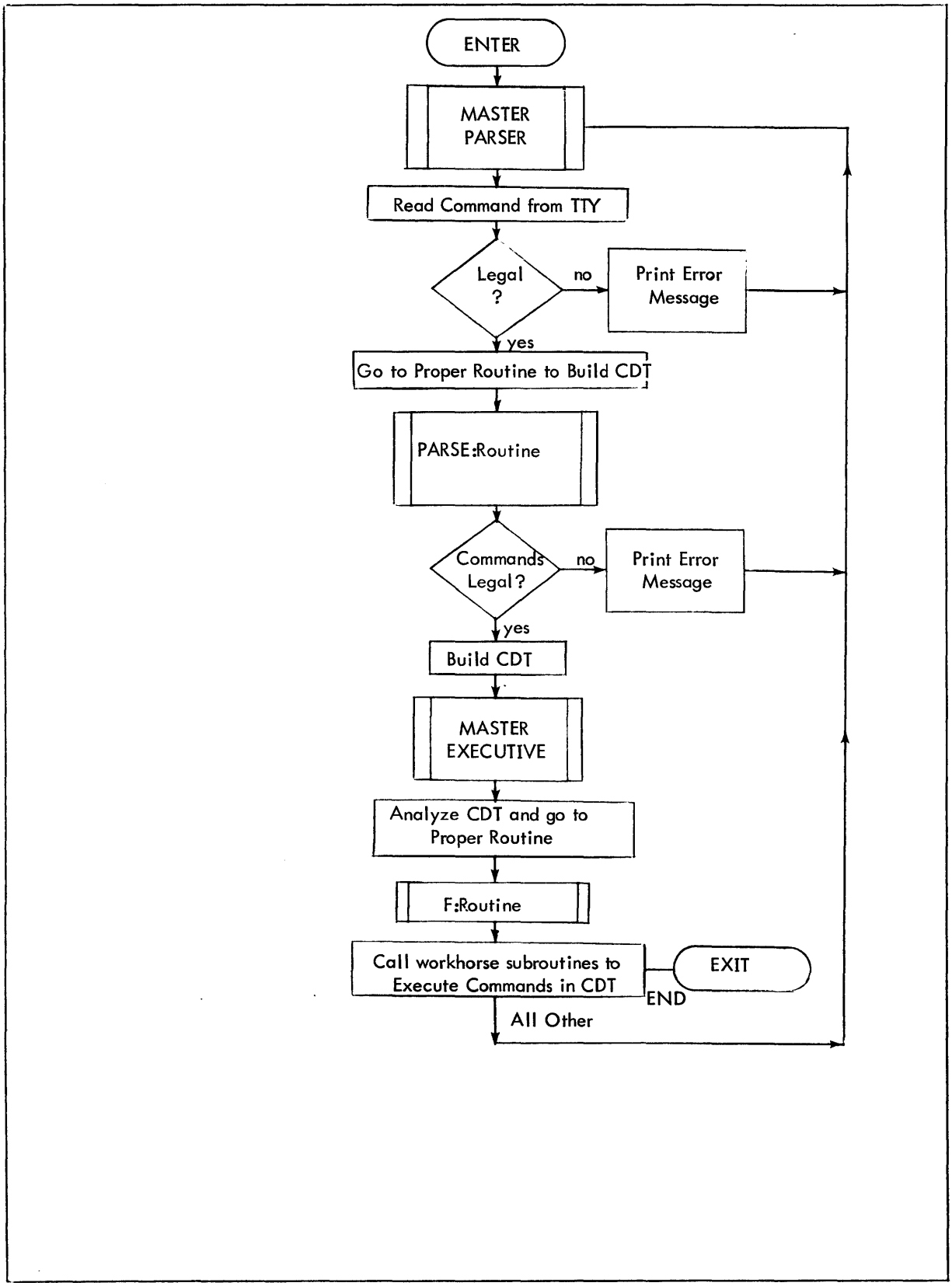


Figure 8-1. Overall Flow Diagram of EDCON

8.4.1 MASTERPARSER

1. Purpose:
   This routine serves as the driver for the command text scanning. It performs initialization of flags, the CDT, and TSTACK.

2. Entry:
   Initially execution falls through to MASTERPARSER from BEGINEDITOR; thereafter it is entered via B MASTERPARSER.

3. Exit:
   (See Figure 1).
   If command type of first string found is: branch is to:

CR	MASTEREXECUTIVE
ALPH	PARSE: (routine)

   where routine is one of the CDT builder routines shown in Table 8-2.

4. Operation:
   After initialization READTELETYPE2 is used to read one line of commands. MASTERPARSER increments CDTADR and the count of CDT entries and resets PARAMPSN. GETNEXTPARAM is used to test command type for one of the following: CR and ALPH. If it is neither, the error message 'C1:ILGL SYNTAX' is typed and branch is made to MASTERPARSER. If the command type is ALPH but the command is not one of those shown in Table 8-2, the message 'C1:UNKN CMND' is typed and branch is made to MASTERPARSER.

8.4.2 PARSE:BUILD

1. Purpose:
   to store FID as first entry in CDT and if present to add first sequence number and increment to CDT.

2. Entry:
   B PARSE:BUILD
   This routine is one of the ones invoked via the CBRCHTBL of MASTERPARSER after a command has been identified.

3. Exit:
   The return destination, MASTEREXECUTIVE, is given in calling sequence to NXTPRM, and will be used upon recognition of a carriage return in the input command buffer. Error exit is to MASTERPARSER via ILGL$SEQ2 if increment in CDT is zero or if a sequence number range is found.

4. Operation:
   This subroutine uses the following:
   NEWCDTENTRY to build new CDT entry,
   GETFILEID to get file ID,
   ADDCDTPARAM to put alpha text in CDT,
   GETNEXTPARAM to get next parameter, and
   TYPEPERR to type error message.

8.4.3 PARSE:COMPRESS
PARSE:WRITE

1. Purpose:
   to add an entry to the CDT for

   $$\left\{\begin{matrix}\text{COMPRESS}\\ \text{WRITE}\end{matrix}\right\}\ \text{fid}\,[,\text{LIST}].$$

2. Entry:
   B PARSE:COMPRESS
   B PARSE:WRITE
   This routine is one of the ones invoked via CBRCHTBL of MASTERPARSER after a command has been identified.

3. Exit:
Normal return destination (MASTEREXECUTIVE) is given in calling sequence to NXTPARAM and will be used upon recognition of a carriage return in the input command buffer.

Error exit is to MASTERPARSER via TYPECERR after printing '-Pn:ILGL SYNTAX' if neither carriage return, comma nor alpha string found after file identification.

4. Operation:
A new CDT entry is built using NEWCDTENTRY. The file identification syntax is checked and PARAMBUF and PARAMBUFSZ set using GETFILEID. ADDCDTPARAM is used to add FID to the CDT. Using GETNEXTPARAM a further scan is performed for carriage return or ",LIST".

8.4.4 PARSE:COPY

1. Purpose:
to add an entry to the CDT for COPY fid_1 to fid_2 [,n[,i]]

2. Entry:
B PARSE:COPY
This routine is one of the ones invoked via the CBRCHTBL of MASTERPARSER after a command has been identified.

3. Exit:
Normal return destination (MASTEREXECUTIVE) is given in calling sequence to NXTPRM and will be used upon recognition of a carriage return in the input command buffer.

Alternate normal return is to GETSEQINCR if [,n[,i]] form of command is used.

Error exit is to MASTERPARSER after printing '-C1:ILGL SYNTAX' if keyword "ON" or "OVER" not used.

4. Operation:
This subroutine uses:
NEWCDTENTRY to build new CDT entry;
GETFILEID;
ADDCDTPARAM to put file ID in CDT and to put "ON" or "OVER" in CDT; and TYPECERR to type '-C1:ILGL SYNTAX' if found.

8.4.5 PARSE:MERGE

1. Purpose:
to add a new entry in the CDT for MERGE fid_1 [,n_1 [-n_2]] INTO fid_2, n_3[-n_4][,i].

2. Entry:
B PARSE:MERGE
This routine is one of the ones invoked via the CBRCHTBL of MASTERPARSER after a command has been identified.

3. Exit:
Normal return is B MASTERPARSER
Error exits are:
TYPEPERR after printing one of:
'-Pn:ILGL SYNTAX',
'-Pn:NOT SEQ#',
or '-Pn :ILGL SYNTAX'.
Alternate exit is B GET$INCREMENT to check for increment.

4. Operation:
   A scan is made to detect invalid command format.
   It uses: NEWCDTENTRY to set up new CDT entry;
   GETFILEID;
   ADDCDTPARAM to add to the CDT;
   GETNEXTPARAM to determine whether a record range has been specified and to convert n_1 and n_2 if given and also to scan for "INTO" and destination sequence number(s);
   ADJINT to compute sequence number *1000;
   and REPSEQ to duplicate value in PARAMBUF and store in PARAMBUF+1.

8.4.6 PARSE:DELETE
PARSE:LIST

1. Purpose:
   to add an entry to the CDT for DELETE fid or LIST fid.
2. Entry:
   B PARSE:DELETE
   B PARSE:LIST
   This routine is one of the ones invoked via the CBRCHTBL of MASTERPARSER after a command has been identified.
3. Exit:
   Normal return is to MASTEREXECUTIVE upon finding a carriage return.
   Error return is to MASTERPARSER via TYPEPERR on finding any character other than carriage return: it prints '-Pn:ILGL SYNTAX'.
4. Operation:
   This subroutine uses:
   NEWCDTENTRY to build new CDT entry;
   GETFILEID;
   ADDCDTPARAM to add entry to CDT;
   and GETNEXTPARAM to scan for carriage return.

8.4.7 PARSE:END

1. Purpose:
   to add an entry to the CDT for END.
2. Entry:
   B PARSE:END
3. Exit:
   Normal return is to MASTEREXECUTIVE on finding a carriage return.
   Error return is to MASTERPARSER via TYPEPERR on finding a character other than carriage return: it prints '-C1:ILGL SYNTAX'.
4. Operation:
   This subroutine uses:
   NEWCDTENTRY to build a new CDT entry and GETNEXTPARAM to scan for carriage return.

8.4.8 MASTEREXECUTIVE

1. Purpose:
   This is the master routine to execute the commands in the CDT. It resets CDTADR to point to start of CDT.

2. Entry:
   B MASTEREXECUTIVE

   This routine's address is given to NXTPRM* as the branch location following identification of a carriage return in the input command buffer.

3. Exit:
   B MASTERPARSER after finding erroneous data in CDT or after properly executing a command.

4. Operation:
   Commands are extracted from the CDT and executed one after another using the appropriate F:_____ routine until no commands remain.

8.4.9 F:BUILD

1. Purpose:
   to execute the command BUILD fid [,n[,i]].

2. Entry:
   F:LNK is the linkage register. This subroutine is entered via BAL, F:LNK F:BUILD from MASTEREXECUTIVE.

3. Exit:
   B *F:LNK

4. Operation:
   File identification fid is obtained from the CDT, and the file is opened for OUTPUT via OPENNEW. If the file already existed, the message "-FILE EXISTS; CAN'T BUILD" is printed via TYPEMSG and exit is taken. Starting sequence number is obtained from CDT if present; otherwise DFLTSEQ is used as default. Similarly the increment from the CDT is used, or if not found 1000 (assumed decimal point 1.000) is used. The EBCDIC mode for M:EI is specified via M:DEVICE. M:EI is read to obtain source records which are to be written into file fid. If the source records are compressed, they are de-compressed by routine RECONSTRUCT$LINE, their sizes are determined by SETEOD and the records written as part of file fid via WRITERANDOM.

   If the source records are not compressed, they are written as follows: the output buffer is first blank-filled using BLANKBUF; characters 1-72 from the source record are placed into the output buffer; the record length is determined by SETEOD and the records written as part of file fid via WRITERANDOM.

   Either a !EOD record for card reader input or an end-of-file for file input results in closing of the file, and exit is taken.

8.4.10 F:COMPRESS
F:WRITE

1. Purpose:
   to execute the file command

   $$\begin{Bmatrix}\text{COMPRESS}\\ \text{WRITE}\end{Bmatrix}\ \text{fid}[,\text{LIST}].$$

2. Entry:
   LNK is the linkage register. The subroutine is entered via BAL, LNK $\begin{Bmatrix}\text{F:COMPRESS}\\ \text{F:WRITE}\end{Bmatrix}$ from MASTEREXECUTIVE.

3. Exit:
   Normal exit is B *F:LNK

* a procedure which generates a calling sequence to GETNEXTPARAM

Error exits are to:
CPY40 if file fid is not present;
LIST80 if file fid is not keyed.

4. Operation:
The file identification fid is obtained from the CDT, and the file is opened via OPEN1. If it is not found or not keyed, the appropriate error exit is taken. If the LIST parameter is found in the CDT, an M:DEVICE procedure call is issued to skip to top of form for M:LO. A maximum of four bytes of fid are extracted from the CDT and used to specify sequencing of M:EO via M:DEVICE. CARDCOUNT is initialized to zero. An M:DEVICE procedure call is issued specifying binary mode for compressed or EBCDIC mode for non-compressed output. The records of file fid are read via READSEQUEN. After each has been read: CARDCOUNT is incremented; SETEOD is used to mark the last non-blank character of the record; if LIST was specified, the record is printed; RECSIZE is incremented; for COMPRESS the record is compressed in CARDIMG via COMPRESSLINE; for WRITE it is written via WRITE$EO. When end-of-file is encountered for a non-compressed file, CLOSE is used to close file fid and CLOSE$EO to close the output file. For a compressed file, RECSIZE is reset to 1.COMPRESSLINE is used to compress the final record and the first two bytes of COBUF are initialized to X'38FF' before closing the files.

8.4.11 F:COPY

1. Purpose:
to execute the copy command COPY $fid_1 \left\{\begin{matrix}ON\\OVER\end{matrix}\right\} fid_2 [,n [,i]]$.

2. Entry:
F:LNK is the linkage register. This subroutine is entered via BAL,F:LNK F:COPY from MASTEREXECUTIVE.

3. Exit:
B *F:LNK

4. Operation:
This subroutine examines fid_1 and fid_2 from CDT to determine whether they are equal. If they are not it tests for ON or OVER:
If ON, it uses OPEN2 to change file mode to OUT;
if fid_2 existed, attempt is to COPY A on A (error), in which case it prints '-P2:FILE EXISTS', closes file via CLOSE2 and exits;
if fid_2 did not exist it initializes for copy and proceeds with normal copy.

If OVER, it uses OPEN1 to open file with fid_1; it uses OPEN2 to open copy file with fid_2; if fid_2 already existed, it uses CLOSE3 to close it with REL and reopens it via OPEN3 as an output file.

Normal copy proceeds by typing '..COPYING' via TYPEMSG and copying either with old sequence numbers or sequence number range specified in command. If no sequence numbers were specified and: fid_1 is not a keyed file or if a record is written with a sequence number which already exists, the following message is printed via TYPEMSG: '-P1:FILE NOT SEQD & P3 NULL', both the files are closed, the copy file is deleted via DELETEFILE and exit is taken. WRITE2 is the routine used to write the copy file.
For $fid_1 = fid_2$:
If password is found, it prints 'PASSWORD ERROR" via TYPEMSG and exits. It opens copy file (F:EO) as output and input file (F:EI) as input and proceeds with normal copy.

Additional error which may occur is:
fid_1 cannot be opened in which case it prints '-P1:NO SUCH FILE' via TYPEMSG and exits.

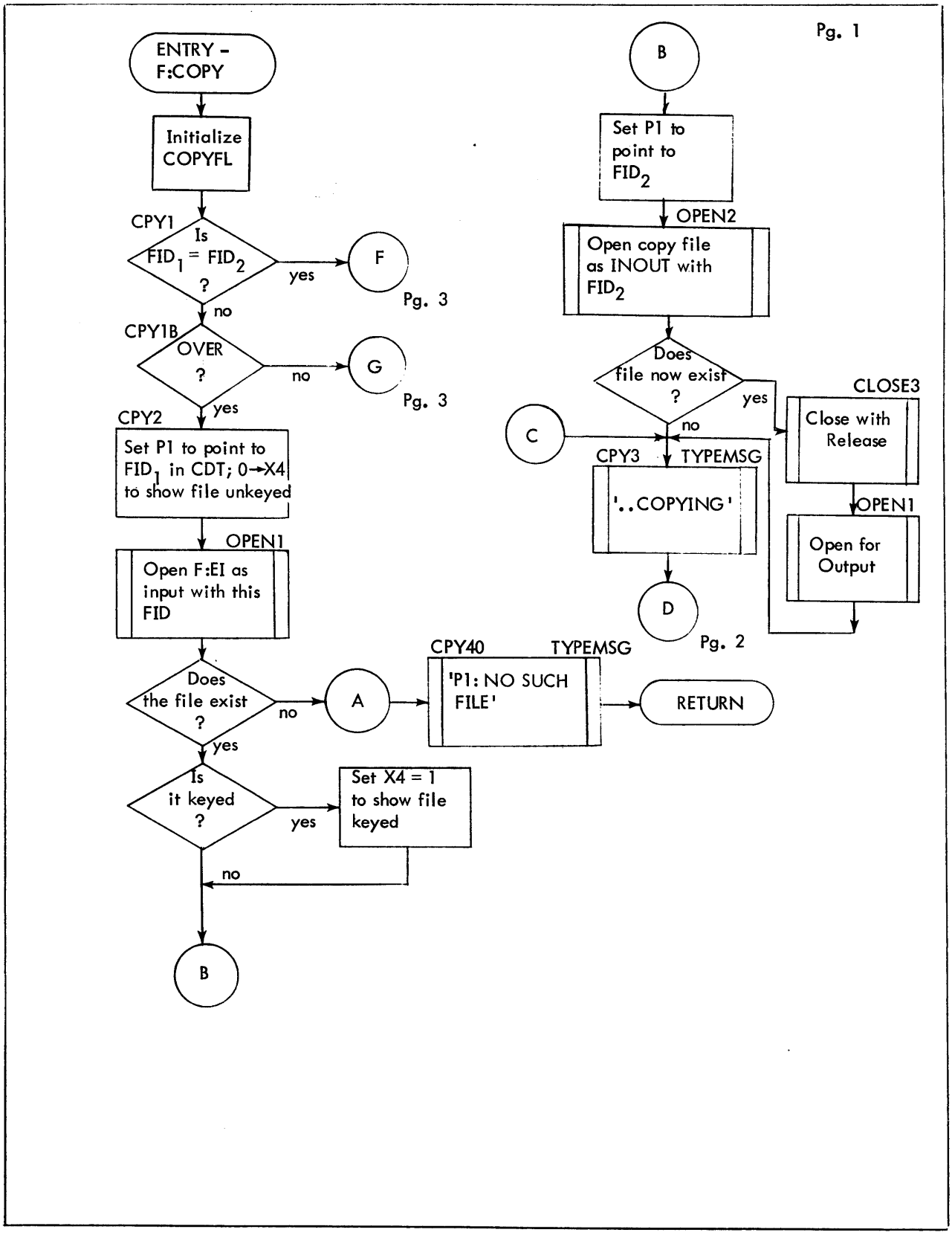


Figure 8-1A. Flow Diagram of F:COPY

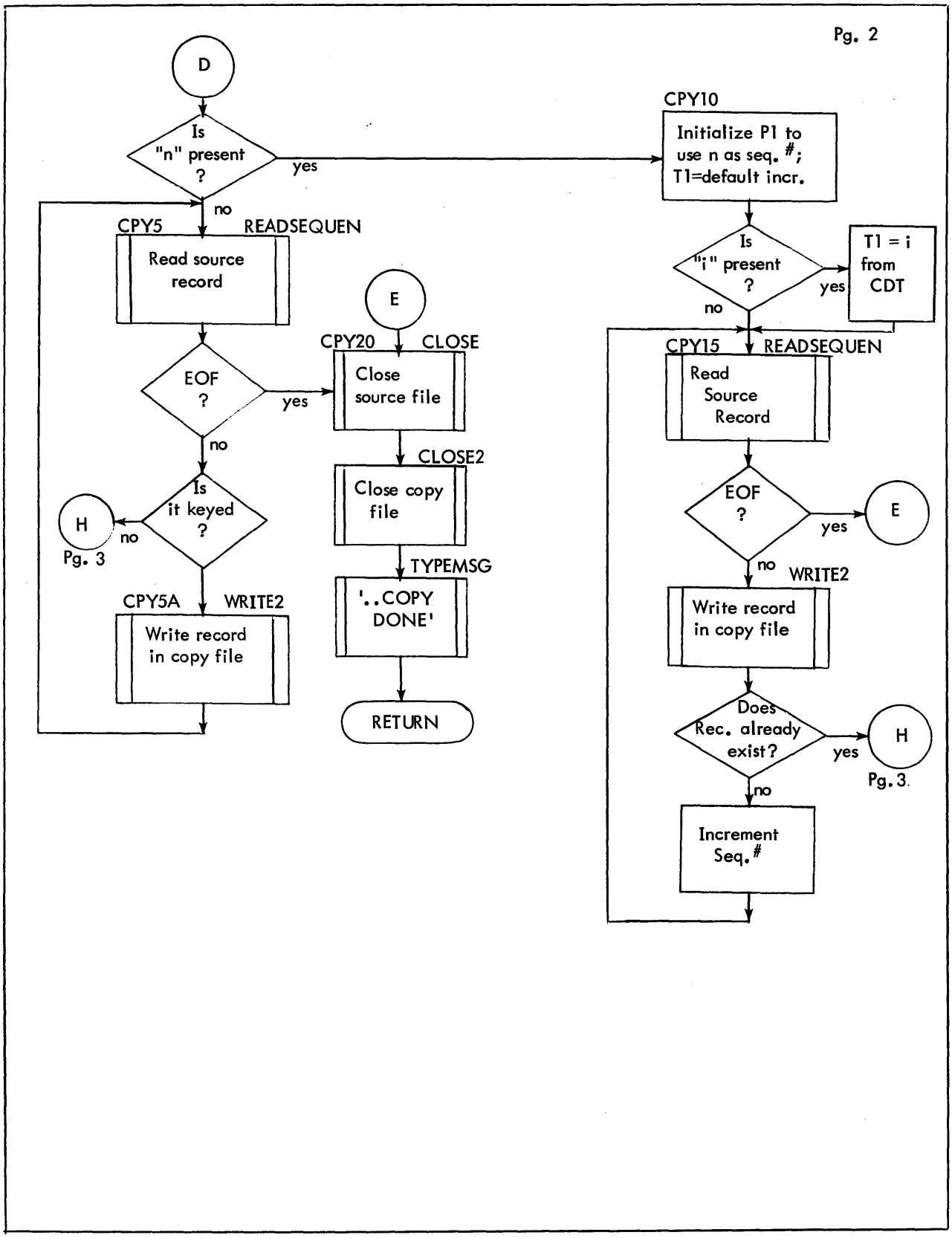


Figure 8-1A. Flow Diagram of F:COPY (Cont.)

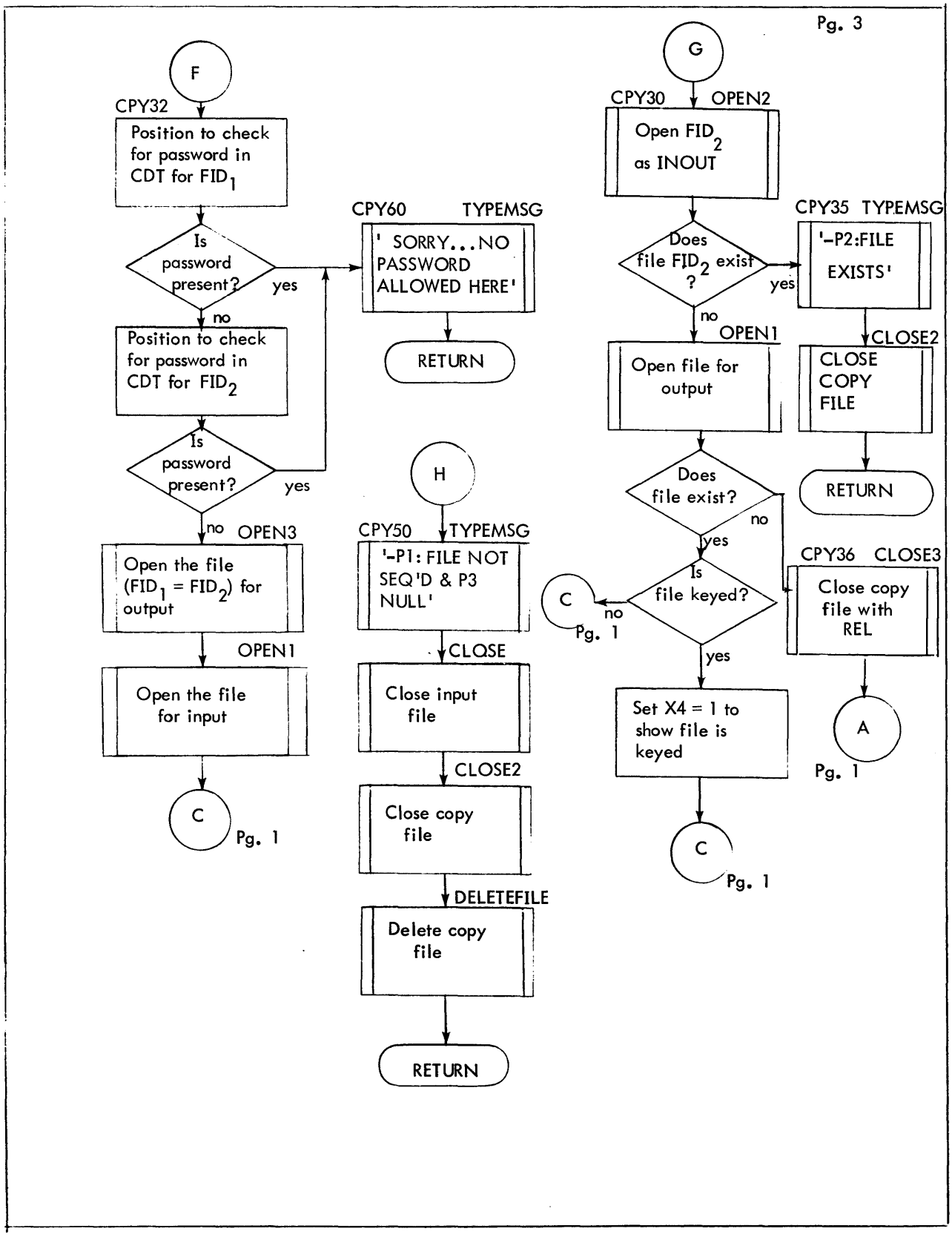


Figure 8-1A. Flow Diagram of F:COPY (Cont.)

8.4.12 F:DELETE

1. Purpose:
   to delete a file.
2. Entry:
   F:LNK is the linkage register. This subroutine is entered via BAL, F:LNK F:DELETE.
3. Exit:
   B *F:LNK which results in return to MASTEREXECUTIVE.
4. Operation:
   This subroutine uses DELETEFILE to delete the file, and it uses TYPEMSG to type appropriate message:
   '..DELETED'
   or '-NO SUCH FILE'.

8.4.13 F:END

1. Purpose:
   to return to the monitor.
2. Entry:
   F:LNK is the linkage register. This subroutine is entered via BAL, F:LNK F:END
3. Exit:
   It returns to the monitor via M:EXIT.

8.4.14 F:LIST

1. Purpose:
   to execute the list command LIST fid.
2. Entry:
   F:LNK is the linkage register. The subroutine is entered via BAL, LNK F:LIST from MASTEREXECUTIVE.
3. Exit:
   Normal exit: B MASTERPARSER
   Error exits are:
   B CPY40 if input file does not exist;
   B MASTERPARSER if file is not keyed.
4. Operation:
   File identification fid is obtained from the CDT and the file is opened for input via OPEN1. If the file does not exist, exit is to CPY40. If it is not keyed, the message
   '-FILE NOT KEYED; MUST COPY' is printed and exit is to MASTERPARSER.

   The location CARDCOUNT is used to identify record number opposite the record contents. Records of the file are read and printed using READSEQUEN and TYPECARD. SETEOD is used to determine individual record lengths.

8.4.15 F:MERGE

1. Purpose:
   to execute the merge command
   MERGE fid_1 [,n_1 [-n_2]]INTO fid_2, n_3 [-n_4][,i].
2. Entry:
   F:LNK is the linkage register. This subroutine is entered via
   BAL, F:LNK F:MERGE
   from MASTEREXECUTIVE.

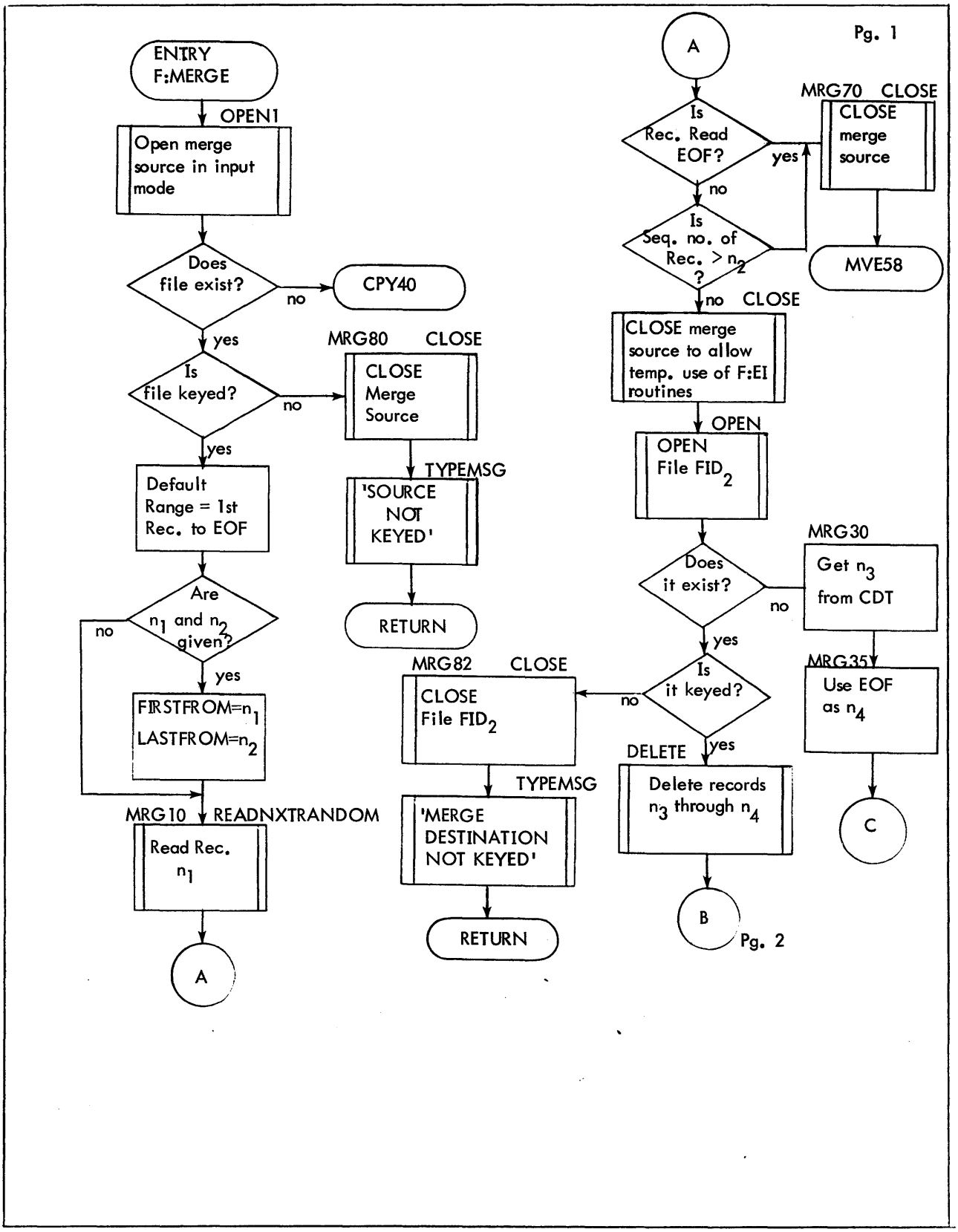


Figure 8-2. Flow Diagram of F:MERGE

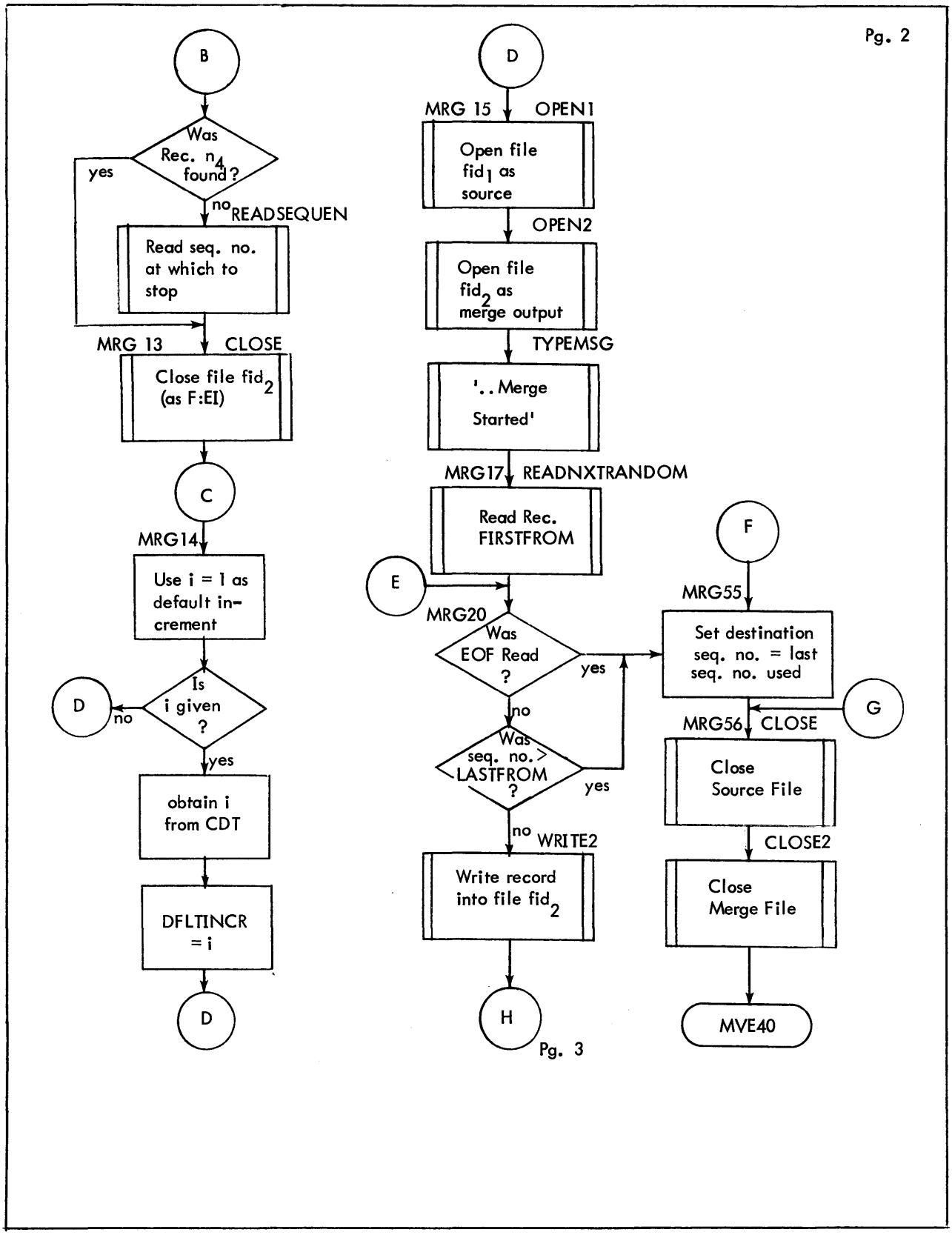


Figure 8-2. Flow Diagram of F:MERGE (Cont.)

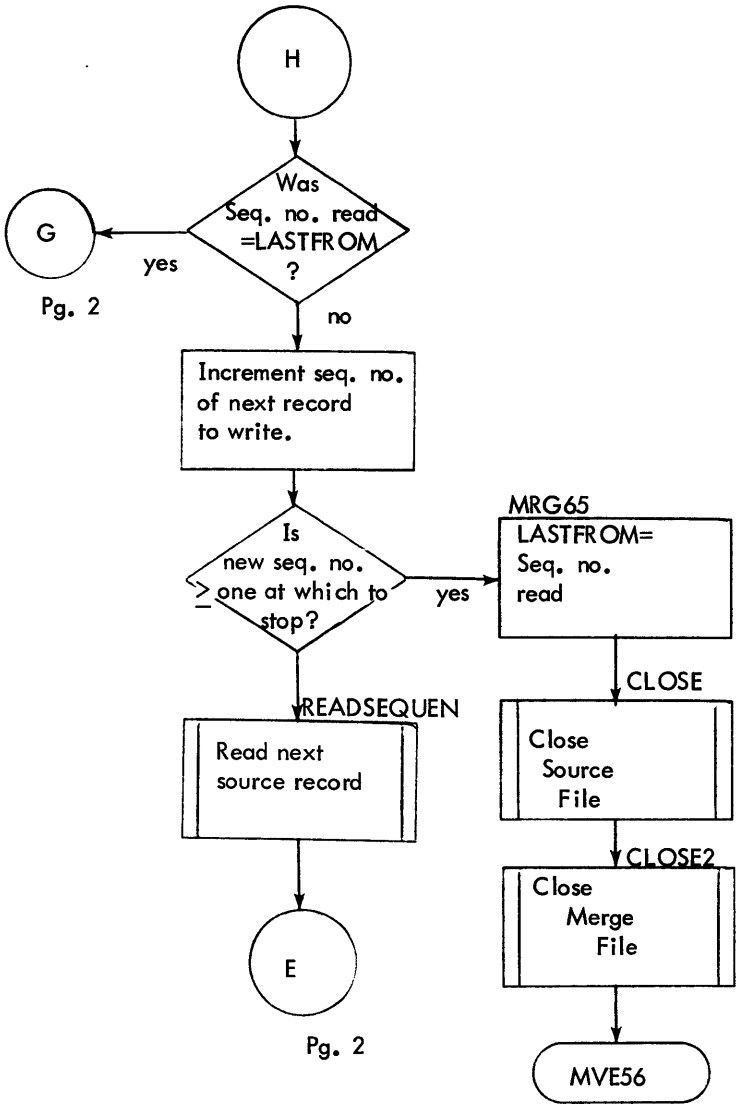


Figure 8-2. Flow Diagram of F:MERGE (Cont.)

3. Exit:
Normal: B MVE40 if destination range completed;
Error exit to MVE58 if source range not found;
Error exit to MVE56 if cutoff occurs;
Error (all other errors): B *F:LNK

4. Operation:
This subroutine uses OPEN1 to open source file in input mode;
if not found it types '-P1:NO SUCH FILE';
if not keyed it types '-MERGE SOURCE NOT KEYED'.
It uses either the range as specified (n_1 $-n_2$) or defaults to entire file.
It attempts to read source file via READNXTRANDOM:
if EOF encountered, it closes source file via CLOSE and exits
to MVE56 (see R:MOVE$DELETE and R:MOVE$KEEP).
It attempts to open fid_2:
if non-existent it picks up n_3 from CDT and proceeds;
if not keyed it closes the file, types
'MERGE DESTINATION NOT KEYED' via TYPEMSG and
exits to MASTEREXECUTIVE.

Using n_3 [$-n_4$]from CDT it uses DELETE to delete records of fid in that range.

It re-opens fid_1 as input and fid_2 as INOUT and types
'...MERGE STARTED via TYPEMSG'.

It reads records from fid_1 using READNXTRANDOM, writes them into fid_2
using WRITE2 until either EOF reached or range completed;
it closes files and branches to MVE40 (see R:MOVE$DELETE and
R:MOVE$KEEP).

8.4.16 ADDCDTPARAM

1. Purpose:
to add a new parameter to the Command Description Table.

2. Entry:
LNK is linkage register used. This subroutine is entered via BAL,LNK ADDCDTPARAM
from several PARSE: routines when it is desired to add to the CDT.
Upon entry P1, PARAMPSN, and PRMBUFSZ are as shown below.

3. Exit:
B O,LNK with expanded CDT entry.

4. Operation:
Words are added to the CDT from PARAMBUF according to the format shown in Table 8-2 using
the following input parameters:
P1 = type of parameter,
PARAMPSN = next available slot in CDT,
PRMBUFSZ = number of words to be added.

8.4.17 ADJINT

1. Purpose:
to form a sequence number as an integer *1000.

2. Entry:
LNK is the linkage register. This subroutine is entered via BAL,LNK ADJINT
from PARSE:BUILD or PARSE:MERGE.

3. Exit:
B *LNK

4. Operation:
This routine multiplies the sequence number in PARAMBUF by 1000 and stores it back in PARAMBUF.

8.4.18 BINTODEC

1. Purpose:
to convert a binary number to decimal.

2. Entry:
LNK is the linkage register.
Upon entry:
P1 contains binary number, and
P2 contains byte address where decimal string is to be stored (right-most byte).
This subroutine is used by MOVESEQ and TYPESEQ.

3. Exit:
Return is to calling routine via B O,LNK with decimal properly stored.

4. Operation:
BINTODEC divides binary number by 10, adds zone bits to remainder (i.e., X'FO'), stores it, moves pointer one to left in output string and repeats until seven digits have been converted.

8.4.19 BLANKBUF

1. Purpose:
to store blanks in CARDIMG.

2. Entry:
LNK is the linkage register.
This subroutine is used by READRANDOM and READSEQUEN. It is called using BAL,LNK BLANKBUF.

3. Exit:
Upon exit blanks are stored CARDIMG. Return is made to calling routine via B O,LNK.

8.4.20 CLOSE

1. Purpose:
to close input file with SAVE.

2. Entry:
LNK is the linkage register.
This subroutine is used by F: routines and TESTEDITACTIVE. It is called using BAL,LNK CLOSE.

3. Exit:
Return is made to calling routine via B O,LNK.

8.4.21 CLOSE2

1. Purpose:
to close output (COPY) file with SAVE.

2. Entry:
LNK is the linkage register.
This subroutine is used by F: routines. It is called using BAL,LNK CLOSE2.

3. Exit:
Return is made to calling routine via B O,LNK.

8.4.22 COMPRESSLINE

1. Purpose:
   to compress a record.

2. Entry:
   LNK is linkage register. This routine is entered via BAL, LNK COMPRESSLINE from F:COMPRESS. Input consists of a record image in CARDIMG.

3. Exit:
   B O, LNK

4. Operation:
   Registers are initialized as follows:
   T1 = 0, blank count;
   T2 = count of total bits used in CO record from COBUSED;
   X2 = count of bits left in current word in COBUF from COBLEFT;
   X3 = record size in bytes from RECSIZE.

   If record size is zero (i.e., and of output file has been reached), subroutine CMPL60 is used to append the 6-bit end-of-file code (item "3") to the record byte 1 of the record is changed from X'38' to X'18', the record is written via WRITECO, COBLEFT and COBUSED are reset, and exit is taken.

   If record size is greater than zero, the source record in CARDIMG is translated into compressed format and stored in COBUF. Characters A-Z, 0-9 and certain special characters found in SCCTAB have 6-bit equivalent codes (compressed items). All other characters are represented by their 8-bit source codes preceded by compressed item "4". If the character can be represented in 6 bits, subroutine CMPL60 is passed a bit count of 6 in P2 and is used to edit the 6-bit code "4" in P1 (use 8-bit character that follows) into COBUF and then the 8-bit code itself in P1 into COBUF. This process continues until the end of the record is reached. If the last output record contains fewer than 80 bytes, it is padded to 80. The end-of-line item "2" is added, COBLEFT and COBUSED are reset and exit is taken.

8.4.23 CMPL60

1. Purpose:
   to control the editing of one 6- or 8 - bit compressed "item" into COBUF.

2. Entry:
   LNK is the linkage register. This subroutine is entered via BAL, LNK CMPL60 from COMPRE SLINE. Upon entry:
   P1 = compressed code (item) for one character;
   P2 = length of code in P1 (6 or 8).

3. Exit:
   B O, LNK

4. Operation:
   If the blank count in T1 is zero, subroutine CMPL70 is used to enter the item into COBUF and exit is taken. If it is non-zero further tests are performed. If it is 1, item "7" is edited into COBUF via CMPL70. If the blank count + 1 is equal to or less then 64, item "5" is edited into COBUF via CMPL70 followed by the count n also edited into COBUF via CMPL70. Similarly, if the blank count + 1 exceeds 64, item "6" is edited into COBUF followed by count n edited in line manner. T1 is reset to zero and exit is taken.

8.4.24 CMPL70

1. Purpose:
to enter a compressed "item" into COBUF and if buffer is filled to write record via WRITECO.

2. Entry:
LNK is the linkage register. This subroutine is entered via BAL,LNK CMPL70 from CMPL60. Upon entry:
P1 contains compressed item;
P2 contains bit count of the item;
registers T2 and X2 are as set in COMPRESSLINE.

3. Exit:
B O,LNK

4. Operation:
The bit count in P2 is added to T2. If the sum exceeds or equals the total allowed per output record, WRITECO is used to write a record of compressed output before current byte is edited into the buffer, COBUF. Editing, i.e., adding, the item into the buffer COBUF takes place as follows: Item length in bits is subtracted from the count of bits remaining to be filled in the current word in COBUF leaving a count in X2. If there is enough room remaining to allow the item to be added, it is shifted left the number of places now specified in X2. The item is then added into COBUF by indirectly addressing COWORD, the address of the destination word. Exit is then taken. If there is insufficient room remaining in the current word in COBUF, the item is split between registers X4 (R) and X1 (Ru 1). This is done by shifting it right leaving in X4 only the number of bits which can be contained in the current word in COBUF with the remaining bits in X1. The bits in X4 are added into the current word in COBUF, the address in COWORD is incremented, and the bits in X1 are added into the next word in COBUF. Register X2 is incremented by 32 making it equal to the number of unfilled bits in the current word in COBUF. Exit is taken.

8.4.25 DELETE

1. Purpose:
to delete records in a specified range.

2. Entry:
LNK is the linkage register. This subroutine is entered via BAL,LNK DELETE from F:MERGE.
Upon entry
P1 = sequence number of first record to delete and
P2 = sequence number of last record to delete.

3. Exit:
Normal exit is B O,LNK with CC1 = 0 to indicate that last sequence number was found.
Error exit is B O,LNK with CC1 = 1 to indicate that last sequence number was passed.
Upon exit in either case:
R1 = sequence number of last record read,
R2 = number of records deleted.

4. Operation:
1. An attempt is made to read the record specified in P1 using READNXTRANDOM.
2. If end-of-file is encountered, the message '--EOF HIT' is printed via TYPEMSG and error exit is taken. If the record obtained has a higher sequence number than that in P2, the error exit is taken. Otherwise, the record is deleted using DELETERECORD. The next record in the file is read using READSEQUEN. Testing at step a. is resumed until either end-of-file is reached or the sequence number specified in P2 is encountered (normal case) or passed.

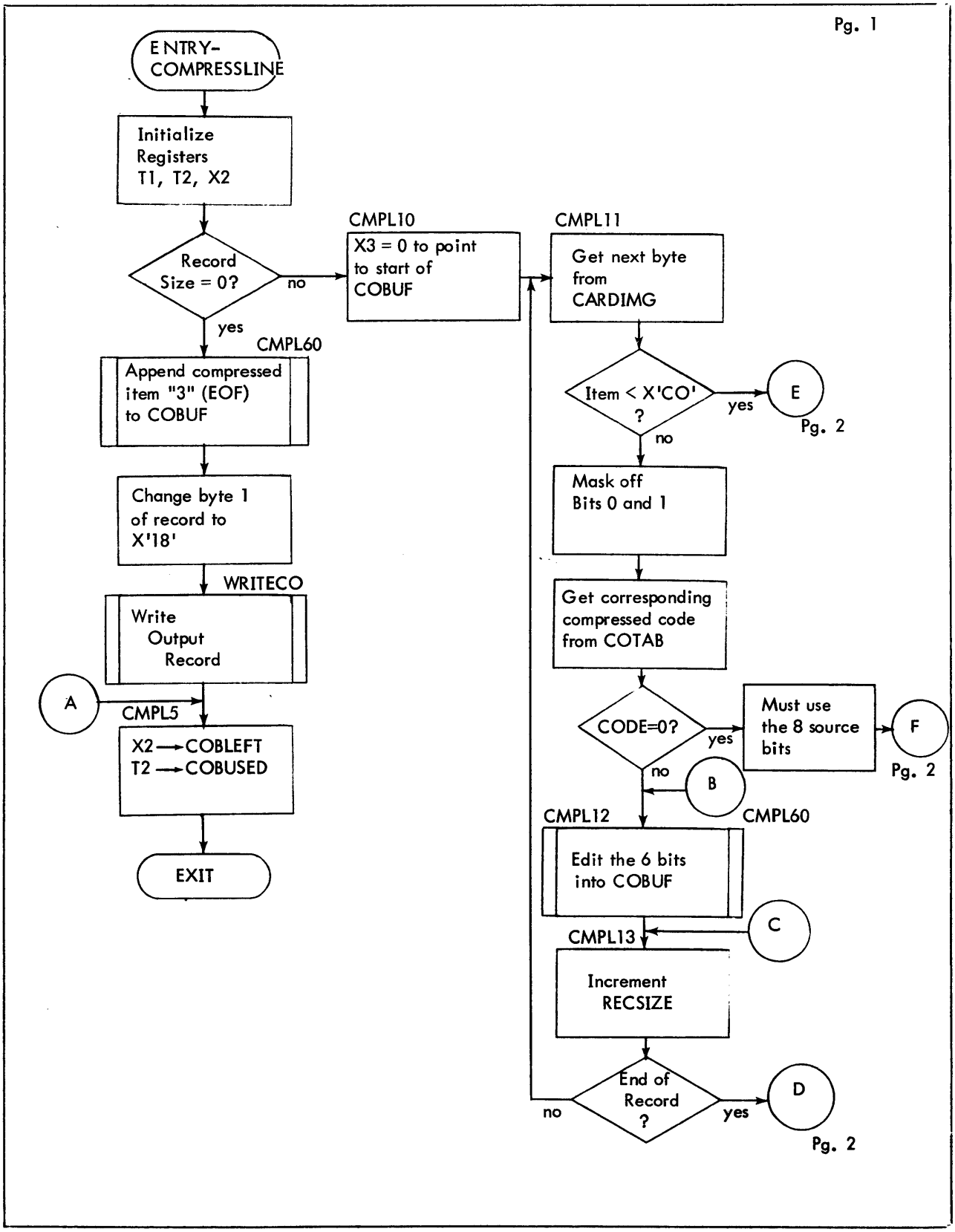


Figure 8-3. Flow Diagram of COMPRESSLINE

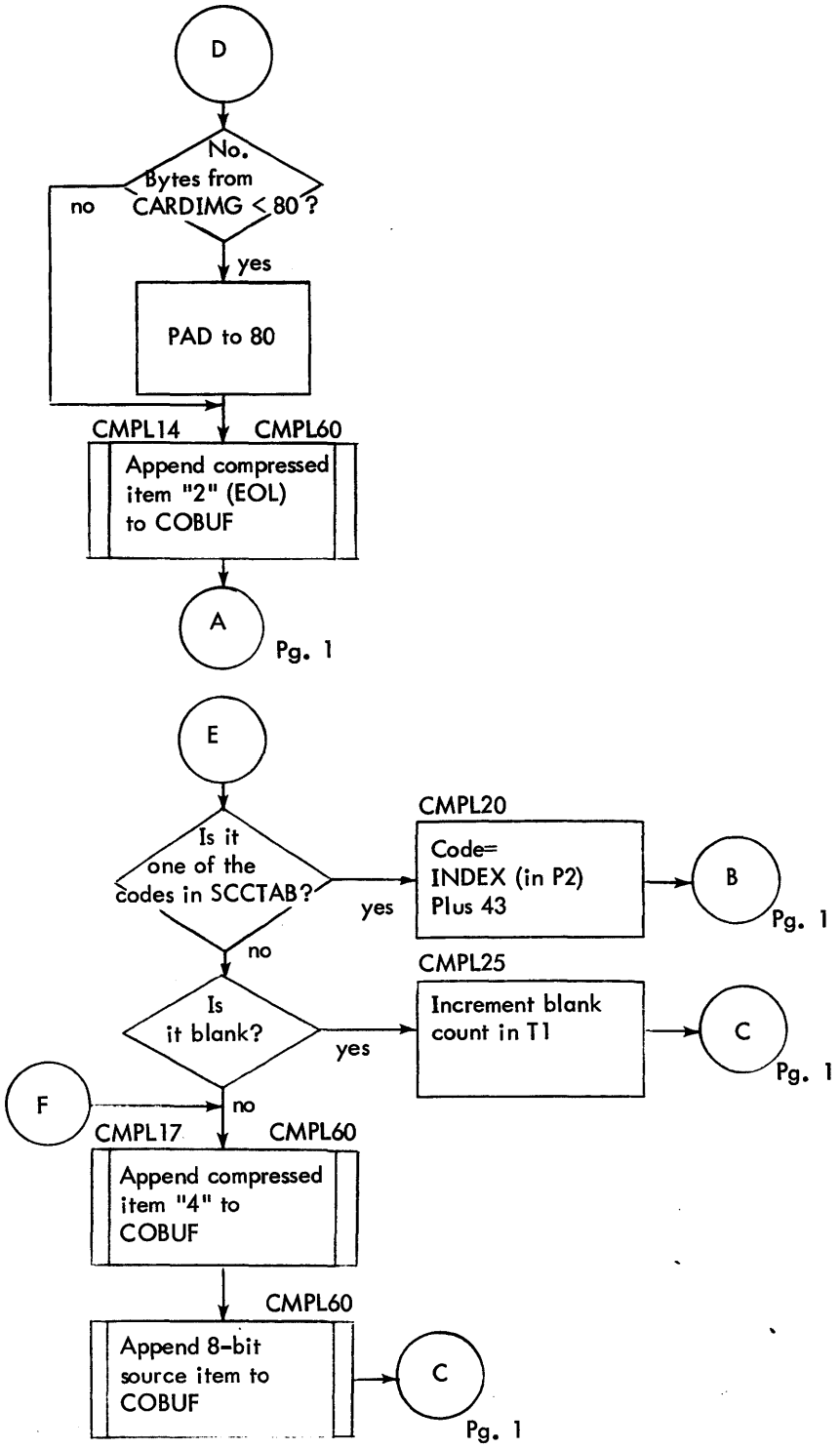


Figure 8-3. Flow Diagram of COMPRESSLINE (Cont.)

8.4.26 DELETEFILE

1. Purpose:
   to delete a file and set CC1.

2. Entry:
   LNK is the linkage register. This subroutine is used by F:COPY and F:DELETE routines. Upon entry P1=address of file ID in Command Description Table. It is entered via BAL,LNK DELETEFILE.

3. Exit:
   CC1 upon exit contains: 0 if file was deleted,
   1 if file does not exist.
   Return is B O,LNK to calling routine.

4. Operation:
   This subroutine uses OPENINIT to open file.
   If file does not exist it sets CC1 = 1.
   If abnormal I/O for another reason, it branches to BADIO1 which prints 'BAD I/O; ABN CODE XX'.

8.4.27 BADIO1

1. Purpose:
   to print abnormal I/O message.

2. Entry:
   At BADIO, error code is loaded into X1 from D1.
   At BADIO1, error code is assumed to be in X1.
   Entry is with branch, because there is no return to calling routine.
   BADIO1 is called by various open, read and write routines.

3. Exit:
   It returns to the Monitor via M:ERR.

4. Operation:
   It sets up error code in message line and prints message.

8.4.28 GETFILEID

1. Purpose:
   to check syntax of FID and if good to set PARAMBUF and PRMBUFSZ.

2. Entry:
   LNK is the linkage register. This subroutine is entered via BAL,LNK GETFILEID from PARSE:BUILD, :COPY, :MERGE, and :EDIT to obtain the file ID so it can be placed in the Command Description Table.

   The file ID is assumed to be in the Teletype input buffer.

3. Exit:
   File ID is now in PARAMBUF as follows:
   PARAMBUF

TEXTC filename
TEXTC acct or 0
TEXTC password or 0

   and it has been checked for proper format.

   Return is to calling routine via B O,LNK.
   Error returns are via GETNEXTNAME:
   messages include '-P1:BAD FID'
   '-C1:CMND ILGL HERE',
   '-P1:ILGL SYNTAX'.

4. Operation:
This routine rejects a file name which is longer than 31 characters. It uses GETNEXTNAME first to build file name in PARAMBUF. It then pushes the name from PARAMBUF into a stack. It repeats this process for account and for password, if present. It then pulls the entries from the stack and stores them in PARAMBUF and stores in PRMBUFSZ the length in words of the entries in PARAMBUF.

8.4.29 GETNEXTNAME

1. Purpose:
to get the next name from the Teletype input buffer (TTYIMG) and place it in PARAMBUF.

2. Entry:
LNK is the linkage register. This subroutine is entered by invoking the NXTNAM command procedure resulting in the calling sequence:

```
BAL,LNK     GETNEXTNAME
GEN, 8, 24  # of branches, addr. of error msg.
GEN, 8, 24  type 1, branch addr. 1
   .                .
   .                .
   .                .
GEN, 8, 24  type n  , branch addr. n
```

3. Exit:
Upon exit a name (file, acct or password) resides in PARAMBUF in TEXTC format.

4. Operation:
Input characters from TTYIMG are scanned and tested to determine whether they are part of a name. They are placed in PARAMBUF in TEXTC format. When a terminator is found (e.g. comma or right parenthesis) the scan is stopped, and the name is padded to the right with three blanks. If an error is found, the error message given in the calling sequence is printed and return is made to MASTERPARSER via TYPECERR.

8.4.30 GETNEXTPARAM

1. Purpose:
This routine scans the Teletype input buffer to isolate recognizable character strings which comprise EDIT commands and to place them in PARAMBUF.

2. Entry:
This routine is called by various PARSE:_____routines. LNK is the linkage register. The routine is invoked by the NXTPRM procedure which sets up a calling sequence as follows:

```
BAL,LNK     GETNEXTPARAM
GEN,8,24    # of branches, addr. of error msg.
GEN,8,24    completion type, branch addr.1
   .                .
   .                .
GEN,8,24    completion type, branch addr.  n
```

3. Exit:
Upon exit a parameter is in PARAMBUF in TEXTC format. Return is to the branch address in bits 8-31 of the word in the calling sequence of which a match was found on bits 0-7. Error return is to MASTERPARSER via TYPEPERR.

4. Operation:
A scan is made character by character with tests being performed to detect invalid command format. Examples: slash (/) must not be the last character of a command; a sequence number must not exceed three digits; the second sequence number in a range must not be greater than the first. Error messages include:

```
'-Pn:ILGL SEQ#'
'-Pn:SEQ2<SEQ1'
```

in addition to that given in the calling sequence.

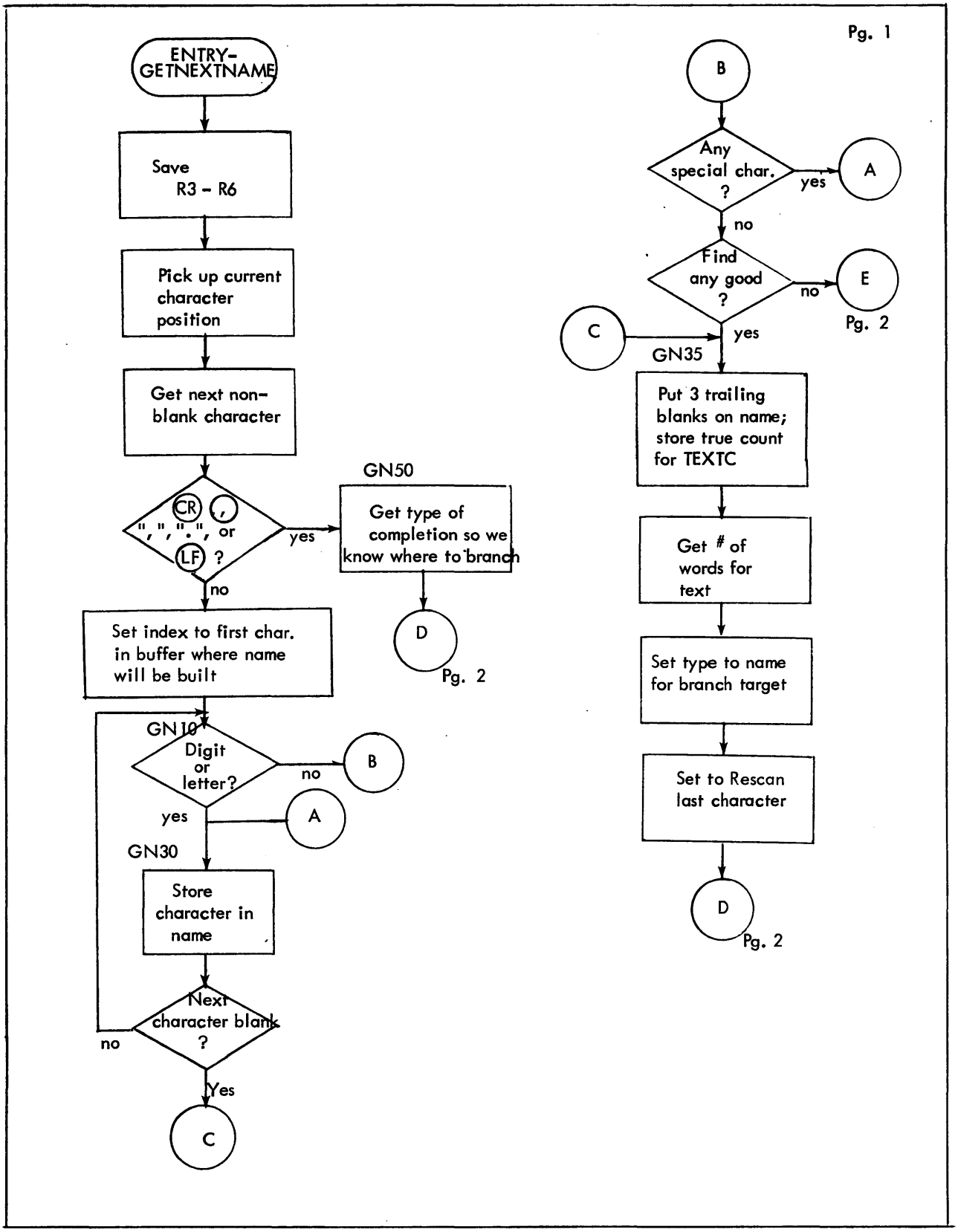


Figure 8-4. Flow Diagram of GETNEXTNAME

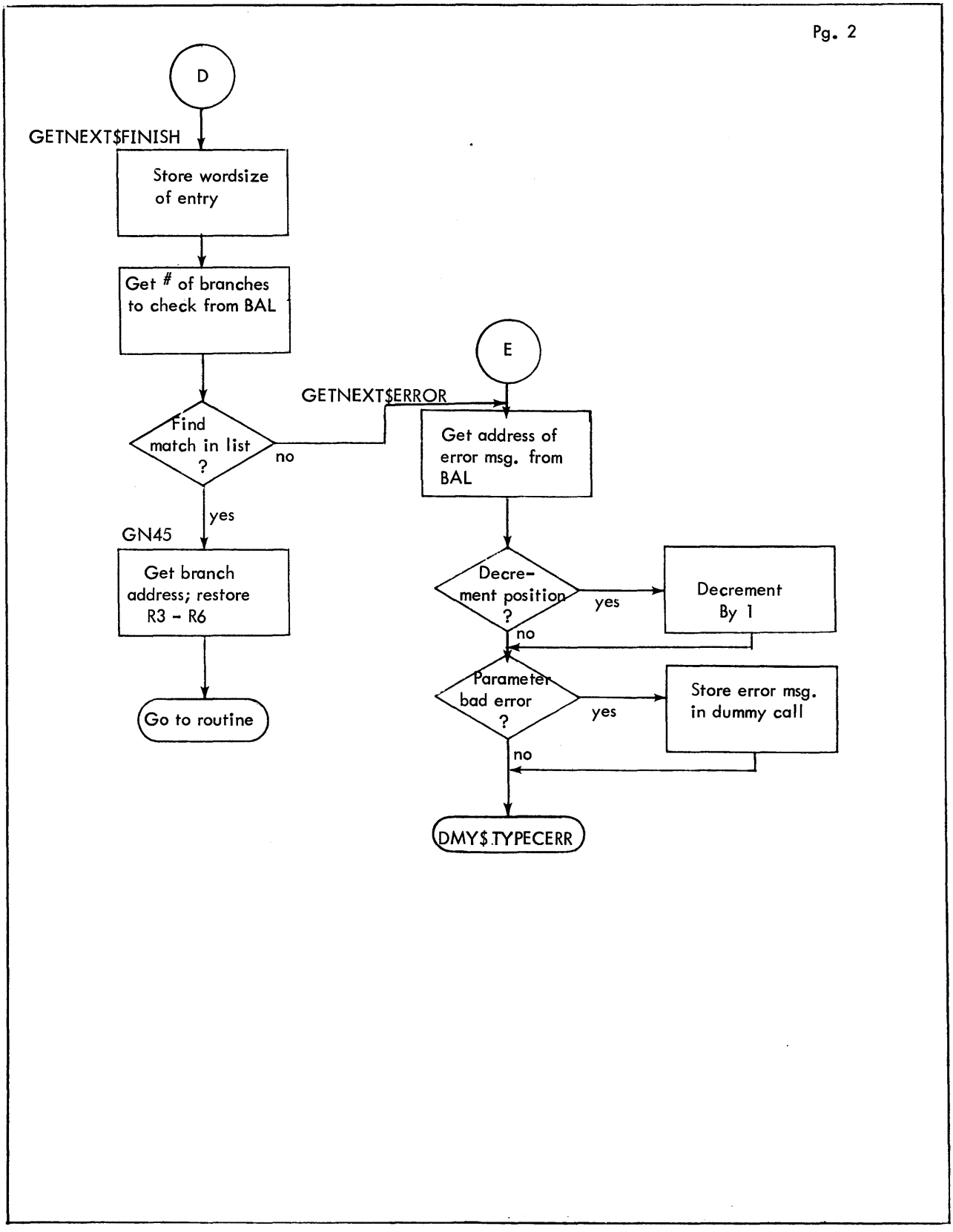


Figure 8-4. Flow Diagram of GETNEXTNAME (Cont.)

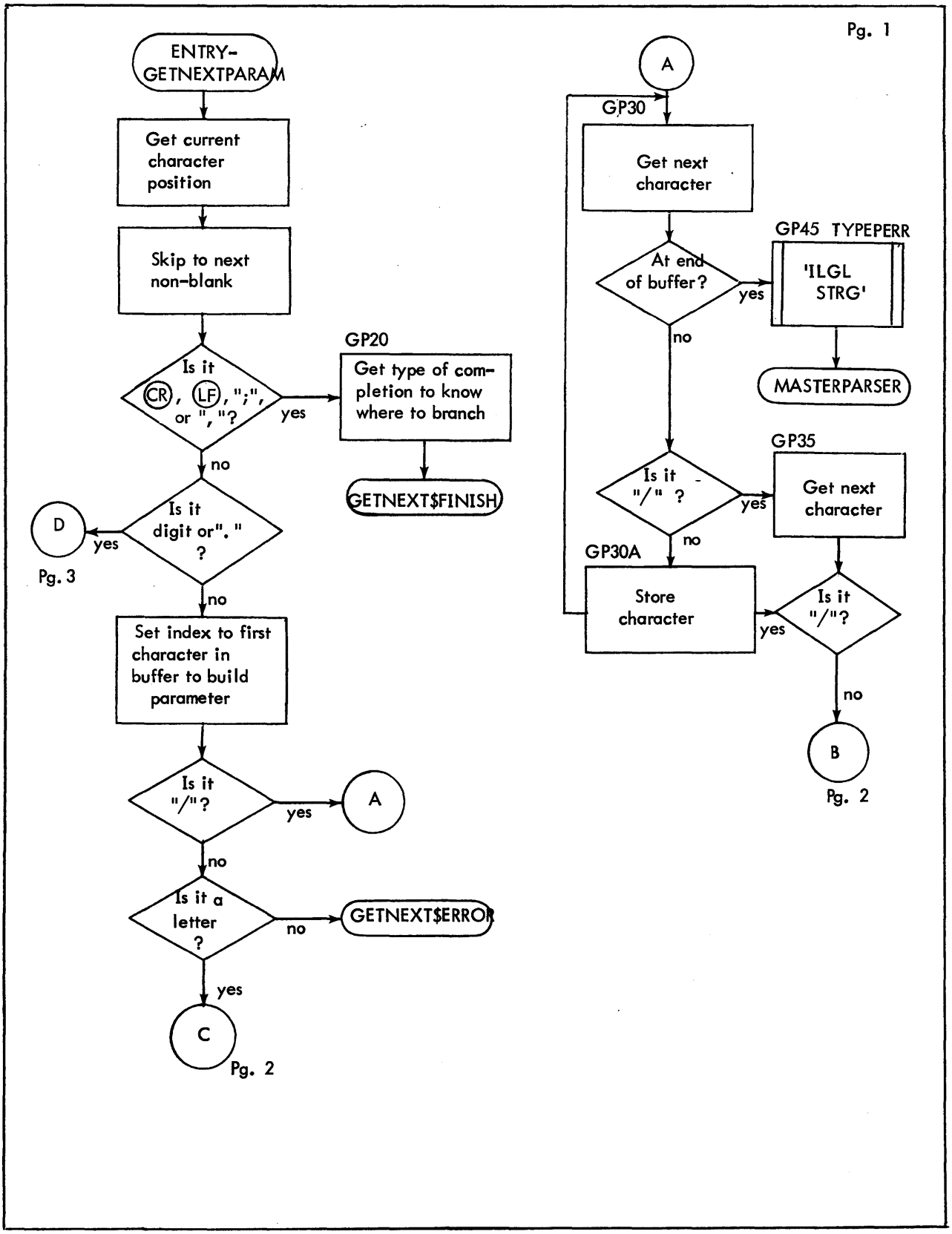


Figure 8-5. Flow Diagram of GETNEXTPARAM

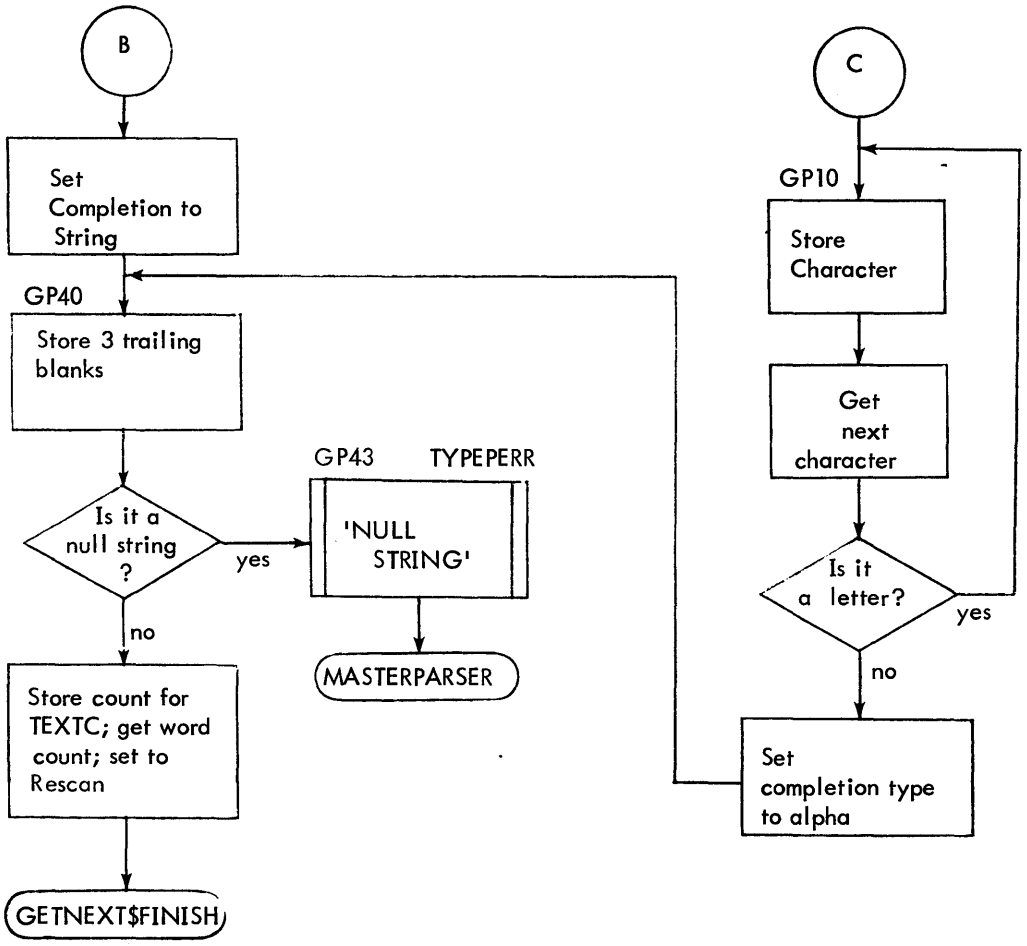


Figure 8-5. Flow Diagram of GETNEXTPARAM (Cont.)

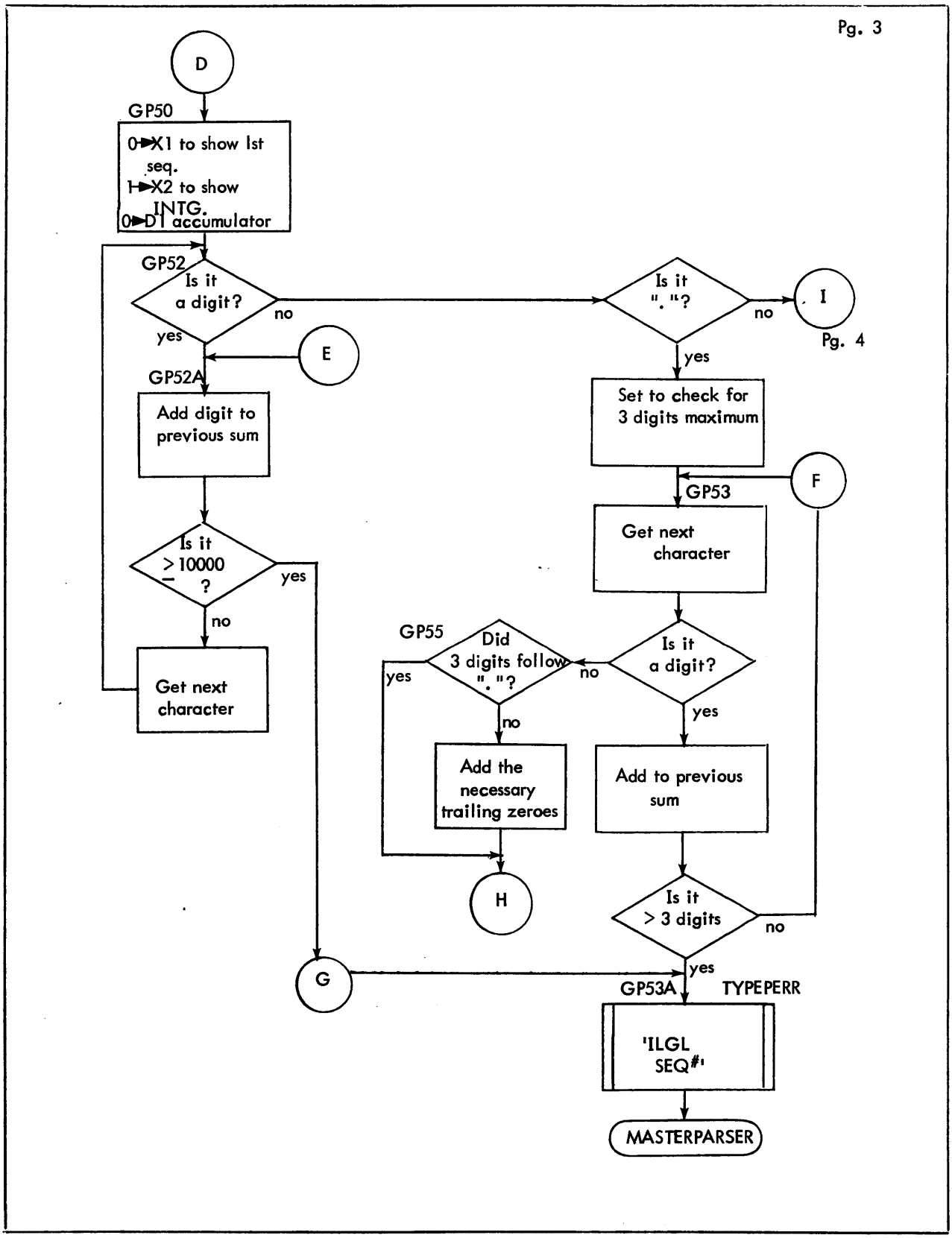


Figure 8-5. Flow Diagram of GETNEXTPARAM (Cont.)

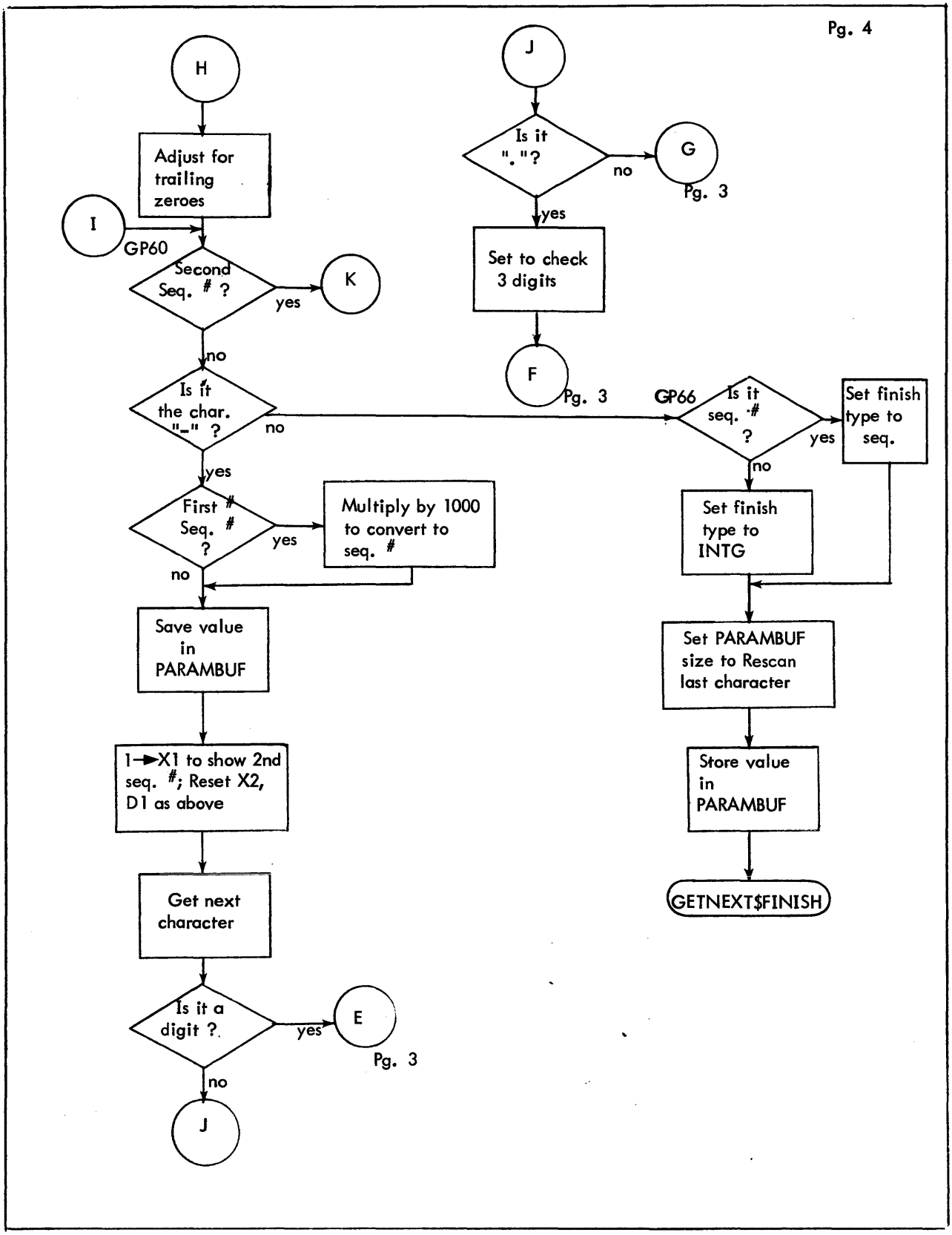


Figure 8-5. Flow Diagram of GETNEXTPARAM (Cont.)

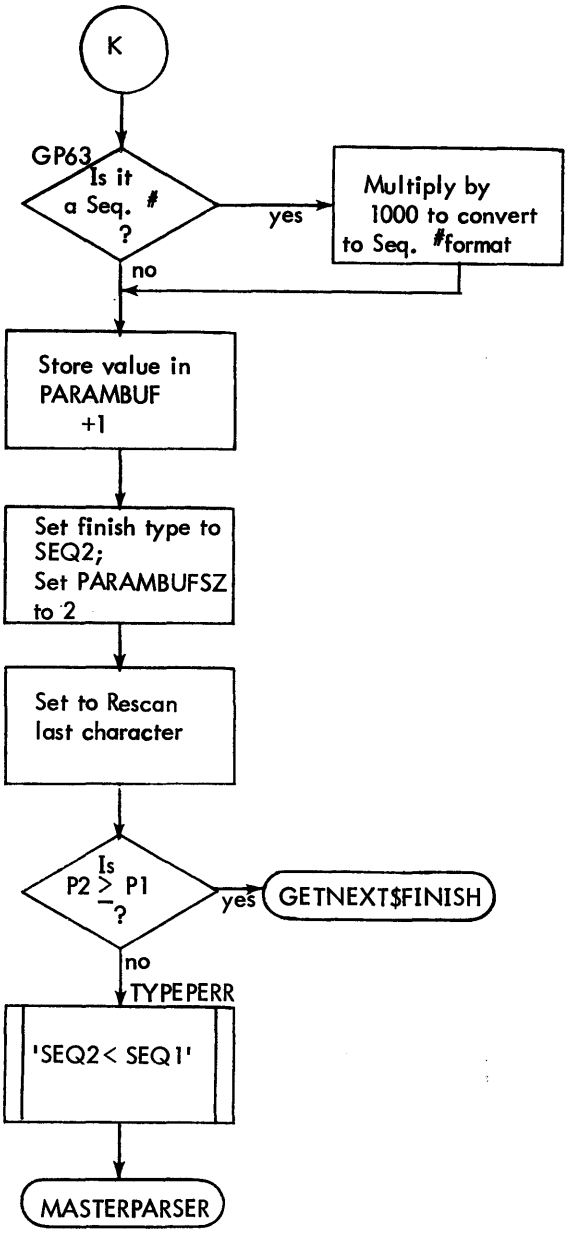


Figure 8-5. Flow Diagram of GETNEXTPARAM (Cont.)

8.4.31 MOVESEQ

1. Purpose:
   to format sequence number in EBCDIC as 'XXXX.XXX' having four characters from calling sequence appended.

2. Entry:
   LNK is the linkage register. This subroutine is entered via BAL, LNK MOVESEQ from of the following routines:

   R:MOVE$DELETE,
   R:MOVE$KEEP,
   READSEQUEN.

   Upon entry:
   P1 = sequence number to be converted to EBCDIC;
   P2 = byte address at which to put the string;
   word following the BAL contains four characters to be appended to the sequence number.

3. Exit:
   Upon exit R1 contains the number of characters in the resultant string.
   Exit is B 1, LNK.

4. Operation:
   It uses BINTODEC to convert sequence number to EBCDIC. It places string in TEMPBLCK with leading zeroes suppressed and the requested characters appended at the right.

8.4.32 NEWCDTENTRY

1. Purpose:
   to set up room in the CDT for a new entry.

2. Entry:
   LNK is the linkage register. This subroutine is entered from several PARSE: routines.
   Upon entry: P1 contains the number of the command type to be added; word following the BAL contains the number of parameters.

3. Exit:
   CDT entry is initialized as follows:
   word 0: byte 0 contains length of entry (= 0 initially)
   byte 1 contains command type (or number)
   e.g. 0 for carriage return,
   1 for file name, etc.
   byte 2 contains number of this entry in the CDT
   byte 3 contains number of parameters
   words [1⟶ # of parameters/2] : zeroes
   word [# of parameters/2+1] : X'00000100'

8.4.33 OPEN

1. Purpose:
   to open an update file.

2. Entry:
   LNK is the linkage register. For F:EDIT and F:MERGE, entry is via BAL,LNK OPEN to open an update file. For F:COPY and F:MERGE entry is via BAL,LNK OPEN1 to open a copy input file.
   Upon entry P1 = address of file ID in CDT.

3. Exit:
   CC1 = 1 if file does not exist
   = 0 otherwise
   CC2 = 1 if file is not keyed
   = 0 otherwise
   Error exit is to BADIO1 if requested file does not exist.
   Normal return is to calling routine via B O,LNK.

4. Operation:
   This subroutine sets mode to INOUT if entered via OPEN or to IN if entered via OPEN1.
   It uses OPENINIT to try to open file.
   If it cannot open file it sets CC1 and returns.
   If file is not keyed it sets CC2 and exits.
   Otherwise it sets CC1 and CC2 = 0.

8.4.34 OPENINIT

1. Purpose:
   to initialize an OPEN FPT.

2. Entry:
   LNK is the linkage register. This subroutine is used by DELETEFILE, OPEN and OPEN3. It is entered via BAL,LNK OPENINIT.
   Upon entry: P1 = address of file ID in CDT;
   T1 = FPT entry at which file name is to go;
   T2 = FPT entry at which account number is to go;
   T3 = FPT entry at which password is to go.

3. Exit:
   Return is to calling routine via B O,LNK.

4. Operation:
   It moves file name, account and password from CDT to FPT locations and sets control words properly.

8.4.35 OPENNEW

1. Purpose:
   to open a file for output.

2. Entry:
   LNK is the linkage register. This subroutine is used by F:BUILD.
   It is entered via BAL,LNK OPENNEW.
   Upon entry P1 = address of file ID in CDT.

3. Exit:
   CC1 = 1 if file not exist and the open in OUT mode was successful.
   CC1 = 0 if file already existed and could not be opened as an output file.
   Normal return is to calling routine via B O,LNK.
   Error return is branch to BADIO1 if it could not open file even through it was present.

4. Operation:
   It sets mode to INOUT.
   It uses OPENINIT to set FPT and tries to open file:
   if file exists, it sets CC1 = 0 and returns;
   if file exists but could not be opened it exits to BADIO1;
   if file does not exist, it changes mode to OUT and issues open.

8.4.36 OPEN2, OPEN3

1. Purpose:
   to open an output file for copying.

2. Entry:
   LNK is the linkage register. OPEN 2 is called by F:COPY and F:MERGE and is entered via BAL,LNK OPEN2.OPEN3 is called by F:COPY and is entered via BAL,LNK OPEN3.

3. Exit:
   CC1 = 1 if file not successfully opened.
   CC1 = 0 otherwise.
   Error exit is to BADIO1.
   Normal return is to calling routine via B O,LNK.

4. Operation:
   If entered via OPEN2 it sets mode to INOUT.
   If via OPEN3 it sets it to OUT.
   It calls OPENINIT and attempts to open file:
   if successful it sets CC1=0;
   if unsuccessful, it tests to see if it were because file did not exist:
   if so it issues open for out file;
   if for any other reason, it goes to BADIO1.

8.4.37 READNXTRANDOM

1. Purpose:
   to read random record or next highest one.

2. Entry:
   LNK is the linkage register. This subroutine is entered via BAL, LNK READNXTRANDOM from the following routines:

   F:MERGE
   DELETE

   Upon entry P1 = sequence number of record to be read.

3. Exit:
   R1 = sequence number of record actually read.
   CC1 = 0 if record existed.
   CC1 = 1 otherwise.
   Return is to calling routine via B O, LNK.

4. Operation:
   This subroutine uses READRANDOM to issue read.
   If read was successful it sets R1 = sequence number,
   CC1 = 0,
   and returns.
   Otherwise it sets CC1 = 1 and returns.

8.4.38 READRANDOM

1. Purpose:
   to clear buffer and read one random record.

2. Entry:
   LNK is the linkage register. This subroutine is entered via BAL, LNK READRANDOM from READNXTRANDOM.
   Upon entry P1 = sequence number of record to be read.

3. Exit:
   CC1 = 0 if record exists.
   CC1 = 1 otherwise.
   Return is to calling routine via B O, LNK if read was successful.
   Error returns: it branches to BADIO if key existed but read unsuccessful;
   it returns with CC1 = 1 otherwise (B O, LNK).

4. Operation:
   It uses: BLANKBUF to clear buffer and SETKEY to set up key for the read.
   It issues read.
   It uses SETLASTKEY to save key from this read.

8.4.39 READSEQUEN

1. Purpose:
   to clear buffer and read next sequential record.

2. Entry:
   LNK is the linkage register. This subroutine is entered via BAL, LNK READSEQUEN from the following routines:

   F:COPY
   F:MERGE
   F:LIST
   F:COMPRESS
   DELETE
   READNXTRANDOM

3. Exit:
   R1 = sequence number of record read in (zero if file not keyed).
   Normal return is to calling routine via B O,LNK.
   Error return is a branch to BADIO if read was unsuccessful for any reason other than EOF.

4. Operation:
   This subroutine uses BLANKBUF to clear buffer; it reads record; it uses SETLASTKEY to save key from the record read.
   If EOF hit it gets and prints the key of last record: '--EOF HIT AFTER YYYY.YYY'. Otherwise it sets R1 and returns.

8.4.40 READSEQUEN2

1. Purpose:
   to read a record sequentially from M:EI and, if compressed, to check sequencing and checksum of the record.

2. Entry:
   LNK is the linkage register. This subroutine is used by F:BUILD and RECONSTRUCT$LINE via BAL,LNK READSEQUEN2.

3. Exit:
   Normal exit is B O,LNK with the following possible condition code settings:

   CC1 = 0 if record not compressed, in which case R1 = byte count of record,
   CC1 = 1 if record is compressed,
   CC2 = 1 if end-of-file encountered,
   CC3 = 1 if EOD encountered.

   Error exit is via M:ERR to the monitor if error found in compressed sequencing or checksum.

4. Operation:
   The record is read into CIBUF. Byte count of record is placed in R1. If end-of-file or EOD is encountered, exit is taken after setting the appropriate condition codes. If the record is not compressed, i.e., byte 0 is not X'3F' or X'1F', exit is taken with CC1 = 0. If the record is compressed and one of the following conditions is met, the error message, 'ERROR IN CI. CONTROL BYTES = XXXXXXXX' is printed and M:ERR exit is taken to the monitor:

   1. sequence number in the record does not equal 1 plus the sequence number in CISEQ;
   2. checksum computed by adding together all bytes in the record does not equal checksum in the record itself.

   Otherwise control words are initialized as follows:
   CIBTOTAL = usable bit count from the record;
   CIBUSED = 32, total bits used from CI record;
   CIBLEFT = 32, number of bits left in current word in CIBUF;
   CIWORD = address of first (i.e., current) word of test in record.

8.4.41 READTELETYPE2

1. Purpose:
   to read and print one input command from the "C" device.

2. Entry:
   LNK is the linkage register. This subroutine is entered via BAL,LNK READTELETYPE2 from MASTERPARSER.

3. Exit:
   B O,LNK is normal exit. End-of-date exit is to F:END.

4. Operation:
   The next input command is read from the "C" device into TTYIMG. If end-of-data is reached, exit is to F:END. The command is printed via M:LO. The command is scanned to count the number of characters read through the last non-blank. This count plus 1 resides in R1 upon exit from the subroutine. A (CR) character X'15', is appended to the text.

8.4.42 RECONSTRUCT$LINE

1. Purpose:
   to reconstruct a symbolic record in CARDIMG using compressed input in CIBUF.

2. Entry:
   LNK is the linkage register. This subroutine is entered via BAL, LNK RECONSTRUCT$LINE from F:BUILD. Upon entry the compressed data to be converted is in CIBUF.

3. Exit:
   B O, LNK with condition code setting as follows:
   CC1 = 1 if end of compressed file reached;
   CC1 = 0 otherwise.

4. Operation:
   Registers are initialized as follows:
   X3 = 0, current byte in output buffer;
   T2 = contents of CIBUSED, total bits used from CIBUF;
   X2 = contents of CIBLEFT, total bits left in current word in CIBUF.

   The output buffer, CARDIMG, is initialized to blanks via BLANKBUF. Control bytes and their corresponding characters if any are obtained from CIBUF and their EBCDIC equivalents are edited into CARDIMG one by one until either an end of line or end-of-file item is found, after which exit is taken with condition code appropriately set. Subroutine RECL60 is used to perform the actual extraction of each character from CIBUF. One character is extracted per entry to RECL60. Action taken for compressed items is as follows:
   1. Items 0 and 1: go to next item.
   2. Item 2 (EOL): save T2 contents in CIBUSED; save X2 contents in CIBLEFT; exit with CC1 = 0.
   3. Item 3 (EOF): save T2 and X2 as in step 2 and exit with CC1=1.
   4. Item 4: use RECL60 to extract next 8 bits from CIBUF; store resultant byte into CARDIMG; increment pointer in output buffer; advance to next item.
   5. Item 5: use RECL60 to extract next 6 bits from CIBUF which is a count of blanks; store blanks into CARDIMG until count is satisfied; increment pointer in output buffer; advance to next item.
   6. Item 6: follow procedure in step 5 adding an additional 65 blanks into CARDIMG.

8.4.43 RECL60

1. Purpose:
   to extract the specified number of bits from CIBUF which comprise one compressed "item".

2. Entry:
   LNK is the linkage register. This subroutine is entered via BAL, LNK RECL60 from RECONSTRUCT$LINE. Upon entry:
   P2 = number of bits to extract;
   T2 = and X2 are as set in RECONSTRUCT$LINE.

3. Exit:
   Normal exit is B O, LNK with requested item in X4. Error exit is to monitor via M:ERR if record is read which is not compressed.

4. Operation:
   If fewer bits are requested then remain unused in CIBUF, an additional record is read into CIBUF using READSEQUEN2. If the record is not compressed, the message '-INCOMPLETE CI.' is printed via TYPEMSG and return is made to the monitor. The current word is moved into register X1 from CIBUF (addressed indirectly via CIWORD). If all bits required are contained in it, the requested bits are shifted if necessary so that they are right justified in X4. The shift is in registers X4 (R) and X1 (RU1) so that after it is done, the contents of X1 are saved in the current word in CIBUF. Exit is taken. If all bits required are not contained in the current word in CIBUF, the bits which are in the current word are shifted into X4, the address in CIWORD is incremented by 1, and the remaining required bits are obtained from the next word from CIBUF.

Register X2 is adjusted to contain the count of unused bits in the current word in CIBUF the unused portion of the current word is stored back into CIBUF, and exit is taken.

8.4.44 REPSEQ

1. Purpose:
   to replicate single sequence number in PARAMBUF + 1.
2. Entry:
   LNK is the linkage register. This subroutine is entered via BAL,LNK REPSEQ.
3. Exit:
   B *LNK
4. Operation:
   It picks up sequence number in PARAMBUF, stores it in PARAMBUF + 1, and adds 1 to PRMBUFSZ.

8.4.45 SETEOD

1. Purpose:
   to scan active card image to locate the right-most non-blank character.
2. Entry:
   LNK is the linkage register. This subroutine is used by a number of routines such as MASTEREXECUTIVE, F:BUILD, and other F: routines. It is called using BAL,LNK SETEOD.
3. Exit:
   EODCLMN contains column of last non-blank character or-1 if all blanks. RECSIZE contains a byte count of zero if all blanks. Return is to calling routine using B O,LNK.
4. Operation:
   This subroutine scans record image from right looking for all blank words (up to word 0). If no non-blanks are found, it sets flag to check word zero byte by byte. Otherwise it sets flag to indicate byte by byte checking in the word where a non-blank character was found.

8.4.46 SETKEY

1. Purpose:
   appends key length of 3 and stores key in KBUF.
2. Entry:
   LNK is the linkage register. This subroutine is entered via BAL,LNK SETKEY from the following routines:
   READRANDOM
   WRITE2
   WRITENEWRANDOM
   WRITERANDOM
   Upon entry P1 = sequence number to put in key for read or write.
3. Exit:
   Return is to calling routine via B O,LNK.
4. Operation:
   It stores input sequence number with appended key length in KBUF.

8.4.47 SETLASTKEY

1. Purpose:
   to store key of last record read in LASTKEY and to store record size in RECSIZE.
2. Entry:
   LNK is the linkage register. This subroutine is entered via BAL,LNK SETLASTKEY from WRITERANDOM and READSEQUEN.
3. Exit:
   It returns to calling routine via B O,LNK with LASTKEY and RECSIZE set.
4. Operation:
   It sets LASTKEY and RECSIZE, removes carriage return, if any, and uses SETEOD to append carriage return if CR ON.

8.4.48 TYPECERR

1. Purpose:
   to type command error message (TYPECERR) or parameter error message (TYPEPERR).
2. Entry:
   LNK is the linkage register. This subroutine is entered via BAL,LNK TYPECERR
   or BAL,LNK TYPEPERR from many routines.
   The word following BAL contains address of message to be printed.
3. Exit:
   B 1,LNK
4. Operation:
   If maximum error messages allowed have been printed it returns.
   It sets command or parameter number to agree with its place in the command: e.g. '--P2----'.
   It types message via CAL3,1 0 followed by carriage return and line feed.

8.4.49 TYPEMSG

1. Purpose:
   to print a message.
2. Entry:
   LNK is the linkage register. This is a service routine used by many routine Calling sequence is:
   ```
   BAL,LNK  TYPEMSG
   DATA     address of TEXTC message
   ```
3. Exit:
   B 1,LNK
4. Operation:
   The TEXTC string is moved into buffer area LISTIMG. The message is then printed via M:LO using M:WRITE.

8.4.50 WRITECO

1. Purpose:
   to append byte count and checksum to a compressed record and to write the record via M:EO.

2. Entry:
LNK is the linkage register. This subroutine is entered via BAL,LNK WRITECO from COMPRESSLINE. Upon entry T2 = total bit count of record in COBUF.

3. Exit:
B O,LNK

4. Operation:
   a. The total bit count in T2 is incremented by 8, converted to bytes and stored in byte 3 of word 1 of COBUF. The sequence number in byte 1 of word 1 of COBUF is incremented by 1.
   b. If the resultant byte count in T2 is 4 or less, the routine resumes processing at step c. Otherwise, all bytes of the record are added together to form the checksum which is placed in byte 2 of word 1 of COBUF. The record in COBUF is written via M:EO.
   c. COWORD is initialized to point to COBUF + 1. Byte 1 of word 1 of COBUF is set to zero as are words 1-29. T2 (bits used in current word) and X2 (total bits used) are set to 32.

8.4.51 WRITENEWRANDOM

1. Purpose:
to write a record with NEWKEY

2. Entry:
LNK is the linkage register. This subroutine is entered via BAL,LNK WRITENEWRANDOM from R:RENUMBER. Upon entry P1 = sequence number of record to be written.

3. Exit:
Error exit is branch to BADIO if error occurred for any reason other than that record already existed.
Normal return is B O,LNK to calling routine (CC1 = 0 to show record written).
If the record already existed, it sets CC1 = 1 before returning to calling routine.

4. Operation:
It calls SETKEY to put sequence number in KBUF.
It calls PUTCR to insure that there is a carriage return in the record.
It writes record with NEWKEY: if successful it sets CC1 = 0;
if not, it sets CC1 = 1.

8.4.52 WRITERANDOM

1. Purpose:
to write a new random record (whether or not it already exists).

2. Entry:
LNK is the linkage register. This subroutine is entered via BAL,LNK WRITERANDOM from F:BUILD.
Upon entry P1 = sequence number of record to be written.

3. Exit:
Normal return is B O,LNK to calling routine.

4. Operation:
This subroutine calls SETKEY to put sequence number in KBUF; it calls PUTCR to insure that CR is present if mode is CR ON and writes record with ONEWKEY.

8.4.53 WRITE2

1. Purpose:
   to write a record in copy file.

2. Entry:
   LNK is the linkage register. This subroutine is entered via BAL, LNK WRITE2 from F:COPY and F:MERGE. Upon entry P1 = sequence number of record to be written.

3. Exit:
   Error exit is a branch to BADIO following write error for any reason other than that the record already existed. Normal return is to calling routine via B O, LNK (CC1 = 0). If record already existed it sets CC1 1 before returning to calling routine.

4. Operation:
   It uses SETKEY to put sequence number in KBUF.
   It uses PUTCR to insure CR is in record (provided CR ON is the mode).
   It writes record with NEWKEY: if successful it sets CC1 = 0;
   otherwise it sets CC1 = 1.

8.4.54 WRITE$EO

1. Purpose:
   to write on EBCDIC record via M:EO.

2. Entry:
   LNK is the linkage register. This subroutine is entered via BAL, LNK WRITE$EO from F:WRITE. Upon entry the record to be written resides in CARDIMG.

3. Exit:
   B O, LNK

4. Operation:
   The record is written using M:WRITE via M:LO.

Table 8-1. Command Description Table (CDT)

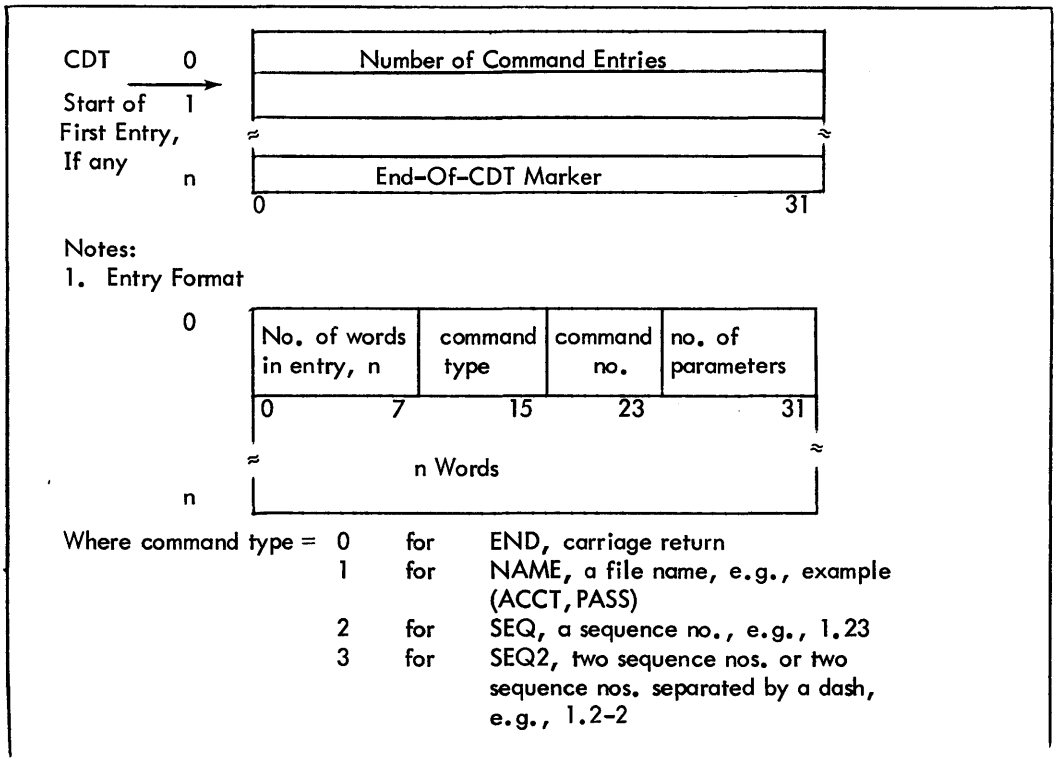


Table 8-1. Command Description Table (Cont.)

4	for	INTG, a numeric string whose value is less than 1000; e.g., 123
6	for	ALPH, a character string not enclosed in slashes; e.g., BUILD
7	for	COM, comma
9	for	LPAR, left parenthesis
10	for	RPAR, right parenthesis
11	for	PERIOD
12	for	BLANK

2. CDT + 100 = CDTADR, address of current command in CDT
3. PARAMPSN = next available slot in CDT
4. PRMBUFSZ = no. of words in PARAMBUF to be added
5. CHARPSN = no. of next character to scan
6. End-of-CDT Marker = X'00000100'
7. Command no. is described in Table 8-2.

Table 8-2. Command Table

COMMAND #	COMMAND	CDT BUILDER	CDT EXECUTER
1	WRITE	PARSE:WRITE	F:WRITE
2	BUILD	PARSE:BUILD	F:BUILD
3	COPY	PARSE:COPY	F:COPY
4	DELETE	PARSE:DELETE	F:DELETE
5	LIST	PARSE:LIST	F:LIST
6	END	PARSE:END	F:END
7	COMPRESS	PARSE:COMPRESS	F:COMPRESS
8	MERGE	PARSE:MERGE	F:MERGE

9.0 FAST SAVE PROCESSOR

9.1 FUNCTIONAL OVERVIEW

The FAST SAVE Processor is a utility designed to provide the capability of saving files at or near tape speed. It will take approximately 5 minutes for each 2400 ft. reel of tape using FAST SAVE with 150 ips mag tape units. A time saving of 80% or more is now achieved over any other file saving procedures under BPM/BTM.

9.2 INTERFACE

FAST SAVE is able to operate under any release version of BPM/BTM since and including F00, providing it has these minimal hardware requirements.

1. At lease one mag tape unit
2. Line Printer
3. Cardreader
4. One or more Rads and or Packs
5. Console teletype

9.3 OPERATIONAL OVERVIEW

The FAST SAVE program must run in the :SYS account as a privileged processor in master mode. This is a necessity in order to gain access to the account directory, to open all Public files, and the BPM/BTM queueing routines. A load requirement of a stack size of X'80' is necessary. Other than the above, no other requirements are necessary.

All tape record blocks written to tape will be 512 words or less. The label sentinel on tape will be PRG1 thru PRG9, as the present BPM/BTM FPURGE processor. Also, the present FPURGE processor has no difficulty in reading any tapes created by FAST SAVE.

Sample Loading Procedure

```
!JOB  :SYS, ME, F
!LOAD  (BI), (LMN,FSAVE),(MAP),(PERM),(TSS,80)
```


```
!EOD
!FIN
```

An assembly switch (start) is set at the beginning of the program, primarily for MBS vs. STB. In the event that the hardware does not have MBS simulation, this switch would be set to 0.

```
Sigma7    EQU    1        MBS Mode
Sigma7    EQU    0        STB Mode
```

FAST SAVE is released in the MBS Mode.

9.3.1 COMMAND SUMMARY

COMMAND	MEANING
+LOG	No output reel will be used. If +LOG is used, you cannot use +DUMP option. In card column 9, you have the option to use STATS. (See NOTE below).
+BLOCKS	Lists output buffers as they are built.
+DEBUG	Enables snaps and dumps of errors found in file management tables.
+LIST	This is the only command that will produce a listing log on M:LL.
+INDEX	List file index sectors.
+FIT	List file information table.
+DIRECTORY	List file and account directories.

The above commands, if used, must always precede the following commands.

+DUMP	Defines tape output to be produced. If +DUMP is used, cannot use +LOG. In Card Column 9 you have the option to use stats (see stats below).
+HOUR	Save all files created since HOUR defined in Card Column 9 of this card. '0900' defines 9:00 a.m., and '1800' defines 6:00 p.m. .
+DAY	Save all files created on and since the Month and Day defined in Card Column 9 of this card. '0831' defines August 31, and '0202' defines February 2.
+START	To save all files starting at the account number defined on the data cards that follow.
+SKIP	To save all files except skipping the accounts and files on the data cards that follow until +END command is read.
+SELECT	Save only the accounts and files specified on the data cards that follow until +END command is read.
+END	Last command to be read. If using data card, +END would terminate the data.

NOTE: The STATS option in card column 9 on the +LOG and +DUMP options will create a statistical file name called 'DISK POOL' in the :SYS account. Five words are reserved for each account. After termination of FAST SAVE, the file 'DISK POOL' can be accessed for accounting purposes, to extract, the following information:

WORD	CONTENTS
0 and 1	ACCOUNT NAME
2	Number of files in account
3	Number of granules in account
4	Number of bytes in account.

Each record can be accessed by key or sequentially.

9.3.2 Data Cards

ACCOUNT#	File Name
CC1	CC14
:SYS	COBOL
:SYSGEN	RDF
:SYSGEN	WRTF
F5608312	

The absence of a file name specifies 'ALL' files.

ALL DATA CARDS must be in sorted order before being read by via SI' Device. FAST SAVE does not link to SORT or search backwards through the account directory or file directory. Therefore the burden is on the user to have the data cards in sorted order before FAST SAVE is run.

1. Job stream to save selective files to tape

   A.
```
!JOB  :SYS, ME, F
!POOL (IPOOL,0),(FPOOL,0)
!FSAVE
+LIST
+DUMP
+SELECT
```

CC1	CC14
:SYSGEN	LOCCTA
	LOCCTB
	LOCCTC
32301	A
	B
	C

```
+END
```

   B. Accounts and files must be in sorted order.

2. Job stream to save all files in entire system, and specifying a series of names to skip.

9.3.3 Error Message Summary

MESSAGE	MEANING
'BPM stack size too small'	Load module was created with 'TSS' of less than X'80' words.
'Not enough core to run less than 5 pages available'	Due to a paging buffer method of operating, less then 5 pages of dynamic storage is insufficient.
'Cannot read 1st Account Directory Sector'	ACNCFU contains invalid disc address- this is a catastrophic failure.
'Link failure in Account Directory'	Cannot link forward due to invalid disc address.
'Link failure in File Directory'	Cannot process any more of this account- continue with next account in directory.
'Link failure in Index Chain'	Cannot process this file any further. Output tape is closed out for the current file.
'Fit disc address error in file directory key'	Unable to process this file.
'Control card error'	Control cards are order dependent, and they are in that order.
'File directory address error in account key'	The disc address for the file directory is invalid - this account is skipped.
'File name does not, match file directory key'	The fit read does not match the current file directory name.
'New reel started'	Self-explanatory
'Irrecoverable tape write error'	Will request the next reel and continue processing.
'Error in file information table'	An entry required to process this file is missing from the 'FIT'
'Error in master index'	An invalid entry was found in the index sector.
'Partial file'	Saved as much as possible
'Processing terminated'	Everything completed

A. !JOB :SYS, ME,F
!POOL (IPOOL,0),(FPOOL,0)
!FSAVE
+LIST
+DUMP
+SKIP

CC1	CC14
:SYSGEN	
F5608312	
F5608318	
G2708300	FILEA
G2708300	FILEB
+END	

B. The absence of a file name in Card Column 14 specifies the entire account is to be SKIPPED/SELECTED.

3. Job stream to list all files in the system without saving the files on tape.

```
!JOB :SYS, ME, F
!FSAVE
+LIST
+LOG
+END
```

4. Job stream to list a file directory

```
!JOB :SYS, ME, F
!FSAVE
+LIST
+DIRECTORY
+LOG
+SELECT
:SYS
+END
```

5. Job stream to list a file information table

```
!JOB :SYS, ME, F
!FSAVE
+LIST
+FIT
+LOG
+SELECT
:SYS        METASYM
+END
```

6. Job stream to save files created since 5:00 a.m., February 25, 1972

```
!JOB  :SYS, ME, F
!POOL (IPOOL,0),(FPOOL,0)
!FSAVE
+LIST
+DUMP
```

CC1	CC9
+DAY	0225
+HOUR	0500

```
+END
```

7. Job stream to save all files in the system, starting at a given account.

```
!JOB  :SYS, ME, F
!POOL (IPOOL,0),(FPOOL,0)
!FSAVE
+LIST
+DUMP
+START
F5608312
+END
```

8. Job stream to save a given file on tape; list its account directory sector, its file directory sector, and its file information table; list all its index sectors; and list out the tape buffer as they are written to tape.

```
!JOB  :SYS, ME,F
!POOL (IPOOL,0),(FPOOL,0)
!FSAVE
+LIST
+FIT
+DIRECTORY
+INDEX
+BLOCK
+DUMP
+SELECT
```

CC1	CC14
:SYS	METASYM

```
+END
```

9.3.4 Error SNAPS

SNAP TITLE	MEANING
'ABNIO'	A TYC other than normal (i.e., '01' was returned).
'DCTERR'	The first tape open did not return a valid 'DCTX' to the M:EO DCB
'OPNFAIL'	Any BPM I/O error with the M:EO DCB. This probably will never occur since all I/O is performed thru 'NEWQ'.
'ACLINK'	Link failure in account directory
'ACNCFU'	'ACNCFU' contain an invalid disc address
'FDLINK'	File directory linkage failure
'MIXLINK'	Link failure in index chain

9.3.5 Read Ahead Logic

To consider the read ahead logic, each segment of the total file system is considered to be a separate level.

LEVEL	AREA INVOLVED
1	Account directory sectors
2	File directory sectors
3	File information tables
4	Master index sectors
5	Data granules

Read ahead for account/file directories is simulated double-buffering, that is, if the program has just finished using the last key in the current sector, the next sector is read into the same buffer. The same logic applies to FIT'S.

Master Indicies are read in order. Each FLINK is read as soon as the current request is satisfied. A maximum of 4 index sectors are read, and as soon as one is released, a new read is attempted.

All data disc addresses from each index sector (as end action occurs) are pushed into a data address stack. Therefore all data reading is queued to as many reads as possible continually (depending on pages available) and everytime a data page is released, read further along.

Tape buffer pages are allocated as necessary, and are released as tape write end action occurs.

9.4 MODULE ANALYSIS

9.4.1 INITIATE

1. Purpose:
   This is the first module executed in the FAST SAVE processor. It saves the processor stack pointer, insures that the stack is large enough, enters the MASTER mode, saves the address of NEWQ, and gets pages from the Operating System tor index buffers.

2. Entry:
   entered in slave mode as first executable statement in FAST SAVE.

3. Exit:
   ERROR EXIT

   B SPECINIT if not enough pages of core available.

   NORMAL EXIT

   B INITIATE1

4. Operation:
   Save (RO) in STKPNTR; RO contains the address of the Stack Pointer doubleword in the processor Task Control Block. Tests the second word of the Stack Pointer doubleword to determine if the stack is large enough. It must be at least 128 words long. If it is not large enough type out '***BPM STACK SIZE TOO SMALL' and do an M:XXX to exit the program.

   Otherwise, execute an M:SYS CAL to enter the master mode. The address of the monitor's I/O Queing routine - NEWQ - is routined in RIO upon return from an M:SYS CAL. This address is then stored in data location NEWQ within FAST SAVE for later use.

   An M:GP CAL is executed to get 4 pages for index sector buffers. The number of pages received is saved in page total. If 4 pages are not readily available, the processor exits to SPECINIT.

   Otherwise, the number of pages requested is saved in INDPAGES and the program exits to INITIATE1.

9.4.2 INITIATE1

1. Purpose:
   This module is responsible for computing and saving the word address of each index buffer in table IBUF.

2. Entry:
   B INITIATE1
   (R1) = number of index buffers
   (R9) = address of first page received from get page CAL.

3. Exit:
   B INITIATE4

4. Operation:
   Register 1 is used as an index into table IBUF for saving the Word address of each page recieved 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 256 to (R9), and this address is saved. This process is repeated until all 8 index buffer addresses have been computed and saved in the IBUF table.

   The routine then exits to INITIATE4.

9.4.3 SPECINIT

1. 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.

2. Entry:
   B SPECINIT
   PAGETOTAL = Number of pages received from Get Pages CAL.

3. Exit:
   ERROR EXIT
   B NOTENUFF if at least 4 pages are not available.
   NORMAL EXIT
   B INITIATE5

4. 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 worda 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.

9.4.4 INITIATE4

1. Purpose:
   This module is responsible for requesting pages from the Operating System for tape buffer and Data buffers and saving their addresses.

2. Entry:
   B INITIATE4
   GETPAGES = Max Page request FPT (M:GP procedure).

3. Exit:
   ERROR EXIT
   B SPECINIT if not enough pages are available.
   NORMAL EXIT
   B INITIATE5

4. 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 VURPOS. 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 SPECINIT.

Otherwise, the address of each page is stored into table TBUF, and the number of tape buffers, TAPAGES, is 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.

These numbers are converted to decimal by a BAL,R15 to HEXTODEC and a message is typed out to inform the operator how many buffers are available.

'NN TOTAL TAPE BUFFERS'
'NN TOTAL DATA BUFFERS'

where NN equals the number available.

The program then exits to INITIATE5.

9.4.5 INITIATE5

1. Purpose:
   This module is responsible for getting the date and time from the monitor.
2. Entry:
   B INITIATE5
3. Exit:
   B RDCARD
3. Operation:
   An M:TIME CAL is executed to store the data and time from the monitor into words 3–6 of DATBUF. The background lower limit is computed from the first address of the program and stored in word 8 of DATBUF.

   The program then reads one card through the M:C device to read the !FAST SAVE control card. It then exits to RDCARD.

9.4.6 RDCARD

1. Purpose:
   This module is responsible for reading control cards, interpreting control cards, and setting the appropriate flags. It calls on moduleRCD 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.
2. Entry:
   B RDCARD
3. Exit:
   B NOIPRI
4. 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 according to the control card option, ejects a page on the M:LL device, and exits to NOPRI.

9.4.7 RCD

1. 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.

2. Entry:
   BAL,R14 RCD

2. Exit:
   B *R14

4. 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.

9.4.8 INTER

1. 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.

2. Entry:
   BAL,R15 INTER
   INBUF contains the control card to be interpreted.

3. 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.

4. 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 FAST SAVE aborts.

9.4.9 NOIPRI

1. Purpose:
   This module opens the stat file if that option was specified. The address of DCT1 and HGP are determined.

2. Entry:
   B to NOIPRI

3. Exit:
   B READFAIL 2 if unable to read Account Directory or
   if bad data is in Account Directory

B ENDUP if abnormal or error return opening the STATS file
NORMAL EXIT

B NACN

4. Operation:
If the STATS flag is non-zero the module opens the STATS file as follows: File name = DISKPOOL, Account = :SYS, KEYED, OUT mode, READ-ALL, WRITE-NONE. An abnormal or error return on the open causes a branch unconditionally to module ENDUP after typing a message "FILE MANAGEMENT ERROR FROM STAT FILE".

The location of the monitor's DCT1 table is determined by accessing the word pointed to by location X'4E'. This address is stored in location DCT1 in the program. The size of the DCT tables is determined by byte 0 of that word. It is stored into DCTSIZ.

The address of the HGP is determined by adding 10 to the location pointed by 'X4E' and accessing that location. The second word of the HGP is then accessed to obtain the DCT index of the System device.

The ACNCFU is then read from the System device into ACNCFU table by a BAL,R15 to DISCIO. If a read failure occurs the program branches to READFAIL.

If the DIRLISTSW flag is non-zero, the ACNCFU is listed on the M:LL device by a BAL,R15 to PLIST.

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 into 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.

9.4.10 NACN

1. Purpose:
This module fetches the next account directory entry from ACBUF, checks if 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.

2. Entry:
   B NACN
   ACBUF contains account directory sector
   NEXACN points to next entry to be processed

3. 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.

4. 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'0B404040' 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 an 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 NOTBL 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. It is 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.

9.4.11 NOTLB

1. Purpose:
   This routine prints the current account number on the M:LL device if specified, reads in the next account directory sector is 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.

2. Entry:
   B NOTLB
   ACN#DISP is displacement to Account Directory entry to be processed.
   ACBUF contains current account directory sector.

3. Exit:
   ERROR EXIT

B ADERROR	if file directory fiest sector address in 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

4. Operation:
   Move the account number from ACBUF to ACN#CURNT and if LIST flag is on 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 sector.

   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 its 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 first word of FDBUF and store in FDBLINK. Get the FLINK from 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 RAADFAIL3. Otherwise, store the current sector address in LASTFD and branch to GETFILE.

9.4.12 GETFILE

1. Purpose:
   This module gets the next file directory entry, picks up the disc address of the FIT, and reads the FIT into FITBUF.

2. 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.

3. Exit:
   ERROR EXIT

B FDERR	if FIT address is not valid or if read error occurs when reading FIT.

   NORMAL EXIT
   B FILECHKS

4. 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 its 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.

9.4.13 FILECHKS

1. 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 NOFCK. If this is the last file in the current file directory sector, it queues up to read in the next file directory sector.

2. Entry:
B FILECHKS
FITBUF contains current fit to be processed
CURFILE points to current file directory entry in FDBUF.

3. 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

NROMAL EXIT

B NOFCK	to process file
B GETFILE	if this file is not to be processed.

4. 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 NOFCK. If ALL is not set, compare the FIT file name with the file name contained in Column 14 then, CAL,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 NOFCK 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 its on, BAL,R14 to LISTFIT to list the FIT. Then, branch to NOFCK.

9.4.14 NOFCK

1. 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.

2. Entry:
   B NOFCK
   FITBUF contains FIT of file to be processed.

3. 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

4. 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 SYNONOMOUS file transfer the synonomous 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 CKT10 (NOTE* CKT10 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.

9.4.15 GETMIX

1 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.

2. Entry:
B GETMIX
FDA contains address of current MIX sector.

3. Exit:
NORMAL EXIT
B FILEDONE after entire file has been processed.

4. 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 it 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 GETKEY1 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 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.

9.4.16 QUEREC

1. Purpose:
   This routine calls MT10 to write out a tape buffer
2. Entry:
   BAL,R15 QUEREC
   CURBUF contains address of current tape buffer.
   KEYDISP has number of bytes in the buffer.
3. Exit:
   NORMAL EXIT
   B MT10
   R15 has address to branch to after tape record has been queued.
4. 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 MT10 with R15 unchanged. MT10 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 MT10. MT10 will then branch to LISTOUTBUF which will return to the original address in R15.

9.4.17 GETKEY1

1. Purpose:
   This routine sets the previous block size into a tape buffer and sets the displacement into the buffer to four.
2. Entry:
   BAL,R15 GETKEY1
   RBS contains previous block size from last record.
   CURBUF contains address of buffer to store PBS into.
3. Exit:
   B *15
4. Operation:
   The program gets the previous tape block size from BBS 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.

9.4.18 MOVEKEY

1. Purpose:
   This module moves the current key to the current blocking buffer.
2. 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.

3. Exit:
   NORMAL EXIT
   B *15

4. Operation:
   The routines increments 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 CURRBUF at the appropriate displacements. It saves the new MIX displacementom 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.

9.4.19 MOVE

1. Purpose:
   This routine moves a data record to a tape blocking buffer.

2. 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.

3. 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.

4. 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. It all bytes can be moved successfully, it updates the granule accounting table and exits skipping.

9.4.20 GETKEY

1. Purpose:
   This module gets the next key from the MASTER INDEX of the file currently being processed.

2. Entry:
   BAL,R15 GETKEY

3. 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

4. Operation:
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 incore, 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.

9.4.21 SCHEDULE

1. Purpose:
This module releases data buffers when they have been processed.

2. Entry:
BAL,R7 SCHEDULE

3. Exit:
B *R7

4. 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 CKT10.

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 in 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.

9.4.22 QUEMIX

1. Purpose:
This routine queues up a read for the next MIX sector, with END ACTION address set to MIXENAC.

2. Entry:
BAL,R15 QUEMIX

3. Exit:
ERROR EXIT
B READFAIL5 if FLINK address is not valid
NORMAL EXIT
B *R15

4. Operation:
The routine requests an index buffer. If none is available it exits.

If th 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 MEXANAC.

9.4.23 RANFILE

1. Purpose:
Writes tape records for RANDOM files.

2. Entry:
B RANFILE

3. 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 CKT10	when entire file has been processed or if no tape is being written.

4. 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 CKT10.

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 CKT10. 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 CKT10.

9.4.24 GETTBUF

1. Purpose:
This routine gets a tape output buffer.

2. Entry:
BAL,R15 GETTBUF

3. Exit:

B *R15	condition codes set non-zero if buffer found. R1 is index into TBUF table for available entry.

4. 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.

9.4.25 GETIBUF

1. Purpose:
   Gets a disc input buffer for MIX.
2. Entry:
   BAL,R15 GETIBUF
3. Exit:
   B *R15 — condition codes set non-zero if buffer is found. R1 is index into IBUF table for available entry.
4. 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.

9.4.26 GETFOUR

1. Purpose:
   This routine performs read ahead for data granules from the disc.
2. Entry:
   BAL,R11 GETFOUR
   DSTACK contains addresses of granules to be read.
3. Exit:
   ERROR EXIT
   B DATAERR — if a bad disc address is encountered
   NORMAL EXIT
   B *R11

9.4.27 BUILD

1. Purpose:
   This routine pushes the address of all data granules specified in the Just Read index sector into DSTACK.
2. Entry:
   BAL,R11 BUILD
   R7 is index into IBUF to current mix sector.
3. Exit:
   NORMAL EXIT
   B *R11

9.4.28 STKSIZR

1. Purpose:
   This routine types a message "BPM STACK SIZE TOO SMALL" and does an M:XXX.
2. Entry:
   B STKSIZR
3. Exit:
   CAL1, 9 3

9.4.29 QSECTOR

1. 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.

2. Entry:
   BAL,R15 QSECTOR
   R8 equals disc address to be read
   R10 = Byte address of buffer

3. Exit:
   B DISCIO2
   R15 points to exit for DISCIO2 routine to original program.

9.4.30 DISCIO2

1. Purpose:
   This routine is the RAD and SIDK Pack handler routine for FAST SAVE. 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.

2. 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 indes (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

3. Exit:
   B *R15

9.4.31 DENAC

1. Purpose:
   This is the DISCIO end action routine; it resets the busy flag RBUSY and saves the TYC information in DSTATUS.

2. Entry:
   BAL,R11 DENAC from IOQ

3. Exit:
   B *R11

9.4.32 FITENAC

1. 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.

2. 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

3. Exit:
   B *R11 returns to IOQ

9.4.33 ENCOM

1. Purpose:

   This is the end action routine specified by routine GETFOUR. The appropriate table entry in RB1 which is pointed to by R14 is set to zero to indicate that the address is now in core.

2. Entry:
   BAL,R11 ENCOM from IOQ
   R12 is TYC information
   R14 is index to flag in RB1
   R11 is return

3. Exit:
   B *R11 returns to IOQ

9.4.34 MIXENAC

1. Purpose:
   This is the end action routine for mix sector reads. It decrements the outstanding I/O count in DOPCNT, saves the TYC from R12 in DSTATUS, sets the new sector flag NEWSEC to -1. It clears the I/O pending bit in INDEXDA pointed to by R14. It then gets the next FLINK, and branches to BUILD, which will exit back to IOQ on R11.

2. Entry:
   BAL,R11 MIXENAC
   R14 = index into INDEXDA
   R12 = TYC status
   R11 = return register

3. Exit:
   B BUILD
   NXTFLNK contains next FLINK to be read.
   BUILDLNK contains return address to IOQ.

9.4.35 MTIO

1. Purpose:
   This is the routine which queues up tape I/O for FAST SAVE. It does so by branching to the monitor's NEWQ routine.

   1. 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

   2. Exit:
      B *R15

9.4.36 SIMULATE

1. Purpose:
   This routine simulates normal I/O end action for the tape requests if no tape is being written, DUMP = 0. This bypasses NEWQ if DUMP is zero. It sets R14 equal to R1. Sets R12 to normal TYC, and does a BAL,R11 *RO to WTENAC.

2. Entry:
   B SIMULATE
   RO Equals address of end action routine to branch to

3. Exit:
   B MTIOX

9.4.37 WTENAC

1. 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 reror, it types a message to the operator and switches the output reels. Otherwise, it exits to the address in R11.

2. Entry:
   BAL,R11 WTENAC from IOQ
   R14 is index into TBUF table
   R12 is TYC
   R11 is return register

3. Exit:
   B *R11 return to IOQ

9.4.38 SENTENAC

1. 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.

2. Entry:
   BAL,R11 SENTENAC from IOQ
   R12 is TYC

3. Exit:
   B *R11

9.4.39 NEWREEL

1. 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.

2. Entry:
   BAL, SR3 NEWREEL

3. Exit:
   ERROR EXIT
   B DCTXERR if bad DCT index returned from Open.
   NORMAL EXIT
   B *SR3

9.4.40 WRTDAT

1. Purpose:
   This routine is called to write a DATE record on reel PRG1.

2. Entry:
   BAL,R7 WRTDAT

3. Exit:
   B *R7

9.4.41 DCTXERR/OPNABN/OPNERR

1. Purpose:
   This routine is an error routine from DCB opens. It types OPNFAIL and does an M:SNAP of the M:EO DCB. It then executes a CAL1,9 3.
2. Entry:
   B DCTXERR
   B OPN ABN
   B OPNERR
3. Exit:
   CAL1,9 3

9.4.42 GOEOR

1. 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, 2 more tape marks, and does a M:CLOSE for that tape reel.
2. Entry:
   BAL,R1 GOEOR
3. Exit:
   B *R1

9.4.43 EOFQ

1. Purpose:
   Thss routine creates and writes a :EOF record onto tape.
2. Entry:
   BAL,R14 EOFQ
3. Exit:
   B WRTMARK (WRTMARK then returns to original routine)

9.4.44 BOFQUE

1. Purpose:
   This routine creates and writes a :BOF record on tape.
2. Entry:
   BAL,R15 BOFQUE
3. Exit:
   B WRTMARK

9.4.45 MOVEI

1. Purpose:
   This subroutine is used to transfer sentinel records into background records to be written to tape.
2. Entry:
   BAL,R14 MOVEI
3. Exit:
   B *R14

9.4.46 WRTMARK

1. Purpose:
   This routine queues up a write tape mark operation.
2. Entry:
   BAL,R15 WRTMARK
3. Exit:
   B MT10+2

9.4.47 NOTENUFF

1. Purpose:
   This routine prints "NOT ENOUGH CORE TO RUN, LESS THAN 4 PAGES AVAILABLE" and then does an M:XXX
2. Entry:
   B NOTENUFF
3. Exit:
   CAL1,9 3

9.4.48 ENDUP

1. 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.
2. Entry:
   B ENDUP
3. Exit:
   CAL1,9 1

9.4.49 ADDONE

1. Purpose:
   This routine is entered when the current account directory sector has been processed. It gets another sector of account directory if possible. Otherwise, it branches to ENDUP if processing of all accounts is completed.
2. Entry:
   B ADDONE
   ACFLINK has FLINK for account directory
3. Exit:
   ERROR EXIT
   B READFAIL2 if FLINK address is not valid
   NORMAL EXIT
   B ENDUP if FLINK ie zero
   B GETAD to queue up another read of account directory
   B GETAD3 after waiting for next sector to be read in.

9.4.50 ADERROR

1. Purpose:

   This routine is entered when an error is found in an account directory key. It types a message, dumps the sector, skips to the next key, and exits.

2. Entry:
   B ADERROR

3. Exit:
   B ADELETE

9.4.51 IORUNDOWN

1. Purpose:
   This routine loops until all DISC and tape operations are completed, then it returns.

9.4.52 FAILURE

1. Purpose:
   This routine is entered if module MOVE checked the granule pointed to by CURRB and it did not match the granule pointed to by current disc address key. It prints "SCHEDULING ERORR***" and branches to MIXSNAP.

2. Entry:
   B FAILURE

3. Exit:
   BAL,RO MIXSNAP

9.4.53 DISPRUNTOTL

1. Purpose:
   This routine displays the final run statistics.

2. Entry:
   BAL,R15 DIS PRUNTOTL

3. Exit:
   B *R15

9.4.54 DTOGRAN

1. 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.

2. Entry:
   BAL,R15 DTOGRAN
   R8 is disc address

3. Exit:
   ERROR EXIT

   B #R15 if address is bad
   NORMAL EXIT
   B *R15 +1 if address is valid

9.4.55 FDERR

1. 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.

2. Entry:
   B FDERR
   B FDLETE

3. Exit:
   B FDLETE

9.4.56 FITSNAP/FITERR

1. 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.

2. Entry:
   B FITSNAP
   B FITERR

3. Exit:
   B FDERR1

9.4.57 LISTFD

1. Purpose:
   This routine lists the current file directory sector.

2. Entry:
   BAL,R14 LISTFD

3. Exit:
   B *R14

9.4.58 LISTFIT

1. Purpose:
   This routine lists the current FIT.

2. Entry:
   BAL,R14 LISTFIT

3. Exit:
   B *R14

9.4.59 LISTOUTBUF

1. Purpose:
   This routine lists the current tape output buffer.

2. Entry:
   BAL,R14 LISTOUTBUF

3. Exit:
   B *R14

9.4.60 LISTMIX

1. Purpose:
   This routine lists the current MIX sector.
2. Entry:
   BAL,R14 LISTMIX
3. Exit:
   B *R14

9.4.61 LISTAD

1. Purpose:
   This routine lists the current account directory sector.
2. Entry:
   BAL,R14 LISTAD
3. Exit:
   B *R14

9.4.62 MIXSNAP/MIXERR

1. 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.
2. Entry:
   B MIXSNAP
   B MIXERR
3. Exit:
   B FDERR1 if no tape is being dumped
   B EOFQ if tape is being written

9.4.63 MIXRATERR

1. Purpose:
   This routine is entered if the mix forward link has not been read in time. It prints an error message and branches to MIXSNAP.
2. Entry:
   B MIXRATERR
3. Exit:
   B MIXSNAP

9.4.64 DATAERR

1. Purpose:
   This routine is entered if a data granule address error has occurred. It types an error message and branches to MIXSNAP.
2. Entry:
   B DATAERR
3. Exit:
   B MIXSNAP

9.4.65 FDDONE

1. Purpose:

   This module is entered when a file directory sector has been completed. If this is not the last link, it reads the next link in.

2. Entry:
   B FDDONE

3. Exit:
   ERROR EXIT
   B READFAIL3 if FLINK address is invalid.
   NORMAL EXIT
   B GETFD1 if next sector already in core.
   B GETFD to read in next sector
   B ENDOFD if FLINK is zero.

9.4.66 ENDOFD

1. 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.

2. Entry:
   B ENDOFD

3. Exit:
   B NACN

9.4.67 CKT10

1. 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 decice. It then branches to GETFILE to process the next file.

2. Entry:
   B CKT10

3. Exit:
   B GETFILE

9.4.68 CODESCAN

1. Purpose:
   This routine scans the FIT for the entry whose code corresponds to that in R2.

2. Entry:
   BAL,R15 CODESCAN
   R2 contains code to search for

3. Exit:
   B *R15 if not found
   B *R15 + 1 if found

9.4.69 MOVENTRY

1. Purpose:
   This routine moves an entry from the FIT to a :BOF record.
2. Entry:
   BAL,R15 MOVENTRY
   RO is word address of entry in FIT
   R4 is address of :BOF record
3. Exit:
   B *R15

9.4.70 HEXTODEC

1. Purpose:
   This routine converts a number from hexadecimal EBCDIC.
2. Entry:
   BAL,R15 HEXTODEC
   R3 is number to be converted
   R1 is address to store result
3. Exit:
   B *R15

9.4.71 FDLETE

This routine branches to FILEDONEO. Keys are not removed from File Directory.

9.4.72 ADELETE

This routine branches to ENDOFD. No key removal is supported for Account Directory.

9.4.73 TACNT/TFILNME

1. Purpose:
   These routine types "ACCOUNT" and an account name, or "FILE" and a FILENAME on the operator's console.
2. Entry:
   BAL,R15 TACNT to type account
   BAL,R15 TFILNME to type file name
3. Exit:
   B *R15

9.4.74 ACCK

1. 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.
2. Entry:
   BAL,R15 ACCK
3. Exit:
   B *R15

9.4.75 READFAIL

1. Purpose:
   Entered because of an ACNCFU disc address error. Causes M:XXX abort.
2. Entry:
   B READFAIL
3. Exit:
   B ENDUP

9.4.76 READFAIL2

1. Purpose:
   Entered because of an account directory LINK failure. Causes M:XXX abort.
2. Entry:
   B READFAIL2
3. Exit:
   B ENDUP

9.4.77 READFAIL3

1. Purpose:
   Entered because of a link failure in the file directory. Causes a branch to ENDOFD to skip to next account.
2. Entry:
   B READFAIL3
3. Exit:
   B ENDOFD

9.4.78 READFAIL5

1. Purpose:
   Entered because of a LINK failure in a MIX. Causes branch to MIXERR1 to skip to next file.
2. Entry:
   B READFAIL5
3. Exit:
   B MIXERR1

9.4.79 LPRINT

1. Purpose:
   This routine prints one line and then clears the buffer.
2. Entry:
   BAL,R15 LPRINT
3. Exit:
   B *R15

9.4.80 PLIST

1. Purpose:
   This routine is used to snap out a buffer on the M:LL device.
2. Entry:
   BAL,R15 PLIST
   R1 contains number of bytes in buffer
   R3 contains beginning buffer address

3. Exit:
   B *R15

9.4.81 BUFSET

1. Purpose:
   This routine is called to move a print line from one buffer to another.
2. Entry:
   BAL,R15 BUFSET
3. Exit:
   B *R15

9.4.82 PRINT

1. Purpose:
   Writes one line of print via a CAL1,1 to WRTPBUF.
2. Entry:
   BAL,R15 PRINT
3. Exit:
   B *R15

9.4.83 SPACE

1. Purpose:
   Writes N lines of blanks on M:LL via CAL1,1 WRTBLNK where number of blanks it supplied by R1.
2. Entry:
   BAL,R15 SPACE
3. Exit:
   B *R15

9.4.84 TYPEIO/TYPEIO2

1. Purpose:
   This routine types a message on the operators console.
2. 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
3. Exit:
   B *R15

9.5 DETAILED FLOWCHART OF FAST SAVE PROCESSOR

Table 9-1. Index to FAST SAVE Flowchart

AREA FUNCTION	PAGE NUMBER
INITIALIZATION	1
ACCOUNT DIRECTORY PROCESSING	2-3
FILE DIRECTORY PROCESSING	3
FILE INFORMATION TABLE	
VALIDITY TESTS AND PROCESSING	4-5
BUILD :BOF RECORD	5-6
FILE INFO TABLE HEAD-AHEAD	7
FILE BLOCKING PROCESS	8-10
FILE BLOCKING SUBROUTINES	11-13
FORCE DATA BUFFER SUBROUTINE	14
SCHEDULE DATA BUFFER SUBROUTINE	15
INDEX SECTOR CONTROL	16
RANDOM FILE PROCESSING	17
BUFFER ALLOCATION	18
DATA READING SUBROUTINE	19
INDEX SECTOR END-ACTION	19-20
RAD/DISC I/O SUBROUTINE	20
RAD/DISC END-ACTION RECEIVERS	21
MAG TAPE I/O SUBROUTINE	22
TAPE SENTINEL CONTROL SUBROUTINE	23
ADDITIONAL TAPE SENTINEL SUBROUTINES	24-25
I/O WAIT SUBROUTINE	25
ACCOUNT DIRECTORY ERROR & END PROCEDURES	26
DISC ADDRESS VALIDITY SUBROUTINE	27
FILE DIRECTORY ERROR SUBROUTINE	27
LISTING SUBROUTINES (BUFFERS, - SECTORS, ETC...)	28
FILE DIRECTORY COMPLETION	29
MASTER INDEX ERROR CONTROL	29
FILE DIRECTORY SECTOR COMPLETED	30
CURRENT FILE COMPLETED	30
MISC. SUBROUTINES	31-32
INTERPRET CONTROL CARD	
LINK FAILURE CONTROL	
PAGING TABLES	

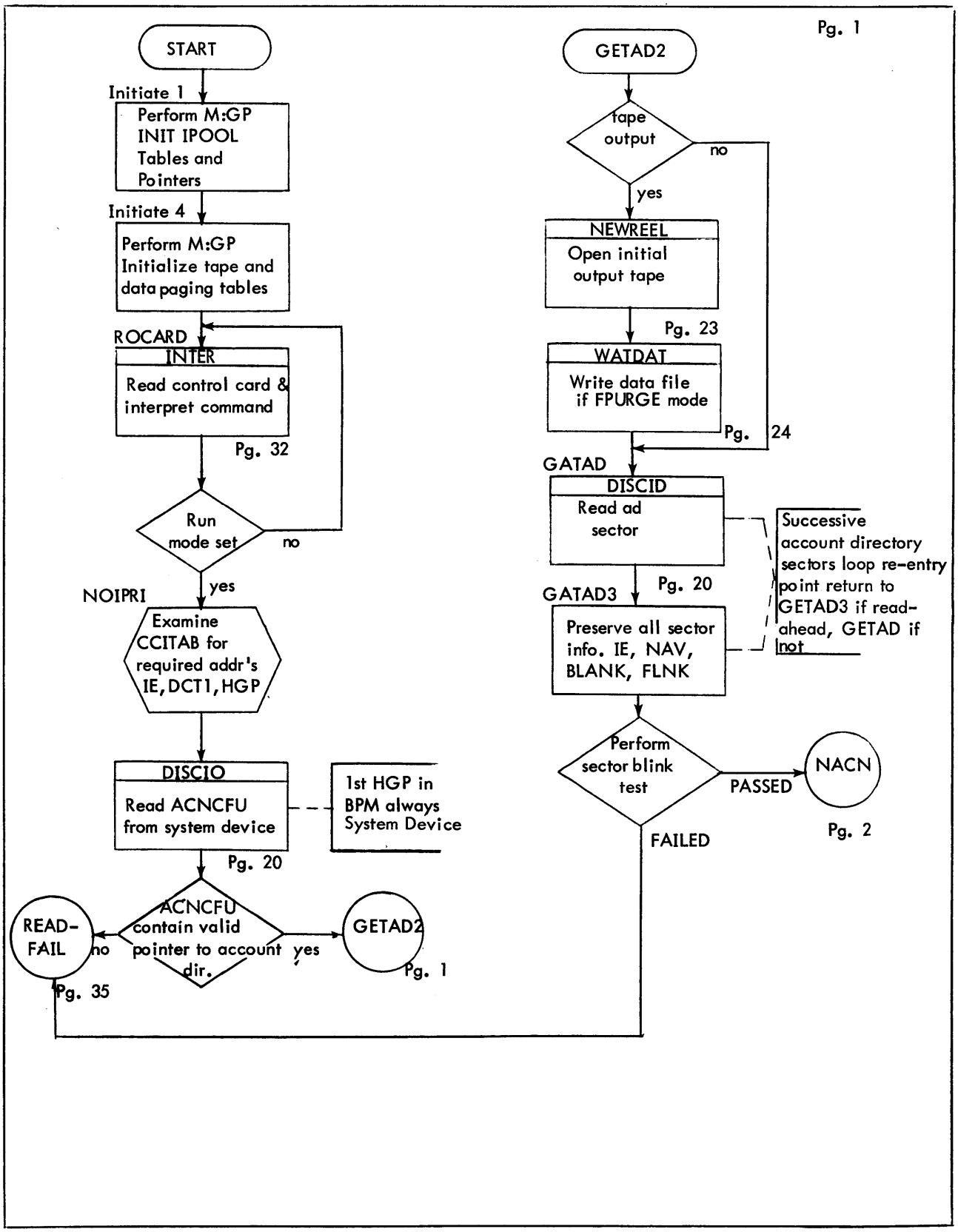


Figure 9-1. Detailed Flowchart of FILE SAVE

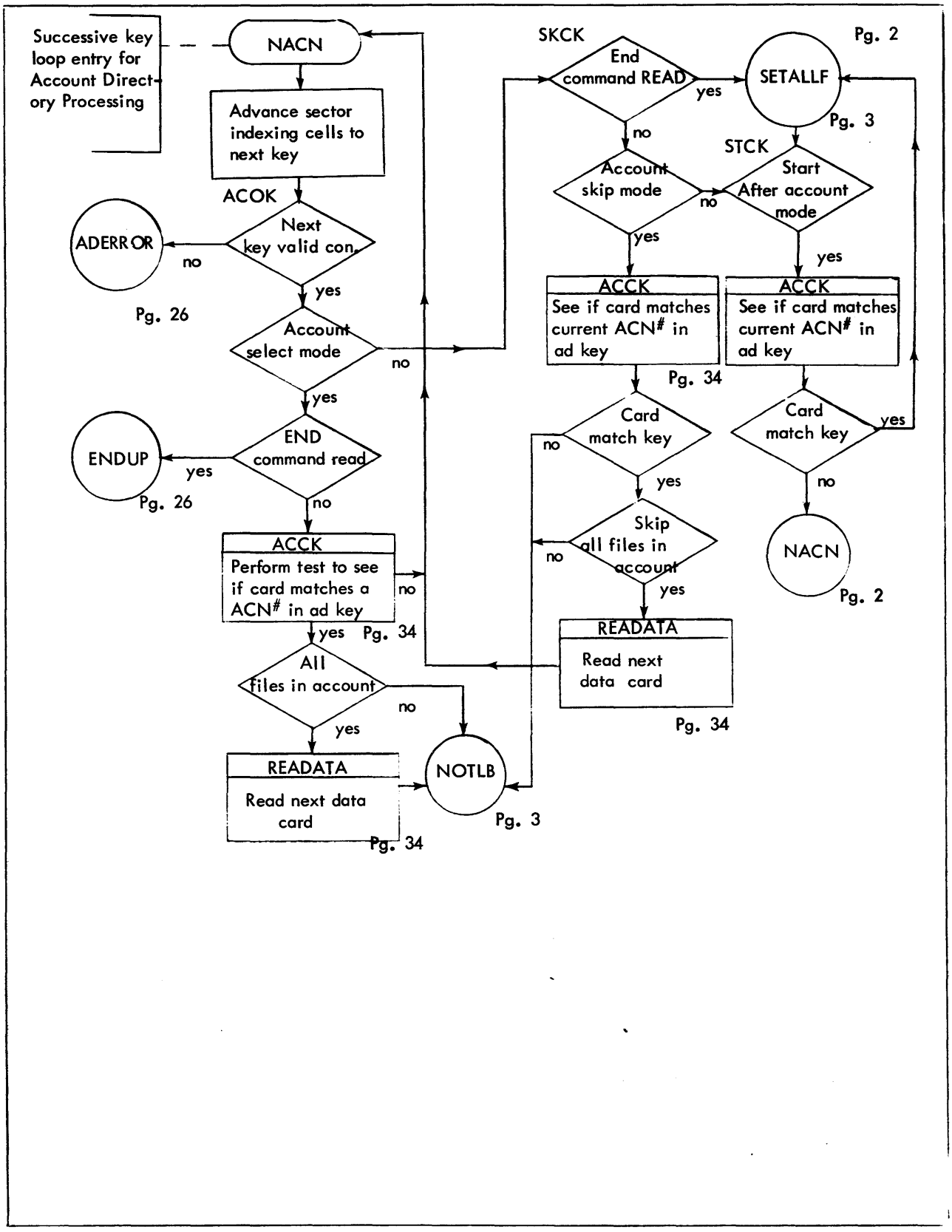


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

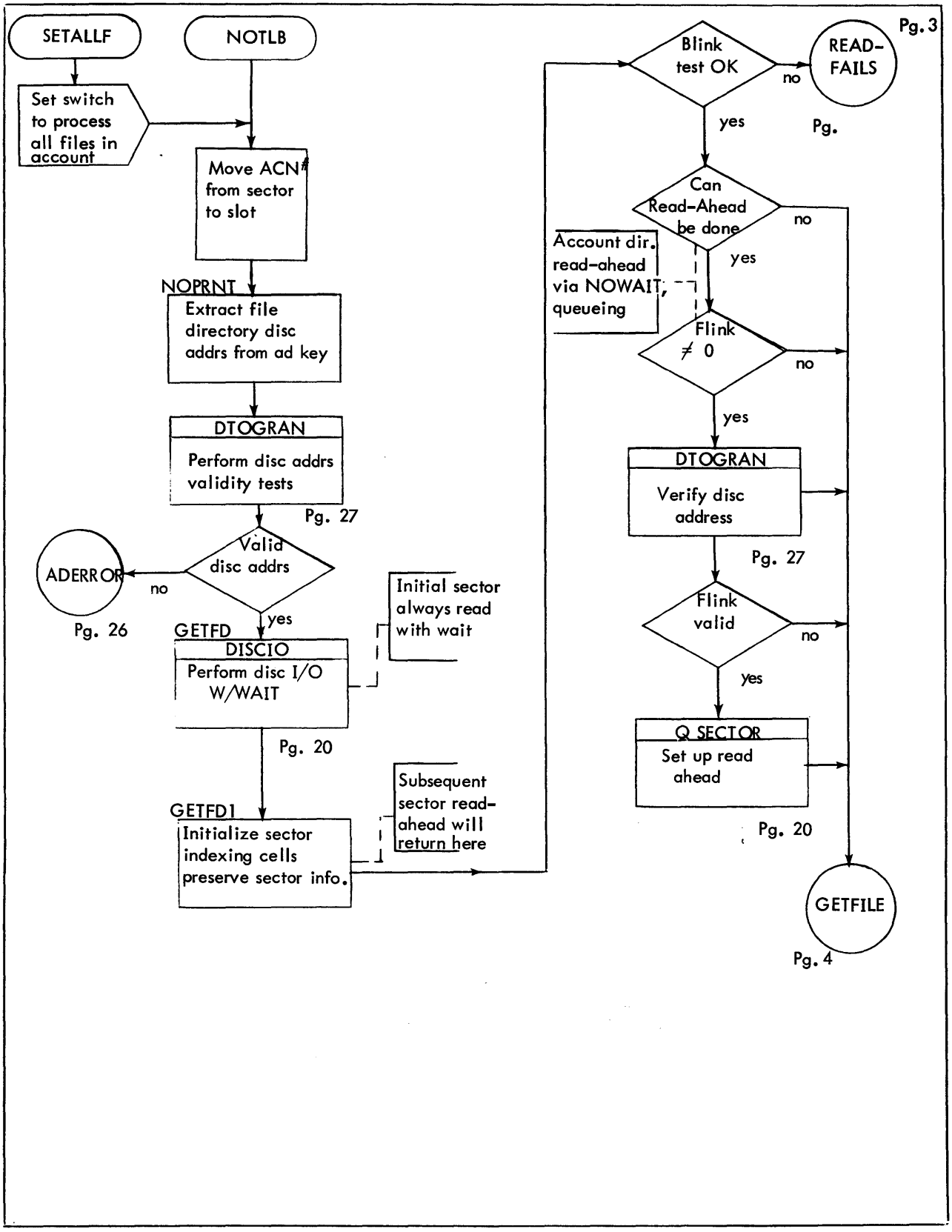


Figure 9-1. Deatailed Flowchart of FILE SAVE (Cont.)

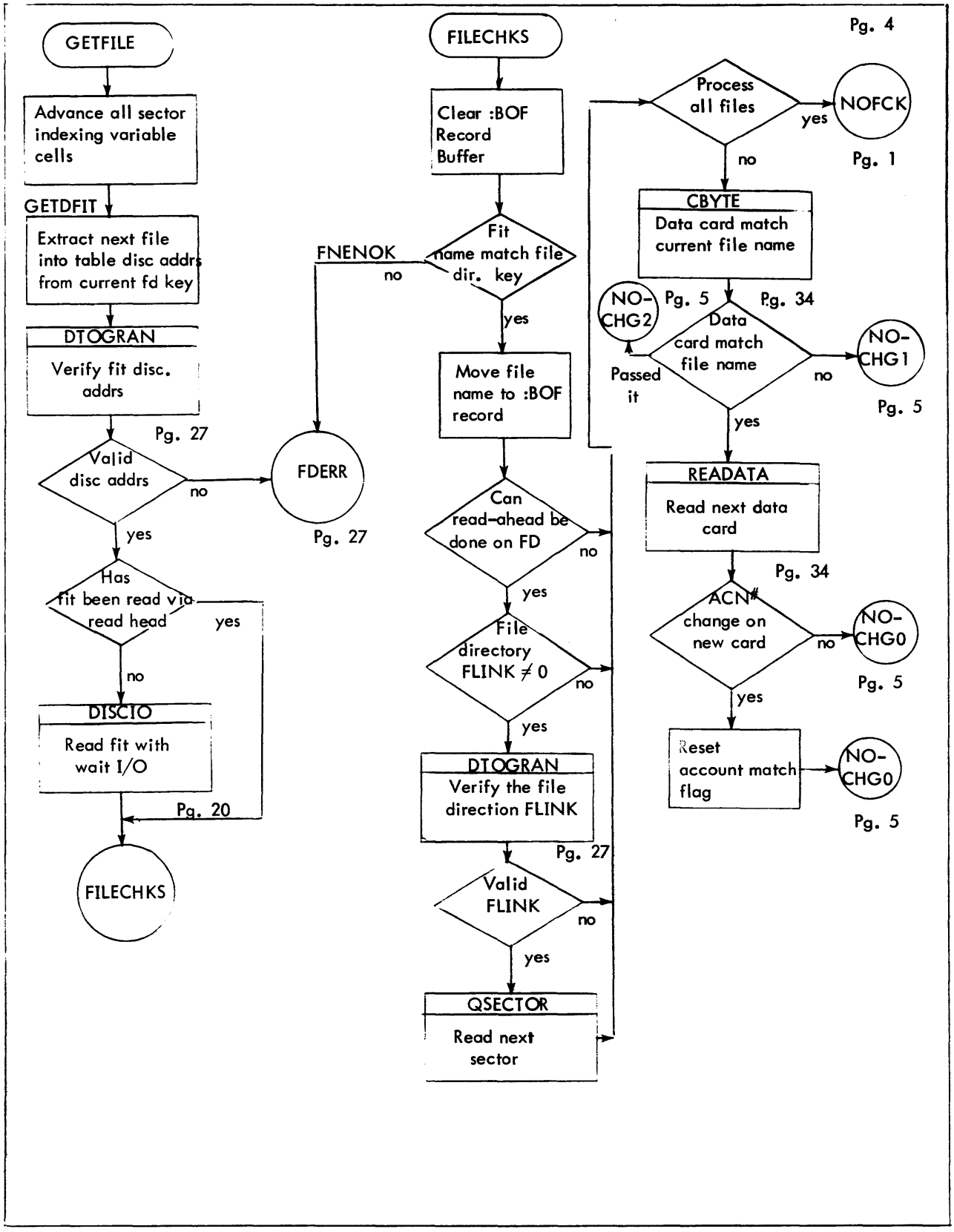


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

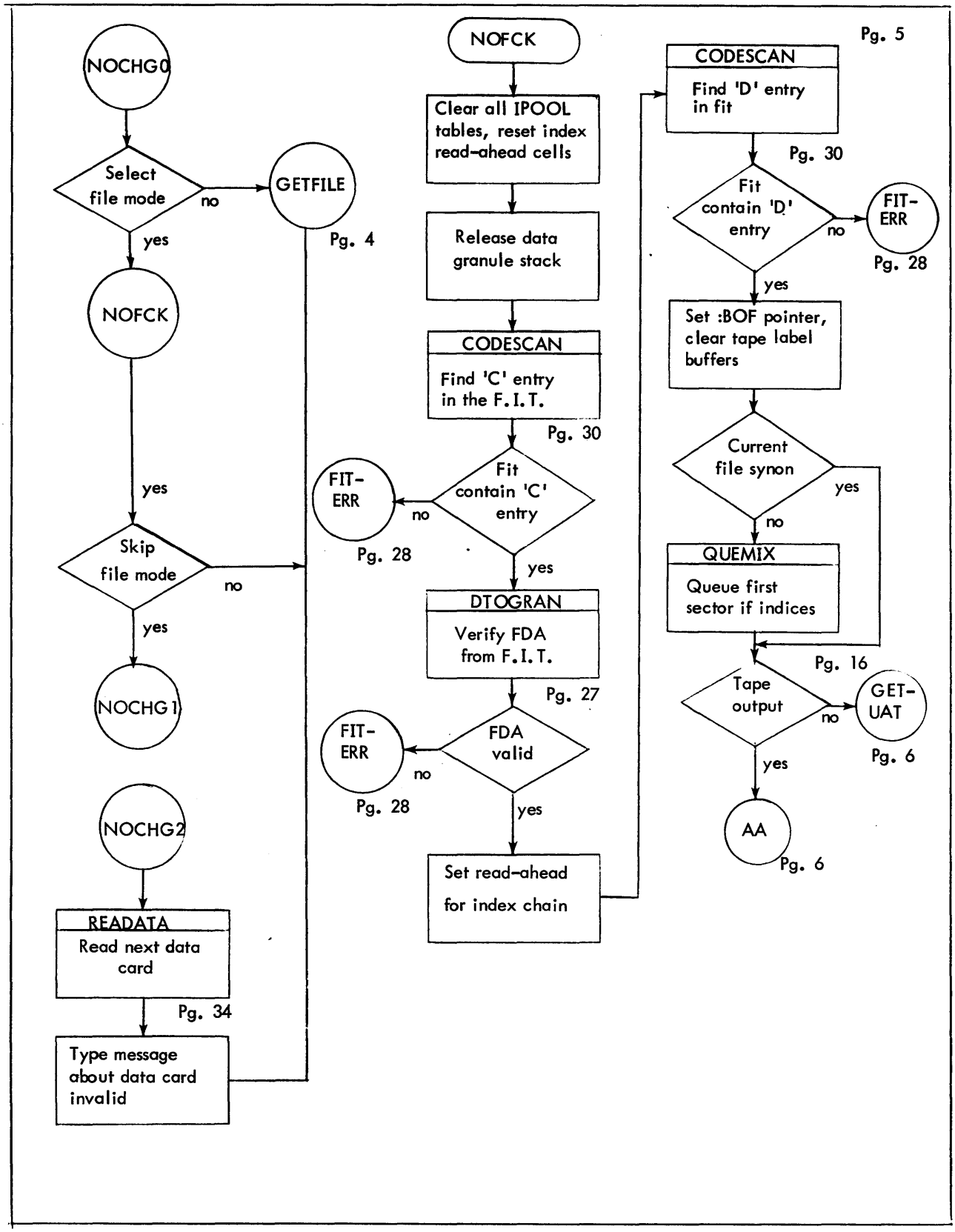


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

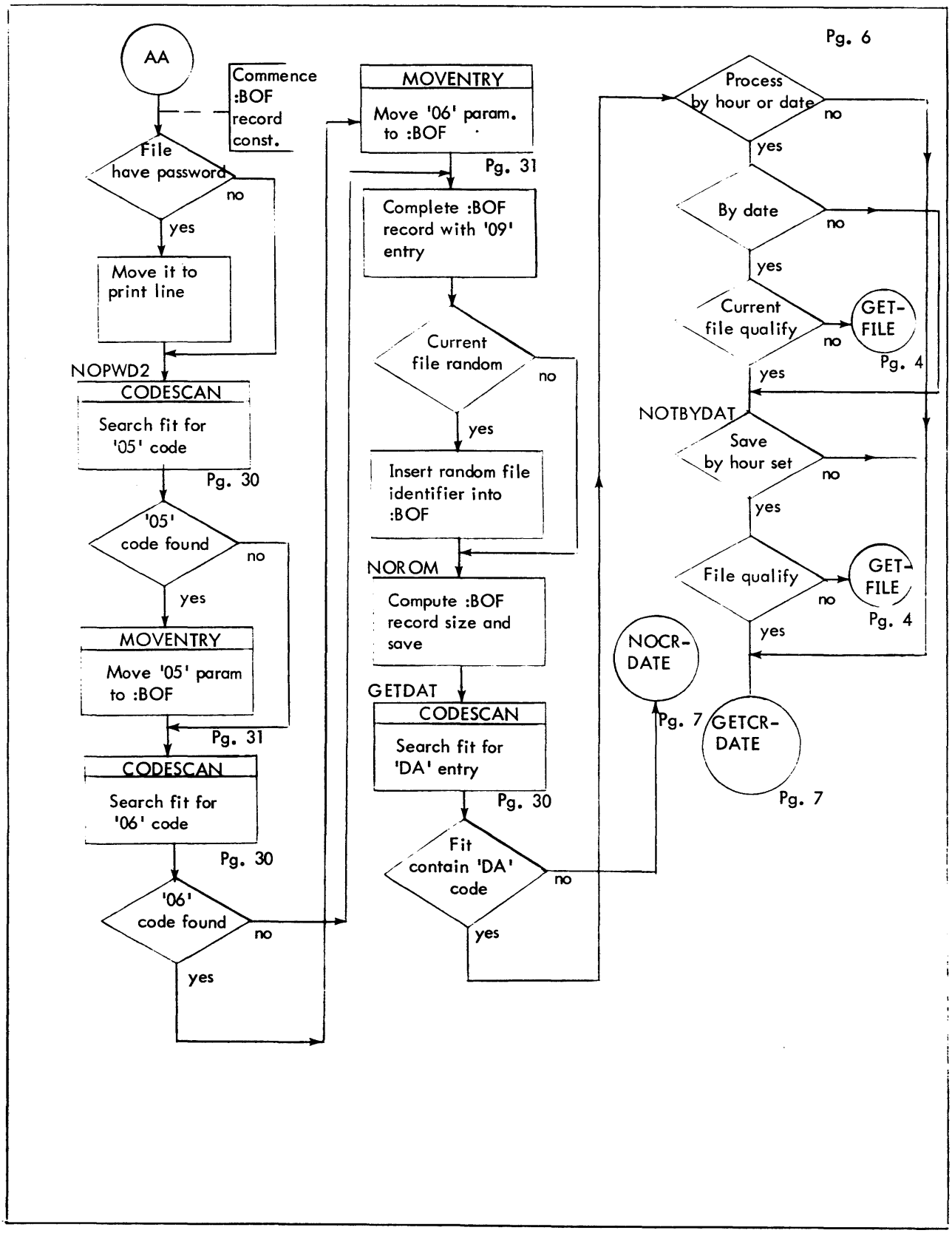


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

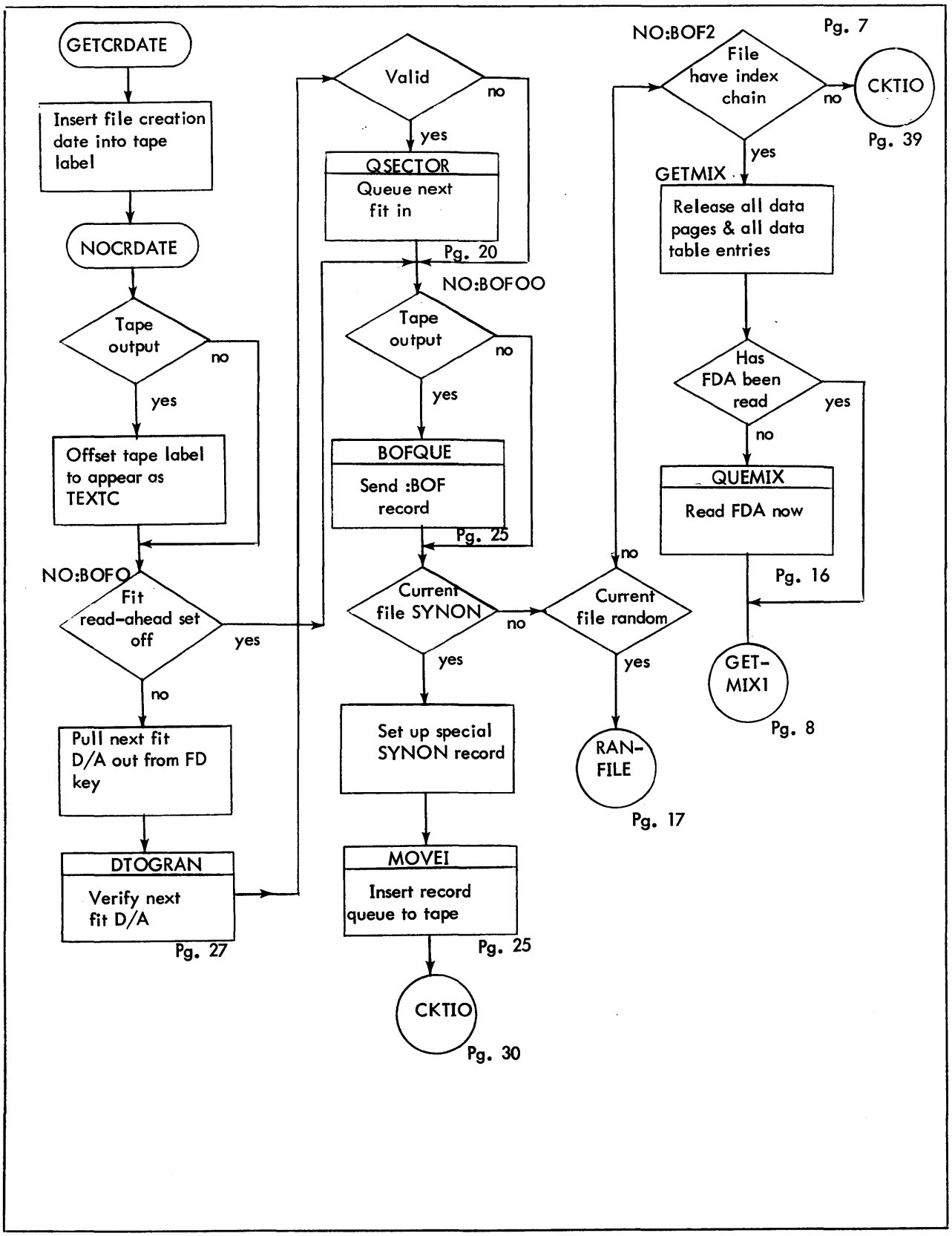


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

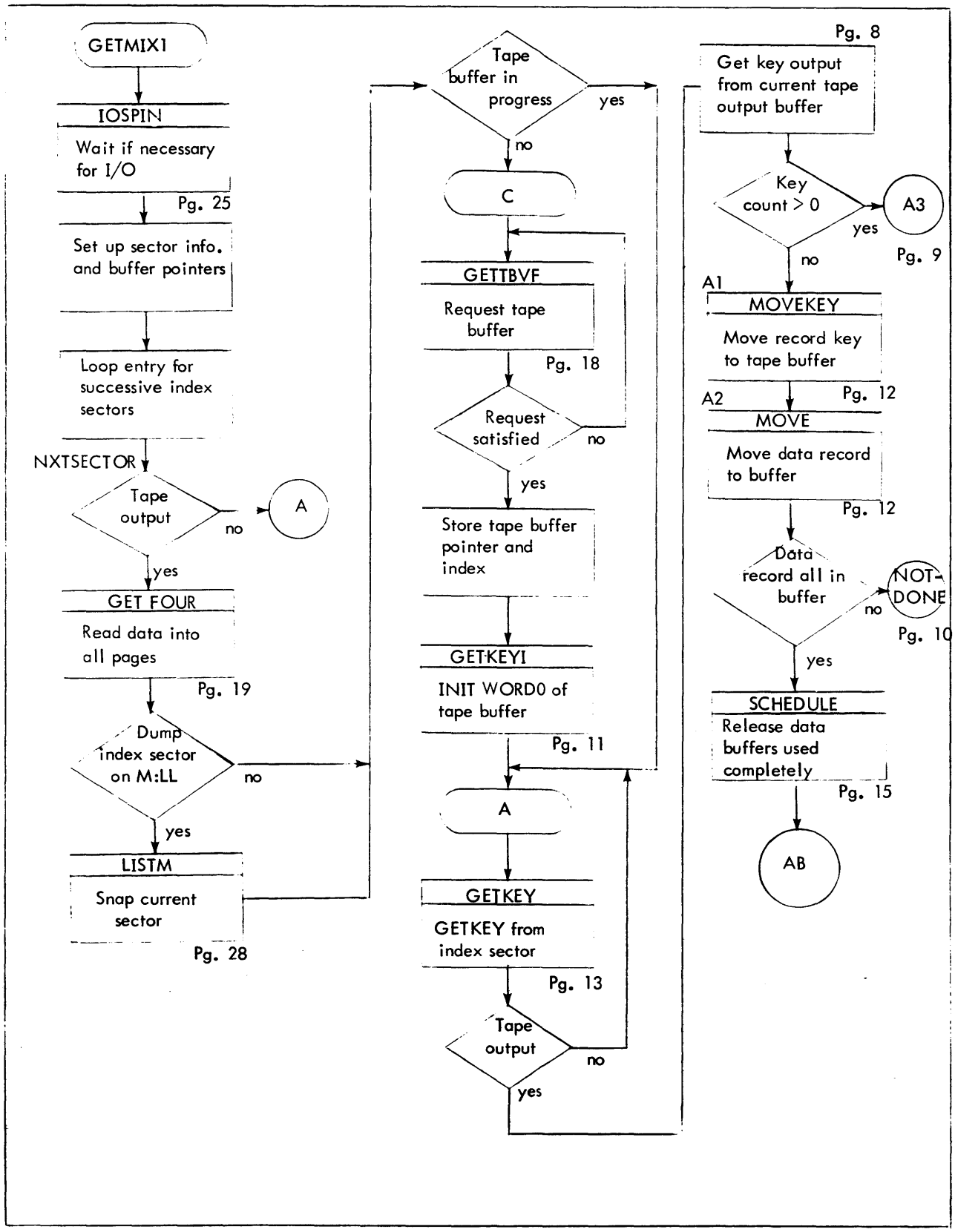


Figure 9-1. Detailed flowchart of FILE SAVE (Cont.)

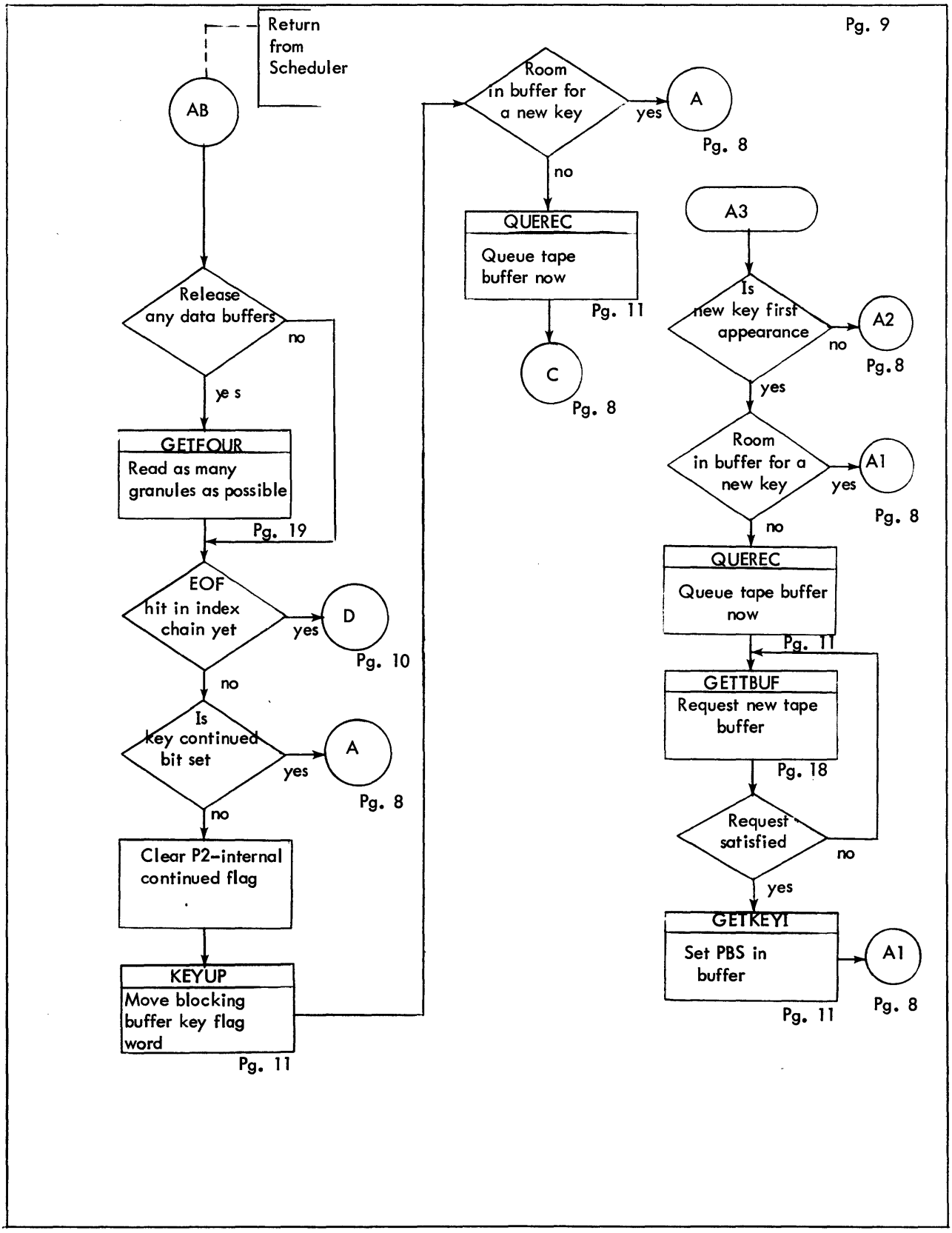


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

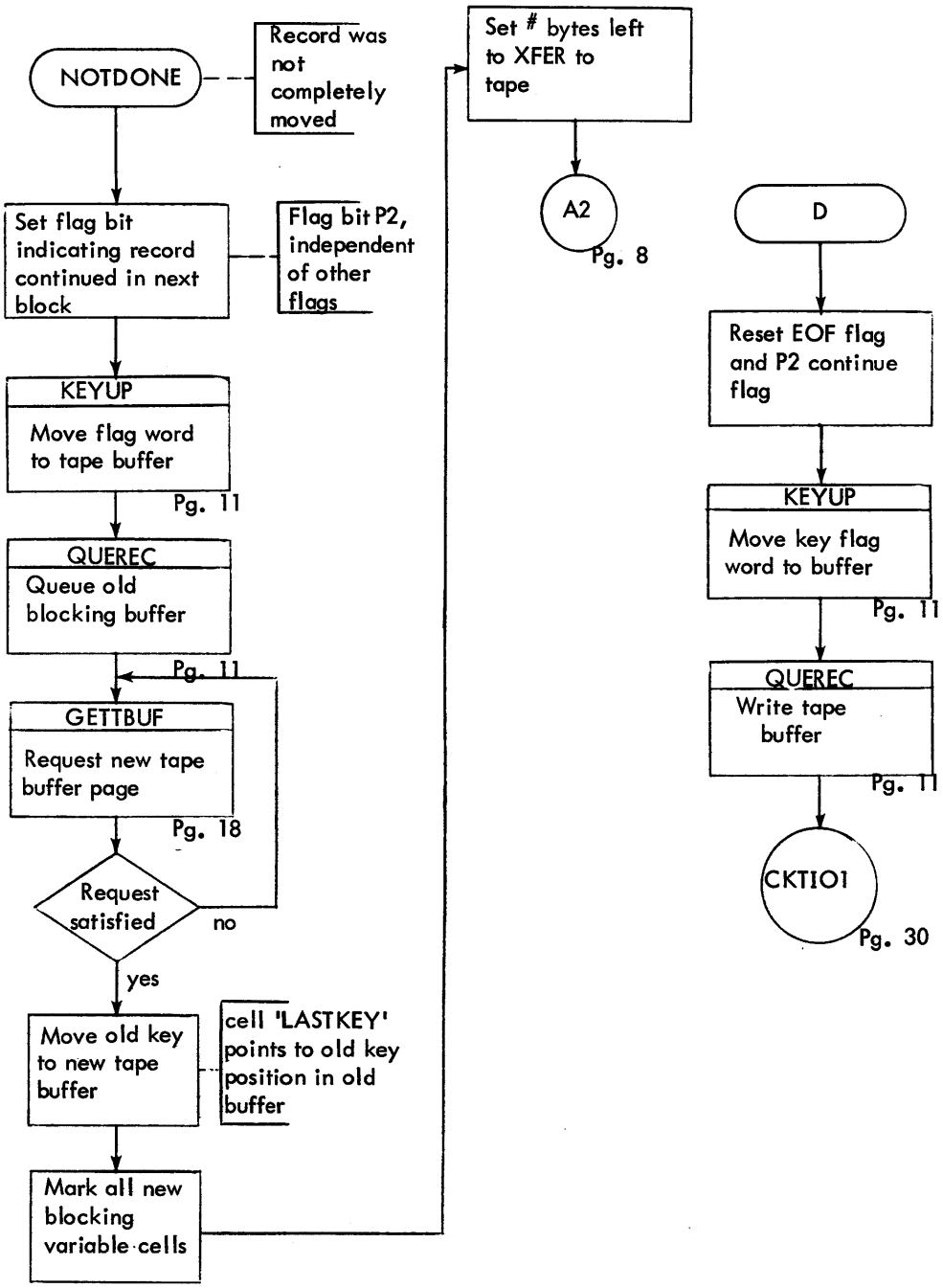


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

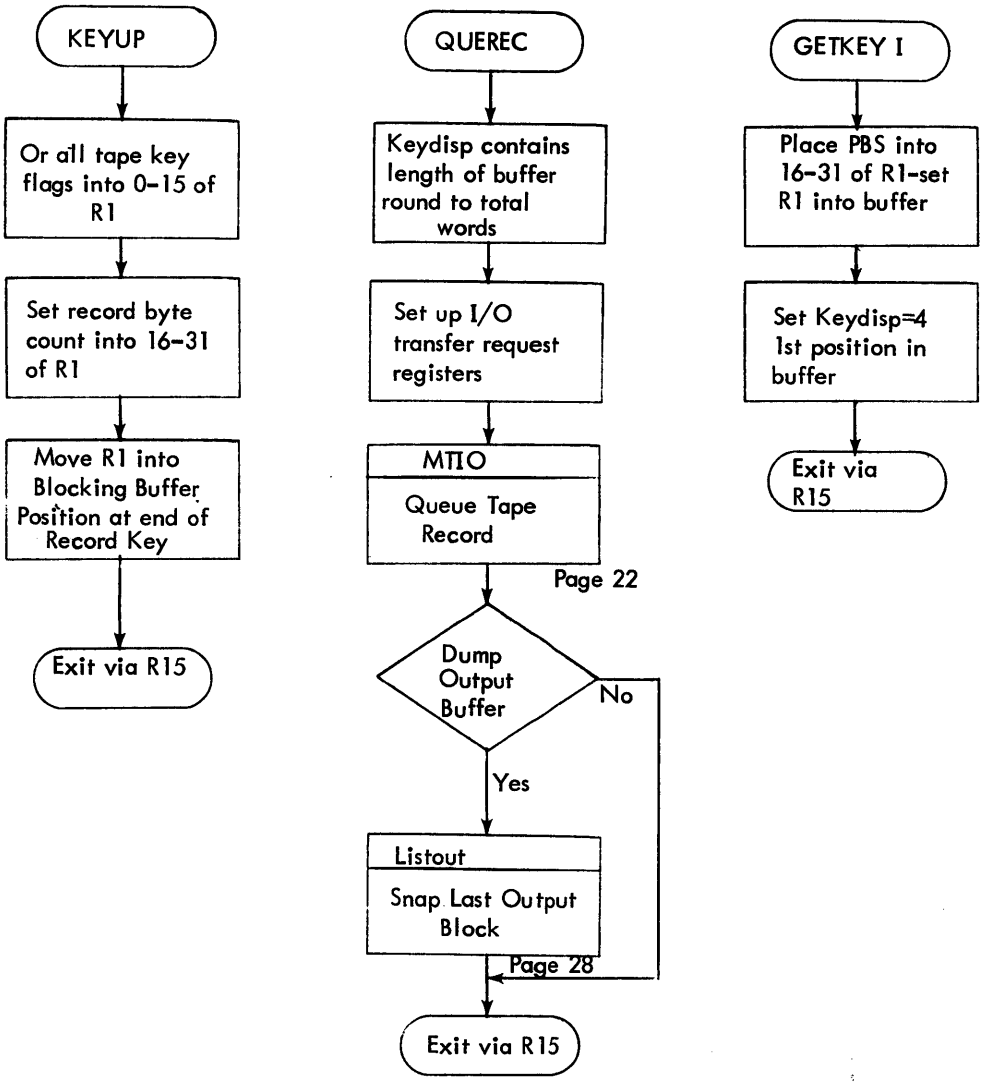


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

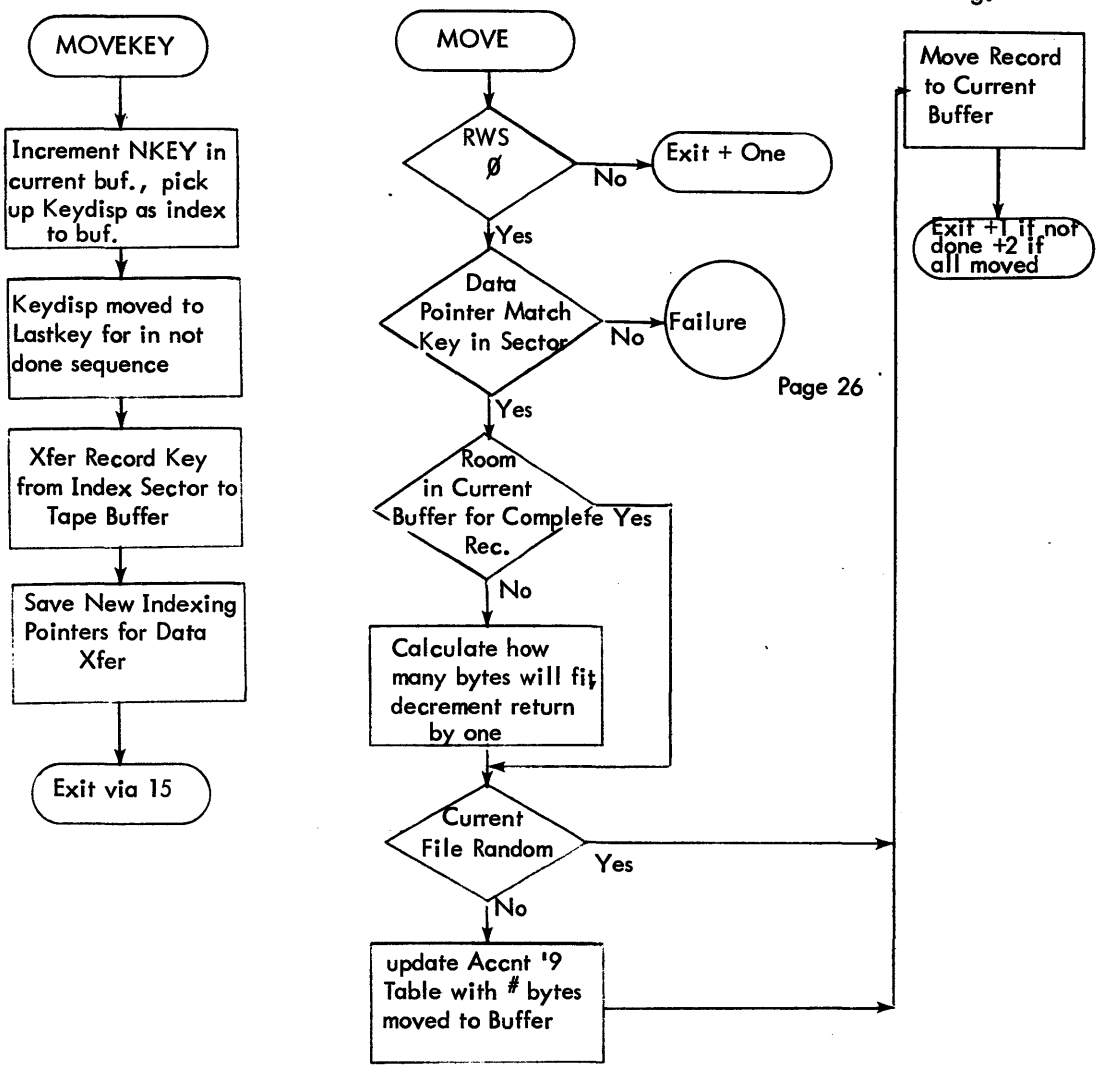


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

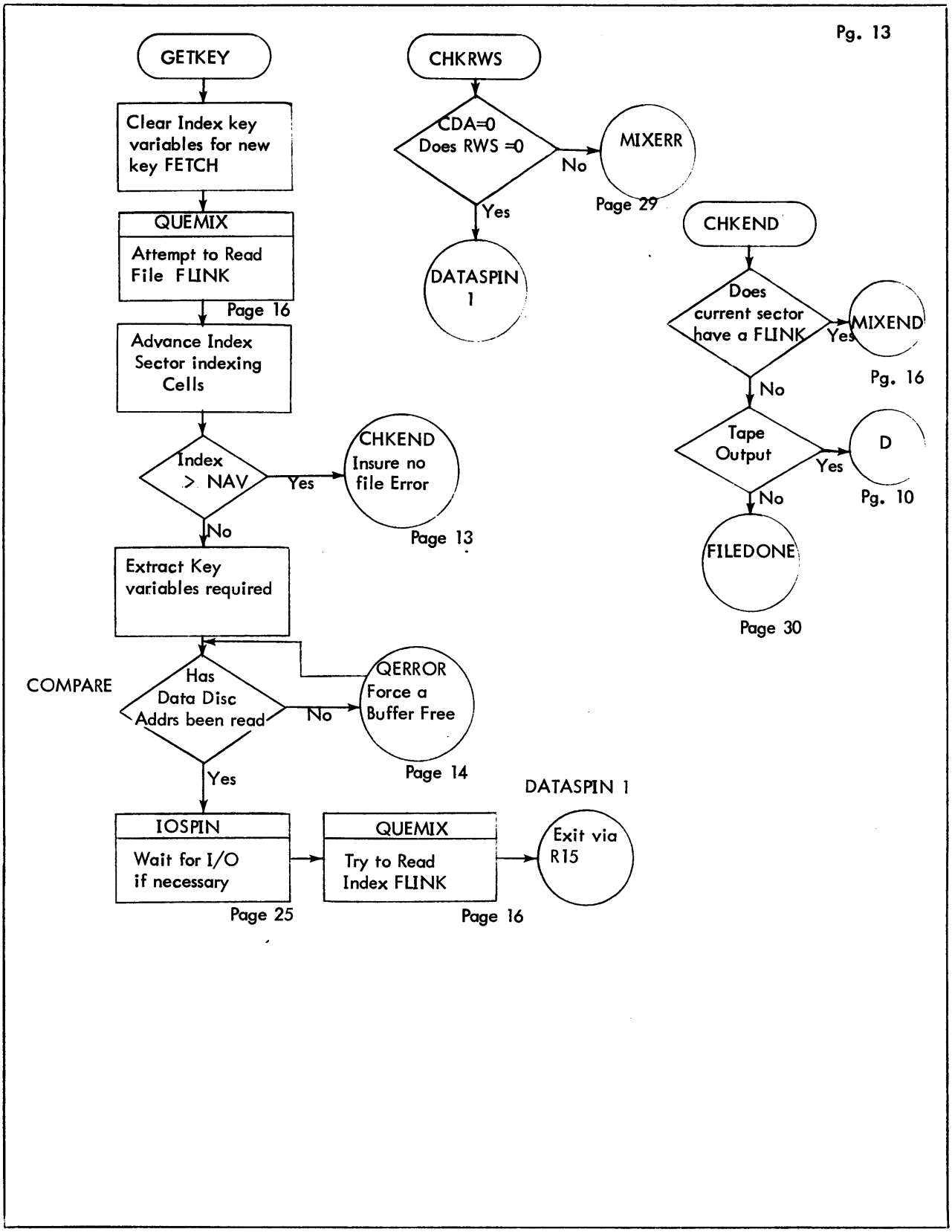


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

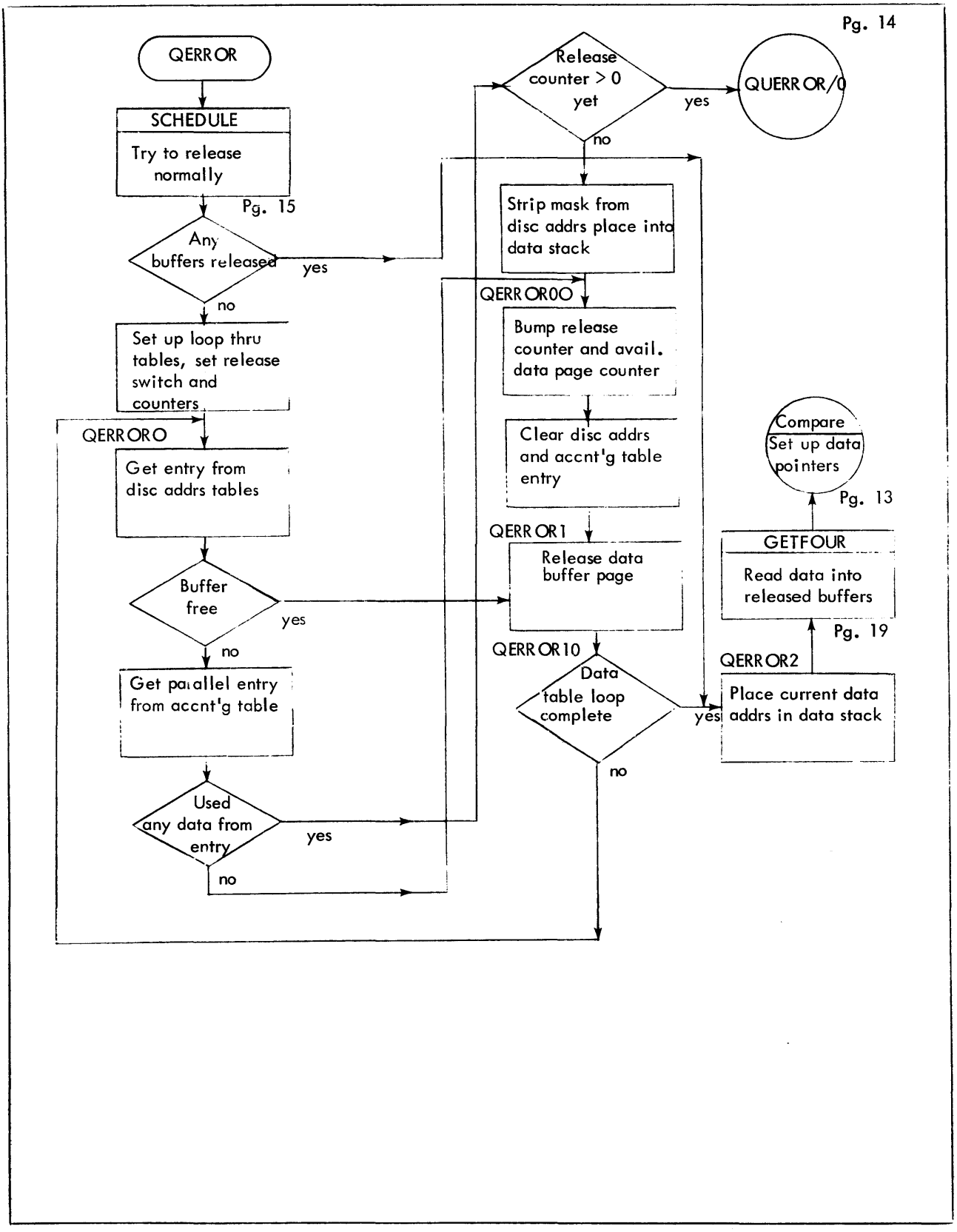


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

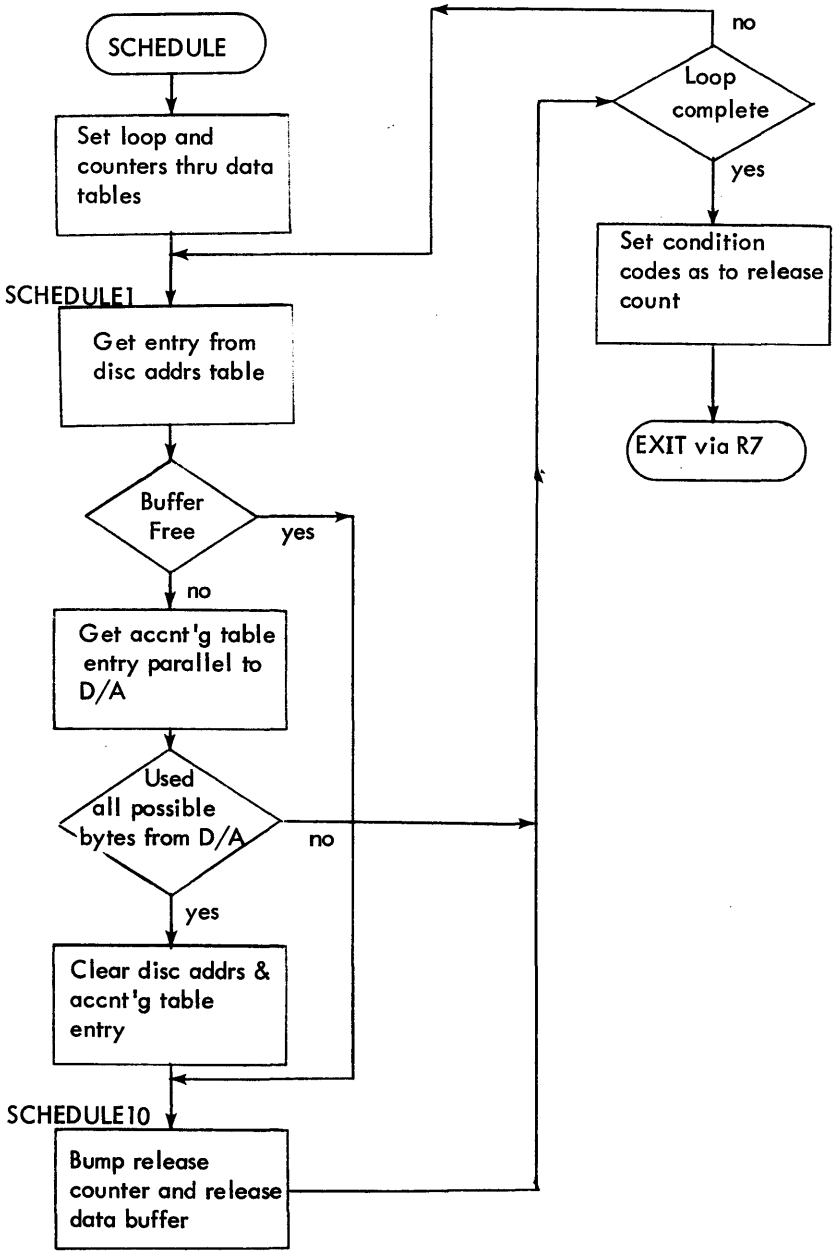


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

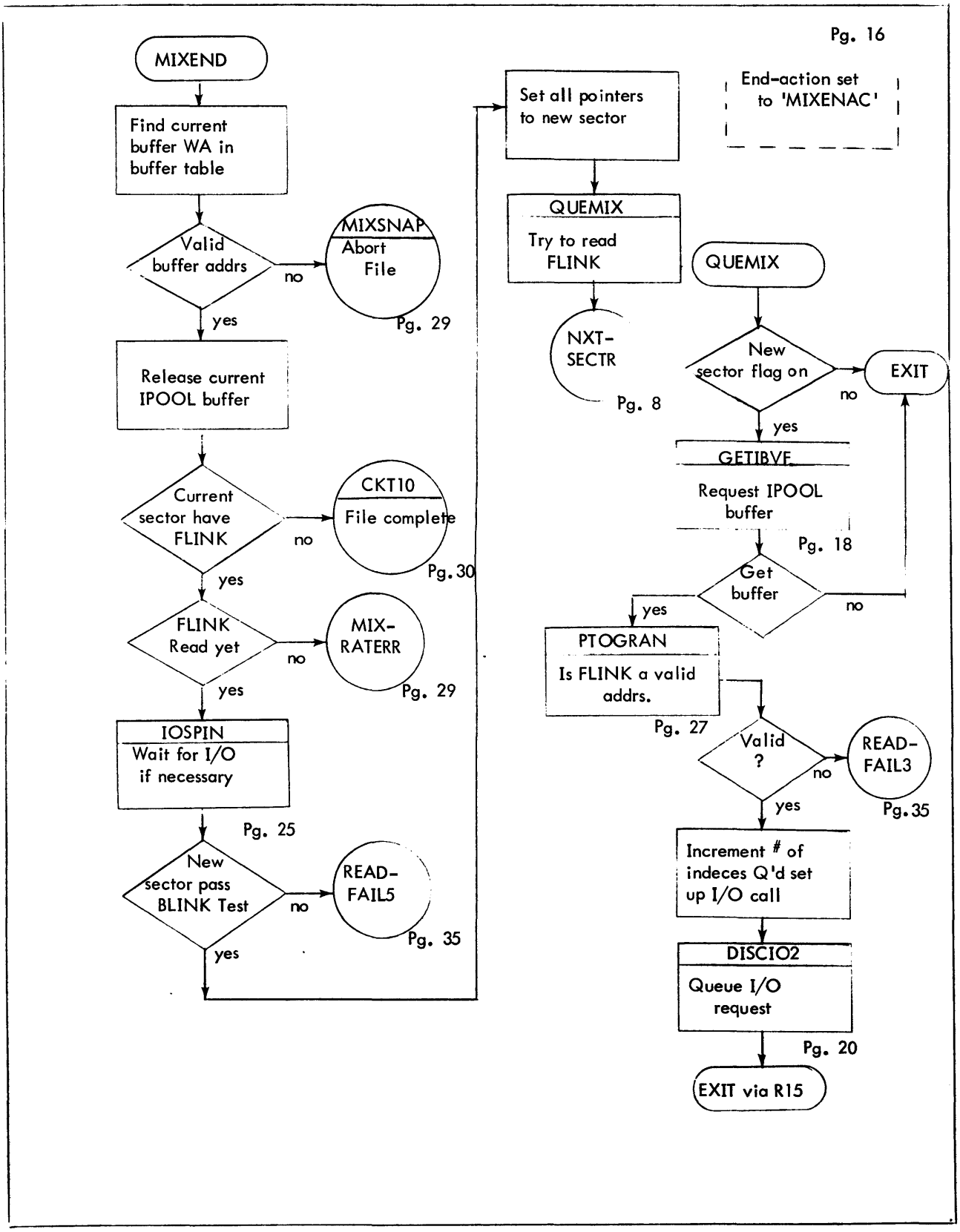


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

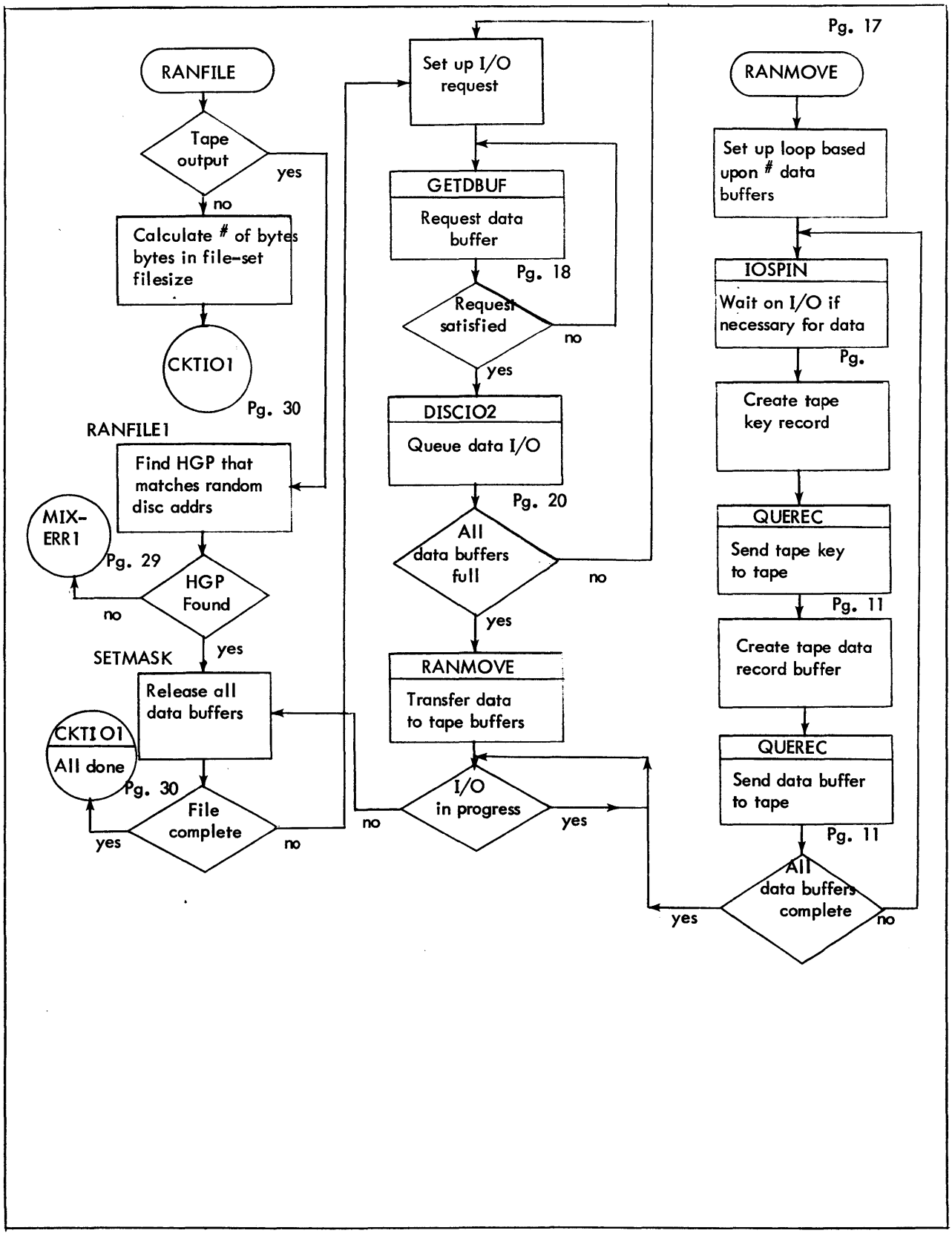


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

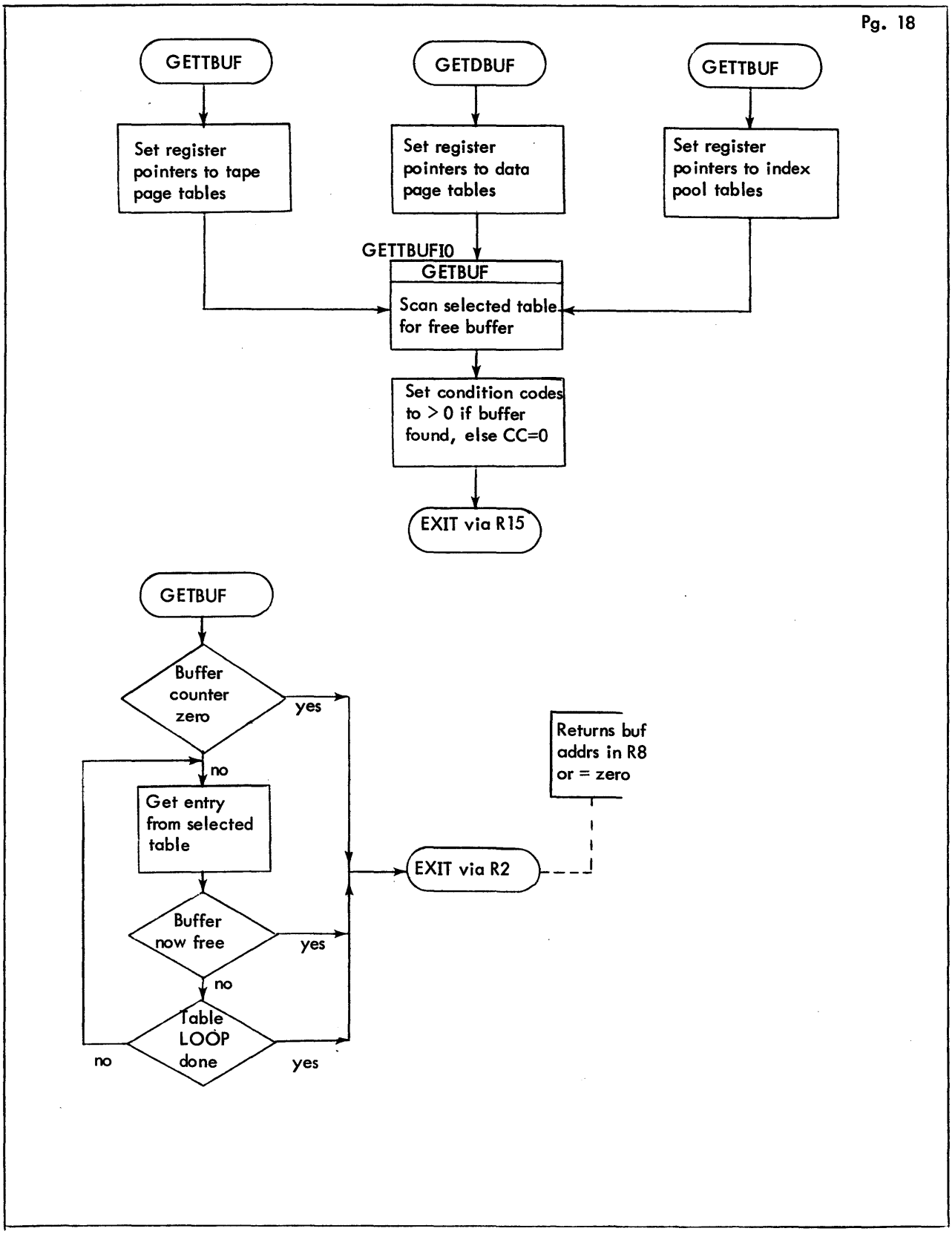


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

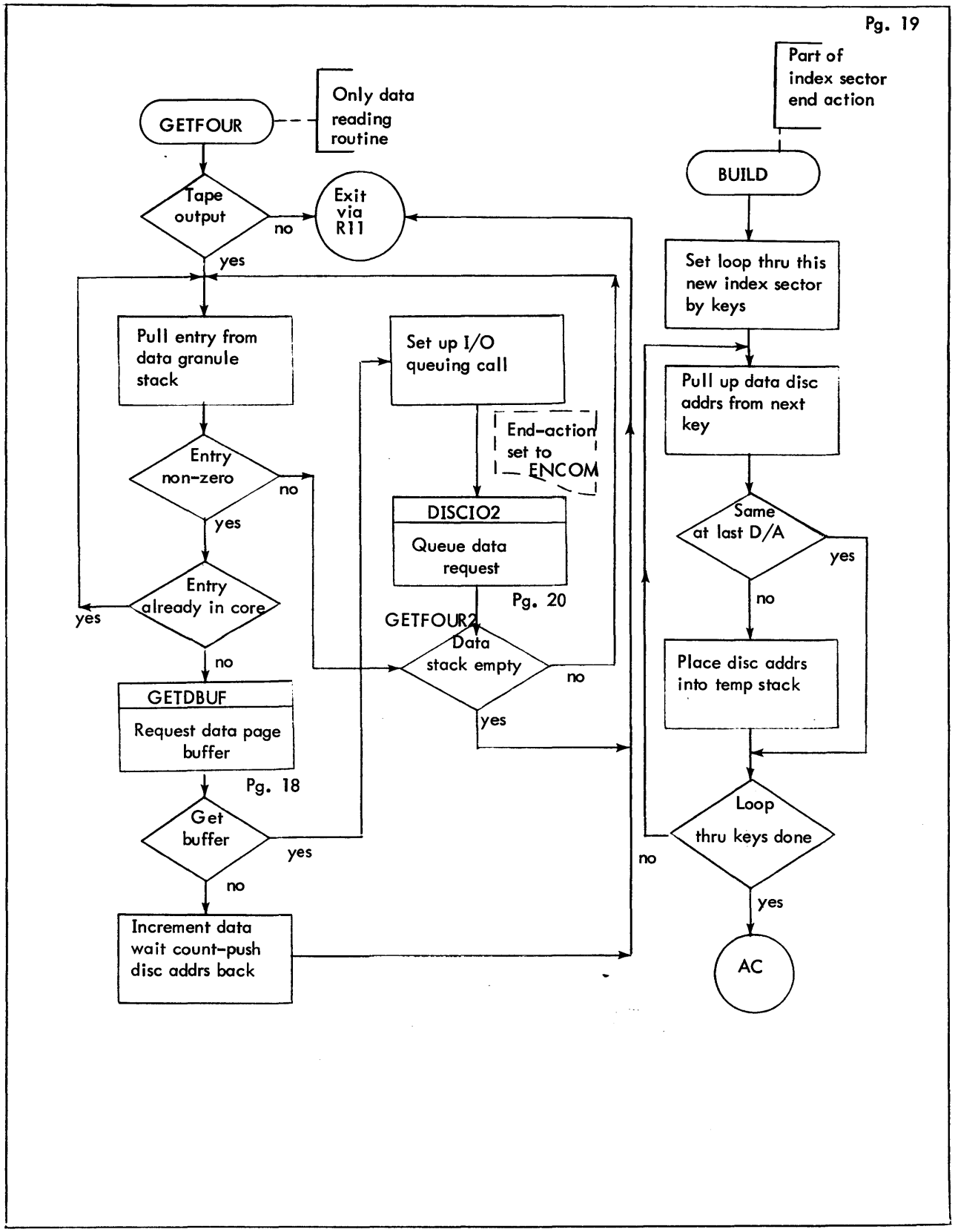


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

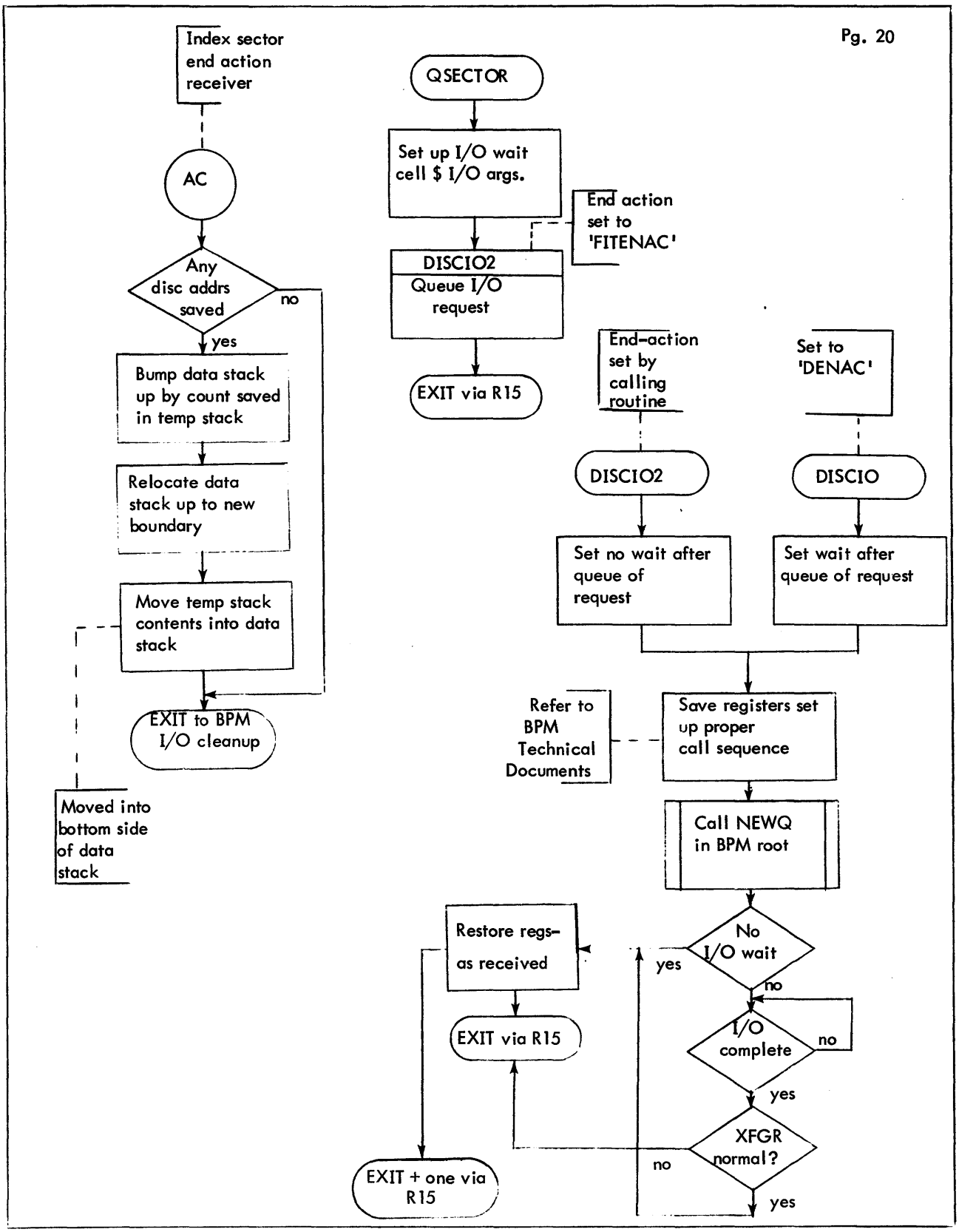
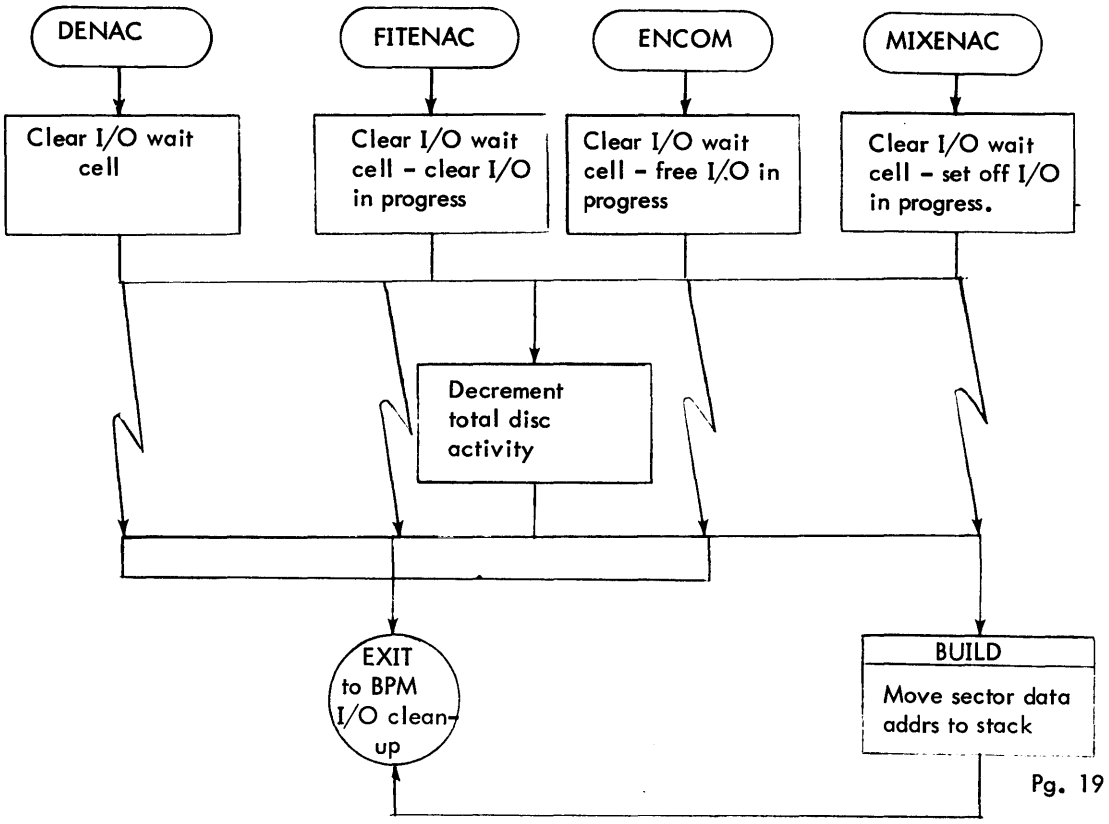


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)



All End-Action Receivers for Disc I/O Activity

Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

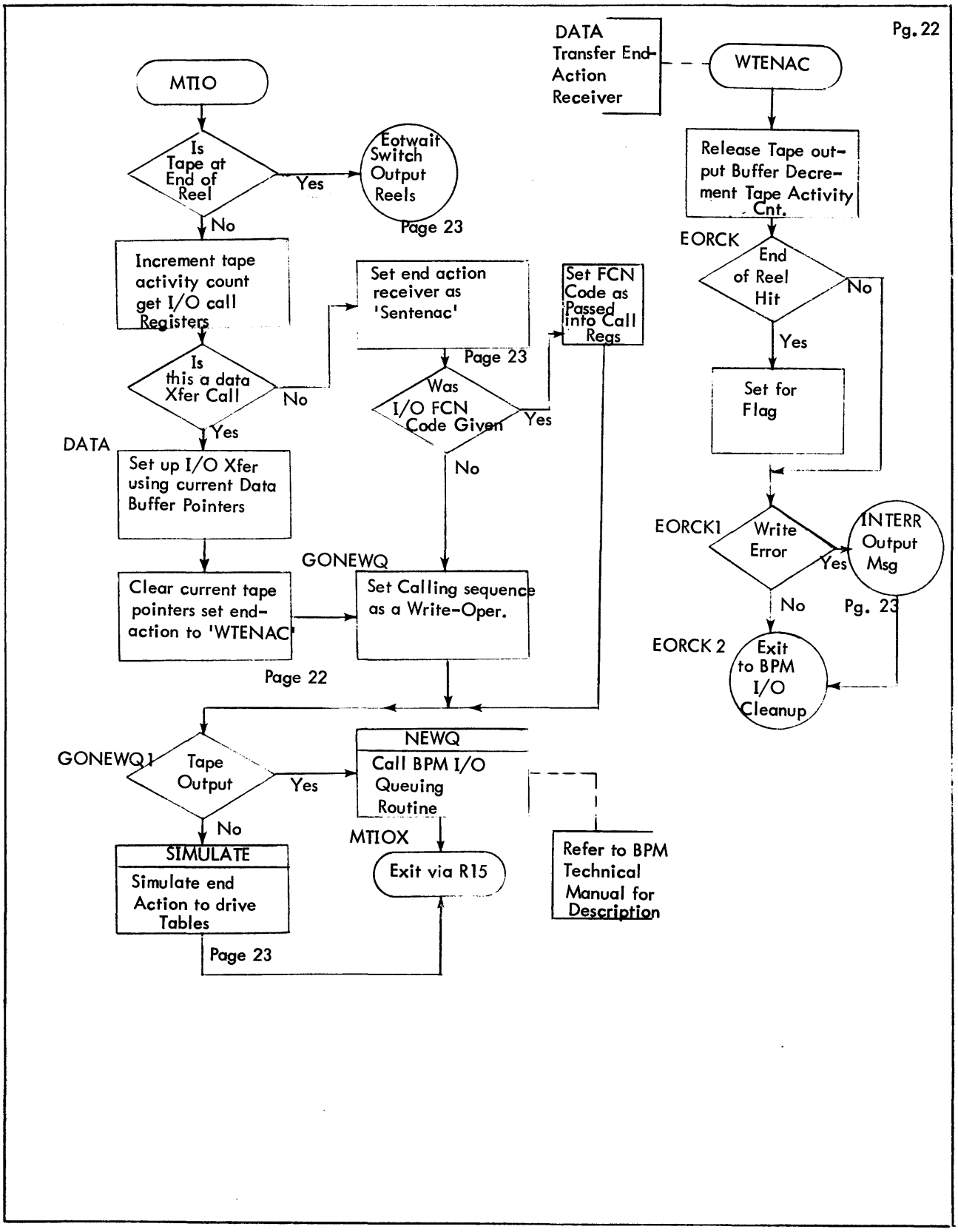


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

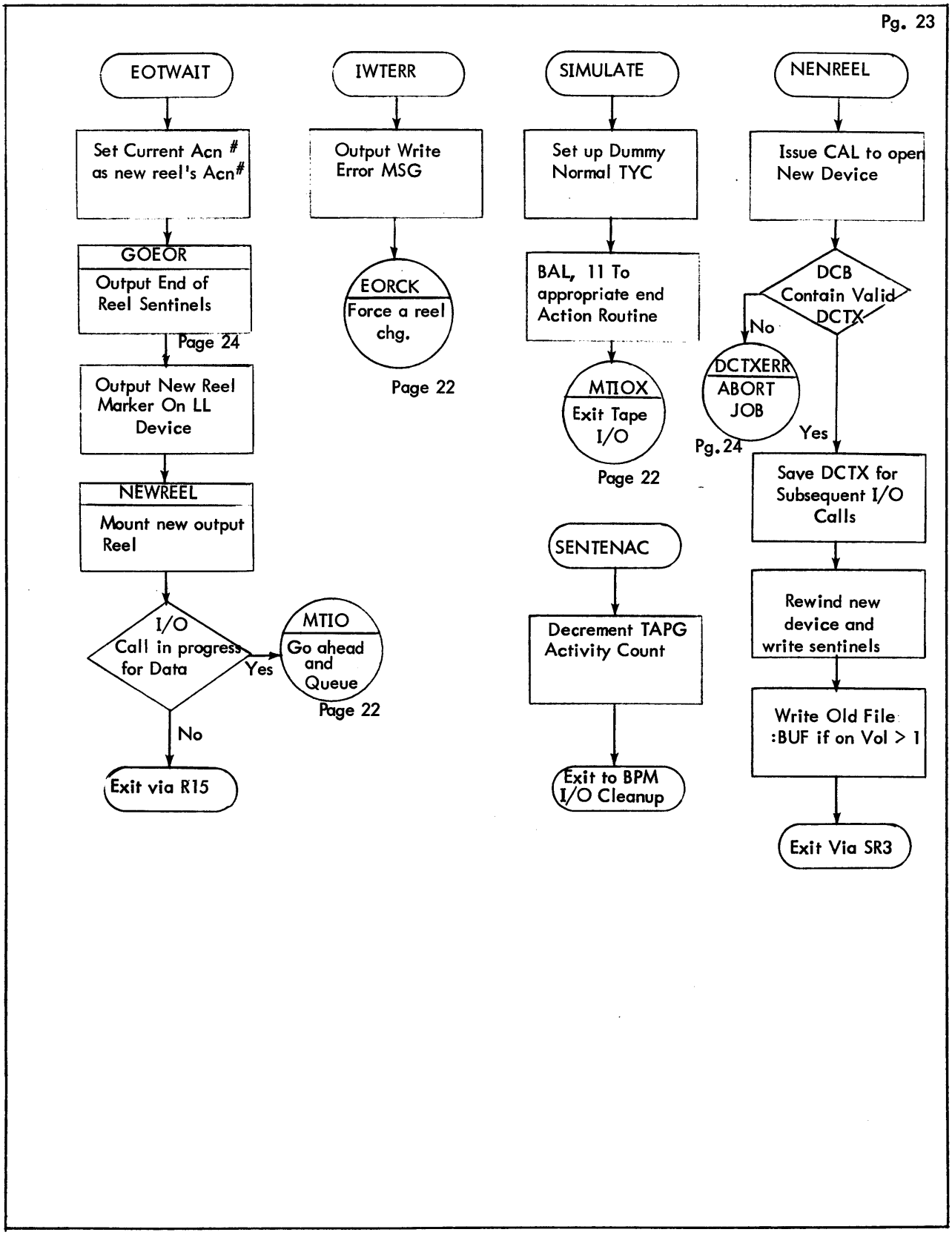


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

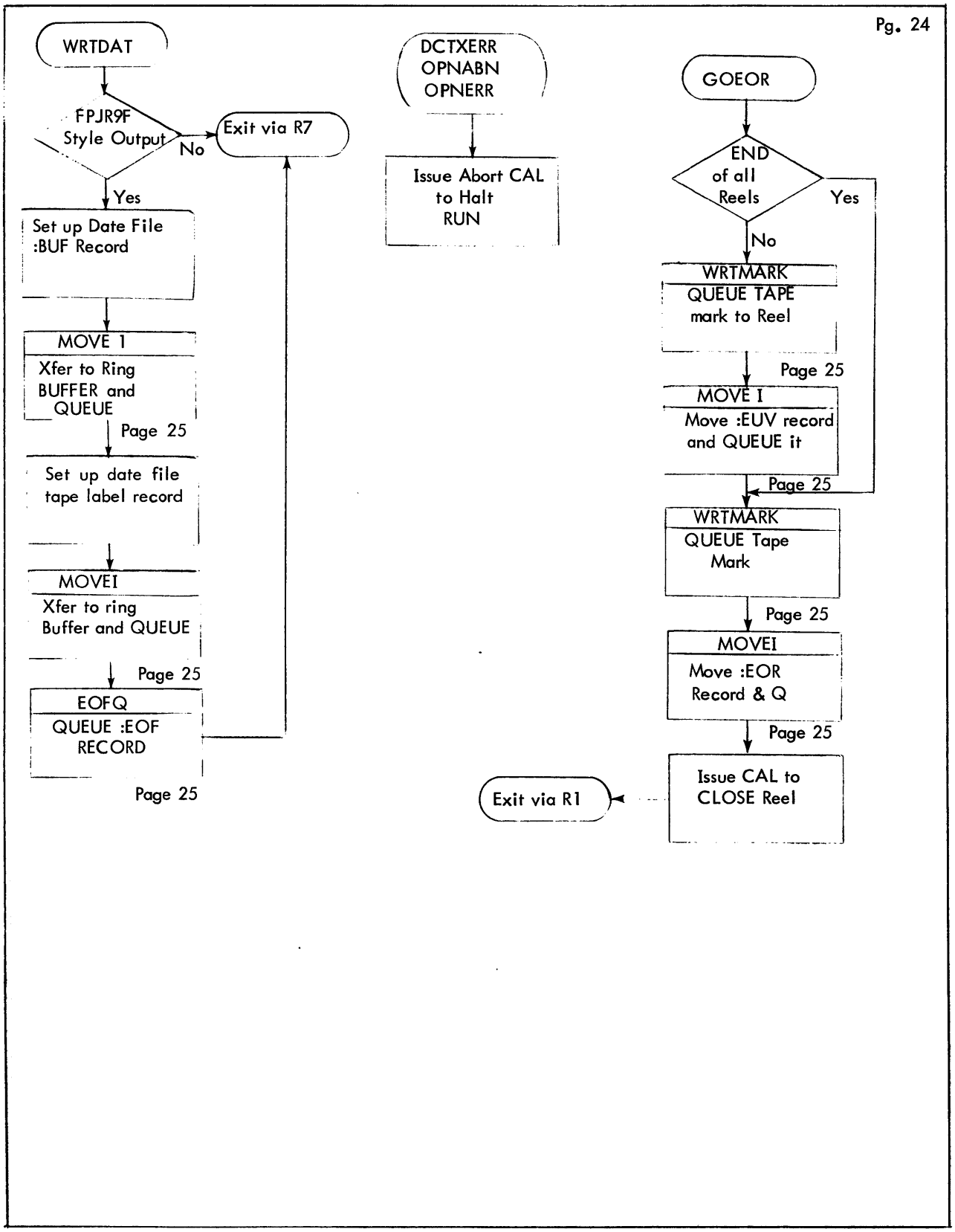


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

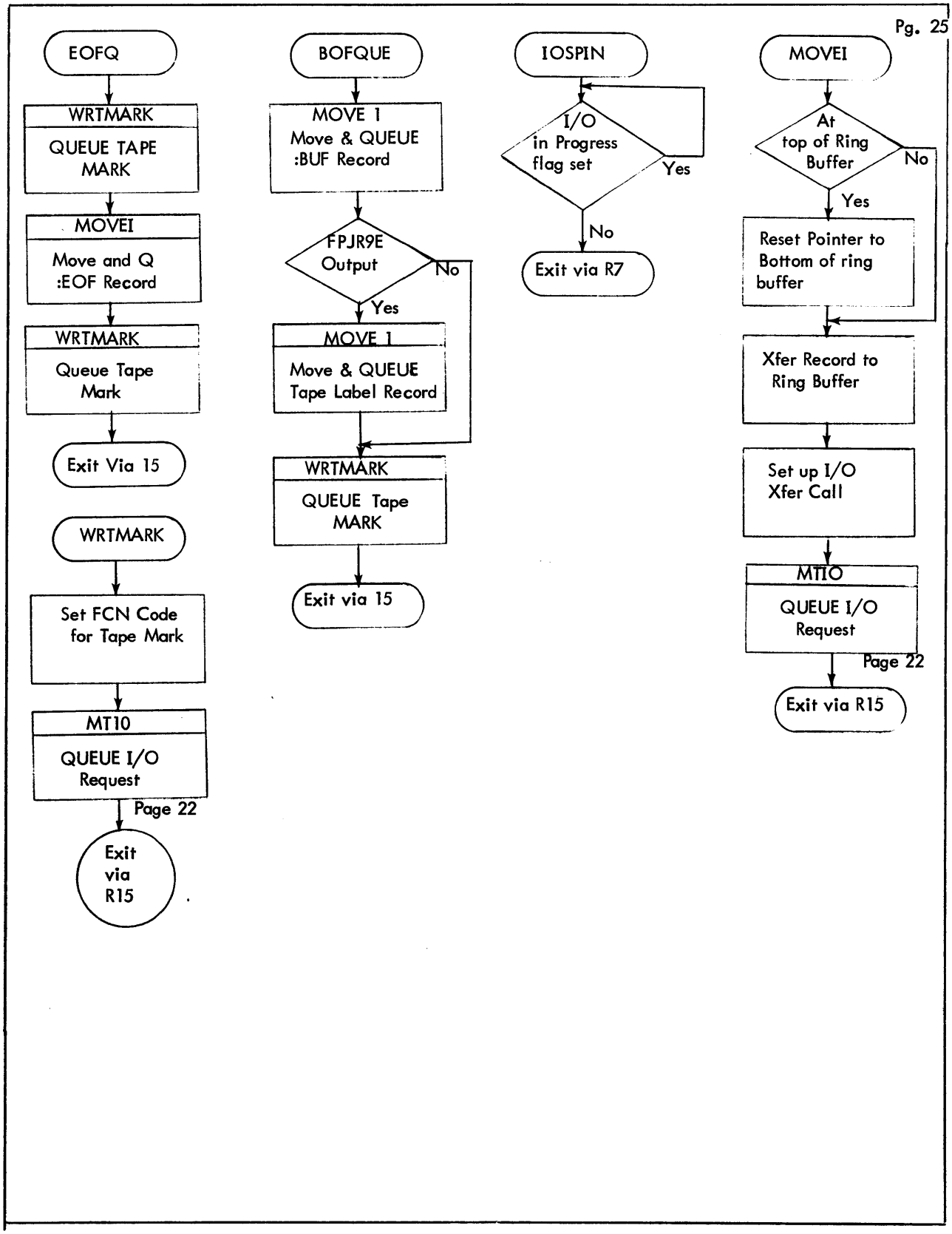


Figure 9-1. Detailed Format of FILE SAVE (Cont.)

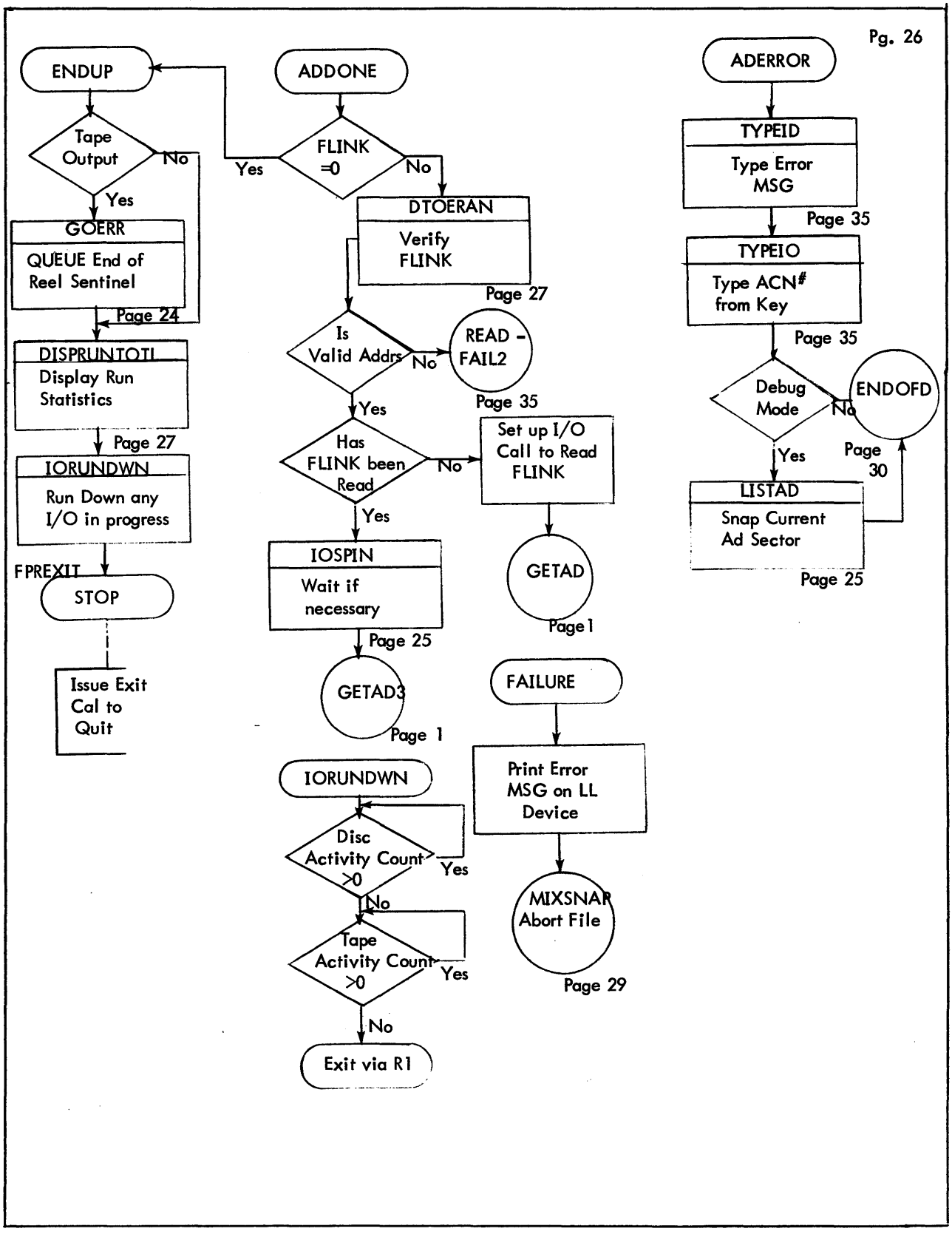


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

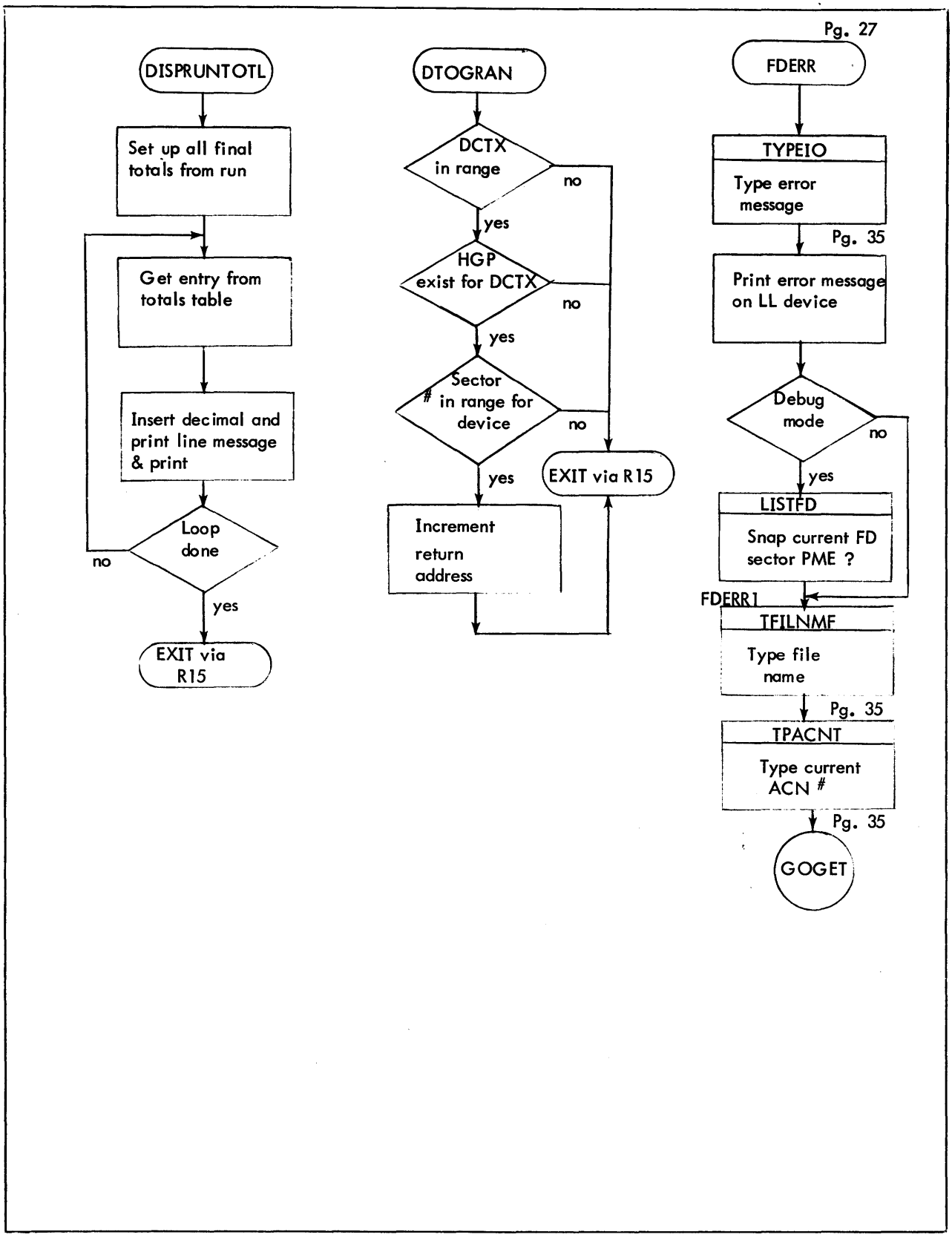


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

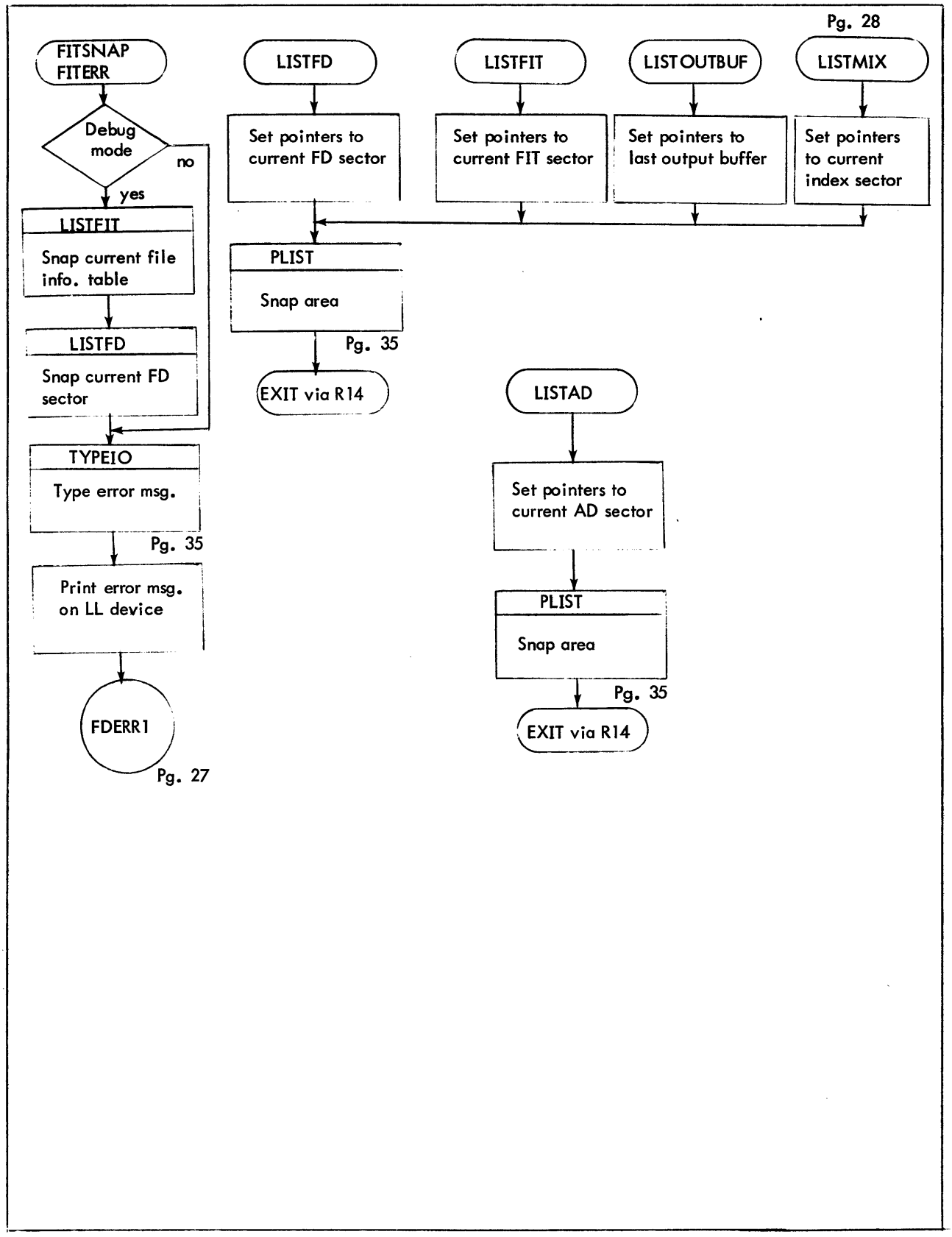


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

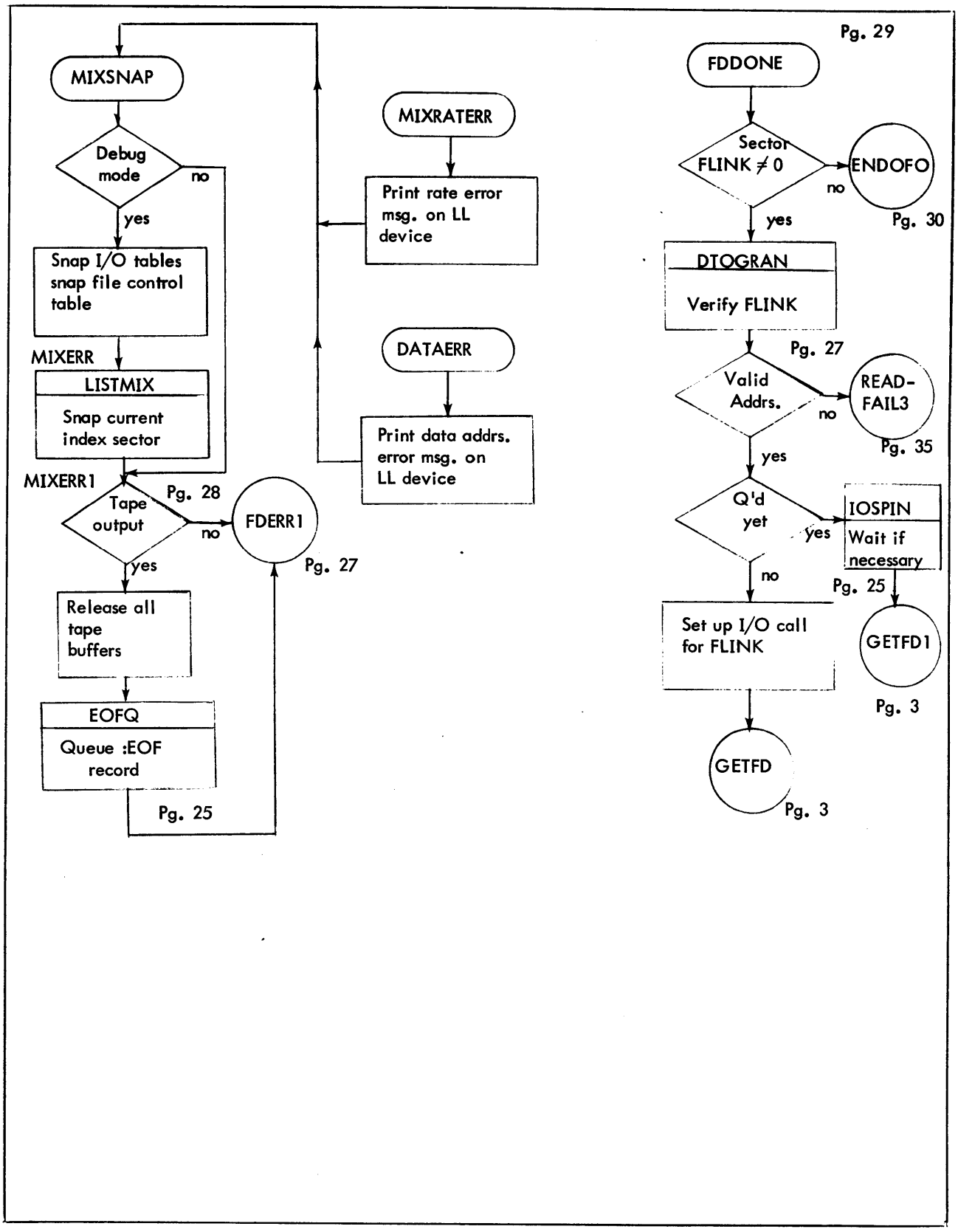


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

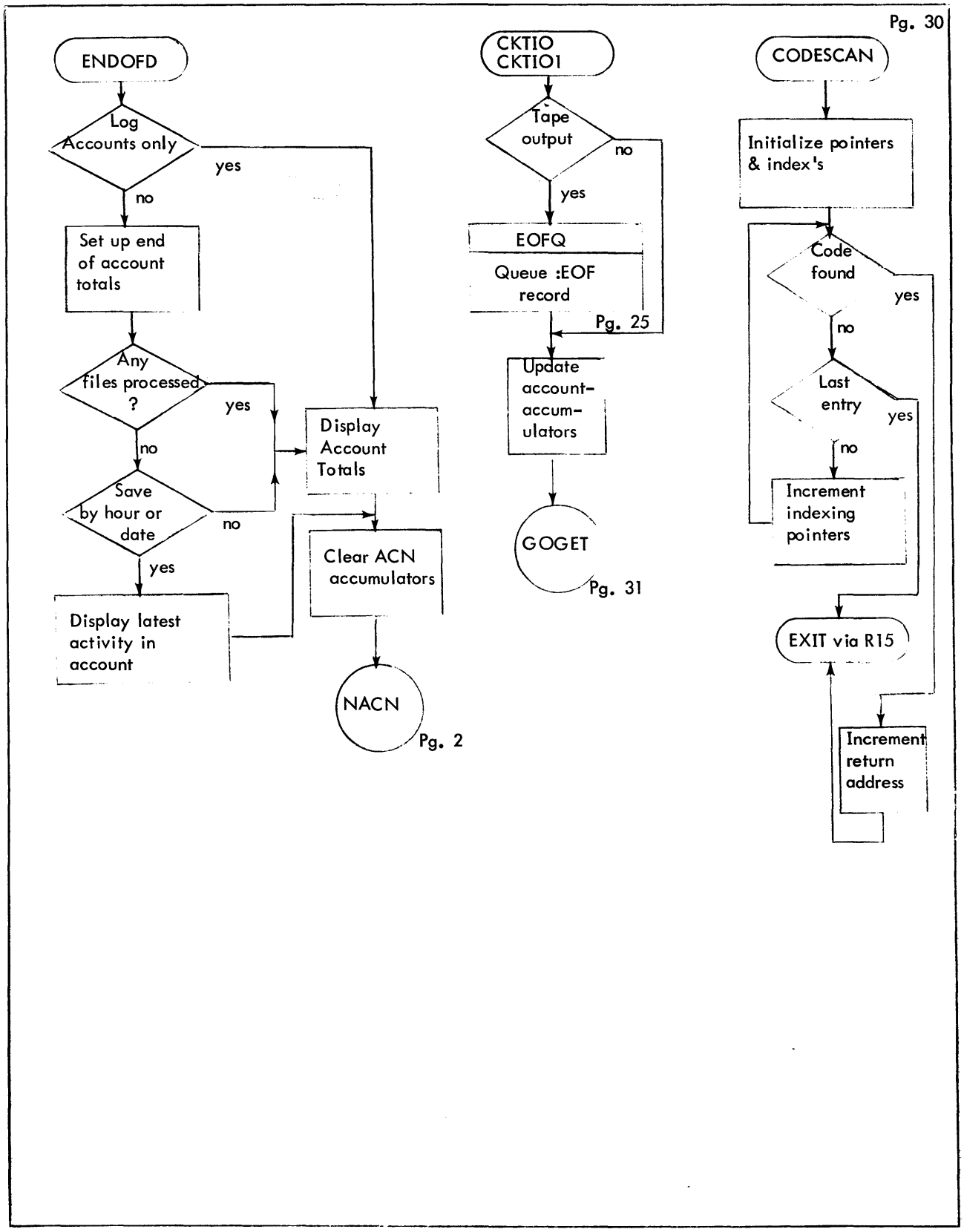


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

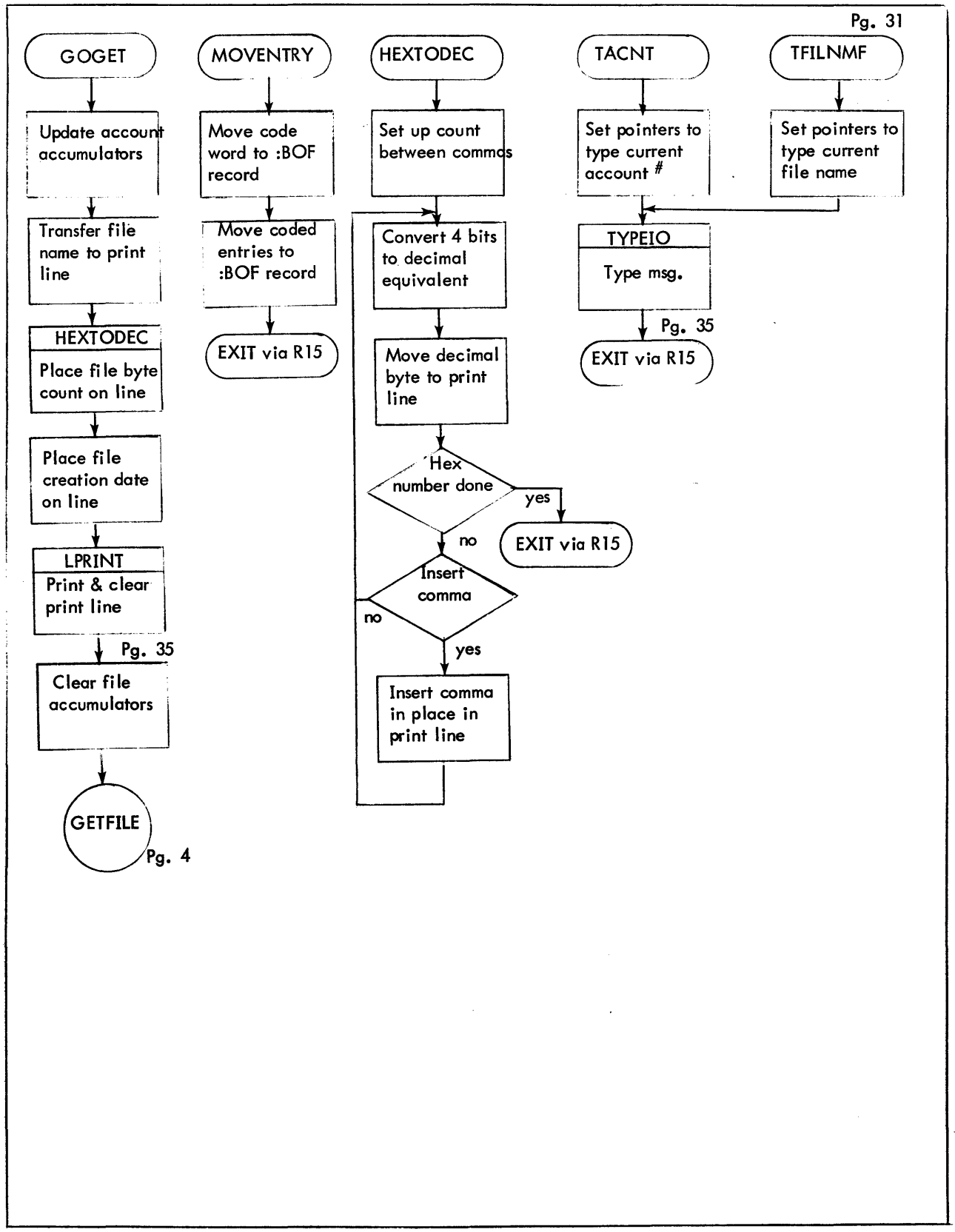


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

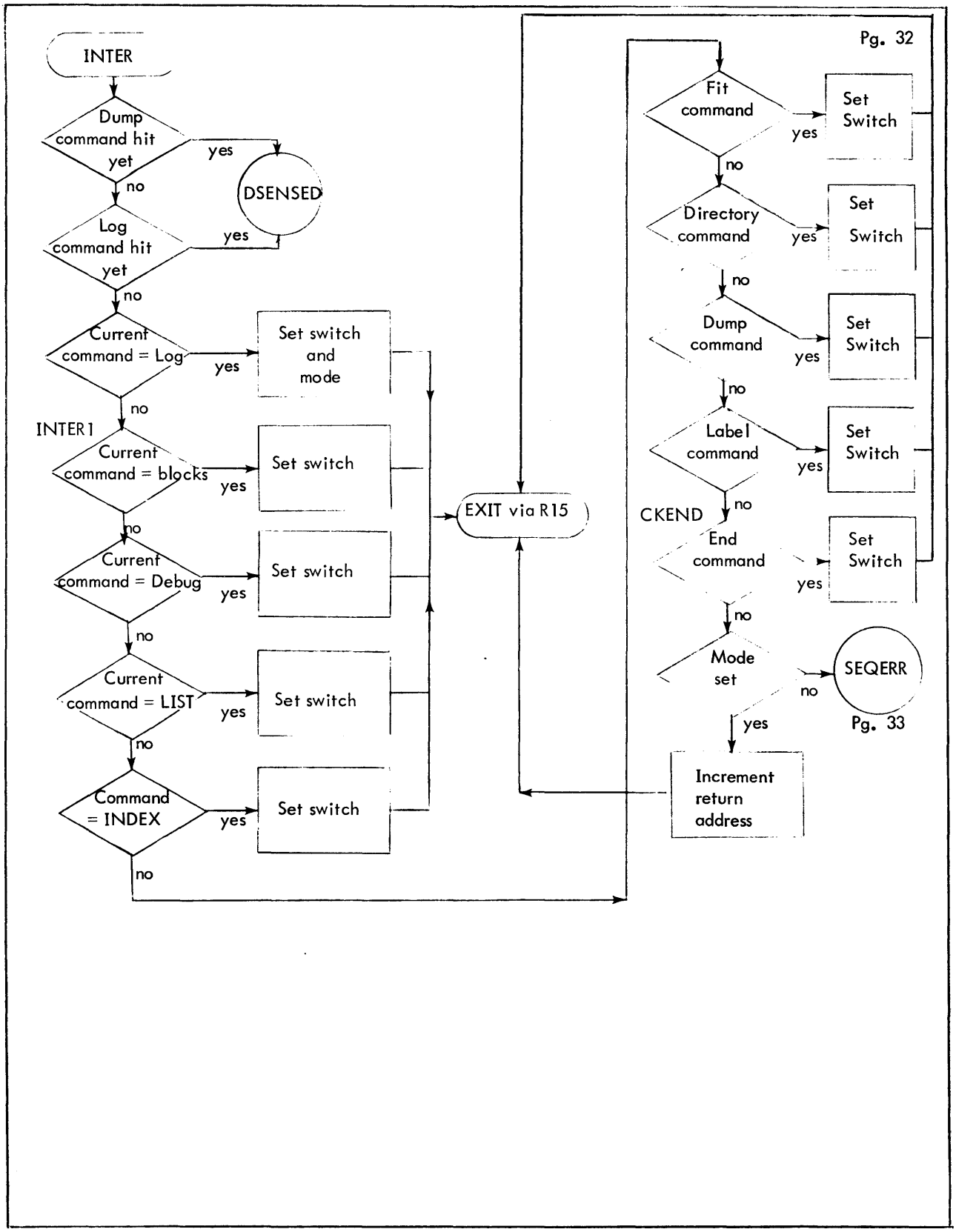


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

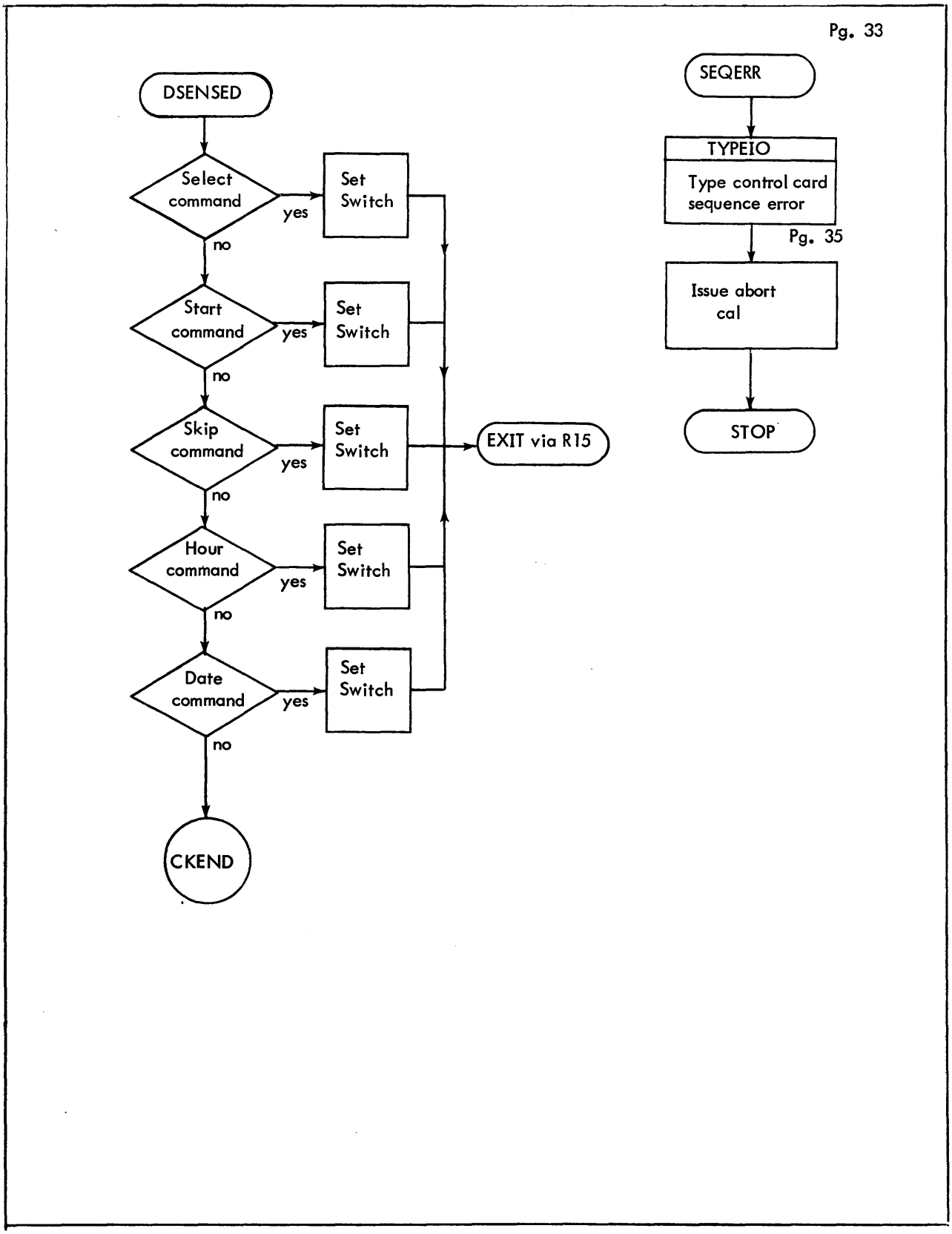


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

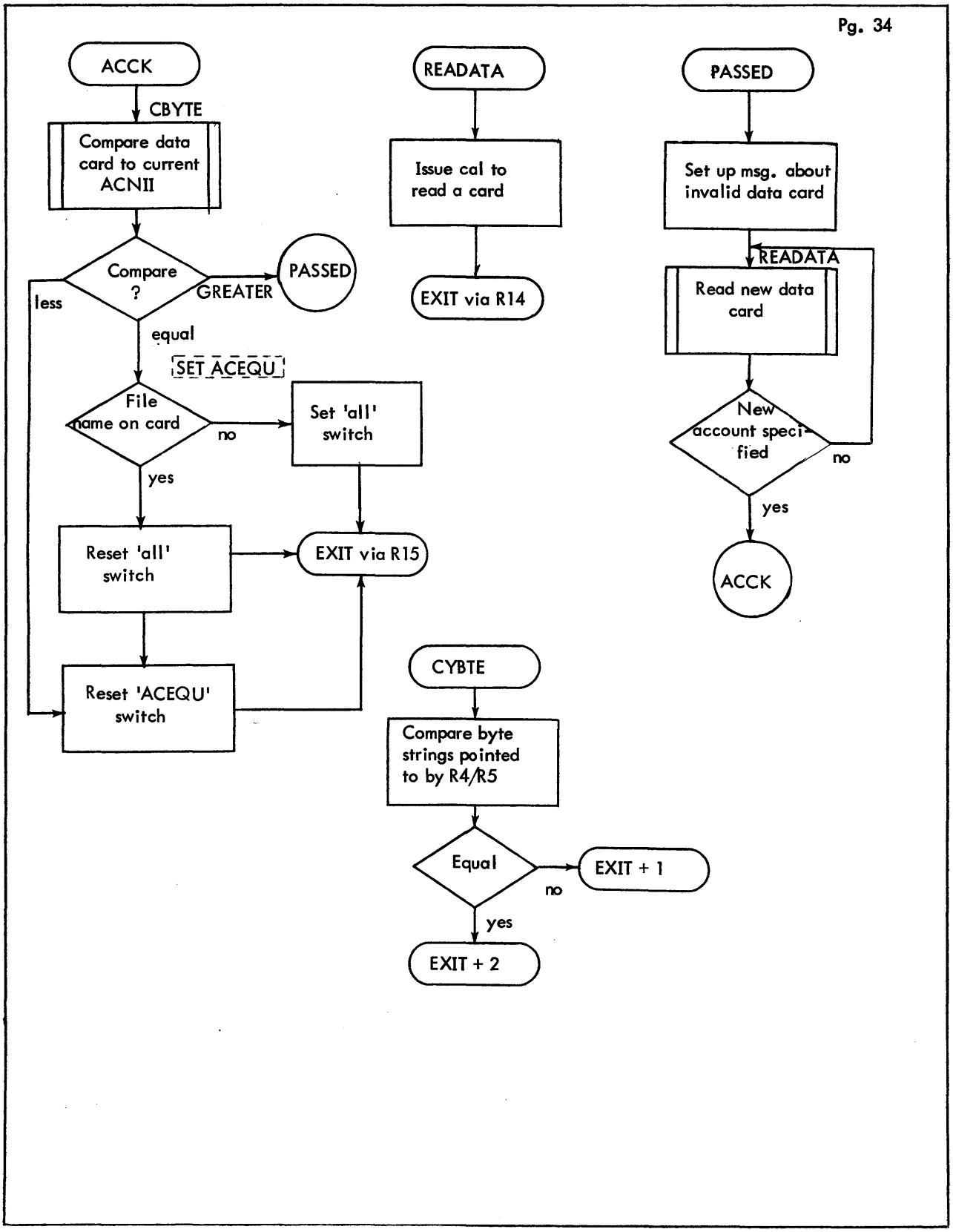


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

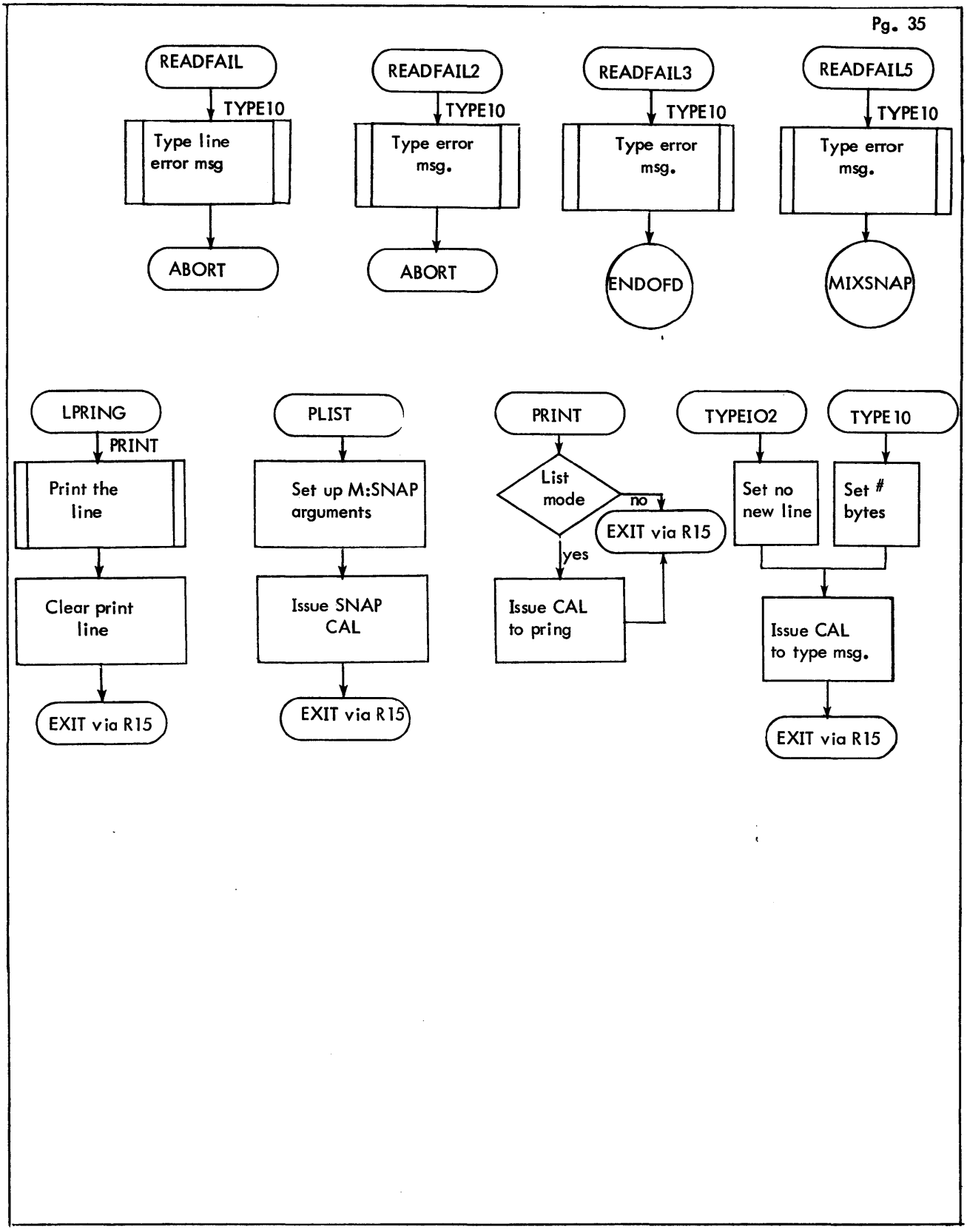


Figure 9-1. Detailed Flowchart of FILE SAVE (Cont.)

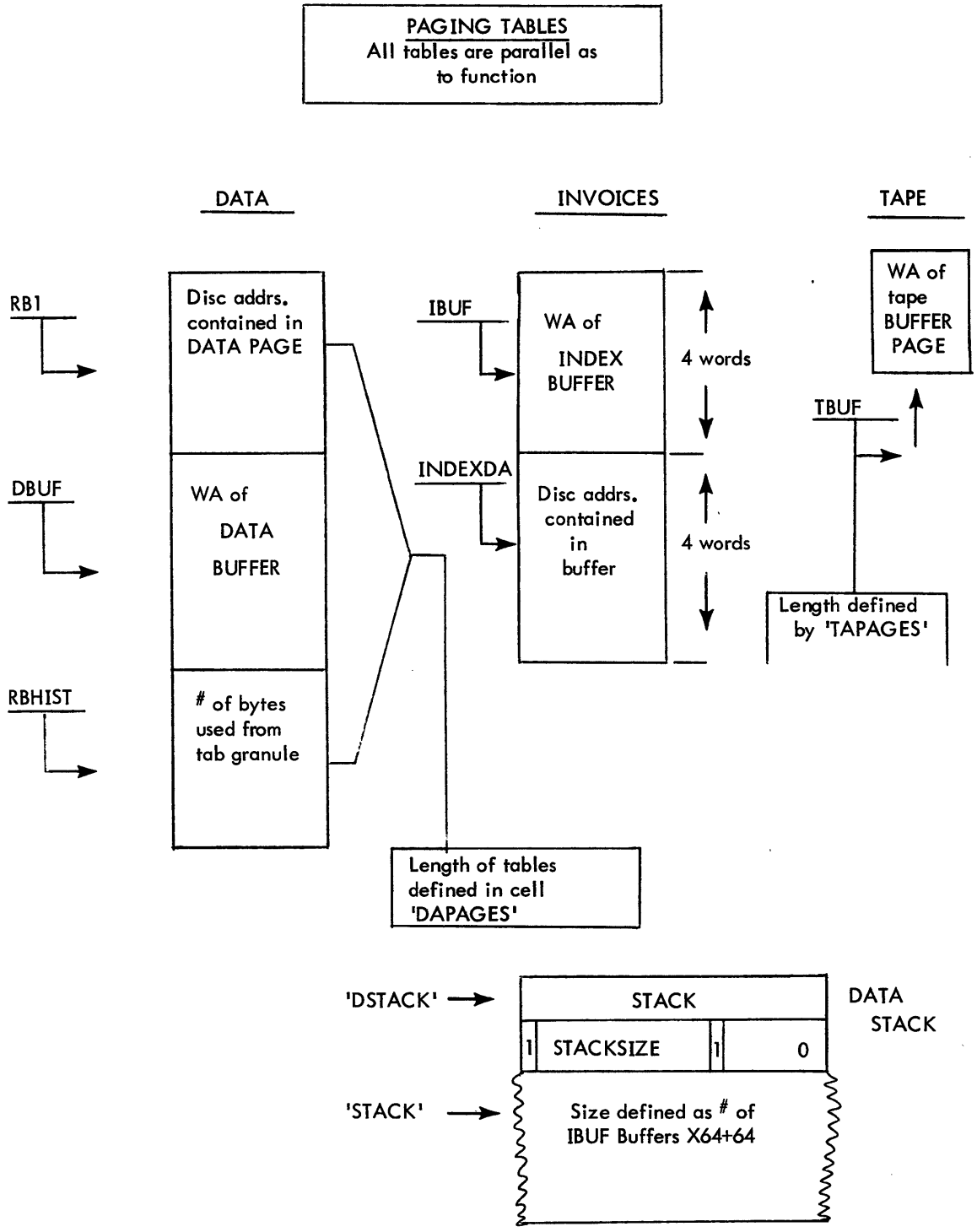


Figure 9-2. Formats of Paging Tables

9.6 TABLES

The following tables describe the parameters and positions of information internally. These tables are set to 'F00' BPM/BTM parameters, and any adjustment will change the operating characteristics of FAST SAVE.

Table 9-2. Table Key Lengths

NAME	EQU	MEANING
FDKEYL	41	File directory keys
ACDKEYL	21	Account directory keys
MIXKEYL	14	Total index sector key

Table 9-3. First Key Displacement in Sectors

NAME	EQU	MEANING
MIDISP	12	Master index displacement
ACDISP	12	Account directory displacement
FDDISP	12	File directory displacement

Table 9-4. Account Directory Displacement

NAME	EQU	MEANING
ADKBD	-6	Key and +1 to byte 3 of disc address
ADKFD	12	Key start to byte 0 of disc address

Table 9-5. File Directory Displacement

NAME	EQU	MEANING
FDKBD	-6	Key end +1 to byte 3 of disc address
FDKFD	32	Keystart to bytes 0 of disc address
FDBKEOF	-3	From key end +1 to EOF byte

Table 9-6. Master Index Displacement

NAME	EQU	MEANING
MIKBD	-6	Key end +1 to byte 3 of disc address

Table 9-7. I/O Queueing Function Codes

NAME	EQU	MEANING
IOPRI	X'FF'	Background I/O priority
READFC	0	GT and/or disc read
WRT	1	Write GT
BSR	4	Backspace record
FSR	5	Forward space record
BSF	6	Backspace file
FSF	7	Forward space file
WTM	8	Write tape mark
REWOL	9	Rewind on-line
REWOFL	10	Rewind off-line
GTSENS	11	GT Sense
READRCVR	14	Read GT with Recovery
READREV	16	GT Read Reverse
READISC	0	Read DC
WRTDISC	1	Write DC
CHKWRT	4	Write with check write
RETRY	20	Retry count

Table 9-8. Data Names and Definitions

DATA NAME	DEFINITION
ACBLINK	Backward link for current account directory sector.
ACBUF	Account directory sector buffer.
ACSIZE	Size of current account directory sector.
ACFLINK	Forward link for current account directory sector.
BLDISP	Displacement into data granule.
CURBUF	Current buffer address.
CURDBLK	Current data granule buffer address.
CURINDX	Current table index.
CURRB	Points to current disc address table.
CURRMIX	Current sector index.
DBUF	Table contains address of data granule buffers.
DCT1	Address of DCT1.
DSTACK	Data Stack containing granule addresses.
FDA	First index sector address.
FDBLINK	Backward link for current file directory.
FDBUF	Current file directory sector buffer.
FDFLINK	Forward link for current file directory.
FDSIZE	Size of current file directory sector.
FITBUF	File information table buffer.
FITDA	Disc address of current FIT.
GRANULEADR	Current data granule disc address.
IBUF	Table contains address of index buffers.
INDEXDA	Table contains disc address of all data granules referenced in current index buffer.
KEYDISP	Current key displacement into Index buffer.
MISIZE	Current sector NAV.
MIXBUF	Address of current MIX buffer.
NEWQ	Address of monitor NEWQ routine.
NEXACN	Next account number index.
NEXFILE	Next file directory index.
NEXTMIX	Next index to current sector.
ORG	X'09' entry from FIT for current file.
PBS	Previous block size for tape record.

Table 9-8. Data Names and Definitions (Cont.)

DATA NAME	DEFINITION
RBHIST	Table contains number of bytes used from each data granule.
RB1	Data disc address for data granules.
RSTORE	Number of granules in current RANDOM file.
RWS	Record size.
SYNFLAG	Specifies synonomous file if non-zero.
TBUF	Table contains address of tape buffers.

For formats of FILE DIRECTORY, ACCOUNT DIRECTORY, FIT, MIX, and ACNCFU see BPM Technical Manual FILE MANAGEMENT CHAPTER.

10.0 FILE ANALYZER (FANALYZE) PROCESSOR

10.1 FUNCTIONAL OVERVIEW

The File Analyzer (Fanalyze) processor is designed to provide fast reliability checks on a large BPM/BTM file management system. It verifies the granule pool bit maps, checks all linkages throughout the account and file directories, and master index blocks, and produces a log on the M:LO device with pertinent information concerning the file structure. Fanalyze provides a general picture of the file system, and through the use of various options, a more detailed examination can be made if errors are found.

10.2 INTERFACE

File Analyzer is able to operate under any release version of BPM/BTM since and including F00, provided it has these minimal hardware requirements:

1. Card Reader
2. Line Printer
3. Console Teletype
4. One or more Rads and or Packs

10.3 OPERATIONAL OVERVIEW

The File Analyzer program must run in the :SYS account as a privileged processor in order to gain access to the account directory, file directory, and the BPM/BTM queuing routines. To load Fanalyze, the standard file $::SPECIALS and FANROM should be selected from the BI tape, and then the LOCCT name Fanalyze can be loaded during Pass3 of a sysgen. If loading from BI deck, a stack size of at least 200 is required. The compressed for Fanalyze is on the CI tape as CN706124 which is RCVR2. An assembly option, at LINE 8 of CN706124, which is:

```
FANLZ   EQU     1       0 = RCVR2
```

Setting FANLZ equal to 1 will give you FANALYZE.

10.3.1 FILE ANALYZER OPTIONS

The File Analyze processor is called by specifying the processor name, and if desired, several options. If no options are specified, FANALYZE will default to the VERIFY option.

```
!FANALYZE  option, .....
```

where option may be anyone or more of the following:

VERIFY perform a validity check. If VERIFY is specified without the NOMAP option, a copy of the existing granule pool map is created in the background area. Than all account directories, file directories, file information tables, master index blocks, (level 0 through level n) and file data granules are allocated space on a master copy granule map in the background area. The monitor map is compared against the master map, and the two maps are snapped for evaluation if an error is detected. All linkages are checked and linkage failures are reported whenever found. The VERIFY option will only compare against existing maps. At program completion, it is up to the user to determine from the snaps taken, if RECOVER2 should be run.

NOMAP performs all the validity checks except for the granule pool maps.

ANLZ (account, filename) specifies how much output the analyzer will produce. No granule map work is attempted but the file information table is listed and all master index entries are listed. All linkage checks are performed and disk addresses are verified. Omission of filename will cause a listing for all files in the specified account.

DATA specifies that all data granules are to be format-listed on the M:LO device at the completion of an analyze pass on one file. This option is valid only when used with ANLZ.

NOLIST specifies that no format listing of file information is to be done. If errors are encountered, the account name and file name are logged before the snap is taken on the M:LO device.

AD specifies that the account directory sectors are to be printed as they are encountered. When used by itself, this option will list all account directory sectors.

FD specifies that the file directory sectors are to be printed as they are encountered. When used by itself, this option will list all file directory sectors.

10.3.2 Error Processing

All snaps (hex dumps, error messages, etc.) produced during FANALYZE processing, are preceded by a one-word title. These one-word titles are as follows.

ANLX-DMP All snaps with this title are due to granule map comparisons. The map words that failed comparisons are found in R2 and R3. The starting address may be found in R6 and R7. The index value to both maps is located in R1.

CALLSET Any snap with this title resulted from an I/O failure (i.e., a TYC of 8 or 9). The I/O calling sequence is placed into the registers as they were for the I/O call to NEWQ. The registers are as follows:

R0 end-action address (word address)

R1 end-action information

R12 In byte format, FC/PRI/NRT/DCTX, where FC is the I/O function code, PRI is the priority for this request, NRT is the number of recovery tries, and DCTX is the DCT1 index.

R13 address of intended buffer (byte address)

R14 Read/write size (number of bytes)

R15 Disk address (DCTX/sector number)

An additional snap is taken of the buffer that was due to receive the data. At this point the program error exits.

FDKEY A snap with this title is a snap of the current file directory key that contained an invalid file information table address.

OLINKER A snap with this title is a snap of the current index used at a BLINK failure when the ANLZ option is being used. No repair is made or attempted.

CHAINERR A snap with this title is a snap of the current pyramid (upper level index granule) block on a BLINK/FLINK error. No repair is made or attempted. This error snap is encountered only when the ANLZ option is running.

PYR-ERR A snap with this title is a snap of the current upper level index granule on BLINK failures. This will occur while the ANLZ option is attempting to find the first block on the highest level and finds a BLINK failure.

CHN-ERR A snap with this title is a snap of the current index block and a BLINK/FLINK failure if the DATA option encounters a link failure.

10.3.3 Error Messages

MESSAGE	DESCRIPTION
Above data address bad	An invalid master index data address location was found during the ANLZ run.
Account directory and file directory conflict	The granule pool map showed dual allocation of granules for the two directories.
Account directory bad, can't reconstruct HGP	A link failure occurred in the account directory
Account directory bad HGP reconstruction halted	VERIFY found a bad ACNCFU address or a nonzero BLINK in the first AD sector
Bad fit address	The file directory key contained an invalid FIT disk address. The address is printed to the right of the error message.
Bad free sector pool linkage	The free sector pool had bad linkages.
BLINK error	The backword link test failed on a new sector. The location and BLINK are printed beneath the message.
Data granule allocation error	A data granule location has previously been allocated.
Disk I/O read failure	The TYC returned at end-action indicated a read error.
Dual Allocation	Any error other than could be defined occurred (e.g., FANALYZE could not accurately determine who owns the other granule).

MESSAGE	DESCRIPTION
Dual allocation in free sector pool	The granule allocation indicated the FSP entry conflicted with other allocation.
File conflicts with account directories	The granule allocation for this file conflicted with the allocation for the account directory.
File conflicts with file directories	The granule allocation for this file conflicted with the allocation for a file directory.
File conflicts with previous files	The granule allocation for this file conflicted with the allocation for other files.
Filename does not correspond to FIT	The file directory entry name differed from the FIT name. The FD name is printed to the left of this message and the FIT is snapped.
Flink error	The forward address was invalid. The location and forward link are printed beneath this message.
Invalid character or option	A bad option or comma appeared on a processor control command.
Invalid data address	A key in a level 0 master index contained an invalid disk address. The address is printed to the right of the message.
Invalid free sector pool entry	A bad address was found in the free sector pool. The address is printed to the right of the message.
I/O call failure	A bad disk address has been passed to the disk I/O handler. The job aborts.
Link error in account directory	A BLINK/FLINK failure occurred in the account directory while the ANLZ option was running.
Link error in file directory	A BLINK/FLINK error occurred in the file directory while the ANLZ option was running.
Master index allocation error	A master index granule location has previously been allocated.
Name sequence error in file directory	The file directory names did not appear in ascending order. A dump is provided of that file directory sector.
Next level address invalid	The upper level index structure's next level address pointer was bad.
Pyramid blink error	A backward link (in a file's upper level index structure) has failed to verify.
Pyramid flink error	A forward link (in a file's upper level index structure) has failed to verify.
Pyramid location error	The TDA (from the file's FIT) had an invalid address.

Random file address error	The FIT pointed to an invalid data address for a random file.
Stack size invalid	The load module FANALYZE was loaded with a TSS of less than 200.

10.4 MODULE ANALYSIS

File Analyzer and RCVR2 have the same catalog number (CN706124), therefore refer to RCVR2 for the Module Analysis and Flowchart.

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