Retricted Materials of IBM Corporation
LY26-3926-1
File No. S370-32

Program Product

MVS/370 Utilities Logic

Data Facility Product 5665-295

Release 1.1



Second Edition (October 1983)

This edition replaces the previous edition (numbered LY26-3926-0) and makes that edition obsolete.

This edition applies to Release 1.1 of MVS/370 Data Facility Product, Program Product 5665-295, and to any subsequent releases until otherwise indicated in new editions or technical newsletters.

The changes for this edition are summarized under "Summary of Amendments" following the preface. Specific changes are indicated by a vertical bar to the left of the change. These bars will be deleted at any subsequent republication of the page affected. Editorial changes that have no technical significance are not noted.

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/370 and 4300 Processors Bibliography, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

Publications are not stocked at the address given below; requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California, U.S.A. 95150. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

This document contains restricted materials of International Business Machines Corporation. © Copyright International Business Machines Corporation 1972, 1973, 1975, 1983. All rights reserved.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

PREFACE

The purpose of this publication is to enable you to locate specific areas of the utility programs provided for MVS/370 Data Facility Product (MVS/370), and to relate those areas to the corresponding program listings.

The publication is divided into three sections, corresponding to the three types of utility programs: system utilities, data set utilities, and independent utilities. Each section contains descriptions of the programs of the corresponding type; these descriptions consist of text, flowcharts, and figures showing record and table formats.

The introduction provides a description of each utility program, and an appendix lists the modules of the utility programs.

Because each utility program is, in effect, a different subject, the numbering of the flowcharts for a given program begins at 01 and continues to as high a number as needed.

To use this publication effectively, you should be familiar with the following publications:

- MVS/370 Utilities, GC26-4065
- OS/VS2 Data Areas, SYB8-0606
- MVS/370 Data Management Macro Instructions, GC26-4057
- MVS/370 Data Management Services, GC26-4058
- MVS/370 DADSM and Common VTOC Access Facility Diagnosis Guide, SY26-3918
- MVS/370 DADSM Diagnosis Reference, SY26-3919
- OS/VS2 MVS System Programming Library: Supervisor Services and Macro Instructions, GC28-1114

The following are related reference publications, and are cited at various points within this book:

- IBM System/370 Principles of Operation, GA22-7000
- OS/VS Message Library: VS2 System Codes, GC38-1008
- MVS/370 Linkage Editor Logic, LY26-3921
- OS/VS2 MVS System Programming Library: Debugging Handbook, Volumes 1 through 3, GC28-1047, 1048, 1049
- Offline IBM 3800 Utility, SH20-9138
- Offline IBM 3800 Utility Logic, LY20-8058

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

SUMMARY OF AMENDMENTS

RELEASE 1.1, OCTOBER 1983

NEW PROGRAMMING SUPPORT

- The IEBCOPY chapter has been updated to include support for the ALTERMOD and COPYMOD functions of IEBCOPY.
- The IEHMOVE chapter has been updated to include support for RACF Release 1.5 Always Call.

OTHER CHANGES

The IEBIMAGE chapter has been updated with service changes.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CONTENTS

Part 1: Overview of the Utilities 1

Introduction 2
System Utility Programs 2
Auxiliary Parameters 2
Device Allocation and Volume Mounting (IEHMVSSF, IEHMVXSF, and IEHVOLMT) 3
Control Card Scanner (RDCDRT) 6
Data Set Utility Programs 7
Independent Utility Programs 8
Chart 01. IEHVOLMT—Volume Mounting Logic 9

Part 2: Functional Descriptions of the Utilities 11

ICAPRTBL Program 12
Program Flow 12
Chart 01. Loading the FCB and UCS Buffers 14

IEBCOMPR Program 15
Structure of the IEBCOMPR Program 15
Method of Operation 18
Invoking the IEBCOMPR Program 18
The Format of the Parameter List 18
Operating on the ddname Parameter List 20
Operating on the Page Number Parameter List 20
Reading and Analyzing the Control Statements 20
Control Statement Analysis 22
Directory Precompare for Partitioned Data Sets 31
Comparing Input Labels for Sequential Data Sets 36
Saving Input Labels 36
Comparing Records of Both Input Data Sets 40
Input Record Comparison 40
Messages and Headers 44
Termination Processing 45
Work Areas 45
IEBCOMPR Flowcharts 51
Chart 01. IEBCMAIN/IEBCQSAM Modules 52
Chart 02. IEBCULET Module 53

IEBCOPY Program 54
Invoking the IEBCOPY Program 54
Program Structure 54
Program Operation 55
Common Communication Area for the IEBCOPY Program 56
Major Tables Used by the IEBCOPY Program 56
SETAB Table 57
NNPTRTAB Table 60
CTLTAB Table 64
INDD Table 65
The 'Split-Table' Approach 65
Storage Allocation 66
Storage Allocation for Tables (I) 68
Storage Allocation for I/O Buffers (I) 68
Storage Allocation for Tables (II) 68
Storage Allocation for I/O Buffers (II) 71
Storage Allocation for I/O Buffers (III) 71
Storage Allocation for Tables (III) 72
Unloaded Data Set Tape Format 72
Descriptor Records 73
Directory Records 74
Unloaded Note List Record 74
Unloaded Member Data Records 75
Module Descriptions 77
Module IEBBAM—Buffer Allocation Routine 78
Module IEBCNVT—Load TTR and MBBCCHHR Conversion Routine 80
Module IEBCPMOD—ALTERMOD/COPYMOD Routine 82
Module IEBDRB—Reblock/Deblock Routine 85
Module IEBDRD—Read Routine 86

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Module IEBDSCPY—Main Flow Routine 90
Module IEBDSU—Directory Setup for an Unload Operation . 96
Module IEBDV1—Data Set Validation Routine 97
Module IEBDWR—Write Routine 98
Module IEBIOE—I/O Error Routine 104
Module IEBLDUL—Unload/Load Initialization Routine . . 104
Subroutine IEBMBT of Module IEBDSCPY 107
Module IEBMCM—Compress-in-Place Routine 108
Subroutine IEBMGD of Module IEBDSCPY 110
Subroutine IEBMGI of Module IEBDSCPY 111
Subroutine IEBMKO of Module IEBDSCPY 111
Module IEBRSAM—Read Routine for Unloaded Data Sets . 112
Module IEBSCN—Scan Routine 113
Module IEBVCT—SETAB Table Sorting Routine 116
Module IEBVDM—Directory Merge Routine 117
Module IEBVMS—Message Print Routine 123
Module IEBVTM—Termination Routine 124
Module IEBVTT—Routine to Sort CTLTAB Table 125
Module IEBWSAM—Write Routine for Unloaded Data Sets . 126
Module IEBWSU—Write Setup Routine 127
Module IGG019C8—PCI Routine 130
Module IGG019FT—SIO Appendage 131
Chart 01. Program Initialization 133
Chart 02. Opening the Input and Output Data Sets 134
Chart 03. Building the Input Directory Table 135
Chart 04. Scanning the Output Directory Table 136
Chart 05. Preparing for Copying, Unloading, or Loading a Member 137
Chart 06. Copying, Unloading, or Loading a Member with COPY 138
Chart 07. Copying or Altering a Member with COPYMOD or ALTERMOD (Part 1 of 5) 139
Chart 08. Copying or Altering a Member with COPYMOD or ALTERMOD (Part 2 of 5) 140
Chart 09. Copying or Altering a Member with COPYMOD or ALTERMOD (Part 3 of 5) 141
Chart 10. Copying or Altering a Member with COPYMOD or ALTERMOD (Part 4 of 5) 142
Chart 11. Copying or Altering a Member with COPYMOD or ALTERMOD (Part 5 of 5) 143
Chart 12. Directory Processing 144

IEBDG Program 145
Program Functions 145
Control Card Scanning 149
The Base Module (IEBDG) 150
Initialization 150
Opening Data Sets 150
Messages 151
Reading Control Cards 151
Base Module Card-Processing 152
The Cleanup Module (IEBDGCUP) 153
The FD Analysis Module (IEBFDANL) 154
FD Card Scanning 154
The FD Table Module (IEBFDTBL) 156
FD Pattern Construction 156
The Create Analysis Module (IEBCRANL) 158
Table Construction 158
Module Entries 158
Module Subroutines 159
Keyword Processing 160
The Create Module (IEBCREAT) 163
Output Record Modifications 163
Updating the FD Table 164
The Message Module (IEBDGMSG) 166
Service Aids 167
Tables and Work Areas Used by Modules of the Data Generator Program 176
Chart 01. Base Module (Part 1 of 3) 178
Chart 02. Base Module (Part 2 of 3) 179
Chart 03. Base Module (Part 3 of 3) 180
Chart 04. Cleanup Module, IEBDGCUP 181
Chart 05. FD-Analysis Module, IEBFDANL (Part 1 of 2) . . 182
Chart 06. FD-Analysis Module, IEBFDANL (Part 2 of 2) . . 183
Chart 07. FD-Table Module, IEBFDTBL (Part 1 of 2) . . . 184

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Chart 08. FD-Table Module, IEBFDTBL (Part 2 of 2) . . . 185
Chart 09. Create Analysis Module, IEBCRANL (Part 1 of 5) 186
Chart 10. Create Analysis Module, IEBCRANL (Part 2 of 5) 187
Chart 11. Create Analysis Module, IEBCRANL (Part 3 of 5) 188
Chart 12. Create Analysis Module, IEBCRANL (Part 4 of 5) 189
Chart 13. Create Analysis Module, IEBCRANL (Part 5 of 5) 190
Chart 14. Create Module, IEBCREAT (Part 1 of 2) 191
Chart 15. Create Module, IEBCREAT (Part 2 of 2) 192
Chart 16. Message Module, IEBDGMSG 193

IEBEDIT Program **194**
Program Structure 194
The Initializing Routine 194
The Main Routine 195
The Postprocessing Routine 196
IEBEDIT Subroutines 196
The Scan Subroutine 196
The Startjob Subroutine 200
The Cardtype Subroutine 200
The Update Subroutine 201
Chart 01. Main Routine (Part 1 of 2) 202
Chart 02. Main Routine (Part 2 of 2) 203

IEBGENER Program **204**
Using the IEBGENER Documentation 204
The IEBGENER Program 204
Program Structure 205
Method of Operation 207
Storing ddnames 207
Processing the Control Data Set 209
Processing the Input Data Set 209
Constructing the Message Data Set 209
Invoking IEBGENER 209
The Parameter Lists 210
Storing Alternate ddnames 211
Processing the Control Data Set 213
The Scan/Processor Interface 213
Storing the Parameters 213
Processing the Input Data Set 216
Buffering Techniques 217
Editing the Input Data Set 219
Constructing the Message Data Set 222
Tables, Switches, and Work Areas 222
Tables 225
Work Areas and Switches 225
Chart 01. Root Segment 236
Chart 02. Control Statement Analysis Routine 237
Chart 03. Main Processing Segment 238
Chart 04. Processing Nonspanned Record Data Sets 239
Chart 05. Processing Variable Spanned Record Data Sets . 240

IEBIMAGE Program **241**
Introduction 241
IEBIMAGE Module 241
IEBIMCHA Module 242
IEBIMCOP Module 242
IEBIMFCB Module 243
IEBIMGRA Module 243
IEBIMTAB Module 243
IEBIMWRT Module 243
IEBISCAN Module 243
IEBIMSGS Module 243
Method of Operation 244
IEBIMAGE Program Modules, Subroutines and Segments . . . 247
Program Organization 248
Chart 01. IEBIMAGE Module (Part 1 of 8): Main Flow . . . 249
Chart 02. IEBIMAGE Module (Part 2 of 8): PRINTQ, PRTCARD, BKSZCK, EODAD, WRITE, OPTIONX 250
Chart 03. IEBIMAGE Module (Part 3 of 8): NAME 251
Chart 04. IEBIMAGE Module (Part 4 of 8): INCLUDE, INVOP 252
Chart 05. IEBIMAGE Module (Part 5 of 8): ABEND, SYNAD, BKSZFIX 253
Chart 06. IEBIMAGE Module (Part 6 of 8): READONLY, READLIB, BLDL 254

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Chart 07. IEBIMAGE Module (Part 7 of 8): CONVRTER . . . 255
Chart 08. IEBIMAGE Module (Part 8 of 8): CONVERTX, CONVERT 256
Chart 09. IEBIMCHA Module (Part 1 of 8): Main Flow . . . 257
Chart 10. IEBIMCHA Module (Part 2 of 8): CHARSET 258
Chart 11. IEBIMCHA Module (Part 3 of 8): CHARINCL, SETCREF 259
Chart 12. IEBIMCHA Module (Part 4 of 8): CHARBLD 260
Chart 13. IEBIMCHA Module (Part 5 of 8): BLDCHAR 261
Chart 14. IEBIMCHA Module (Part 6 of 8): BLDCHAR3 . . . 262
Chart 15. IEBIMCHA Module (Part 7 of 8): CHARCHK, CHARPRT 263
Chart 16. IEBIMCHA Module (Part 8 of 8): SEGMSG, CHARCHK3 264
Chart 17. IEBIMCOP Module (Part 1 of 4): Main Flow, COPYMOD, COPYPRT 265
Chart 18. IEBIMCOP Module (Part 2 of 4): COPCHK, COPYINCL 266
Chart 19. IEBIMCOP Module (Part 3 of 4): COPYMSG, OVERRUN 267
Chart 20. IEBIMCOP Module (Part 4 of 4): SETTEXT 268
Chart 21. IEBIMFCB Module (Part 1 of 3): Main Flow, FCB 269
Chart 22. IEBIMFCB Module (Part 2 of 3): FCBBLD, FCBCHECK 270
Chart 23. IEBIMFCB Module (Part 3 of 3): FCBPRT, FCBMARG, FCBSTAT 271
Chart 24. IEBIMGRA Module (Part 1 of 3): Main Flow, GRAPHIC, SETREF 272
Chart 25. IEBIMGRA Module (Part 2 of 3): GRFCHK, GCMPRT 273
Chart 26. IEBIMGRA Module (Part 3 of 3): GRFCHK3, SEGMSG 274
Chart 27. IEBIMTAB Module (Part 1 of 4): Main Flow, TABLE 275
Chart 28. IEBIMTAB Module (Part 2 of 4): TABCHK 276
Chart 29. IEBIMTAB Module (Part 3 of 4): TABLPRT 277
Chart 30. IEBIMTAB Module (Part 4 of 4): TABLINIT, SETLOC 278
Chart 31. IEBIMWRT Module (Part 1 of 2): Main Flow, WRBLDL 279
Chart 32. IEBIMWRT Module (Part 2 of 2): WRBLOCK, WRTEXT, STOWMSG 280
Chart 33. IEBIMSGS Module 281
Chart 34. IEBISCAN Module (Part 1 of 4): Main Flow, GETCARDN 282
Chart 35. IEBISCAN Module (Part 2 of 4): OPSCAN, KEYWORD, REPSCAN 283
Chart 36. IEBISCAN Module (Part 3 of 4): KWDEXCEP . . . 284
Chart 37. IEBISCAN Module (Part 4 of 4): SAMPKWD, SCAN, FINDPARM 285
Data Areas 286
IEBIBLKS—IEBIMAGE Work Area 286
KEYTAB—IEBIMAGE Keyword Table (One Entry) 291
OPTAB—IEBIMAGE Operation Table (One Entry) 292
Load Module Records 293
CESD (Composite External Symbol Dictionary) 293
Identification Record 293
HMASPZAP Data 294
Linkage Editor Data (IEBIMAGE Acts as a Linkage Editor) 294
Translator Data (IEBIMAGE Acts as a Language Translator) 295
Control Record 295

IEBISAM Program **296**
Initializing IEBISAM 296
Copying an Indexed Sequential Data Set 296
Unloading an Indexed Sequential Data Set 298
Obtaining Indexed Sequential Records 298
Building the Output Data Set 298
Loading an Indexed Sequential Data Set 300
Printing Logical Records of an Indexed Sequential Data Set 301
Terminating the IEBISAM Program 301
Chart 01. Overall Flow 303
Chart 02. Initialize IEBISAM Program 304
Chart 03. Copy Indexed Sequential Records (IEBISC) . . . 305
Chart 04. Retrieve Indexed Sequential Records (IEBISU) . 306
Chart 05. Unload Physical Sequential Records (IEBISSO) . 307
Chart 06. Reconstruct Indexed Sequential Records (IEBISL) 308
Chart 07. Retrieve Physical Sequential Records (IEBISSI) 309
Chart 08. Print Logical Records (IEBISPL) 310
Chart 09. Terminate IEBISAM Program (IEBISF) 311

IEBPTPCH Program **312**
Using the IEBPTPCH Documentation 312
The IEBPTPCH Program 312
Program Structure 313

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Method of Operation 314
Storing Input Parameters—(1) 314
Processing Control Statements—(2) 314
Sorting Member Names—(3) 314
Initializing the Main Processor—(4) 314
Processing SYSUT1 Records—(5) 317
Writing Messages—(6) 317
Invoking IEBPTPCH 317
Storing Input Parameters—(1) 317
The Parameter Lists 319
Storing the Parameters 319
Processing Control Statements—(2) 319
The Scan/Analysis Interface 320
Constructing Tables 323
Storing Control Statement Parameters 323
Sorting Member Names—(3) 324
Initializing the Main Processor—(4) 326
Obtaining Processing Parameters 328
Locating Members 329
Processing SYSUT1 Records—(5) 329
Processing Formatted Records (PREFORM Specified) . . . 331
Processing Unformatted Records (PREFORM Not Specified) 331
Identifying Records 332
Editing to Standard Output Format 333
Editing to User Specifications 333
Headers and Sequence Numbers 336
Writing Messages—(6) 336
Tables and Work Areas 338
IEBPTPCH Flowcharts 344
Chart 01. IEBPPUN1 344
Chart 02. IEBPPAL1 345
Chart 03. IEBPPCH1 (Part 1 of 3) 346
Chart 04. IEBPPCH1 (Part 2 of 3) 347
Chart 05. IEBPPCH1 (Part 3 of 3) 348

IEBTCRIN Program **349**
Program Structure 349
The Driver Module (IEBTCRIN)—Chart 01 349
The Common Communication Region 350
IEBTCRIN Organization 353
The Message Module (IEBTCR02) 355
The Job File Control Block Analyzer Module (IEBTCR03) 355
The Control Card Analyzer Module (IEBTCR04)—Chart 03 357
The Read/Edit Module (IEBTCR05)—Charts 04 Through 13 361
Program Switches 366
System Control Block Usage 368
Error Description for MTDI With Editing 369
Character Code Dependency 369
Chart 01. Driver First Level Routine 370
Chart 02. Job File Control Block (JFCB) Analyzer, IEBTCR03 371
Chart 03. Control Card Analyzer—First Level Routine, IEBTCR04 372
Chart 04. Read/Edit Entry Routine of IEBTCR05 373
Chart 05. Read/Edit Read Routine of IEBTCR05 (Part 1 of 2) 374
Chart 06. Read/Edit Read Routine of IEBTCR05 (Part 2 of 2) 375
Chart 07. Read/Edit Scan Routine of IEBTCR05 (Part 1 of 2) 376
Chart 08. Read/Edit Scan Routine of IEBTCR05 (Part 2 of 2) 377
Chart 09. Read/Edit Move Routine of IEBTCR05 378
Chart 10. Read/Edit Output Processor Routine of IEBTCR05 (Part 1 of 3) 379
Chart 11. Read/Edit Output Processor Routine of IEBTCR05 (Part 2 of 3) 380
Chart 12. Read/Edit Output Processor Routine of IEBTCR05 (Part 3 of 3) 381
Chart 13. Read/Edit End of Utility Processor Routine of IEBTCR05 382

IEBUPDTE Program **383**
Program Structure 383
The Root Segment 383
The Control Card Analyzer Segment 384
Initialization Module 384
Program Flow 385
Processor Data Flow 385

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Chart 01. Overall Flow (Part 1 of 3) 389
Chart 02. Overall Flow (Part 2 of 3) 390
Chart 03. Overall Flow (Part 3 of 3) 391

IEHATLAS Program **392**
The IEHATLAS Utility Program 395
Program Flow 395
The Alternate Track Location Assignment (ATLAS) Routine 401
Load Module IGC0008F 402
Load Module IGG0860A 403
Load Module IGG0860B 405
Load Module IGG0860C 408
Load Module IGG0860D 410
Load Module IGG086AE 415
Chart 01. Overall Logic Flow 418
Chart 02. Utility Logic Flow 419
Chart 03. ATLAS Modules IGC0008F and IGG0860A 420
Chart 04. ATLAS Module IGG0860B 421
Chart 05. ATLAS Module IGG0860C 422
Chart 06. ATLAS Module IGG0860D 423
Chart 07. ATLAS Module IGG086AE 424

IEHINITT Program **425**
Program Flow 425
Program Structure 425
The Control Module (IEHINITT) 425
The SVC 39 Routine (IGC0003I) 426
The Control Statement Scan Routine (IEHSCAN) 428
The Message Writer (IEHPRNT) 428
Chart 01. Control Routine (Part 1 of 2) 429
Chart 02. Control Routine (Part 2 of 2) 430
Chart 03. Scan Routine (Part 1 of 6) 431
Chart 04. Scan Routine (Part 2 of 6) 432
Chart 05. Scan Routine (Part 3 of 6) 433
Chart 06. Scan Routine (Part 4 of 6) 434
Chart 07. Scan Routine (Part 5 of 6) 435
Chart 08. Scan Routine (Part 6 of 6) 436
Chart 09. Print Routine 437
Chart 10. SVC 39 Tape Label Routine (Part 1 of 2) . . . 438
Chart 11. SVC 39 Tape Label Routine (Part 2 of 2) . . . 439

IEHLIST Program **440**
Program Structure 440
Chart 01. Listing System Control Data (Part 1 of 2) . . 446
Chart 02. Listing System Control Data (Part 2 of 2) . . 447

IEHMOVE Program **448**
Overall Flow 448
Program Structure 450
Program Setup (IEHMOVE, IEHMVXSE, IEHMVXSF) 450
Request Setup (IEHMVEST, IEHMVESJ, IEHMVESS) 450
Message Writing (IEHMVESA, IEHMVESU) 451
DSGROUP Setup (IEHMVESI, IEHMVESC, IEHMVESH) 451
Data Set and Volume Setup (IEHMVESZ, IEHMVXSF, IEHMVESX, IEHMVESV, IEHMVESY) 452
PDS Subroutines (IEHMVESR, IEHMVETG, IEHMVXSF) 454
Copying, Unloading, and Loading 455
DSGROUP Wrapup (IEHMVESH, IEHMVETA) 457
Data Set Wrapup (IEHMVESN, IEHMVESO, IEHMVESP, IEHMVESQ, IEHMVESK) 458
Communication Area (IEHMVV) 458
IEHMOVE Work Data-Set Record Formats 459
Obtaining Space for a Work Data Set 460
Releasing Space Used by a Work Data Set 460
Chart 01. Overall Logic 461
Chart 02. VOLUME Logic 462
Chart 03. PDS Logic 463
Chart 04. DSNAME Logic 464
Chart 05. DSGROUP Logic 465
Chart 06. CATALOG Logic for CVOL Catalogs 466

IEHPROGM Program **467**
Program Structure 467
Control Load Modules 468

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

The Root (IEHPROG1) 468
The Parameter List Builder (IEHPROG6, IEHPROGB, IEHPROGC) 469
The Volume Mounter (IEHPROG5, IEHVOLMT) 471
The SVC Return Analyzer (IEHPROG2) 472
Subordinate Load Modules 473
The Auxiliary Parameter Analyzer (IEHPROG4) 473
The Message Writer (IEHPROG3) 473
Volume Lookup (IEHPROGA) 473
Program Flow 474
Phase 1 474
Phase 2 474
Phase 3 479
Chart 01. Phase 1—Modifying System Control Data 480
Chart 02. Phase 2—Modifying System Control Data 481
Chart 03. Phase 3—Modifying System Control Data (Part 1 of 2) 482
Chart 04. Phase 3—Modifying System Control Data (Part 2 of 2) 483

IFHSTATR Program 484
Program Flow 484
Chart 01. Program Flow 486

Offline IBM 3800 Utility 487

Appendix A. Modules of Utility Programs 488
ICAPRTBL 488
IEBCOMPR 488
IEBCOPY 488
IEBDG 489
IEBEDIT 490
IEBGENER 490
IEBIMAGE 492
IEBISAM 492
IEBPTPCH 493
IEBUPDTE 493
IEHATLAS 494
IEHINITT 494
IEHLIST 494
IEHMOVE 495
IEHPROGM 497
IFHSTATR 497

Appendix B. User Label Processing and Totaling 498
Parameter List 498
Parameter List Modifications 500
Return Codes 500
Return Code Modifications 501

Appendix C. DSCB Formats for the IEHLIST Program . . . 505

Appendix D. Module-Microfiche Cross-Reference 508

Index 513

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

FIGURES

1. Auxiliary Parameter Formats for IEHATLAS, IEHINITT, IEHLIST, IEHMOVE, and IEHPROGM 3
2. Internal Table Header 4
3. Volume Mounting Request 5
4. Internal Table Maintained by IEHVOLMT 5
5. Structure of the IEBCOMPR Program 16
6. Processing and Data Flow in IEBCOMPR 17
7. Parameter List Passed by User's Program 19
8. Alternate ddname Parameter List Established by User's Program 20
9. Moving ddname List Data 21
10. Moving Header List Data 22
11. Processing Page Number (by IEBCROOT) 23
12. Processing and Information Interchange for Statement Analysis 24
13. Parameter List Passed to IEBCCS02 25
14. Control Statement Field Delimiters 26
15. Common Work Area Passed between IEBCANAL and IEBCCS02 27
16. Passing a Parameter List from IEBCCS02 28
17. Switches Set by IEBCCS02, and Their Significance . . 29
18. Handling User Routine Names 30
19. Moving ddnames and a Parameter List 33
20. Obtaining Count of True Names on a Partitioned Data Set 34
21. Building the Member Name Table and TTRN Tables . . . 35
22. Input Label Saving and User Exits 37
23. Moving User Label Exit Routines and DCB Information . 38
24. Comparison of Input Data Set Header or Trailer Labels (Sequential Data Sets Only) 39
25. Access Methods Used to Obtain Records for Comparison 40
26. Locating and Reading Members on PDS via INITBSAM . . 41
27. QSAM Processing of Records to Be Compared 42
28. Printing Messages and Headers 46
29. Conditions for Termination of IEBCOMPR Utility Program 47
30. Common Communication Region for IEBCOMPR 48
31. Constants and Work Area for IEBCANAL 49
32. Work Area Contents for IEBCMAIN 50
33. Communication Area DSECT (IEBMCA) for IEBCOPY 58
34. SETAB Table Entry After Initial Scan of SELECT Statement by Module IEBSCN 59
35. To Show Construction of Tables Used by the IEBCOPY Program 61
36. Merging Selected Input Members Onto an Output Data Set 63
37. Split-Table Representation of SETAB/CTLTAB Table . . 66
38. Storage Allocation for IEBCOPY Program 67
39. Spilling Input Directory Entries from Main Storage . 70
40. Records Present in an Unloaded Partitioned Data Set . 72
41. Contents of the COPYR1 Descriptor Record 73
42. Contents of the COPYR2 Descriptor Record 73
43. Example of Directory Record Format for the Unloaded Partitioned Data Set 74
44. Example of Format of an Unloaded Note List Record . . 75
45. Example of the Format of Unloaded Member Data Records 76
46. Example of the Format of a Member Data Block 76
47. Direct Access End-of-File Block Format for an Unloaded Member 77
48. IEBCOPY Program Module Interface Relationship 78
49. Generalized Buffer Allocation for IEBCOPY 81
50. Block Sizes of Data Types Read by Module IEBDRB . . . 87
51. Channel Program for PCI-Read of Data 88
52. Channel Program to Read Data when Track Overflow is Specified 90
53. Channel Program for Writing Member Data or Merged Directory Blocks 100
54. Member Data Write Channel Program 102
55. Directory Write Channel Program 103
56. Channel Program for Erasing a Track 103
57. Data Flow in Module IEBMCM 109
58. Example of Sorting SETAB Table by Module IEBVCT . . 118

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

59. Data Movement for Directory Merging 120
60. Directory Block Relationships for Merge Processing 123
61. Examples of IEBCOPY Termination Conditions and Processing 125
62. Validity Checking Performed by Module IEBWSU . . . 129
63. Conditions Under which Module IEBWSU Passes Control 129
64. Summary of Program Operation 132
65. Information Summary and Overall Flow of Data Generation Program 146
66. Storage Area Obtained by Base Module for Current DCB 152
67. FD Table Constructed by FD Analysis Module and FD Table Module 155
68. FD Control Card Keyword Parameter Processing, and Default Values Assigned, If Required 156
69. Create Table Constructed by Create Analysis Module 159
70. FD Address Table Constructed by Create Analysis Module 161
71. User Exit Name Table Constructed by Create Analysis Module 162
72. Picture Table Constructed by Create Analysis Module 163
73. Values of Increment-Restore Fields in the FD Table 165
74. Changes Made to FD Table Values as Create Module Builds Output Records 167
75. Common Communication Area 169
76. Data Control Block 173
77. Defined Constants for Modules of the Data Generator Program 174
78. Equated Symbols for Modules of the Data Generator Program 175
79. Data Generator Modules Information Tables and Area 176
80. Module Inputs and Outputs 177
81. EXEC Statement Include/Exclude Processing 197
82. Scan Routine Operation Code Table Entry 198
83. Scan Routine Parameter Table Entry 198
84. Scan Routine Fixed Operand Table Entry 199
85. Overlay Structure 206
86. The IEBGENER Program 208
87. Input Parameter Lists 211
88. Storing Alternate ddnames 212
89. Scan/Processor Interface 214
90. Processing the Control Data Set 215
91. Selecting a Buffering Technique 217
92. Input and Output Data Set Buffering 218
93. Editing the Input Data Set 220
94. Constructing the Message Data Set 223
95. Member Description Table (MDT) 225
96. Record Description Table (RDT) 226
97. The Edit Control Block (EDCB) 226
98. Exit List 227
99. CWORK Communication Area 227
100. Constants and Switches Area for Module IEBGSCAN . 228
101. Constants and Switches Area for Module IEBGENR3 . 232
102. Constants and Switches Area for Module IEBGENS3 . 233
103. Constants and Switches Area for Module IEBGEN03 . 235
104. Overview—IEBIMAGE Operation 244
105. Module Directory, Summary, and Chart IDs for IEBISAM Program 297
106. Unloading and Loading an Indexed Sequential Data Set 299
107. Program Structure 313
108. The IEBPTPCH Program 315
109. Storing Input Parameters 318
110. IEBPPAL1/IEBCCS02 Interface 320
111. Storing Control Statement Parameters 321
112. Reading, Counting, and Extracting Directory Records 325
113. IEBPPCH1 Member Entry Sort 326
114. Initializing the Main Processing Routine (IEBPPCH1) 327
115. Locating a Member 328
116. PREFORM Processing 330
117. Identifying Records 332
118. Editing to Standard Format 334
119. Editing to User Specifications 335
120. Headers and Sequence Numbers 337
121. Writing Messages 338
122. DTABLE 339
123. Group Control Block and Field Control Block . . . 342

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

124. Member Name Table (MNT) 343
125. IEBTCRIN Overlay Structure 350
126. Control Card Option Table (OPTABLE) 351
127. Module Address Table (ADDRTAB) 351
128. Name Table (NAMETAB) 351
129. Message Pointer Table (MSGPTRS) 352
130. Structural Flow of the Driver 354
131. Message Module Tables 356
132. DCB Parameter Defaults (MTDI with Editing) 357
133. DCB Parameter Defaults (MTST and MIDI without Editing) 357
134. Structural Flow of the Control Card Analyzer . . . 359
135. Control Card Delimiters 360
136. Structural Flow of Read/Edit Module 362
137. Record Destination as Determined in the Output Processor Routine 366
138. Error Description Word 367
139. Overall Flow 385
140. Principle of Operation 386
141. Overview of IEHATLAS Utility Program and ATLAS Routine 394
142. Main Work Area for IEHATLAS Utility Program . . . 396
143. Channel Program to Write Replacement Record . . . 400
144. Parameter List for ATLAS (SVC 86) 401
145. Return Parameter Register 0 from ATLAS (SVC 86) . 402
146. Track-Descriptor Field 404
147. Read-Count Channel Program Modification for a Single Error-Causing Read Count Command 407
148. Read-Count Channel Program Modification for Two Adjacent Error-Causing Read Count Commands 407
149. Representation of Read-Count Channel Program and Count Fields in Storage 411
150. Example of Channel Programs Used by Module IGG0860D to Transfer Information from Error Track to Alternate Track 413
151. Return Codes from ATLAS 416
152. Channel Programs Developed by IEHATLAS Utility Program and ATLAS Routine 417
153. Writing Tape Labels 426
154. The Structure of the Program 441
155. The Structural Flow of the IEHLIST Program 442
156. The Design of the IEHMOVE Program 449
157. SYSUT1 Record Format (for a PDS Request Only) . . 451
158. SYSUT2 Record Format (for a PDS Request Only) . . 451
159. SYSUT1 and SYSUT2 Record Formats for DSGROUP; SYSUT1 Record Formats for CATALOG 453
160. SYSUT3 Record Format 455
161. Load Module Grouping for Copying, Unloading, and Loading 455
162. Label Save Area Pointers 459
163. The General Design of the IEHPROGM Program 468
164. The Structure of the IEHPROGM Program (Each block represents one control section.) 469
165. The Structural Flow of IEHPROGM Program (Each block represents one load module.) 470
166. IEHECHAR, the Communication Table for FNDECODE, KODECODE, IEHESCAN, and IEHETLU 475
167. The CATALOG Routing Table 475
168. Parameter Lists Built by IEHPROGM for Data Management Routines 477
169. Parameter Lists Built by IEHPROGM for Data Management Routines (for SVC 98) 478
170. The Return-Indexing Entry (for the Catalog SVC) of the Catalog Operation 479
171. ESV Record Format 485
172. General Logic of Utility Program with User Label Processing Routine Exits 499
173. Parameter List Passed to User-Label Exit Routine . 500
174. Return Code Modification for IEBCOMPR Program . . 504
175. Description of Fields in Data Set Control Block Formats 1 to 6 on Printout by IEHLIST Program . . 506

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

PART 1: OVERVIEW OF THE UTILITIES

The following part describes the different categories of utility programs available and gives a brief description of each utility.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

INTRODUCTION

MVS/370 Data Facility Product (MVS/370) provides utility programs that perform basic data management operations. These programs are grouped into three categories:

- System utilities
- Data set utilities
- Independent utilities

SYSTEM UTILITY PROGRAMS

System utility programs are executed under the operating system in the problem program mode. These utilities handle data associated with the structure of the operating system.

- **IEHATLAS** dynamically attempts to recover data from a track that is defective because of either a data check or a missing address marker condition. The program does this by either replacing the data in error or by assigning an alternate track and transferring data from the defective track to the assigned alternate track.
- **IEHINITT** creates volume labels on magnetic tape.
- **IEHLIST** lists a volume table of contents and the directories of partitioned data sets. IEHLIST also lists an OS CVOL or a portion thereof.
- **IEHMOVE** duplicates collections of data sets to provide backup copies or to rearrange existing ones.
- **IEHPROGM** modifies control data in volume and catalog structures and in the PASSWORD data set.
- **IFHSTATR** selects, formats, and writes information about tape errors from the IFASMFDP tape.
- **Offline IBM 3800 Utility** (when used in conjunction with the IBM Tape-to-Printing Subsystem Feature) allows for an IBM 3800 Printing Subsystem, with all its functions and features, to be used offline in a stand-alone tape-to-printer operation.

The system utility programs, IEHINITT, IEHLIST, IEHMOVE, and IEHPROGM, use the queued sequential access method (QSAM) to read and write the SYSIN and/or SYSPRINT data sets or their (user-designated) equivalents. For these programs, SYSIN and SYSPRINT data sets also may have a blocking factor that is other than one.

AUXILIARY PARAMETERS

The IEHATLAS, IEHINITT, IEHLIST, IEHMOVE, IEHPROGM, and IFHSTATR utilities may be invoked by a problem program. In this case, the calling program provides the utility program with certain auxiliary parameters in main storage, as shown in Figure 1 on page 3. If the utility program is invoked by the job scheduler, only the pointer to the EXEC statement parameters is present.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

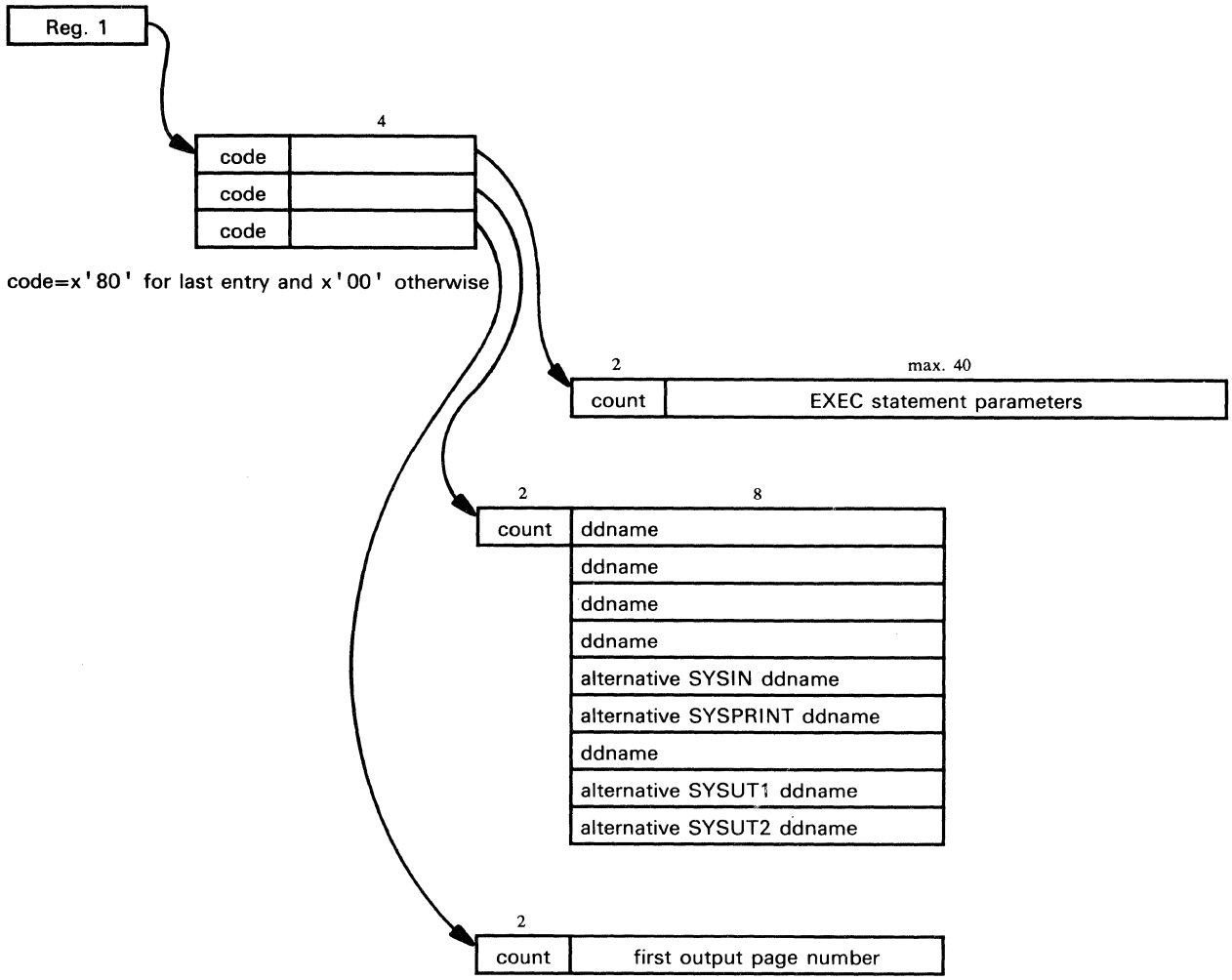


Figure 1. Auxiliary Parameter Formats for IEHATLAS, IEHINITT, IEHLIST, IEHMOVE, and IEHPROGM

DEVICE ALLOCATION AND VOLUME MOUNTING (IEHMVSSF, IEHMVXSF, AND IEHVOLMT)

IEHPROGM, IEHMOVE, and IEHLIST require that volumes be mounted dynamically. However, the serial numbers and device types of these volumes are not necessarily known to the user at the time the job is submitted. Once this information is known, data control blocks must be constructed within the program itself, containing ddnames associated with units on which the appropriate volumes may be mounted, using the OPEN (type=J) routine.

In order to ensure that necessary volumes are mounted or mountable, two routines reside on SYS1.LINKLIB:

- IEHMVSSF, which is used by IEHPROGM and IEHLIST
- IEHMVXSF, which is used by IEHMOVE

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Each routine contains the control section via branch and link, whereas linkage to the second is via transfer control (XCTL).

The logical flow of IEHVOLMT is shown in "Chart 01. IEHVOLMT—Volume Mounting Logic" on page 9. Figure 2 and Figure 3 on page 5 show the format of data supplied to IEHVOLMT by the calling routine. Figure 4 on page 5 shows the format of an internal table maintained by IEHVOLMT in allocated main storage; the internal table is built once for each execution of IEHMOVE and IEHLIST, and once for each time IEHPROGM gives control to the volume mounter.

IEHVOLMT makes use of an internal table that it builds and which contains pointers to ddnames in the TIOT. It uses a mask that it constructs from information in the UCB for the device, indicating which devices on the system have been allocated to the current task. It also uses a device mask table built by the scheduler to indicate which of the devices on the system will accept the specified device types.

For each volume mounting request, IEHVOLMT returns to the calling routine a pointer to a ddname associated with a unit on which the desired volume may be mounted. The ddname is inserted into a field of the DCB, and the desired volume is mounted by the open routine (type J). Actual mounting is accomplished by either IEHVOLMT or the calling routine, as indicated by Field 3 of the internal table header (Figure 2).

The essential processing in IEHVOLMT lies in the comparison of two masks. The first is obtained from the device mask table, using the device type supplied by the calling routine; the second is constructed by IEHVOLMT, using the UCBs allocated to the current task.[1]

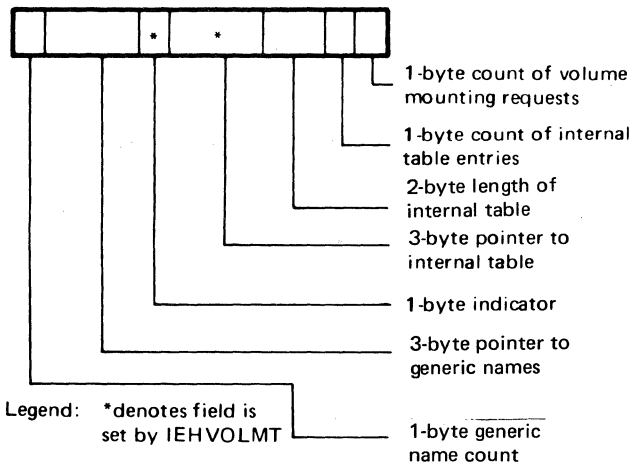


Figure 2. Internal Table Header

[1] The UCBs are found as follows: Location 16 in main storage points to the communications vector table, which in turn contains a pointer to a list of UCB pointers. The task I/O table (TIOT) is then used to distinguish the appropriate UCBs.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

In each mask, each bit represents a unit: In the first mask, an "on" condition means that the unit will accept the device type under consideration; in the second, an "on" condition means that the unit has been allocated to the current task. When both conditions occur for a given unit, IEHVOLMT checks to see if the desired volume is already mounted; if it is, an indication to that effect is returned. If the volume is not already mounted, the ddname associated with that unit (as found in the TIOT) can be used by the OPEN routine (type J) to mount the desired volume on the allocated unit. As explained earlier, the open routine may be invoked by either IEHVOLMT or the calling routine.

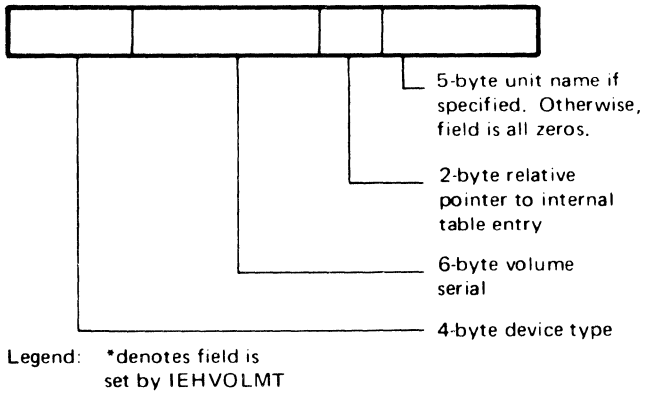


Figure 3. Volume Mounting Request

Internal Table Length	Displacement to Mask
2 bytes	2 bytes

Usage Code	Pointer to DD name
Usage Code	Pointer to DD name
Usage Code	Pointer to DD name
Mask length[1]	
Device mask[1]	
Address of the device mask table	

[1]Device mask length, in bytes, is equal to the number of UCB pointers in the UCB pointer table divided by 8.

Figure 4. Internal Table Maintained by IEHVOLMT

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

INDICATOR SETTINGS: Field 3 of the internal table header (Figure 2 on page 4) may have the following settings:

Bit	Value	Meaning
0	0	No mounting is to be done.
	1	Mounting is to be done.
1	0	No demounting.
	1	Volume mounting requests having the high-order bit of the relative table address set to 1 are to be demounted (the units made available).
2	0	Ignore old requests (ignore usage code, Figure 4 on page 5).
	1	Old requests are valid.
3	0	Build internal table.
	1	Internal table is already built.
4		Unused.
5		Unused.
6-7	01	All volume mounting requests accomplished.
	10	No volume mounting requests accomplished.
	11	Some volume mounting requests accomplished; those volume mounting requests having a relative table address of zero were not accomplished.

CONTROL CARD SCANNER (RDCDRT)

IEHMOVE and IEHLIST contain copies of a control card scan routine, RDCDRT. Each call to RDCDRT results in the return of the item scanned, together with an indication of the type (operation, keyword, or parameter). In IEHMOVE, the routine has the load module name IEHMVESJ; in IEHLIST, it has the name RDCDRT.

This routine reads control cards (using QSAM), checks syntax, and returns a command word, a keyword, or a parameter to the calling routine.

The calling routine supplies a 192-byte work area (on a fullword boundary) followed by the DCB for the control card data set. The calling routine must open this data set. The control card routine inserts the address of the end-of-file routine KEOF in the DCB.

The calling sequence for RDCDRT is as follows:

- Register 13 points to the first byte of the work area.
- Register 14 points to the return address in the calling routine.
- Register 15 points to the entry point RDCDRT.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Upon returning control to the calling routine, the control card scan routine returns the following information:

- Register 1 points to the starting address of the item scanned.
- Register 2 contains the length of the item scanned.
- The first byte of the 192-byte work area is labeled SWITCHRD; when set to 1, its bits have the following meanings:

Bit	Meaning
0	Syntax error
1	Bypass switch
2	End-of-file
3	Initial entry
4	Command word
5	Keyword
6	Parameter
7	Parameter or keyword delimited by a right parenthesis

The control card scan routine contains the following subroutines:

RDCARD
resets switches and saves registers 3 through 14.

KGTCD
reads a card into the work area, using QSAM.

KTRT
saves the "start" address of the scan, and scans for a delimiter.

KPPARQ
stores the address and length (of the item scanned) in registers 1 and 2.

KINVAL
is entered when an invalid delimiter is found.

DATA SET UTILITY PROGRAMS

Data set utilities are executed under the operating system and perform operations on data sets at the logical record level. The data set utilities are as follows:

- **IEBCOMPR** compares two data sets at the logical record level.
- **IEBCOPY** copies all or a specified portion of a partitioned data set, renames and/or replaces specified members of a data set, compresses existing data sets, and alters, copies, or reblocks load modules.
- **IEBDG** produces test data sets for use in program debugging procedures.
- **IEBEDIT** produces an edited input job-stream data set from a master input job-stream data set.
- **IEBGENER** copies or converts a sequential data set to a partitioned data set.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- **IEBIMAGE** helps maintain the printer image library for the IBM 3800 Printing Subsystem.
- **IEBISAM** copies, unloads, loads, and prints indexed sequential data sets.
- **IEBPTPCH** prints or punches all or selected portions of a sequential data set, a partitioned data set, or specified members of a partitioned data set.
- **IEBTCRIN** reads data from the IBM 2495 Tape Cartridge Reader, edits as specified by the user, and produces one or two sequentially organized data sets.
- **IEBUPDTE** incorporates source language modifications into sequential and partitioned data sets.

The data set utility programs use QSAM for both reading the SYSIN data set and writing the SYSPRINT data set. Both data sets may have blocking factors that are other than one.

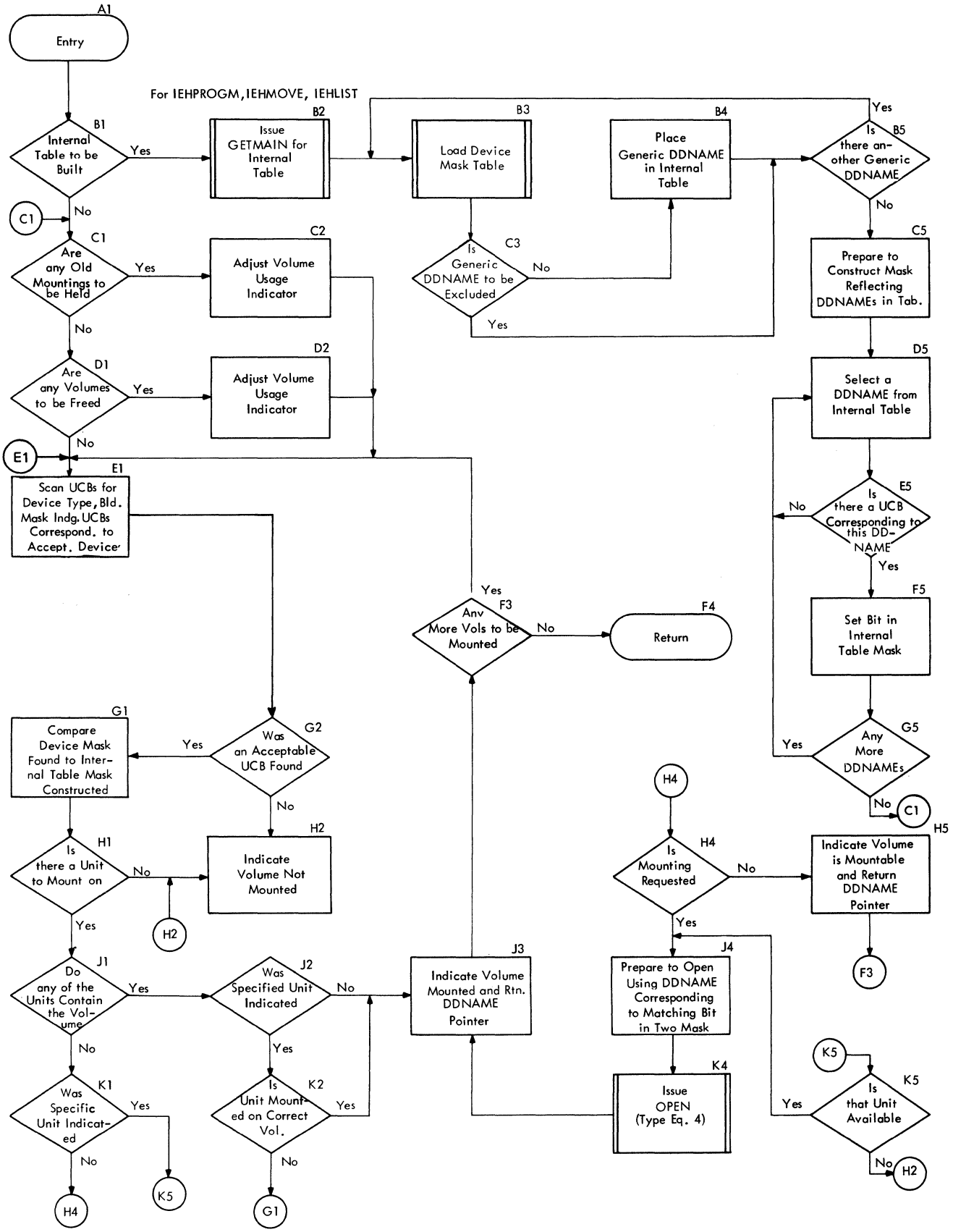
To optimize space allocation and thus permit the data set utility program to take full advantage of any storage that is available, the storage requirements for buffers and tables are dynamically determined at execution time. If more storage is requested than can be supplied by the main storage supervisor routines, the task is automatically terminated. If, however, the request cannot be filled immediately because of priority scheduling within a multitasking environment, the execution of the utility program can be delayed until its storage requirements are met.

INDEPENDENT UTILITY PROGRAMS

ICAPRTBL is the only independent utility program executed outside and in support of MVS/370. It loads the universal character set (UCS) buffer and the forms control buffer (FCB) of an IBM 3211 or 3203 Model 5 Printer.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 01. IEHVOLMT—VOLUME MOUNTING LOGIC



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

PART 2: FUNCTIONAL DESCRIPTIONS OF THE UTILITIES

This part contains functional descriptions of each of the utilities available for MVS/370 Data Facility Product (MVS/370).

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

ICAPRTBL PROGRAM

The ICAPRTBL utility program loads the universal character set (UCS) buffer and the forms control buffer (FCB) of an IBM 3211 or 3203 Model 5 Printer. Both functions are performed during each execution of the program.

See "Chart 01. Loading the FCB and UCS Buffers" on page 14 for a diagram of the ICAPRTBL program.

PROGRAM FLOW

The ICAPRTBL program consists of six routines: INIT, INPUT, CONTROL, OUTPUT, IO, and WAIT.

INIT
gains control from the Initial Program Loader. It loads a new PSW, saves the address of the card reader, and initializes required registers and data areas.

INPUT
reads a control card and analyzes its contents. After a statement is read, a flag bit for that statement is turned on. JOB and END must be the first and the last control statements read. DFN, UCS, and FCB, the intermediate control statements, can appear in any order, except for the 3203 Model 5; in this case, the DFN statement must precede any UCS statement. All information retrieved from the control cards is stored in work areas for the CONTROL and OUTPUT routines.

During the reading of control statements, the LPI parameter of the FCB statement is analyzed and bit 3 of the first image byte is set on if the specified print density is 8 lines per inch. The end-of-forms bit is set on in the FCB image after the FORMEND parameter is examined.

INPUT has five subroutines:

- GET sets up the CCW to read a control card before it calls IO to start the CCW and test it for completion.
- SEARCH receives an address within the control statement input area in register 2. This address is incremented until a blank is found in the control statement.
- SEEK receives an address within the control statement input area in register 2. This address is increased until a nonblank is found in the control statement.
- INTRP receives the address of a zoned decimal field in register 2 and converts the field to a binary value.
- DEFAULT builds the UCS image as specified on the UCS parameter of the DFN statement, or as defaulted when a UCS statement was not provided and a UCS parameter was either not provided or its image-id option was not recognized.

  Upon normal completion, INPUT exits to CONTROL.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CONTROL
performs the functions requested through control statements. It gets the address of the printer, looks at the image provided, and generates an additional 64 image bytes that are used to screen data checks. It then sets up the CCWs to enable IO to next load the UCS and FCB buffers and to read back the UCS buffer. Upon normal completion, CONTROL exits to OUTPUT.

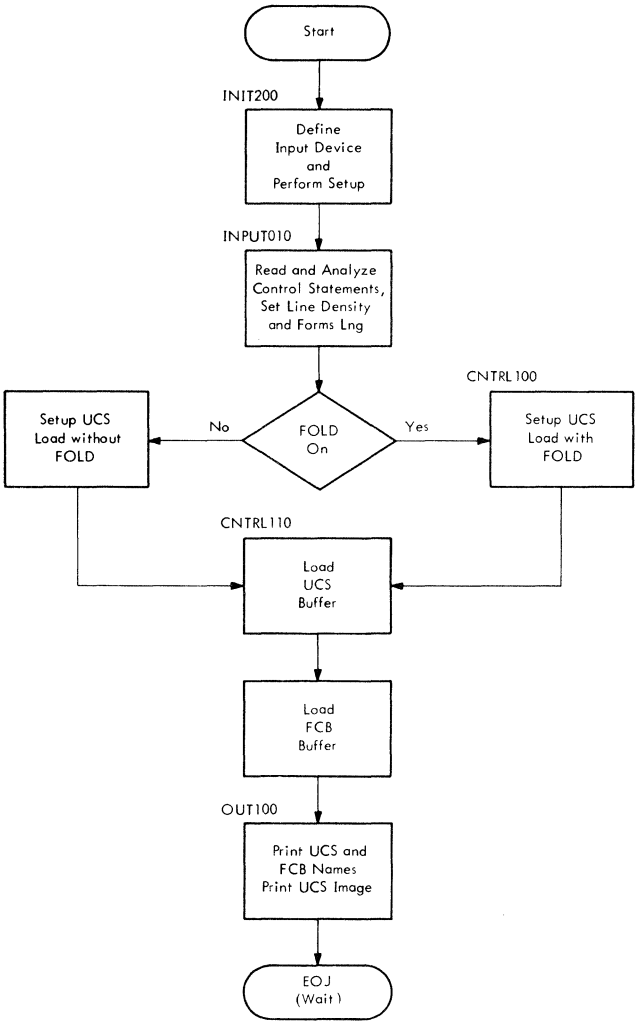
OUTPUT
sets up a CCW to print UCSNAME, FCBNAME, and the UCS image. UCSNAME and FCBNAME are moved to the output area and a branch is taken to IO to print the line. Upon return to OUTPUT, the first of eight lines (for the 3211), or of five lines (for the 3203 Model 5) that comprise the UCS image is moved to the output area and printed by IO. The first seven lines for a 3211 contain 56 characters each; the 8th line contains 40 characters. The first four lines for a 3203-5 contain 56 characters each; the 5th line contains 16 characters. After the last line of the UCS image has been printed, OUTPUT exits to WAIT.

IO
is entered from INPUT, CONTROL, or OUTPUT to handle any input or output. It issues a START IO and subsequently checks for unit busy, unit operational, and acceptance of the START IO. IO then tests for unit available, unit check, channel error, and device end. If a unit check occurs, sense information is stored for display by the user. Normal completion returns control to WAIT.

WAIT
loads a wait state code into the address portion of the PSW to indicate either an error condition or normal completion of the program. All codes are provided by the calling routine. X'B01' is the code for normal end of job. Wait state error codes can be found in OS/VS Message Library: VS2 System Codes.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 01. LOADING THE FCB AND UCS BUFFERS



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

The IEBCOMPR utility program is a data set utility program that compares either two sequential or two partitioned data sets. Depending on the characteristics of the data sets, the comparison is performed at either the physical or the logical record level. Messages are provided to describe any errors that may be detected during the execution of the program.

The major input to the program must consist of two identically organized data sets, SYSUT1 and SYSUT2. The formats of the records in the data sets may be either fixed, variable, variable spanned, or undefined. (Variable spanned records for partitioned data sets are not compared.) For fixed, variable, and variable spanned records, the logical records may be blocked or unblocked. If the records have keys, the keys are also compared.

In addition to data set records, the program will compare user input header and/or trailer labels unless otherwise requested. Optional user exit routines may be specified so a user can process user labels.

Note: Processing for user labels is valid only for sequential data sets.

In addition to the data sets to be compared, the program requires as input a control data set (SYSIN) and, if the program is invoked by a user's program, a parameter list to specify any alternate names for standard ddnames. A starting page number may also be provided.

Output from the IEBCOMPR program consists of a data set that may contain diagnostic messages, a count of the records compared, a listing of records that were not compared, and a listing of records that compared unequally. In addition, the program provides the user a return code that indicates the degree of success of the execution of the program.

STRUCTURE OF THE IEBCOMPR PROGRAM

As shown in Figure 5 on page 16, the IEBCOMPR program consists of three segments. The root segment consists of assembly modules IEBCOMPM, which contains the messages for the program, and IEBCROOT, which contains the entry point to the program. IEBCROOT also performs initialization procedures, processes the input parameter list, and writes all output on the SYSPRINT (output) data set.

The control card analyzer segment consists of the statement analyzer module, IEBCANAL, and the statement scan module, IEBCCS02. IEBCCS02 opens the SYSIN data set, reads the statements contained therein, passes parameter information to the statement analyzer, and closes the SYSIN data set. (This is the same module that is used in the utility programs IEBGENER and IEBPTPCH for the control card scan function.) IEBCANAL analyzes and stores the parameter information supplied by IEBCCS02, and sets switches based on the parameter analysis.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

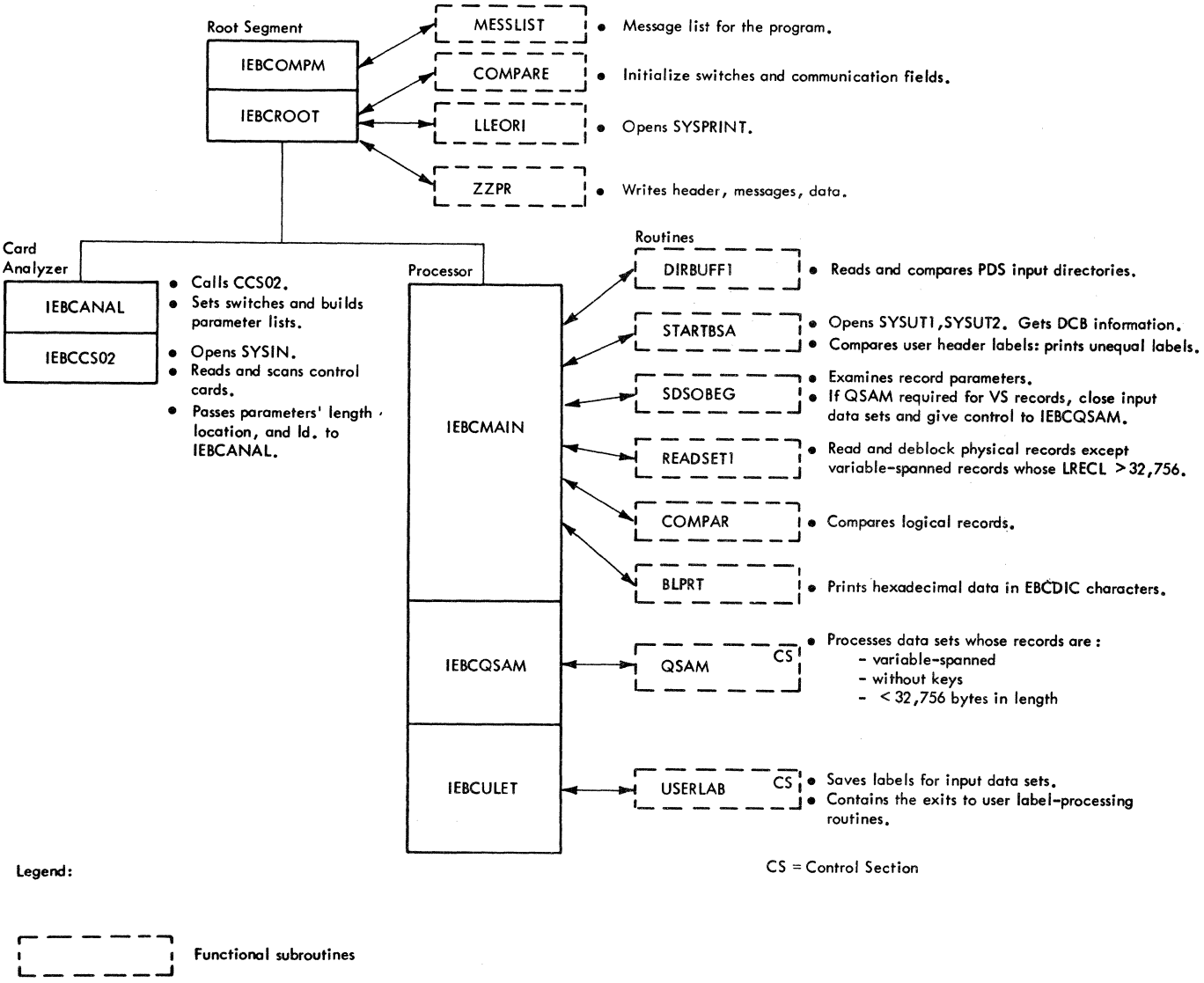


Figure 5. Structure of the IEBCOMPR Program

The processor segment consists of the main processing module, IEBCMAIN, and two secondary modules, IEBCQSAM and IEBCULET. IEBCMAIN compares input labels and the records of the input data sets. IEBCQSAM receives control if the input data sets require QSAM processing. (IEBCMAIN uses BSAM processing to00096 access the data set records.) If label processing is required, IEBCULET receives control to save input labels for later processing and to provide exits to user-processing routines. The last two modules act, in effect, as subroutines of the main processing module. Figure 6 on page 17 indicates the passage of control and data flow among the modules of the program.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

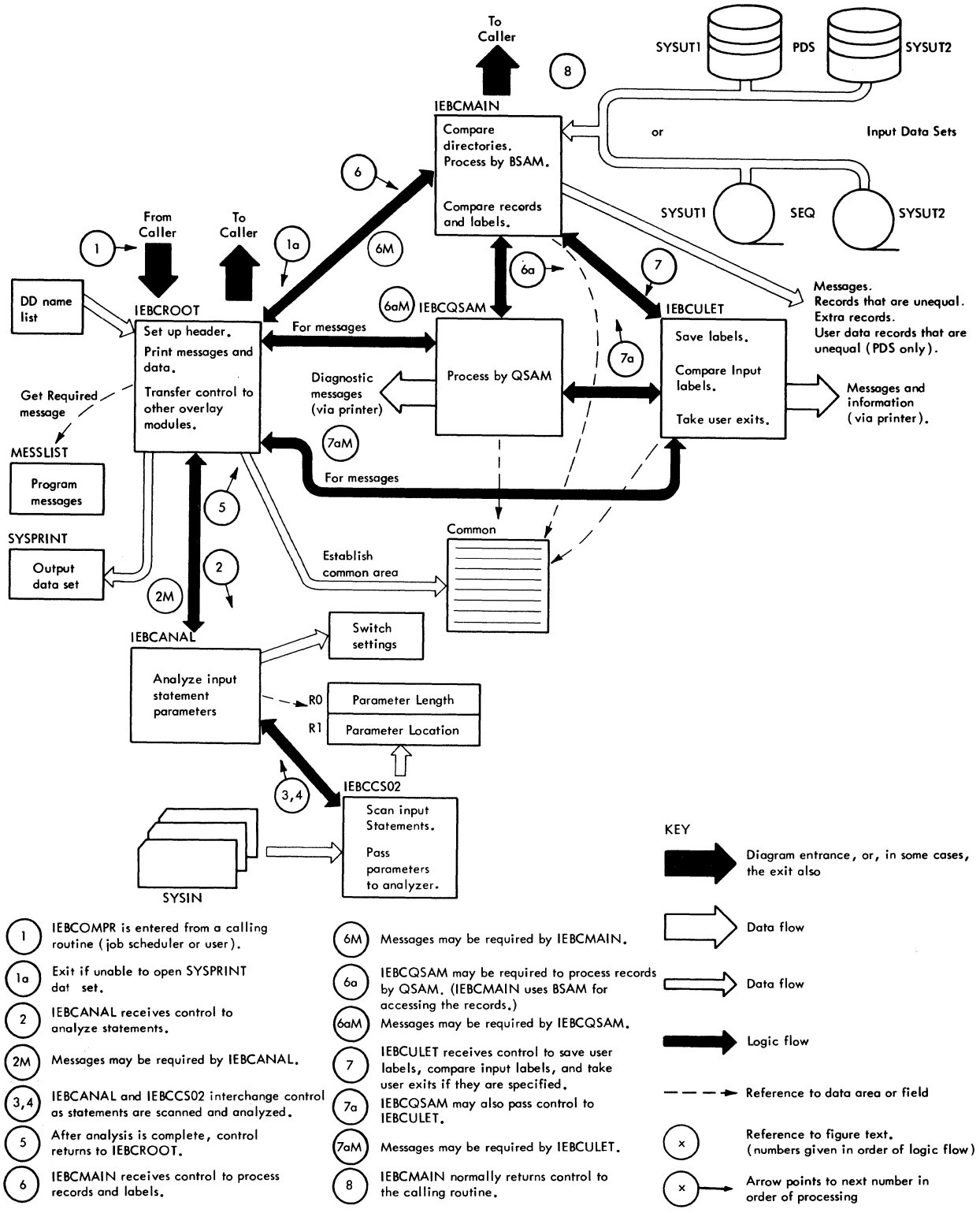


Figure 6. Processing and Data Flow in IEBCOMPR

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

METHOD OF OPERATION

The basic operations performed by the IEBCOMPR program are:

- Analyzing the input parameter list
- Reading and analyzing the control statements
- Precomparing directory entries of partitioned data sets
- Comparing labels for sequential data sets
- Comparing records of both input data sets
- Providing message and data results

INVOKING THE IEBCOMPR PROGRAM

The IEBCOMPR program may be invoked by the job scheduler or by the user's program. If the user relies on the job scheduler to invoke the program, standard ddnames must be used for the control statement data set (SYSIN), for the two input data sets (SYSUT1 and SYSUT2), and for the output data set (SYSPRINT). In addition, the operand of the EXEC statement must specify IEBCOMPR or a procedure that executes IEBCOMPR. The job scheduler then passes control to IEBCOMPR, and register 1 will point to a 4-byte field that contains X'80' in its high-order byte.

To invoke IEBCOMPR from a user program, the user must supply the necessary job control language (JCL) statements to define certain data sets and cause control to be given to the IEBCOMPR program. The user may specify alternates for any of the four standard ddnames (SYSIN, SYSUT1, SYSUT2, SYSPRINT), and may specify the first page number to be used on the message data set. These alternates will override the ddnames supplied by the operating system. The user must indicate the storage location of this information by means of a parameter list in register 1. If there is a parameter list, the parameters are read and stored in areas of a common communication region (referred to by its DSECT label, COMMX), which is passed from module to module. The parameters are extracted from this common area as needed during the rest of the program. If the user wants to invoke IEBCOMPR but doesn't want to pass a parameter list, the high-order byte of the fullword pointed to by register 1 should contain X'80'. Control may be passed to IEBCOMPR by means of either a LINK or an ATTACH macro instruction.

Utilities contains further discussion of the requirements for invoking utility programs.

THE FORMAT OF THE PARAMETER LIST

When a user's program invokes the IEBCOMPR utility program, it may pass a parameter list as input to the program. If there is such a list, register 1 contains its storage address. As shown in Figure 7 on page 19, the list consists of three 4-byte entries. The first entry consists of zeros and is not used by the IEBCOMPR program.

Note: If the high-order byte of this first entry contains X'80', either the job scheduler was used to invoke IEBCOMPR, and there are no further entries in the list, or the user didn't pass a parameter list.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

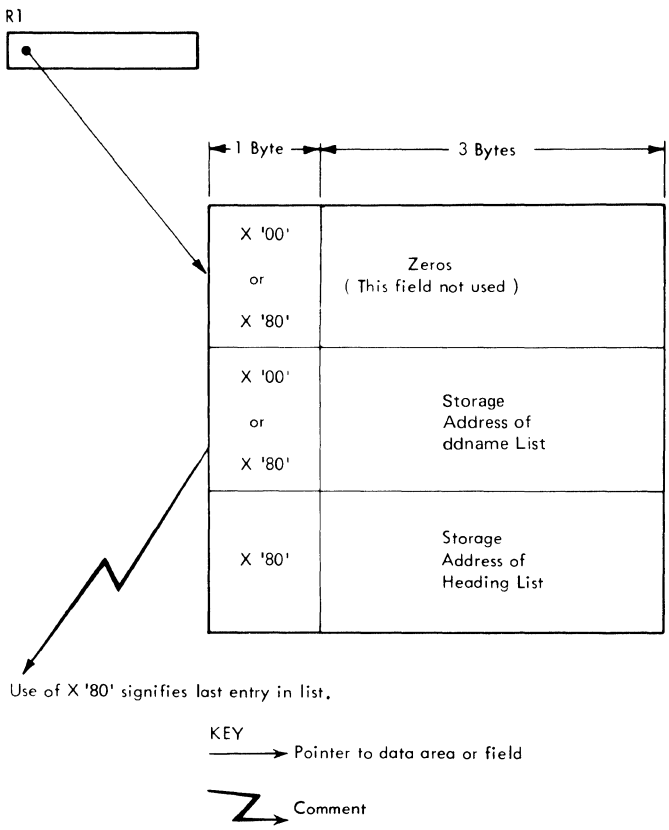


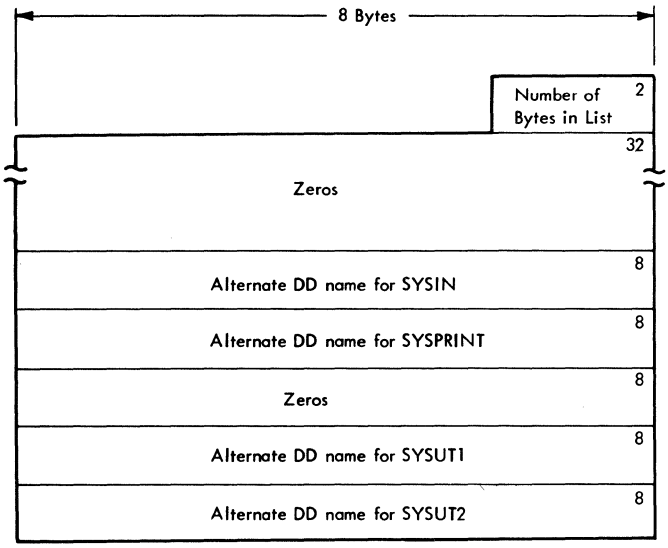
Figure 7. Parameter List Passed by User's Program

The second entry in the parameter list pertains to alternate ddnames the user may have indicated. The high-order byte of this entry contains either zeros (if the third list entry is present) or a X'80' if there is no third entry. The low-order 3 bytes of the entry contain the storage address of the ddname list. This list begins on a halfword boundary that is not also a fullword boundary. The first 2 (high-order) bytes of the list constitute the list's length field and contain the total number of bytes in the remainder of the list. The remainder of the list (if the length value is not zero) contains 32 bytes of zeros followed by up to five 8-byte entries. Figure 8 on page 20 illustrates the structure of the list. If the user does not specify an alternate ddname for any given slot in the list, the slot must contain binary zeros. The list may be shortened only by omitting names from the end of the list, and the length field must be adjusted accordingly.

The third parameter list entry is reserved for header-paging information. If this entry is present, its high-order byte always contains X'80', signifying the last entry in the list.

The low-order 3 bytes contain the address of the header parameter list. The header parameter list contains both the user's page number that the IEBCOMPR program is to place on the first page of printed output (to override the standard value) and the length (in bytes) of this page number. (This length is less than or equal to 4 bytes.) The header parameter list, as well as the ddname parameter list, is optional, and its presence and location in main storage are determined by interrogating the user's input parameter list.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1



Note: If alternate name is missing, standard name is assumed.

Figure 8. Alternate ddname Parameter List Established by User's Program

OPERATING ON THE DDNAME PARAMETER LIST

When the IEBCOMPR program is entered, module IEBCROOT determines the presence of any alternate ddnames. If there are alternate names for one or more of the data sets SYSIN, SYSUT1, or SYSUT2, the module moves each name into a specified field of the region, COMMX, which is defined by IEBCROOT. Later, the analyzer module, IEBCANAL, moves a specified alternate name for the SYSIN data set into a work area so the statement reader, IEBCCS02, can refer to it. If an alternate name for the output data set, SYSPRINT, exists, the module places it over the standard name in the DDNAME field of the output DCB (see Figure 9 on page 21).

OPERATING ON THE PAGE NUMBER PARAMETER LIST

If the header-paging parameter list exists, module IEBCROOT adjusts the standard starting page number (0001) in the header as given at PRHEAD, a location in the constant area for that module (see Figure 10 on page 22 and Figure 11 on page 23). This header is used when the message writer routine begins the first page of program output.

READING AND ANALYZING THE CONTROL STATEMENTS

The SYSIN data set contains the control statements that describe the processing to be done on the data sets SYSUT1 and SYSUT2 by the IEBCOMPR program. These statements contain parameters that indicate the data set organization, the set of data set labels (if any) that are to be processed as data, and the names of user routines for label processing, error processing, and preliminary record processing.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

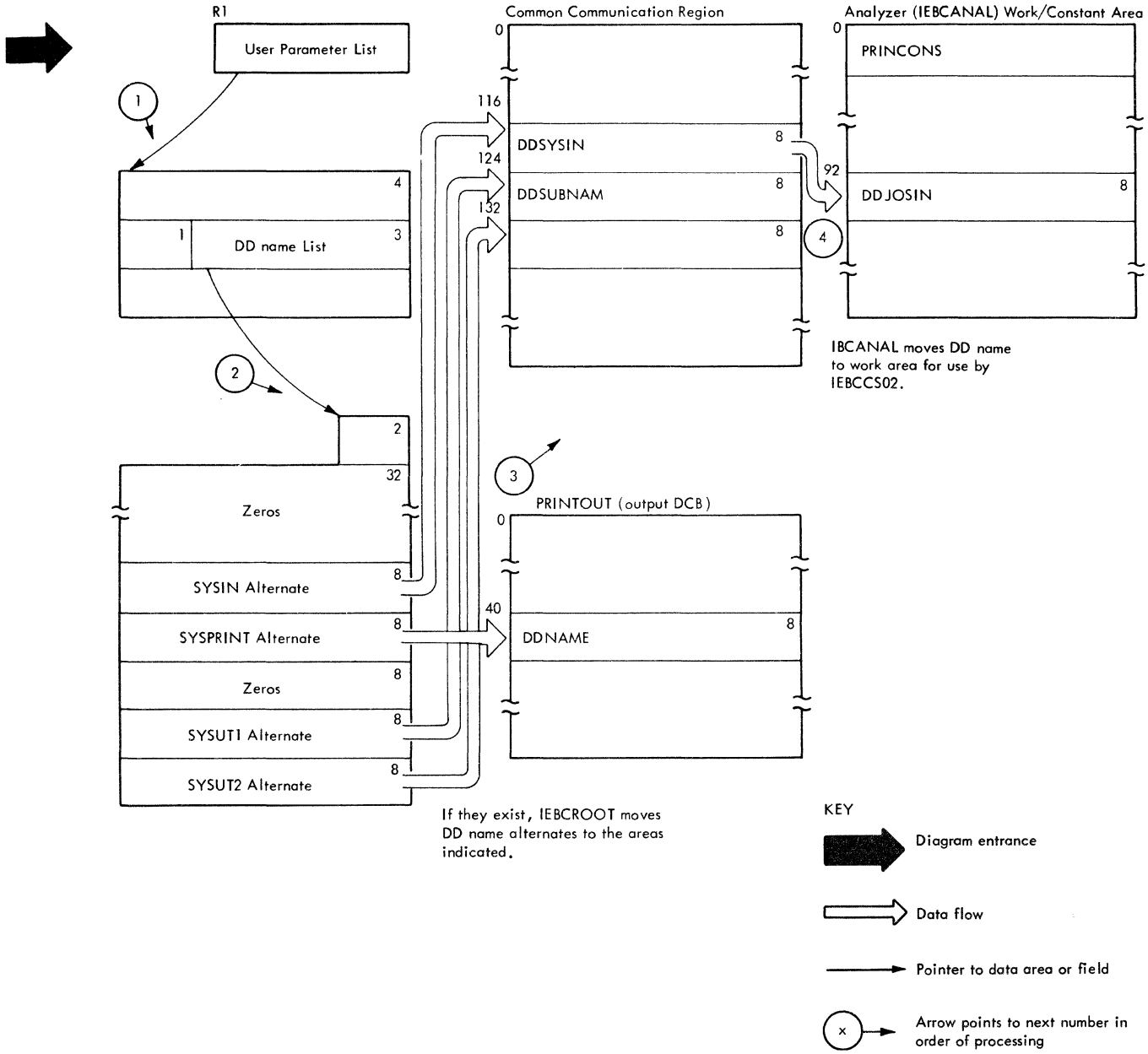


Figure 9. Moving ddname List Data

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

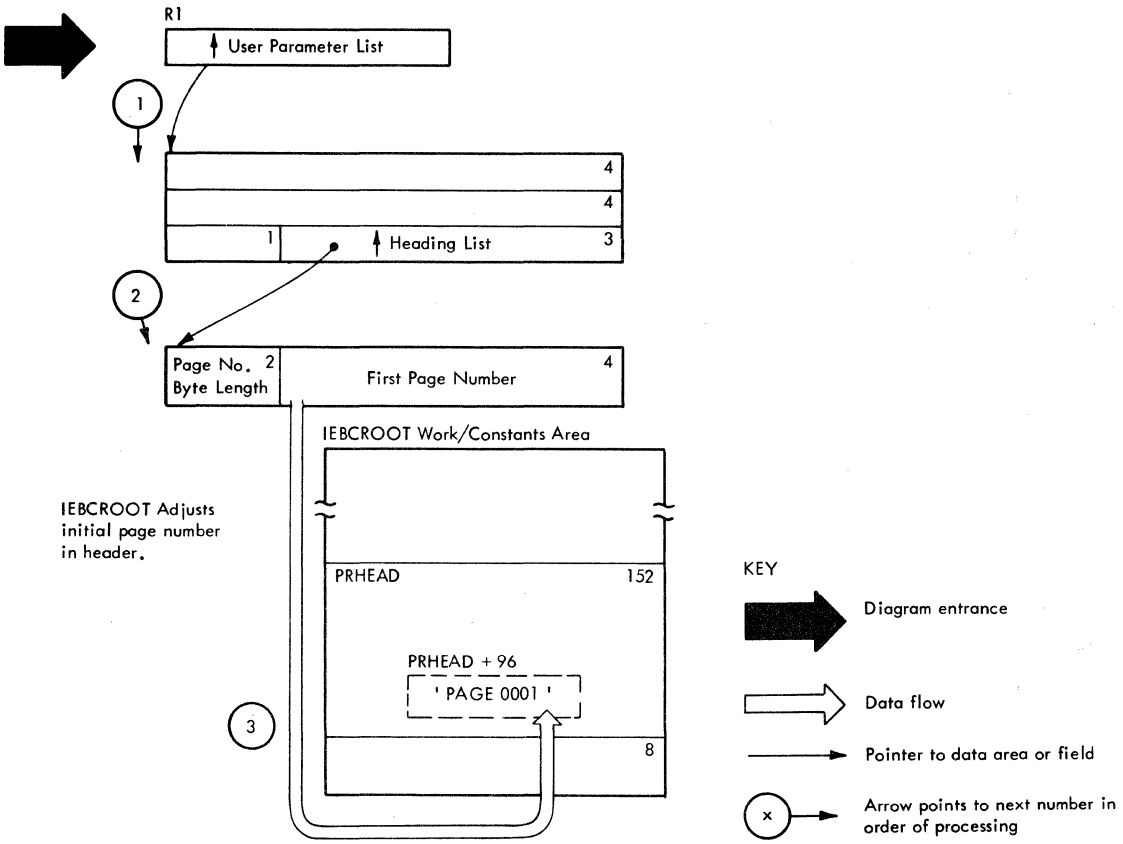


Figure 10. Moving Header List Data

The utility scan routine, module IEBCCS02, performs the actual statement scan, checks for statement validity, and passes statement parameters, one at a time, to the statement analyzer. The statement analyzer places the parameter values in the communication region that has been established by the root segment. From there, other modules move the values to work areas as needed.

CONTROL STATEMENT ANALYSIS

Modules IEBCANAL and IEBCCS02 alternate with each other in the processing of the control data set (see Figure 12 on page 24). Module IEBCANAL gives control to module IEBCCS02 (the scan module) by issuing a CALL macro instruction. At that time, a parameter list, pointed to by the contents of register 1, is passed to the scan module. Figure 13 on page 25 shows the parameter list that is passed. Module IEBCCS02 receives control each time a new statement command, keyword, or parameter is to be read and scanned, and it passes information to module IEBCANAL, which analyzes the information and stores it in work areas or tables.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

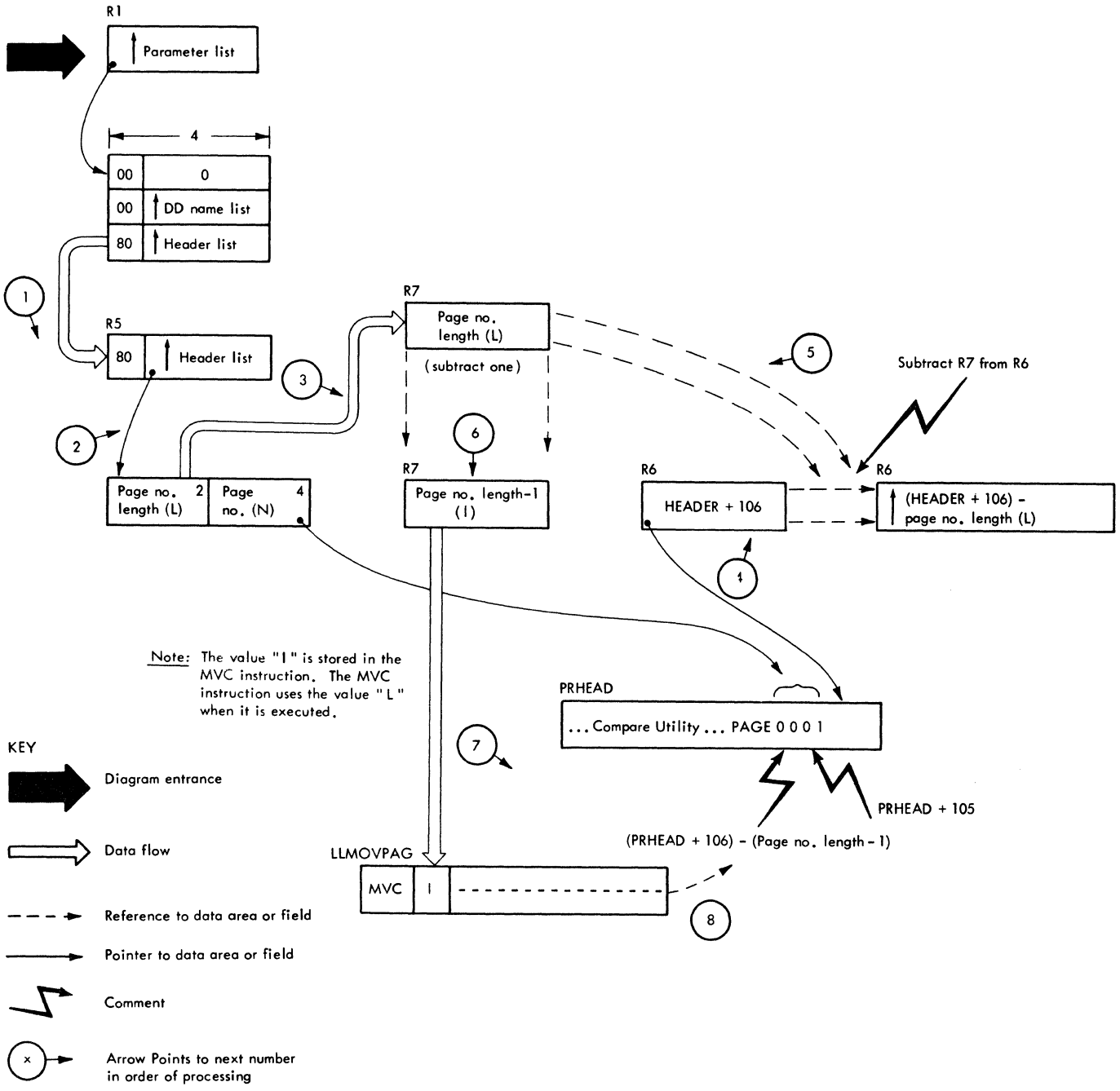
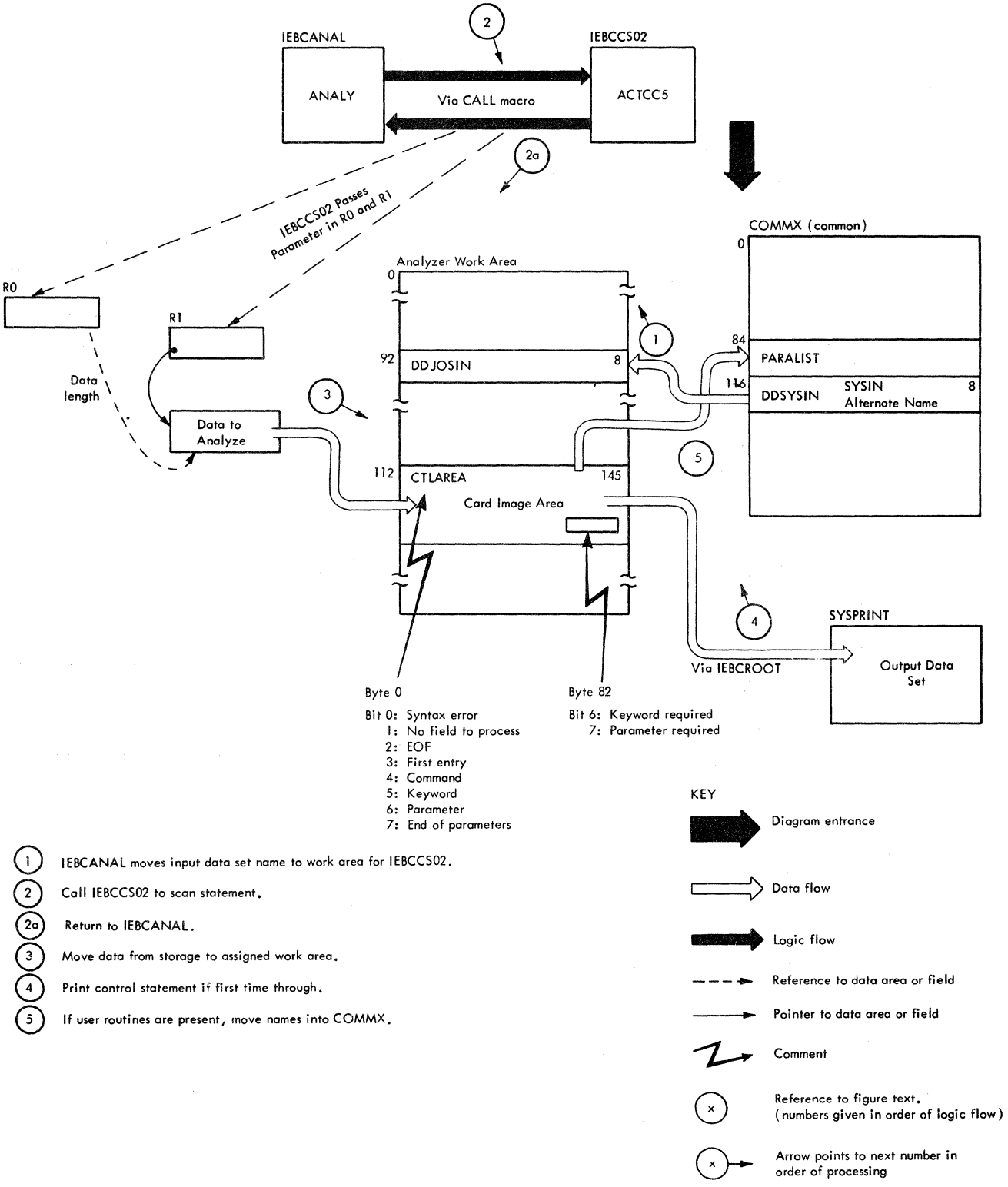


Figure 11. Processing Page Number (by IEBCROOT)

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1



(1) IEBCANAL moves input data set name to work area for IEBCCS02.

(2) Call IEBCCS02 to scan statement.

(2a) Return to IEBCANAL.

(3) Move data from storage to assigned work area.

(4) Print control statement if first time through.

(5) If user routines are present, move names into COMMX.

Figure 12. Processing and Information Interchange for Statement Analysis

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

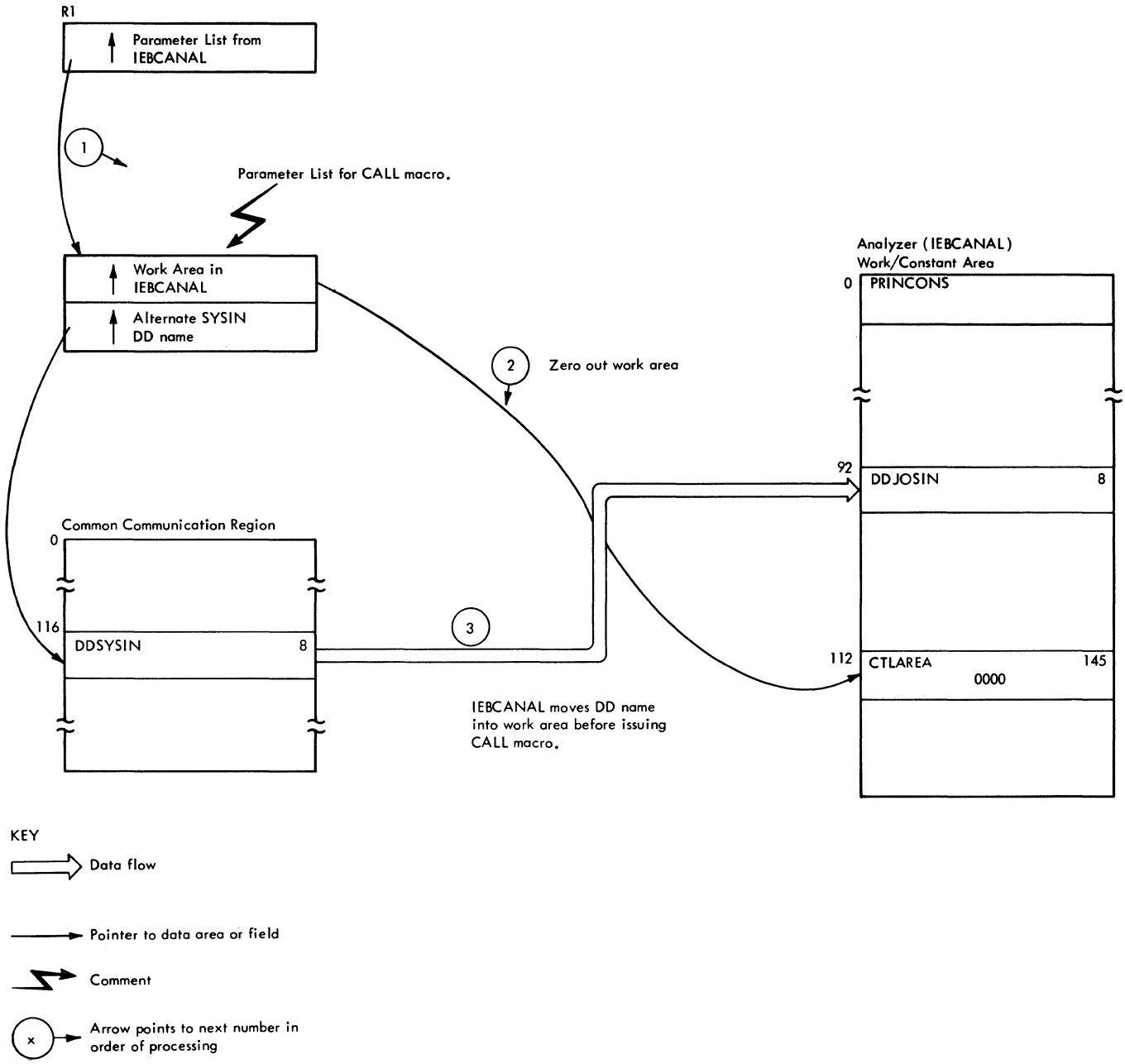


Figure 13. Parameter List Passed to IEBCCS02

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

The first time module IEBCCS02 is entered, the control data set (either SYSIN or the SYSIN substitute name passed in the parameter list) is opened. IEBCCS02 then issues a GET macro instruction to obtain a control statement from the control data set. The module then scans the statement for a command field. On subsequent entries, the module either scans for the next logical field in the current statement or obtains the next statement. When the end of the control data set has been reached (at EOD), the scan module closes the control data set.

When module IEBCCS02 returns control to module IEBCANAL, it passes the beginning of the next field to be analyzed and the length of that field as parameters. Registers 0 and 1 contain these parameters, as shown in Figure 16 on page 28.

When the scan module receives control, it tests switches in the common work area (see Figure 15 on page 27), passed between it and the analysis module, to determine the status of the processing to that point. As the scan module moves along a statement in the input buffer of the work area, it detects the beginning and ending of fields according to Figure 14 and sets switches to record its findings.

Statement Contents	Preceding Character	Following Character
Command word	Blank, or nothing	Blank
Keyword	Blank or comma	Equal sign
Parameter	Equal sign	Blank or comma

Figure 14. Control Statement Field Delimiters

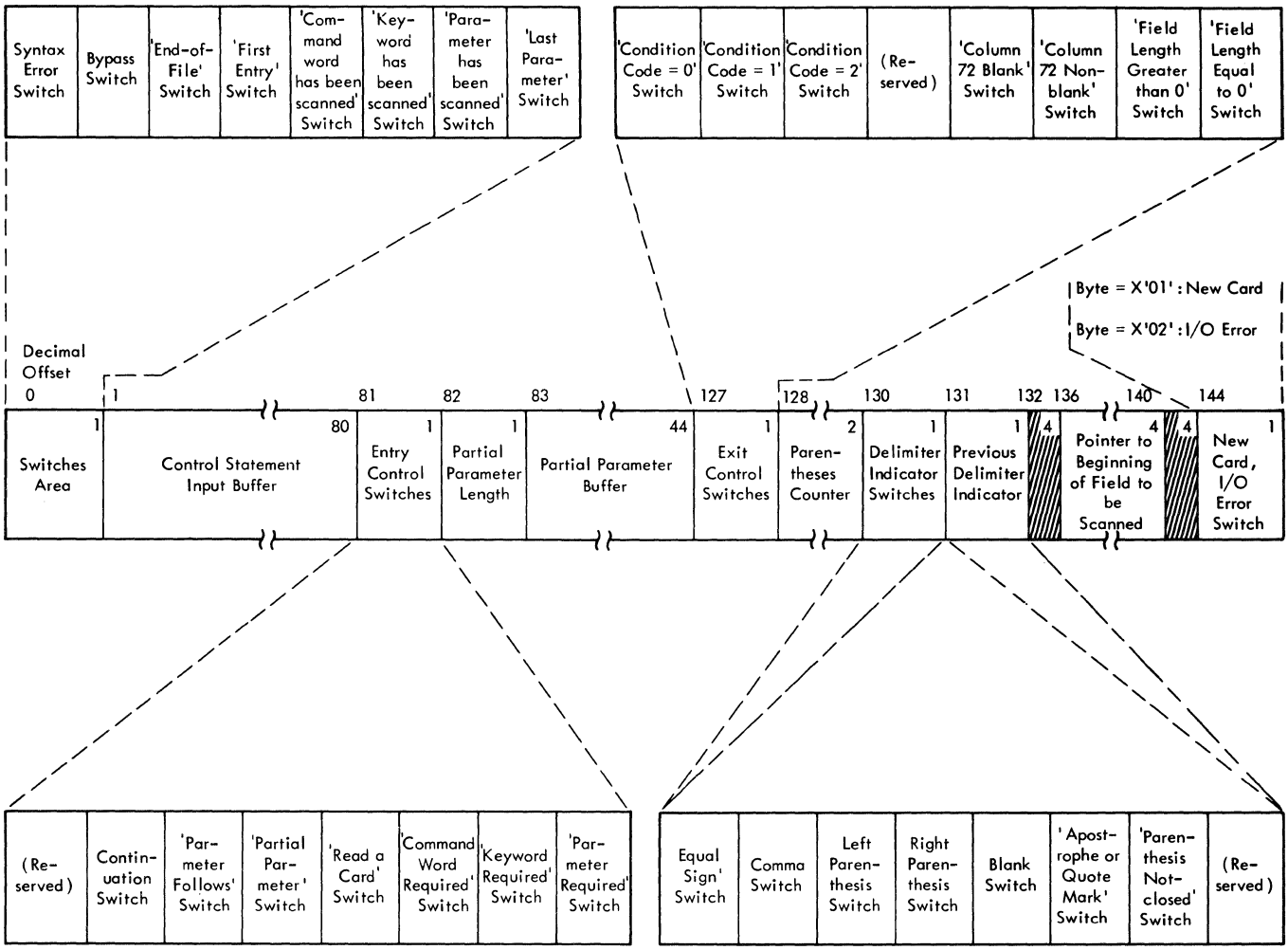
Each time the analysis module receives control, it analyzes the switches set by module IEBCCS02 to determine the errors that may have been detected and the generalized identity (that is, command word, keyword, or parameter) of the word that has been returned.

When the scan routine returns a field pointer in register 1, it has taken account of the requirements imposed on the field being returned. For a command word, the register points to the first nonblank character. (The first control statement must have the command word COMPARE.) For a keyword, the register points to the first nonblank character following either a blank or a comma, depending on the preceding field in the statement. For a parameter, register 1 points to the first character following an equal sign. The field length passed in register 0 does not include the delimiter following the field.

Figure 15 contains an explanation of the various switch and delimiter bytes that appear as part of the common work area used by modules IEBCANAL and IEBCCS02. When statement field errors occur, module IEBCCS02 sets on appropriate bit combinations to indicate them to the analysis module. Figure 17 on page 29 indicates the switch settings that occur.

When a continuation card is encountered, a buffer area within the work area is used to store first the part of the field from the current statement, and then the rest of the field taken from the continuation statement. When the field is passed to the analysis routine, it is passed as a complete, unbroken field.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1



Note: Bits are numbered from 0 to 7 and, except as noted, a setting of 1 indicates the field has the meaning shown in the figure.

Byte	Bit	Meaning
0	3	The current entry to module IEBCCS02 is not the first entry.
0	7	The parameter scanned is the last parameter in a given statement.
81	1	The current statement is continued on another card.
81	2	Either a keyword or a parameter field should follow next on the current statement.
81	3	The current parameter is continued on another statement.
81	5 6 7	Each of these bits indicates that the particular field, which a given bit references, should be the next field scanned on the current statement.

Figure 15. Common Work Area Passed between IEBCANAL and IEBCCS02

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

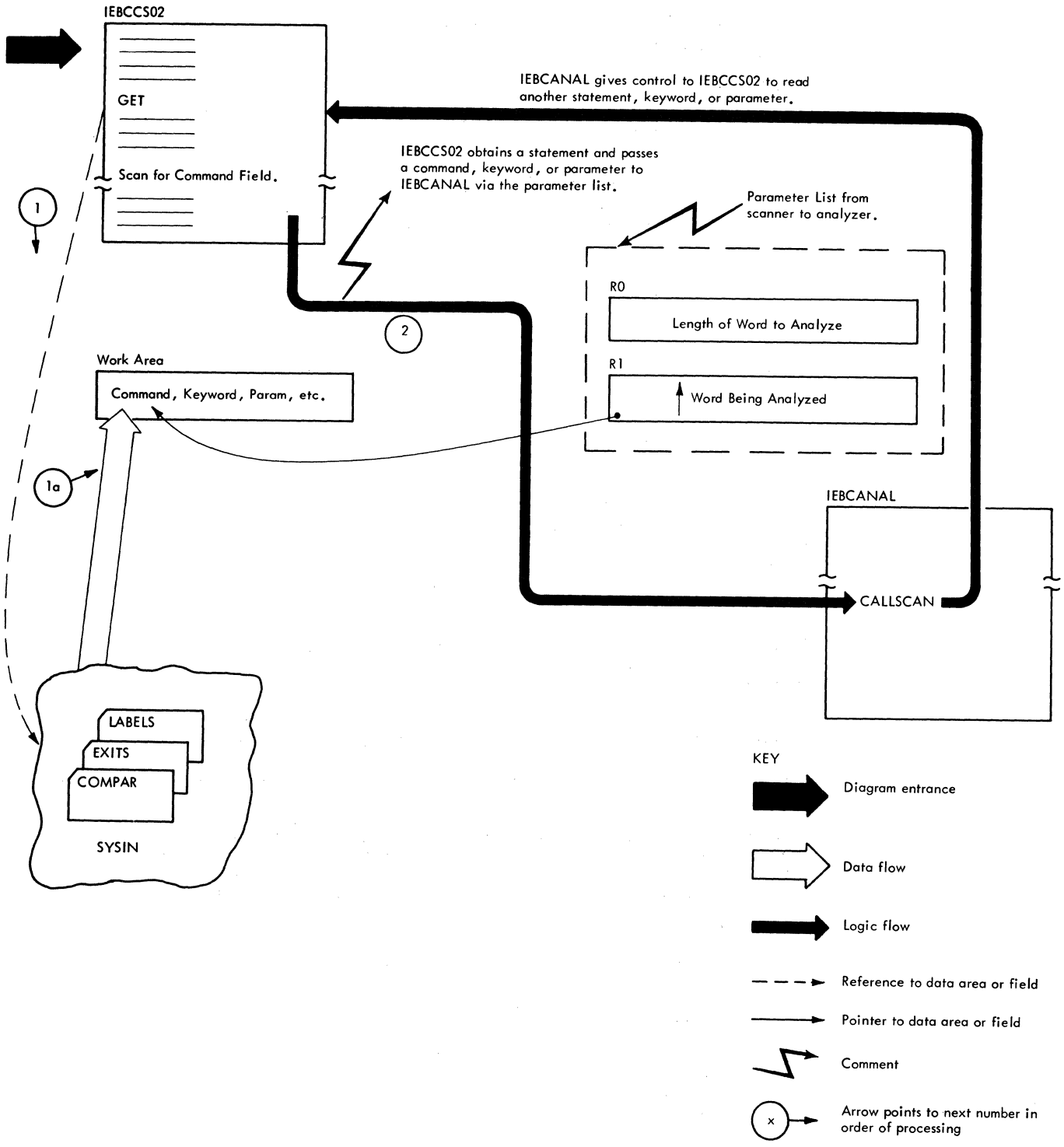


Figure 16. Passing a Parameter List from IEBCCS02

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Switches Set On (=1) in Work Area (CTLAREA)			Error Condition Indicated
Byte	Bit	Name	
0	0	Syntax	Invalid Name
0 0	0 1	Syntax Bypass	Invalid Continuation
0 0 0	0 1 4	Syntax Bypass Command	Invalid Command
0 0 0	0 1 5	Syntax Bypass Command	Invalid Keyword
0 0 0	0 1 6	Syntax Bypass Parameter	Invalid Parameter
0 0 0 0	0 1 4 5	Syntax Bypass Command Keyword	Missing Command
0 0 0 0	0 1 4 6	Syntax Bypass Command Parameter	Missing Command
0 0 81	0 1 7	Syntax Bypass Parameter required	Missing Parameter
0 0 81	0 1 6	Syntax Bypass Keyword required	Missing Keyword
0 0 130	0 1 6	Syntax Bypass Parentheses not closed	Parentheses not closed

Figure 17. Switches Set by IEBCCS02, and Their Significance

The bypass bit will be set by the scan routine when it determines either that there is no valid field to be passed to the analysis routine (that is, if the remainder of the statement contains only comments), or if an error has been found in a statement and thus the rest of the statement need not be scanned.

Module IEBCANAL tests the return switches set by IEBCCS02, and, if it finds an error, determines the statement column in which the error occurred or whether it was a syntax error. If there is an error, the module then gives control to the message writer (routine ZZPR) in module IEBCROOT to write the appropriate message on the SYSPRINT data set. As the scan module gives statement fields to the analysis module, the latter sets switches in its own constant area to indicate the validity of the fields. When user routine names are encountered in the analysis, the names are moved into the region, COMMX, that was defined by module IEBCROOT (see Figure 18 on page 30).

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

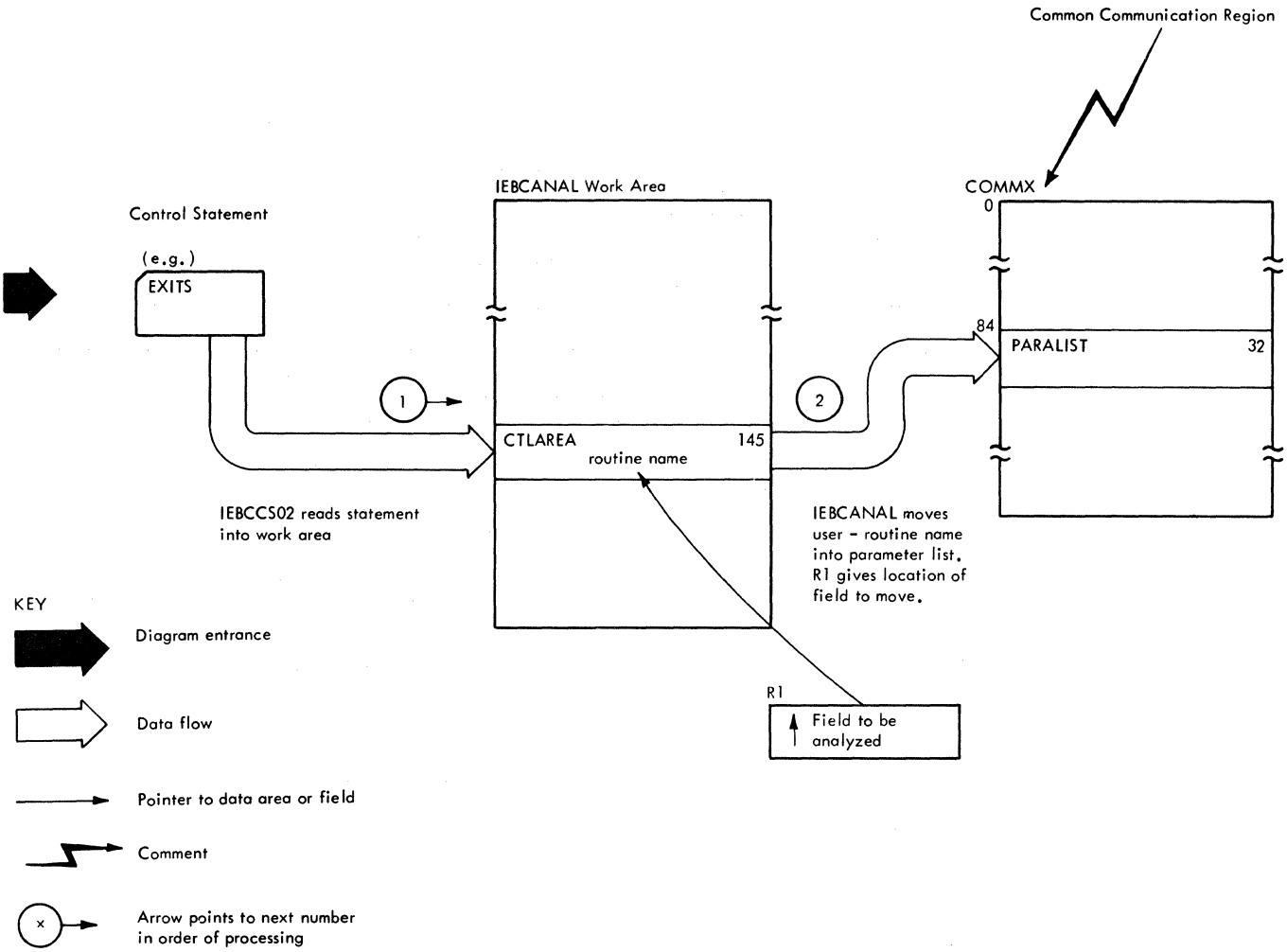


Figure 18. Handling User Routine Names

If the LABELS command is encountered, the analysis routine determines which keyword parameter is present and sets the appropriate indicator in the communication region for use by module IEBCMAIN. It also determines the parameter associated with the TYPORG keyword and sets an indicator if the parameter is PO. The default value, when the parameter is not specified, is zero (PS).

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

The following actions are taken by module IEBCANAL for the control statements indicated:

Control Statement	Action Taken
COMPARE	Set switch DCSW = 01.
	If keyword = TYPORT, set KWORDSW (in the constant area) = 01.
	If TYPORG parameter = PO, set CCADSORG (in COMMX) = 01.
EXITS	Set switch CDSW = 02.
	If keyword = INHDR, set KWORDSW = 02.
	If keyword = INTLR, set KWORDSW = 04.
	If keyword = ERROR, set KWORDSW = 08.
	If keyword = PRECOMP, set KWORDSW = 10.
	The appropriate keyword name is then moved to the field PARALIST in the COMMX area.
LABELS	Set switch CDSW = 04.
	If keyword = YES, set LABEL (in COMMX) = 00.
	If keyword = NO, set LABEL = 01.
	If keyword = ALL, set LABEL = 20.
	If keyword = ONLY, set LABEL = 10.

DIRECTORY PRECOMPARE FOR PARTITIONED DATA SETS

For partitioned data sets (PDSs), the program first makes a comparison of the member names in the two input data set directories; the comparison is on the basis of true names rather than aliases. If one PDS directory has no counterpart true names to correspond to those in the other directory, the member records are not compared. To perform the member name comparison, the program allocates storage for tables that will contain the true names of both data sets. Storage is also allocated for note list pointers (TTRNs) for each member of both data sets.

Therefore, after preliminary initialization related to moving and storing ddnames and user exit routines (if there are any of the latter) as indicated in Figure 19 on page 33, module IEBCMAIN issues a GETMAIN macro instruction for space in which to analyze the SYSUT1 data set's directory. Figure 20 on page 34 shows the representation of this space (or buffer) in storage and the communication region's pointer to the space. The diagram also illustrates the movement of SYSUT1 directory information into the buffer.

IEBCMAIN checks each member name in each directory block to determine the name's status as either true or alias. The count of all true names thus found is recorded in the work area for the module. The last name in a PDS directory is X'FFFFFFFFFFFFFFFF' so that the end of a directory can be detected.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Using the number of member true names (each of maximum size = 8 bytes) on SYSUT1, the program determines the amount of storage required to contain the true names. To contain the TTRs that may be associated with each member name, IEBCMAIN obtains storage for up to three 4-byte TTRs for each member name.

The module then compares the member names (which have been read into the buffers) from each data set's directory. For equal true names, the member name is placed into the member name table (MNT), and any corresponding TTRs are entered in the TTR table for each data set. User data fields for both true member names and aliases are also compared. Figure 21 on page 35 shows the storage buffers and table areas that are used in the directory comparison process. The diagram also shows the pertinent pointers in COMMX and in the module's work area.

If one data set has a true name, and the other data set has the same name or an alias, members will not be compared, but a message indicating ALIAS/TRUE name failure will be given.

When two member names are compared and found to be unequal, the name with the lower binary value is printed on the SYSPRINT data set and is assumed to be missing from the other data set. The program also prints the user data fields for the names that do not compare equally.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

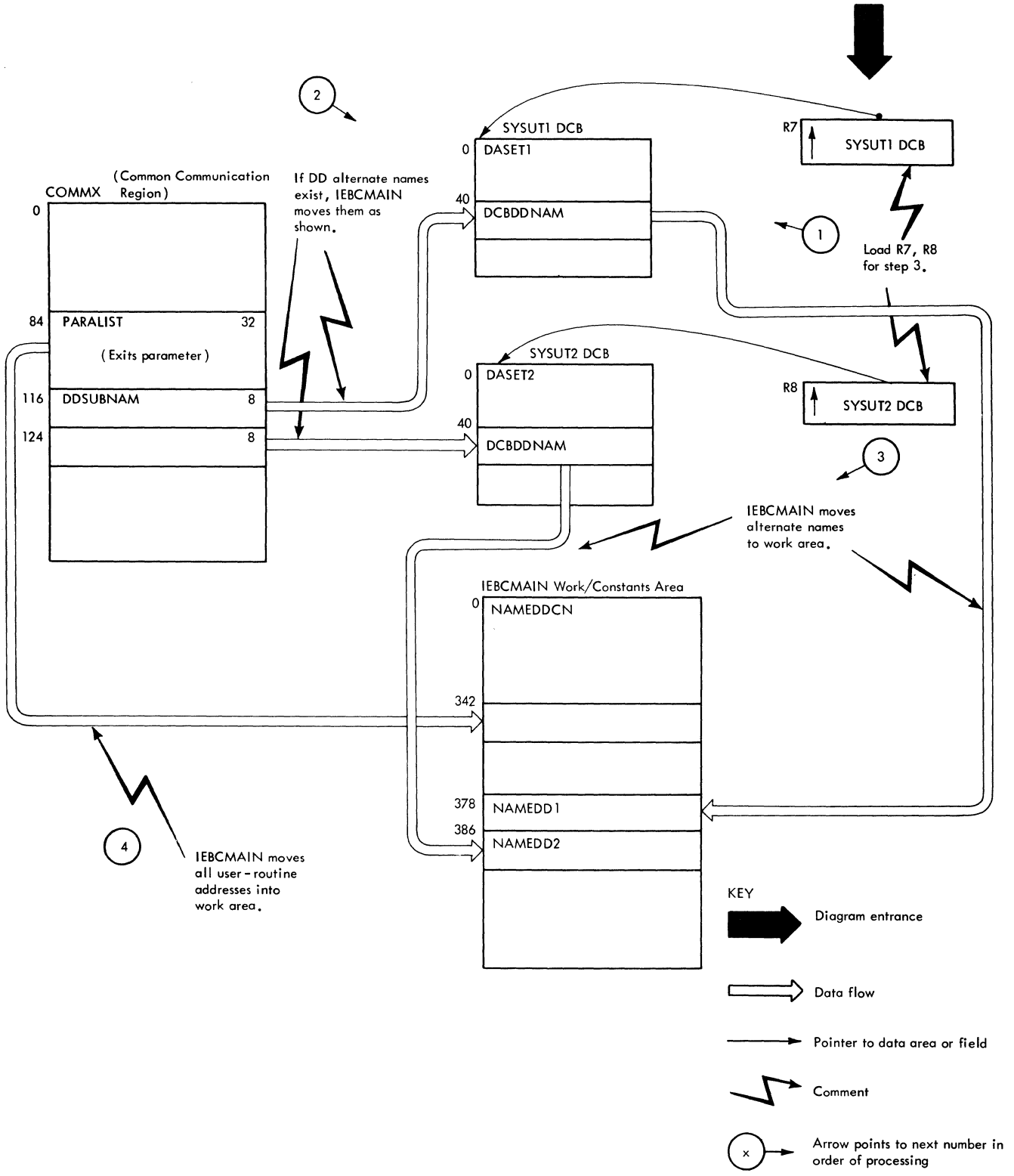


Figure 19. Moving ddnames and a Parameter List

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

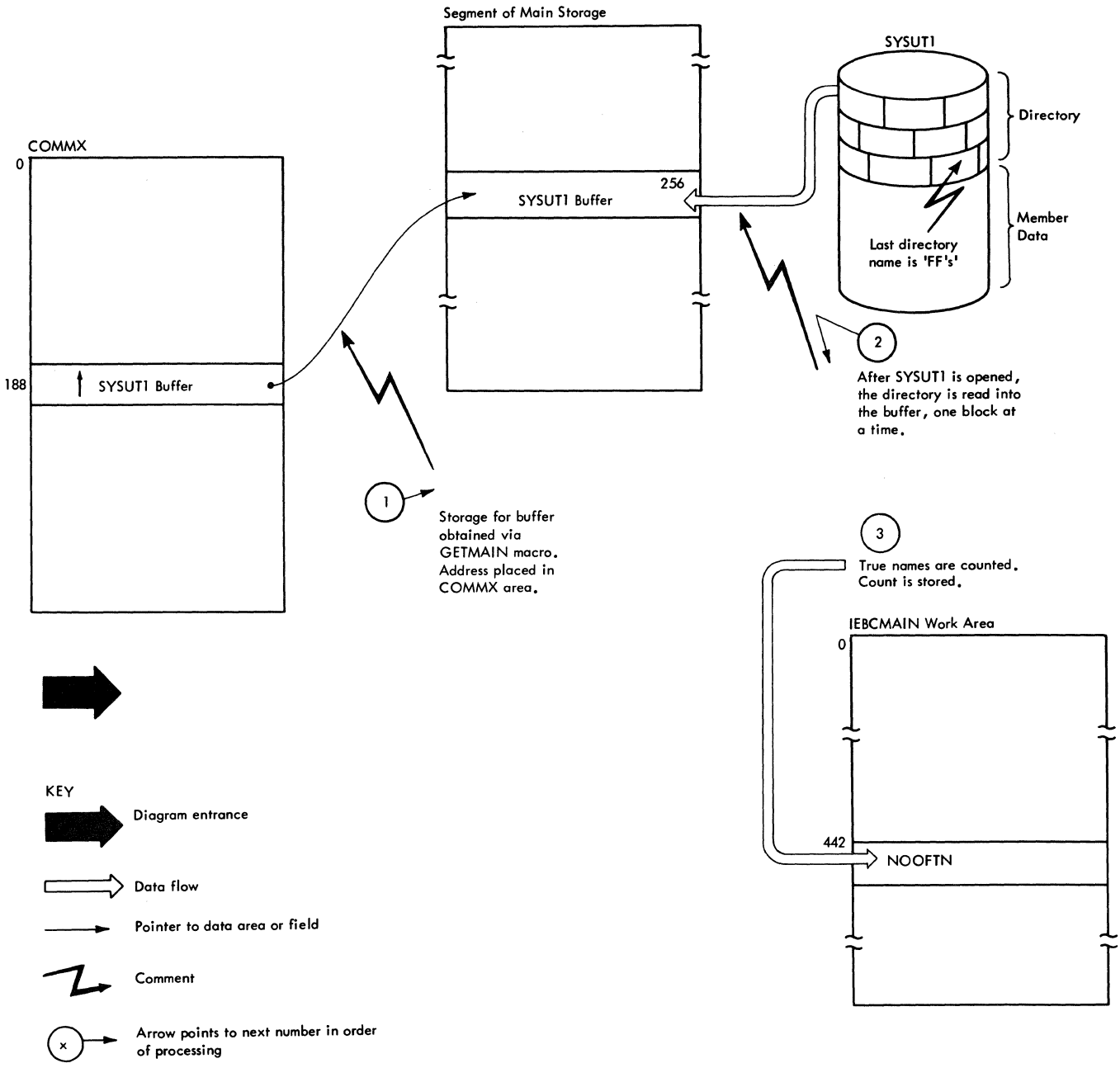


Figure 20. Obtaining Count of True Names on a Partitioned Data Set

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

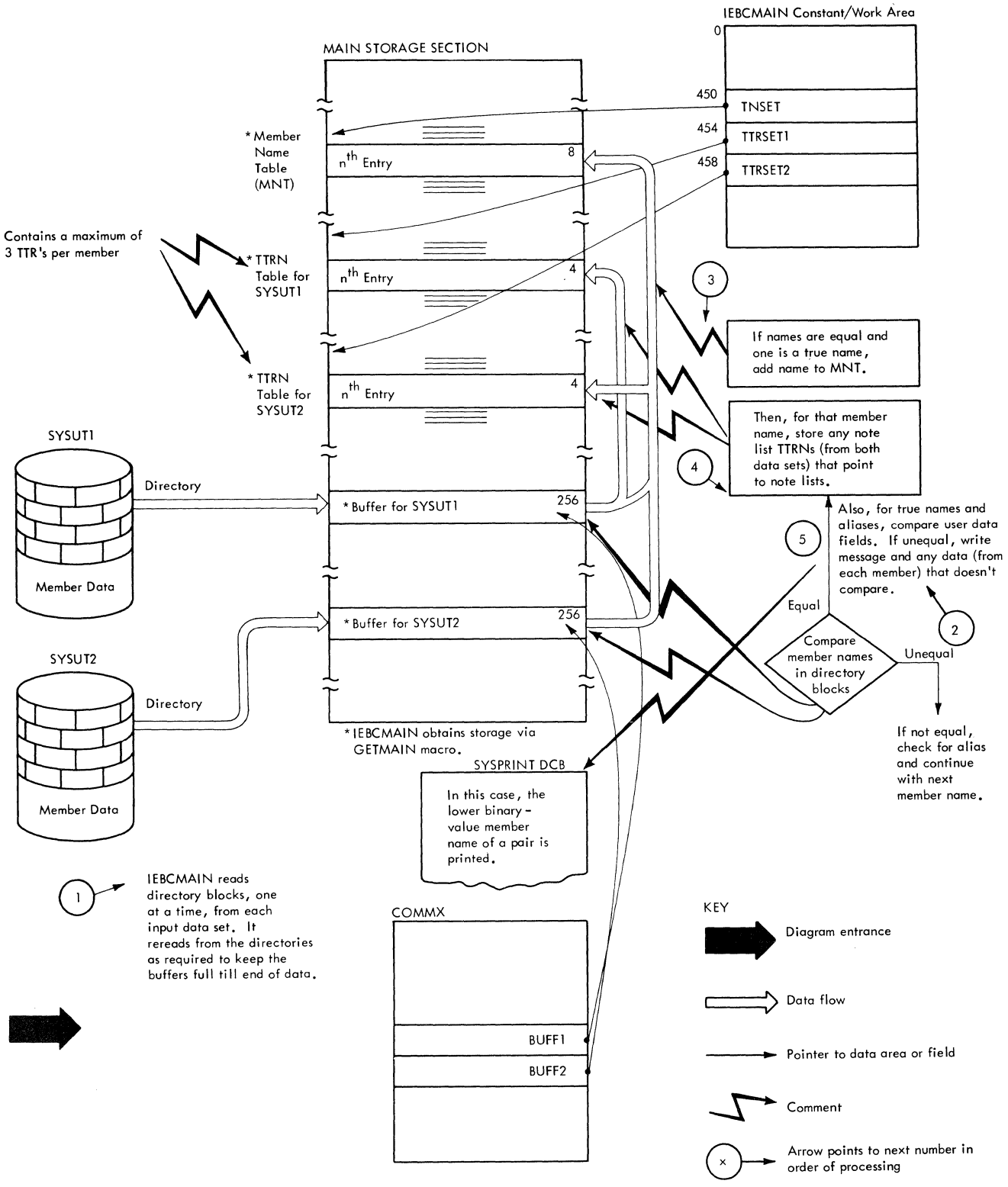


Figure 21. Building the Member Name Table and TTRN Tables

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

COMPARING INPUT LABELS FOR SEQUENTIAL DATA SETS

Module IEBCULET obtains an area of storage for header and trailer labels and saves the labels in that area. If the user has not requested the processing of labels as data, or if no user exit routines have been requested, this module is not required. The labels are compared by module IEBCMAIN. If a user's program specifies exits to its own label processing routines, the exits are made from IEBCULET. Control statements EXITS and LABELS apply, respectively, to the requesting of user label processing routines and to the treatment by IEBCMAIN of user labels to be compared as data. When label comparison for the input data sets is requested, the header labels are saved before the labels are compared. Trailer labels are saved and compared after header labels have been completely processed and all data has been compared.

The program uses a LOAD macro instruction to bring the user label routines into storage. The data sets are opened and some of the DCB parameters are moved into COMMX for reference. During the opening of the data sets, an exit is taken, via the DCB exit list to module IEBCULET, and from there to user routines that are present to process labels. IEBCULET saves in storage any user labels that are to be compared as data (see Figure 22 on page 37).

When comparing labels, as shown in Figure 23 on page 38 and Figure 24 on page 39, IEBCMAIN successively compares an 80-byte label from each of the input data sets. The labels that do not compare equally, and any surplus labels on either data set, are printed on the output data set. Each input data set may have up to eight header and/or trailer labels. The DCB exit routines at open time (for header labels) and at end of data (EOD) time (for trailer labels) are entered once for each label to be saved.

SAVING INPUT LABELS

If a user has requested processing of labels as data, module IEBCULET will receive control from module IEBCMAIN. For each input data set, either IEBCMAIN has issued an OPEN macro instruction or an EOD condition has been reached, depending on the state of the processing. During the processing by the macro instruction or during the EOD processing, an exit to IEBCULET is taken via the exit list entry for the type (header or trailer) of input label currently being handled.

IEBCULET checks the LABSW field in the COMMX region to determine which labels are being saved (see Figure 22). The module then issues a GETMAIN macro instruction for space to contain up to eight 80-byte labels. The appropriate labels are saved one at a time in this space. After saving a label, IEBCULET returns control to the open (or EOD) routine to get another label. The address at which each label is to be saved is determined by adding, to the start of the label save area, 80 times the number of labels already saved.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

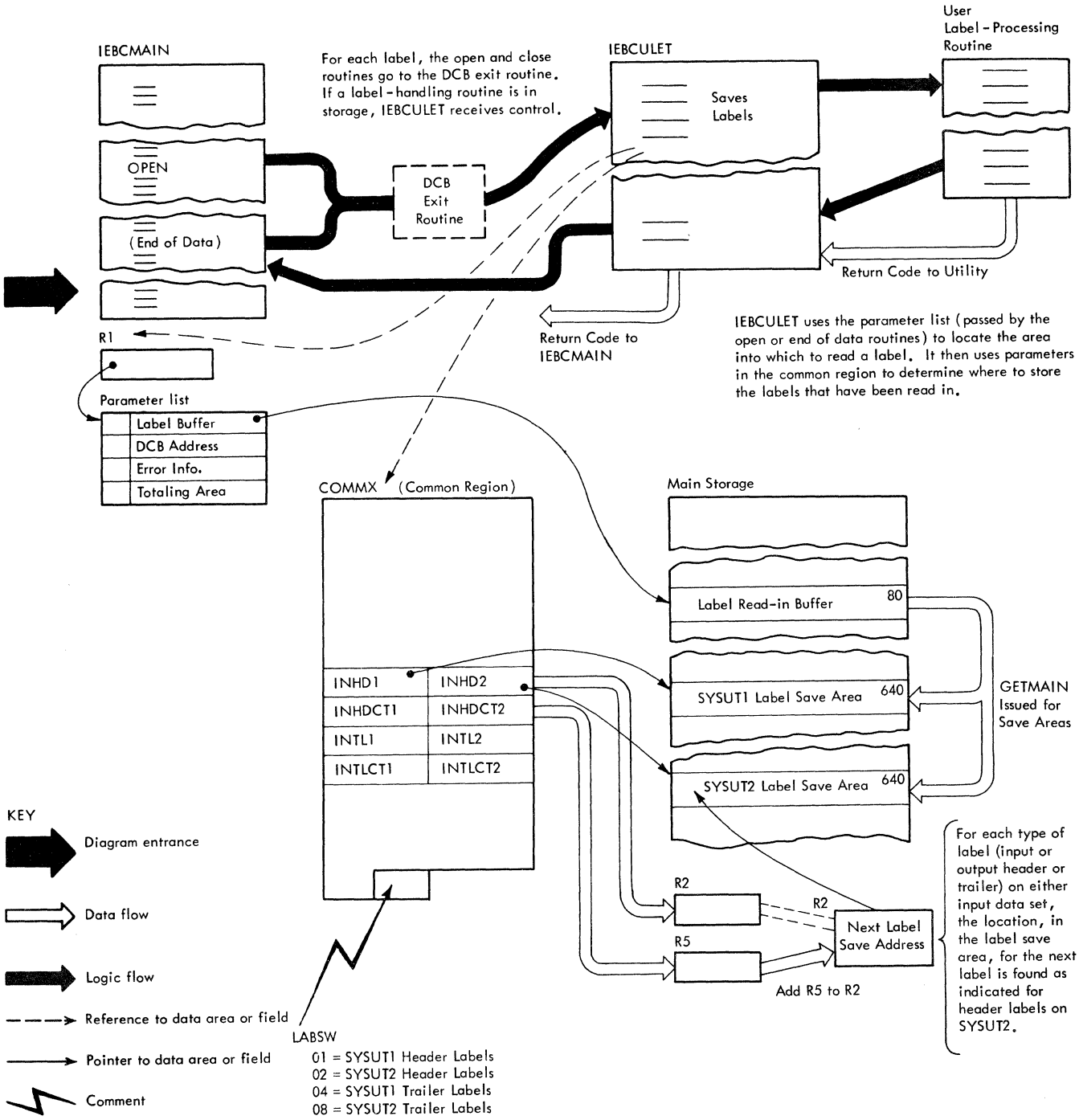


Figure 22. Input Label Saving and User Exits

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

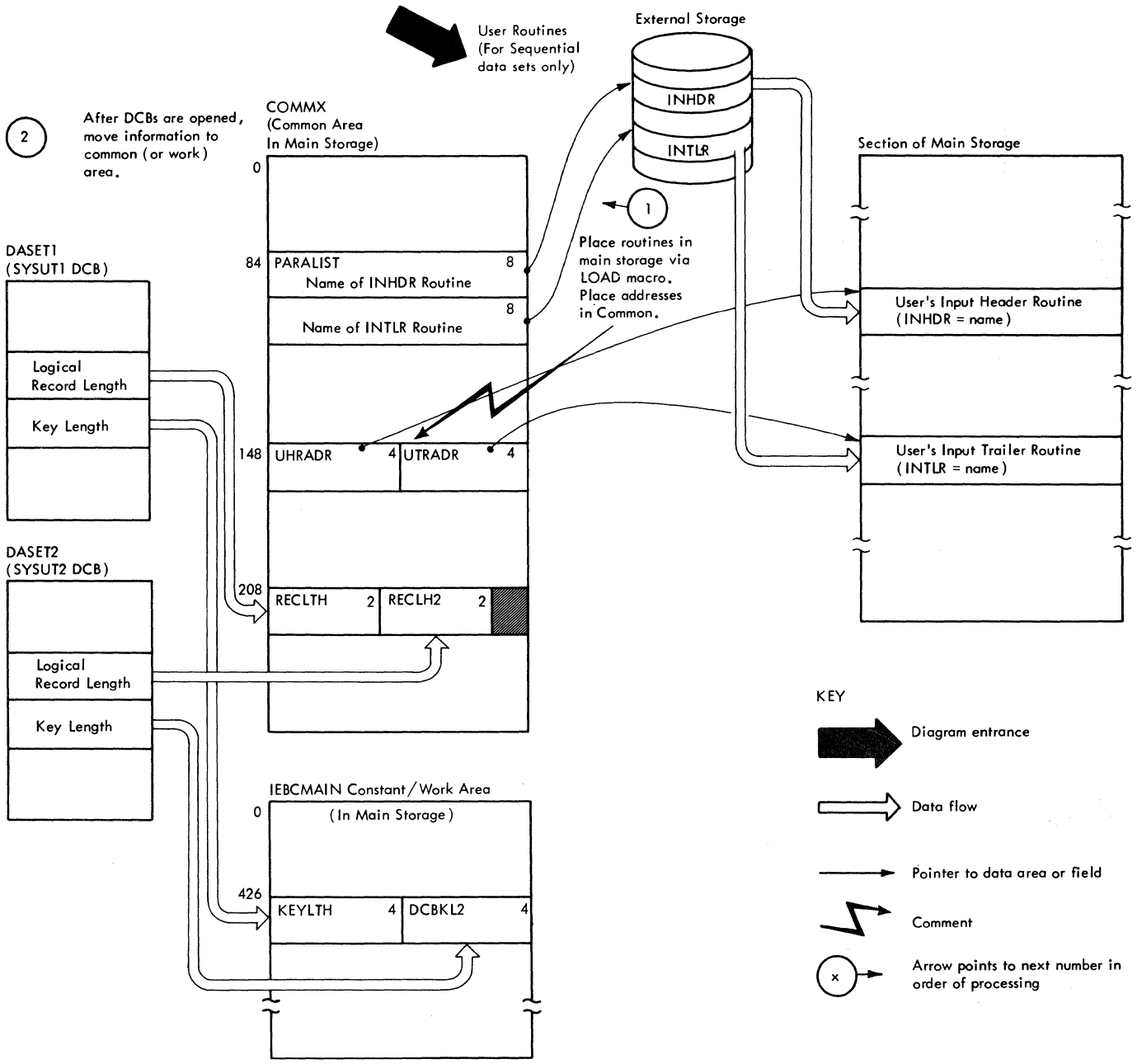


Figure 23. Moving User Label Exit Routines and DCB Information

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

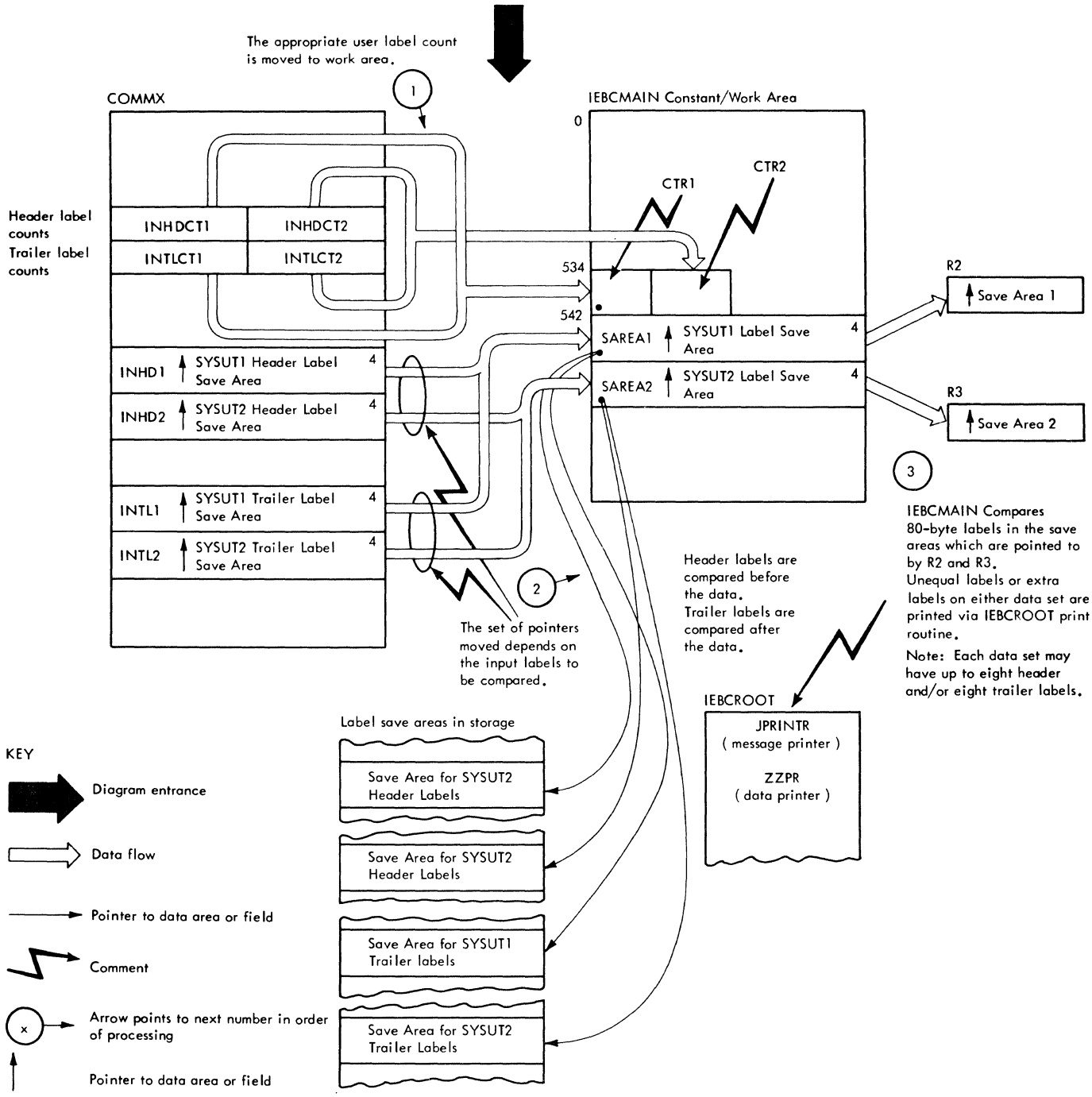


Figure 24. Comparison of Input Data Set Header or Trailer Labels (Sequential Data Sets Only)

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

COMPARING RECORDS OF BOTH INPUT DATA SETS

Module IEBCMAIN initializes the input data set DCBs with any alternate ddnames that may have been supplied. It also sets switches to indicate any user routines that may have been specified for the processing labels, and it opens the input data sets (SYSUT1 and SYSUT2) for processing by BSAM. If QSAM is to be used for processing the input data sets, module IEBCQSAM will open the data sets.

Depending on the type of records, either module IEBCMAIN will use the READ macro instruction or module IEBCQSAM will use the move mode of the GET macro instruction to obtain records for the input buffers. Records in the buffers are compared one character at a time. Records that are longer than 256 bytes are compared in increments of 256 bytes or less. If a user has specified a precompare routine, the routine will be given control (via a LINK macro instruction) prior to the record comparison process.

INPUT RECORD COMPARISON

Before actually comparing records from the two input data sets, module IEBCMAIN determines record characteristics and gives control to the routine that uses the appropriate access method. If BSAM processing is required (see Figure 25), IEBCMAIN uses its own subroutine, INITBSAM. For QSAM processing, IEBCMAIN transfers control to module IEBCQSAM.

The access method obtains a block of records from each input data set and places each block in the appropriate input buffer in main storage (see Figure 26 on page 41 and Figure 27 on page 42). As the buffers become empty during record-comparison processing, new blocks of data are brought into the buffers. The record processing routine deblocks the records in the input buffer if necessary and gives control to the COMPAR subroutine of module IEBCMAIN. (Deblocking is done only for data set records that are less than or equal to 32 756 bytes in length.)

Level of Comparison	Data Set Organization	Data Set	Records Have Keys	RECFM			Logical Record Length	Access Method
Physical Block	Sequential	SYSUT1 SYSUT2	Yes Yes	VS VS			Not a factor	BSAM
Physical Block	Sequential	SYSUT1 SYSUT2	No No	VS VS			Greater than 32,756 bytes in at least one data set.	BSAM
Logical Record	Sequential	SYSUT1 SYSUT2	No No	VS VS	VS V	V VS	Equal to or Less than 32,756 bytes for both data sets.	QSAM
Logical Record	Sequential or Partitioned	SYSUT1 SYSUT2	Not a Factor	F F	U U	V V	Less than 32,756 bytes for both data sets.	BSAM

Figure 25. Access Methods Used to Obtain Records for Comparison

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

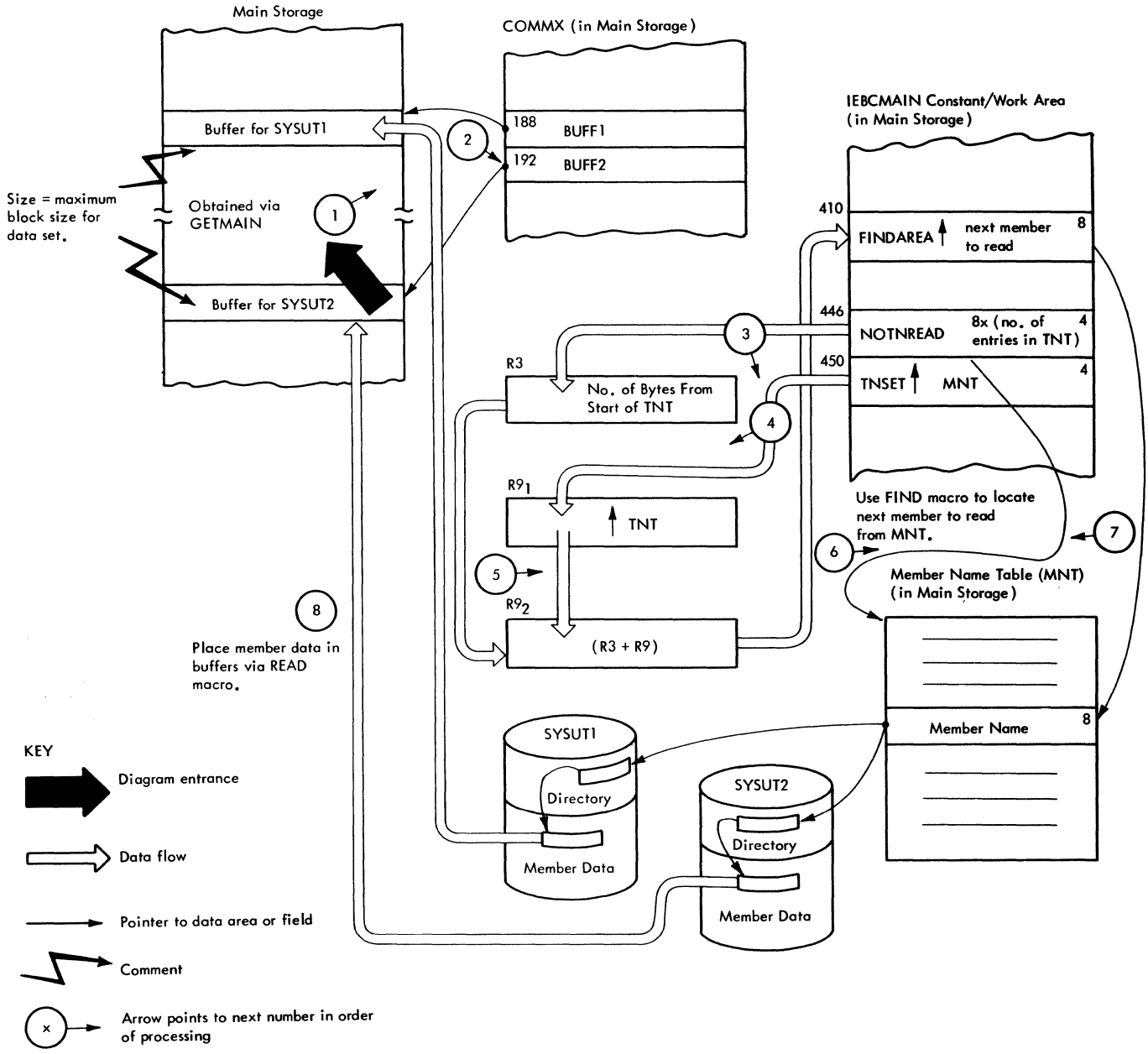


Figure 26. Locating and Reading Members on PDS via INITBSAM

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

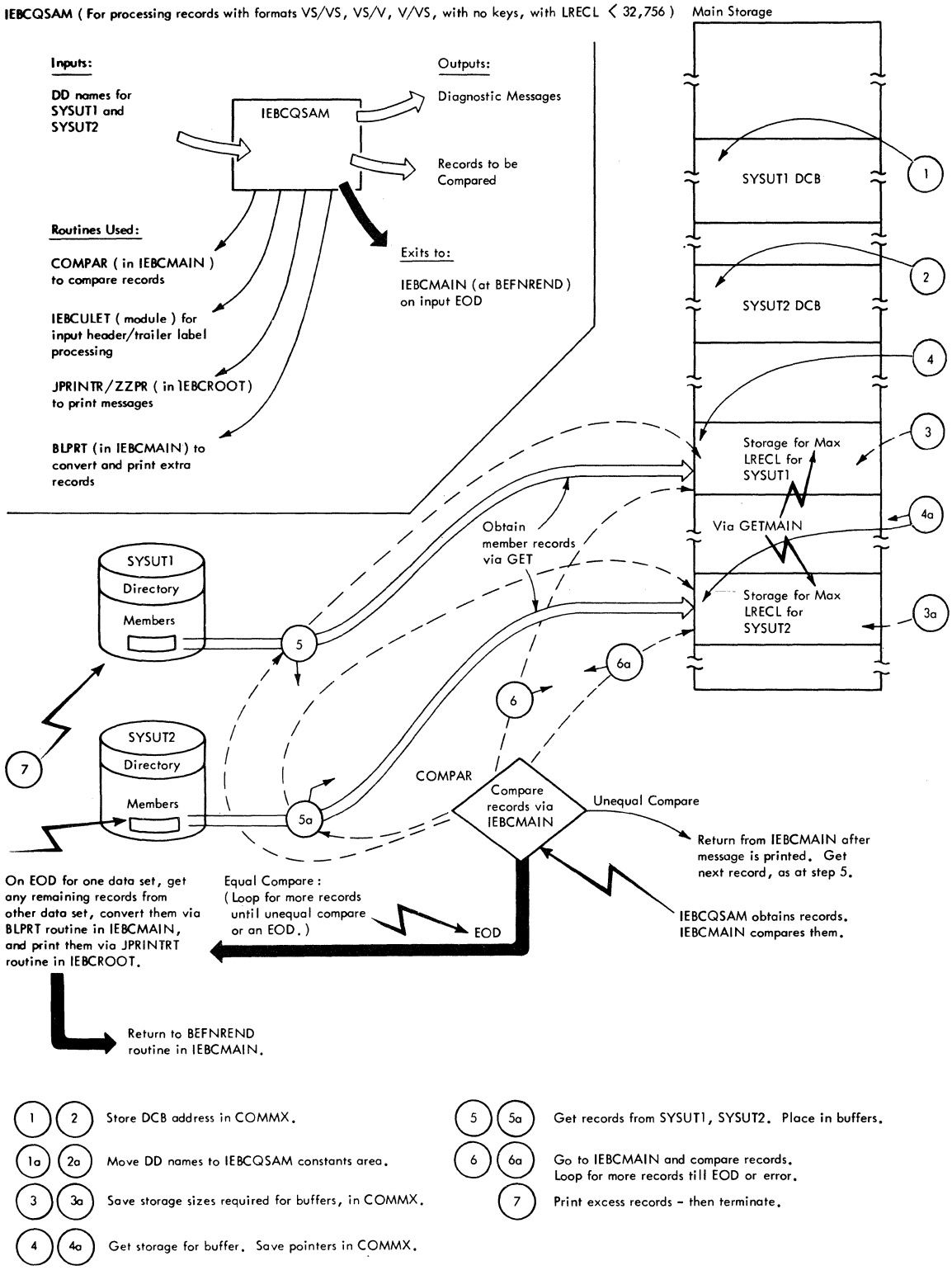


Figure 27 (Part 1 of 2). QSAM Processing of Records to Be Compared

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

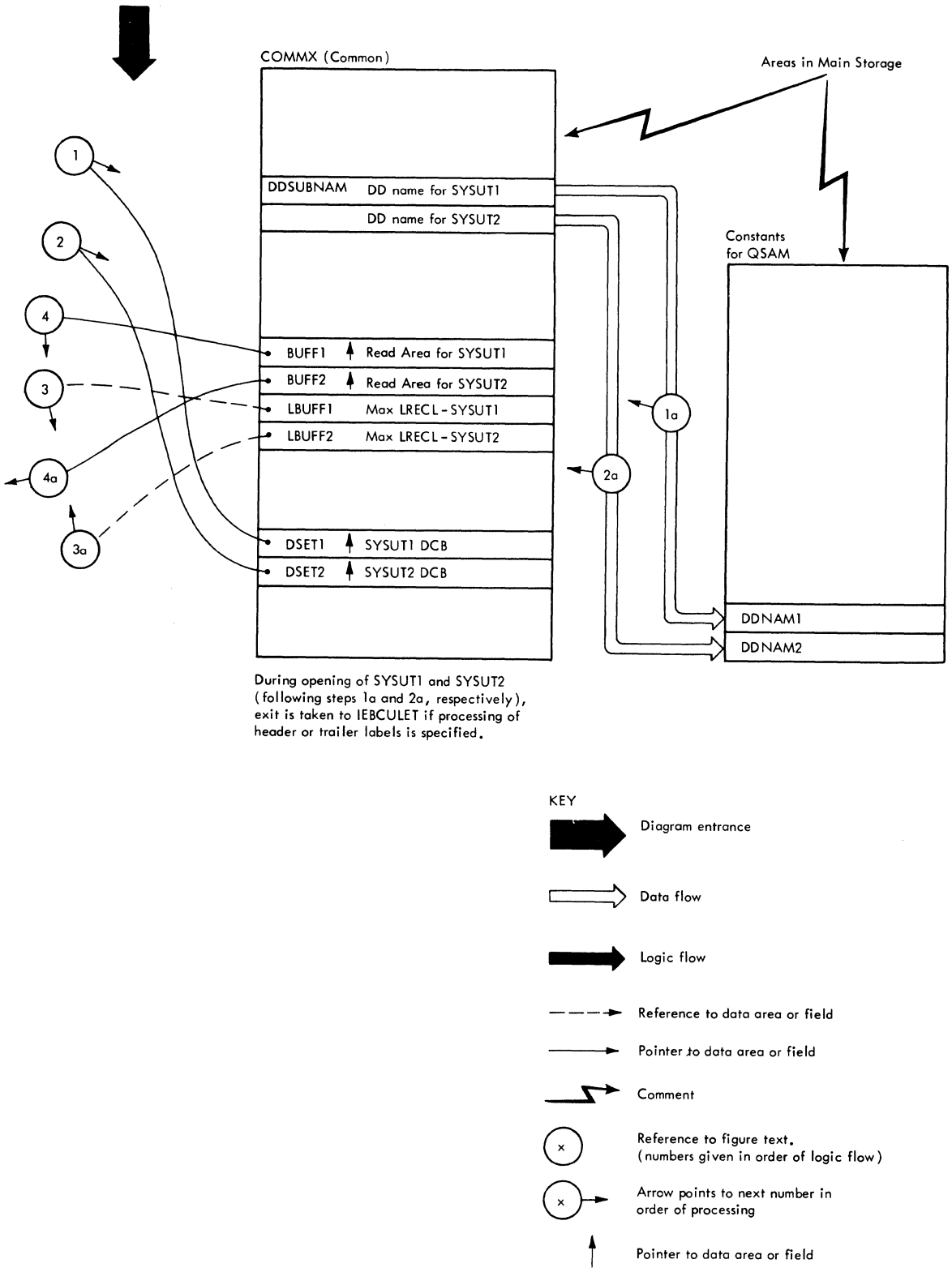


Figure 27 (Part 2 of 2). QSAM Processing of Records to Be Compared

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

At this point, if one has been specified, the module gives control to a user's precompare routine. If the return code from the user's routine does not indicate a request for program termination, or if there is no user precompare routine, the COMPAR subroutine compares one record from each input data set. The comparison is done one character at a time, and in increments of 256 bytes or fewer.

If the records compare unequally, the program prints both records. One of the following choices is then taken:

- If the user has specified an error routine, module IEBCMAIN gives control to that routine. At the conclusion of the routine, the user's return code is stored and termination processing is performed.
- If the data sets are sequentially organized and there is no user error routine, then either:
  - The program obtains the next record in each data set and compares them if the current unequal compare is less than the 10th encountered, or
  - Termination processing is performed if the current unequal compare is the 10th encountered.
- If the data sets are partitioned, and there is no user error routine, and if the current unequal compare is less than the 10th encountered for a given member, the program obtains the next record from each member and compares them.
- If the data sets are partitioned, and there is no user error routine, and if the current unequal compare is the 10th encountered for a given member, then either:
  - The next members are obtained and processing continues with the next set of records to be compared if the current members are not the last ones in their data sets, or
  - Termination processing is performed if a current member is the last one in its data set.

After processing all available records and printing any extra records that may exist in one of the data sets, module IEBCMAIN closes the input data sets and prints the number of records that have been compared.

MESSAGES AND HEADERS

The output data set, SYSPRINT, may contain any of the following information:

- Diagnostic (error) messages
- Informational messages
- Names of records that compare unequally
- Extra records that exist on one data set after all records on the other data set have been compared
- Member names (from a PDS directory) that compare unequally
- User data fields for members (true or alias) that compare unequally
- Input labels that compare unequally

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Information is printed by the routine ZZPR in module IEBCROOT, which is in storage throughout execution of the program (see Figure 28 on page 46). To print messages, the module refers to CSECT MESSLIST in module IEBCOMPM. This CSECT contains messages that are up to 72 bytes long (including an 8-byte identifier). All modules of the program use the print routine when there is output to be written.

Upon entry to routine ZZPR to have a message printed, register 0 contains the length (either 32 or 64 bytes) of the message, and register 1 contains the address (or number) of the message. Subroutine JPRINTRT, also of module IEBCROOT, determines whether the message is of 32 or 64 bytes. When IEBCROOT saves registers 14 through 3 in the ZZSAVE field of the module's work area, the third byte of register 0 occurs as the first byte of the field ZZSW. The program can reference this indicator to determine whether the output is to be a header or a diagnostic message.

When the print module puts page numbers on the output data set, it places them in ascending sequence, starting with the standard number (0001) or the user's assigned first page number.

As the print routine puts out messages, it increases the line count for each page. To determine whether to begin a new page before each message, the line count is compared with a constant (X'32'), which the ZZPR routine has moved into the message writer's work area. During the writing of output, the program sets the line control character, initially at zero, to values indicating the line spacing required.

For printing the header on each page of output, routine ZZPR moves the header, with updated page number, into the print buffer (field ZZPRINT in the work area) and issues a PUT macro instruction.

TERMINATION PROCESSING

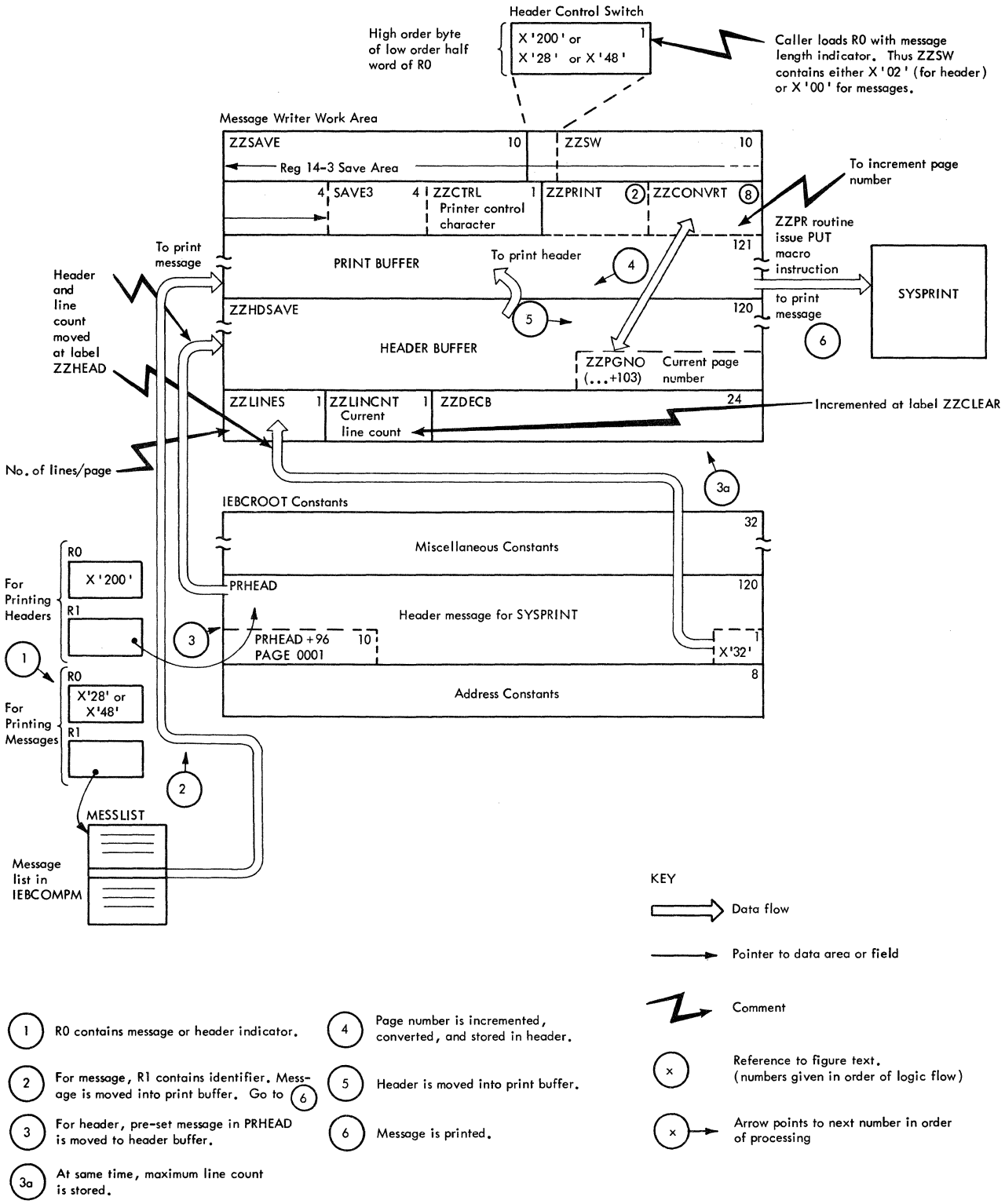
For normal terminations (those involving a return code less than 12), module IEBCMAIN frees any main storage areas that have been used for buffers, tables, work areas, or user routines, and returns control to the calling program.

For abnormal terminations (see Figure 29 on page 47), either module IEBCANAL or module IEBCMAIN sets the proper return code, requests module IEBCROOT to print an error message, frees storage areas as described above, and then returns control to the calling program.

WORK AREAS

This section contains layouts of the three major work and communication areas of the IEBCOMPR program (see Figure 30 on page 48, Figure 31 on page 49, and Figure 32 on page 50). For each area, the sizes (in bytes) of fields are given in the upper right corner of each field, and the decimal and hexadecimal offsets are given for fields beginning along the left side of the area. In most cases, the content of each field is given, and the field's label is shown.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1



1. R0 contains message or header indicator.
2. For message, R1 contains identifier. Message is moved into print buffer. Go to 6.
3. For header, pre-set message in PRHEAD is moved to header buffer.
3a. At same time, maximum line count is stored.
4. Page number is incremented, converted, and stored in header.
5. Header is moved into print buffer.
6. Message is printed.

Figure 28. Printing Messages and Headers

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Description	Return Code	Message Number (xx of IEB2xxI)
Control statement is invalid.	12	01
The size of a PDS directory entry is not between 14 and 256 bytes, inclusive.	12	02
True names in one PDS do not have counterparts in the other PDS.	12	10
Records in both data sets do not have same key length.	12	11
One or both data sets do not have standard record format.	12	12
The input stream has two COMPARE or LABEL commands.	12	13
Fixed record lengths of both data sets are unequal.	12	14
SYSUT1 and SYSUT2 data sets have different record formats.	12	15
First control statement is not COMPARE.	12	16 or 64
A V or VS record has 'LL' value that is not between 5 and 32,756 bytes, inclusive. ('LL' = logical record length)	12	17
Permanent I/O error detected by FIND macro during PDS directory search.	12	18
A V or VS record has 'LL' value that is not between 9 and 32,760 bytes, inclusive. ('LL' = blocksize)	12	19
Extra records on SYSUT1 or SYSUT2	8	23, 24
A user's routine requests termination of the program.	12, 16	25
	16	57
Ten consecutive unequal record comparisons occur as follows:		
On a sequential data set	12	27
On a member of a partitioned data set	8	27
A DD NAME is not present.	12	29
The SYSIN blocksize is invalid. (It should be a multiple of LRECL.)	12	30
An uncorrectable I/O error occurs on reading user input header or trailer labels on either SYSUT1 or SYSUT2.	12	43, 44, 45, or 46
User specifies DATAONLY so that only header labels are processed.	0	49
User labels are not supported for PDS.	12	50
One (but not both) input data set has 'LL' >32K bytes.	12	51
One (but not both) input data set has keyed VS records.	12	52
IEBCOMPR program does not support PDS with VS records.	12	56
Program is unable to take a user's exit to process labels as data.	12	58
Analysis program detects an invalid or a missing keyword, command word, parameter, or an error on a continuation statement.	12	59, 60, 61, 62, 63, or 65
A continuation statement begins in wrong column.	12	66
An I/O error occurs during processing of a data set.	12	67

Figure 29. Conditions for Termination of IEBCOMPR Utility Program

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

COMMX

Offset Hex	Offset Dec						
		SAVE1315					12
0	0	Save Area for Reg 13-15					
C	12	SAVEREGS					72
		Register Save Area					
54	84	PARALIST					32
		Parameter List for Exits					
74	116	DDSYSIN Alternate Name for SYSIN					8
7C	124	DDSUBNAM Alternate Name for SYSUT1					8
84	132	Alternate Name for SYSUT2					8
8C	140	HICONCODE 2	CCADSORG 1 PO or PS Switch	ABORTSW 1 Abort Switch	LABEL 1 Label Parameter	UEXIT 1 Label Exit Ind.	2 Reserved
94	148	UHRADR 4 Address of Input Header Exit			UTRADR 4 Address of Input Trailer Exit		
9C	156	INHD1 4 Addr of SYSUT1 Header Label Save Area			INHD2 4 Addr of SYSUT2 Header Label Save Area		
A4	164	INHDCT1 4 No. of Input Header Labels for SYSUT1			INHDCT2 4 No. of Input Header Labels for SYSUT2		
AC	172	INTL1 4 Addr of SYSUT1 Trailer Label Save Area			INTL2 4 Addr of SYSUT2 Trailer Label Save Area		
B4	180	INTLCT1 4 No. of Input Trailer Labels for SYSUT1			INTLCT2 4 No. of Input Trailer Labels for SYSUT2		
B6	188	BUFF1 4 Address of Read Area for SYSUT1			BUFF2 4 Address of Read Area for SYSUT2		
C4	196	LBUFF1 4 Length of Buffer to hold SYSUT1 Record being Processed			LBUFF2 4 Length of Buffer to hold SYSUT2 Record being Processed		
CC	204	LORECO 4 Number of Logical Records			RECLTH 2 Length of Logical Record-SYSUT1	2 Length of Logical Record-SYSUT2	
D4	212	DSET1 4 SYSUT1 DCB Address-IEBCQSAM			DSET2 4 SYSUT2 DCB Address-IEBCQSAM		
DD	220	INPUTLOC 4 Input Buffer Address-BLPRT Routine			NOEXITSW 1 User Routine Ind.	TERMSW 1 Termination Sw.	EXIT1SW 1 INHDR Exit	EXIT3SW 1 INTLR Exit
E4	228	VSSWT 1 BSAM vs QSAM	NOLB 1 No Label Switch	LABSW Type 1 of Label Proc.	PRINTOUT 4 Beginning of SYSPRINT DCB		

Figure 30. Common Communication Region for IEBCOMPR

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

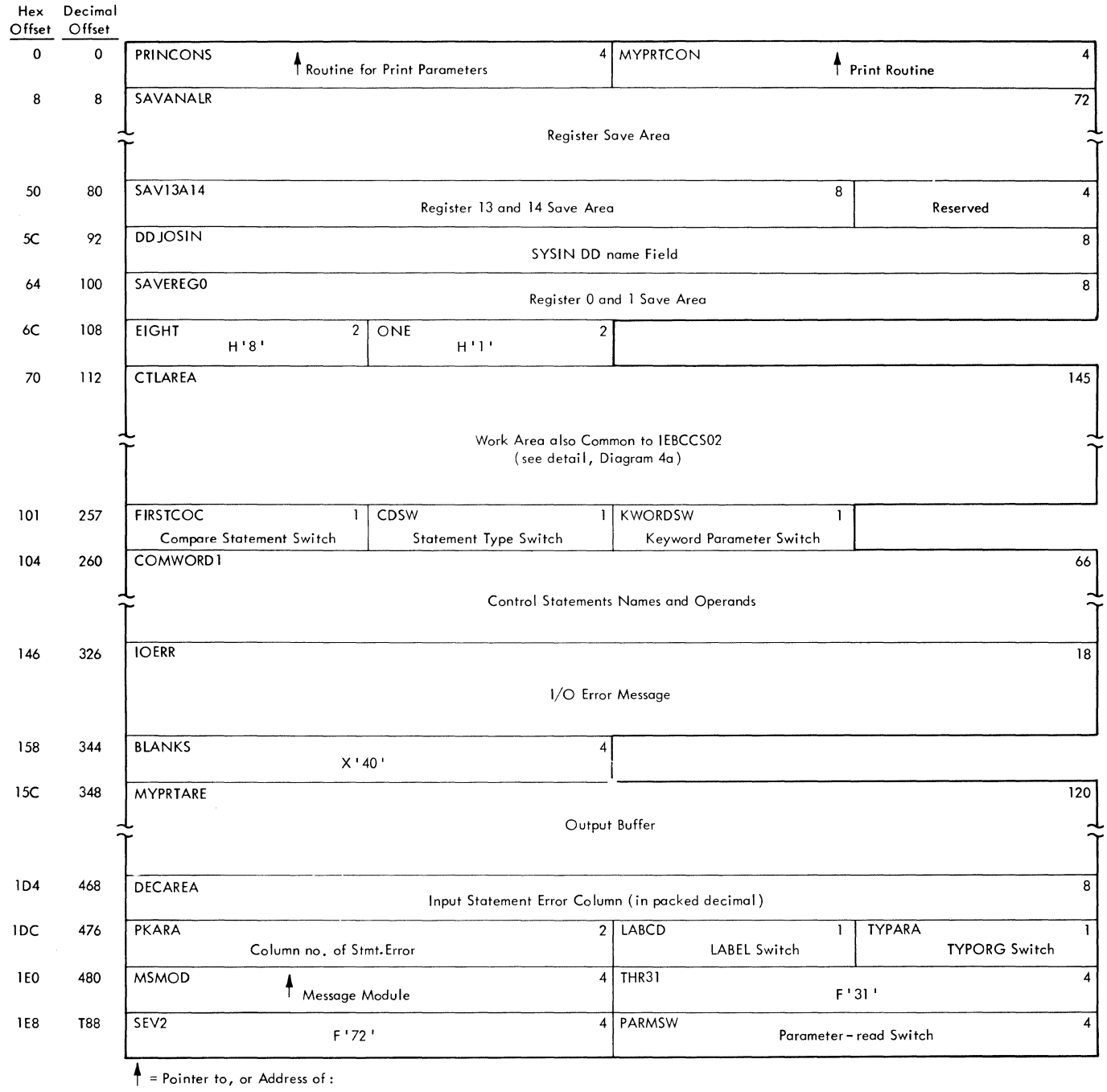


Figure 31. Constants and Work Area for IEBCANAL

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Offset Hex	Offset Dec						
0	0	NAMEDDCN 9 'DDNAME='	MEMRNAME 14 'MEMBER NAME='				
17	23	ALIANAME 14 'ALIAS NAME='	PHYSRENU 25 'PHYSICAL RECORD NUMBER='				
3F	63	LOGWIPHY 47 'LOGICAL RECORD WITHIN PHYSICAL RECORD='					
6D	109	1 Reserved	PRINCONS 4 V(JPRINTRT)	MYPRTCON 4 V(ZZPR)			
76	118	NODDNAME 4 ↑ Unopened DD name	BUFSW1 1 Buffer 1 Empty Sw.	BUFSW2 1 Buffer 2 Empty Sw.			
7C	124	ONE 2 H'1'	TWO 2 H'2'	FOUR 2 H'4'	EIGHT 2 H'8'	TEN 2 H'10'	TWELVE 2 H'12'
88	136	2 Reserved	SIXTY 4 F'60'	TWOFS 4 F'256'	MASK1 4 F'31'	MASK2 4 F'96'	
9A	154	0000FFFF 4 Block Comparison Constant	000100FE 4 Block Comparison Constant	ALLFS 8 X'FF'			
AA	170	SAVEREG2 4	SAVEREG3 4	SAVEREG4 4			
B6	182	SAVEREG6 4	SAVEREG9 4	CSAVE Save Reg 2-8			
		28 (continued)	DIROPOK 1 Directory Block Read Switch	EOD1SW End of Data for PDS on SYSUT1 1	EOD2SW End of Data for PDS on SYSUT2 1	BLANKS 1	
DE	222	MYPTAREA 120 Print Buffer					
156	342	PARATABLE 8 Header Exit Parameter List	PARATBLX 8 Trailer Exit Parameter List	PARATBLY 8 Error Exit Parameter List			
172	370	PARATBLZ 8 Pre-compare Exit Parameter List	NAMEDD1 8 DD name for SYSUT1 Data Set	NAMEDD2 8 DD name for SYSUT2 Data Set			
18A	394	PTAREA 8 Unpacked No. Area	CONVTDEC 8 Conversion Work Area	FINDAREA 8 Member Name for FIND Macro			
1A2	418	MAXBLLH1 4 Block Size for SYSUT1	MAXBLLH2 4 Block Size for SYSUT2	KEYLTH 4 Key Length for SYSUT1 Records			
1AE	430	DCBKL2 4 Key Length for SYSUT2 Records	TNTABLEL 4 Length of Member Name Table	TTRTABLL 4 Length of TTR Table			
1BA	442	NOOFTN 4 Number of True Names in Table	NOTNREAD 4 8 Times Number of True Names Already Read	TNSET 4 Address of Member Name Table			
1C6	454	TTRSET1 4 ↑ TTR Table for SYSUT1	TTRSET2 4 ↑ TTR Table for SYSUT2	PHREC01 4 Physical Record Count for SYSUT1			
1D2	466	PHREC02 4 Physical Record Count for SYSUT2	OUTSAV Save Area Used by Hex-EBCDIC Conversion Routine 4	BLKLTH1 2 Physical Block Length for SYSUT1	BLKLTH2 Physical Block Length for SYSUT2 2		
1DE	478	TTRLTH 2 Length of TTRNs for SYSUT1	TTRLTH2 2 Length of TTRNs for SYSUT2	USERDAL No. of User Data Half Words-SYSUT1 2	USERDAL2 No. of User Data Half Words-SYSUT2 2		
1E6	486	ERRCO 2 Error Counter	KEYERCT 2 Count of "Key" Errors	LOREPP1 No. of Logical Recs. per Phys. Rec-SYSUT1 2	LOREPP2 No. of Logical Recs. per Phys. Rec-SYSUT2 2	LENSAV Save Area for Bytes to be Convd 2	
1F0	496	SWFORBP1 17 Seventeen Switches	EQSIGN 2 '='	3 Reserved	EIGHTY 4 F'80'		
20A	522	ADQSAM 4 V(QSAM)	ADULAB 4 V(USERLAB)	SAVE9 4 Save Area for Register 9	CTR1 No. of SYSUT1 Labels to Compare 4		
21A	538	CTR2 4 No. of SYSUT2 Labels to Compare	SAREA1 ↑ Save Area for SYSUT1 Labels 4	SAREA2 ↑ Save Area for SYSUT2 Labels 4	MAXLL X'7FF8' (Maximum Block Size) 2		
228	552	MAXLLL X'7FF4' (Maximum Record Length) 2	F14 2 H'14'	HD 7 'HEADER'	TL 7 'TRAILER'		
23A	570	LBSW 1 Header/Trailer Label Switch	IOERR 18 I/O Error Message	LABMSG 10 Label Message Identifier			
257	599	LABMSG1 62 Input Label Message					

Figure 32. Work Area Contents for IEBCMAIN

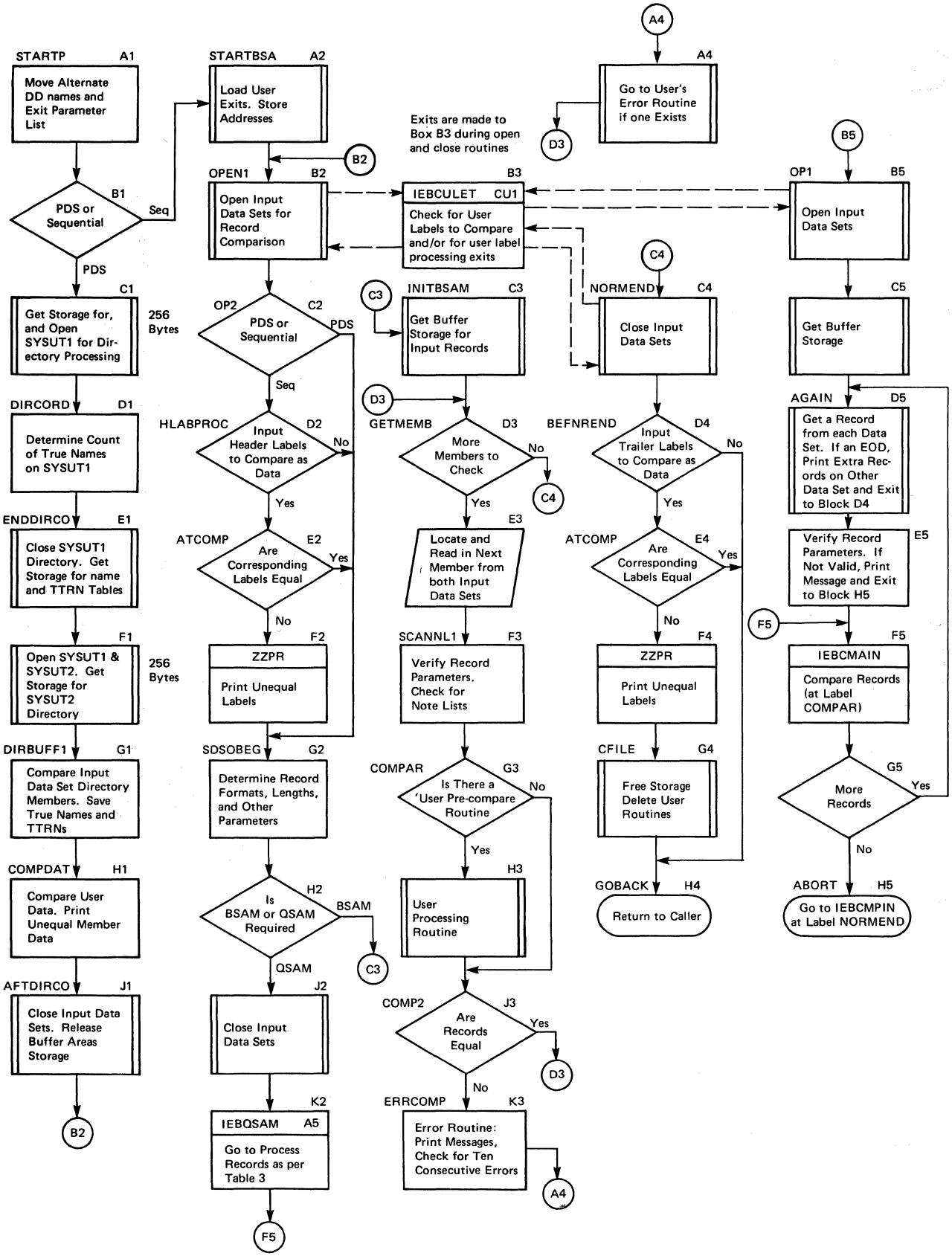
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEBCOMPR FLOWCHARTS

The following flowcharts represent the major comparison processes used by the IEBCOMPR program. Where possible, labels corresponding to the program listings have been incorporated.

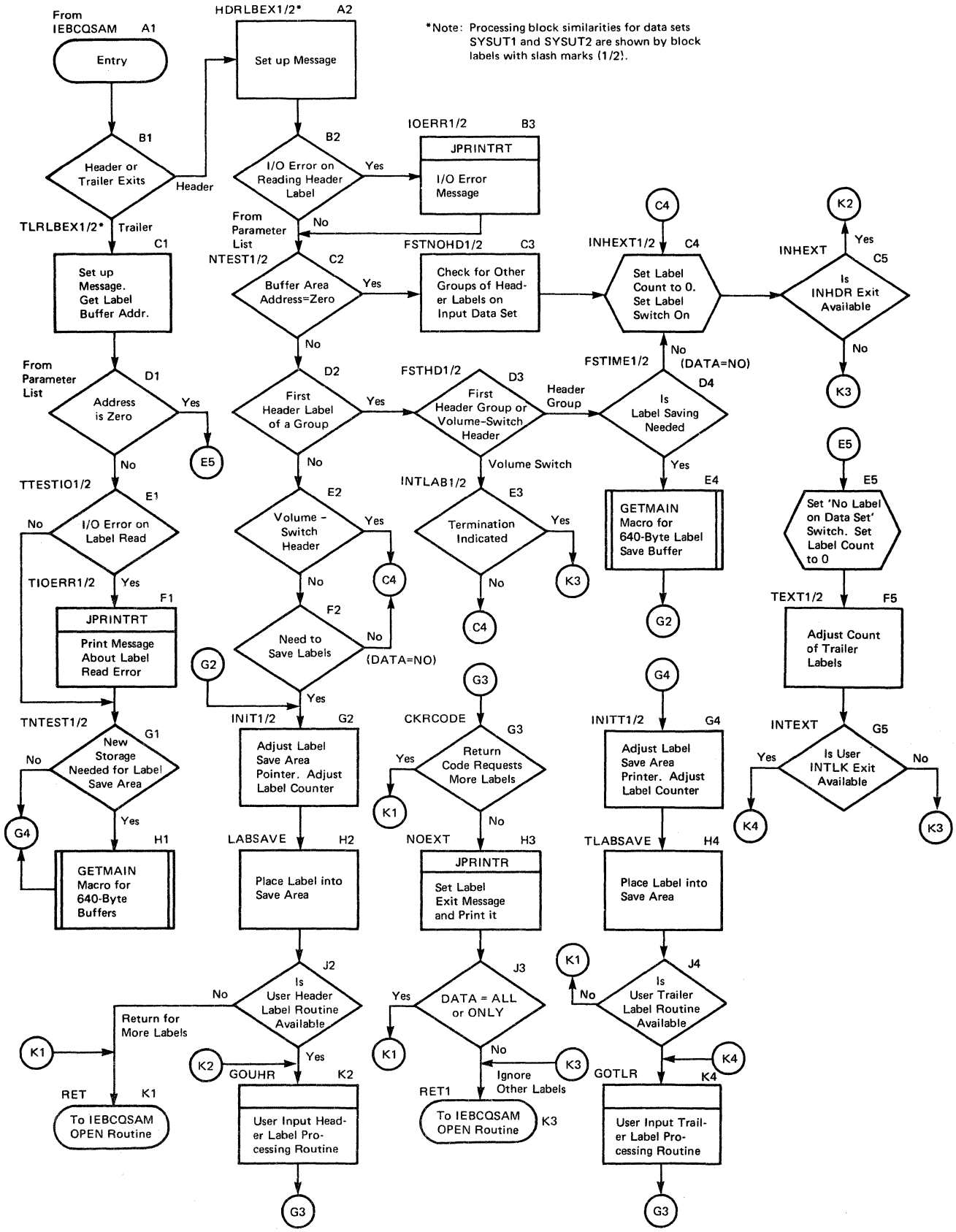
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 01. IEBCMAIN/IEBCQSAM MODULES



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 02. IEBCULET MODULE



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

The IEBCOPY program is used to:

- Copy or merge members of one or more partitioned or unloaded data sets
- Unload members from a partitioned data set onto a physical sequential data set
- Load an unloaded data set onto a partitioned data set

In particular, the program copies all members, or selects or excludes specified members, from one or more input data sets to be copied or loaded, or from one input data set to be unloaded. The program can rename specified selected members of an input data set. It can also, optionally, replace members of an output partitioned data set with identically named members from an input data set. If unused space 'gaps' exist on a data set, the program can be used to eliminate these gaps by performing a full copy of the data set to compress it in place. This makes the space equivalent to the gaps available following the end of this data set.

The ALTERMOD function of IEBCOPY updates the PDS directory entry and control records of load modules by inserting relocation dictionary (RLD) counts. The COPYMOD function copies, reblocks, and also performs the ALTERMOD function on load modules.

INVOKING THE IEBCOPY PROGRAM

The IEBCOPY program may be invoked either by the job scheduler or by a user's program. In the former case, the EXEC statement of the job control language (JCL) must specify either the PGM=IEBCOPY keyword or a procedure in which PGM=IEBCOPY is specified, and the JCL for the program must define the required data sets using standard ddnames (SYSIN, SYSUT3, SYSUT4, and SYSPRINT). If a user's program invokes the IEBCOPY program, it may specify alternate names for any or all of the standard ddnames. Additionally, a user's program may specify a page header parameter containing the first page number to be used. The user's program passes control to the IEBCOPY program via either a LINK or an ATTACH macro instruction. The IEBCOPY program can also be loaded using the LOAD macro instruction. (See Utilities for more information.)

PROGRAM STRUCTURE

The basic functions of each of the modules and I/O appendages used by the IEBCOPY program are summarized below:

Module	Function
IEBDSCPY	Controls the program flow; initializes the program; allocates storage; locates directory entries; converts addresses; and constructs control table entries.
IEBVMS	Prints messages and control statements.
IEBDV1	Opens and validates partitioned data sets.
IEBLDUL	Opens and validates unloaded data sets; reads descriptor records.
IEBSCN	Scans control statements; constructs the input data set name table and the table of members to be selected or excluded.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEBRSAM Reads the unloaded data set directory into the buffer; reads member data into the buffer; places member note list in the note list work area.

IEBCNVT Handles all TTR (relative record address) to MBBCCHHR (full disk address) conversion requests during a load operation for the unloaded data set.

IEBDRD Reads directories, member data, and note lists from partitioned data sets into buffers.

IEBVCT Sorts entries in the table of selected or excluded members.

IEBIOE Analyzes I/O errors.

IEBWSU Prepares data for partitioned data sets to be written from buffers.

IEBDWR Writes directory and member data for partitioned data sets from buffers onto the partitioned output data set.

IEBWSAM Writes directory and member data from buffers to the unloaded data set; writes the header records used for unloaded data sets.

IEBDSU Builds output directory blocks for unloaded data sets; indicates which members have been copied.

IEBBAM Allocates the initial I/O buffer area and places old-name/new-name member pairs in the I/O buffer from a table of selected members.

IEBVDM Merges input and 'current' output directory entries to produce a new output directory for partitioned data sets.

IEBDRB Deblocks or reblocks members that are to be copied or loaded.

IEBMCM Locates the first gap between members in the data set to be compressed. Prepares the input and output DCB so that a full copy can be performed.

IEBCPMOD Copies and reblocks load modules and alters RLD counts in PDS directory entries and in control records.

IEBVTT Sorts entries in the control table.

IEBVTM Terminates processing for each copy step and operation.

IGG019C8 Determines 'next record' information required for modifying channel programs while reading or writing.

IGG019FT Builds extended translation list for CCW strings. Fixes storage locations so that disabled page faults will not occur in PCI appendage IGG019C8.

PROGRAM OPERATION

"Chart 01. Program Initialization" on page 133 through "Chart 12. Directory Processing" on page 144 provide an overview of the IEBCOPY program. Basically, the IEBCOPY program first performs some general initialization processing (for example, it opens data sets, and obtains and allocates main storage). Then, the program scans control statements to determine the type of operation to be performed, and, except for a full copy or load (in which there are no specified member names), the program constructs a table of specified member names that are taken from parameters on the input control statements.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

For the rest of this discussion, assume that a separate pass through the IEBCOPY program is made for each input data set (taken from the control statements for the program) to be processed. Actually, the initialization procedures will vary depending on the operation performed. The program then opens other data sets and allocates more storage, if it is available.

For a selective or an exclusive copy, unload, or load, the program then sorts the table alphamerically (by member name) and compares the individual member names to those appearing in the input data set. From this comparison, the program constructs a control table (CTLTAB table) containing references (in the form of pointers to directory entries) to those members that are to be copied, unloaded, or loaded. For an exclusive copy, unload, or load, the control table will contain references to all members in the input data set except those that are to be excluded.

For a full copy, unload, or load, a control table is built to contain references to all the members in the input data set.

The output directory is then scanned to eliminate CTLTAB table references to duplicate-named input members that are not to replace corresponding output members.

The program then arranges the control table into two sequences, one consisting of the directory entry pointers arranged in alphameric sequence according to the member names, and the other containing the same information but arranged in a sequence based on the TTRs of the member names.

After the program checks for any user and/or note list TTRNs, aliases, or specified blocking considerations, it copies (or alters, if ALTERMOD is specified) the input data set's members onto the output data set, determines the new member-TTRs, and updates the directory for the output data set. (The two previously mentioned sequences of the control table are used in this processing.)

The basic processing just described is repeated for each input data set of each INDD-group. When a new copy, unload, or load operation is to be performed, the program repeats the processing from the point at which control statements are scanned. Following the processing of the last copy, unload, or load operation, control returns to the caller of the IEBCOPY program.

COMMON COMMUNICATION AREA FOR THE IEBCOPY PROGRAM

The general contents of the communication area DSECT, IEBMCA, are listed in Figure 33 on page 58. Specific values or meanings of constants and switches may be found by consulting the microfiche cards for the modules of the IEBCOPY program.

MAJOR TABLES USED BY THE IEBCOPY PROGRAM

The tables discussed in this section are the select-exclude (SETAB) table, the new name pointer (NNPTRTAB) table, the control (CTLTAB) table, and the INDD table. The second and third tables are constructed on the basis of information contained in the first table. For a selective copy, unload, or load step, the SETAB table is transformed into the CTLTAB table in the course of processing done by the program. The CTLTAB table will occupy the same physical area in storage as the SETAB table. For an exclusive copy, unload, or load step, a CTLTAB table is constructed in a physically different area of storage than the one containing the SETAB table. For a full copy, unload, or load step, the IEBCOPY program constructs only a CTLTAB table as it has no need for a SETAB table.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

SETAB TABLE

Figure 34 on page 59 illustrates the general format of an entry in the SETAB table, which module IEBSCN constructs following the scanning of either a SELECT or an EXCLUDE statement. Each entry in the table refers to a member that is to be selected, a new name for a selected member, or a member that is to be excluded during a copy, unload, or load step.

The flag bytes for each SETAB table entry define the characteristics of the member whose name appears in the last 8 bytes, and indicate processing information concerning the member.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Label at Start of Field[1]	Field[1] Size (bytes)	Field[1] Contents (in general)
ICPT	21	Constants and addresses.
FLG2	4	Switches and flags.
UTTRFLAG	1	Note list processing flags.
OBCT	2	Count of output blocks in storage.
BCB	16	Buffer control block.
MAXBUF	32	I/O buffer sizes and read-to/write-from pointer.
ADWANT	32	Information for obtaining/releasing storage.
ADNL	78	Information on areas, tables, member and block counts, available storage, etc.
ZROS	4	All zeros.
DBKEY	16	Directory block information.
OUTNAME	8	Name of current output data set.
CCIMAGE	80	Scan control card buffer.
WKA1	40	General work area.
ALO	48	Addresses and areas used by various routines.
SV1	532	Seven register save areas.
MCAMOD	40	"Change" area for debugging.
VIEBDVO	208	Subroutine addresses.
LIST1	24	JFCB and DCB exit information.
WRPCICNT	8	PCI information.
WRPTR1	24	CCW addresses.
OUTPTTR	48	End-of-file and track balance information.
OUTFDAD	24	Output data set information.
ERPLMFD	28	Error information.
OBLKSIZE	15	I/O validation routine constants.
NXTTTR	7	Miscellaneous constants.
DDLIST	54	DDname defaults (for initialization).
OUTVOL	6	VOLIN of output data set.
OTDSNAME	44	Area for input and output DSNAMEs.
CARDCB	732	Data control block information (SYSIN, SYSPRINT, SYSUT1, SYSUT2, SYSUT3, SYSUT4).
INIOB	272	IOB area for input, output, SYSUT3, SYSUT4.
INCHAR	80	Device type information.
EVENTCB	4	Event control block for EXCP.
SSECTOR	216	Channel command words.
SECTOR	2	Temporary sector save area.
LASTADDR	36	Restart address for writing.
ROLENGTH	116	Working area for appendages.
BLOCK	35	Area used by IEBVDM.
FLG3	6	General switches and flags.
TAG	1	Switches and flags for writing.
WSFLAG	1	Module IEBWSU interface switches.
PARMSWCH CCSWITCH COMDCDSW CPARAMSW CCDELIM CCDELIM2 STATUS	7	Scan routine switches (one byte each).

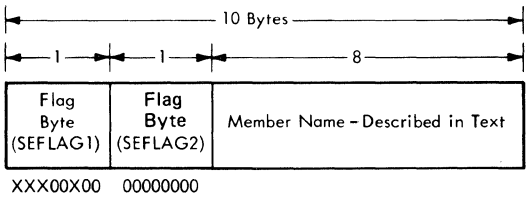
Figure 33 (Part 1 of 2). Communication Area DSECT (IEBMCA) for IEBCOPY

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Label at Start of Field[1]	Field[1] Size (bytes)	Field[1] Contents (in general)
SENSE	1	Read routine switches.
VTMFLG1	1	Flags for use by IEBVTM.
IOEF1	2	Flags for I/O error definition.
MSGLIST	8	Parameter input to message module.
MSGPARAM	20	Message parameter field.
RCBUF	1	Program completion code area.
LINECT	1	Paging information.
MSGBUF	121	Message buffer.
PGNO	4	Message output starting page number.
SAVERGS	112	IEBWSU work area.
MCAEND		End of communication area (on a doubleword boundary).

[1] The word "field" as used here refers to a section of the communication area that begins at the label indicated but may extend through several succeeding labels that are not given in this table.

Figure 33 (Part 2 of 2). Communication Area DSECT (IEBMCA) for IEBCOPY



X = bit may be 1 or 0 depending on control statement parameters.

Figure 34. SETAB Table Entry After Initial Scan of SELECT Statement by Module IEBSCN

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

With the convention that a bit is on when it is set to '1', the values of the flag byte (SEFLAG1) bits are given below:

Bit	Set	Meaning
Bit 0	On	Member name is a new name entry.
Bit 1[1]	On	Member name is an old name (that is, member is to be renamed entry.
Bit 2[2]	On	REPLACE option has been specified for the member.
Bit 3	On	Member is not to be processed. (Called the 'don't copy' bit.)
Bit 4	On	Member has been located ('found') on an input data set.
Bit 5	On	Member is last entry in the sorted SETAB table.
Bits 6 & 7		Not used at this time (that is, reserved).

Notes:

1 This bit is set on only when a member's name is to be changed.

2 This bit does not have to be on if bit 2 of the flag byte in the INDD table (described later) is on. However, it is not an invalid condition if both bits are on.

NNPTRTAB TABLE

After module IEBSCN constructs the initial SETAB table for a selective operation (see Part A of Figure 35 on page 61), any old-name/new-name pairs are placed in an I/O buffer used here as a work area (see Figure 35, part B). Module IEBVCT sorts the SETAB entries in alphameric collating sequence by member name and establishes a pointer table (NNPTRTAB table) to the new names (see Figure 35, parts C and D). Each NNPTRTAB table entry contains 4 bytes: an indicator byte (currently not used) and a 3-byte address of an old name SETAB table entry that corresponds to a new name (in the old-name/new-name pair). The NNPTRTAB table is contiguous in storage to the SETAB table, and the table entries are in alphameric collating sequence according to the old names.

The following text describes Figure 35.

Assume a control statement for a selective operation is as follows:

```
SELECT MEMBER=(A,(H,C),(M,,R),(B,D,R),S,L)
```

in which:

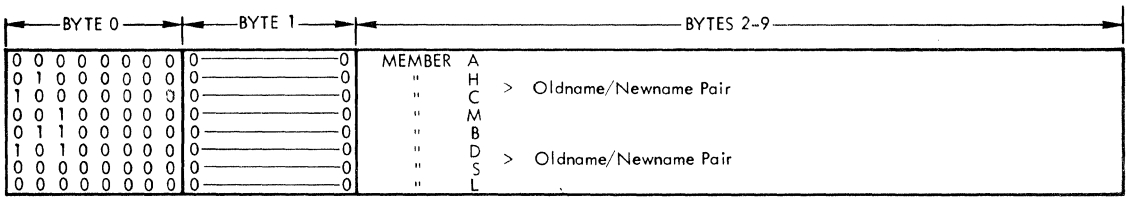
Member H is to be renamed C.

Member M is to replace the identically named member M on the output data set.

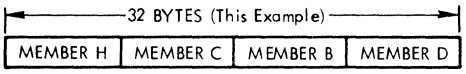
Member B is to be renamed D and replace the current member D on the output data set.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

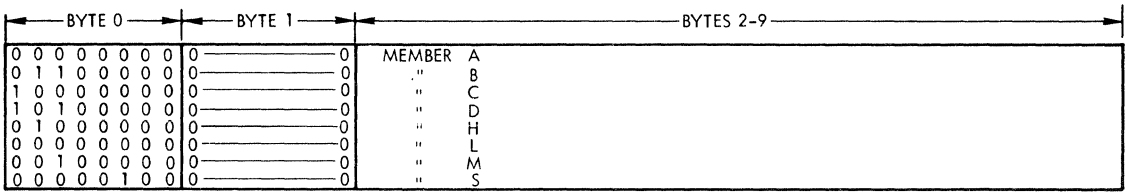
A
Initial SETAB



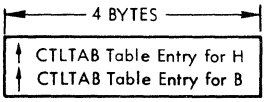
B
I/O buffers with newname/oldname pairs



C
SETAB, sorted by alphameric sequence, low to high

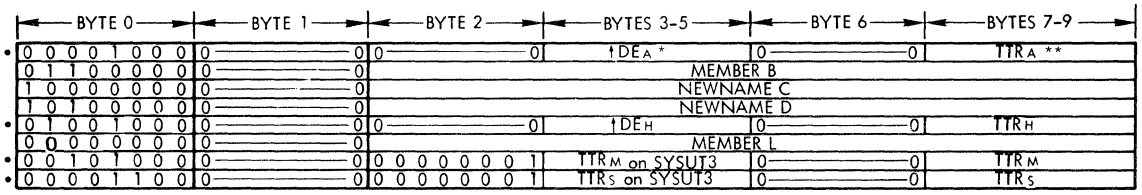


D
Newname pointer table



E
Construction of CTLTAB entries

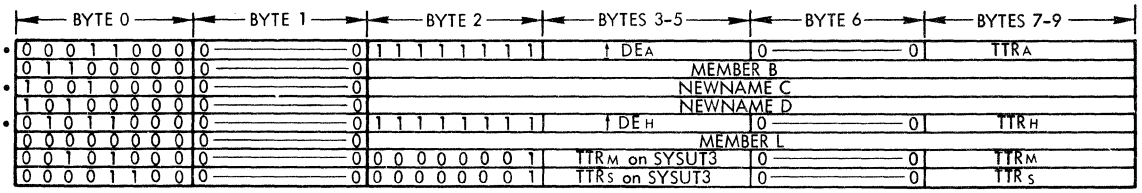
Bullet indicates changes from preceding corresponding figure



* Read as "address of directory entry for member A"

** Read as "TTR of member A as taken from directory entry referencing input member A"

F
CTLTAB table entries after output directory comparison



G
CTLTAB table with directory entry pointers sorted according to TTR sequence

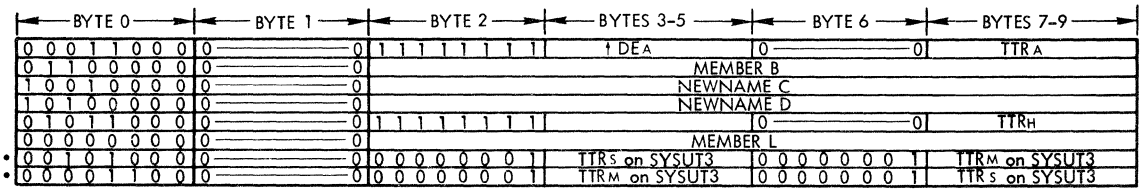


Figure 35. To Show Construction of Tables Used by the IEBCOPY Program

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Members A, L, and S are not to be renamed and are not to replace other members. (Throughout this description, other assumptions will be made as needed to illustrate certain features.)

Part A of the figure illustrates the initial formation of the SETAB table by module IEBSCN.

Part B of the figure illustrates the old-name/new-name member pairs that module IEBBAM places in the I/O buffer after the SETAB table has been constructed.

Part C of the figure represents the SETAB table after module IEBVCT has sorted the member names in alphameric collating sequence.

Part D of the figure illustrates the new name pointer table (NNPTRTAB table) containing the storage addresses of old names corresponding to new names in the sorted SETAB table. This table is arranged in a sequence according to the alphamerically sorted new-name references.

Part E of the figure illustrates the CTLTAB table entries that subroutine IEBMBT places in the SETAB table as a result of finding members on the input data set that correspond to the 'not-new' member names listed in the SETAB table. In this part of the figure, assume that members A, H, M, and S are 'found' on the current input data set, and that input directory entries for members M and S are spilled to the SYSUT3 data set and that input directory entries for members A and H are retained in main storage. Also, assume no alias entries in this example.

Part F represents the SETAB/CTLTAB table after module IEBDSCPY compares the output data set member names with the CTLTAB table member names representing members in the current input data set. In this example, assume that member names A, C, and M have been found on the output data set directory. Further assume that the REPLACE option is not specified on the INDD (keyword or control statement) level.

Part G of the figure indicates the final manipulation (sorting by input member TTR) that is performed on the SETAB/CTLTAB table (by module IEBVTT) before the members are copied and directory entries are subsequently merged onto the output data set. For the purpose of illustration, assume that the input TTR addresses of the members are as follows:

Member	TTR
A	14C
H	0C4
M	158
S	03B

Figure 36 on page 63 illustrates the output data set that results from merging the members listed in the example shown in Figure 35 on page 61.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

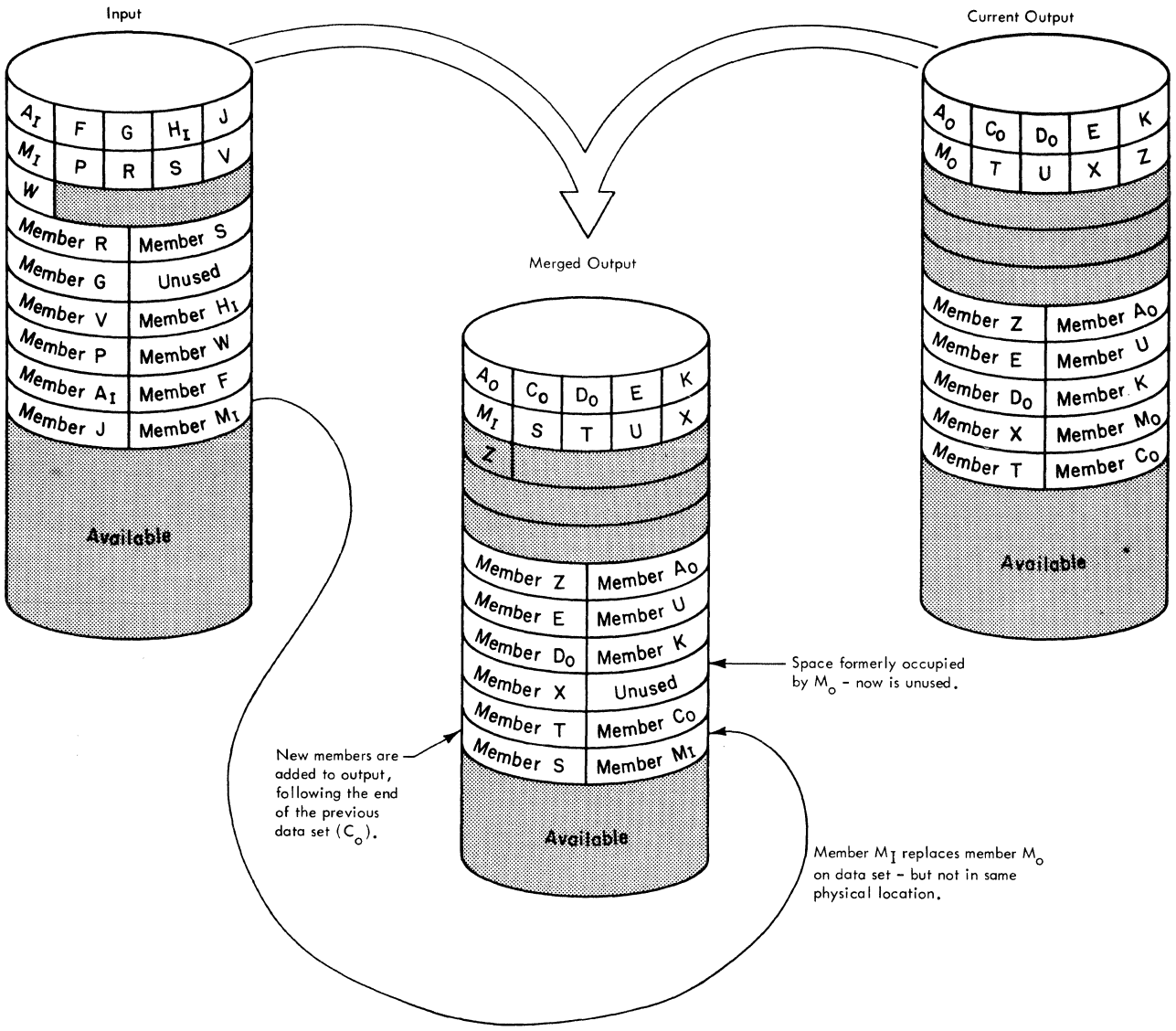


Figure 36. Merging Selected Input Members Onto an Output Data Set

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

To construct the CTLTAB table for a selective operation, module IEBDSCPY compares the entries in the input data set directory, which module IEBDRD has read into an I/O buffer in main storage, with all except new names in the SETAB table. For those names for which a match is found, the module either retains the directory entry in main storage, if space permits, or writes (spills) it onto the SYSUT3 data set. A CTLTAB table entry is then built by placing both a pointer to the directory entry and the input member's TTR into the corresponding SETAB table entry as shown in Figure 35, part E. If the directory entry remains in storage, the address used in bytes 3 to 5 is the storage address of the entry. If the directory entry is spilled to the SYSUT3 data set, bytes 3 to 5 contain the TTR address of the entry on SYSUT3 and byte 2 is set to hexadecimal '01'. In either case, bytes 7 to 9 contain the input member's TTR taken from the corresponding directory entry. (For a description of a data set directory entry, see Data Management Services.) If the directory entry is an alias (or alternate name) entry, the high-order bit of byte 1 of the corresponding CTLTAB table entry, referred to in Figure 34 as SEFLAG2, is set on. In all cases, the module sets bit 4 of byte 0, SEFLAG1, on when the input member has been found. The indicator byte, byte 2, signifies the location of the directory entry: The byte's value is 0 if the entry is retained in main storage, and it is hexadecimal '01' if the entry is spilled onto the SYSUT3 data set.

After subroutine IEBMBT of module IEBDSCPY has completed all CTLTAB table entries for a given input data set, module IEBDRD reads the output data set directory, if any, into a buffer in main storage. The output data set member names are then compared with the names of the found members (those SETAB table entries for which a CTLTAB table entry has been constructed) for the current input data set.

Note: Module IEBDSCPY performs this prescan comparison against either the old name of the input data set's members in the CTLTAB table, or, if the members are to be renamed, against the new names in the table.

If the same member name appears on both the input and the output data sets, and if the REPLACE option has not been specified, module IEBDSCPY sets bit 3 of byte 0 in the appropriate CTLTAB table entry to '1' and the indicator byte (byte 2) of the entry to hexadecimal 'FF' to indicate that the entry is not to be processed. At this point, the SETAB/CTLTAB table (so called because the table may now contain original SETAB table entries as well as CTLTAB table entries) may contain entries for members that have not been found as well as for those that have been found in either the current or any previous input data set(s) specified in the current copy step. Figure 35, part F, illustrates the types of entries that the table may contain.

The final step in manipulating the SETAB/CTLTAB table is to sort the member TTRs into numeric sequence. This sorting affects only the table entries for members that have been found in the current input data set. Module IEBVTT performs the sort by moving the data from bytes 2 to 5 of each found entry to bytes 6 to 9 of the same entry (see Figure 35, part G). The TTR data that was in bytes 6 to 9 of the entry is placed in its relative numeric position in a sequence of TTRs so that the existing data in bytes 2 to 5 is now also arranged in sequence according to member TTR.

The CTLTAB is also used to keep track of the actions performed against a member. Bit 4 of byte 1 (SEFLAG2) of the table is used by IEBVDM to mark that the member has been successfully copied and that the entry now contains the member name as it was merged into the output directory entry. Bit 4 through bit 7 of SEFLAG2 are set by IEBCPMOD when the member is not processable by ALTERMOD or COPYMOD. The values of the second flag byte (SEFLAG2) are given below:

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Bit	Set	Meaning
Bit 0	On	Member name is an alias (or alternate name).
Bit 1		Reserved
Bit 2		Reserved.
Bit 3	On	The entry contains the name of the member just copied.
Bit 4[1]	On	The member is in scatter-load format.
Bit 5[1]	On	The member is in overlay format.
Bit 6[1]	On	The member is not a load module.
Bit 7[1]	On	The member is not link-editable.

[1] These flags are set only during ALTERMOD or COPYMOD processing. When set, they indicate that the member could not be processed by module IEBCPMOD. If ALTERMOD was specified, the member was skipped. If COPYMOD was specified, then IEBCPMOD returned control to IEBDSCPY so that the member could be copied without reblocking or altering.

SEFLAG2 is referenced by IEBVTM and IEBCPMOD (entry point IEBCPMSG) when the completion messages are written for the input data set.

INDD TABLE

The INDD table contains the names of input DD statements. The scan module, IEBSCN, constructs the table as it scans COPY and INDD control statements. Each table entry contains 2 flag bytes and 8 bytes for the statement name.

The 'Split-Table' Approach

This section presents another view of the SETAB/CTLTAB table that may be helpful. (See Figure 37 on page 66.) The structure of the table is considered at successive stages of its formation. For the CTLTAB table, consider bytes 2 to 9 of the table as being split vertically into two halves, one half consisting of bytes 2 to 5 and the other consisting of bytes 6 to 9. The high-order byte of each half is an indicator byte that is ignored in this discussion. This development is for a SELECT copy, unload, or load step.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

BYTES			
0	1	2 - 5	6 - 9
FLAG BYTES		Left Table (including indicator bytes)	Right Table (including indicator bytes)

Stage of Processing	Contents of Bytes 2-9	
I. Initial SETAB table built.	Member names for selection, as they appear on input statement.	
II. SETAB table sorted.	Member names (including 'newnames') for selection in a sequence from low to high (00 to FF) collating sequence.	
*		
*		
*		
*	**Contents of Left Table (bytes 2-5)** (See Note)	**Contents of Right Table (bytes 6-9)**
III. CTLTAB table after input data set directory is searched for names to match non-new names in SETAB table, and also after output data set directory is compared against 'found' members.	Main storage addresses of SYSUT3 TTR locations for directory entries of found members. In sequence corresponding to the sorted SETAB table member names.	Member TTR locations (in directory entries for found members) in sequence corresponding to sorted SETAB table member names.
IV. CTLTAB table after sorting members for the copy, unload, or load phase of the program.	Main storage addresses of SYSUT3 TTR locations of directory entries for members to be copied, unloaded, or loaded from current input data set. In ascending sequence according to member input TTR locations.	Main storage addresses of SYSUT3 TTR locations of directory entries for members to be copied, unloaded, or loaded from current input data set. In alphameric collating sequence (low to high).

Note: CTLTAB table entries for members not found or not to be copied (processed) are interspersed among the sequenced entries described.

Figure 37. Split-Table Representation of SETAB/CTLTAB Table

STORAGE ALLOCATION

After module IEBDSCPY has control, it issues a variable conditional GETMAIN macro instruction for all of storage that is available between the end of the program and the start of the access method module area. Figure 38 on page 67 indicates how storage is allocated for IEBCOPY.

Beginning at the low address portion of this area, IEBDSCPY reserves 1300 bytes for a note-list buffer area. Contiguous to this area, the module allocates a 128-byte work area that is used primarily for I/O operations on the SYSUT3 data set. The module then uses a FREEMAIN macro instruction to release approximately 3000 bytes of storage from the high address end of the existing available storage. This released area is available for use by system macro instructions (for example, OPEN, CLOSE, SYNADAF) used by the IEBCOPY program.

The preceding one-time initialization is performed each time the IEBCOPY program is executed. The following discusses allocation procedures as they apply to each copy, unload, or load step, or INDD-group (those INDD=ddnames contained between copy, unload, or load step delimiters as described in Utilities).

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

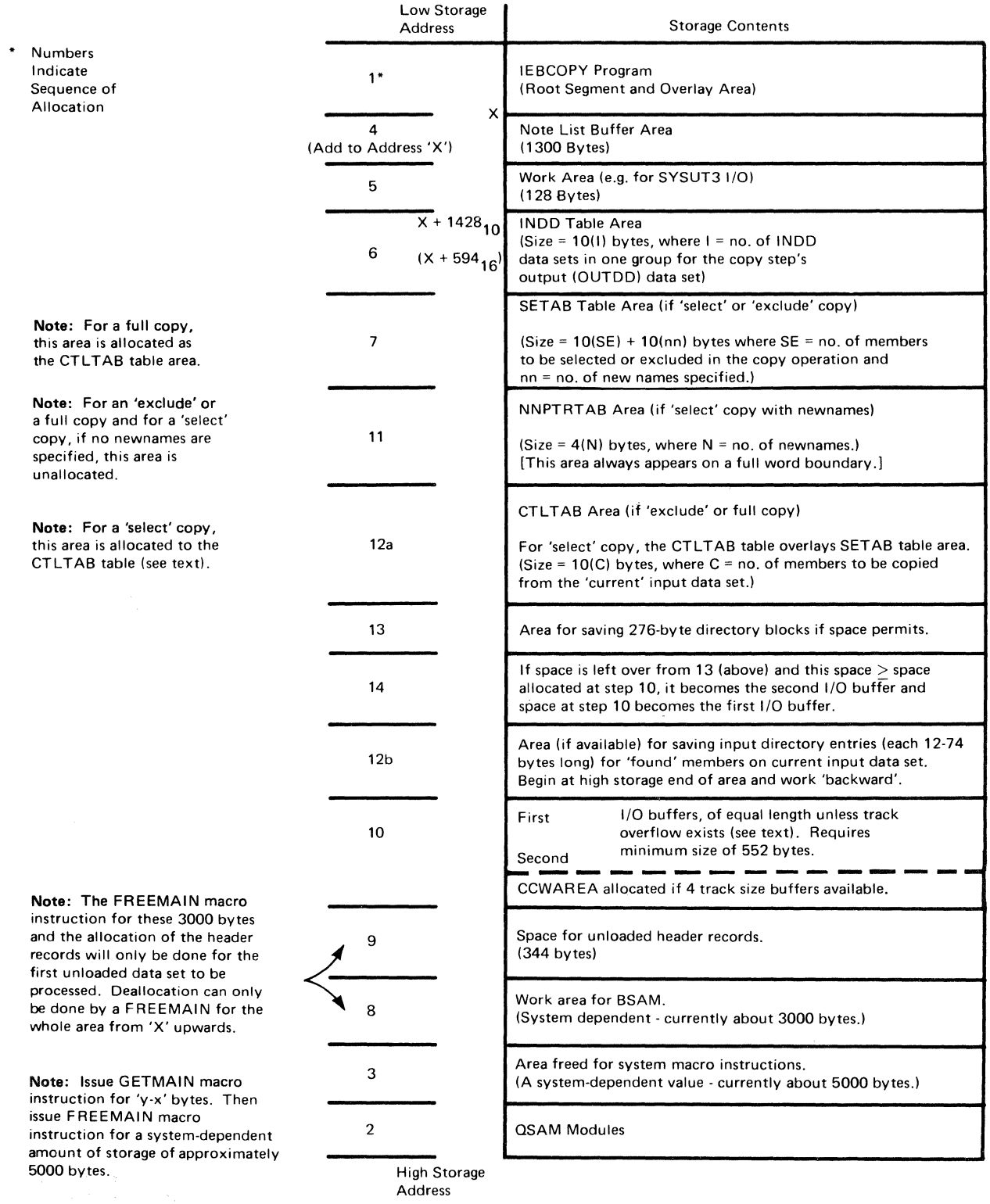


Figure 38. Storage Allocation for IEBCOPY Program

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

STORAGE ALLOCATION FOR TABLES (I)

Note: In this and the next three equal-level text headings, the Roman numerals indicate that there is more than one distinct stage of storage allocation for tables and for I/O buffers.

For each copy, unload, or load step, module IEBSCN (statement scan module) builds an INDD table that contains 10 bytes for each input data set specified in the INDD statement(s) for the step. The storage allotted for this table is contiguous to the 128-byte work area.

For each SELECT or EXCLUDE statement in the copy, unload, or load step, there are member name(s) to be placed in the select/exclude (SETAB) table.

Because each specified member name requires 10 bytes, the scan module allocates a SETAB table storage space that is equal to 10 times the number of member names involved in the copy, unload, or load step. This table is contiguous to the INDD table. Because a full copy, unload, or load step does not require explicit specification of member names, the module does not construct a SETAB table for a full copy, unload, or load.

STORAGE ALLOCATION FOR I/O BUFFERS (I)

After module IEBSCN has constructed the SETAB table (if one was to be built), the data sets are opened. If an unloaded data set is found, space is allocated for the descriptor records and the BSAM modules in the high address section of the remaining available storage, if this was not done previously. Module IEBBAM then establishes the input/output buffers in the high address section of the remaining available storage. (If the absolute minimum buffer area is allocated (see "Module IEBBAM"), an indicator is set to prevent an attempt to obtain more space later on as input member data is copied, unloaded, or loaded.) These buffers are of equal length unless track overflow conditions are specified. In the latter case, the second, or output, buffer will be larger. If the compress-in-place option is to be performed, the length of each I/O buffer is equal to the maximum number of bytes that can be written onto two tracks of the device used for the input and output data set.

STORAGE ALLOCATION FOR TABLES (II)

If the SELECT option for the copy, unload, or load step has been specified, and if some of the members of the INDD-group have been renamed (that is, new names are present in the SETAB table), module IEBDSCPY places a new-name pointer (NNPTRTAB) table contiguous to the SETAB table. This pointer table consists of 4 bytes for each renamed entry and retains the old-name/new-name relationships once the SETAB table is restructured (as described under "NNPTRTAB TABLE").

(For each subsequent input data set that is processed for a given copy, unload, or load step, storage allocation begins with the I/O buffer allocation process previously mentioned, but omits the NNPTRTAB table allocation process because the SETAB table is restructured only once for each copy, unload, or load step.)

The next phase of storage allocation is the building of the control (CTLTAB) table for the copy, unload, or load step. For a selective copy, unload, or load, the CTLTAB table does not require any additional storage, because it is formed as a replacement for specified parts of the SETAB table. (See Note below.) If the copy, unload, or load is an exclusive one, subroutine IEBMBT of module IEBDSCPY constructs the CTLTAB table contiguous to the SETAB table area. For a full copy, unload, or load, the subroutine builds the CTLTAB table contiguous to the INDD table (where the SETAB table would be for a select or an exclude copy, unload, or load). Concurrent with the building of

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

the CTLTAB table, subroutine IEBMBT places directory entries in storage (if space permits) in the following manner:

Note: The SETAB table contains member names that are to be selected or excluded during a copy, unload, or load step. As entries in the SETAB table are found for a selective copy, unload, or load they become CTLTAB table entries overlaying their SETAB table counterparts. The table then contains two types of entries and is actually a SETAB/CTLTAB table. For an exclusive copy, unload, or load, a separate CTLTAB table is constructed. For a full copy, unload, or load, there is no SETAB table (only a CTLTAB table).

DIRECTORY ENTRY STORAGE ALLOCATION: Beginning at the storage address immediately preceding the start of the I/O buffer area, and working in the direction of lower storage addresses, input directory entries (which may be from 12 to 74 bytes in length) for members that are to be copied from the current input data set are placed into storage from the I/O buffer into which they were read by module IEBDRD. Subroutine IEBMBT also places a pointer to each retained directory entry into the corresponding CTLTAB table entry. As CTLTAB table entries are built in the direction of high storage addresses and directory entries are placed in storage in the opposite direction, there may come a time when the construction of a CTLTAB table entry requires storage space in which a directory entry has already been retained. When this happens, the subroutine IEBMBT causes the most recent directory entry that was retained in storage to be written to the SYSUT3 data set. The TTR of the entry's location on the SYSUT3 data set is placed into the corresponding CTLTAB table entry, and the storage space, which the directory entry occupied, is then available for more CTLTAB table entries. All subsequent directory entries (corresponding to these additional CTLTAB table entries) are also spilled to the SYSUT3 data set.

This procedure is repeated as often as necessary until the CTLTAB table is constructed. Because of this technique to efficiently utilize available storage, the members spilled to the SYSUT3 data set are not necessarily in the same sequence as their corresponding CTLTAB table entries. Figure 39 on page 70 illustrates this process.

EXAMPLE: Construct 17 control table (CTLTAB table) entries and retain or spill their corresponding directory entries.

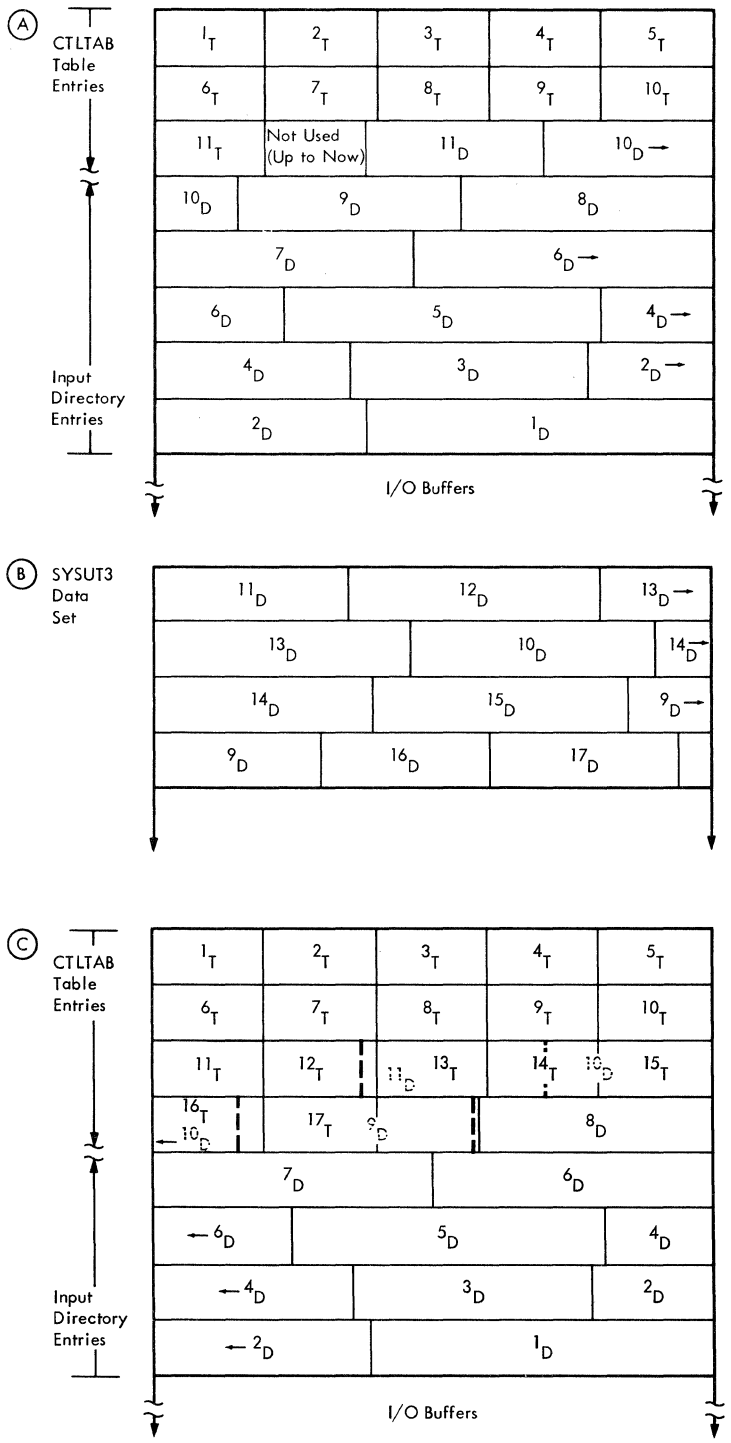
METHOD: Assume, in Figure 39, that CTLTAB table entries of 10 bytes each have been constructed for 11 'found' members from an input data set, and that the corresponding 11 input directory entries, of various lengths, have been moved to the directory storage area (area 12b of Figure 43) from the I/O buffer (area 10). In the case of a selective copy, load, or unload, table entries for members not 'found' are not shown, but they may be interspersed among the existing CTLTAB table entries.

Now, when CTLTAB table entry 12T is required, the existing space (between spaces 11T and 11D) is less than 10 bytes. Therefore, subroutine IEBMBT spills directory entry 11D onto the SYSUT3 data set as indicated in Figure 39. This permits CTLTAB entry 12T to be constructed.

The directory entry 12D corresponding to CTLTAB table entry 12T is then spilled onto the SYSUT3 data set from the I/O buffer. To construct CTLTAB table entry 13T, the necessary space is available (because of the area released by spilling entry 11D). Thus, table entry 13T is constructed, and its directory entry is spilled to the SYSUT3 data set as indicated. (See Figure 39.)

A similar procedure is followed for the construction of table entries 14T through 17T and the spilling of corresponding directory entries 14D through 17D plus the spilling of directory entries 10D and 9D to provide space for table entries.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1



Legend:
T = Control table entry (10 bytes in length)
D = Directory entry (variable length)
→←= Continued entry

Figure 39. Spilling Input Directory Entries from Main Storage

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Figure 39 shows the resulting space allocation for CTLTAB table entries 1T through 17T and as many corresponding directory entries as will fit in the available space. Dashed lines and dotted directory entries indicate space allocations that existed in Figure 39 before the first directory entry was spilled to the SYSUT3 data set.

As more CTLTAB table entries are constructed, with the release of more directory entries to the SYSUT3 data set, enough storage may be used so that a CTLTAB table entry may require space that has been allocated to the I/O buffers. (This implies that there are then no input directory entries remaining in main storage.) If this happens, and more than the minimum size I/O buffer space has been allocated, subroutine IEBMBT decreases the buffer space as needed, if possible, in increments of 'two times the minimum buffer size' until either twice the minimum size buffer space is left or there are no more CTLTAB table entries to construct.

Note: The CTLTAB table itself cannot be spilled. If all available storage beyond the minimum size I/O buffer space has been used but more CTLTAB table entries must be built, a message is written to the user and the processing of this input data set stops.

If there is main storage remaining immediately following the CTLTAB table (or following the NNPTRTAB table if the SELECT option is specified and new names are present), as many output directory blocks (and related device information) as possible are retained there. The actual track address (MBBCCHHR) of the first output directory block that cannot be retained is saved in the communication area (CSECT IEBMCA).

STORAGE ALLOCATION FOR I/O BUFFERS (II)

If the user did not specify the SIZE= parameter, the following phase is bypassed.

If the storage space allocated to the program still contains an unused area after all relevant input and output directory entries and blocks for the current input data set have been stored there, the unused area may be large enough for use as a second I/O buffer. (If a compress operation is being performed or if the output data set has a track overflow specification, a second I/O buffer is not allocated.) If the unused area is at least as large as the total area currently used as an I/O buffer area, module IEBDSCPY uses the previously allocated two I/O buffers as one large I/O buffer, and the unused area becomes the second I/O buffer for the actual copying, unloading, or loading and merging operations.

STORAGE ALLOCATION FOR I/O BUFFERS (III)

When ALTERMOD or COPYMOD is specified, module IEBCPMOD defines input and output buffers of 18K bytes each (18K bytes is the maximum record size permitted by the linkage editor). These buffers are defined separately from the previously discussed I/O buffers because of the reblocking that must take place. If a load operation is not being done, and if these I/O buffers will fit, they are put within the COPY I/O buffers previously defined. Otherwise, the COPYMOD I/O buffers are built in the reserve storage area between the unspilled directory blocks (or CTLTAB) and the unspilled input directory entries.

In addition to the COPYMOD I/O buffers, IEBCPMOD defines work areas for RLD records (256 bytes) and control records (240 bytes) in the reserve storage area described above.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

STORAGE ALLOCATION FOR TABLES (III)

If ALTERMOD or COPYMOD is specified, module IEBCPMOD builds several tables used when rebuilding RLD and control records. These tables are built in the reserve storage area between the unspilled directory blocks and the beginning of the unspilled input directory entries.

UNLOADED DATA SET TAPE FORMAT

Figure 40 illustrates the tape format of a partitioned data set that was unloaded by the IEBCOPY program. A detailed description of the different records is given in subsequent paragraphs.

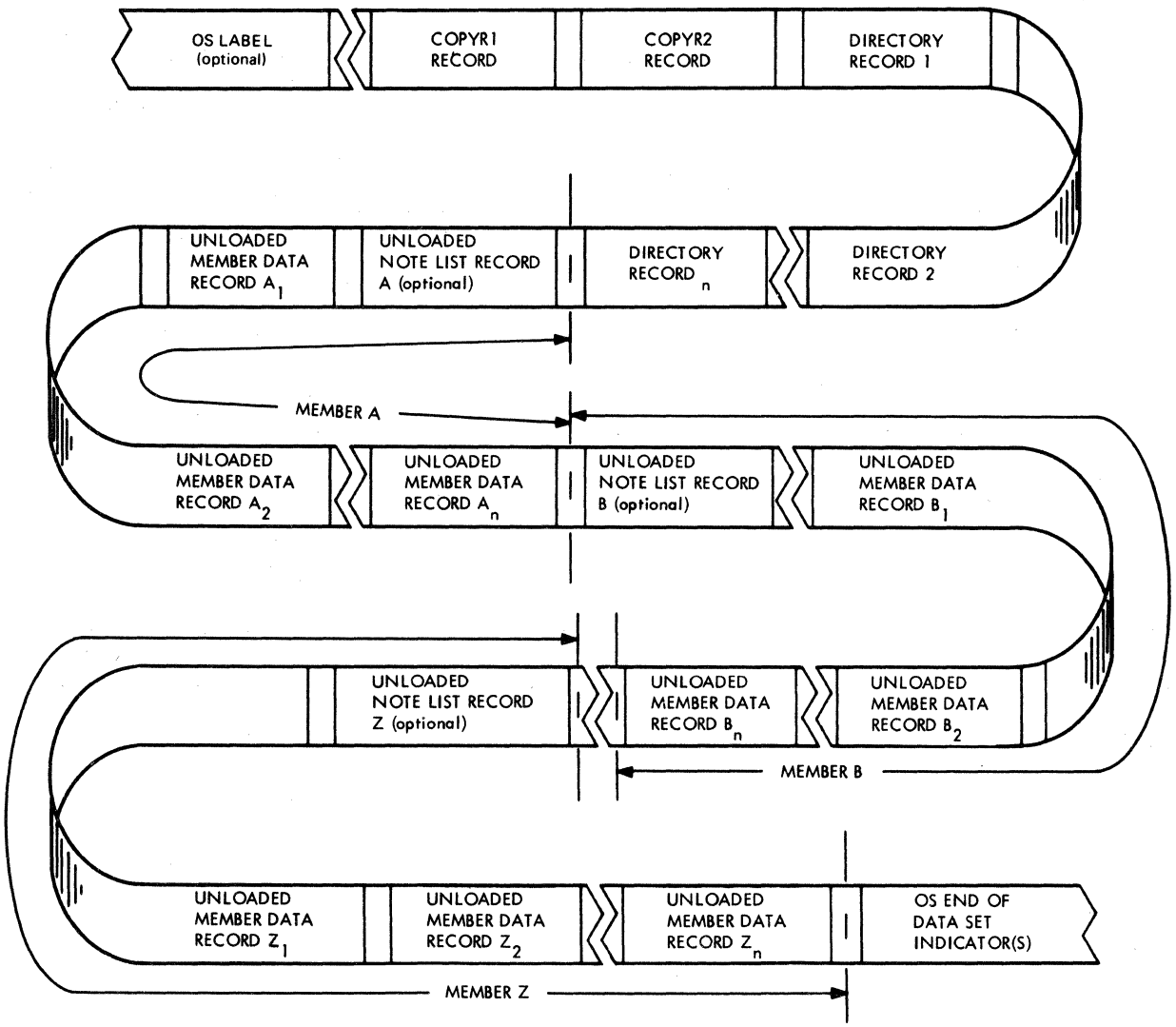


Figure 40. Records Present in an Unloaded Partitioned Data Set

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

DESCRIPTOR RECORDS

The first two records of an IEBCOPY unloaded data set contain information that is required to load the data set. The first record (COPYR1) contains a status field, an ID field, and a DCB information field. The status and ID fields are used for validity checking procedures. The DCB field is used to initialize the output data set during a load function. The second record (COPYR2) contains parts of the original DEB (data extent block) of the unloaded partitioned data set. When the data set is to be loaded, this information is used to update all the user and note list TTRs. Figure 41 and Figure 42 show the different fields in the COPYR1 and COPYR2 records.

Offset from Start of Area (in Hex)	Name of Field	Field Size (bytes)	Field Contents
-8	LL	4	Block descriptor for variable data set records.
-4	II	4	Record descriptor word for variable data set records.
0	INDC	1	Indicator bits. If Bit 0 is OFF, data set has been unloaded successfully. If Bit 0 is ON, data set cannot be loaded (an error occurred when unloading).
1	ID	3	IEBCOPY unloaded data set identifier (X'CA6DOF')
4	UDSORG	2	Data set organization of the unloaded data set.
6	UBLKSIZE	2	Block size of the unloaded data set.
8	ULRECL	2	Logical record size of the unloaded data set.
A	URECFM	1	Record format of the unloaded data set.
B	UKEYLEN	1	Key length of the unloaded data set.
C	UOPTCD	1	Option codes (OPTCD) of the unloaded data set.
D	--	1	Reserved.
E	TBLKSIZE	2	Block size of data set containing the unloaded data set.
10	UDEVTYPE	20	Device type information for the unloaded data set.

Figure 41. Contents of the COPYR1 Descriptor Record

Offset from Start of Area (in Hex)	Name of Field	Field Size (bytes)	Field Contents
-8	LL	4	Block descriptor word for variable data set records.
-4	11	4	Record descriptor word for variable data set records.
0	UDEBL16	16	Last 16 bytes of the basic section of the original DEB of the unloaded data set.
10	UDEBX	256	Complete extent description of the original DEB of the unloaded data set.
110	--	4	Reserved

Figure 42. Contents of the COPYR2 Descriptor Record

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

DIRECTORY RECORDS

The directory records are written immediately after the descriptor records. They consist of "pseudo" directory blocks containing the original directory entries for all members to be unloaded. In addition, the last directory record contains a pseudo end-of-file record.

The length of each directory record, except the last, is 8+n(276), where n represents the blocking factor. (n is an integer and, in this case, may not be zero.) The length of the last directory record is 20+n(276), where n represents the blocking factor (n is an integer and, in this case, may be zero). The length of any directory record, however, never exceeds the block size of the data set to contain the unloaded partitioned data set.

Note: As a result of dynamic buffer sizes, directory blocks within a directory record do not necessarily have the same blocking factor.

Figure 43 gives the directory record format for the unloaded partitioned data set. The following items have been assumed:

- The block size of the data set to contain the unloaded data set is 900 bytes.
- Seven pseudo directory blocks are required to contain the original directory entries for all of the unloaded members.

The directory blocks in the figure contain a count, key, and data field in the original directory block format. The count field is set to zero, except for the key length (X'08') and the data length (X'0100'). For the directory block format, refer to the Debugging Handbook.

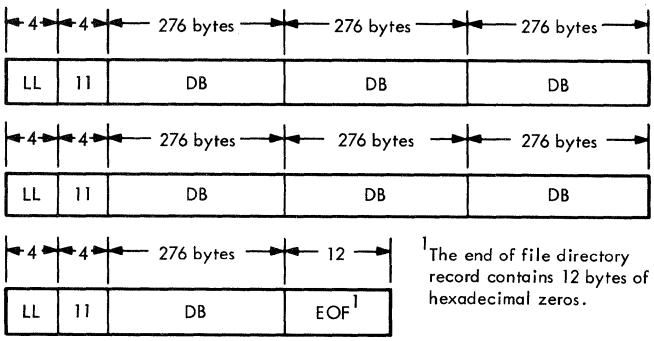


Figure 43. Example of Directory Record Format for the Unloaded Partitioned Data Set

UNLOADED NOTE LIST RECORD

If a member to be unloaded contains a note list, the note list is unloaded preceding the member data. The format for an unloaded note list record is shown in Figure 44 on page 75.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

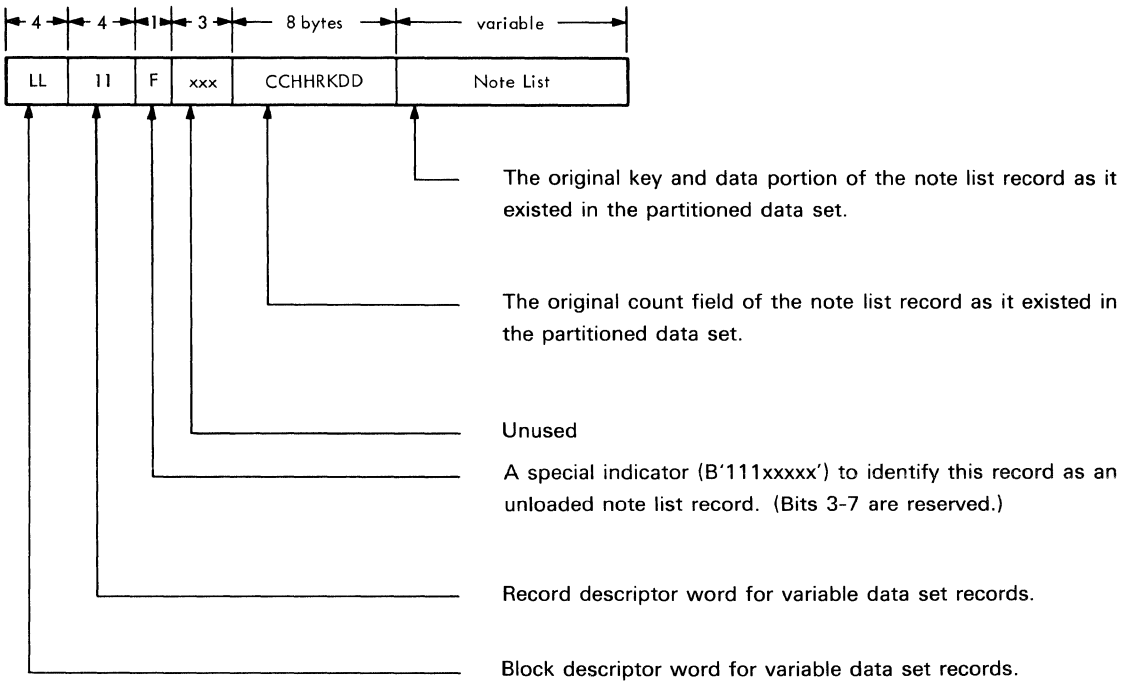


Figure 44. Example of Format of an Unloaded Note List Record

UNLOADED MEMBER DATA RECORDS

An unloaded member data record consists of the maximum integral number of partitioned data set member data blocks (see Figure 46 on page 76) that fit into the block size of the unloaded data set. Also included in this maximum number is the direct access end-of-file record at the end of each partitioned data set member. The length of the unloaded record never exceeds the block size of the unloaded data set, and the record never contains member data blocks from more than one member.

Note: Because of the variable length of member data blocks, and of dynamic buffer size considerations, member data blocks within an unloaded member data record do not necessarily have the same blocking factor. See Figure 47 on page 77.

Figure 45 on page 76 is an example of the format of unloaded member data records. The following items have been assumed:

- The block size of the partitioned data set is 350 bytes.
- There are six member data blocks per member (including the direct access end-of-file block).
- The record format of the partitioned data set is fixed.
- The block size of the data set to contain the unloaded data set is 900 bytes.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

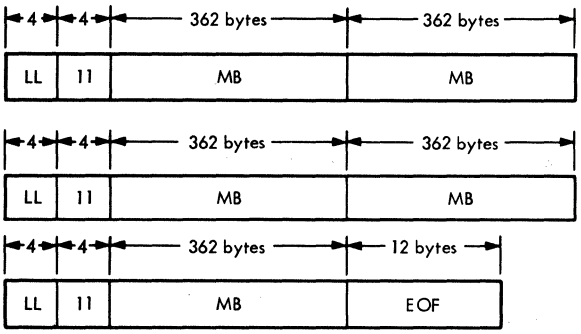


Figure 45. Example of the Format of Unloaded Member Data Records

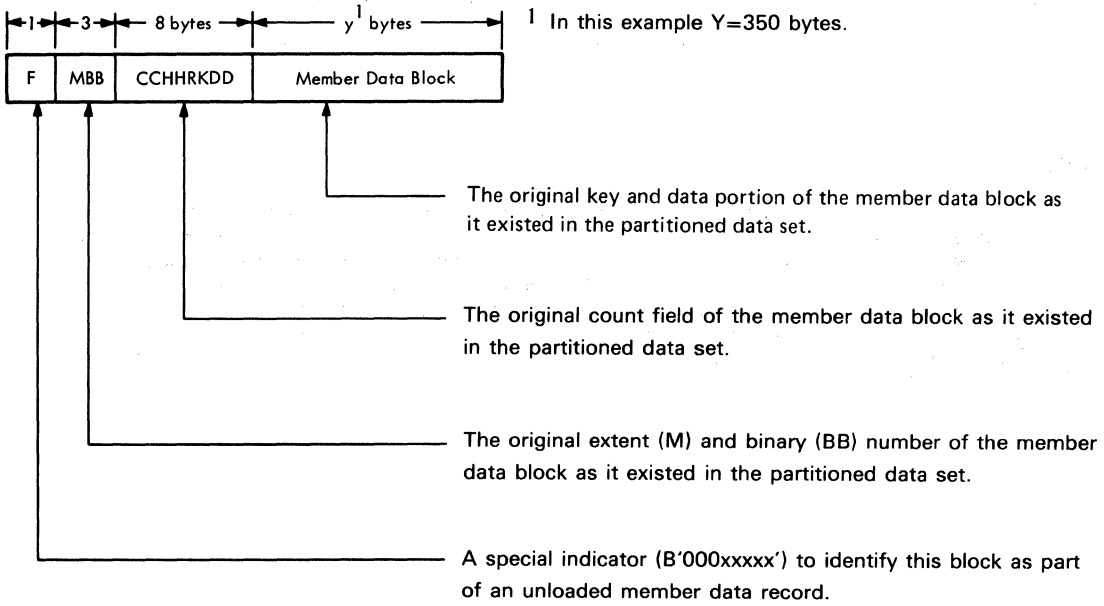
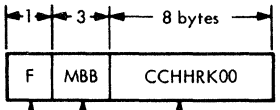


Figure 46. Example of the Format of a Member Data Block

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1



The original count field of the end-of-member indicator as it existed in the partitioned data set.

The original extent (M) and binary (BB) number of the end-of-member indicator as it existed in the partitioned data set.

A special indicator (B'000xxxxx') to identify this block as part of an unloaded member data record. (Bits 3-7 are reserved.)

Figure 47. Direct Access End-of-File Block Format for an Unloaded Member

MODULE DESCRIPTIONS

The following sections describe the various modules (and subroutine) of the IEBCOPY program. The descriptions are in alphabetic order by module name. For the main load modules, the descriptions contain the entry points, the inputs and outputs, the modules from which the given module receives control, and the modules to which the given module may pass control. Figure 48 on page 78 summarizes the control transfer interfaces among the program's modules.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

	IEBDSCPY	IEBMKO	IEBMGI	IEBMBT	IEBMGD	IEBMNM	TTOCC	CCOTT	IEBVMS	IEBVCT	IEBVTT	IEBVTM	IEBVDM	IEBDVI	IEBMCM	IEBSCN	IEBDRB	IEBDRD	IEBDWR	IEBWSU	IEBIOE	IEBBAM	IEBCNVT	IEBDSU	IEBRSAM	IEBWSAM	IEBLDUL
IEBDSCPY		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	
IEBMKO							X											X									
IEBMGI							X											X						X			
IEBMBT								X	X			X		X						X							
IEBMGD																		X							X		
IEBMNM																		X		X					X	X	
TTOCC																							X				
CCOTT																							X				
IEBVMS																											
IEBVCT									X			X															
IEBVTT							X							X				X		X							
IEBVTM	X						X	X	X							X*		X		X							
IEBVDM			X		X		X		X					X				X		X							
IEBDVI							X		X			X															X
IEBMCM			X				X	X	X									X									
IEBSCN									X			X															
IEBDRB									X			X						X		X						X	
IEBDRD																					X						
IEBDWR							X														X						
IEBWSU							X	X	X			X							X								
IEBIOE								X	X			X															
IEBBAM																											
IEBCNVT																											
IEBDSU	X		X									X														X	
IEBRSAM																					X						
IEBWSAM																					X						
IEBLDUL									X			X															

*via IEBDSCPY

Modules (or subroutines) in column at left give control (via a BAL or BALR instruction) to the modules (or subroutines) given across the top of the table in each case where an "X" sign appears at the intersection of the column and row for the modules (or subroutines) involved.

Figure 48. IEBCOPY Program Module Interface Relationship

MODULE IEBBAM—BUFFER ALLOCATION ROUTINE

Module IEBBAM allocates I/O buffer space (primary allocation); moves the old-name/new-name member pairs from a table of selected members to the I/O buffer; builds the new-name table if a selective copy, unload, or load is done; and initializes pointers for the building of the CTLTAB (control table).

ENTRY POINT: IEBBAM

INPUT:

- Opened data sets
- Block size and track capacity for the input and output data sets
- Pointers to available storage

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

OUTPUT: Allocated I/O buffers and new-name table if a selective copy, unload or load is done.

RECEIVES CONTROL FROM: IEBDSCPY

PASSES CONTROL TO: Caller

LOGIC: For each input data set encountered, module IEBBAM allocates I/O buffer space, taking the following into consideration:

- The amount of storage available.
- For data blocks ≤ 276-bytes in size, the minimum program requirement for two 276-byte buffer areas. (Each area can contain a 256-byte directory block, an 8-byte key length, a 1-byte flag field, a 3-byte MBB field, and an 8-byte count field for the current record.)
- For data blocks larger than 276 bytes, a program requirement for two buffers, each of which can contain the larger of:
  - An output data set data block plus key plus 12, or
  - An input data set data block plus key plus 12.

  Note: Whenever an unloaded input or output data set is involved, the original block size is ignored and replaced by the block size of the sequential data set.
- The track capacity (number of data bytes per track) for the input device if the input data set is partitioned and the output device if the output data set is partitioned.

  (If the compress-in-place operation is to be performed, the required buffer space is equal to twice the track capacity for the 'output' device that is used.) If available, an I/O buffer of four times track capacity will be used.

Based on the preceding considerations, the module determines the minimum space required for a two-buffer area, each area having the same size. (If minimum space is not available, the module gives control to the terminate module, IEBVTM, to end the processing of the current copy operation.) If calculations show that the minimum space is available, the module attempts to double that space. If successful, and if still more space is available, the module attempts to allocate storage for enough space to contain the information that can be contained on four tracks of the larger of either the device used for the input data set or the device used for the output data set, whichever provides the greater storage capacity. If track overflow is specified and if one buffer area is not greater than the output track block size capacity, module IEBBAM provides for the overflow of a record from one track to the next by adding 12 bytes to twice the size of the buffer area.

If the first buffer area is greater than the output track block size, the module calculates a factor whose value is equal to 12 times the ceiling quotient plus one. (The ceiling quotient represents the maximum number of times an overflow may occur when writing from a full buffer. This quotient is obtained by dividing the number of bytes in the previously determined buffer area by the output track block size value. If this quotient is not an integer, it is rounded to the next higher integer.)

If there is not enough storage available to contain the overflow storage factor (described above), module IEBBAM attempts to shorten the first buffer by the amount of the overflow factor. If the buffer can be shortened, or if there is enough storage for the two buffer areas plus the additional factor, the buffer control block can be completed.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

If there is enough storage for 4 tracks in the buffer, a CCWAREA is allocated. The size of this area is 1/32 of the 4-track area (8 bytes, one CCW per 128 bytes of buffer).

IEBBAM then extracts any old-name/new-name member pairs from the SETAB table and moves them to the I/O buffer (refer to Figure 35 on page 61, part B, for format).

Figure 49 on page 81 illustrates the buffer allocation process in a generalized manner.

MODULE IEBCNVT—LOAD TTR AND MBBCCHHR CONVERSION ROUTINE

This module handles all input data set TTR (relative record address) to MBBCCHHR (full disk address) and MBBCCHHR to TTR requests during a load operation. It also provides the principal means for device-independent and selective-load processing.

ENTRY POINTS: IEBCNVT and IEBCNVT1

INPUT:

Entry Point: IEBCNVT—TTR to be converted in register 0. Address of an 8-byte field for the resulting MBBCCHHR in register 2. Address of the unloaded data extent block (DEB) in register 1.

Entry Point: IEBCNVT1—Address of MBBCCHHR to be converted in register 2. Address of unloaded DEB in register 1.

OUTPUT:

Entry Point: IEBCNVT—MBBCCHHR placed in field originally pointed to by register 2 on entry to IEBCNVT.

Entry Point: IEBCNVT1—TTR in register 0.

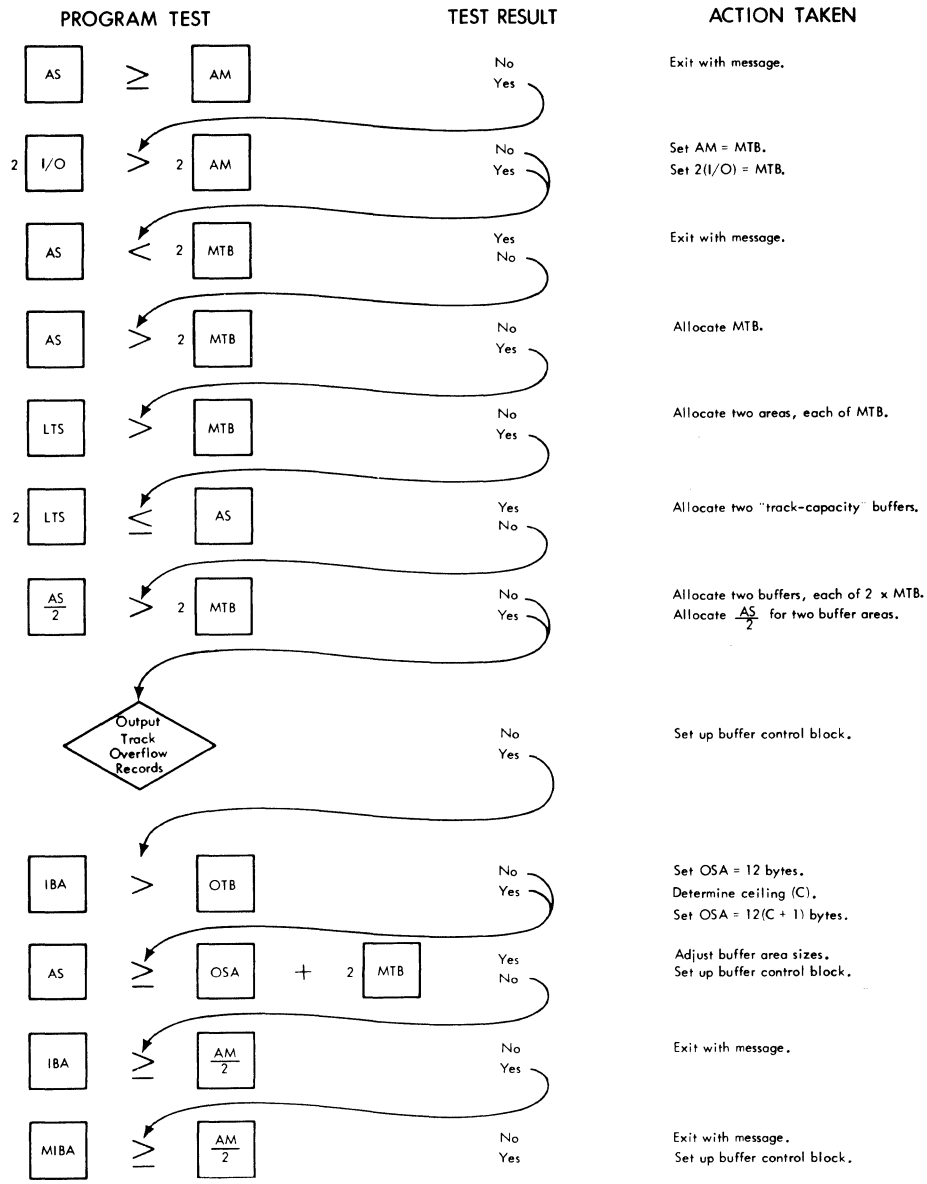
RECEIVES CONTROL FROM: Module IEBDSCPY

PASSES CONTROL TO: Module IEBDSCPY

LOGIC: Module IEBCNVT converts TTRs or MBBCCHHRs for partitioned data sets that were unloaded from devices not supported in the system under which the load is to be done. The input TTR or MBBCCHHR is converted to a full or relative disk address by using the DEB extent descriptions in the COPYR2 descriptor record and the UDEVTYPE field of the COPYR1 descriptor record. These fields contain the device type characteristics of the partitioned data set when it was originally unloaded.

If the conversion is successful, return is given to the caller with a return code of 0 contained in register 15. If the conversion is not successful, the return code is 4.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1



AM = Absolute minimum storage needed (=2 x [directory block length + key length + 12]).
AS = Available Main Storage (at time of 'test').
C = Ceiling (one more than either the integral quotient or the rounded-up fractional quotient of [buffer area + output track blocksize]).
IBA = Input Buffer Area available.
I/O = Larger of [input blocksize + keylength] or [output blocksize + keylength].
LTS = Larger of [track blocksize capacity for input (data set) device] or [track blocksize capacity for output (data set) device].
MIBA = IBA - OSA.
MTB = Minimum total buffer size required (based on copy step requirements).
OSA = Storage allocated for overflow records.
OTB = Output track blocksize (from DEVTYPE macro instruction).

Figure 49. Generalized Buffer Allocation for IEBCOPY

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

MODULE IEBCPMOD—ALTERMOD/COPYMOD ROUTINE

At entry point IEBCPMOD, the module copies and reblocks and/or alters load modules. Altering involves updating the RLD counts in the member PDS directory entries and in the control records of the member.

At entry point IEBCPMSG, the completion messages are issued for ALTERMOD and COPYMOD operations.

ENTRY POINTS:

IEBCPMOD—Copy and reblock and/or alter a member.

IEBCPMSG—Issue completion messages for ALTERMOD and COPYMOD.

INPUT

- Switches indicating whether an ALTERMOD or a COPYMOD operation is to be performed
- A maximum (MAXBLKSZ) and minimum (MINBLKSZ) output block size for COPYMOD operations
- Opened data sets
- Input directory entry address
- CTLTAB addresses—start of table and current entry
- Pointers to available storage and I/O buffers

OUTPUT

- Copied and reblocked and/or altered member
- Switches indicating the action performed
- Messages indicating the action performed (entry point IEBCPMSG)

RECEIVES CONTROL FROM: Modules IEBDSCPY and IEBVTM.

PASSES CONTROL TO: Modules IEBWSU, IEBRSAM, IEBVMS, IEBVTM, and IEBDSCPY (subroutine TTOCC).

SUBROUTINES

CALBLOCK	Calculate the output block size
DOALTER	Update control records per CTL table
EOD	End-of-data processing
GETINPUT	Reads or moves input text to the output buffer
GIVEMSG	Issue a message via IEBVMS
IOMOVE	Move characters from buffer to buffer
MODERROR	Handle members which cannot be reblocked by COPYMOD
READREC	Read or load a text record into a buffer
READRLD	Read RLD records and save the data in storage
RSAM1	Call IEBRSAM, read a sequential file (within READREC)
SAVECTL	Save the data from the control record
SETFREE	Set FREECORE to a new value and test for storage overrun

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

SYNADERR	Issue SYNAD error message and quit
TESTCTL	Add control records to CTL table for later update
UPDATE	Update RLD counts in control records
WRITECTL	Write control records
WRITEOUT	Write the output buffer and associated records
WRITEREC	Call IEBWSU, write a record from a buffer
WRITERLD	Write RLD records from data saved in storage

LOGIC: Upon entry at entry point IEBCPMOD for the first time for an input data set, a DCB is opened against the data set and the local I/O buffers are defined.

After the data set initialization, counters, flags, and pointers are reset for the current member, and the input PDS directory entry is validated.

Note that, at various points throughout the ALTERMOD/COPYMOD processing, checks are made to ensure that the current member is a valid and processable load module (that is, not in scatter-load, overlay, or noneditable format). Members that cannot be processed are identified by checking the length and flags of the PDS directory entry, the types of records encountered, and the lengths of the nontext records. The lengths of these latter records must equal the length specified in the length field of the records. In addition, the lengths of the text records must equal the text record length as specified in the preceding control record.[2]

If a record that cannot be processed is found, control is returned to module IEBDSCPY and the member is either skipped (for ALTERMOD) or copied (for COPYMOD) as if a COPY control statement had been specified.

The module then reads all records prior to the first text record. The records are validated and, in the case of a COPYMOD operation, written to the output member. The read function is performed via procedure READREC, which uses BPAM reads (in the case of a load operation, module IEBRSAM is called). Write functions throughout processing (except when control records are updated) are done through procedure WRITEREC. WRITEREC sets up the records within the I/O buffer and calls module IEBWSU (which calls module IEBDWR) such that each record is written immediately.

When CESD records are encountered in this initial process, and COPYMOD is specified, the CSECT data—length, relative address, and CESD offset—is stored in the ESD data table (ESDT). This information is used later by procedure CALBLOCK to calculate the output block size and to build the control record data.

Input record processing ends after the first control record has been read. At this time, the relative track address (TTR) and RLD count of the record are saved in the control record table (CTLT). This information is used when the record's RLD count needs updating. The entry is removed from the CTLT if the RLD count is found to be correct.

At this point, processing for ALTERMOD and processing for COPYMOD become independent, and are described separately.

ALTERMOD Processing: Text records are read and verified for length. RLD records are then read, verified, and counted.

---

[2] For more information on the contents of load modules, see Linkage Editor Logic.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

When a control record is read, it is verified and entered in the CTLT. The previous record's CTLT entry is also checked. If it is found to be correct, the entry in the CTLT is removed. If the CTLT entry is in error, the RLD count in the entry is corrected.

When the last member is processed, the last CTLT entry is checked and either removed or updated to reflect the current RLD count. Control records with entries in the CTLT, if any, are then read, updated with the correct RLD count, and rewritten. Writing for updating uses BPAM unformatting write procedures.

The RLD count in the PDS directory entry is then checked and updated (if necessary) and control is returned to module IEBDSCPY.

At entry point IEBCPMSG, module IEBCPMOD creates and writes (via module IEBVMS) the completion messages for the ALTERMOD operation just completed.

COPYMOD Processing: When COPYMOD is specified, the data in the ESDT is sorted by relative address. The lengths of CSECTs in the table are also rounded up to doubleword boundaries.

After pointer and count initialization, all the CSECTs listed in the ESDT are copied.

First, procedure CALBLOCK is used to compute the output block size. The size returned is based on the following factors:

- The values of MAXBLK and MINBLK specified or defaulted.
- The space remaining on the output track.
- The sizes of the CSECTs to be put into the output block. (Note that, if one entire CSECT cannot fit within the computed maximum block size, a single CSECT may be split into one or more pieces. However, a text block containing text from a split CSECT may contain text from no other CSECTs.)

Procedure CALBLOCK also builds the data for the control record associated with the text block.

After return from CALBLOCK, procedure GETINPUT is called to read enough input text blocks to fill the computed output block size. Text blocks are filled one CSECT (or part of a CSECT) at a time, so that the linkage editor CSECT constraints can be maintained. CSECTs that were split in the input text blocks may be reconstructed for the output text blocks.

After each input text block has been read by GETINPUT, procedure READRLD is called to read and save the RLD data associated with that text block. The data from the RLD and RLD-control records is saved in a hold area. This information is used when the RLD records are rebuilt.

Control is returned to the main module flow, where procedure WRITEOUT is called to write the control, text, and RLD records for a text block that procedure GETINPUT has built.

WRITEOUT first passes control to procedure WRITECTL, which builds and writes the control record for the text block. If an RLD record exists from a previous text block, the control data for the current text block may be inserted within it to produce an RLD-control record. If the control data will not fit within this record, the RLD record is written and a new control record is constructed. The control data used in the control record is that which was produced by the procedure CALBLOCK.

Procedure WRITECTL also makes an estimate of the number of RLD, RLD-control, and control records that will follow the current text record and inserts this estimate in the current control record. The TTR of the control record and the estimated RLD

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

count are saved in the CTLT. At the same time, the previous control record's CTLT entry is checked. If the RLD count for that entry is correct, the CTLT entry is removed. If not, the correct RLD count is inserted in the table so that the control record can be updated. Only control records with incorrect RLD counts have CTLT entries which remain in the table.

After the control record is written, control returns to procedure WRITEOUT, where the text record is written.

Control is then passed to procedure WRITERLD, which builds and writes RLD records until all RLD data for the current text record has been written. Note that, if space remains in the last RLD record, the record is not written, in anticipation that this space could be used by the next control record.

If the last record of a member is processed and no more control records are built, procedure WRITECTL is called and the dangling RLD record, if any, is written. In addition, the end-of-member condition causes procedure WRITECTL to check the last control record's CTLT entry, removing it if the RLD count is correct. In this case, procedure WRITECTL is called and no control record is built or written.

Upon return from procedure WRITERLD, control is returned to the main module flow. The CALBLOCK/GETINPUT/WRITEOUT loop continues until the end-of-member condition is found.

For any CTLT entries that exist, the corresponding control record is read, the RLD count is updated, and the record is rewritten. In this case, BPAM is used to read and write without formatting.

The information in the PDS directory entry is then updated and control is returned to module IEBDSCPY.

At entry point IEBCPMSG, module IEBCPMOD creates and writes (via module IEBVMS) the completion messages for the COPYMOD operation just completed.

MODULE IEBDRB—REBLOCK/DEBLOCK ROUTINE

The primary purpose of this module is to block or deblock the fixed- or variable-length records that are being moved from an input buffer to an output buffer. The block size required for the output data set record is used to determine whether deblocking or (re)-blocking is to be done. The module calls modules IEBDRD and IEBWSU as needed, and the processing continues until an end-of-file record is read from an input buffer and written on the output buffer. In the case of a load operation, module IEBRSAM is called instead of IEBDRD.

ENTRY POINT: IEBDRB for initial entry and to handle fixed-length records.

INPUT: The members of the input data set currently being processed.

OUTPUT: An output data set whose records have been properly block-formatted.

RECEIVES CONTROL FROM: Module IEBDSCPY

PASSES CONTROL TO: The calling module as a normal exit.

Module IEBVMS to have a message placed in SYSPRINT.

Module IEBDRD to read member records into input buffer.

Module IEBRSAM to read the members of the unloaded data set into the buffer.

Module IEBWSU to set up output addresses for the write routine.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

SUBROUTINE: VARIAB routine is used for variable-length record processing.

LOGIC: Because reblocking or deblocking cannot be done on records having overflow characteristics, a test of the FORM switch, set by the validate routine, indicates whether processing by the module can continue. If overflow records are indicated, a message is written and the copy or load operation is terminated. Otherwise, the current input member is read into the input buffer, the record format is determined, and the pertinent buffer addresses are placed in registers.

After an input member record has been read into the input buffer, a test is made to determine if the output buffer is full. If it is, the write routine, IEBDWR, writes the data from the output buffer after preliminary operations by the write-setup routine, IEBWSU. Otherwise, an input logical record length (LRECL) of data is moved into the output buffer (one increment of LRECL at a time) until either the output buffer is full or an output record block has been completed. At that time, count field data for the output record is recorded. If the next record to be read is an end-of-file record, either an end-of-file record is then built in the buffer, if there is room, or the buffer will be written out and then the end-of-file record will be written as the first record in the emptied buffer.

After writing the last record from the output buffer, a return is made to the routine from which IEBDRD received control.

In general, the same type of processing is performed whether the input records are fixed length or variable length. However, for variable-length records, the length of a logical record (LRECL) +4 must be less than, or equal to, the output data set block size if complete processing is to be done.

After an output buffer has been written out, the processing continues with the moving of an LRECL amount of an input record from the input buffer to the output buffer.

MODULE IEBDRD—READ ROUTINE

Module IEBDRD receives control when data is to be read by the program. This data may consist of members, note lists, and directory entries. Depending on the data, it may be read from any of the following data sets:

Input
Output
SYSUT3
SYSUT4

All reading is performed by issuing an EXCP macro instruction for a channel program previously established in this module.

ENTRY POINT: IEBDRD

INPUT: Switches to indicate the data set that is being read. Pointers to indicate the area (buffer) into which data is to be read. (This includes the address of the next location into which the next record is to be read.)

OUTPUT: Data read into specified location(s). Switches indicating: end-of-file, last record read into a buffer, and last record read into last buffer.

RECEIVES CONTROL FROM: Modules IEBDSCPY (including subroutines IEBMGD and IEBMNM), IEBVTM, IEBVDM, IEBMCM, and IEBDRB.

PASSES CONTROL TO: Module IEBIOE

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

LOGIC: The module determines the type of data to be read. If input member data or output directory blocks are to be read, the block sizes are determined (see Figure 50 on page 87), and the channel command words constituting the 'Read' channel program are established.

Data Type	Block Size
SYSUT3 directory entry	80 bytes
SYSUT4 directory entry	264 bytes
Input directory blocks	264 bytes
Output directory blocks	264 bytes
Input member data	(input block size plus input keylength)

Figure 50. Block Sizes of Data Types Read by Module IEBDRB

This module does a 'PCI-read' (see below) when it is reading member data, when it is reading from SYSUT4, or when it is reading input directory blocks and there is insufficient storage to hold 4 tracks and the CCWAREA (except from SYSUT3).

When reading entries from SYSUT3, when reading note lists, or when there is track overflow, the module uses a channel program that reads one record at a time (that is, uses a Search ID Equal command each time another record is to be read).

If the device type information indicates the presence of a rotational position sensing (RPS) device, a Set Sector/TIC CCW chain is appended to the normal channel program. In addition, all CCW chains are constructed to end with a Read Sector CCW.

The PCI-read is used to save time in that, as one record's data is being read, the PCI appendage routine (IGG019C8) is entered (via the I/O supervisor) to calculate the address into which the next record will be read. A switch is set to indicate to appendage IGG019FT that it is necessary to give an extended translation list to the I/O supervisor. PCI appendage routine IGG019C8 then uses the key and data length portions of the count field for the record currently being read to determine the number of bytes to be read for that record. Then, using the location of the start of the current record and allowing for the FMBB field (4 bytes), the count field (8 bytes), and the key and data fields (x bytes), the new address can be computed.

The PCI appendage routine also uses the address of the end of the existing buffer space to determine if at least one 'block size-length' record plus the FMBB and count fields can be included in the buffer after the record currently being read. As the appendage module determines the availability of buffer space and places the next address (for the record to be read) into the CCW, it changes the NOP or Read Sector (RS) command word (which follows the Read Key and Data (RKD) command word that has the PCI bit set on) to a TIC command with a transfer address to the next Read Count (RC) command in the channel program.

The following illustration and text show how the channel program for reading member data, etc. is set up and modified.

Assume the channel program (set by module IEBDRD) shown in Figure 51 on page 88, with CCWs numbered as shown in the left column. Arrows represent pointers to CCWs. Alphabetic superscripts represent changes to the channel program as described (and keyed) in text.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

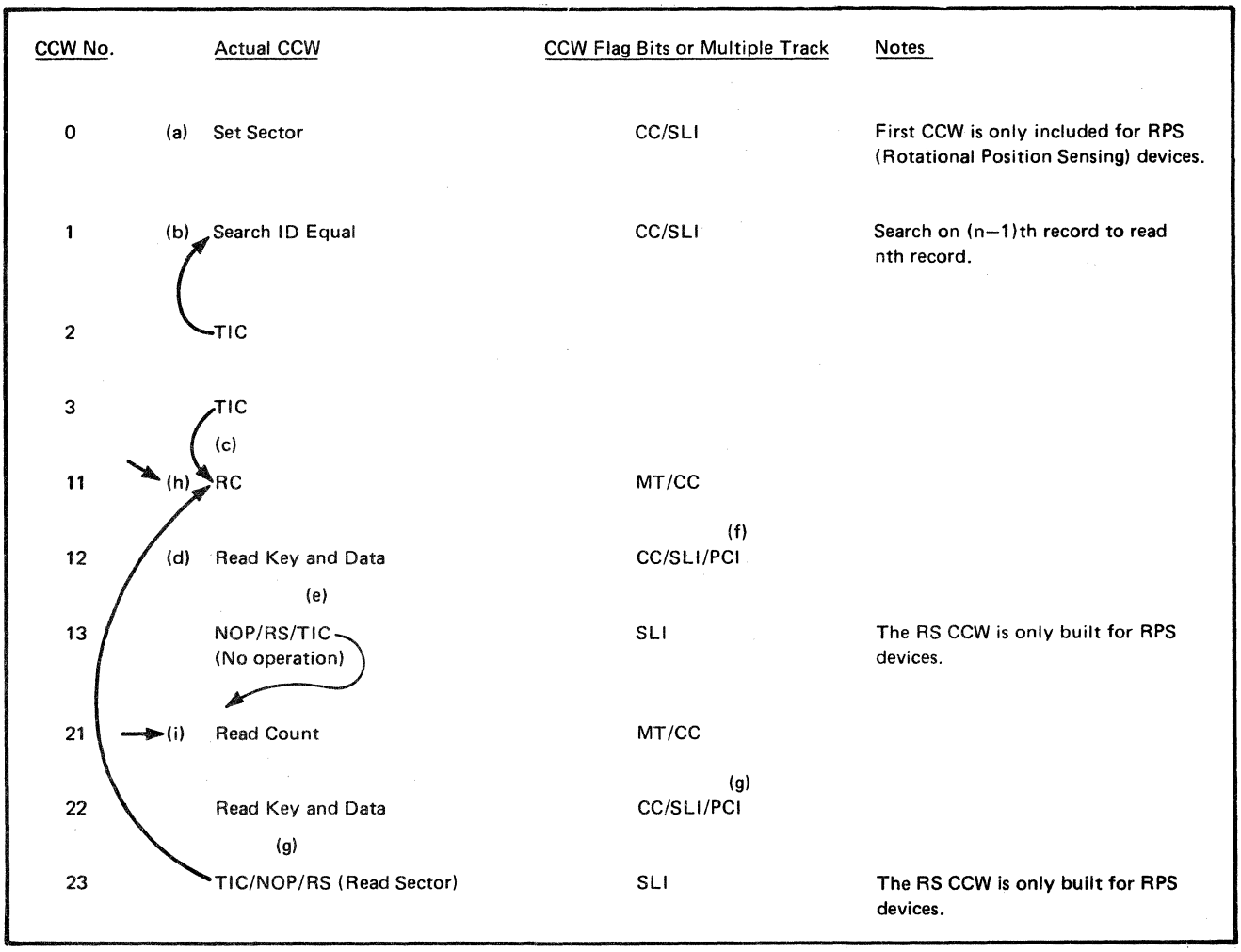


Figure 51. Channel Program for PCI-Read of Data

Sequence of processing shown in Figure 51:

(a) CCW0 is used for sector orientation on RPS devices only.

(b) Execute CCW1, CCW2 to find the record to be read. Then execute CCW3.

(c) Execute CCW11 to read the record count field. The MT indicator permits head switch to next track if necessary.

(d) Execute CCW12 to read key and data of record. During this reading, the PCI appendage module gets control concurrently. The module uses the key and data length information (read into the buffer by CCW11) to calculate the address of where the next record should be read. The appendage module also determines whether the buffer has enough space for one more 'block size-length' record, and whether there are more input records to be read.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

(e) If there is space available, and if there are more records, the appendage module changes CCW13 to a TIC command with a transfer address for CCW21. Otherwise, CCW13 remains as a NOP or RS to cause the channel program to stop after the current record is read.

(f) The appendage module turns off the PCI bit in CCW12 to suppress entry to the PCI appendage in case error retry procedures become necessary for CCW12.

(g) The appendage module turns on the PCI bit for CCW22 and sets CCW23 to a NOP.

(h) Then, the module exchanges the pointers to the command words (in module IEBDRD) so the 'next' pointer will indicate CCW11 and the 'current' pointer will indicate CCW21.

The appendage module then places the track address (CCHHR) of the last-record-read into the IOB being used for the reading of the data set so that the retry procedures can use it if an I/O error occurs. Module IGG019C8 then returns control to the I/O supervisor, which returns control to IEBDRD.

If there was sufficient storage available for IEBBAM to allocate a 4-track size buffer and the CCWAREA, directory blocks are read with a non-PCI channel program that is dynamically constructed, on each invocation of IEBDRD, in the CCWAREA. The channel program consists of a string of Read Count, Key, and Data CCWs terminated by a NOP or Read Sector.

The number of CCWs constructed is a function of the I/O buffer and CCWAREA size. One CCW is built for each 276 bytes (directory block plus flag and MBB) of the I/O buffer or the CCWs are built until the buffer or the CCWAREA is exhausted. If the CCWAREA is exhausted before the I/O buffer, EXCP is issued to fill the portion of the buffer covered by the CCW string, and another string of CCWs is built and executed until the I/O buffer is full or an end of file is encountered. The string of Read Count, Key, and Data CCWs is transferred to or from CCW3 of Figure 51. CCWs 0, 1, and 2 perform the same functions as discussed with PCI.

When not using PCI, IEBDRD performs four functions that are implemented in IGG019C8; they are: counting the number of directory blocks, determining the highest record number on any track, appending the FMBB to the front of each directory block read, and computing the track balance following the last directory block.

If an RPS device is used, IEBDRD updates the sector value associated with the data set being read, before returning control to the caller. (The value used for updating is the last sector address read.)

When reading records that are in a data set specifying track overflow, a different form of channel program is used. The records of such a data set were initially placed on the direct access device by means of a Write Special channel command word. The overflow bit in the flag byte preceding the count field of each record segment that overflows to another track is turned on. When an overflow record is subsequently read, the count field preceding each overflow segment of the record is bypassed, and only the data portion of the record is read. The channel program for reading this type of record is indicated in Figure 52 on page 90.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CCW No.	Actual CCW	CCW Flag Bits or Multiple Track	Notes
0	Set Sector	CC/SLI	The first CCW is only included for RPS devices.
1	Search ID Equal	CC/SLI	Search on (n−1)th record.
2	TIC		
3	TIC		
11	Read Key and Data	CC/SLI/SKIP	Dummy read (of record n−1). (No transfer of data.)
12	Read Count, Key, and Data	MT/SLI/CC	Read record n.
	Read Sector	SLI	Only included for RPS devices.

Figure 52. Channel Program to Read Data when Track Overflow is Specified

MODULE IEBDSCPY—MAIN FLOW ROUTINE

The IEBDSCPY module is the controlling module for the IEBCOPY program. In addition to determining which of the other program modules is to receive control, it performs several other functions of its own by means of subroutines contained within it. These functions include:

- Program initialization
- Storage allocation
- Directory scanning
- Control table construction and the saving of directory entries
- Conversion of TTRs to MBBCCHHRs
- Conversion of MBBCCHHRs to TTRs
- The returning of pointers to output directories in storage
- Preparation for copying a member
- The releasing of storage for a BSAM work area
- QSAM I/O error handling

ENTRY POINTS:

IEBDSCPY—Initial entry: Storage allocation; INDD processing initialization.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEBDSCME—For reissuing a GETMAIN if IEBVTM issued a FREEMAIN between copy operations.

IEBDSCNI—I/O buffer allocation; directory scans.

IEBDSFRC—Used by module IEBLDUL to release main storage.

IEBDSCNM—Preparation for member copying without blocking function.

IEBDSCEF—Interface for end of data on SYSIN.

IEBDSOE2—Interface for end of member data.

IEBDSCCE—SYNAD exit for SYSIN.

IEBDSCPE—SYNAD exit for SYSPRINT.

IEBDSCSX—Interface between IEBVTM and IEBSCN modules.

IEBDSCP2—Entrance to the CSECT containing subroutines in the root segment of the program.

INPUT: Input and output directories referencing members to be processed. An optional parameter list from a calling routine or program.

OUTPUT: An output directory referencing copied members, renamed members, and merged members, according to the parameters on the control statements for the program.

RECEIVES CONTROL FROM: The operating system, or another user (via either the LINK or ATTACH macro instruction).

PASSES CONTROL TO MODULES:

IEBDV1—to open and validate data sets.

IEBSCN—to scan control statements and construct INDD and SETAB tables.

IEBDRD—to read directories and member data from an input device into buffers.

IEBVCT—to sort the select/exclude table entries.

IEBVTT—to sort control table entries.

IEBCPMOD—to copy and reblock or alter a member.

IEBWSU—to prepare for writing data from buffers.

IEBDWR—to write data from buffers to output device.

IEBVDM—to merge input and output directory entries in buffers.

IEBVTM—to perform termination processing.

IEBVMS—to print error messages, diagnostic messages, and control statements.

IEBMCM—to perform 'compress-in-place' processing of members that do not have to be moved (are already in place).

IEBDRB—to perform deblocking or reblocking of members being copied.

IEBIOE—to handle I/O errors in all data sets except SYSIN and SYSPRINT.

IEBBAM—to allocate primary I/O buffers.

IEBDSU—to handle directory entries in the case of load and unload operations.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEBRSAM—to read data from an unloaded data set into buffers.

IEBWSAM—to write data from buffers to an unloaded data set.

IEBCNVT—to handle all input TTR and MBBCCHHR conversion requests in the case of a load operation.

PASSES CONTROL TO SUBROUTINES:

TTOCC—Converts member TTR address to MBBCCHHR form.

CCOTT—Converts actual MBBCCHHR address to TTR form.

IEBMGD—Obtains pointer to a directory entry in the I/O buffer.

IEBMBT—Builds CTLTAB table and saves directory entries.

IEBMGI—Obtains pointer to an input directory entry.

IEBMKO—Saves output directory blocks in main storage.

IEBDSFRC—Allocates space for descriptor records and BSAM modules.

LOGIC: Upon initial entry to this module at IEBDSCPY, tests are performed to determine if the IEBCOPY program was invoked (via either an ATTACH or a LINK macro instruction) and if so, to what extent the invoking program's parameter list will modify the standard program parameters. The module then gives control to the validation module, IEBDV1 (at entry point IEBDV0), to open and validate both the control data set (SYSIN) and the message data set (SYSPRINT). (Throughout the description of the modules of this program, normal (or error-free) processing is assumed unless otherwise indicated.)

Upon return from the validation module, module IEBDSCPY obtains all available main storage in which to set up the necessary buffers and work areas (including control tables). See "Storage Allocation."

The module then gives control to the scan module, IEBSCN, to read and analyze (interpret) control statements. The scan module uses the information in the control statements to build the following:

- INDD table, which contains the names of input DD statements (scanned from the INDD keyword or control statements).
- SETAB table, which contains the names of members (if any) to be selected, renamed, or excluded from an input data set. This table is not built if there are no members to be selected or excluded from an input data set.
- OUTNAME field in the communication area. This field contains the name of the current output data set DD statement.

  Note: When ALTERMOD is specified, there is no INDD parameter. In this case, the OUTNAME field is used to construct the INDD table (containing one INDD name).
- MODLIBSW field to indicate whether a COPY, COPYMOD, or ALTERMOD control statement was read.
- MINBLKSZ and MAXBLKSZ fields to specify the minimum and maximum block sizes to be used during a COPYMOD operation.

After the control statements have been scanned, module IEBDSCPY performs further initialization functions and then gives control to module IEBDV1 to open and validate either:

- The output data set and the first input data set in the current copy operation (entry point IEBDV1), or

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- The next input data set in the current INDD statement group (entry point IEBDV2).

Module IEBDV1 checks for such parameters as logical record length, block size, I/O device characteristics, and record format. (An output data set will be opened only when the current input data set is the first one to be processed for this output data set.) If module IEBDV1 encounters a sequential input or output data set (load or unload function), control is given to module IEBLDUL to open these data sets. (If the input data set is in an unloaded form, the descriptor word is read. If it is the first unloaded data set to be processed, IEBLDUL gives control to IEBDSCPY subroutine IEBDSFRC.) This module then returns control to module IEBDSCPY, which gives control to module IEBBAM. Module IEBBAM then allocates the initial I/O buffer area and moves the old-name/new-name member pairs from the SETAB table to the I/O buffer. Control is then returned to module IEBDSCPY.

The next major process is the building of control table (CTLTAB table) entries for directory entries for members to be copied from the current input data set. The entries that will be used for CTLTAB table entries are in the SETAB table if the operation is one of selecting members. Module IEBVCT receives control from module IEBDSCPY to sort the SETAB table member in alphameric sequence. Module IEBDRD or module IEBRSAM then reads the input directory into an I/O buffer, ceasing either when the buffer is full or when an end-of-file (EOF) for the directory has been reached.

Entries in the SETAB table are compared to names contained in the directory entries in the buffer. If the operation being performed is a 'select' operation, subroutine IEBMBT of module IEBDSCPY then constructs a CTLTAB table entry corresponding to each member name for which there is a match. For an 'exclude' operation, subroutine IEBMBT constructs a CTLTAB table entry corresponding to each member name for which there is not a match. For a full copy operation, the subroutine constructs CTLTAB table entries corresponding to all member names in the current input data set. This process continues until either there are no member names left in the SETAB (for a 'select' copy operation) or there are no entries remaining in the directory.

Subroutine IEBMGD (in module IEBDSCPY) is called to locate within the I/O buffer, and module IEBDRD or module IEBRSAM is called to read the entries in the directory as they are needed. Subroutine IEBMBT constructs the CTLTAB table entries to either overlay corresponding entries in the SETAB table (for a select copy operation) or occupy new space in the CTLTAB table (for an exclude or a full copy operation). This procedure is more fully discussed under "Major Tables Used by the IEBCOPY Program."

Subroutine IEBMBT also determines whether there is sufficient main storage to contain the input directory entries for members that have been found (that is, those members which are to be copied from the current input data set). If there is no storage available, module IEBMBT writes (spills) the directory entries onto the SYSUT3 data set.

Following the building of the CTLTAB table entries, the output directory is 'prescanned' to prevent the processing of duplicate-named input members (unless the REPLACE option has been specified on either the member name level or the INDD level). (**Note:** The output directory prescan is not performed if the operation is one of compressing a data set, when the operation is an ALTERMOD, or when an unload function is performed.) Module IEBDRD reads the output directory into the I/O buffers (which are contiguous at this time), and subroutine IEBMKO (of module IEBDSCPY) will retain as many directory blocks in main storage as space permits. (If module IEBVDM has previously merged the directory to the SYSUT4 data set rather than to the output data set referenced by the current OUTDD keyword, it is read in from there.) If input directory entries have been spilled onto the SYSUT3 data set, no attempt is made

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

to retain output directory blocks in main storage. (The spilling of input directory entries indicates that main storage is not available for output directory blocks.)

To compare output directory member names with names from the CTLTAB table, module IEBDSCPY uses two subroutines from CSECT IEBDSCP2. Subroutine IEBMGD obtains a pointer to the member names as needed from the output directory; subroutine IEBMGI obtains a pointer to the input member names by using CTLTAB table information to access the input directory entries. If the REPLACE specification is absent for a member name that appears on both 'lists' (input and output), module IEBDSCPY flags the corresponding CTLTAB table entry so that it will be ignored in subsequent processing.

The scanning of the output directory continues until either there are no more names remaining in the directory or until there are no more input member names to check against. As a current 'buffer-full' of output directory entry names has been checked, module IEBDRD receives control to read in more members until one of the above conditions is satisfied. After the CTLTAB TTR sort by module IEBVTT, in the case of an unload operation, control is given to module IEBWSAM to write the descriptor records onto the unloaded output data set. Then, module IEBDSU receives control to set up the directory for the unloaded data set and control is returned to module IEBWSAM to write the directory records onto the unloaded output data set. If subroutine IEBMKO can retain output directory blocks in main storage, it may be that all relevant input and output directory entries (pertaining to the current input data set in the current copy step) can be simultaneously held in main storage.

If module IEBDSCPY is processing the current output data set directory for the first time (that is, the first pass through the output directory), it determines the number of output directory blocks that are allocated to the output directory space on the output device. This information is used during a subsequent operation of merging the input and output directory entries.

After all input and output directory entries for a given copy step have been accounted for (for example, as being 'found,' 'duplicated,' 'replaceable,' etc.), module IEBVTT receives control to sort the pertinent CTLTAB table entries into a sequence based on the relative track address (TTR) of the input members to be copied in the current step. This sequence is used during the program's copy phase.

When module IEBVTT has completed the sorting of the CTLTAB table entries, module IEBDSCPY determines the amount of available (unused) main storage remaining (if output directory blocks have been retained in main storage). If there is an available area that is at least equal in size to the total I/O buffer area, the module reassigns the original buffer area as one I/O buffer and assigns this available area as another I/O buffer. These buffers may be used interchangeably for input or output operations except for blocking considerations. If the data set is being compressed-in-place or if track overflow is specified for the output data set, the module does not perform this second buffer allocation. If the data set is being compressed-in-place, then, instead of allocating buffer storage, module IEBDSCPY gives control to module IEBMCM to perform the compress-in-place processing up to the point of locating the first member not already in place. If track overflow is specified, or after the second buffer has been allocated, module IEBDSCPY uses its subroutine IEBMGI to obtain the input directory entry for the next member that is to be copied. If the directory entry is an alias but there was not a preceding main member, the subroutine turns off the 'alias' bit (in the CTLTAB table entry) to make the entry appear as a main member. In all cases, the member's TTR is converted to the MBBCCHHR form so that the member can be read from the input data set.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Module IEBDSCPY then determines whether both blocking (or deblocking) and user and/or note list TTRNs have been specified. If so, processing of this data set cannot continue, because blocking (or deblocking) cannot be done when these TTRNs have been specified (because individual record identification would not be possible). If blocking or deblocking has been specified, module IEBDRB processes the member data and copies it onto the output data set. When unloading, module IEBDRB does not reblock the data. However, the unloaded records are formatted by module IEBWSAM, but, when loading, a default of the original block size is taken.

For the copying of members (when COPY is specified) with or without user and/or note list TTRNs, module IEBDSCPY uses its subroutine IEBMNM. Both module IEBDRB and subroutine IEBMNM use module IEBDRD to read members into a buffer area, and both use module IEBWSU to have the members prepared for writing from the buffer onto the output data set. (When the buffer is full, module IEBWSU gives control to module IEBDWR to write the members.)

In the case of an unload operation (when COPY is specified), control is given to module IEBWSAM instead of to module IEBWSU, to prepare for writing onto the unloaded data set. (Writing is done by module IEBWSAM even when the buffer is not yet full.) In the case of a load operation, control is given to module IEBRSAM to read the records from the sequential data set.

When COPYMOD or ALTERMOD is specified, control is given to module IEBCPMOD. This module uses its own BPAM read routine (except in the case of a load operation, when module IEBRSAM is used) and module IEBWSU to have the member data prepared for writing. (In this case, module IEBWSU gives control to module IEBDWR to write each record as it is prepared.) If a member is found to be in scatter-load, overlay, or noneditable format, or if a member is not a load module and COPYMOD has been specified, the input and output data sets are reset to the state they were in before IEBCPMOD was called. The member is then processed as if a COPY operation was specified. Otherwise, further COPY operations on the member are skipped.

After the member data has been processed for writing (whether or not writing has actually taken place), the member's updated (output) TTR is placed into the corresponding directory entry. If the directory entry was read from the SYSUT3 data set, its 'updated' version is written back "in-place" on SYSUT3. Module IEBDSCPY then uses subroutine IEBMGI to get another input directory entry (if there are any left in the input data set being processed). After all members in the input data set have been processed (including alias and TTR processing and updating), module IEBDSCPY determines if the I/O buffer contains any unwritten member data. If so, module IEBWSU receives control to cause the remaining member data to be written from the buffer onto the output data set.

After all members of the input data set have been copied and the I/O buffer has been cleared of data, module IEBDSCPY gives control to module IEBVDM and, in the case of an unload operation, to module IEBDSU for merging and writing appropriate input and output directory entries. When the merging has been completed, control is given back to module IEBDSCPY, which then gives control to the termination routine, module IEBVTM. IEBVTM checks to determine if further processing is needed for the current INDD group, copy step, or copy operation, and takes appropriate steps to continue or end the job.

Two other subroutines of module IEBDSCPY are the TTOCC and CCOTT subroutines. These subroutines receive control when it is necessary to convert track addresses. The TTOCC subroutine converts the relative track address (TTR) of data to an absolute address of the form MBBCCHHR. The CCOTT subroutine converts an absolute address to a relative address. In the case of a load operation, module IEBCNVT is given control when TTRs or MBBCCHHRs for the input DCB have to be converted and IEBDSCPY's

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

subroutines TTOCC or CCOTT cannot handle the conversion. (This depends on the device type on which the data set resided originally.)

Each subroutine obtains the address of the appropriate system conversion routine from the communication vector table and gives control to that routine when required.

MODULE IEBDSU—DIRECTORY SETUP FOR AN UNLOAD OPERATION

Module IEBDSU builds output directory blocks for an unloaded data set and restores member names from members that have been unloaded to the control table for use by the message routine.

ENTRY POINTS:

IEBDSU—to build an output directory from the input directory.
IEBDSU1—to restore member names to the control table.

INPUT:

- Switches indicating whether a 'new' name has to be moved in the directory entry and whether there are more entries to be processed.
- Pointers to the beginning and end of the buffer area in which to build the directory blocks.
- Input directory entry addresses.

OUTPUT:

- An output directory of the members that are to be unloaded.
- The control table containing the member names (not the member TTRs).
- A bit turned on in the control table to indicate that the member has been unloaded.

RECEIVES CONTROL FROM:

IEBMGI
IEBWSAM

PASSES CONTROL TO:

IEBWSAM
IEBVTM
IEBDSU
IEBDSCPY
IEBMGI (subroutine of IEBDSCPY)

LOGIC FOR DIRECTORY SETUP: The output directory is built from input directory entries only, because no merge is allowed while unloading. Module IEBDSU receives one entry at a time (in alphabetic order) from module IEBMGI; the entries are inserted into directory blocks until these are filled, and a new name is inserted if necessary. When there is no more space in the buffer, control is given to module IEBWSAM to write all preformatted data. Control is then returned to module IEBDSU and the same process as above is maintained until the directory has been totally processed. When there are no more entries to be processed, an end-of-file mark is built. Module IEBDSU then returns control to the root segment, IEBDSCPY.

LOGIC FOR UNLOADING MEMBER NAMES: Member names are restored to the control table and a bit is turned on to indicate that the member has been successfully unloaded. The termination routine, IEBVTM, uses this information to issue the appropriate messages.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

MODULE IEBDV1—DATA SET VALIDATION ROUTINE

Module IEBDV1 opens the following data sets and verifies their characteristics:

Input	partitioned data set
Output	partitioned data set
SYSUT3	
SYSUT4	
SYSIN	
SYSPRINT	

The module also determines, in the case of a partitioned data set to partitioned data set copy operation, whether blocking or deblocking has to be done. If an unload or load condition is found, control is passed to module IEBLDUL for opening the appropriate data sets.

If COPYMOD is specified, the values of MINBLKSZ and MAXBLKSZ are validated and corrected such that they fall within the parameters of the output device.

ENTRY POINTS:

IEBDV1—for opening and validating the output and the first input data sets of a copy operation.

IEBDV0—for opening and validating the SYSIN and SYSPRINT data sets.

IEBDV2—for opening and validating other input data sets.

IEBDV3—for opening and validating the SYSUT3 data set.

IEBDV4—for opening and validating the SYSUT4 data set.

INPUT: Job file control block (JFCB), data set control block (DSCB), data extent block (DEB).

OUTPUT: Switches to indicate data set characteristics (set in the communication area, IEBMCA). Opened data sets.

RECEIVES CONTROL FROM: Modules IEBDSCPY (including IEBMBT), IEBVDM, and IEBVTT.

PASSES CONTROL TO: Modules IEBVMS and IEBVTM, and subroutine TTOCC of module IEBDSCPY.

LOGIC: Depending on the data set to be processed, module IEBDV1 receives control at one of the entry points previously listed. Before opening the input, output, SYSUT3, and SYSUT4 data sets, the module uses a DEVTYPE macro instruction to obtain the device characteristics, which are saved in the communication area.

If any one of these data sets does not reside on a direct access storage device, control is given to module IEBLDUL for unload processing. If any other unload or load condition is found, control is also passed to module IEBLDUL. Otherwise, the order of processing varies depending on the data set that is being processed.

SYSUT3 OR SYSUT4 DATA SETS: For these data sets, the module reads the JFCB into storage, using the RDJFCB macro instruction. The logical record length (80 bytes for SYSUT3 or 264 bytes for SYSUT4) is placed into the JFCB, and the module then issues the OPEN (type J) macro instruction for the data set being processed. The module then initializes the data set's DCB to indicate the beginning of the data set, and returns control to the caller. If a validation error occurs, the module gives control to module IEBVTM to terminate the copy operation.

INPUT DATA SET: For this data set, the module issues the RDJFCB macro instruction and then the OPEN (type J) macro instruction. The module then issues the OBTAIN macro instruction to read the

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

DSCB into storage. From the information in storage as a result of the execution of these macro instructions, module IEBDV1 verifies the data set characteristics. If a validation error occurs, the module gives control to module IEBVTM to terminate the copy operation. Otherwise, the module compares the input data set's characteristics with those of the output data set. If the characteristics are compatible, and if data set compression is indicated, the module checks the requirements for compression (no track overflow, no SELECT or EXCLUDE statement, no reblocking or deblocking, and no keyed records).

If the data set's characteristics are not compatible, the module has a message printed and then gives control to the termination module. If the compress-in-place requirements are not satisfied, the termination module receives control to end the processing for the current copy operation. Then the scan module will flush control statements through to search for the next COPY, COPYMOD, or ALTERMOD statement. Otherwise, the calling program receives control to continue processing the current data set.

If the low-order bit in the DSORG field of the Format 1 DSCB is on to signify that a data set is unmovable (that is, contains location-dependent information), data set compression cannot be done.

OUTPUT DATA SET: For this data set, the module issues the RDJFCB and OBTAIN macro instructions. Equivalent parameters in the output DSCB and JFCB are then compared. If equivalent parameters are equal to zero, the value of the parameter in the JFCB is set equal to the value in the current input DCB (which has already been opened and verified). If the parameters are not both zero, they are left unchanged.

If COPYMOD is specified, the value in the MAXBLKSZ field is changed, if necessary, such that it does not exceed the device maximum block size. This value is then used as the JFCB block size if the JFCB block size is not greater than the value of MAXBLKSZ.

The module then issues the OPEN (type J) macro instruction, followed by the OBTAIN macro instruction (used to read the updated output DSCB into storage). From the information in the DSCB and the JFCB, the module verifies data set characteristics. If a validation error occurs, the module gives control to module IEBVTM to terminate the copy operation.

SYSIN AND SYSPRINT DATA SETS: Before module IEBDV1 processes the SYSIN and SYSPRINT data sets, it specifies the SYNAD error-recovery routine and establishes DCB exit list information. Then, the module opens the SYSPRINT data set (DCB) and performs checks to validate blocking information. (The block size should be an integral multiple of the logical record length.) If the SYSPRINT data set is not successfully opened, the error is ignored, but the module sets a switch so the message module won't write messages onto the SYSPRINT data set. Then, the module uses the OPEN macro instruction to open the SYSIN data set, checks the blocking characteristics of the data set, and returns control either to the caller (if no errors occur on opening the data set) or the terminate routine (if the data set cannot be opened) to end the job step.

MODULE IEBDWR—WRITE ROUTINE

Module IEBDWR writes information for the IEBCOPY program. It writes: member data onto the output data set; directory entries onto the SYSUT3 spill data set; and output directory blocks onto either the SYSUT4 spill data set or the output data set. It may also erase information from a track.

In addition, the module always reads back ('read-back check') a buffer-full of directory blocks for verification after they are written on either the output data set or the SYSUT4 data set

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

(that is, it performs a 'dummy' read operation in that no data is transferred). A read-back check is performed on member data if the user specifies validity checking for the output data set either on a DD statement or via information in the DSCB. During read-back check processing, the skip bit in the 'Read' channel program is on to prevent data transfer. For writing onto the SYSUT3 data set, read-back checking is not done.

Module IEBDWR uses a 'Format Write' type of channel program when writing member data onto the output data set or when writing directory blocks onto the SYSUT4 data set. The module uses a nonformat type of channel program for writing merged output directory blocks onto the output data set, since the blocks are already formatted. Similarly, when the module writes directory entries (with updated member TTRs) back to the SYSUT3 data set during the copy or load phase of the program, it uses the nonformat type of a 'Write' channel program.

Note: When initially creating the SYSUT3 data set for a given input data set, module IEBDWR uses a 'Format Write' channel program. See the appropriate Component Description publication for a description of Format channel commands.

ENTRY POINT: IEBDWR

INPUT: Indication of which data set is being written. Indication of whether a read-back check option is desired. Switches and pointers indicating last-record information, record to be written next, and overflow record information. Indication of first and last tracks to be erased.

OUTPUT: Location of last record written. Erased tracks.

RECEIVES CONTROL FROM: Modules IEBDSCPY, IEBWSU

PASSES CONTROL TO: Modules IEBIOE, IEBVMS

LOGIC FOR WRITING: When module IEBDWR receives control to write data, it determines what information is to be written and where it is to be written.

If there is sufficient virtual storage to allocate a 4-track size I/O buffer and CCWAREA, a non-PCI channel program is constructed. If there is not sufficient storage, a PCI channel program is constructed. To write the data, the module uses the EXCP macro instruction. The initial seek address in the channel program is placed in the IOB, but after the PCI program has started, the next address to which the data is to be written is obtained by the PCI appendage routine, IGG019C8. (For writing one record at a time, as onto the SYSUT3 data set, the PCI appendage routine is not used.) For the non-PCI channel programs, all addresses are established when the channel program is constructed.

If an uncorrectable I/O error occurs, module IEBDWR gives control to module IEBIOE to analyze the error and set a return code to indicate that an error message will be given by the message module. The channel program used for writing member data onto the output data set or for writing merged directory blocks onto the SYSUT4 data set is shown in Figure 53 on page 100. Operation flags (bits) and notes are given as necessary to explain the action produced by each channel command word (CCW). CCW W2 is used for sector orientation on RPS devices only. For all other devices, W3 is the first CCW in the chain. In addition, for RPS devices, the NOP CCWs at W8 and W11 are replaced with a Read Sector CCW. Using the command words (3 to 14) in Figure 53, the channel program proceeds in the following manner (Wn represents channel command word n; for example, W5 is CCW5 in the figure).

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

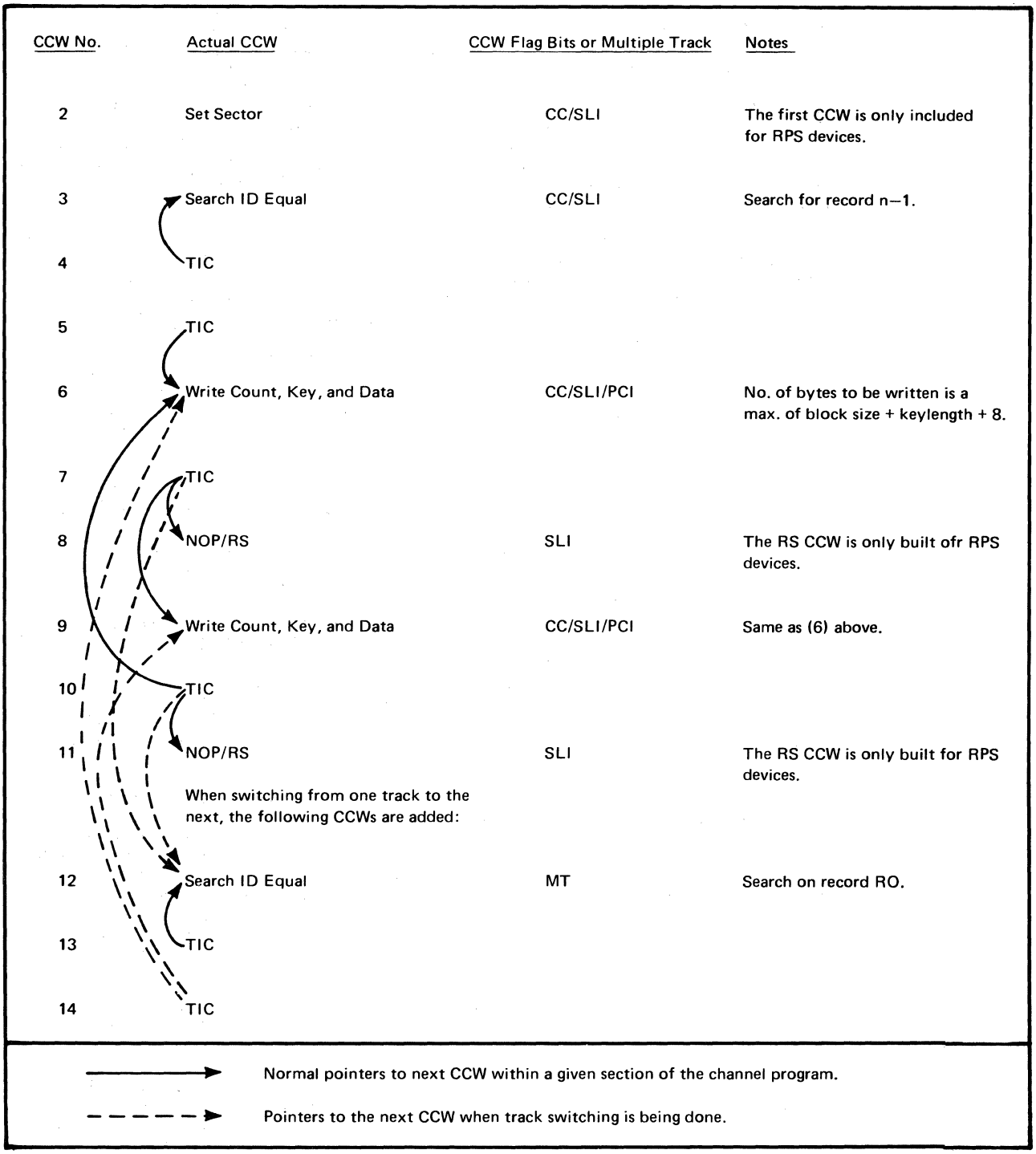


Figure 53. Channel Program for Writing Member Data or Merged Directory Blocks

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

W3 and W4 are executed sequentially (in a loop if necessary) until the (n-1)st record is located (where n is the number of the record that is to be written). Then W5 causes transfer of control to W6. As W6 is being executed, the PCI flag bit causes the I/O supervisor to give control to the PCI appendage module for the subroutine at label IGG019. A switch is set to indicate to appendage IGG019FT that an extended translation list must be given to the I/O supervisor. The PCI appendage module determines where the next record is to be written. If the count field of the next record indicates that the record is the first data record on a track, W7 is changed to point to W12 so that the multitrack search command can find the next track. Once the next track is located, W4 transfers control back to W9 so the channel program can continue.

If the next record is not the first one on a track, the PCI appendage module changes the TIC in W7 to point to W9. Also, the TIC in W10 is changed to point to W11 to stop the program if an error occurs. As the channel program proceeds, W7 and W10 have their addresses updated to continue the channel program. Each time a track has to be changed, the sequence W12-W14 receives control to find record R0 on the next track; in this case, W14 will point back either to W6 or W9, depending on which command would have been the next one if the track switching had not been required. The dashed lines in the command word sequence illustrate the commands that receive control from other commands in the case of track switching.

When writing the merged directory blocks onto the output data set, the module uses a PCI channel program that is identical in form to the one just described except for: substituting Write Key and Data CCWs to replace the Write Count, Key, and Data CCWs; changing the search argument of W3 to record n; and changing the search argument of W12 to record 1.

When writing directory entries onto the SYSUT3 data set, module IEBDWR uses a PCI channel program consisting of CCWs W3, W4, W5, and W6. If entries are being written for the first time: W3 searches on record n-1; W6 has the form shown; and the PCI flag bit is not on. If SYSUT3 entries are being updated: W3 searches on record n; W6 is changed to a Write Key and Data CCW; and the PCI flag bit is not on. In both cases, records are written one at a time.

Figure 54 on page 102 depicts the non-PCI channel program used in writing member data records. In the figure, the label in the address fields, 'MAn', refers to the nth record in the buffer and MALT refers to the last record on a track within the buffer. 'DLn' refers to the data length of the nth record. The sequences labeled Track 1 write, read back (only present if write verify was selected on the OUTDD card), and track switch are repeated for as many tracks as are contained in the buffer or storage available for CCWs. In the latter case, when storage for CCWs is exhausted before all the records in the buffer are processed, multiple EXCPs are used to output the contents of the buffer. This is accomplished by looping back through the channel program construction phase after successfully completing the I/O.

If the output data set is in track overflow format, the CCW preceding the read back portion of the channel program (that is, the CCW that writes the last record on the track) would be a Write Special (X'01'), unless the record exactly fits on the track or it is the last track in a cylinder.

The channel programs used to write the directories are shown in Figure 55 on page 103. They are basically the same as the member data channel programs, except that the count fields are not being written, necessitating the search on the CCHHR preceding the Write Key and Data. To simplify the CCW construction code, the search ID = multiple track mode and associated TIC for the first record to be written are constructed even though they aren't used. The prefix channel

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

program's TIC points directly to the Write Key and Data, labeled with an asterisk in Figure 55.

The terminating CCW is either a NOP or a Read Sector. The latter is used if the device is a rotational position sensing (RPS) device.

If an RPS device is used, IEBDWR updates the sector value associated with the data set that is being written before returning control to the caller. The value used for updating this sector value is the last sector address read.

```
TRACK 1          CCW  X'1D', MA1,X'60', DL1+8 WRT COUNT, KEY & DATA (CKD)
WRITE            CCW  X'1D', MA1+ DL1+12,X'60', DL2+8
                        .
                        .
                        .
                 CCW  X'1D', MALT,X'60', MALTDL+8
                 CCW  X'1A',*,X'70',16 Rd Home Address
                 CCW  X'31',*+24,X'60',5 Search ID =
READ             CCW  X'08',*-8,X'00',1 TIC
BACK             CCW  X'08',*+16,X'00',1 TIC
                 DC   CL8 'CCHHR' of Record Preceding MA1
                 CCW  X'1E', MA1, X'70', DL1+8 Rd CKD No Transfer
                 CCW  X'1E', MA1+DL1+12, X'70', DL2+8
                        .
                        .
                 CCW  X'1E', MALT, X'70', MALTDL+8
                 CCW  X'BJ', *+24, X'60',5 Search ID = Multitrack
TRACK            CCW  X'08', *-8, X'00', 1
SWITCH           CCW  X'08', *+16,X'00', 1
                 DC   CL8 'CCHHO' Record Zero of Next Track
TRACK 2          CCW  X'1D', MALT+MALTDL+12,X'60', DL+8
WRITE                   .
                        .
                        .
TERMINATING      CCW  X'22',SECTOR, X'00', 1 RD SECTOR
CCW                     or
                 CCW  X'03', * - *, X'00', 1 NOP
```

Figure 54. Member Data Write Channel Program

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

```
                CCW   X'B1', MA1+4,X'60',5 Search ID = , Multitrack
                CCW   X'08', * - 8, X'00', 1 TIC
TRACK 1        *CCW   X'0D', MA1+12,X'60', 264 Write Key and Data
WRITE           CCW   X'B1', MA1+4+264+12,X'60',5
                CCW   X'08', * - 8,X'00', 1
                CCW   X'0D', MA1+12+12+264,X'60', 264
                        .
                        .
                        .
                        .
                      Read back the same as in Figure 54 except
                      Read Key and Data (X'0E').
                        .
                        .
                      Track switch the same as Figure 54.
                        .
TRACK 2               Same sequence as Track 1 but for second track
WRITE                 in buffer.
                        .
                        .
                        .
                      Termination as in Figure 54.
```

Figure 55. Directory Write Channel Program

LOGIC FOR ERASING: When module IEBWSU is setting up an output record to be written with track overflow, it may find that all the record segments cannot be contained within the same cylinder or extent. (It is invalid to cross cylinder or extent boundaries with segments of a record.) Therefore, module IEBDWR receives control to erase all the tracks, within the cylinder or extent, that will not be used. The input to the module for this process consists of the TTRs of both the first and last tracks to be erased. The module uses the channel program (shown in Figure 56) once for each track to be erased.

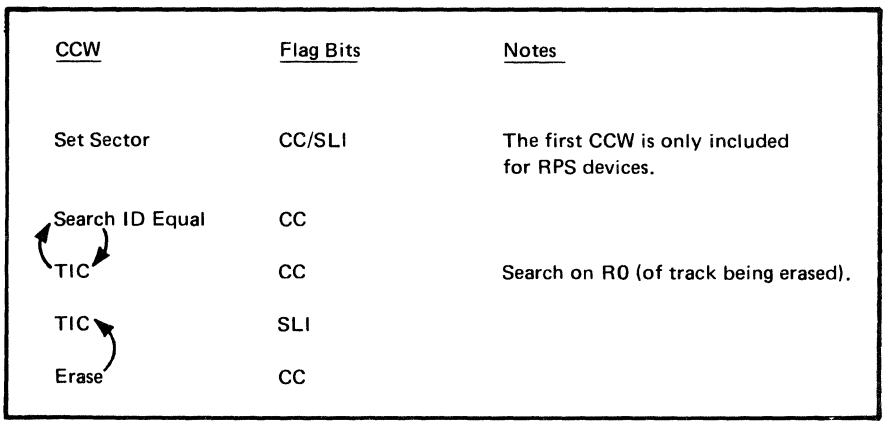


Figure 56. Channel Program for Erasing a Track

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

MODULE IEBIOE—I/O ERROR ROUTINE

Module IEBIOE provides the error analysis for I/O errors that occur during the reading or writing of the following data sets for the IEBCOPY program:

Input
Output
SYSUT3
SYSUT4

Except when entry point IEBIOE2 is used, module IEBIOE uses the SYNADAF macro instruction to: obtain both a message buffer area (which is outside the storage assigned to the IEBCOPY program) and a save area for control transfer, and to insert in the buffer a message describing the error. This message will then be moved to the communication area (IEBMCA) and written out by module IEBVMS. To release the message buffer storage area and the save area, the module uses a SYNADRIS macro instruction.

ENTRY POINTS:

IEBIOE—for the direct access SYNAD routine.

IEBIOE2—for an end-of-file during a load operation, or for a block size error during an unload operation.

IEBIOE3—for a load SYNAD routine.

IEBIOE4—for an unload SYNAD routine.

INPUT: At entry point IEBIOE register 5 giving the address of the IOB.

At entry points IEBIOE3 and IEBIOE4 parameters in registers 0 and 1 to be used by the SYNADAF macro.

OUTPUT: A message for the message-printer module. Switches indicating the error type found.

RECEIVES CONTROL FROM: Modules IEBDRD, IEBDWR, IEBWSAM, IEBLDUL, and IEBDSCPY.

PASSES CONTROL TO: Modules IEBVMS and IEBVTM, and subroutine CCOTT (in module IEBDSCPY), and, if a recoverable error occurs, to the caller.

LOGIC: Module IEBIOE first sets up a message buffer, determines the category of error detected, and gives control to the message module, IEBVMS, to write the appropriate message. Then module IEBIOE branches to an appropriate internal subroutine to further define the error, set indicator switches, and issue an error-descriptor message. In the case of an unload, the unloaded data set is flagged as not loadable.

If the module has recognized an error in the writing of the output directory blocks onto the output data set partitioned, it may use the EXCP macro instruction to read (into a main storage buffer) the last block that had been previously written correctly. It will then update the last entry in that block by setting the block's key and the last entry's name to hexadecimal 'FF's, and reissue the EXCP macro instruction, but this time with a channel program to write the updated block back onto the output data set. (If the error was in writing the first directory block of the data set, the updating is not done, and control is given to the termination module, IEBVTM.)

MODULE IEBLDUL—UNLOAD/LOAD INITIALIZATION ROUTINE

The primary purpose of this module is to initialize load and unload processing.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

If called to initialize a load operation, the load DCB is opened, the COPYR1 and COPYR2 header records are read, load processing requirements are checked for validity, and various switches and I/O control block values are set to look as if a partitioned data set to partitioned data set copy operation were being performed.

If called to initialize an unload operation, the unload DCB is opened, the COPYR1 and COPYR2 header records are built, the unload processing requirements are checked for validity, and various switches and I/O control block values are set to look as if a partitioned data set to partitioned data set copy operation were being performed.

ENTRY POINT: IEBLDUL

INPUT:

Pointers to the following areas:

- DSCB buffer area
- JFCB buffer area
- IEBCOPY's communication area
- Buffer area for input VOLID
- Buffer area for input DSNAME
- Data set names of the input and output data sets
- VOLIDs of the input and output data sets
- Load DCB
- Unload DCB

Switches indicating whether:

- A load or an unload operation is to be initialized.
- PARM=COMPRESS has been specified.
- Main storage has been released (FREEMAIN) for data management BSAM modules.

OUTPUT: Switches set to show whether a load and/or an unload operation has to be performed.

Opened load and/or unload DCB.

COPYR1 and COPYR2 header records read and/or built.

Register 15 containing a 0 for a successfully opened and validated data set and containing a 4 for an unsuccessfully opened and/or validated data set.

Several fields within the master communication area (IEBMCA) are set to look as if a partitioned data set to partitioned data set copy operation were being performed.

Logical record length and block size fields within the DSCB and JFCB buffer areas.

RECEIVES CONTROL FROM: IEBDV1

PASSES CONTROL TO:

IEBDSFRC—to release main storage

IEBVMS—to give appropriate error messages

LOGIC: Upon entry, switches are checked to determine whether an unload or a load initialization is to be performed.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

If a load is to be initialized, the LOAD switch is set, and the device type of the data set to be loaded is checked. If the data has been allocated to a device other than a tape or a direct access device, an error message will be issued by module IEBVMS and a return code of 4 in register 15 gives return to the caller.[3] If a correct device type has been allocated, a check is done to see if main storage has been released for data management modules during a previous unload or load operation. If no main storage has been released, IEBDSCPY's subroutine IEBDSFRC is called to issue a FREEMAIN.

The following fields related to the load DCB are then initialized:

- DCB ddname field
- DCB exit list
- DCB end of data routine address
- DCB block size and logical record length fields

The load DCB is now opened, and exit routine 'LOADVAL' is given control to ensure the DCB BLKSIZE and LRECL fields are valid and large enough to read the COPYR1 and COPYR2 header records. Subroutine LOADVAL then returns control to the OPEN routine.

Upon return from the OPEN routine, the DCB is checked to verify if the data set was successfully opened. The COPYR1 header record is read and validated, and the TBLKSIZE field of the COPYR1 record containing the block size of the unloaded data set is compared with the block size field of the load DCB. If they are found not to be equal, the load DCB is closed and the correct values are filled in the block size and logical record length fields.

At this point, the COPYR2 header record is read. Several fields and pointers are initialized from the information contained in the COPYR1 and COPYR2 header records to complete the interface of the TTR conversion.

A return code of 0 is then set in register 15, and control is returned to the caller.

If an unload is to be initialized, the unload switch is set. An error message is issued by module IEBVMS if the user specified the same data set for the input and output data set, and, with a return code of 4 in register 15, control is returned to the caller.[4] If the input and output data sets are not identical, tests are made to determine if the input and output data sets are on the same tape volume, or if an unload has been specified to a device other than a tape or direct access device. If no main storage has been released, IEBDSCPY's subroutine IEBDSFRC is called to issue a FREEMAIN.

The following fields related to the unload DCB are then initialized:

- DCB ddname field
- DCB exit list
- DCB synchronous error routine address
- DCB block size and logical record length fields

---

[3] If, in the course of processing, an error is encountered, control is given to the error message routine IEBVMS, and a return code of 4 is placed in register 15.

[4] If, in the course of processing, an error is encountered, control is given to the error message routine, IEBVMS, and a return code of 4 is placed in register 15.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

The unload DCB is now opened, and exit routine LOADVAL is given control to ensure the DCB BLKSIZE and LRECL fields are valid and large enough for unload processing. Subroutine LOADVAL then returns control to the OPEN routine.

Upon return from the OPEN routine, the DCB is checked to verify if the data set was successfully opened.

Several fields within the master communication area (IEBMCA) and the TBLKSIZE field of the COPYR1 record are now initialized. If the LOAD switch is on, control is returned to the caller, because the COPYR1 and COPYR2 header records are already in their respective buffers. If the LOAD switch is not on, the COPYR1 and COPYR2 records are built, using the output partitioned data set characteristic field in IEBMCA for the COPYR1 record and the input partitioned data set DEB for the COPYR2 record.

A return code of 0 is then set in register 15, and control is returned to the caller.

SUBROUTINE IEBMBT OF MODULE IEBDSCPY

Subroutine IEBMBT builds the entries for the CTLTAB table, and saves the directory entries. The entries are retained in main storage if space is available or are written (spilled) onto the SYSUT3 data set if space is not available. The main storage space requirements for CTLTAB table entries (for exclusive or full copy, unload, or load processing) take priority over the space requirements for input directory entries. Thus, directory entries that have already been retained in main storage may have to be spilled to permit the construction of CTLTAB table entries.

When the subroutine constructs CTLTAB table entries for a selective copy, unload, or load, both the SETAB table and the NNPTRTAB table (if the latter exists) have already been formed. Thus, contention between table entries and directory entries for space does not exist because, for a selective copy, unload, or load, the CTLTAB table overlays the SETAB table. In this case, directory entries are spilled only if main storage is not available.

Working with only one input directory entry at a time (that is, the subroutine is requested each time a CTLTAB table entry is to be constructed), subroutine IEBMBT moves the appropriate directory entry from the I/O buffer area to main storage (or causes it to be written onto the SYSUT3 data set). Depending on which action was taken, the subroutine then places into the CTLTAB table entry for the member either the member's main storage address or its relative track address (TTR) on the SYSUT3 data set. The member's input TTR within the directory entry is also extracted by the subroutine and placed into the same CTLTAB table entry. The subroutine increases counters indicating both the number of input directory entries that are saved in main storage and the total number of CTLTAB table entries that have been constructed for found members of the current input data set of a copy, unload, or load step.

Subroutine IEBMBT also places the main storage address of the most recently saved directory entry (if any have been saved in main storage) in location ICPT of the communication area. For either a full or an exclusive copy, unload, or load, this address is changed if an 'already-retained' entry must be subsequently spilled to make storage available for one or more CTLTAB table entries. For either an exclusive or a full copy, unload, or load, the subroutine stores (in location BUMP of the communication area) the address of the next available unused main storage location after the most recently built CTLTAB table entry.

When the subroutine has to spill a directory entry to the SYSUT3 data set for the first time, it gives control to module IEBDV1

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

(at entry point IEBDV3) to open and verify parameters for the data set. The actual writing of an entry on the SYSUT3 data set is done by the 'write' module to which the subroutine gives control via the 'write setup' module. After the entry has been written, subroutine IEBMBT uses subroutine CCOTT (of module IEBDSCPY) to convert the SYSUT3 device address (MBBCCHHR) to a TTR value. This TTR value is then placed into the corresponding CTLTAB table entry.

If, in building CTLTAB table entries for either an exclusive or a full copy, unload, or load, the space for a table entry is not available and all input directory entries have already been spilled to the SYSUT3 data set, it may be possible to acquire more storage space by reducing the space allocation for the I/O buffers. ("Storage Allocation" describes this procedure.)

After all processing associated with the construction of a CTLTAB table entry (or a SETAB/CTLTAB table entry if the copy, unload, or load is a selective one) has been completed, subroutine IEBMBT returns control to the calling routine.

MODULE IEBMCM—COMPRESS-IN-PLACE ROUTINE

The main function of module IEBMCM is to locate the first gap that exists between members in an input data set during a compress-in-place operation. When a gap is located, the module prepares the input and output DCBs so that a full copy can be performed on the data set members following the gap, thus closing up the data set to provide more available space following the last member in the data set.

ENTRY POINT: IEBMCM

INPUT: Communication area containing switches and pointers.

OUTPUT: Indication that data set is already 'compressed,' or preparation for module IEBDSCPY to perform a full copy.

RECEIVES CONTROL FROM: Module IEBDSCPY

PASSES CONTROL TO: Module IEBVMS to have a message placed on SYSPRINT.

Module IEBDRD to read data for the purpose of obtaining the actual address of end-of-file records.

Subroutine TTOCC (of module IEBDSCPY) to convert TTR address to actual address.

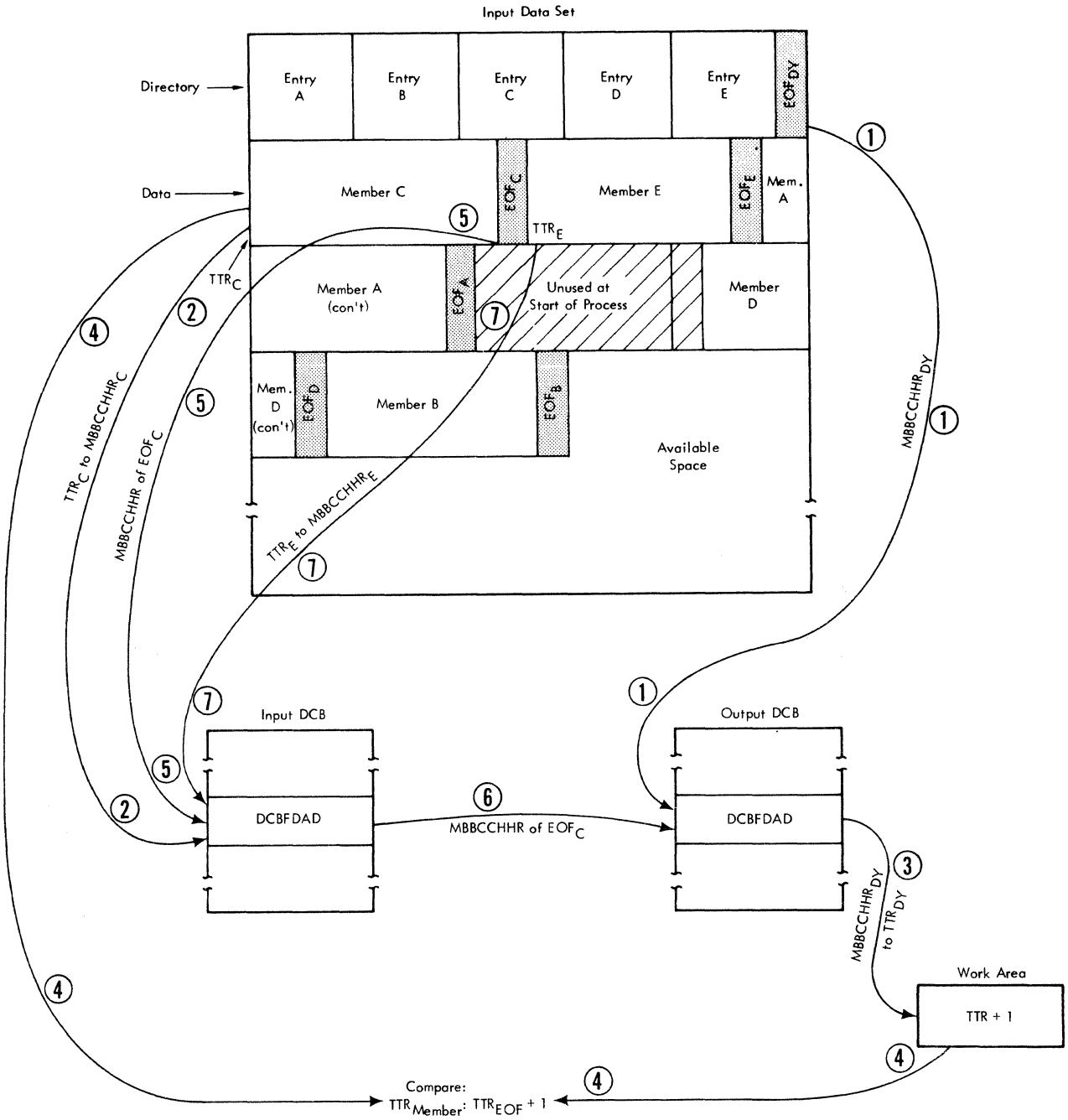
Subroutine CCOTT (of module IEBDSCPY) to convert actual address (MBBCCHHR) to TTR relative address.

Subroutine IEBMGI (of module IEBDSCPY) to get TTR-ordered directory entries.

LOGIC: Module IEBMCM is called to perform data set compression under either of the following conditions:

1. When the data set name and volume serial number are the same for both the input and the output data sets.
2. When the EXEC control statement contains PARM = COMPRESS for job control statements of an earlier version of the IEBCOPY program, and condition 1 is satisfied. Figure 57 on page 109 illustrates the procedures described in the following text. Numbers in parentheses in the text are keyed to the numbers (or steps) shown in the figure.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1



Note: Data movement or processing is in order of encircled numbers (1) to (7). See text for details.

Figure 57. Data Flow in Module IEBMCM

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Module IEBDRD performs a dummy reading (that is, no data is transferred) of the 'input' data set's directory to locate the end-of-file record (EOF-DY in Figure 57) that follows the directory. The MBBCCHHR address of the EOF record is placed in the DCBFDAD field of the DCB for the 'output' data (1). (For compress-in-place, the input and output data sets are actually the same data set.)

Subroutine IEBMGI (of module IEBDSCPY) then obtains the first (and, subsequently, the next) input data set directory entry (IDE) based on the member TTR-sequenced section of the CTLTAB table. In Figure 61 on page 125, member C has the lowest TTR. From the IDE for member C, module IEBMCM gets the member TTR and then gives control to the TTOCC subroutine (of module IEBDSCPY) to convert the TTR to an actual (MBBCCHHR) address and place the converted value in the DCBFDAD field of the DCB for the input data set (2).

Subroutine CCOTT (of module IEBDSCPY) then converts the actual address in the output DCBFDAD field to a TTR value. This value is placed in a work area and increased by 1 (3). Increasing by 1 effectively makes the TTR value equal to the TTR value at which the first (or 'next') data set member should be, if there is no gap in the data set.

Step (4) of the figure represents the comparison between the input TTR of the next member and the TTR of the position in the data set where the member should be. If the IDE's TTR is larger than the value [(TTR of EOF DY) + 1] to which it is compared, the data set has a gap and compression is to be done. Module IEBMCM returns control to module IEBDSCPY to proceed with a full copy of the input data set starting with first member not already in place.

If the IDE's TTR is equal to the value [(TTR of EOF DY) + 1] (it can never be less), the member is already 'in-place' and the next member is considered. Module IEBDRD again receives control to do a dummy read to locate the last record of the current member (in this case, member C), and to place the MBBCCHHR value (EOF C) of the last record in the input DCBFDAD field (5). Module IEBMCM then makes the output DCBFDAD field equal to the input DCBFDAD field (6).

If there are more input directory entries to consider, step (2) is performed, this time using the member for input directory entry E (7). The convert routine then receives control to change the MBBCCHHR of EOF C (which is in the output DCBFDAD field) to a TTR value and place it in the work area. This TTR is then increased by 1 as in step (3). Now, the comparison at step (4) involves the TTR of member E and the [TTR + 1] of the EOF C. Depending on the result of the comparison, the processing proceeds as previously indicated (that is, either to do a full copy beginning with member E or to indicate that member E is in place and to consider the next member).

The preceding cycle of processing is repeated until either the first gap is located or all members have been found to be in place. If the latter condition exists, module IEBMCM causes module IEBVMS to write a message indicating this, and it then returns control to module IEBDSCPY with an indication that a directory merge is not needed (because no members were moved) and that the termination routine can be given control.

SUBROUTINE IEBMGD OF MODULE IEBDSCPY

Subroutine IEBMGD obtains the address of a directory entry in the I/O buffer. This address is used in the building of a CTLTAB table entry, in the scanning of output directory entries for duplicate members, and (when required) in the merging of an output directory with input directory entries. The subroutine does not obtain a pointer to a directory entry if it determines that any of the following conditions exist:

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- An end of file record (data length = 0) encountered.
- The last entry (indicated by a hexadecimal 'FF' name) in the directory is now encountered.
- The last directory entry in the last directory block in the I/O buffer already was encountered (determined by examining the name in the block's key field) during either the output directory prescan or the directory merging process.

In any of these cases, the subroutine returns control to the calling routine with an indication that a directory entry address has not been obtained.

During the input directory scan phase of the program, the subroutine issues a request for the read module, IEBDRD, to refill the I/O buffer with more directory blocks as necessary.

To indicate that there are no more directory entries to be processed by the caller, subroutine IEBMGD returns control to the address pointed to by the caller's register 14. If a pointer (to an entry) is returned to the caller, the subroutine returns control to 4 bytes past the address pointed to by the caller's register 14. In this latter case, the communication area field, DEPTR, contains the address of the directory entry requested by the caller.

SUBROUTINE IEBMGI OF MODULE IEBDSCPY

Subroutine IEBMGI obtains the address of the next input directory entry that is to be considered in: prescanning the output directory (to check for duplicate-named input members); copying input members; or merging directory entries. It also returns to the caller, when appropriate, an indication that the last directory entry address has been obtained. In addition, if "new names" were specified during a selective copy, unload, or load operation, a pointer to the new name itself is returned during the output directory prescan.

Depending on the reason for obtaining the directory entry address, either the alphamerically sequenced list of pointers to input directory entries, or the input member-TTR sequence list of pointers to input directory entries in the CTLTAB table, will be entered to locate the entry. For the member copy phase, the former list is entered. For the output prescan phase and the directory merge phase of the program, the latter list is entered.

If the input directory entry is on the SYSUT3 data set, the TTOCC subroutine of module IEBDSCPY is used to convert the TTR of the location of the directory entry (on SYSUT3) to an MBBCCHHR address format. Then the entry is read (via module IEBDRD) into a special SYSUT3 I/O buffer for processing.

During a directory merge, this subroutine inserts a new name into the directory entry if a new name is specified. The subroutine does this after the appropriate directory entry has been read into storage from the SYSUT3 data set (if necessary) and before the entry is merged with the output data set directory.

SUBROUTINE IEBMKO OF MODULE IEBDSCPY

Subroutine IEBMKO retains output directory blocks in main storage if space is available. The subroutine finds the main storage address for saving the first block by using the ALO field of the communication area. Beginning at the lowest available main storage location (as given in the ALO field), the directory blocks are saved in the direction of the highest available main storage address (specified in the HICOR field of the communication area). In addition to saving the directory

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

blocks, the subroutine also saves the following information for each block:

- A 1-byte flag field
- A 3-byte field containing the MBB value of the block's actual device address
- An 8-byte count field (CCHHR + key length data length)
- An 8-byte key field containing the name of the last entry in the block

The OBCT field of the communication area contains a running count of the total number of output directory blocks saved in main storage for the current copy step. Subroutine IEBMK0 updates this field each time a block is saved. This information is used during the merge that produces the final output directory.

Other information that the subroutine saves includes the address of the last output directory block saved in main storage, and an indication of whether none, some, or all of the output directory blocks were saved in main storage. When an entire I/O buffer-full of blocks is saved, the subroutine returns control to the caller, so that the blocks may be processed.

MODULE IEBRSAM—READ ROUTINE FOR UNLOADED DATA SETS

Module IEBRSAM reads the directory of the unloaded data set into the buffer. When the buffer is full or when the directory has been totally read, control is returned to the caller. This module also reads member data into the buffer until it is full or until the end of a member is reached.

Note: If the member to be processed possesses a note list, it is placed in the note list work area.

ENTRY POINTS:

IEBRSAM—to read the directory of the unloaded data set into the buffer.

IEBRSAM1—to read member data into the buffer.

INPUT:

- Switches indicating whether the buffers are contiguous or not and whether both buffers are available for data records.
- An address indicating the beginning and end of the available buffer space.
- An MBBCCHHR count field to indicate the member to be read.
- A pointer to the note list work area.

OUTPUT:

- Buffer full with unloaded data set directory blocks or with member data.
- If present, the note list placed in the note list work area.

RECEIVES CONTROL FROM: Modules IEBDSCPY, IEBCPMOD

PASSES CONTROL TO: Modules IEBDSCPY and IEBIOE

LOGIC: As the unloaded data set has a variable spanned record format that is not recognized by other IEBCOPY modules, IEBRSAM at the entry point IEBRSAM saves an 8-byte area preceding the location where the actual data is expected to be. After a block of data is read, the area saved is restored. Because this is

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

done for each data block, the buffer contains data without trace of the record descriptor words that preceded the data in the unloaded data set. All other IEBCOPY modules can now handle the data contained in the buffer.

The IEBRSAM module reads the directory blocks into the buffer until the buffer is full or until the last directory block has been read. In either case, control is then returned to the caller.

At entry point IEBRSAM1, member data is read into the I/O buffer using the method described for the IEBRSAM entry. When a block is read, a test is made to see whether at least one whole partitioned data set record has been read or if only a segment has been read, in which case the other segments are read to constitute the whole partitioned data set record before any of the data is tested. The MBBCCHHR (full disk address) of the DCBFDAD field in the DCB is compared to the MBBCCHHR in the count field of the member data to find the member data. A test is then made for the presence of a note list, which, when present, is moved to the note list work area. The member is then read into the buffer until the buffer is full or until the end of the member is reached. Control is then returned to the caller.

MODULE IEBSCN—SCAN ROUTINE

The primary purposes of this module are:

- To scan the control statements and give messages for syntax and/or statement sequencing errors
- To check for compatibility of control statements from an earlier version of the IEBCOPY program
- To build a table (INDD table) of ddnames of input data sets that contain members to be copied, unloaded, or loaded
- To build a select/exclude table (SETAB) of members to be selected or excluded from a data set being copied, unloaded, or loaded

ENTRY POINTS:

IEBSCN—for initial and normal scanning.

IEBSCF—for an end-of-file for the SYSIN data set.

INPUT: The SYSIN data set consisting of control statements for the program.

OUTPUT: Statement error messages, and the contents of the SYSIN data set, are placed in the SYSPRINT data set. The SETAB and the INDD tables. Switches set in the communication area to indicate the type of operation being performed (copy, exclude, select, etc.)

RECEIVES CONTROL FROM: Module IEBDSCPY

PASSES CONTROL TO: Calling program, with message to user if error occurs.

MODULES:

IEBVMS to write a message and print a statement.

IEBVTM to terminate, or check for termination of, a job step.

SUBROUTINES:

RDCARD—Statement analysis routine.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

INDDLOOP—General input ddname or member name scan routine. Other subroutines, each defined (in the text) by the label of its first instruction.

LOGIC: Upon entry to this module, a test is made to determine whether or not the last control statement read was an end-of-file (EOF), and whether or not the current statement is another COPY, COPYMOD, or ALTERMOD statement delimiting the next copy operation (see Utilities for a definition of a copy operation). A positive response in either case indicates that an exit is to be made to the message-writing routine. Otherwise, the RDCARD subroutine is entered to read the first control statement from the SYSIN data set into main storage.

The RDCARD subroutine also receives control after each nonjob-terminating error that causes an exit to the message-writing module.

Each control statement is verified to be sure it is valid for either the current version or an earlier version of the IEBCOPY program. If so, the statement's command, keywords, and parameters are scanned and analyzed. The module contains both a command word table and a keyword table to be used for this purpose.[5] Depending on the results of the scan, the appropriate subroutine receives control to process the associated information.

For each statement containing an error, the scan module gives control to the message-writing module to print an appropriate message. If the job termination is not indicated, the scan module then flushes through the remaining control statements in the current copy, alter, unload, or load operation until it encounters either an EOF or the next COPY, COPYMOD, or ALTERMOD statement.

In scanning a COPY, COPYMOD, or ALTERMOD statement, the module's subroutine (beginning at the label CHKCOPY) determines whether an acceptable combination of command words and keywords has been specified. For example, if a SELECT command word has been specified but no MEMBER keywords have been specified, the copy, unload, or load operation is terminated. Also, if an EXCLUDE command word has been specified but no MEMBER keywords have been specified, the copy operation will be treated as a full copy, unload, or load request. If two consecutive (current version) COPY statements occur without an intervening INDD statement, a statement sequencing error has occurred; the subroutine will ignore the first COPY statement and begin processing with the second COPY statement.

As the module scans the control statements, it constructs a table of input ddnames appearing with INDD keywords. This is the INDD table. If either the select or the exclude option has been requested, the select/exclude table (SETAB) is built. This table contains the names of members that are to be either selected or excluded from the input data set.

In scanning the OUTDD keyword, a subroutine beginning at the label OUTDDANL checks whether the keyword appears on a COPY, COPYMOD, or ALTERMOD statement, whether it has the proper length, and whether the parameter is continued on a following statement. The parameter represents the name of the output data set for the current copy, unload, or load operation. If it is valid, the name is placed, left-adjusted, into the output ddname area and is padded with low-order blanks if it is less than 8 bytes in length.

The CHK4COMT subroutine of module IEBSCN controls the reading and printing of any comment statements that follow the last keyword or parameter on a statement. Comments that are on the

---

[5] Acceptable abbreviations for IEBCOPY command words and keywords are contained in Utilities.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

same card as, and that immediately follow, the last parameter of a statement are printed with the statement.

If the LIST keyword is being processed, the subroutine at label LISTANL determines whether or not the keyword is on a COPY, COPYMOD, or ALTERMOD statement, and, if it is, whether or not the NO parameter is included. If the result of either test is negative, a message is issued and the module searches for the next COPY, COPYMOD, or ALTERMOD statement to process. Another condition for validity of the LIST keyword is that it not appear on a COPY statement of an earlier version of the IEBCOPY program. If the keyword does appear on such a statement, module IEBSCN gives control to the terminate module to end the job step.

If the MINBLK or MAXBLK keyword is encountered, a check is made to ensure that the keyword appears with a COPYMOD statement. If it does not, a message is issued and the operation is terminated. A check is then made to ensure that a valid number accompanied the keyword. If it is found to be invalid, the default value is assumed. (See Utilities for the default values.)

The subroutine beginning at label INDDANL processes any INDD statements that occur in a copy, unload, or load operation, and the subroutine beginning at MEMBRANL processes any MEMBER keywords that occur in a copy, unload, or load operation. In checking an INDD keyword a test is made to determine: whether more than one INDD keyword appears on a COPY statement; whether there is an INDD control statement following a COPY statement that contains an INDD keyword; or whether there is an INDD control statement following either a SELECT or an EXCLUDE statement. The result of the preceding test(s) determines whether an error message is required or whether the subroutine is to add a member to either the INDD table or the SETAB table. Preparatory to adding the member name to the appropriate table, the subroutine scans and verifies the statement.

Any invalid symbols or parameters that are detected during the scan are noted; the subroutine then gives control to the message-writing module to put out a message on the SYSPRINT data set.

If parameters or comments are continued on another card of the same statement, the RDCARD subroutine issues a GET macro instruction to bring the next card image into main storage for scanning.

After the member name has been read, it is moved into the proper table, and an indicator is set to reflect whether the name is an old name, a new name, and/or whether the 'replace' option is specified.

The "MEMBRANL" subroutine then analyzes a MEMBER keyword in the copy, unload, or load step to determine whether the keyword is part of either a SELECT or an EXCLUDE statement. If it is, and if the SELECT or EXCLUDE statement is the first one in the current copy, unload, or load step, pointers and counters for the SETAB are initialized before the keyword is analyzed. If the SELECT or EXCLUDE statement is not the first in the copy, unload, or load step, a SETAB has already been established (for a previous valid statement), and the subroutine will add to it as appropriate. If a copy, unload, or load step containing an EXCLUDE statement is encountered, the module's subroutine beginning at the label CHKEXCLD is entered. If a copy, unload, or load step containing a SELECT statement is encountered, the subroutine beginning at label CHKSELCT is entered.

Both routines perform essentially the same type of processing:

- Because the SELECT and EXCLUDE statements are mutually exclusive within the same copy, unload, or load step, each subroutine checks for the presence of the other statement.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- For a given statement (SELECT or EXCLUDE), if a like statement has already been processed in the same copy, unload, or load step, the member name(s) are to be added to the current SETAB table.

- If a SELECT or an EXCLUDE statement precedes rather than follows either an INDD statement or a COPY statement containing an INDD keyword (both within the same copy, unload, or load step), the subroutine gives control to the message-writing module to issue an error message. The module then flushes statements to either the next COPY, COPYMOD, or ALTERMOD statement or to a /* card.

When a SELECT or EXCLUDE statement has been analyzed and verified, the appropriate subroutine returns control to the main routine of the module to read the next control statement(s) until it reaches either the next COPY, COPYMOD, or ALTERMOD statement, an end-of-file, or the beginning of the next copy, unload, or load step. The scan module then retains the scanned delimiter statement for subsequent processing, and returns control to the caller to process the current copy, alter, unload, or load step.

MODULE IEBVCT—SETAB TABLE SORTING ROUTINE

Module IEBVCT sorts the SETAB table into an alphameric collating sequence. If one or more selected members are renamed, it constructs a new name pointer table (NNPTRTAB), which appears in storage following the SETAB table. The IEBCOPY program uses this module once for each select or exclude copy, alter, unload, or load step.

ENTRY POINT: IEBVCT

INPUT: Pointers indicating where the SETAB table begins and where the NNPTRTAB table should begin, if one is to be constructed. Counters indicating the number of entries in the SETAB table and the number of new names present, if any. New-name/old-name pairs in an I/O buffer, if new names are present.

OUTPUT: The SETAB table sorted alphamerically. A new-name pointer table, if new names are present. A pointer to the lowest available main storage address.

RECEIVES CONTROL FROM: Module IEBDSCPY

PASSES CONTROL TO: Module IEBDSCPY

LOGIC: Initially, the module determines whether or not a full copy, unload, or load (which does not require a sort of the SETAB table) is requested. If the SETAB table is to be sorted, module IEBVCT establishes two work area "slots" (ENTRY1 and ENTRY2) that are to contain pointers to the table entries for sorting. Starting with the first member name in the table, the module enters, into the slots, pointers to the first two member names (see Note) in the initial SETAB table. An alphameric comparison (for reference purposes, a primary comparison) is done on the two names to determine the lower one in the collating sequence. If the second name is lower, the contents of the slots are exchanged. The higher of the two names is then used with the next (that is, the third name in the initial SETAB table) to again fill the slots. The module then determines which of these two names is lower. At this point, the lower name from this second primary comparison may be lower than the member name resulting from the first primary comparison. To determine this, the module compares the "winners" of the first two primary comparisons. (This comparison is a form of secondary comparison and uses two other work area slots, NEWENT and SAVED.)

Note: The work area slots actually contain the addresses of the member entries, but the actual member names are used in the

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

comparison. If the SETAB table has more names to be sorted, the module places the next (now, the fourth) name from the initial form of the table into one (ENTRY2) of the slots whose contents have been exchanged and performs a primary comparison between it and the member name that is in the other slot (ENTRY1). (When the primary comparison procedure occurs, ENTRY1 always contains the member having the highest collating sequence value of the members already compared, and ENTRY2 always contains the next member name to be collated.) The winner of this (third) primary comparison then enters the secondary comparison (which is a form of "bubble" sort) and is compared against as many member names ahead of (lower than) it in the table until either there are no more names ahead of it or there is a lower name ahead of it.

Proceeding in a similar manner, the module compares (via a primary comparison) each name in the initial form of the SETAB table with the highest (collated) member name resulting from preceding comparisons. The winner of each of these primary comparisons is "bubbled" as far as it can go to the top of the table by means of the secondary comparison procedure. The result is that all member names ('new' and 'old') are positioned into an alphameric collating sequence in the SETAB table.

Figure 58 on page 118 is an example that illustrates the primary and secondary comparisons needed to sort a SETAB table (with initial entries given in the left column of the figure) into an alphameric sequence.

Another function of this module is to sort the old-name/new-name pairs that may exist in a work area buffer. Before the buffer is sorted, the old-name/new-name pairs exist in the order of the old names as they were entered into the SETAB table by the control statement scan module (IEBSCN). Module IEBVCT uses the same work area slots (ENTRY1 and ENTRY2) to sort the pairs into a sequence based on the new names. Using the new-name sequence of the name pairs, the module then constructs the new name pointer table (NNPTRTAB table), which contains pointers to the old names (of the new-name/old-name pairs) contained in the sorted SETAB table. The pointers in the NNPTRTAB are sequenced in the same order as the sorted new names.

MODULE IEBVDM—DIRECTORY MERGE ROUTINE

IEBVDM merges the input data set directory entries with output data set directory entries (if there are any) to build an updated output directory. For an unload operation, module IEBDSU creates the output data set directory.

ENTRY POINT: IEBVDM

INPUT: Directory entries for the current input data set. Directory entries (contained in directory blocks) for the output data set. (An "output" directory is not used if a compress operation is being performed.) Pointers and switches indicating addresses and status of output directory blocks and entries. Number of blocks allocated to the output directory.

OUTPUT: A merged output directory with an indication of the number of blocks in the directory and copied member names in CTLTAB entries.

RECEIVES CONTROL FROM: Module IEBDSCPY

PASSES CONTROL TO: Subroutines IEBMGI, IEBMGD, and TTOCC of module IEBDSCPY. Modules IEBVMS, IEBDV1, IEBDRD, and IEBWSU.

LOGIC: IEBVDM (the 'merge module') compares member names contained in input directory entries (IDEs) for the current input data set with member names contained in output directory entries (ODEs) for the output data set. The ODEs are contained in output directory blocks (ODBs) that constitute the output directory. (This directory may not be used, as previously noted, or it may initially contain no entries.)

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

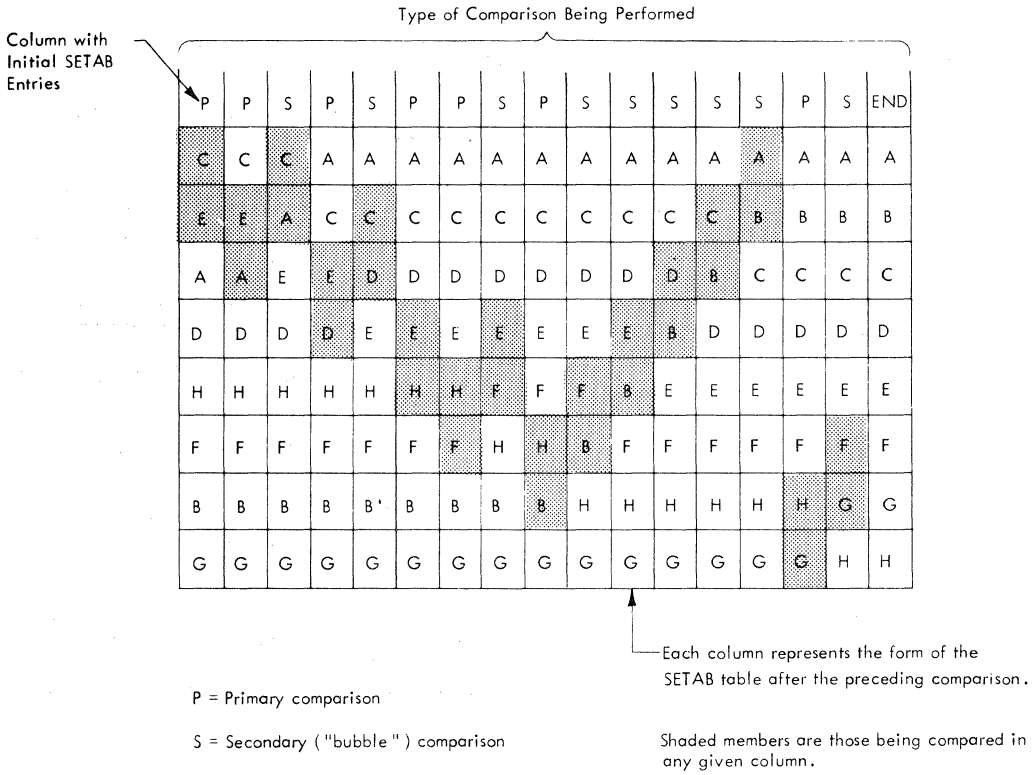


Figure 58. Example of Sorting SETAB Table by Module IEBVCT

This module performs its merging functions in the buffer space that had been previously allocated. If all the ODBs from the output directory had been retained in main storage by subroutine IEBMKO (of module IEBDSCPY), module IEBVDM merges the entries in the available main storage buffer space and writes the merged directory blocks onto the output data set. If some (or all) of the ODBs were not retained in storage, the first of the two available buffers is used for reading in unretained ODBs, and the second is used for constructing the updated (merged) ODBs.

Module IEBVDM obtains pointers to the available IDEs, one at a time, via subroutine IEBMGI of CSECT IEBDSCP2. The IDEs used are the ones that reference members that have just been copied to the output data set. Subroutine IEBMGI obtains the IDE pointers in alphameric order by using one of the sequences in which the IDE pointers in the CTLTAB table have been arranged. (See Figure 59 on page 120.)

In Figure 59 on page 120, the key letters (A to G) have processing associated with them as follows:

A: IDBs initially read into buffer. (Read until buffer is full or EOF is reached.)

B: IDEs moved into dynamic storage and/or spilled (B1 and B2) onto SYSUT3 if needed as CTLTAB table is built.

C: ODBs read into buffer from either output data set (C1) or SYSUT4 (C2).

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

D: ODBs moved into dynamic storage if SYSUT3 not used for IDEs. If IDEs have been spilled to SYSUT3, no ODBs are saved in dynamic storage.

E: Member data read into buffer and processed, if necessary, for note lists and/or user TTRs.

F: Member data written onto output data set. Updated TTRs of member data placed in corresponding IDEs.

G: Using all available buffer space (first or first and second buffers), updated IDEs from storage and/or SYSUT3 (G1) are merged with ODBs from storage and/or the output directory, which may be on either SYSUT4 or the output data set (G2).

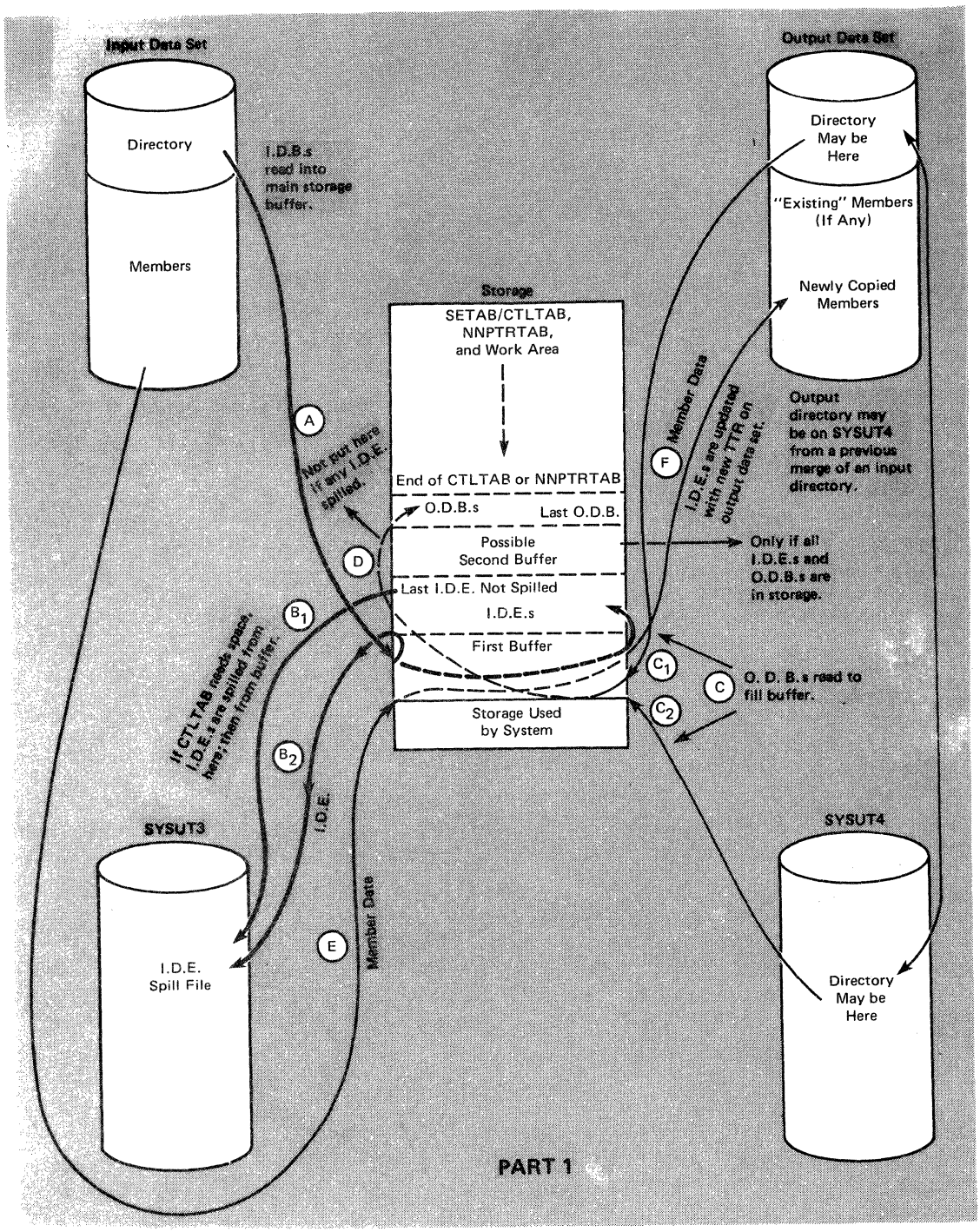
The merged output is placed either on the output data set (G3), if all ODBs were in dynamic storage, or on either:

- The output data set (G3), if ODBs were read from SYSUT4 or
- SYSUT4 (G4), if the ODBs were read from the output data set.

After the merge of each IDE, the corresponding CTLTAB table entry is flagged as 'don't copy' for the rest of the copy operation.

Note: Directory blocks are written on the output data set by a 'nonformat' write command to avoid erasing member data that may be on the same track at the end of the directory. Directory blocks are written on the SUSYT4 data set by a 'format' write command.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1



IDB = Input Directory Block
IDE = Input Directory Entry
ODB = Output Directory Block
ODE = Output Directory Entry

Figure 59 (Part 1 of 2). Data Movement for Directory Merging

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

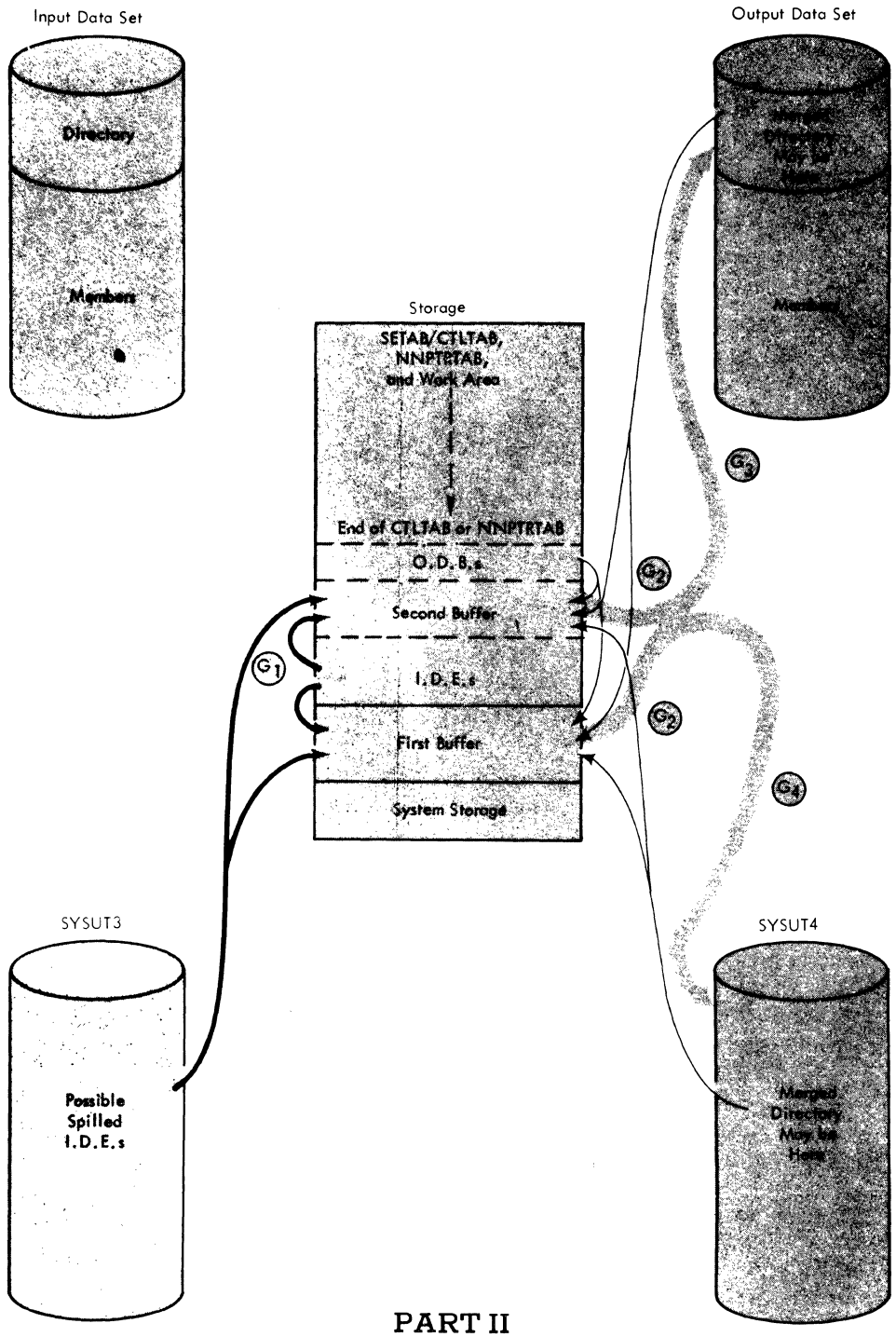


Figure 59 (Part 2 of 2). Data Movement for Directory Merging

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

If the data set is being compressed, there will be no output directory used to merge with the IDEs. Therefore, the IDEs will, in effect, be merged with themselves and will constitute the ODEs that make up the ODBs in the new output data set that is built.

If there are no entries in an existing output directory, for the case of a select or an exclude copy step, only the IDEs corresponding to members that have been copied will be used to build new ODBs. In a full copy, all IDEs are used, because all input members are copied when there are no entries in the current output directory.

Otherwise, if not all the ODBs are in storage, the directory merge module determines where the output directory currently exists. It may be either on the output data set, or on the SYSUT4 data set as the result of a previous merge of an input data set within the same copy operation. If the merged directory is to be written to the SYSUT4 data set for the first time in the current job step, the SYSUT4 data set will then be opened for use as a spill data set. Module IEBVDM uses IEBDV1, the validation module, to open the SYSUT4 data set and verify its characteristics.

The merge module either obtains a pointer to the next ODE in the buffer, or first uses module IEBDRD to read ODBs into the buffer and then uses subroutine IEBMGD of CSECT IEBDSCP2 to get a pointer to the first ODE in the first ODB in the buffer being used. In either case, if the member name for the ODE is FFFFFFFF, the ODE is the last one in the last ODB for the data set's directory, and any further merging is done using only the remaining pertinent IDEs.

Having both an IDE pointer and an ODE pointer, the merge module determines the alphameric sequencing relationship between the member names. If the names are equal, another ODE pointer is obtained (because members having the same name would have had the replace option applied to them if they are to be used at merge time). Otherwise, the module calculates whether the current directory entry (resulting from the IDE/ODE comparison) will fit into the current ODB. (Each ODB can contain a maximum of 254 bytes of directory entry information.) If the entry will fit, it is placed into the ODB, which still is in the buffer. If the entry will not fit, the current ODB in the buffer is ready to be 'set up' for writing onto an output device (as discussed under "Module IEBWSU"). Module IEBVDM then obtains a pointer to the area in the buffer at which the next ODB is to be built. If the buffer is full, its contents will be written onto the output device via modules IEBWSU and IEBDWR. Then the next entry to be merged will fit in the buffer area where the ODBs are being built.

For each IDE that is merged, module IEBVDM places the member name back in the CTLTAB table slot that referenced the member's IDE in the alphameric sequence portion of the table, and turns on the SEBIT4 bit for the table entry. If the member was renamed, the new name is the one that is placed in the CTLTAB table slot. The member's old name (obtained from a reference to the NNPTRTAB table) is also flagged by turning on SEBIT4. This prevents processing the entry for another input data set. The indicator bit, JSTCPD, in the second byte of the CTLTAB table entry is set on to indicate that this member name is to be printed when the terminate module gets control.

The output device on which the merged ODBs are written depends on the source of the ODBs for the current operation. In merging a previous input data set with the output data set for the current operation, module IEBVDM may have used the SUSYT4 data set. In that case (which will exist only if all the ODBs were not retained in main storage for the merge processing with the previous input data set), all the existing ODBs may be on either the SYSUT4 data set or the output data set.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

For merging the IDEs of the current input data set, one of the following will occur:

- If all the current ODBs are on the SYSUT4 data set (prior to the merge), they will be merged in a buffer with the current IDEs and the resulting 'new' ODBs will be placed on the output data set.
- If all the current ODBs are on the output data set (prior to the merge), but main storage cannot contain them all during the merge processing, then the merged ODBs will be placed on the SYSUT4 data set.
- If all the current ODBs are on the output data set (prior to the merge) and main storage can contain them all during the merge processing, the ODBs resulting from the merge are placed back onto the output data set.

Figure 60 summarizes the conditions just described.

When a given data set has been merged with the existing output data set, module IEBVDM returns control to the main flow module, IEBDSCPY.

Location of ODBs at Entrance to Module IEBVDM	Can Main Storage Contain All ODBs?	Location of "new" ODBs after Current Merge with IDEs
Output data set	Yes	Output data set
Output data set	No	SYSUT4 data set
SYSUT4 data set	Yes or No	Output data set

Figure 60. Directory Block Relationships for Merge Processing

MODULE IEBVMS—MESSAGE PRINT ROUTINE

The message module, IEBVMS, receives control to print all program error messages, diagnostic messages, and control statements for the IEBCOPY program.

ENTRY POINT: IEBVMS

INPUT: A message list with entries that indicate the numbers of the messages to be printed. An indicator for the last message to be printed before control returns to the calling routine. An indication of the return code to be given back to the caller of IEBCOPY.

OUTPUT: A printed record of program control statements and error or diagnostic messages.

RECEIVES CONTROL FROM: Any of the processing modules of the IEBCOPY program.

PASSES CONTROL TO: The calling routine.

LOGIC: Before module IEBVMS can print its output, the SYSPRINT data set must be open. This is accomplished (or at least attempted) by module IEBDV1. If, for some reason, a previous error occurred making the SYSPRINT data set unavailable to contain program messages, four courses of action are available:

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- A synchronous error routine message may be written out from a buffer, followed by a return to the calling module.
- A message indicating that the SYSPRINT data set cannot be opened may be written, followed by a return to the calling module.
- An end-of-job message with the highest severity code for the program included may be written, followed by a return to the calling module.
- A direct return to the calling module may be taken with the requested message not printed or issued.

If the SYSPRINT data set was successfully opened, module IEBVMS prints a header message when it receives control for the first time. Thereafter, as messages are to be printed, the module prints header messages only at the beginning of a new page of output. Header-message processing includes the setting up of the desired message, adjusting the line count for the print page involved, and setting the carriage space control (in the first byte of the message buffer) for skips between messages. Each page may contain up to 56 lines of output.

For the printing of control statements and diagnostic or error messages, the module either:

- Moves the control statement image into the message buffer, or
- Completes the message by inserting any necessary return codes or parameter list entries that are required and places the message into the buffer. To obtain the proper message from the table of messages included within the module, module IEBVMS uses a message number that is a part of the input to the module.

For the printing of any I/O error messages (for example, for errors encountered by modules IEBDRD, IEBDWR, IEBWSAM, IEBRSAM, and IEBLDUL), the error message will already have been placed in the message buffer by the I/O error routine, IEBIOE. This form of message passing is also used for some completion messages by module IEBCPMOD (entry point IEBCPMSG).

MODULE IEBVTM—TERMINATION ROUTINE

IEBVTM receives control to perform termination procedures for the IEBCOPY program. The functions provided by the module include: closing and/or reinitializing appropriate DCBs; writing termination messages; printing lists of names of members that have been copied, altered, unloaded, or loaded onto the output data set; recopying a merged output directory from the SYSUT4 data set to the output data set; and analyzing errors to determine the type of termination action to be taken.

ENTRY POINT: IEBVTM

INPUT: Communication area with fields (or switches) set to indicate error conditions.

OUTPUT: Messages, member lists, merged output directory on output data set.

RECEIVES CONTROL FROM: Modules IEBDSCPY, (including subroutine IEBMBT), IEBVTT, IEBDV1, IEBSCN, IEBDRB, IEBIOE, IEBVCT, IEBCPMOD, IEBWSU.

PASSES CONTROL TO: Modules IEBDSCPY, (including subroutine TTOCC and CCOTT), IEBVMS, IEBCPMOD (entry point IEBCPMSG), IEBDRD, IEBWSU, IEBSCN (via IEBDSCPY).

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

LOGIC: Initially, module IEBVTM closes the current input DCB, if it is open. The remaining procedures followed by the module depend on the conditions under which the module receives control. In order to clarify the main steps that are taken for certain forms of termination, Figure 61 can be used.

COPY OUTDD=XX INDD=AA,CC INDD=PP,QQ,SS	An INDD-group	A Copy Step	A Copy Operation
COPY OUTDD=YY,INDD=HH,LL,EE INDD=RR,TT,WW	An INDD-Group	A Copy Step	A Copy Operation
SELECT MEMBER=(V,K,M,P,R,B) INDD=BB,DD,FF	An INDD-Group	A Copy Step	
EXCLUDE MEMBER=(A,D,H,W,X) /*			

Section I presents conditions under which module IEBVTM receives control.
Section II illustrates the general processing that module IEBVTM performs for each condition when copying or loading.
Section III illustrates the general processing that module IEBVTM performs for each condition when unloading.

Section I				
Type I Members from data set AA have been copied, loaded, or unloaded; or members from data set CC have been copied or loaded.	**Type II** Members from data set SS have been copied or loaded.	**Type III** Members selected from data set WW have been copied or loaded.	**Type IV** End-of-file read (/* card). All copies, loads, and unloads that can be done are complete.	**Type V** Error termination of copy, load, or unload operation.
Section II				
Close data set AA (or CC). Reinitialize SYSUT3 and/or SYSUT4 if open. Print member names copied onto XX from AA (or CC). Return to entry point IEBMNI (in module IEBDSCPY) to copy data set CC (or PP, which is in the same INDD-group).	Close data set SS and data set XX. If output directory on SYSUT4, recopy it to output data set XX. Indicate space left on XX. Print members copied (onto XX) from SS. Return to IEBSCN (via IEBDSCPY) to scan control statements for next copy operation. Print unused tracks and unused directory blocks messages.	Close WW. Print the selected members from data set WW. (Only those members that have been copied.) Return to scan the next INDD statement.	Print message giving highest completion code for the job step. Close all open data sets. Return to caller (via operation system).	Close all open input and output data sets. If continuation is possible, reinitialize SYSUT3 and SYSUT4 if they are open. Either: a. return to scan as in Type II, or b. if can't continue or no more copy operations, terminate as in Type IV.
Section III				
Close data set AA and data set XX. Print members unloaded onto XX from AA. Return to print error message (more than one INDD specified). Return to module IEBSCN to scan control statements for the next copy, unload, or load operation.	This situation cannot occur when unloading.	This situation cannot occur when unloading.	The same applies as described in Section II.	The same applies as described in Section II.

Figure 61. Examples of IEBCOPY Termination Conditions and Processing

MODULE IEBVTT—ROUTINE TO SORT CTLTAB TABLE

Module IEBVTT sorts the relevant CTLTAB table entries according to the input TTRs of the entry members.

The relevant entries are those that were 'found' in the current input data set and for which:

- The same member name does not already exist on the output data set, or

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- If the same member name exists on the output data set, the REPLACE option was specified.

ENTRY POINT: IEBVTT

INPUT: The address of the CTLTAB table. The number of sortable entries for the current input data set.

OUTPUT: A doubly sorted CTLTAB table: alphamerically and by input member TTR.

RECEIVES CONTROL FROM: Module IEBDSCPY

PASSES CONTROL TO: Modules IEBDSCPY, IEBVMS, and IEBVTM

LOGIC: The module takes one CTLTAB entry at a time and determines if the entry is to be sorted by member TTR. (For example, an entry is not to be sorted if the 'don't-copy' bit is on or if it is a 'new name' entry.) If so, the module moves the alphamerically sorted directory entry pointer from bytes 2 to 5 of the CTLTAB slot into a buffer. If there are more entries to be sorted, the process is repeated. After all entries have been moved to the buffer, the relevant table entries (all of which are still in the CTLTAB table) are sorted.

For each member to be copied, the module compares this input member's TTR (contained in bytes 7 to 9 of each relevant CTLTAB table entry) to the member TTRs for other input members. The TTRs are compared in a manner that arranges them in ascending TTR order with respect to the first (lowest TTR) input member to be copied. As a member's TTR is given a place in the list, its associated directory-entry pointer is moved into the corresponding location (the second through the fifth bytes) of the CTLTAB table.

After all relevant CTLTAB table entries have had their directory-entry pointers arranged in an order corresponding to that of the ascending input member TTRs, module IEBVTT moves the alphamerically sequenced directory-entry pointers, from the buffer where they had been stored, back into the CTLTAB table. This time, each pointer is placed back into the last 4 bytes of the 10-byte CTLTAB table 'slot' from which it was extracted, overlaying the input member TTR.

The process results in a table in which the directory entry pointers appear in one sequence according to an alphameric arrangement, and in another sequence according to the input member TTRs.

MODULE IEBWSAM—WRITE ROUTINE FOR UNLOADED DATA SETS

IEBWSAM writes directory blocks and member data with its note lists from the buffer to the unloaded data set until all data has been written. This module also writes the two specific header records used for the data sets that have been unloaded by the IEBCOPY program.

ENTRY POINTS:

IEBWSAM—to write directory blocks and member data.

IEBWSAM1—to write the two header records used for unloaded data sets.

INPUT:

- Switches set to indicate whether a member, and possibly a note list, has to be written.
- Addresses of the two descriptor records.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

OUTPUT:

- All data from the buffer which was written onto the unloaded output data set.

  Note: If the data is member data, only one member is written at a time.

- The two descriptor records written onto the unloaded output data set.

LOGIC: At entry point IEBWSAM, the module checks whether member data or directory blocks are to be written. In the case of member data, a test is made for the presence of user TTRs (relative track address), and, if present, the directory entry of this member is inspected for the indication of a note list. If a note list is present, it is written onto the unloaded data set followed by its member data present in the buffer. Next, member data without note list or directory blocks, are processed in the same way.

Because the unloaded data sets have variable spanned records, the IEBWSAM module saves an 8-byte area preceding the actual block to be written.[6] This area is reset to its initial value after it has been used by the write instruction. Each partitioned data set record is tested to see whether it is the last one in the buffer. As many partitioned data set records as possible are blocked in one unloaded data set block. An error message might be issued when a partitioned data set record exceeds the size of an unloaded data set block (determined by the device block size capacity). When the total buffer has been written, control is returned to the caller.

MODULE IEBWSU—WRITE SETUP ROUTINE

IEBWSU (the write setup module) determines an output address to which each record (member or directory entry) will be written from a buffer. The count field of each input member's record is changed to reflect the 'new' address when the member is written on the output partitioned data set. If the member that has been read from the input data set contains user and/or note list TTRNs, this module adjusts the TTRs to correspond to the output data set addresses of these records.

ENTRY POINT: IEBWSU

INPUT: Pointer to first input buffer record to process. Switches indicating the need, or lack of need, for user and/or note list TTRN processing.

OUTPUT: A pointer to the next location (in the input buffer) following the last record that has been processed.

RECEIVES CONTROL FROM: Modules IEBDSCPY (including subroutines IEBMBT and IEBMNM), IEBVTM, IEBVDM, IEBCPMOD, and IEBDRB.

PASSES CONTROL TO: Modules IEBVMS, IEBVTM, IEBDWR, and subroutines TTOCC and CCOTT of the module IEBDSCPY.

LOGIC: Basically, this module obtains the output addresses for use by the write module, IEBDWR, in writing:

- Member data onto the output partitioned data set.
- Directory information onto the output data set or onto the SYSUT4 data set.

---

[6] The 8-byte area is not saved for records following the first record in the buffer, because only records that have already been processed are overwritten by the record descriptor words.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- Input directory entries onto the SYSUT3 data set.

If there are compress-in-place, blocking, track overflow, or user TTRN/note-list considerations, this module uses self-contained subroutines to do the indicated processing.

For each record, the module determines the number of data bytes remaining on the first available track of the output data set and determines, in turn, whether the record can fit on that track. This number is called the 'track balance' for the particular track. To determine balance characteristics, the module uses information obtained by the DEVTYPE macro instruction and placed in the communication area by module IEBDV1.

The first available track TTR address is converted to the MBBCCHHR form and placed in the count field of the record that is to be written so that the channel program of the IEBDWR (write) module can properly write the record. After each record is prepared for writing, a loop is made, if necessary, to continue until all input records are processed. Then, the write module receives control if the buffer space is exhausted. Otherwise, control returns to the caller.

For user TTRNs and/or note lists, each TTR is updated to reflect the address of this record on the output data set. The presence of more than one note list per member causes the setup module to give control to module IEBVTM for termination of the copy step.

When the setup module structures the records (in the output buffer) for track overflow conditions, it sets a bit in each overflow segment of a record that is to go on the output data set. Either the PCI appendage module (IGG019C8) or the write module (IEBDWR) examines this bit and, if the bit is on, it sets a Write Special (X'01') command in the channel program. If the bit is off, the PCI or IEBDWR module sets a non-Write Special (X'1D') command. As overflow records are written, the system hardware detects the X'01' command code in either CCW W4 or W6 (see the first channel program described under "Module IEBDWR") and sets its own indicator for any subsequent reading of the overflow record.

For reblocking of member records, there must not be a user and/or note list TTRN specified. The presence of either is an error and causes termination. The record length for reblocking does not have to be validated by this module, because the deblock module IEBDRB performed that function.

Except when member records are being blocked, module IEBWSU performs validity checking as indicated in Figure 62 on page 129.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Record Format	Characteristic Required	Action if Characteristic not Satisfied
U, UT, V, VT, VB, VBT	Record length equal to or less than block size.	Issue warning message and continue processing.
FT, FBT	Record length equal to block size; or less than block size by an amount that is an integral multiple of the logical record length.	
FS, FBS	Record length equal to block size; or, if the last record before an EOF, less than block size by an amount that is not an integral multiple of the logical record length. (In Standard format, the last record may be shorter than the other blocks.)	

Key:
F = Fixed length
V = Variable length
U = Undefined length
B = Blocked
T = Track overflow
S = Standard

Figure 62. Validity Checking Performed by Module IEBWSU

Figure 63 lists the conditions under which module IEBWSU passes control to module IEBDWR for writing various types of member data.

Condition for Passing Control to Module IEBDWR	Types of Data to be Written				
	Member Data: Straight Copy	Member Data: Compress	Member Data: Track Overflow on Output	Member Data: Block or Deblock	Directory Entries onto: SYSUT3, SYSUT4, or output
For each new output track, until last record is processed. Then return to caller.		X (Writes as many integral tracks as are contained in the buffer.)			
If no buffer space available after last record processed. Then return to caller. **Note:** If buffer space is available, return to caller.	X				
When last record in buffer is processed. Then return to caller.			X (IEBWSU also calls IEBDWR once for each track to be erased.)	X	X

Figure 63. Conditions Under which Module IEBWSU Passes Control

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

MODULE IGG019C8—PCI ROUTINE

IGG019C8 is a PCI appendage module that receives control from the I/O supervisor. The I/O supervisor gets control after a channel program executes a channel command word (CCW) that has the PCI flag bit turned on. If the CCW specifies a Read operation, the appendage module determines (while that operation is being performed) whether more records are available for reading, and whether the buffer into which the records are being read can contain the next record; in addition, it determines the main storage address into which the record is to be read. If the CCW specifies a Write operation, the module determines (while that operation is being performed), whether there are more records to write, whether there is another (unwritten) record in the buffer, and, if so, where the record is located in the buffer.

ENTRY POINT: IGG019C8

INPUT: Registers set by the I/O supervisor. Pointers and switches set to indicate the data set involved, the type of I/O operation being performed, the Read or Write CCW being executed, the Read or Write CCW to be executed next, and the location of data.

OUTPUT: Address of last record successfully read or written. Updated CCWs (for next I/O operation). Pointers and switches set to indicate the condition of the data buffer.

RECEIVES CONTROL FROM: IOS (as result of PCI bit being on in a Read or a Write CCW.)

PASSES CONTROL TO: IOS.

LOGIC FOR READING: When the PCI appendage module receives control during a Read channel program, it determines the condition for which the operation was being performed. If data set compression is being done, the track balance (which includes factors such as gap length and tolerance) is computed and made available for use by the write setup module (IEBWSU), because that module determines the locations for placing data on the output data set.

For the reading of input member data, and for reading directory blocks, the appendage module determines buffer and record availability and location. The channel command program described for module IEBDRD is modified so that it will either continue to read more records, or stop and return control to the main flow of that module. To do this, the appendage module exchanges the CCWs or pointers to the CCWs as necessary. Module IGG019C8 reinitializes the CCW being executed so that, if an I/O error condition requires that the channel program be restarted from the CCW in control at the time of the error, the same record will be read, but the PCI-exit will not be taken.

If a new output directory is being read, it may have space allocated for one or more directory blocks. Because these blocks have no entries other than the standard hexadecimal 'FF's for the last entry, the appendage module only counts the number of blocks so that the module performing the merging operations will be able to determine when to stop writing 'new' directory blocks. For this type of processing, the module sets the two 'Read' CCWs to the same form to permit reading into the same buffer area until an end-of-file indication is reached.

On return to the I/O supervisor, a list of changed CCWs is passed. This list is necessary to update the real storage copy of the CCWs used by the supervisor.

LOGIC FOR WRITING: When module IGG019C8 is entered during a 'Write' channel program, it first determines whether a read-back check operation is being performed. If so, it prepares the CCWs for a restart if the current read-back operation should fail.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

When the appendage module is given control in the case of writing members or directory blocks, its basic function is to determine the place from which the next block or member is to be written, if such a block or member exists in the buffer. In order that the I/O operation may be restarted if necessary, the module sets the PCI flag bit in the currently-executing CCW to an off condition, and readjusts the record number of the record to be written next so that it is equal to that of the record currently being written.

When the PCI routine determines that the next record to be written is to go on the next track of the output device, it provides for head switching by adjusting the pointer to the next CCW so that it points to a 'Search' command with the multitrack code. To write the overflow record segment, the next 'Write' CCW is changed to a Write Special CCW.

On return to the I/O supervisor, a list of changed CCWs is passed. This list is necessary to update the real storage copy of the CCWs used by the supervisor.

MODULE IGG019FT—SIO APPENDAGE

IGG019FT is an appendage module that receives control from the I/O supervisor. It builds an extended translation list for either the Read, Write, or Read-back Check CCWs. This module also determines which storage locations should be fixed so that disabled page faults do not occur in the PCI appendage IGG019C8.

ENTRY POINTS: IGG019FT and IGG019FT+4

INPUT:

Entry Point IGG019FT—A pointer to the IEBMCA main communication area. A switch (within IEBMCA) indicating whether appendage module IGG019C8 is to be used. Several fields in the IEBMCA used for addresses and lengths of lists to be built.

Entry Point IGG019FT+4—A register containing the address of the fix list.

OUTPUT:

If PCI Handling Was Requested—Registers containing the address of the CCW list and the number of entries in that list. The fix list containing the start and end addresses of the IEBMCA main communication area and of the first and second buffers. A register containing the number of areas delimited.

If No PCI Handling Was Requested—The register giving the number of entries is set to zero.

LOGIC FOR CCW TRANSLATION: At entry point IGG019FT, an extended translation list is built. The addressability (using register 4) to the IEBMCA main communication area is set up. A test is made to check whether the I/O request requires PCI handling and, if not, control is returned to the caller. If PCI handling is requested, in the case of a read request, the Read CCW list address is given in register 10 and the number of addresses in that list is put into register 11. Similar actions are taken, in the case of a write request, with the Write CCWs and, in the case of a read-back check, with the Read-back Check CCWs. The return address is then increased by 8 and control is returned to the caller.

LOGIC FOR PAGE FIXING: At entry point IGG019FT+4, addressability to the IEBMCA main communication area is set up. A test is made to check whether the I/O request requires PCI handling and if not, control is returned to the caller. If PCI handling is requested, the start and end addresses of the IEBMCA main communication area and the two buffer are stored in the fix list. The count of entries in the fix list is put into register 11. Control is then returned to the caller.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

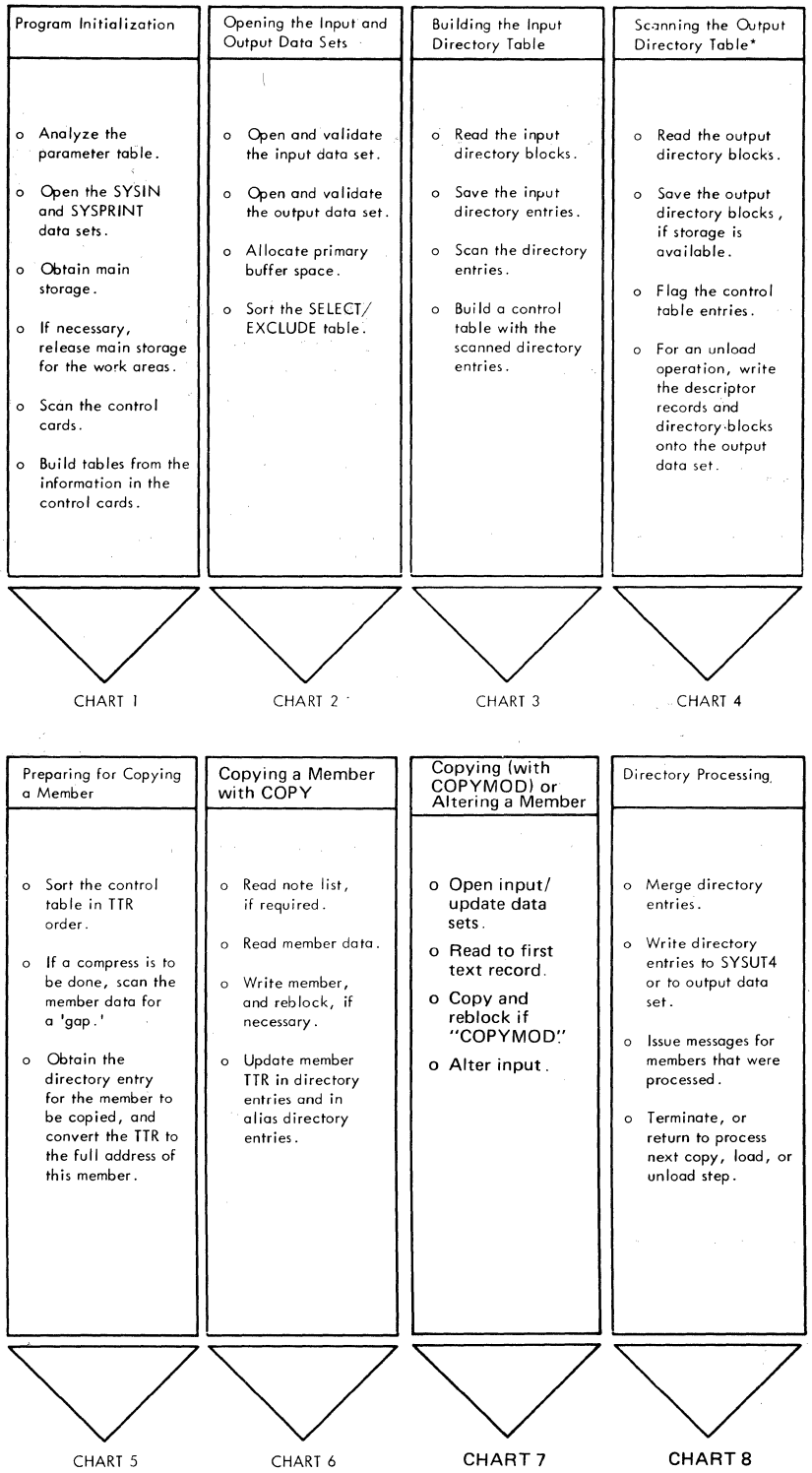
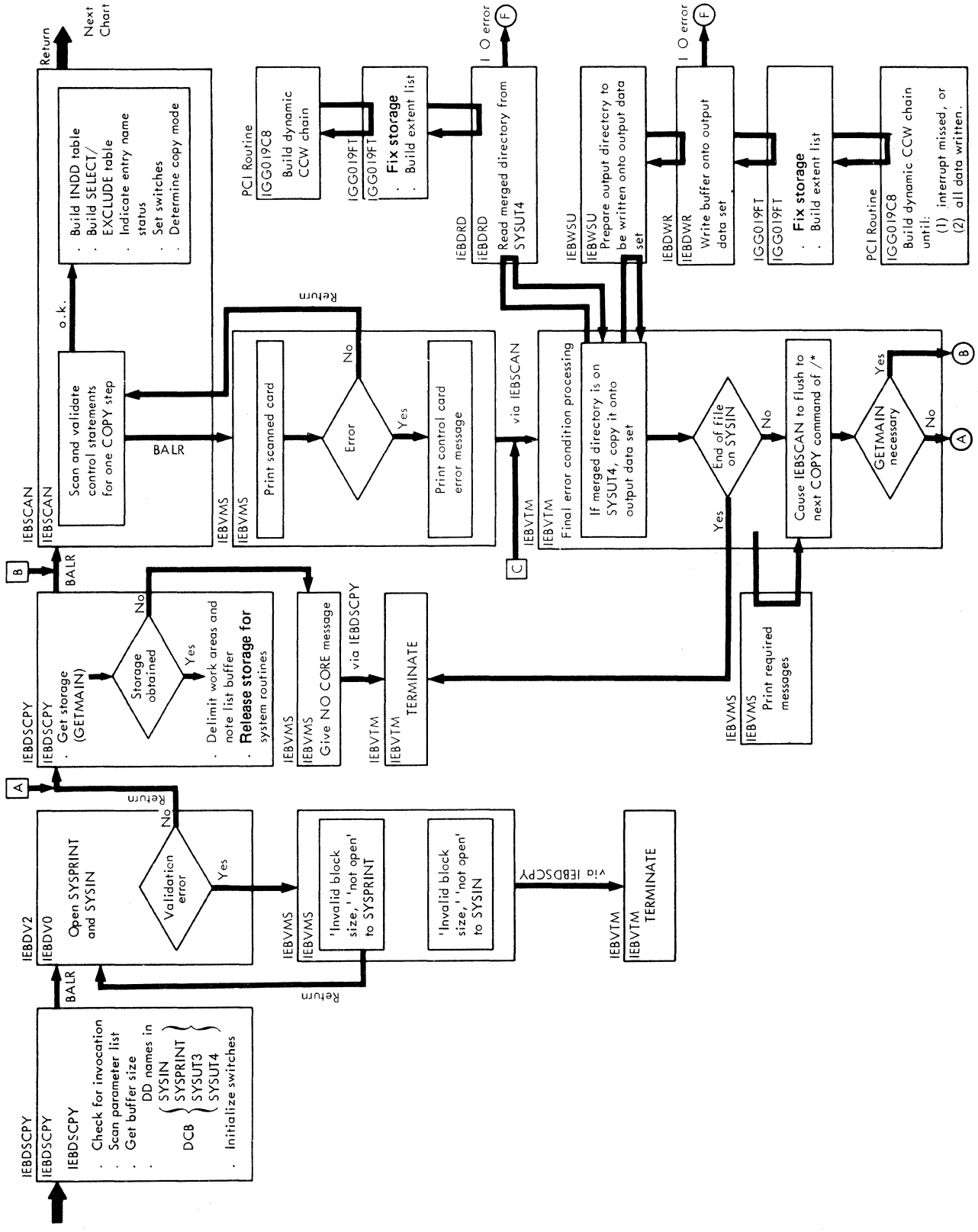


Figure 64. Summary of Program Operation

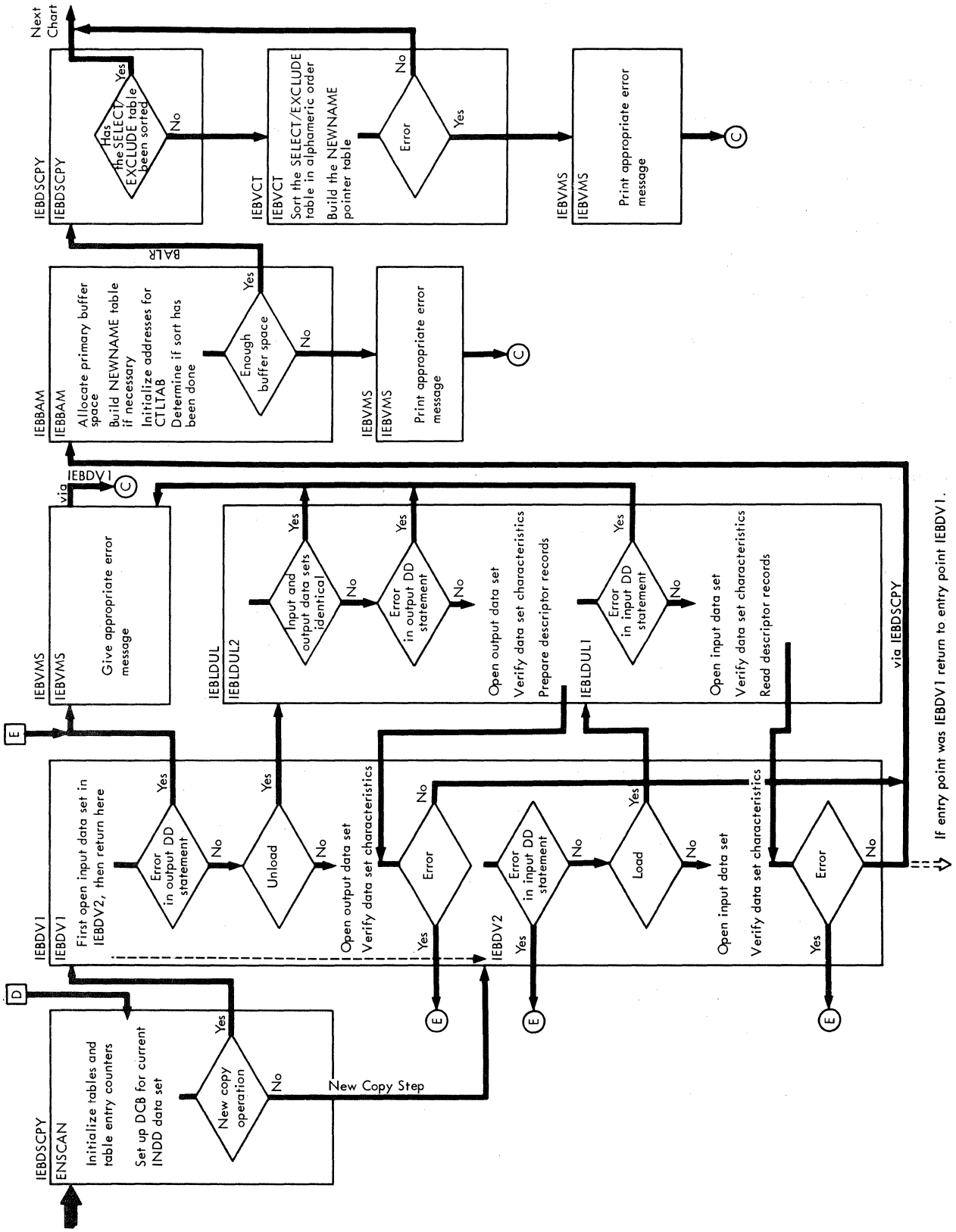
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 01. PROGRAM INITIALIZATION



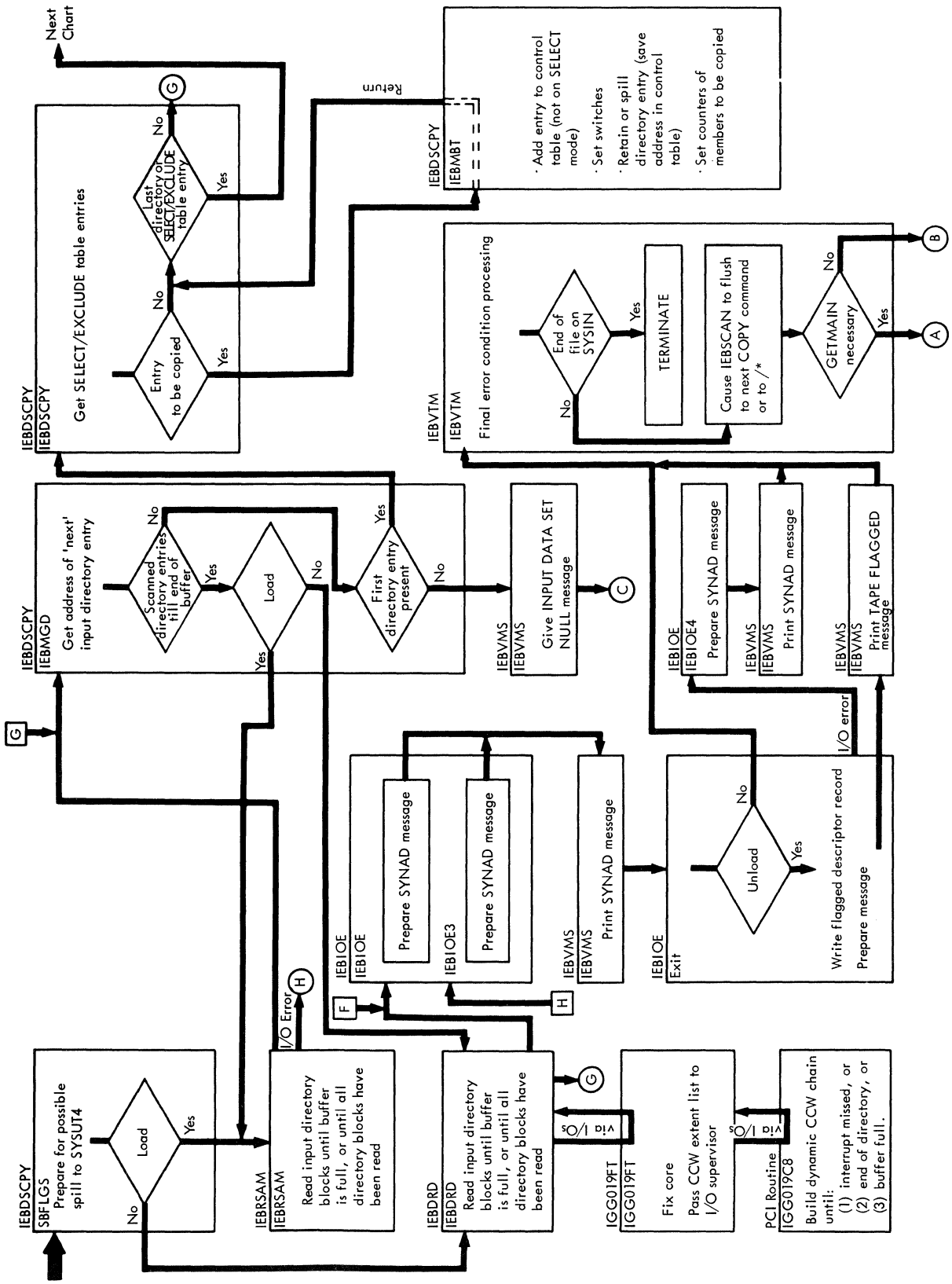
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 02. OPENING THE INPUT AND OUTPUT DATA SETS



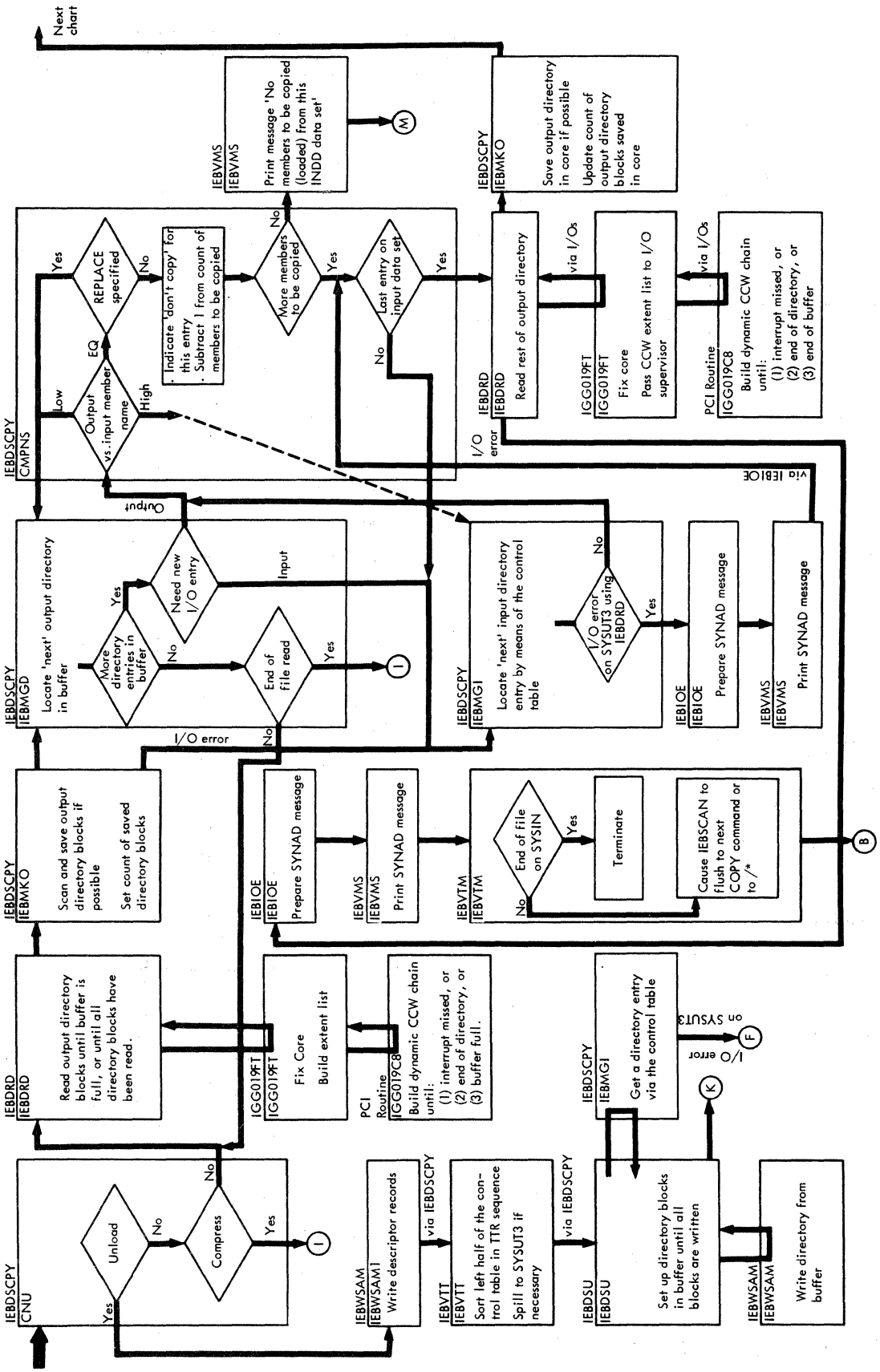
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 03. BUILDING THE INPUT DIRECTORY TABLE



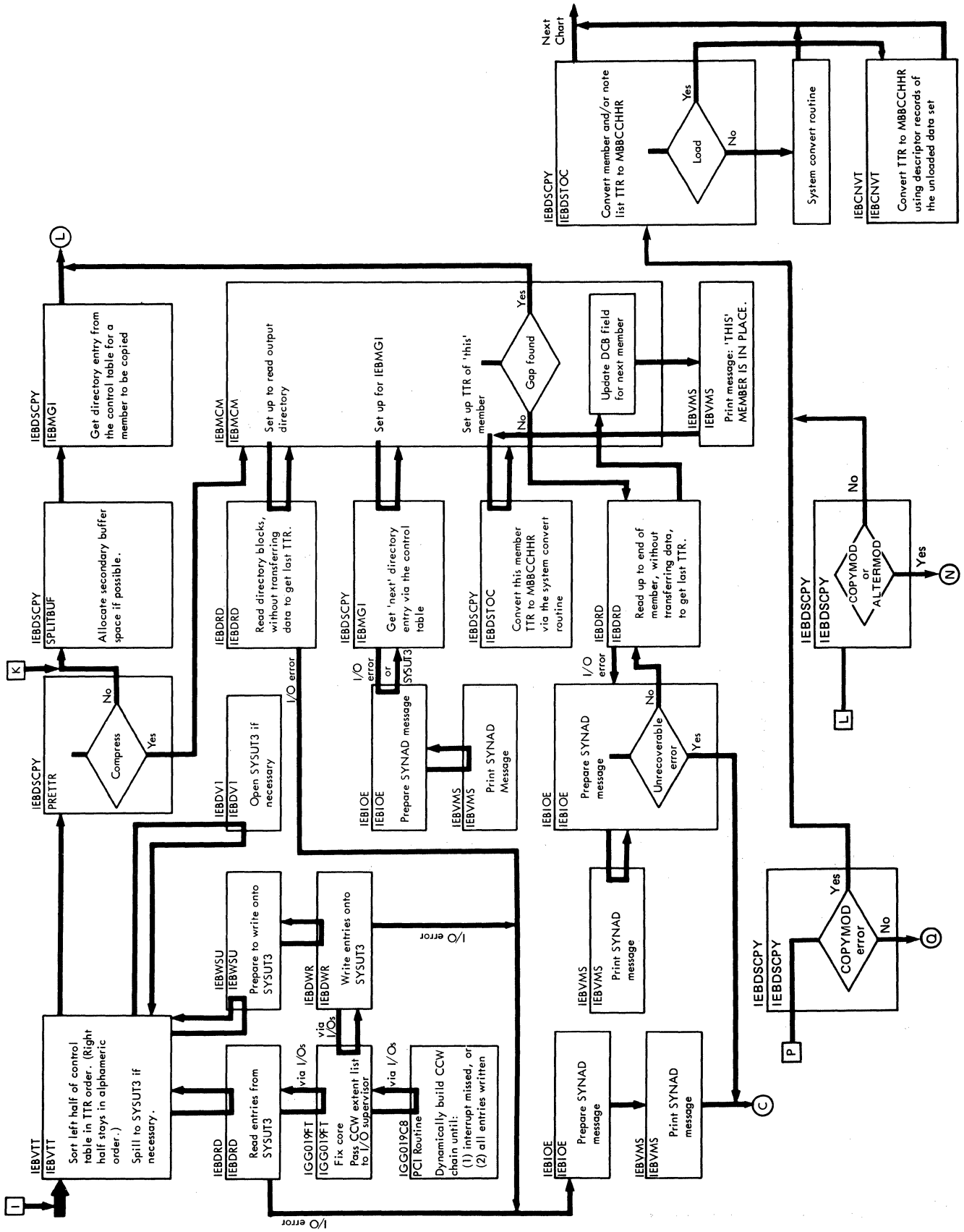
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 04. SCANNING THE OUTPUT DIRECTORY TABLE



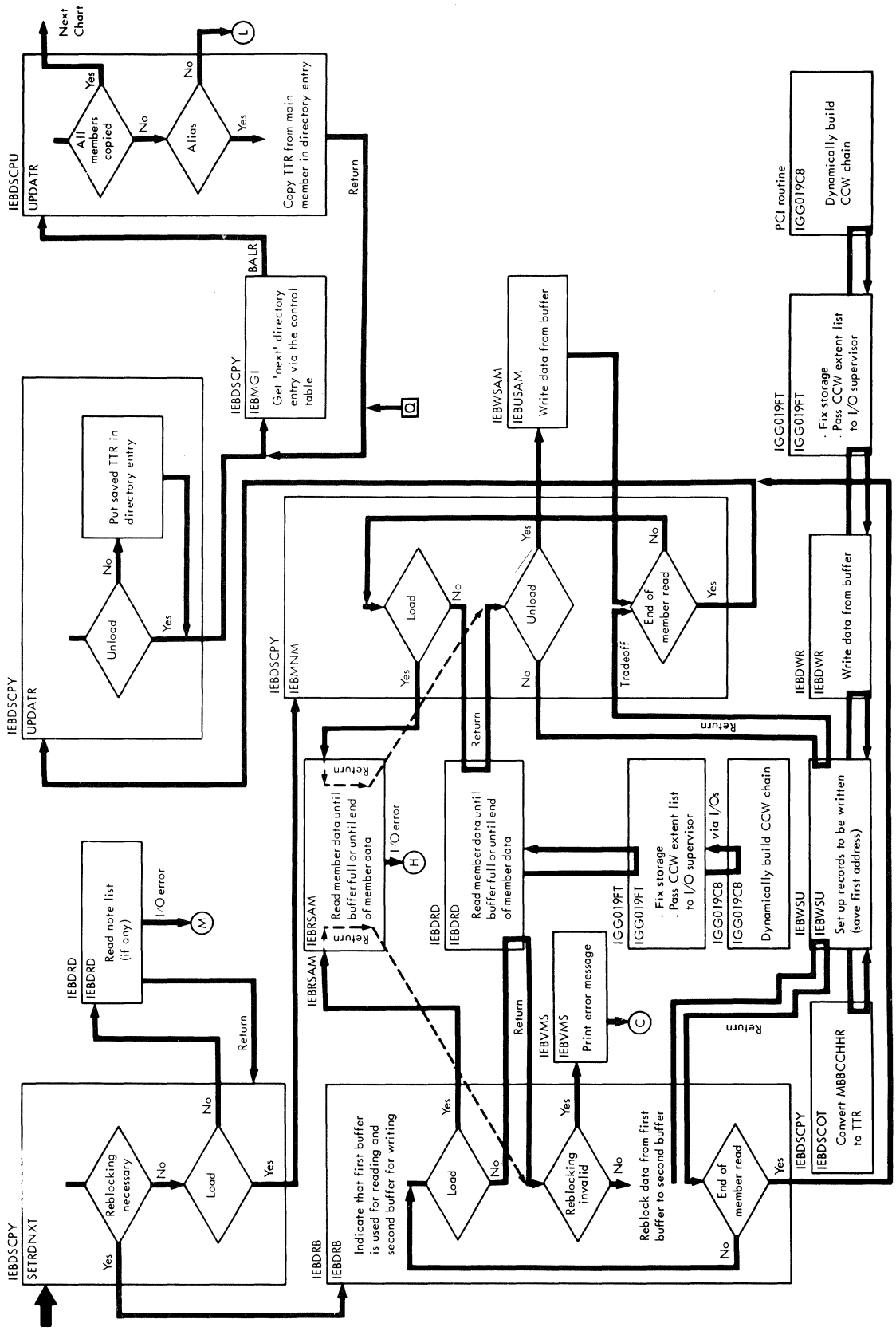
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 05. PREPARING FOR COPYING, UNLOADING, OR LOADING A MEMBER



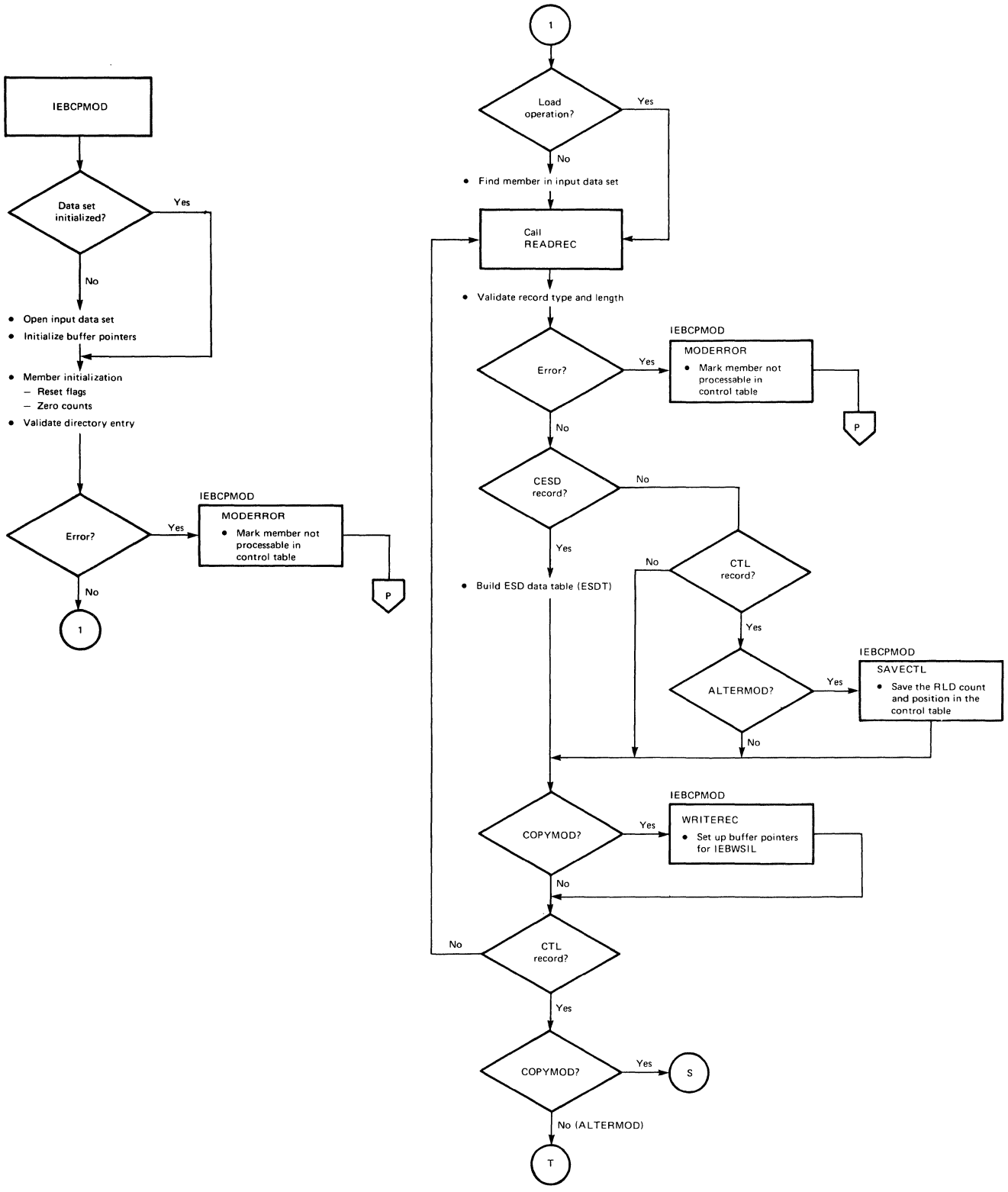
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 06. COPYING, UNLOADING, OR LOADING A MEMBER WITH COPY



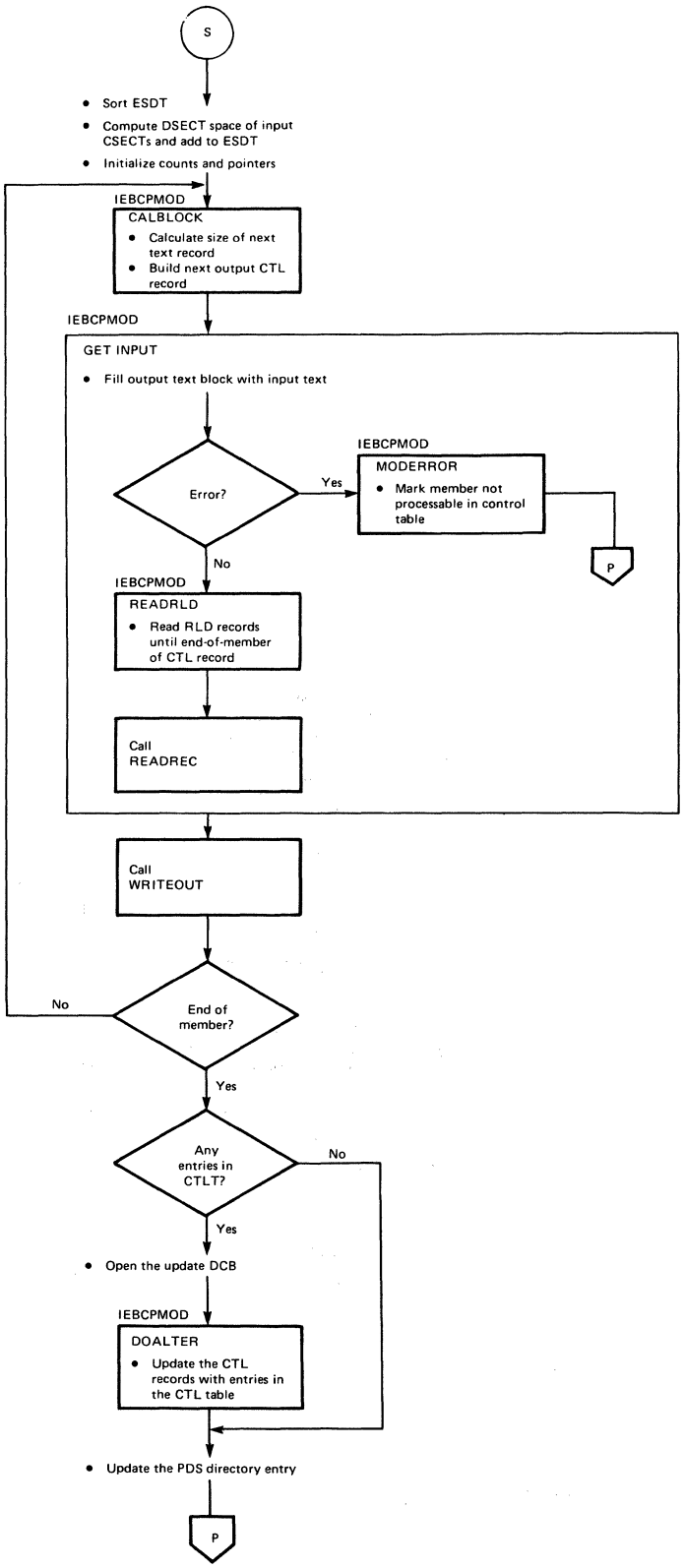
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 07. COPYING OR ALTERING A MEMBER WITH COPYMOD OR ALTERMOD (PART 1 OF 5)



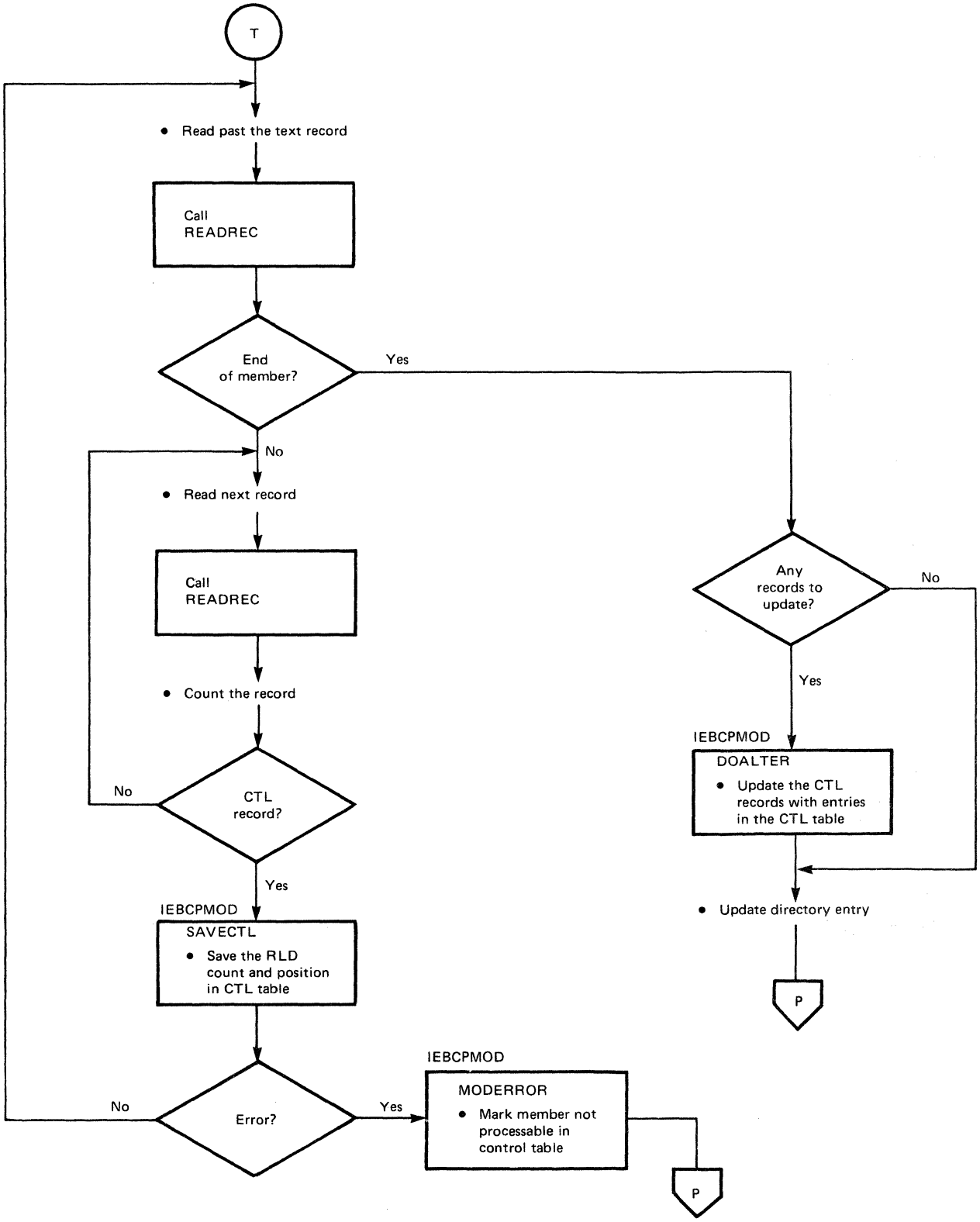
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 08. COPYING OR ALTERING A MEMBER WITH COPYMOD OR ALTERMOD (PART 2 OF 5)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

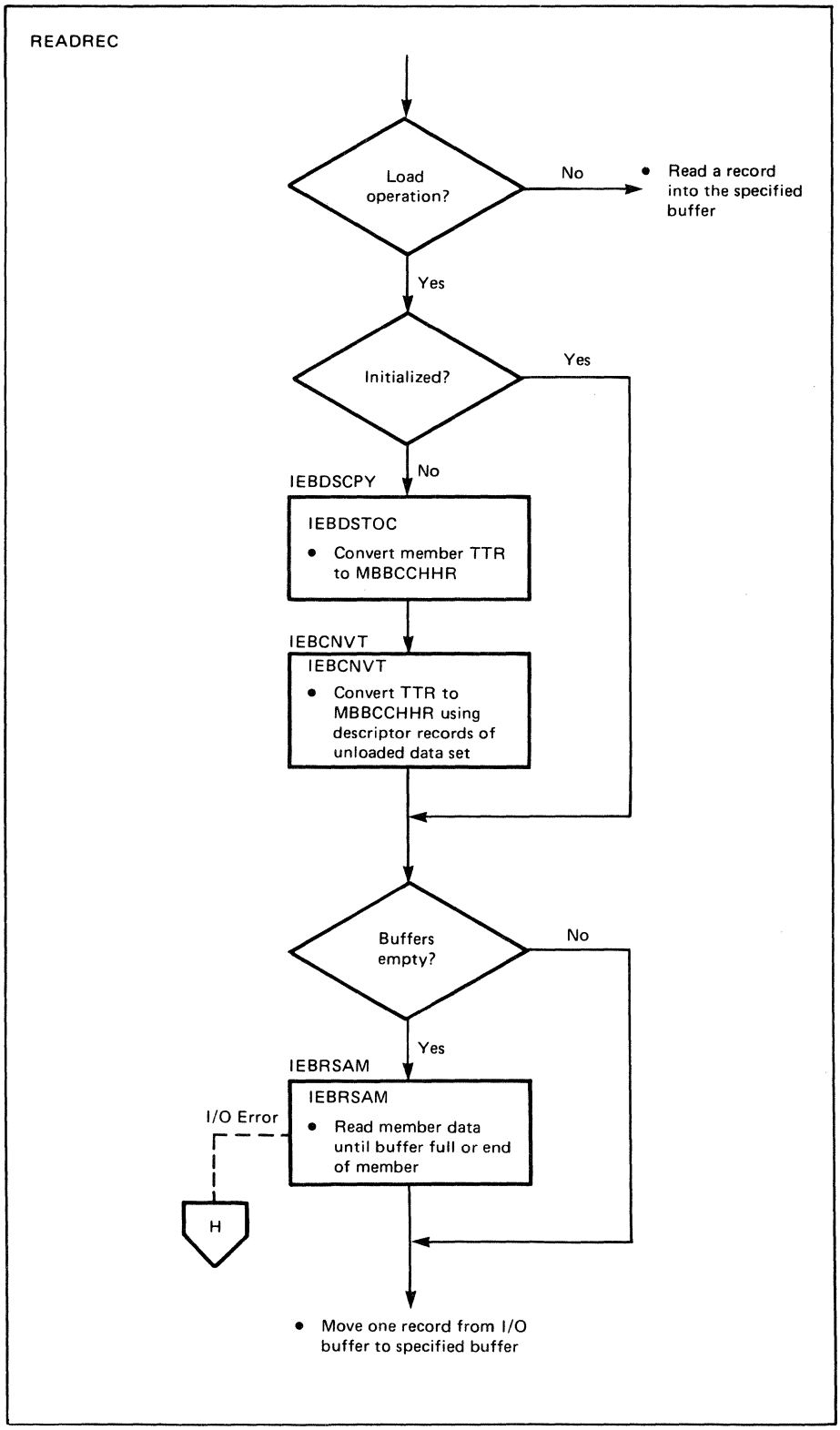
CHART 09. COPYING OR ALTERING A MEMBER WITH COPYMOD OR ALTERMOD (PART 3 OF 5)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

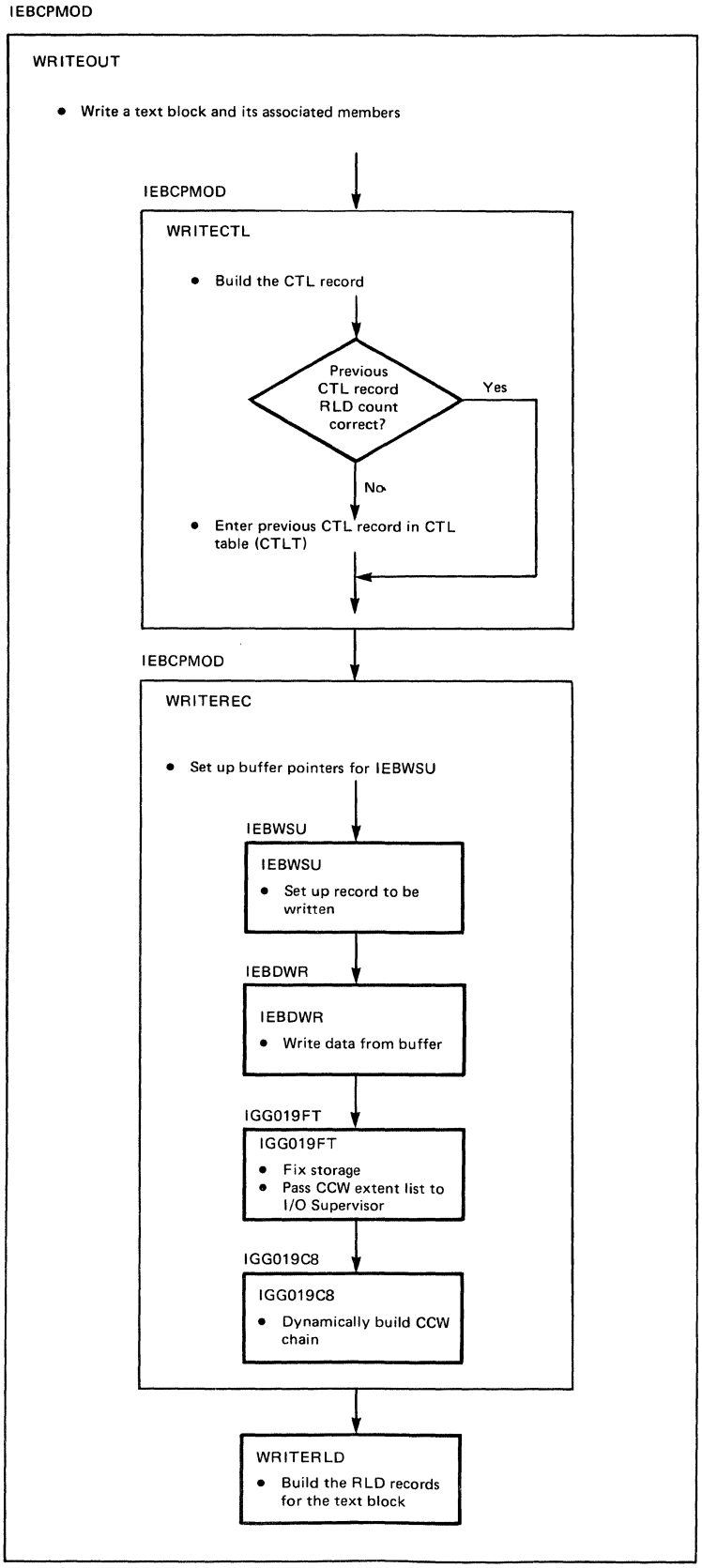
CHART 10. COPYING OR ALTERING A MEMBER WITH COPYMOD OR ALTERMOD (PART 4 OF 5)

IEBCPMOD



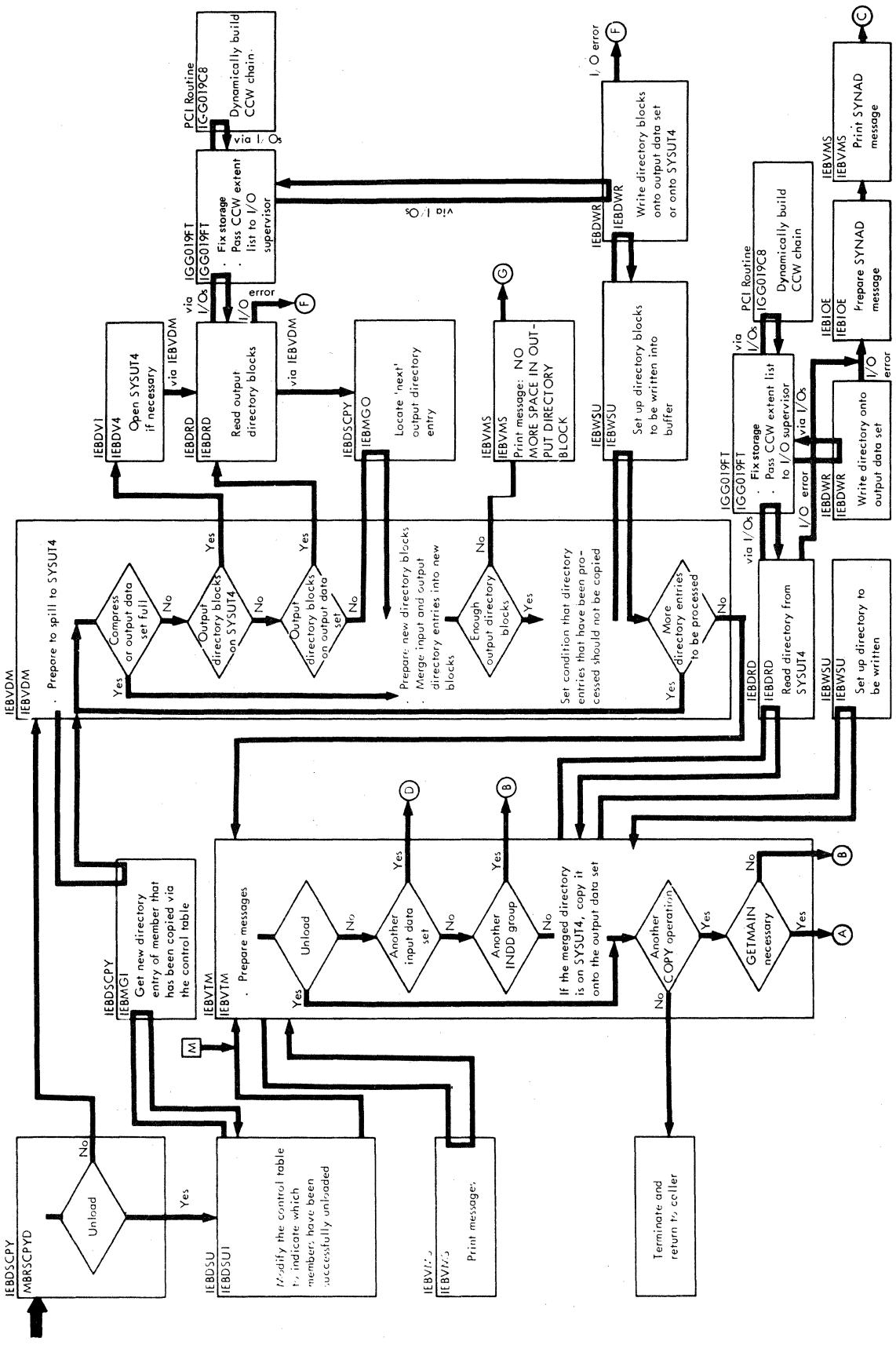
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 11. COPYING OR ALTERING A MEMBER WITH COPYMOD OR ALTERMOD (PART 5 OF 5)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 12. DIRECTORY PROCESSING



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

The data generator program (IEBDG) creates test data that can be used in program debugging procedures. The program will construct multiple data sets within a job that uses either the physical sequential, the indexed sequential, or the partitioned access method. The records within these output data sets may consist of fields that are defined by any one of seven IBM character formats, each of which may be modified by any one of eight types of action. Alternatively, a user may elect to provide an output pattern in the form of a "picture" instead of an IBM format. If desired, a user may also inspect and modify the output records before the records are written in the test data set.

The IEBDG program is a problem program, which may be executed as a job step by use of the job control language, or which may be invoked by a calling program using either the LINK or the ATTACH macro instruction. Specification of either the program name (IEBDG) or the procedure name on the EXEC statement causes control to be given to the data generator program. In the case of invocation of the IEBDG program, the entry point (EP) parameter in the macro instruction operand specifies the program's symbolic name. Job control statements or parameter list information, and the IEBDG utility control statements, maintain control of the program and describe or specify the functions to be performed. They also describe or define the input and output data sets to be used. Depending on the specifications of the user, the records of the input data set may be either blocked or unblocked.

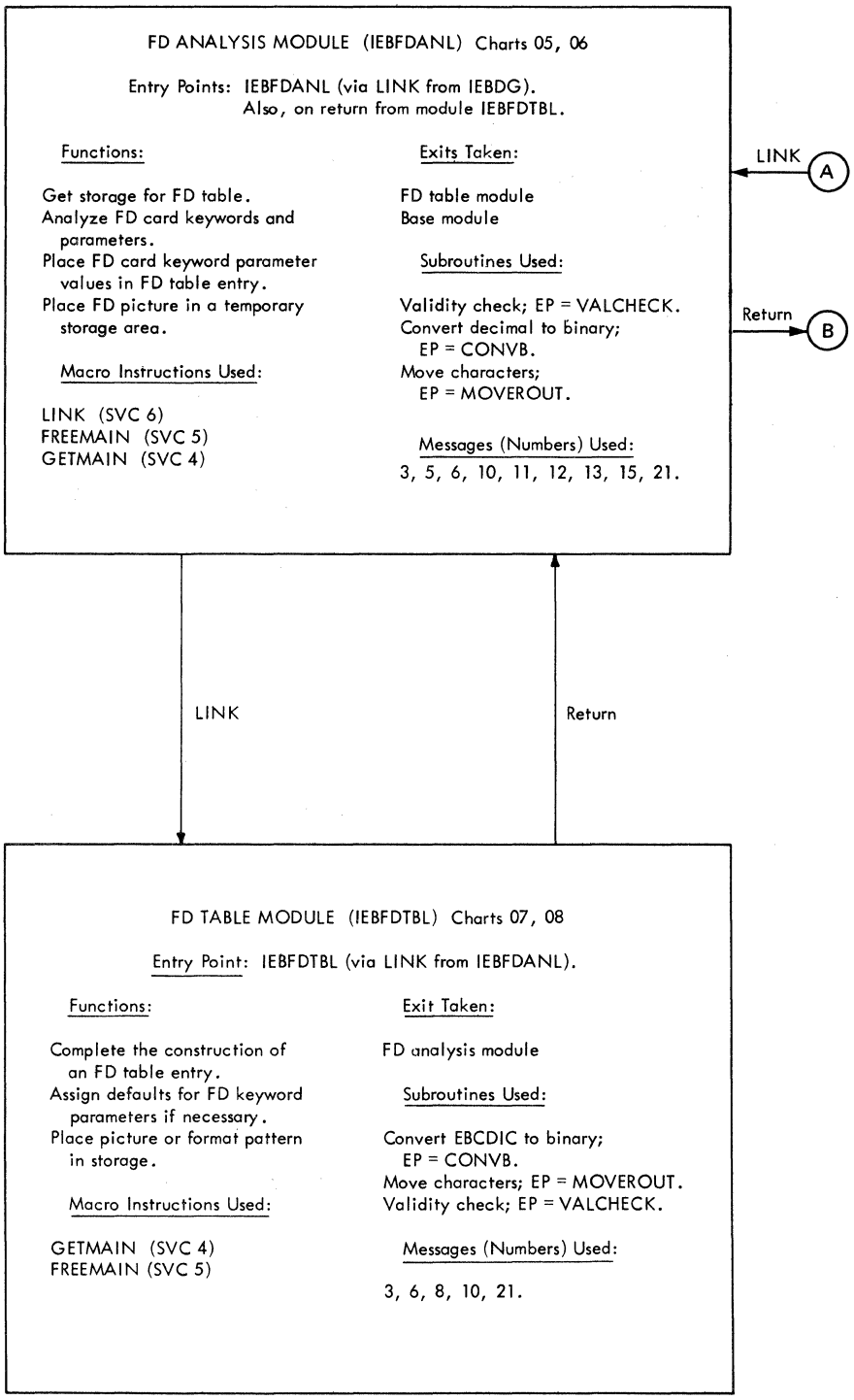
In the case of output records, the fields within a record may be repeated as desired, and the output records may be a part of a logical block, which may also be repeated. If an existing data set is used as the input data to the program, the fields within the individual records of the data set may be retained, modified, or replaced as desired. The contents of the output (test data) records are defined by the utility program control statements.

The data generator program has a "field-select" capability that permits specific fields from an input data record to be placed in the output record at a given starting point in that record. This capability is applicable to all data sets supported by the program.

PROGRAM FUNCTIONS

The functions of the IEBDG program are performed by seven modules, which reside in the link library, SYS1.LINKLIB. At any given time, at least two modules (the control module and one or more other modules) will reside in the region assigned to the program. (If the region has enough space, all seven of the modules may be resident at the same time in the region.) The program contains a control module, a message module, two analysis modules, a table-building module, an output record generating module, and a cleanup module. Control passes within the modules of the program by means of the LINK macro instruction. If an exit to a user routine is specified, a branch-and-link procedure is used. Figure 65 on page 146 indicates how control is passed between the modules of the IEBDG program.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1



Note: EP = Entry Point

Figure 65 (Part 1 of 3). Information Summary and Overall Flow of Data Generation Program

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

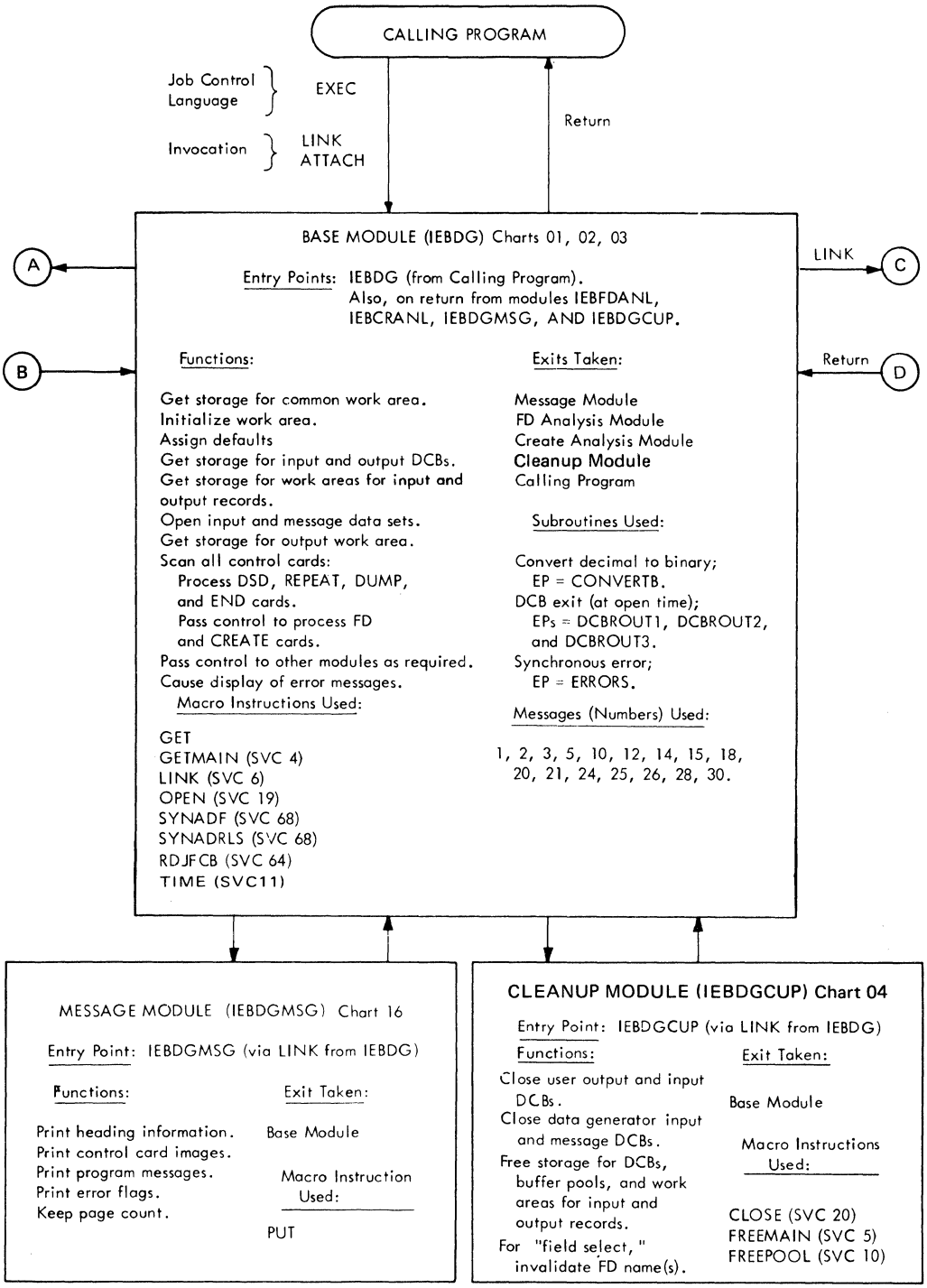


Figure 65 (Part 2 of 3). Information Summary and Overall Flow of Data Generation Program

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

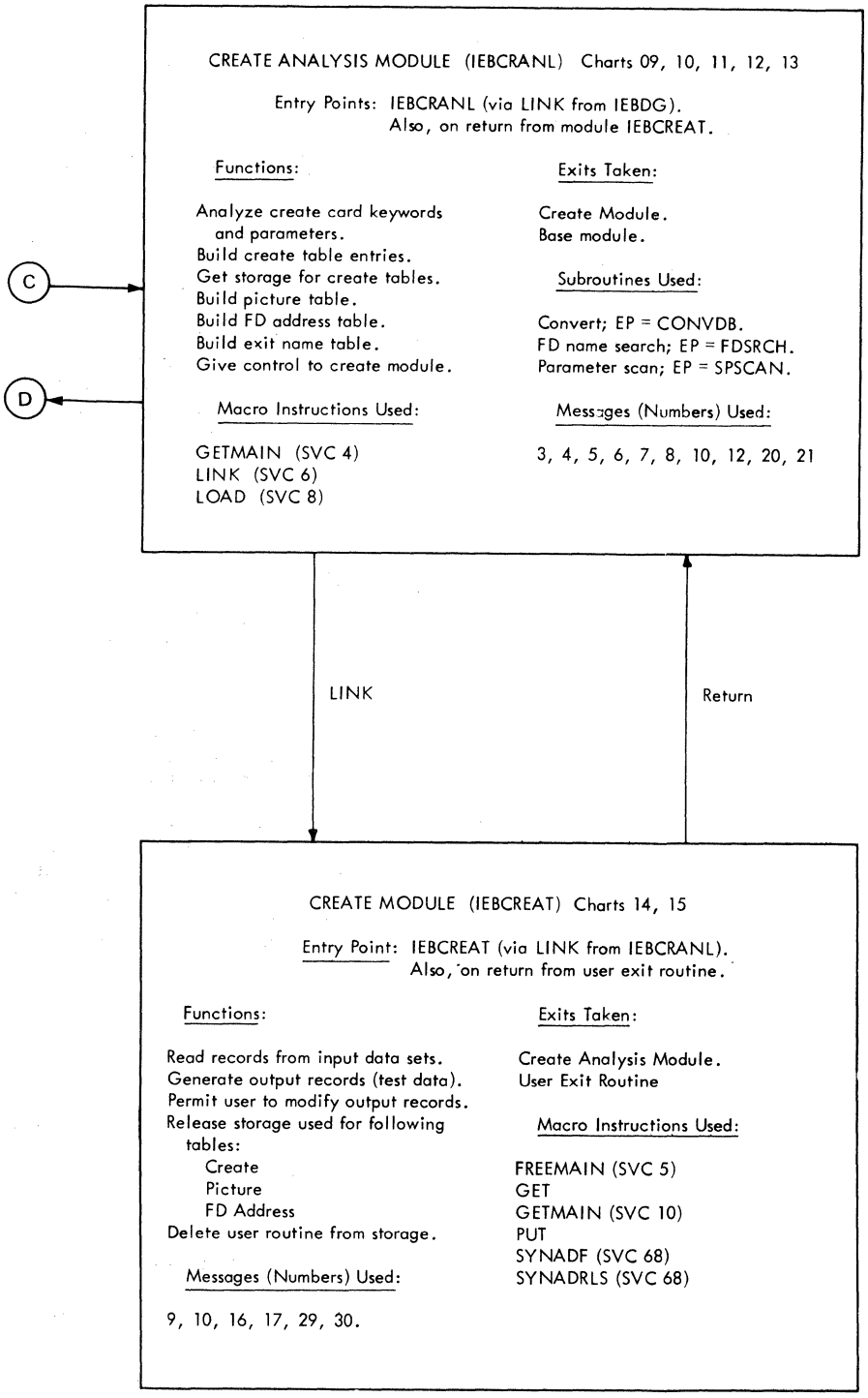


Figure 65 (Part 3 of 3). Information Summary and Overall Flow of Data Generation Program

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

The control (or base) module receives initial program control from the calling program and returns control to the calling program at the completion of the IEBDG program. This module scans the utility control cards for the function (for example, FD card analysis, REPEAT card analysis) to be performed and passes control to the appropriate module that performs the function.

The message module (IEBDGMSG) has the prime function of issuing images of the control cards, and of issuing messages required as a result of program operation. It receives control from, and passes control to, the base module.

The message module places information about the operation of the data generator on the system output (SYSOUT) device. This information includes processed control cards, heading and paging information, and normal completion messages. Error messages caused by abnormal conditions encountered by the data generator program also appear on the SYSOUT device. Incorrect control card parameters cause messages that will be printed immediately below the printout of the control card. Messages begin at print position one, and the printout of control cards starts at print position ten.

Both the create analysis (IEBCRANL) and the FD analysis (IEBFDANL) modules analyze the parameters found on one of the two field- or record-defining control cards. Using the parameters found, these modules construct tables for use by subsequent modules that may require the information in the tables. In the case of the FD analysis module, control is given to the FD table building module, IEBFDTBL, to complete the construction of the table.

The output record generating module, or create module, IEBCREAT, controls the generation of the test (output) data records for the user. This module also passes control to a user exit routine if output record modification is to be performed.

The cleanup module (IEBDGCUP) receives control from the base module to close the input and output data sets and to release the storage areas that were used by the program. If the field select feature is specified, the module invalidates the transfer of a given FD control card across DSD card groups by replacing an entry in the FD table built for the parameters on the FD card. This module returns control to the base module.

CONTROL CARD SCANNING

Whenever any control card scanning is to be done, all modules within the IEBDG program employ the same general scanning techniques. The information to be scanned is placed in an input work area to which a register points. Information within this work area is scanned 1 byte at a time as the scan method looks for a nonblank character in a given column. If a nonblank character is encountered in column one of a card image, a control card name has been found. This name is of no significance to the program, and it may be up to 8 bytes in length; but it must be followed by a blank column. The card type (DSD, FD, END, etc.) is then determined. If the type is not valid, the program is terminated.

Following the card type and blank column, the finding of a nonblank indicates the presence of a keyword. As the scan encounters a keyword, an attempt is made to match the keyword with valid keywords in the program. If a match is made, a branch is made to the appropriate routine to process the keyword. If no match is made, or if incorrect parameters are associated with a given valid keyword, an error is indicated and a message is printed. A comma or a blank signifies the end of a parameter. A continuation card is to follow when either a nonblank character in column 72 of a card or a comma followed by a blank column is encountered. The scan of a continuation card begins in column 4 of the card.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Except for the continuation of the scan of a PICTURE parameter, the first nonblank character in a continuation card indicates the presence of a keyword. In the case of the PICTURE parameter scan continuation, the characters (or blanks) in column 4 and any succeeding columns are recognized as belonging to the PICTURE parameter. This permits the presence of embedded blanks and delimiter characters in the PICTURE parameter.

THE BASE MODULE (IEBDG)

IEBDG (see "Chart 01. Base Module (Part 1 of 3)" on page 178, Chart 02, and Chart 03) is the first module of the data generator program to be placed in main storage. It is entered from a calling program and returns control to the calling program at the completion of the data generator program. Depending on the requirements encountered during the processing performed by this module, it will give control to (and receive control from) one of the following modules: the message module, the cleanup module, the FD analysis module, or the create analysis module. The primary functions of the base module are:

- To get storage for a work area (the common communication area)
- To open the input, output, and message data sets
- To read the utility control cards, analyze their contents, and determine if any error messages should be printed
- To pass control to the appropriate module as required

INITIALIZATION

Upon entry to this module, registers are saved for a later return to the caller. By use of an SVC 10 instruction, storage is obtained for a common communication area. This area is then given initial (default) values for ddnames, line count (for printer control), and page number. To provide for the specification of a random binary number format for the output data set, an initial multiplier value is established for a random number generator and placed in the communication area.

If a calling program has invoked the IEBDG program by means of a LINK or an ATTACH macro instruction, the previously assigned default values in the communication area are replaced by the values specified in the parameter list for the invocation. The assigned names of SYSIN, for the utility input control data set, and SYSPRINT, for the utility output control data set, may be changed as a result of an invocation. If so, the changed names are effective for the duration of the job. After invocation, the input and output control data sets are then opened.

OPENING DATA SETS

If the IEBDG program is called by use of the job control language statements, the input (SYSIN) and message (SYSPRINT) data sets are opened and default values assigned as required.

Each time a data set is opened, a DCB exit routine in the base module is entered. The entry points to this routine are determined by the function (input, output, SYSIN, or SYSOUT) of the data set being opened. At each entry, the routine establishes default values for the record format, the logical record length, and the block size for the data set.

A common section of the DCB exit routine is then entered to inspect the actual values of record format, logical record length, and block size. These values normally have already been placed in the respective fields of the DCB by the open routine.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

For data sets having a fixed record format, the common routine determines if the block size is an integral multiple of the logical record length. An integral multiple is required; otherwise, default values are assigned (if not previously assigned) so that an integral multiple is assured.

As the DCB exit routine evaluates the preceding record parameters for input or output data sets, it sets the FLUSHSW switch (at COMMON + 572)[7] to one if default values are assigned. (If the switch is set, then, when the base module again receives control, it flushes the control cards and proceeds to terminate the job.) The exit routine then returns control to the open routine to complete the opening of the data set.

In testing for a successful data set opening, only the input (SYSIN) data set is tested by the base module. Because a user may not desire any messages, or may not have enough space available for an output data set for messages, the testing for a successful opening of the output (SYSPRINT) data set is done by the message module when the module is first needed.

MESSAGES

When messages are required during the processing by the base module, a linkage is made to the message module. Upon return from the message module, processing will continue or, depending on the severity of the situation causing the message, a return is made to the calling program. (When any of the modules of the data generator program require the printing of an error message, control is returned from the module in command to the base module, which will then link to the message module. Depending on the severity of the error causing the message, control may or may not be returned from the base module to the module that was in command.) A condition code CONDCODE (at the field COMMON + 404) is set prior to giving control to the message module. Upon return from the message module, this code is checked to determine the severity of the situation. The base module returns control to the calling program by freeing the storage space for the common communication area, restoring the calling program's environment (registers), and issuing a BCR instruction.

After the input data set has been opened, a program heading message and an indication of any PARM field (on the EXEC card for the program) errors that may be present are placed on the SYSPRINT data set.

READING CONTROL CARDS

The GET macro instruction is used to place a control card in the input work area. The card image is printed on the SYSPRINT data set, and tests are made to determine the type of card in the input area. For either an FD control card or a CREATE control card, the base module will give control to the appropriate processing module.

For any new group of data generator control cards, the first nonblank card must be a DSD control card. If a blank card is present, it is merely flushed through and the next card is checked. In order to indicate when a DSD control card is detected, a switch, DSDSW (at COMMON + 550), is set to 1. This switch is tested for all but the DSD card in a group of control cards. If the first card in a group is not a DSD card, the syntax of the other control cards may be checked, but the

---

[7] In the discussion of the modules of the data generator program, references to locations in the common communication area are indicated by giving the decimal value of the displacement, or offset, from the start of the area. As an example, the offset of the field CONDCODE (condition code setting) would be given as COMMON + 404.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

program will not be executed. An error message will indicate the reason.

Following the test for a DSD card, the other utility control cards are checked for card types. The finding of a particular type causes the base module to give control to the proper module for processing of that control card. If a continuation card belonging to a given control card is encountered, the base module gives control to the appropriate control card processing module to scan the card. Should a DSD control card have no CREATE control cards between it and either an END card or a "/*" card, the resulting output data set that is created will be a null data set (that is, no picture or pattern will be produced).

As the base module continues its scan, a check is made for a blank following the card type (DSD, FD, etc.), as well as for improper control card names or name length. Errors in one of these areas will cause a message to be printed and the program will not be executed.

Included within the routines of the base module is a SYNAD routine for the SYSIN data set. The SYNAD routine obtains unrecoverable I/O error information that is to be printed on the SYSPRINT data set. (The message module contains a SYNAD routine for the SYSPRINT data set; the create module contains a SYNAD routine for input and output data sets.) After the information is printed, control is given to the cleanup module.

BASE MODULE CARD-PROCESSING

The following data generator control cards are processed by the base module: the DSD card, the REPEAT card, the END card, and the DUMP card.

DSD Card Processing: In obtaining storage for a user's DCB, the base module requests enough space (280 bytes) to contain a DCB for an indexed sequential (ISAM) data set. For a non-ISAM data set, much of this space is unused (see Figure 66).

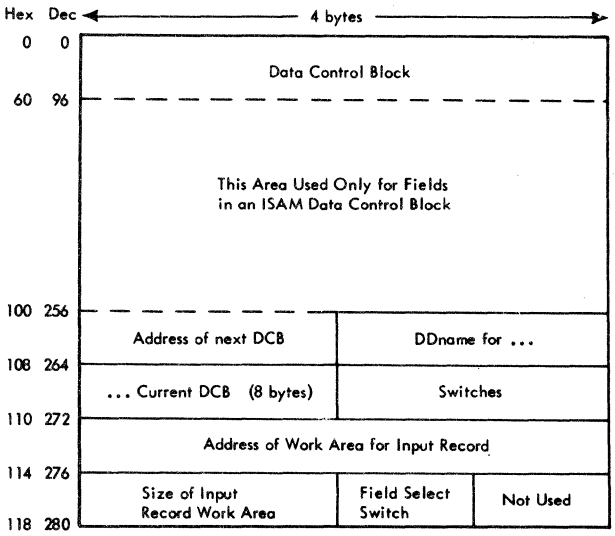


Figure 66. Storage Area Obtained by Base Module for Current DCB

A storage area is obtained as required for each of the data sets described by the ddnames on the DSD control card. In the case of storage for the last data set's DCB, the 4-byte field beginning at location 256 (X'100') is zero.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Before OPEN, the DSORG in the DCB is set to PS. An RDJFCB is issued to determine the actual organization of the data set. If the organization specified in the JFCB is indexed sequential, the DSORG is changed to IS before the data set is opened.

REPEAT Card Processing: When the base module scans the parameters of the REPEAT card, it sets an indicator, QUANSW (at COMMON + 576), to record the finding of the required keyword. After each valid keyword is found, the numeric value of its parameter is packed and converted to binary. Because 65 535 is the largest number that can be held in a 2-byte storage field, any parameter value that is greater than that results in a message to the programmer. Acceptable values for the QUANTITY and the CREATE parameters are stored for use by the create analysis module.

END Card Processing: When an END control card is encountered, the base module gives control directly to the cleanup module if all of the required number of entries specified on the REPEAT control card have been processed. Otherwise, a message is printed and then control is given to the cleanup module. Upon return from the cleanup module, the base module reads the next control card, which may be either a data generator control card or a "/*" delimiter card. There may be one or more additional groups of data generator control cards before a "/*" card.

DUMP Card Processing: The reading of a DUMP control card causes a printout of the user's program and/or storage areas assigned 0 to his program. When the DUMP control card is encountered, the base module places a zero in register 15 and forces an abend dump by branching to that register. Further description of the use of a DUMP control card is given in "Service Aids," later in this section.

THE CLEANUP MODULE (IEBDGCUP)

When either an END control card, indicating the end of a group of data generator control cards, or a "/*" delimiter card, indicating the end of a job, is encountered, the base module gives control to the cleanup module (see "Chart 04. Cleanup Module, IEBDGCUP" on page 181).

All user input and output data control blocks (DCBs) that have been opened are closed. For each of these DCBs, any buffer pools that data management routines had obtained for use by the data generator program, as well as the 280-byte storage area obtained by the base module, are released to the system.

When the field-select capability has been used, it is necessary to prevent possible errors that might arise from an inadvertent specification (on a CREATE control card) of an FD name associated with a closed input data set. Therefore, this module replaces the FDNAME field in the FD table with 8 bytes of hexadecimal 'FF's to prevent further access to the table. In order to use the selected data across DSD groups, a user must include a duplicate of the FD card in question in each DSD group in which the data is desired.

If the entry to this module was the result of encountering an END control card, the module returns control to the base module for the purpose of checking for another group of control cards.

If the entry to this module resulted from encountering a "/*" delimiter card, the module will close both the system input (SYSIN) and the system output (SYSPRINT) DCBs of the data generator program, and free any related buffer pools for these data sets.

The storage area that was obtained for the data generator program's input and output DCBs (96 bytes each) was initially obtained as a part of the common communication area by the base module. Therefore, the base module will release this area after it receives control from the cleanup module.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

THE FD ANALYSIS MODULE (IEBFDANL)

IEBFDANL (see Charts 05 and 06) scans and analyzes the parameters on the FD control card. It is initially entered from the base module. If IEBFDANL does not encounter a condition that causes termination of the job, it will use the FD table module (described later) as a subroutine. After the FD table module returns control to the FD analysis module, the latter module returns control to the base module.

The FD analysis module begins the assignment of information to a table called the FD table. This table is used by both the create analysis module and the create module. The FD table module completes the construction of the table.

An FD table entry has 64 bytes. Storage for the FD table is obtained in increments of 512 bytes (enough for 8 table entries) by the FD analysis module. Each entry contains most of the parameter information (or a processed version of the information) from one FD control card. If a PICTURE keyword has been specified on the FD control card, the picture information is placed in another area of main storage. The FD table is shown in Figure 67 on page 155.

Upon entry to the FD analysis module, tests are made to determine whether or not the entry is caused by a continuation card. Such an entry may be caused by the continuation of the parameter string on a card, or by the continuation of the PICTURE parameter on a card. If the entry is caused by either a continuation card or a picture continuation, storage for an FD table entry may already be available as the result of processing a previous FD control card in the same set of data generator control cards. If the entry is not caused by a continuation card, an FD table entry is to be constructed. A GETMAIN macro instruction is issued to obtain storage for an FD table.

FD CARD SCANNING

The scan of the actual FD card keywords and their associated parameters is then performed. As each keyword is encountered, its parameter is scanned, validated, and/or converted if required, and then placed in a reserved spot in the FD table. If a keyword error or a parameter error is encountered, an appropriate message will be printed on the system output device. The severity of the error determines whether the program is terminated at that point or whether modified processing (for example, syntax checking only) will continue. Control and storage tables are constructed even for syntax checking procedures.

A user may specify either the FORMAT or the PICTURE keyword, but not both, on the same control card. The FD analysis module sets a switch, either FDFMTSW (at COMMON 539) or FDPLSW (at COMMON + 540), when it encounters one of these keywords. If the other keyword is then encountered in the scan of the same card, a test of the previously mentioned switch for the keyword first encountered reveals the error.

A user specifies the field-select capability by including the INPUT= keyword and parameter either with or without the FROMLOC= keyword and parameter on a control card. If the INPUT= keyword's parameter does not match a name that has been specified on a DSD card, or is not "SYSIN", a message is issued and the program will be terminated. If a match is found, the module sets the FIELDSEL switch (in the DCB) to X'FF'.

For later use by the create module, the FD analysis module initializes the address (at location FDFROMAD in the FD table) from which the input record will be selected. The FD analysis module uses the FROMLOC keyword's parameter if it has been specified. If the FROMLOC keyword and/or its parameter has been omitted from the FD control card, the module sets the FDFROMAD field value to the start of the input record (which is either at

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

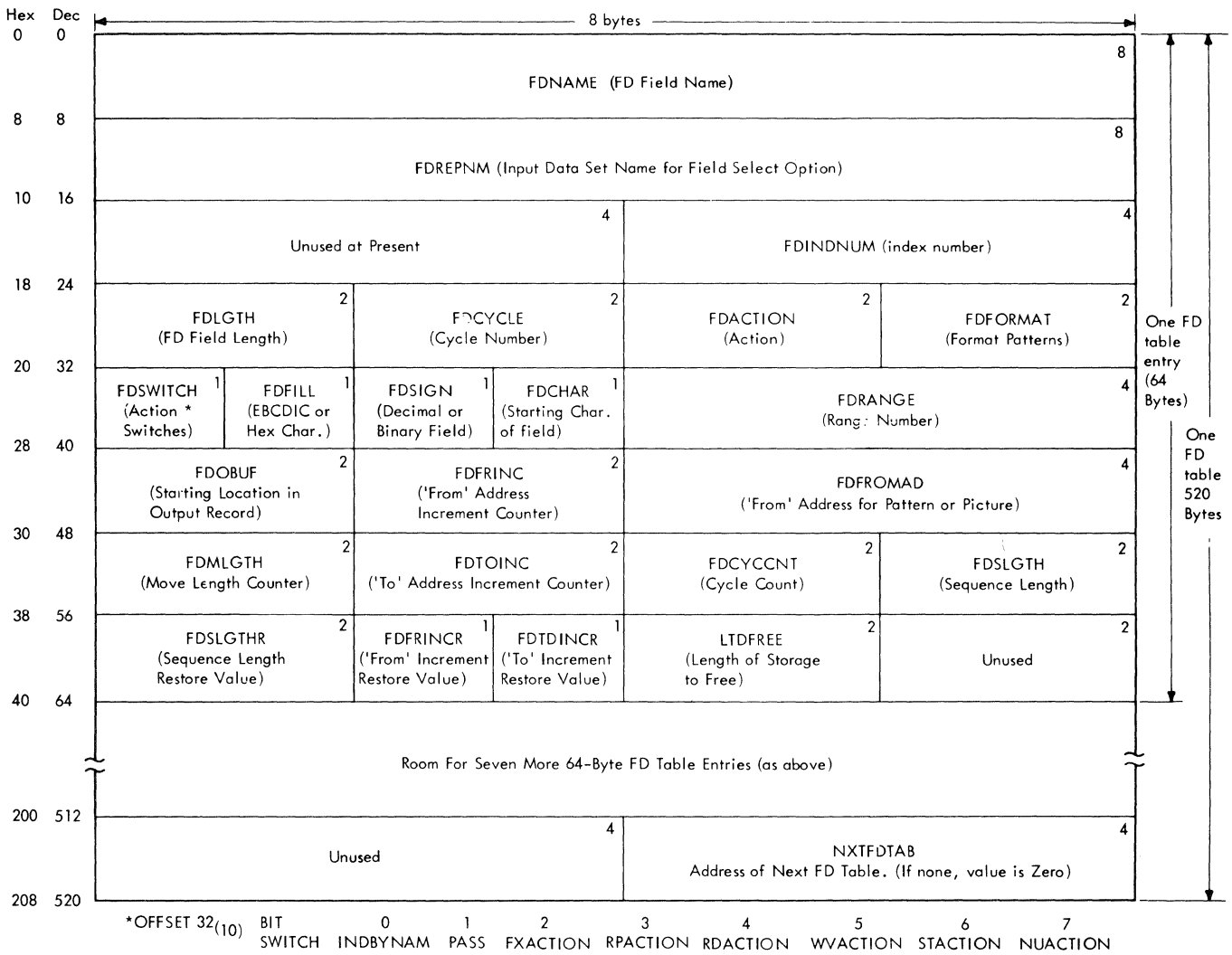


Figure 67. FD Table Constructed by FD Analysis Module and FD Table Module

location INBUFA if the input name is SYSIN or at location INREC if the input name is other than SYSIN).

Figure 68 on page 156 lists the keywords of the FD control card and indicates the processing done on the parameters of the keywords by the FD analysis module.

After the keyword parameters on the FD control card have been scanned and placed in the FD table, the FD analysis module gives control to the FD table module to complete the construction of the FD table entry. The FD analysis module assigns only the initially specified values of parameters to the FD table. If any keyword except LENGTH and NAME was omitted from the FD control card, the FD analysis module does not perform processing for the keyword and does not fill in the appropriate space in the FD table entry. Default values, if any, for keyword parameters are assigned by either the FD analysis module or the FD table module.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

FD Keyword	Processing Applied to Keyword Parameter			Default Value (if any)**
	Validity Check	Value Converted	Other Processing	
NAME*	Yes	No	Length checked for maximum of eight characters.	None.
LENGTH*	Yes	Yes	None	None.
INPUT	Yes	No	Check for value "SYSIN" or for matching name on DSD card. Set action switch for "fixed."	None.
FROMLOC	Yes	Yes	Value is temporarily placed at FDM LGTH in FD table. It is used in determining the "from" address for field-select option.	Start of record.
STARTLOC	Yes	Yes	Subtract one from value.	First available byte in record.
PICTURE (length)	Yes	Yes	Check for occurrence of FORMAT keyword.	None.
PICTURE (field)	No	No	Get storage for picture. Determine type of picture. Move picture to storage. (Include continuation cards.)	Fill character.
FORMAT	No	No	Check for occurrence of PICTURE keyword. Check for two-character pattern.	Fill character.
ACTION	No	No	Check for two-character type.	FX (Fixed).
FILL	No	Yes (Hex only)	Check for EBCDIC or Hex type with two digits.	Binary zeros.
CYCLE	Yes	Yes	None.	One.
RANGE	No	No	None.	None.
CHARACTER (of FORMAT)	No	No	None.	A (for alphabetic and alphameric). "Blank" (for collating).
SIGN	No	No	Determine if sign value is valid.	Plus.
INDEX	Yes	Yes	None.	None.

* These keywords are required to be present in the FD control card. If not present, the program will be terminated.
** Default values, except for FROMLOC, are assigned by the FD table module.

Figure 68. FD Control Card Keyword Parameter Processing, and Default Values Assigned, If Required

THE FD TABLE MODULE (IEBFDTBL)

IEBFDTBL (see Charts 07 and 08) completes the construction of the FD table, which was begun by the FD analysis module. At the time of entry into this module, an FD card has been completely scanned and initial values from the card have been placed in the FD table. The FD analysis module uses a LINK macro instruction to give control to the FD table module, which is then used as a subroutine by the FD analysis module. The FD table module returns control to the FD analysis module.

FD PATTERN CONSTRUCTION

Initially, module IEBFDTBL determines the type of picture or format specified in an FD control card. (This field will be used by the create module when it constructs the output records.) If neither a picture nor a format is specified, the FD table module assigns a default value to the field.

Before further processing is done on a non-EBCDIC picture, the picture numbers are checked for validity by comparing the zone bits of the numbers against X'F'. An incorrect value results in an error message indication, and control is returned to the calling module. (A picture having a packed decimal specification must have a length specification that is less than or equal to 16, because the Pack instruction can handle up to 16 bytes.) Otherwise, the numbers are converted to the specified

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

form, storage is obtained for the picture, and the picture is moved into the storage area (from the temporary storage area into which the picture had been placed by the FD analysis module). The temporary storage area is then released and the FD table module gives control back to the FD analysis module.

Except for the NAME, LENGTH, INPUT, and FROMLOC keywords, the FD table module assigns default parameter values for each keyword that is omitted from an FD control card. The values assigned are shown in Figure 68 on page 156, and they are placed in the FD table.

For a pattern, which may be either a user EBCDIC picture specification or an IBM format specification, module IEBFDTBL determines the action that is specified on the FD control card. Based on this determination (including a possible default determination), the module makes an entry in the FDACTION field of the appropriate FD table and sets the appropriate bit in the FDSWITCH field of the table to one.

The module then determines the amount of storage required to hold the pattern. The amount of storage required depends on the action the create module will later apply to the pattern. By means of the GETMAIN macro instruction, the FD table module obtains the necessary storage. To provide for a "wave" or a "ripple" type of action, the storage area must contain two contiguous copies of the pattern. If the action is a "roll," three contiguous copies of the pattern must fit in the storage area. The create module requires the repeated patterns when it generates the output records.

After storage is obtained to accommodate the desired pattern action, the module places the specified fill character or a default fill character in each byte of the area. It then moves the pattern into the storage area the required number of times. Any leftover space (caused by differences in field length and picture length specifications) contains the fill character.

If a PICTURE keyword had been specified, the temporary storage area that the FD analysis module had used to hold the picture is released before the FD table module returns control to the FD analysis module. If a FORMAT keyword had been specified, the starting character for an alphabetic, alphameric, or a collating sequence field is resolved before control is returned to the FD analysis module. For other formats, the storage field is initialized to a value that depends on the format specified. (For binary format, the value is a binary 1; for packed decimal, the value is a packed decimal 1; for zoned decimal, the value is a zoned decimal 1.)

For the FD table module, a 63-byte sequence of characters resides in storage at location COPAT. The 28th byte of this sequence is at location ALPAT. After resolving the starting character for an AL, AN, or CO format, the module fills the pattern field, using the characters of this sequence. If the starting location value is a default value, a collating sequence pattern begins at location COPAT, and an AL or AN pattern begins at location ALPAT.

The pattern field is filled only in increments that are equal to or less than the length of the sequence that is being used for the format pattern. If the length of the field (given at decimal offset 24 in the FD table) to be moved is less than the indicated sequence length, the number of characters moved will be equal to the FDLGTH field value. If the length of the field to be moved is greater than, but not equal to, an integral multiple of the indicated sequence length, the number of characters moved for all moves but the last will be equal to the sequence length. The last move will contain the characters remaining after an integral number of moves have been made, each move containing the number of characters in the given sequence. If the FDLGTH value is equal to an integral multiple of the sequence length, the number of moves is equal to the integral number.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

THE CREATE ANALYSIS MODULE (IEBCRANL)

IEBCRANL (see Charts 09 through 13) scans and analyzes the parameters on the CREATE control card. The initial entry to IEBCRANL is from the base module when a CREATE card is encountered. Other entries to the module occur when create continuation cards or create card comments cards are encountered. If the create analysis module does not encounter a condition that suppresses the creation of output records, it will use the create module as a subroutine to generate output records. The create module will return control to the create analysis module, which will, in turn, return control to the base module.

TABLE CONSTRUCTION

The create analysis module constructs four types of tables that are used by the create module:

- The create table
- The picture table
- The FD address table
- The exit name table

The create table is the largest of the four. It may contain one or more create entries. Module IEBCRANL establishes a 28-byte create entry for each CREATE control card that it processes (see Figure 69 on page 159). One create table may contain up to 18 create entries. These entries contain pointers to picture tables and FD address tables. More than one create table may be constructed.

The picture table contains information about the picture string that may be specified on a CREATE card, as well as the actual picture string itself.

The FD address table contains the addresses of the FD table that have been constructed to contain information from FD cards.

The exit name table contains the names of any user exit routines that have been specified. When a user's exit routine is loaded into main storage, the storage address of the routine is placed in the create table.

MODULE ENTRIES

Because the create analysis module may have already been entered in processing a given group of data generator control cards, the initial analysis performed upon entry to the module consists of determining the cause for the module's receiving control. The create-continue switch, CRCSW (at COMMON + 564), is tested for this purpose. If the entry to the module is the first one for a given CREATE control card, storage for a create entry is obtained either from an existing create table or by the issuance of a GETMAIN macro instruction for space for another 512-byte create table. (As each new create table space is obtained, it is "chained" to the previous space and initialized to all zeros.) Then the module scans the control card keywords one at a time. If an invalid keyword is encountered, the create analysis module indicates a message, sets the NOGOSW switch (at COMMON + 551) to suppress the creation of output records, and gives control back to the base module to continue the checking of syntax on other control cards.

If the entry to the create analysis module is because of a continuation of a CREATE control card, a check is made to determine if the parameters of either the NAME keyword or the PICTURE keyword were interrupted. (Except for the NAME and PICTURE keyword parameters of the CREATE card, all other CREATE

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

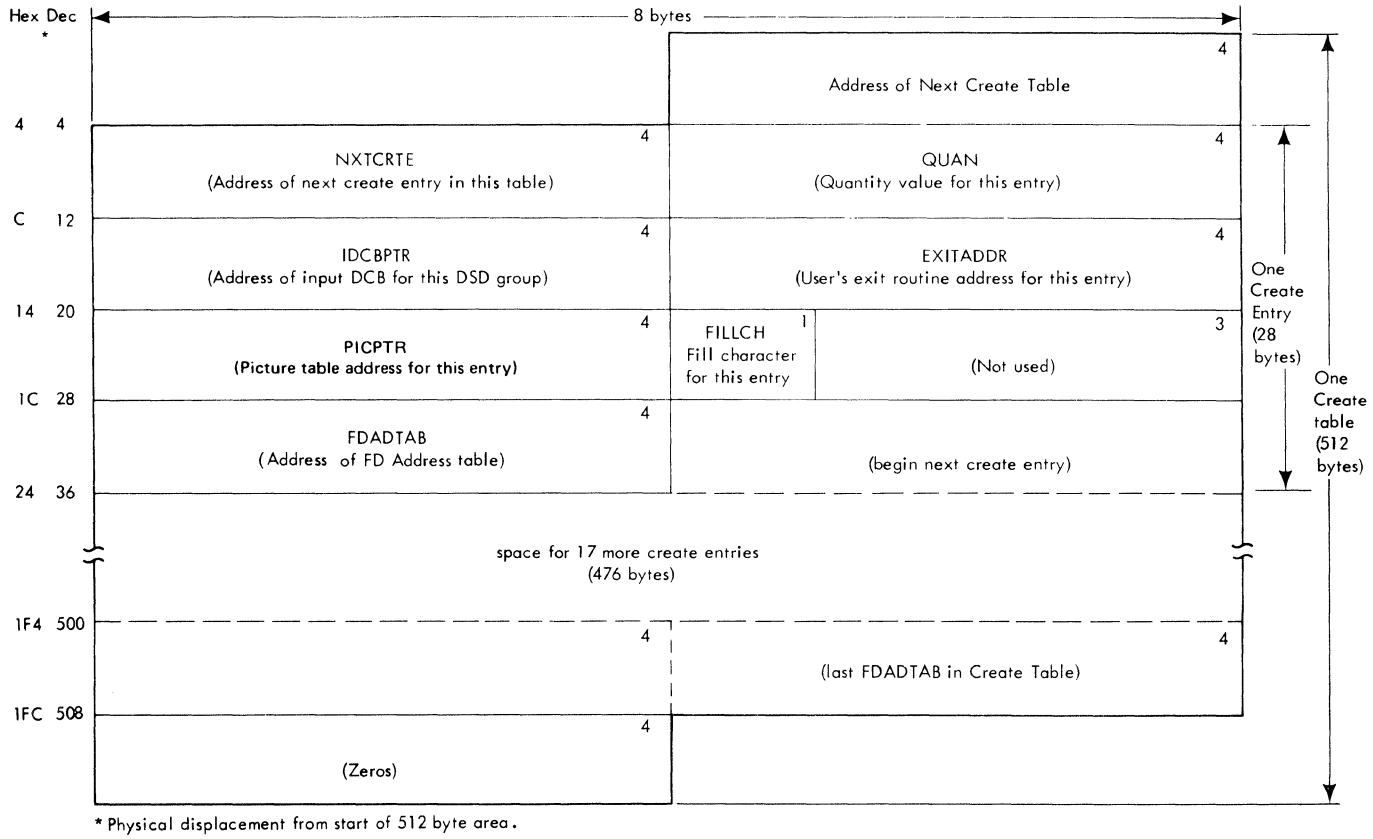


Figure 69. Create Table Constructed by Create Analysis Module

card parameters must be on the same card as their associated keywords. The picture string parameter of the PICTURE keyword is the only one that may in itself be continued to another card.) The name-continue switches, NAMCSW1 and NAMCSW2 (at COMMON + 561), and the picture-continue switches, PICCSW1, PICCSW2, and PICCSW3 (at COMMON + 562), are tested to determine if a parameter may be on a continuation card. For a continuation card, the scan begins in column 4.

If the entry to the create analysis module is because of a comments continuation, control is given to the base module, IEBDG, to process the comments and to read the next control card.

MODULE SUBROUTINES

As the create analysis module processes each valid keyword, it may branch to subroutines within the module. These subroutines perform functions of parameter scanning (SPSCAN routine), packing and/or conversion (CONVDB routine), and table searching (FDSRCH routine). When the processing for a given keyword is completed, the keyword scanning section of the module scans the next keyword unless a continuation card has been indicated. If the latter action has occurred, module IEBCRANL gives control to the base module to read the next control card.

The SPSCAN Subroutine: The function of the SPSCAN subroutine is to check the validity of a keyword parameter. The routine moves a pointer across each character in the parameter and checks for

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

commas, blanks, and parentheses. It also determines whether a parameter has been extended into column 72 of a control card. If so, an error message is indicated and control is given back to the base module.

After a comma or a blank is encountered, the parameter length is determined and checked for an invalid length of zero, and the routine returns control to the calling section of the module. An invalid parameter length causes the module to indicate a message and return control to the base module.

The CONVDB Subroutine: The convert subroutine, CONVDB, performs two functions: It converts a decimal number to a packed decimal number, and it converts a decimal number to a binary number. The subroutine is used in processing the parameters of the QUANTITY, the NAME, and the PICTURE keywords. The convert subroutine processes a number that can be contained in 4 bytes or fewer. Therefore, a decimal value that is greater than 2 147 483 647 will not be processed, and a message will be issued. Module IEBCRANL then returns control to the base module.

In order to determine if a parameter to be converted is numeric, the zones of its characters are checked for X'F'. Valid numeric characters are then converted to packed decimal format and placed in the storage area Q (at COMMON + 216). For all cases except a packed decimal picture specification, the parameter value is then converted to a binary value and placed in a general register.

The FDSRCH Subroutine: Module IEBCRANL contains and uses the FDSRCH subroutine in processing the NAME keyword parameter. The subroutine places a valid name from the CREATE control card into storage and then compares this name with the names of the FD tables (which have been established by the FD analysis module). If the list of FD tables does not contain a name that matches the name on the CREATE card, a message is indicated and the create analysis module returns control to the base module.

When an FD table name that matches a CREATE card name parameter is found, the address of the FD table bearing that name is placed in an FD address table (see Figure 70 on page 161). If there is no room in an existing FD address table, the FDSRCH subroutine will obtain storage for a new table. The current create entry, whose address is given at CURCRTE (COMMON + 316), contains the address of the first FD address table. All FD address tables are chained together by pointers in the tables themselves.

KEYWORD PROCESSING

To determine if all keywords on a given CREATE card have been processed, module IEBCRANL tests the column after the last parameter of each keyword. If this column is blank and if column 72 of the control card is blank, the last parameter on the card has been processed. The module then establishes a default value for the QUANTITY keyword parameter if a value has not already been supplied on a CREATE control card.

If there are no more continuation cards for a given CREATE control card and if the create value (from a preceding REPEAT control card group) is 1, module IEBCRANL gives control to the create module, IEBCREAT. Otherwise, if the create value (determined by testing the field CREATENO at COMMON + 18) is greater than 1, the create analysis module gives control to the base module to read the next CREATE control card. (The field CREATENO is initially set to 1 in case a CREATE card is not part of a REPEAT card group.)

If the column after the last parameter contains a comma, the next card column is checked. If this next column either is column 72 or contains a blank, the module gives control to the base module to read a continuation card. Otherwise, either a

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

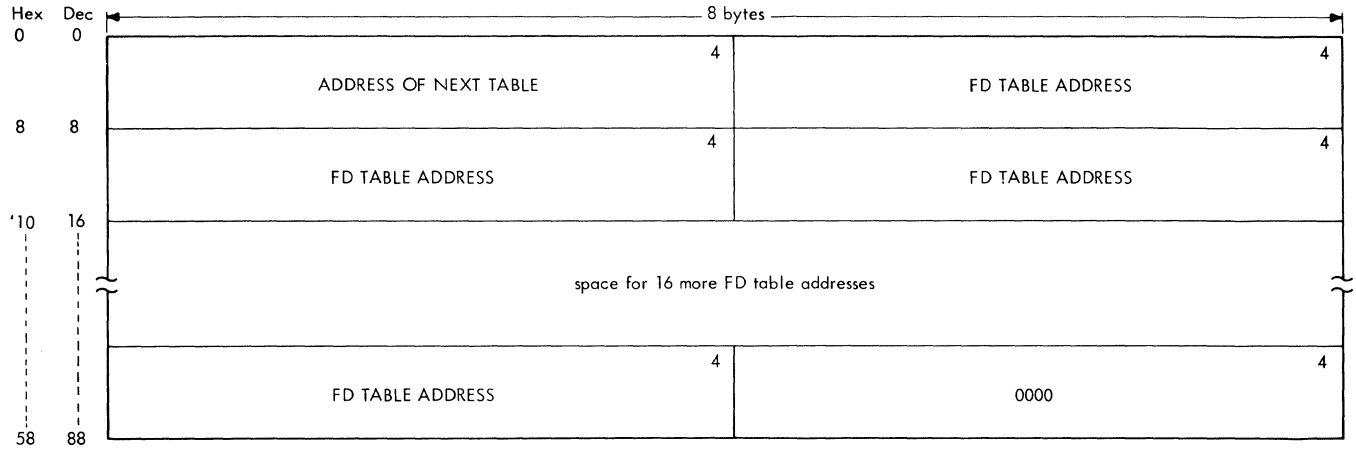


Figure 70. FD Address Table Constructed by Create Analysis Module

message would have been indicated and control returned to the base module or the subroutine for scanning the next keyword will be entered.

When the create analysis module processes the parameter of the DDNAME keyword, it tests the number of characters in the DDNAME and determines whether the parameter value is SYSIN. If the name length is valid and the name is SYSIN, the address of the SYSIN data control block (at COMMON + 116) is placed in the create entry (whose address is given at COMMON + 316) for the CREATE card being processed. If the name is not SYSIN, the input DCBs are scanned to find a name equal to the CREATE card's ddname. In doing the scanning, the address of the first input DCB is placed in the DCB pointer (at COMMON + 300). The name of the DCB (at DCBD + 260) is compared with the ddname given on the CREATE card. Unless an equal name is found, the process repeats with the next input DCB until there are no more input DCBs to check. If a successful DCB name comparison is made, the input DCB address is placed in the create entry. Otherwise, a message is issued and the create analysis module gives control to the base module.

If the ddname search was successful, either the given delimiter or a default delimiter is placed in the DELIM field (at COMMON + 344), or a message signifying an invalid delimiter is issued and the create analysis module returns control to the base module.

When the EXIT keyword is encountered, the length of the user's exit routine name is checked for validity. If the length is valid, the name is placed in a table called the exit name table (see Figure 71 on page 162). The user's routine is then placed (via a LOAD macro instruction) into main storage, and the storage address of the routine is placed in the create entry.

The NAME Keyword: In processing the parameters of the NAME keyword, module IEBCRANL searches for multiple names, for "copy" groups (based on the COPY keyword), and for breaks or interruptions in series of names within outer and inner parentheses. If the COPY keyword is not present and if multiple names have been indicated (by encountering a left parenthesis in the scan), a default value of 1 is assigned to the COPYVAL field (COMMON + 640). The complete processing of the NAME keyword parameters includes the use of the subroutines FDSRCH, SPSCAN, and, if the COPY keyword is present, the CONVDB subroutine. If multiple names are present, the SPSCAN and FDSRCH subroutines are used for each name that is encountered. The create analysis module indicates that there is a continuation in the parameters

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

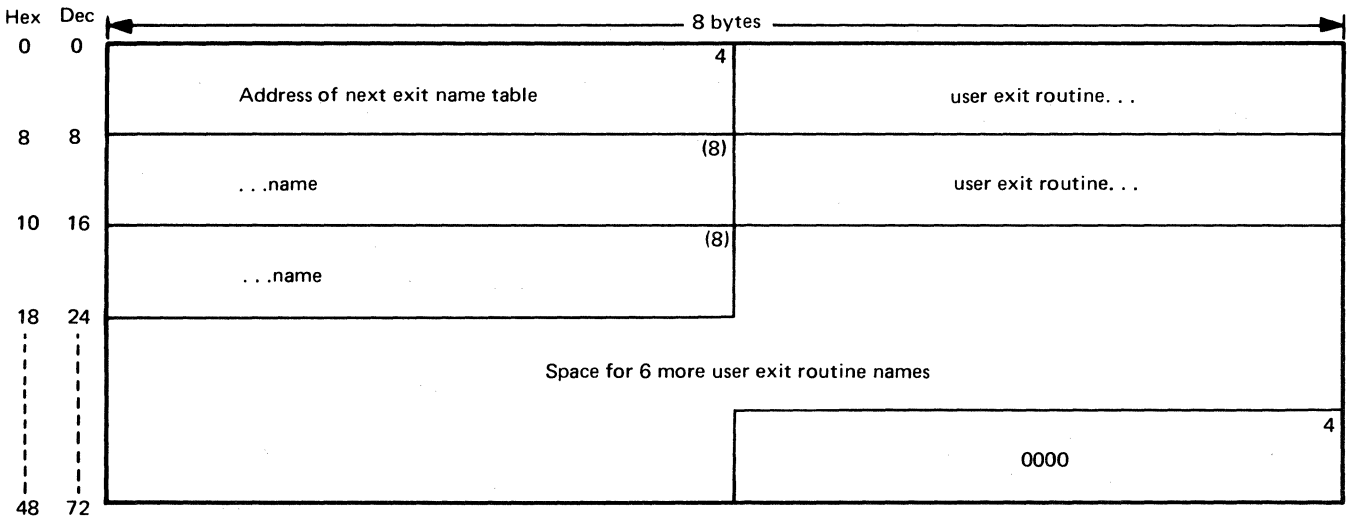


Figure 71. User Exit Name Table Constructed by Create Analysis Module

of the NAME keyword by setting switches in the NAMCSW field (COMMON + 561) of the communication area. If the continuation occurs after a "name" subparameter within only the outer set of parentheses, the high-order bit (bit 0) of the NAMCSW field is set to 1. For a continuation indication that occurs after either the COPY keyword or a "name" subparameter within the inner set of parentheses, the second bit (bit 1) of the NAMCSW field is set to 1.

If an inner group of names is to be copied more than once, the create analysis module checks the current FD address table for enough space each time a name is to be copied. If space is not available, storage for a new 88-byte FD address table is obtained, and the new table is chained to the previous one by the first word in the current FD address table located at the address given in the CURFDGM field (COMMON + 632).

The Picture Keyword: The PICTURE length parameter is processed first; the start location of the picture string is then processed; and the actual picture processing is done last. In the case of the PICTURE keyword parameters, there are three ways in which a continuation card may be encountered.

- The PICTURE parameter list may be interrupted after the length parameter. In this case, the first (high-order) bit of the PICCSW field (COMMON + 562) is set to 1.
- The PICTURE parameter list may be interrupted after the STARTLOC parameter. In this case, the second bit of the PICCSW field is set to 1.
- The actual picture string may be interrupted. In this case, the third bit of the PICCSW field is set to 1.

"Chart 11. Create Analysis Module, IEBCRANL (Part 3 of 5)" on page 188 indicates the entry points to the section of the module that handles the processing of the PICTURE keyword parameter.

In processing the length parameter, the module first scans the length and then converts the length value to a binary equivalent. Based on this length, the module obtains a storage area called the picture table (see Figure 72 on page 163). The binary length value is then placed in the field PICLGTH, which begins at the fifth byte of the associated picture table. The picture table is located at an address given in the PICBASE

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

field (at COMMON + 664) of the communication area. This same picture table address is also placed in the create entry for the current CREATE card.

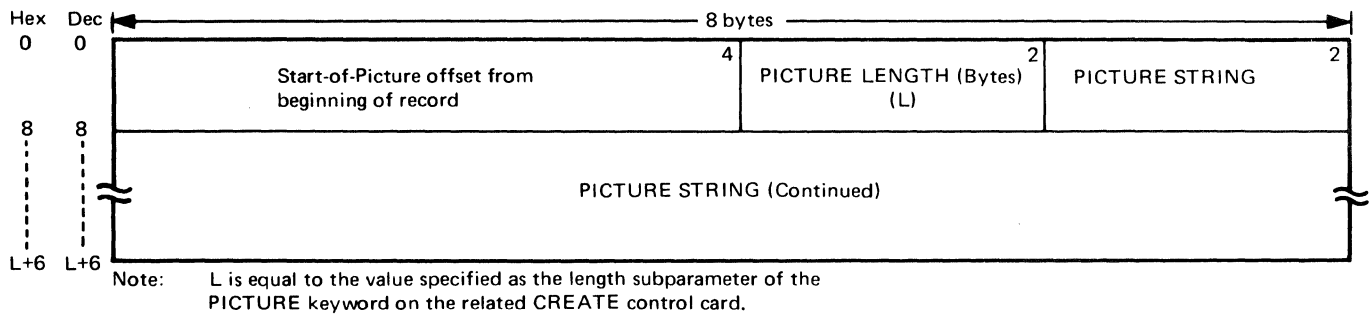


Figure 72. Picture Table Constructed by Create Analysis Module

The start location (STARTLOC) parameter is scanned; if valid, it is converted to a binary value; and the value is then placed in the associated create entry. The module then checks to see if the next parameter is on the same or a continuation card. If a continuation card is indicated, the module returns control to the base module to read the next card. Otherwise, the picture string will be processed.

If the picture string is specified as EBCDIC (character), the string characters are moved directly from the card to the picture table. If the picture string is to be continued, the continuation switch (second bit of the field PICCSW) is set to 1, and control is returned to the base module to read the next card.

If the picture string is specified as either packed decimal or binary, the complete string must be on one card. The card format is checked, and, if valid, the string value is converted either to a packed decimal value or to a binary value, as specified on the control card. The converted value is then placed in the picture table for use by the create module.

After each parameter on the CREATE control card has been processed, the create analysis module checks for a valid delimiting character and for an indication of a continuation card.

THE CREATE MODULE (IEBCREAT)

IEBCREAT (see Charts 14 and 15) maintains control over the generation of output records. This module receives control from the create analysis module after create entries (in one or more create tables) and related tables have been constructed. If, upon entry to this module, the switch, NOGOSW (at COMMON + 551), is on as the result of the action of a previous module, the generation of output records will be suppressed and the create module will perform only its cleanup functions. IEBCREAT always returns control to the create analysis module, which then returns control to the base module for the printing of messages and/or the reading of the next control card.

OUTPUT RECORD MODIFICATIONS

Upon entry to the create module, the NOGOSW switch is tested to determine whether to continue processing or whether to immediately release storage areas and return control to the create analysis module. If processing is to continue, the

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

record characteristics (length, format) are determined; a counter, RECREM (at COMMON 348), is initialized with the quantity value from the create entry; and the input record size (if present) is compared with the output record size. The output record field is then filled with the create entry fill character prior to reading in a record from an input DCB.

FD Table Modification: If there is no input DCB, modifications based on values in the FD tables are made directly to the output area containing the fill character. Otherwise, the modifications are made to the input record that has overlaid the fill characters in the output area.

The modifications based on the FD table values involve the action, index, cycle, and range parameters from the FD control card. Initially, an FD pattern at the storage address given in the field FDFROMAD of an FD table is moved to the output area, which, at this point, contains either a fill character or an input record. The create module then inspects other FD tables that may have been indicated by the CREATE control card as belonging to the current create entry and moves the patterns from these tables into the output area. The output record starting location for each FD pattern is given in the field FDOBUF of the FD table. Note that, as each modification is made to the output record, it may overlap part or all of a previous modification depending on the starting location specifications involved.

PICTURE and User Modifications: After the create module moves the FD patterns to the output record area, the module moves in the picture string from the CREATE control card, if the PICTURE parameter has been specified. Otherwise, or after the picture string has been moved, module IEBCREAT determines if a user exit routine is present. If so, it indicates that a user may want to inspect and/or modify the record before it is placed on the output device. An exit is taken to permit user modification if this is the case. After the user routine (if one is used) gives control back to the create module, the create module checks the return code that has been placed in register 15 by the user's routine and moves the code to register 6. If the job is not to be terminated at this point, the create module places in the output data set the record that is in the output area. If termination is to take place, an indicating switch (FLUSHSW or FLUSHSW1, depending on the return code) is set, storage areas are released, and control is given to the create analysis module, which then gives control to the base module.

UPDATING THE FD TABLE

After each record is placed in the output data set, certain values in each applicable FD table are updated to prepare for the next output record that is to be created. Multiple references, by the same create entry, to the same FD table are indicated by the setting of a "pass" switch (bit 1 of the FDSWITCH field). If this occurs, the FD table is processed (and updated) only once.

For the binary, packed decimal, and zoned decimal patterns, the create module performs the following actions by using FD table values:

- The cycle count field (FDCYCCNT) value is increased if the cycle value (FDCYCLE) is other than zero.
- The pattern values are then converted to a binary equivalent, if not already binary, and placed in a work area (register 4). The module then increases the binary-equivalent pattern value by using the index number (FDINDNUM).
- The increased pattern value is then tested against the range value given in the field FDRANGE. If the range value has been exceeded, the current pattern value in the storage area

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

to which the FD table refers is not changed. Otherwise, the indexed binary value is reconverted to decimal form if necessary and placed in the storage area.

For a random number format, the random generator routine of the create module provides another value to be used for the next record and places the value in the pattern storage area to which the FD table refers.

For alphabetic or character patterns, the generation of the pattern to be placed in the output area for the next record requires that values of the "from" address in the FD table and the "to" (or output work area) address be changed. These addresses are used in moving the pattern (or a part of it) from the pattern storage area to the output record area. In the FD table, there are two fields (FDFRINC and FDTOINC) that contain the increment values used to modify the "from" and the "to" addresses. These fields initially contain a value of zero for the first output record. For subsequent records, the values may be increased by values given in the increment restore fields (FDFRINCR and FDTOINCR) of the FD table.

The FD table module establishes the values of the increment-restore fields after the specified action has been determined. Figure 73 lists the values of the increment-restore fields for the various actions that may be specified.

Pattern	FDFRINCR ("From" Increment Restore)	FDTOINCR ("To" Increment Restore)
Shift Left	1	0
Shift Right	0	1
Truncate Left	1	1
Truncate Right	0	0
Roll	(*) 1 g1	0
Ripple	1	0
Wave	1	0
Fixed	0	0

* This value will alternate between 1 and g1 as the roll pattern is developed in first one direction and then the other.

Figure 73. Values of Increment-Restore Fields in the FD Table

Depending on the action specified in the FD table, the create module may vary the values of the move length counter field, FDMLGTH, and the sequence length counter field, FDSLGTH, to prepare for the next output record. Figure 74 on page 167 summarizes the changes that may occur to values of fields in the FD table as the create module generates output records.

After all FD tables to which a given create entry refer have been updated, the create module inspects the NXTCRTE field (in the create table) to determine the address, if any, of the next create entry to be processed. (When there are no more create entries to be processed, the NXTCRTE field of the current create entry contains zeros.) If there is another to be processed, the create module will process the entry in the manner already described.

If there is a repeat function to perform, the entire list of create entries must be processed as many times as necessary to satisfy the repeat requirement. When that is done, the cleanup

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

functions of the create module will be performed. If there is no REPEAT card function to perform for the current set of data generator control cards, the field REPEATNO (at COMMON + 16) contains zeros.

After the last create entry has been processed, the create module will release the storage areas that have been obtained for create tables, FD address tables, and the CREATE card picture string. The module then resets switches and communication area field values for an initial entry to the create analysis module, and returns control to the create analysis module.

THE MESSAGE MODULE (IEBDGMSG)

IEBDGMSG (see Chart 16) is entered from the base module whenever there is an indication of a message to be printed or to be placed on the output device. To indicate the need for a message, the other modules of the data generator program set a message number in the MS field (at COMMON + 406) of the communication area.

This module places four types of messages on the output device: heading messages, control card images, error messages, and error flags (messages). The messages used for headings and errors exist as 121-byte entries within the message module. The location of each message within the module is contained in a 4-byte address entry in a message pointer table.

Before any messages are placed on the output device, module IEBDGMSG determines if the output data set has been successfully opened. If it has not been opened, control is returned to the base module and the job is terminated. If the data set is open, the value in the MS field is checked to determine the reason for requesting the module.

Initially, the module is requested to print a heading message (MS field value = 1). Thereafter, a heading message is printed when the module finds an indication of either a channel 12 printer carriage tape or the correct line count. All heading messages begin at position 1 on the output device. After each heading message printout, the line count value is reset to either a user-specified value or the default value, and the page number value is increased. Before printing a heading message, a printer will skip to channel 1 to set up a new page. When any other message is to be printed, a printer will space one line before printing the message.

If the MS field value is not 1, the module determines if the carriage control tape indication is 12 and, if the indication is not 12, whether the line count value has reached its maximum value. If either situation has occurred, a heading message will be printed, the line count will be reset, and the page number will be increased.

Otherwise, the module tests the MS field to determine if a control card image is to be printed from the input buffer. If this is the case, the image is printed at position 10 on the output device. If a card image is not printed, the module tests for a regular error message indication from the processing modules. These messages have message numbers from 2 to 28. For each message to be printed, the printer is placed at position 1 to receive the message.

The message module places a flag message (consisting of the word ERROR) in the message data set when one of the other modules of the data generator program requests an error flag. This message begins on the line below the corresponding control card image and in the column corresponding to the card column that is in error.

After a message has been placed on the output device, the message module increases the line count value, determines if a

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Field	Format	Change
FDCYCCNT	Numeric	Increase by 1. When = FDCYCLE value, set to 0.
FDMLGTH	Alphabetic (Shift or Truncate)	Initially = FDLGTH value. If FDMLGTH > 1, decrease by 1. When FDMLGTH ≤ 1, set = FDLGTH value.
FDFRINC	Alphabetic (Shift or Truncate)	If FDMLGTH > 1, increase by value in FDRINCR field. When FDMLGTH ≤ 1, set = 0.
FDTOINC	Alphabetic (Shift or Truncate)	If FDMLGTH > 1, increase by value in FDTOINCR field. When FDMLGTH ≤ 1, set = 0.
FDSLGTH	Alphabetic (Ripple)	Initially = FDSLGTHR. If FDSLGTH > 1, decrease by 1. When FDSLGTH ≤ 1, set = FDSLGTHR value.
FDFRINC	Alphabetic (Ripple)	If FDSLGTH > 1, increase by 1. When FDSLGTH ≤ 1, set to 0.
FDFRINC	Alphabetic (Wave)	If FDSLGTH > 1, increase by 1. When FDSLGTH ≤ 1, restore to 0.
FDMLGTH	Alphabetic (Wave)	When FDSLGTH ≤ 1, restore to FDLGTH value.
FDFRINC	Alphabetic (Roll)	Increase by 1 for roll to left. Decrease by 1 for roll to right.

Figure 74. Changes Made to FD Table Values as Create Module Builds Output Records

heading message has just been placed on the device, and either continues processing or returns control to the base module.

SERVICE AIDS

You can obtain useful information for use in debugging a (nonexecuted) run of the data generator program by rerunning the program with a DUMP control card inserted in the group of control cards. When the base module recognizes a DUMP control card, it takes the action described in the microfiche copy of the base module code. Debugging Handbook describes both types of dumps that may be obtained when a DUMP card is used. An indicative dump is a limited dump that results from an incorrect, or from a lack of a proper, SYSABEND DD statement. For a complete storage dump, a correct SYSABEND DD statement to define a dump data set must be included in the control cards for the program.

In using the contents of a dump, you will find that register 5 contains the address of the common communication area (common area). This area contains pointers to control blocks and to tables constructed by the data generator program; it also contains parameter values that the modules of the program (1) have obtained from control cards, (2) have assigned as default values, or (3) have arrived at through computation and/or conversion. Figure 75 on page 169 indicates the contents of the fields corresponding to the more frequently used labels in the common area.

Certain debugging information is available as the result of processing the control cards preceding a DUMP control card. In the following text, the information given for a specific location of the DUMP card is in addition to any information resulting from processing any control cards that may have preceded the specified location.

1. DUMP card follows a DSD card: Addresses of DCBs have been determined. The work area for the output record has been established and the area's address is located in the common area.

2. DUMP card follows an FD card: The storage area dump contains the FD table entry relating to that FD card. If other FD cards have been processed, the corresponding FD table entries are also included in the dump.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

3. DUMP card follows a CREATE card: The storage area dump contains tables created by the corresponding CREATE card. An updated copy of any related FD tables is also included. The most recent create table is contained in the dump, and the CURCRTE field of the common communication area contains the address of this table.

The following sections summarize or further explain material appearing in the program listings for the data generator program. This information also supplements the overall view of the program as supplied by Figure 65 on page 146.

Figure 75 on page 169 lists the entries in the common communication area. This area is used by all modules of the data generator utility program.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Offset from Start of Common Area			
Decimal	Hex	Label	Notes
0	0	COMMON	
0	0	PAGENO	Number of next page to be printed.
4	4	LINECT	Number of lines to print on a page.
8	8	LINECTR	Number of lines already printed on current page.
12	C	PARM	Used during invocation. Also used by create module to save SYNAD addresses.
16	10	REPEATNO	"Quantity" value from REPEAT card.
18	12	CREATENO	"Create" value from REPEAT card.
20	14	SYSP	Data Generator SYSPRINT DCB.
116	74	SYSI	Data Generator SYSIN DCB.
216	D8	Q	
216	D8	QFILL	
223	DF	QSIGN	Work Area.
224	E0	QFILL1	
231	E7	QSIGN1	
232	E8	COUNTER	Used in scanning for continuations.
236	EC	OPENLIST	Used during DCB open processing.
236	EC	OPTBYTE1	
240	F0	OPTBYTE2	
244	F4	EXLST	
244	F4	INHDR	
245	F5	INHDR1	
248	F8	OUTHDR	
249	F9	OUTHDR1	
252	FC	INTRL	
253	FD	INTRL1	
256	100	OUTTRL	
257	101	OUTTRL1	
260	104	EXITDCB	
261	105	EXITDCB1	
264	108	TOTAL	
265	109	TOTAL1	
268	10C	EXLST1	
268	10C	EDCB1	
269	10D	EDCB2	
272	110	EXLST2	
272	110	EDCB3	
273	111	EDCB4	
276	114	EXLST3	
276	114	EDCB5	
277	115	EDCB6	

Figure 75 (Part 1 of 4). Common Communication Area

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Offset from Start of Common Area			
Decimal	Hex	Label	Notes
280	118	DLRECL	Default value of record length for DCB opening.
282	11A	DBLKSI	Default value of block size for DCB opening.
284	11C	DRECFM	Default value of record format for DCB opening.
288	120	LEFTOVER	
292	124	OFFSET	
296	128	LPTR	
300	12C	DCBPTR	Address of current DCB.
304	130	COMMON1	
304	130	SAVEMS	Save area for message number if more than one message.
306	132	CONCODE	Condition code to be returned to caller.
308	134	OUTREC	Address of output work area.
312	138	CRTABPT	Address of first create table.
316	13C	CURCRTE	Address of current create entry.
320	140	CURCRGM	Address of current create table.
324	144	CURPIC	Address where next portion of picture is to be placed in picture table.
328	148	PICCTR	Counter to keep track of length of picture remaining to be moved.
332	14C	EXITTAB	Address of first exit name table.
336	150	EXITGM	Address of current exit name table.
340	154	CUREXIT	Address of current exit name in exit name table.
344	158	DELIM	Delimiter for SYSIN records.
348	15C	RECREM	Counter for "Record" quantity.
352	160	CURFD	Address of current FD table (in FD address table).
356	164	CUROUT	Current location in output record being constructed.
360	168	SAVE14	Contents of register 14 saved here.
364	16C	GETMLIST	Parameter list for GETMAIN macro instruction.
364	16C	GLENGTH	
368	170	ADRLIST	
372	174	IND	
372	174	GCODE	
373	175	SPOOL	
374	176	CCODE	
376	178	GCADDR	Address of last storage obtained by a GETMAIN macro instruction.
380	17C	FIRSTGMO	Address of first output DCB storage area.
384	180	CURRGMO	Address of current output DCB storage area.
388	184	LASTGMO	Address of last output DCB storage area.
392	188	FIRSTGMI	Address of first input DCB storage area.
396	18C	CURRGMI	Address of current input DCB storage area.
400	190	LASTGMI	Address of last input DCB storage area.
404	194	CONCODE	
406	196	MS	Current message number.
408	198	INBUFA1	Starting address of input work area (121 bytes).
408	198	INFILL	(10 bytes)
418	1A2	INBUFA	Control card is read into this 111-byte section of INBUFA1.
532	214	DDPTR	
536	218	COMMON2	
536	218	SWITCH	Start of 52-switch area.

Figure 75 (Part 2 of 4). Common Communication Area

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Offset from Start of Common Area			
Decimal	Hex	Label	Notes
536	218	FDCSW	FD-card continuation switch.
537	219	FDNAMESW	
538	21A	FDPCSW	FD picture-continuation switch.
539	21B	FDFMTSW	FD format switch.*
540	21C	FDPLSW	FD picture switch.*
541	21D	RANGESW	FD card keyword indicator switches
542	21E	FILLSW	
543	21F	REPSW	
544	220	INDEXSW	
545	221	INDNMSW	
546	222	BQUOTESW	Binary picture indicator.
547	223	PQUOTESW	Packed decimal picture indicator.
548	224	EQUOTESW	EBCDIC (character-string) picture indicator.
549	225	FDSW	
550	226	DSDSW	
551	227	NOGOSW	"No execution" switch. Indicates syntax checking only.)
552	228	CREATESW	First CREATE-card switch.
553	229	DSDCSW	DSD continuation-card switch.
554	22A	CRCSW	CREATE continuation-card switch.
555	22B	EXITSW	Indicator that an initial exit-name table exists.
556	22C	EODSTOP	Switch to stop generation on input end-of-data.
557	22D	DSDNULSW	Not used.
558	22E	DSDORGSW	
559	22F	DSDDDSW	
560	230	CRTBLK	Indicator for a blank CREATE card.
561	231	NAMECSW	Name continuation switch.
562	232	PICCSW	CREATE card picture-continuation switch.
563	233	BUFPSW	
564	234	ENDSW	
565	235	COMCSW	Comments continue switch.
566	236	FLAGSW	
567	237	PAGESW	
568	238	EPSW	
569	239	SYSISW	
570	23A	SYSPSW	
571	23B	OLDNEWSW	Input/output data set indicator.
572	23C	FLUSHSW	
573	23D	FLUSHSW1	
574	23E	DSDOSW	
575	23F	DSDISW	
576	240	QUANSW	Create card "quantity" switch.
577	241	PARENSW	Indicates detection of a left parenthesis.
578	242	REPEATSW	Used to test if a REPEAT card remains to be processed.
579	243	SYSINEOD	Address of end of SYSIN data.
* Only one of these switches should be on.			

Figure 75 (Part 3 of 4). Common Communication Area

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Offset from Start of Common Area			
Decimal	Hex	Label	Notes
588	24C	FDPLGTH	FD picture length.
592	250	SGCADDR	Save area for address of storage obtained by GETMAIN macro instruction.
596	254	FDPTR	Address of current FD TABLE ENTRY
600	258	FDPTR1	Address of first FD table.
604	25C	FDPTR2	Address of current FD table.
608	260	COMMON3	
608	260	FDCTR	Count of number of entries in current FD table.
612	264	LREMAIN	Length of FD picture remaining at end of scanning an FD card.
616	268	COMPCTR	
620	26C	LMOVED	
624	270	U	Current random number value.
628	274	PICEND	Location of end of picture in output record.
632	278	CURFDGM	Address of current FD-address table.
636	27C	SWTCH	
636	27C	SYSINSEL	Field-select indicator switch.
637	27D	FIRSTSW	
638	27E	FRSTSW	
639	27F	STOPSW	
640	280	COPYVAL	Value of COPY parameter from a CREATE card.
644	284	COPYFD	Pointer to FD address used in copying a "name" group.
648	288	COPYFDGM	Address of FD-address table.
652	28C	NAMCTR	Number of FD addresses to be copied for a "name" group.
654	28E	NAMCTR1	Counter used in copying a "name" group.
656	290	INRECSZ	Logical record length of an input record.
658	292	OUTRECSZ	Logical record length of an output record.
660	294	INRECFM	Input record format.
661	295	RECOFFST	Offset to start of data in output record.
662	296	OUTRECFM	Output record format.
664	298	PICBASE	Address of start of picture table.
668	29C	MESSAGE	

Figure 75 (Part 4 of 4). Common Communication Area

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Figure 76 lists the fields of the data control block (DCB). The labels, as given in this dummy section, may vary in name from the labels for the DCB fields as given in the Debugging Handbook. However, the offsets from zero correspond in meaning with those given in the Debugging Handbook.

Offset from Start of DCB		Label
Decimal	Hex	
0	0	DCBD
0	0	FILL
16	10	KEYLEN
17	11	DEVT
18	12	FILL1
26	1A	DSORG1
26	1A	DSORG
28	1C	FILLER
28	1C	IOBAD
32	20	BFTEK
33	21	EODAD
36	24	RECFM
37	25	EXLIST
40	28	DDNAME
40	28	DEBAD
40	28	IFLGS
48	30	GETAD
48	30	OFLGS
49	31	OFLGS1
50	32	MACRF
52	34	FILL2
56	38	SYNAD
60	3C	CIND
62	3E	BLKSI
64	40	FILL3
82	52	LRECL
84	54	FILL4
256	100	NEXTDCB
260	104	DDNAME1
268	10C	EODSW
269	10D	DCBSW1
270	10E	DCBSW2
271	10F	DCBSW3
272	110	INREC
276	114	GMLGTH
278	116	FIELDSEL
279	117	SPARE

Figure 76. Data Control Block

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Figure 77 lists the defined constants (DCs) that are used by the various modules of the data generator program.

Label	Base	Clean-up	FD Analysis	FD Table	Create Analysis	Create
'C1				C'FX'		
'C2				C'SL'		
'C3				C'TL'		
'C4				C'SR'		
'C5			C'NAME='	C'TR'	C'QUANTITY='	
'C6	C'LINECT='		C'LENGTH='		C'NAME='	C' (see 'C30 for base)
'C7			C'STARTLOC='		C'PICTURE='	
'C8			C'PICTURE='	C'RP'	C'FILL='	C'RA'
'C9			C'FORMAT='	C'RO'	C'INPUT='	C'ZD'
'C10			C'ACTION='	C'WV'	C'EXIT='	C'PD'
'C11			C'FILL='			C'BI'
'C12			C'CYCLE='	C'BI'		
'C13	C'END'		C'RANGE='	C'PD'		
'C14	C'FD'		C'CHARACTER='	C'AL'	C'COPY='	
'C15	C'CREATE'		C'SIGN='	C'AN'		
'C16	C'REPEAT'		C'INDEX='	C'CO'	C'P'''	
'C17	C'DUMP'		C'REPLACE='		C'B'''	
'C18	C'OUTPUT=('					
'C19	C'INPUT=('					
'C20				C'RA'		
'C22				C'ZD'		
'C24					C'SYSIN	
'C25	C'QUANTITY='				C'$$$E'	
'C26	C'CREATE='					
'C28			C'B'''			
'C29			C'P'''			
'C30	C' IEB729I PERMANENT I/O ERROR'					
'D1	H'256'		H'-64'	H'-1'	H'-1'	F'0'
'D2	F'123456'		H'32767'	H'256'	F'0'	H'-1'
'D3	F'65535'		H'256'	H'2'	H'256'	H'8'
'D4				H'-2'	H'1'	H'4'
'D5				H'1'	H'16'	F'524291'
'D6				H'-4'	F'1'	H'-4'
'D7				H'-3'		H'256'
'D8				H'32767'		
'X2	X'F0F0F0F1'					
'X5	X'000B'					X'0000'
'X7	X'0030'			X'0000'		
'X8	X'0000000000000000'					
'X9	X'0028'					
'X17				X'001A'		
'X18				X'0024'		
'X19				X'003F'		
'X23	X5000'					
'X26					X'02147483647F'	
'X30			X'0000'			
'X31			X'0000000000000000'	X'0004'		
'X32			X'000002147483647F'	X'0003'		
'X36				X'000002147483647F'		
FOXZEROS		X'F0F0F0F0F0F0F0F0'			X'F0F0F0F0F0F0F0F0'	
NO						X'00'
OFF		X'00'	X'00'	X'00'	X'00'	X'00'
ON		X'FF'	X'FF'	X'FF'	X'FF'	X'FF'
ONE						FL4'0'
'SIZ001'	A('DATEND-'DATD)					
'T1					F'0'	F'0'
'TEMP4			F'0'	F'0'	F'0'	F'0'
YES						X'FF'

Figure 77. Defined Constants for Modules of the Data Generator Program

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Figure 78 lists the equated symbols (EQUs) that are used by the various modules of the data generator program.

Label	Module					
	Base	Clean-up	FD Analysis	FD Table	Create Analysis	Create
A2J7 F6D3 'L RET '9CE		'EL01 1	1	F8B4 1	1 'EL01	1 'EL01
'9D4 '9D7 '9EC '9EF '9E2	'EL01 A2B3		F4G11		CARDSCAN	A7A12
'9E9 '9FC '9FD '9F0 '9F2	PDDNAMER PDDNAMER		'EL02	'EL02	A6C5	A7A18
'9F3 '9F4 '9F5 '9F7	LABEL1 SCAN1			F9E5	A6A11 KEYSCAN	

Figure 78. Equated Symbols for Modules of the Data Generator Program

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

TABLES AND WORK AREAS USED BY MODULES OF THE DATA GENERATOR PROGRAM

Figure 79 is a grid indicating the modules that establish, use, and modify the major work areas and information tables of the data generator program. Mnemonic names for the tables and areas are placed in parentheses and correspond to the names given in the module cross-references on microfiche listings.

Modules	BASE	FDANL	FDTAB	CRANL	CREAT	MSG	CLNUP	
Table/Area	Module Action Code: B = Module gets storage for, and/or enters data into. U = Module uses or modifies the area.							
Common Comm. Area (COMMON)	B	U	U	U	U	U	U	
Create Table (CRTAB)				B				
Exit Name Table (EXITTAB)				B	U			
FD Table (FDTBL)		B	B		U			
FD Address Table (FDADTAB)				B	U			
Input Buffer (INBUF)	B, U	U		U				
Input/Output DCB	B				U		U*	
Message Table (MESSAGE)						U		
Message Pointer Table (MSGPTR)	B					U		
Create Picture Table (CRPICT)				B	U			

*Also Closes the DCBs and releases their storage.

Figure 79. Data Generator Modules Information Tables and Area

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

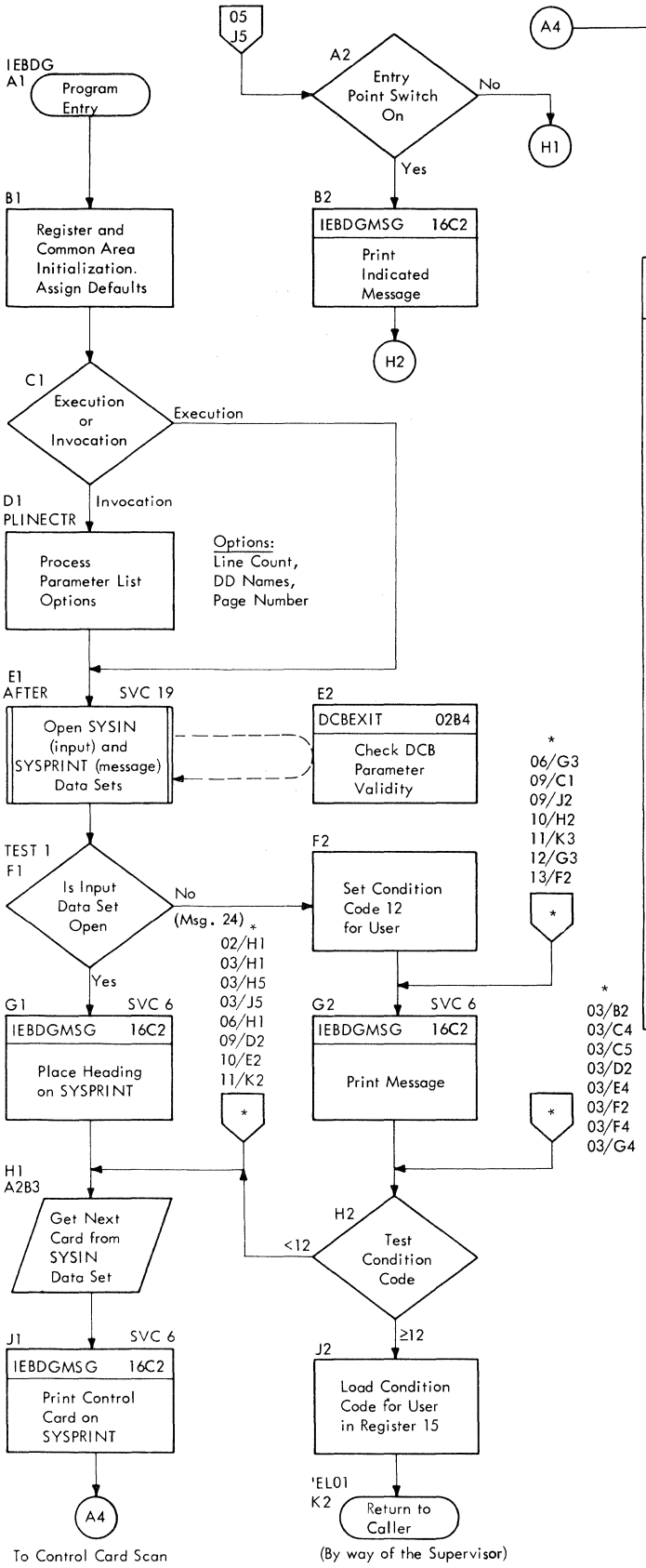
Figure 80 contains a summary of the input and output information available on microfiche listings of the data generator program.

Inputs	Outputs
Base Module	
Data Generator Control Cards. DD cards for all data sets used. Either: Parameter List address for invocation, or Job Control Language EXEC card parameters.	Register 5, pointing to a common communication area. Message indicator in the common area. Register 9 pointing to a data generator control card operation field in an input buffer—this is for output to an analysis module.
Clean-Up Module	
Register 5 and other pointers (in the communication area) indicating, respectively, the addresses of the communication area and control tables.	DCB storage areas and associated buffer areas, and the input record work area, are released to the system.
Message Module	
Register 5, pointing to communication area. Message number indicator. Actual or default values for line count and page number. Output DCB name. Indicators for: Output DCB open or not. Channel 9 carriage control. Channel 12 carriage control	Heading and paging information. Program messages. Error return codes. Control card images.
FD Analysis Module	
Register 9, pointing to a control card image in an input buffer.	One or more FD tables—520 bytes each. FD table entry with some parameter values. Temporary storage with a picture or format pattern.
FD Table Module	
FD table entry. Register 5, pointing to communication area. SGCADDR pointer to picture temporary storage. Switch indicating picture type, if any.	Completed FD table entry—64 bytes.
Create Analysis Module	
Register 5, pointing to communication area. Register 9, pointing to control card image in input buffer.	One or more create tables—512 bytes each. Create table entry—28 bytes. Picture table—(L 6)* bytes. FD address table(s)—88 bytes each. Exit name table(s)—72 bytes each. *See Figure 73, "Picture Table Constructed by Create Analysis Module," for definition of L.
Create Module	
Register 5, pointing to communication area. Create table(s). Picture table(s). FD address table(s). Exit name table(s).	Records written on an output device as specified by DD name on a DSD control card.

Figure 80. Module Inputs and Outputs

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 01. BASE MODULE (PART 1 OF 3)



A4 → CONTROL CARD SCAN

For a given control card type, check for the initial card, a continuation card, and a comments card.

The first control card of a set of control cards for this utility program must be a DSD card. In the following table, the indicated switches are tested, or the indicated tests are preformed. The action taken depends on whether a switch is "on" (= 1) or "off" (= 0), or whether a test result is "yes" or "no".

Switch or Test No.	Switch or Test Name	"On" or "Yes" Action (*)	"Off" or "No" Action
SW 1	Comments Continue SW	Test SW 2 or go to Chart 01, Box H1.	Test SW 2
SW 2	FD Continue SW	Go to Chart 05, Box A1	Test SW 3
SW 3	Create Continue SW	Go to Chart 09, Box A2	Test Sw 4
SW 4	DSD Continue SW	Go to Chart 03, Box C3	Test SW 5
SW 5	FD Picture Continue SW	Go to Chart 05, Box A1	Test SW 6
SW 6	Create Picture Continue SW	Go to Chart 09, Box A2	Test for DSD Control Card
Test 1	DSD Control Card	Go to Chart 03, Box B3	Test for FD Control Card
Test 2	FD Control Card	Go to Chart 05, Box A1	Test for CREATE Control Card
Test 3	CREATE Control Card	Go to Chart 09, Box A1	Test for REPEAT Control Card
Test 4	REPEAT Control Card	Go to Chart 03, Box B1	Test for END Control Card
Test 5	END Control Card	Go to Chart 02, Box B1	Test for DUMP Control Card
Test 6	DUMP Control Card	Terminate the Job	Return to Supervisor

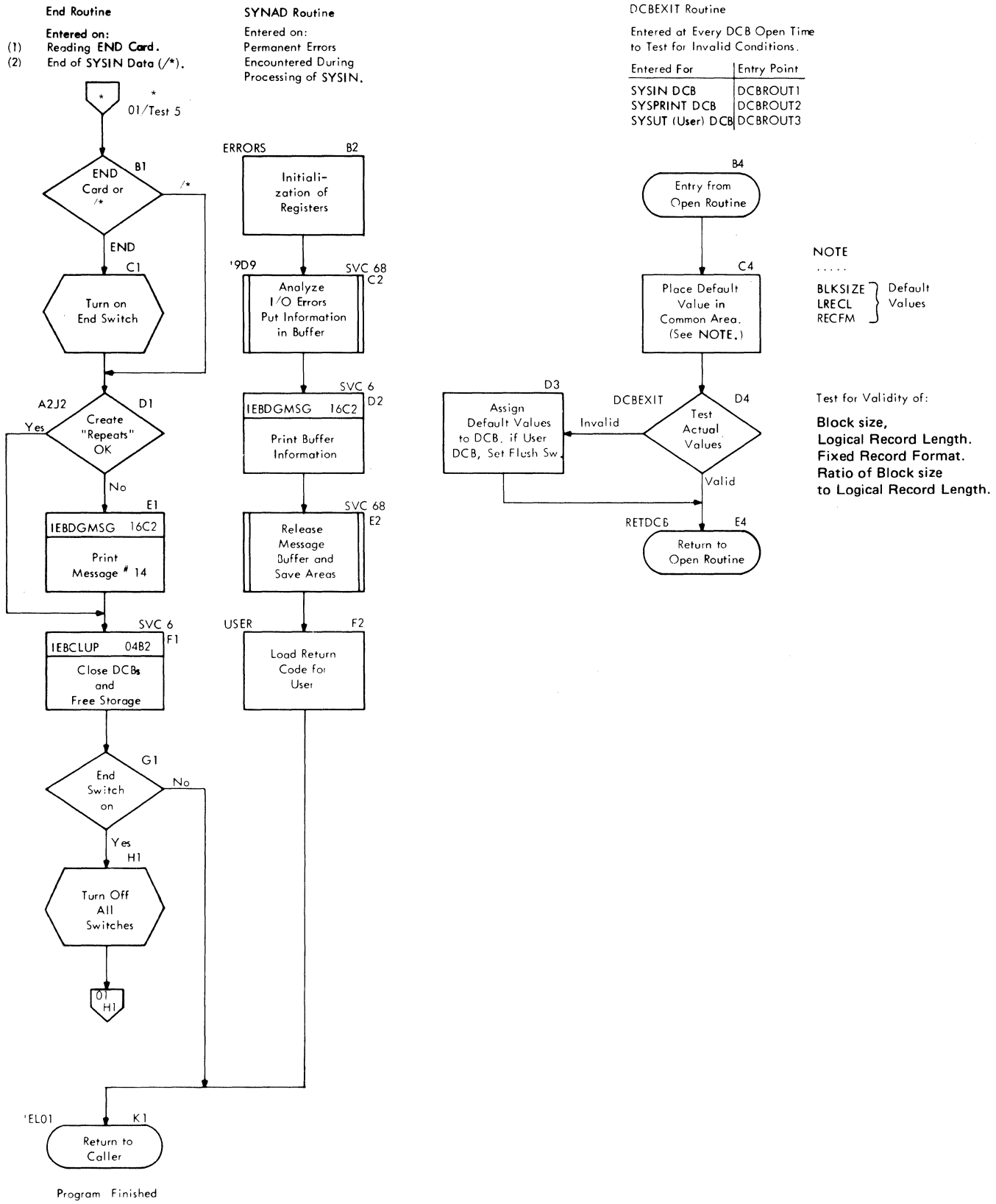
To Point Indicated in Above Table.

(*) Chart Designations:

01-03	Base Module
05,06	FD Analysis Module
09-13	Create Analysis Module
16	Message Module

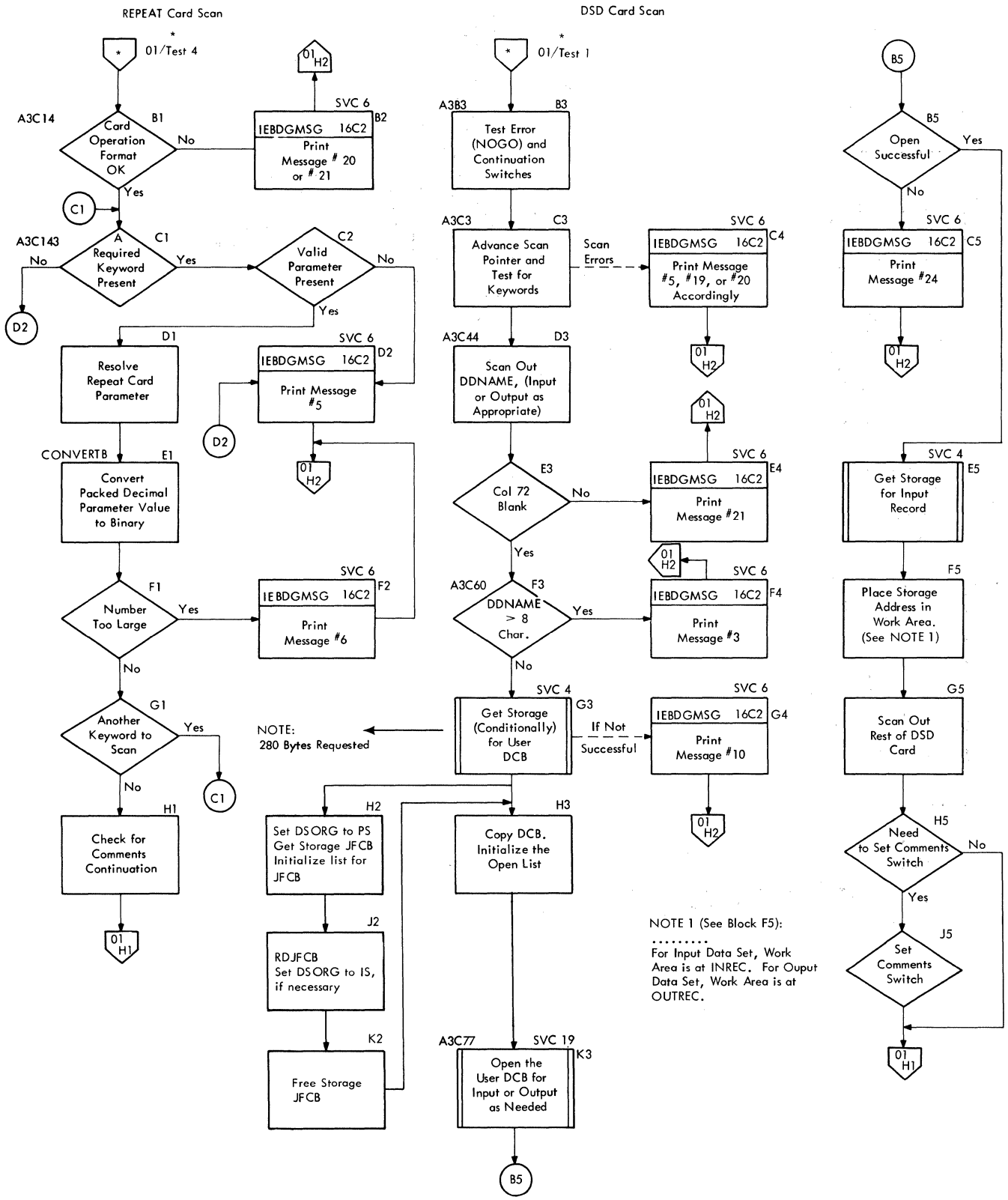
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 02. BASE MODULE (PART 2 OF 3)



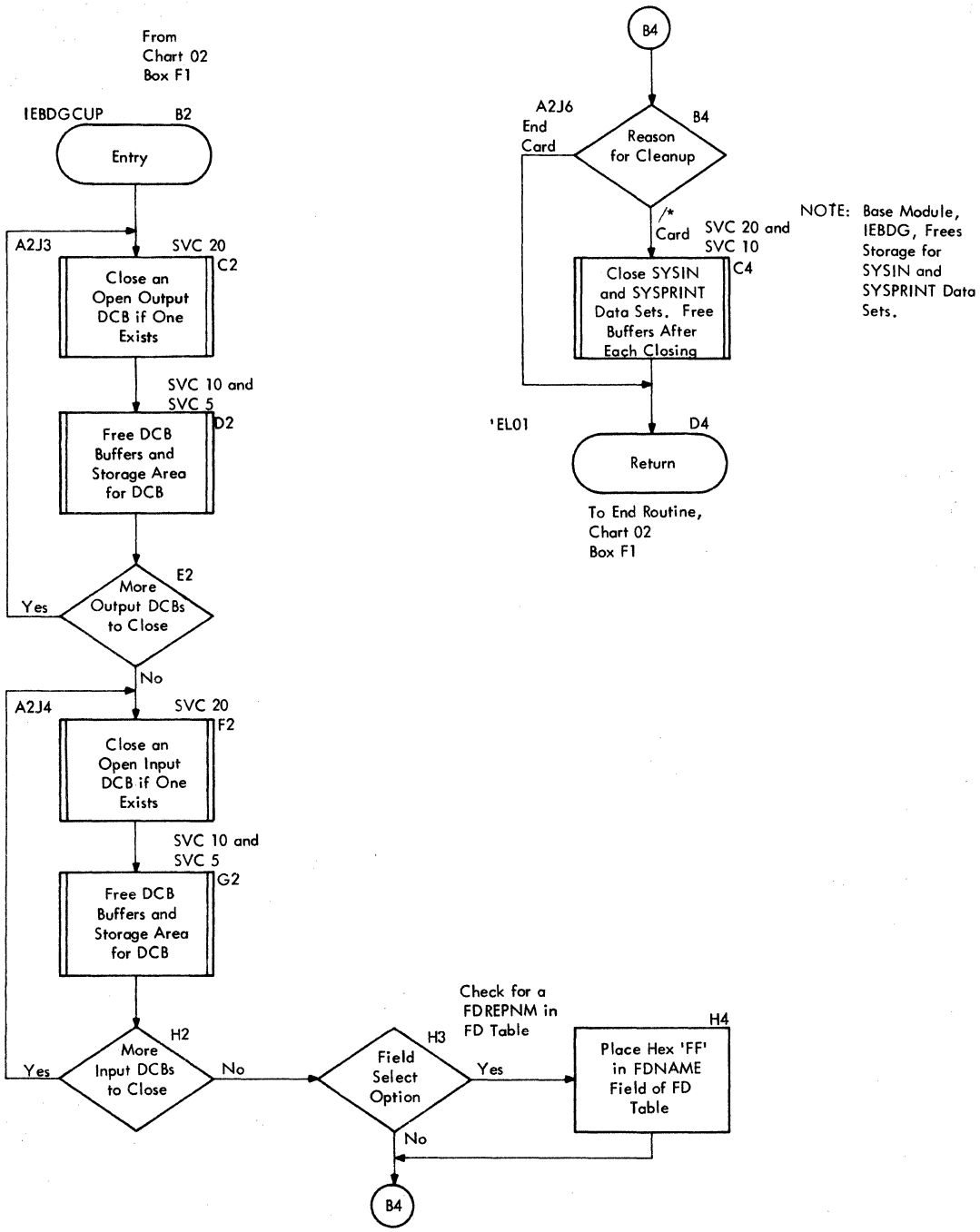
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 03. BASE MODULE (PART 3 OF 3)



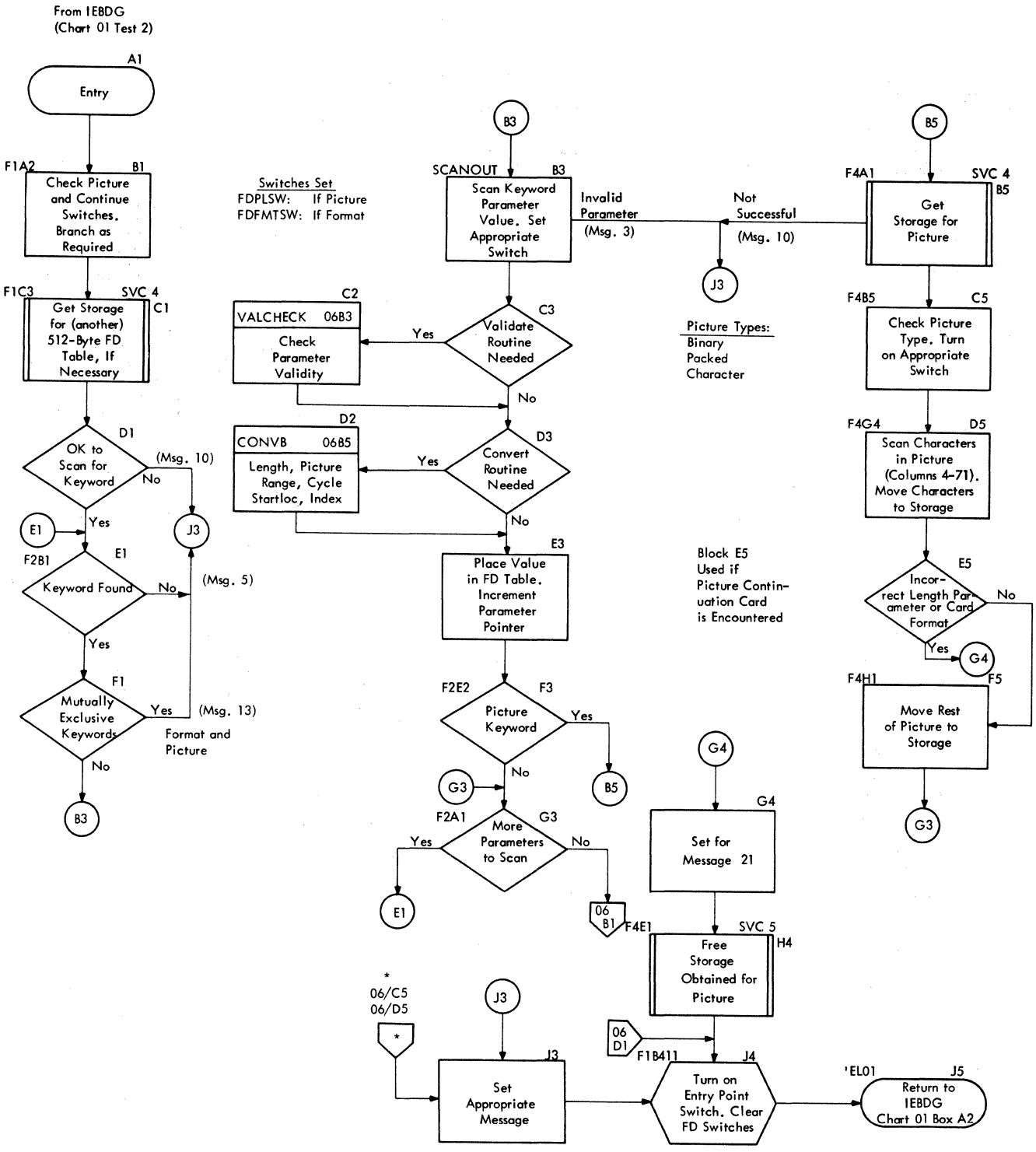
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 04. CLEANUP MODULE, IEBDGCUP



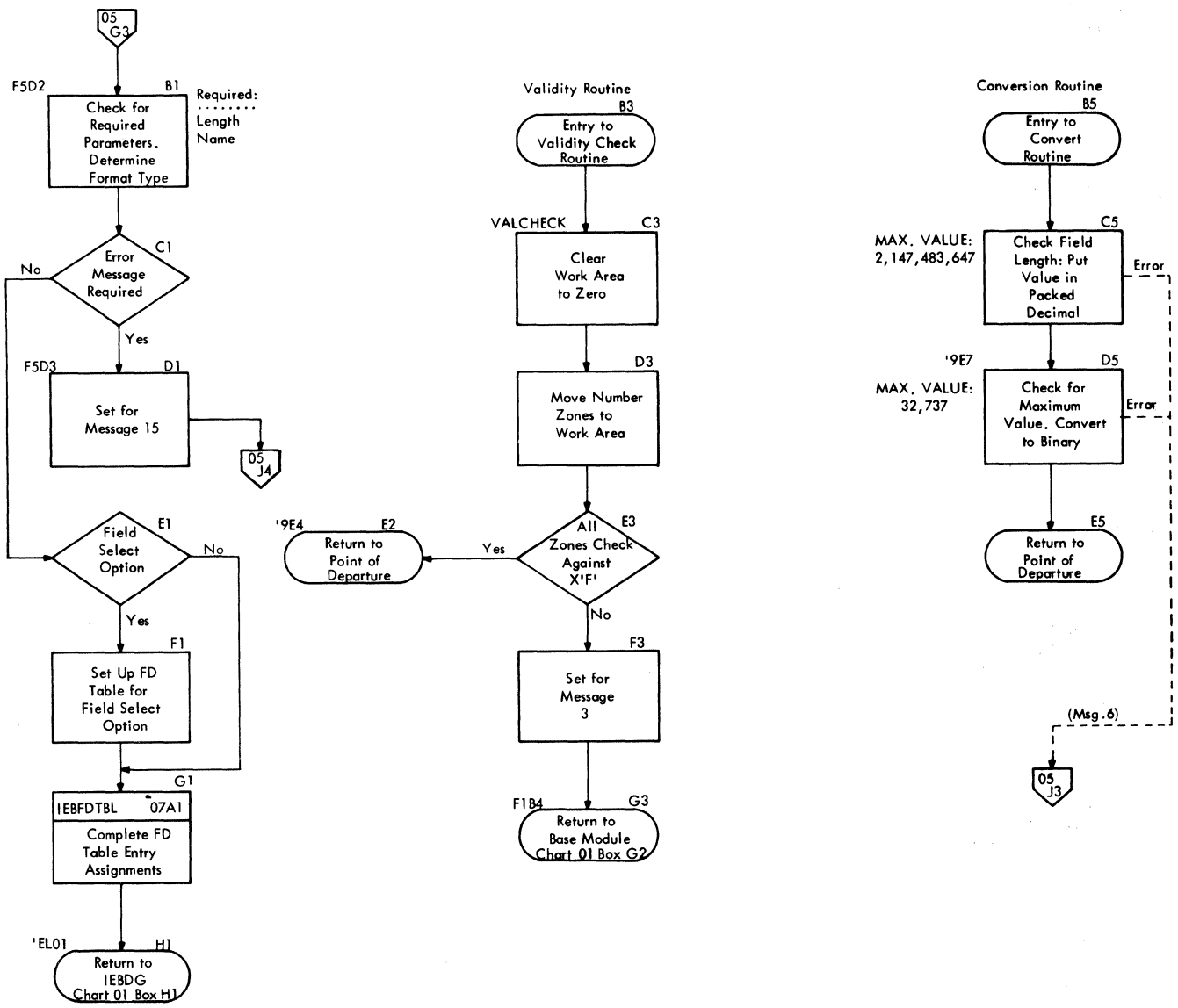
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 05. FD-ANALYSIS MODULE, IEBFDANL (PART 1 OF 2)



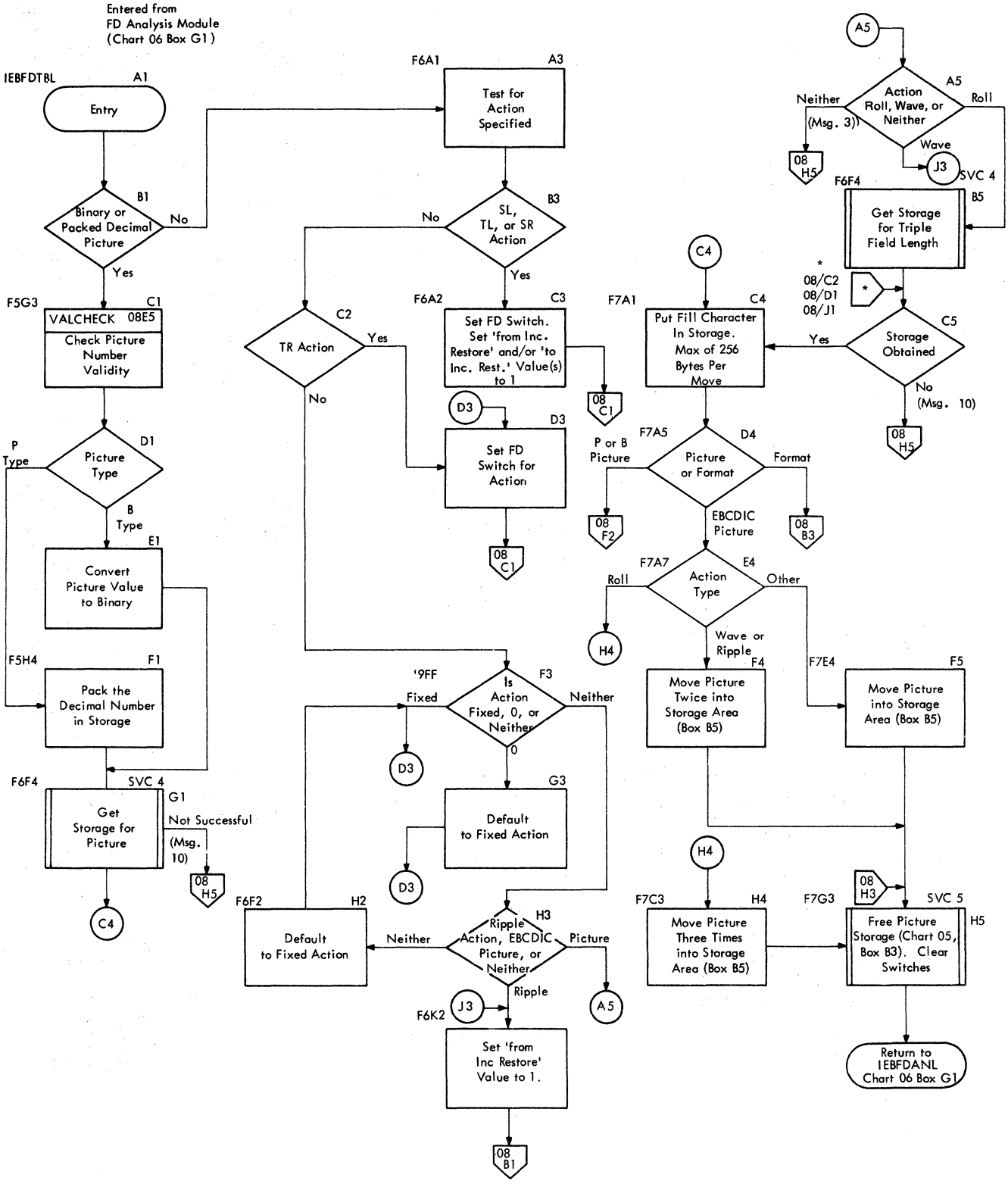
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 06. FD-ANALYSIS MODULE, IEBFDANL (PART 2 OF 2)



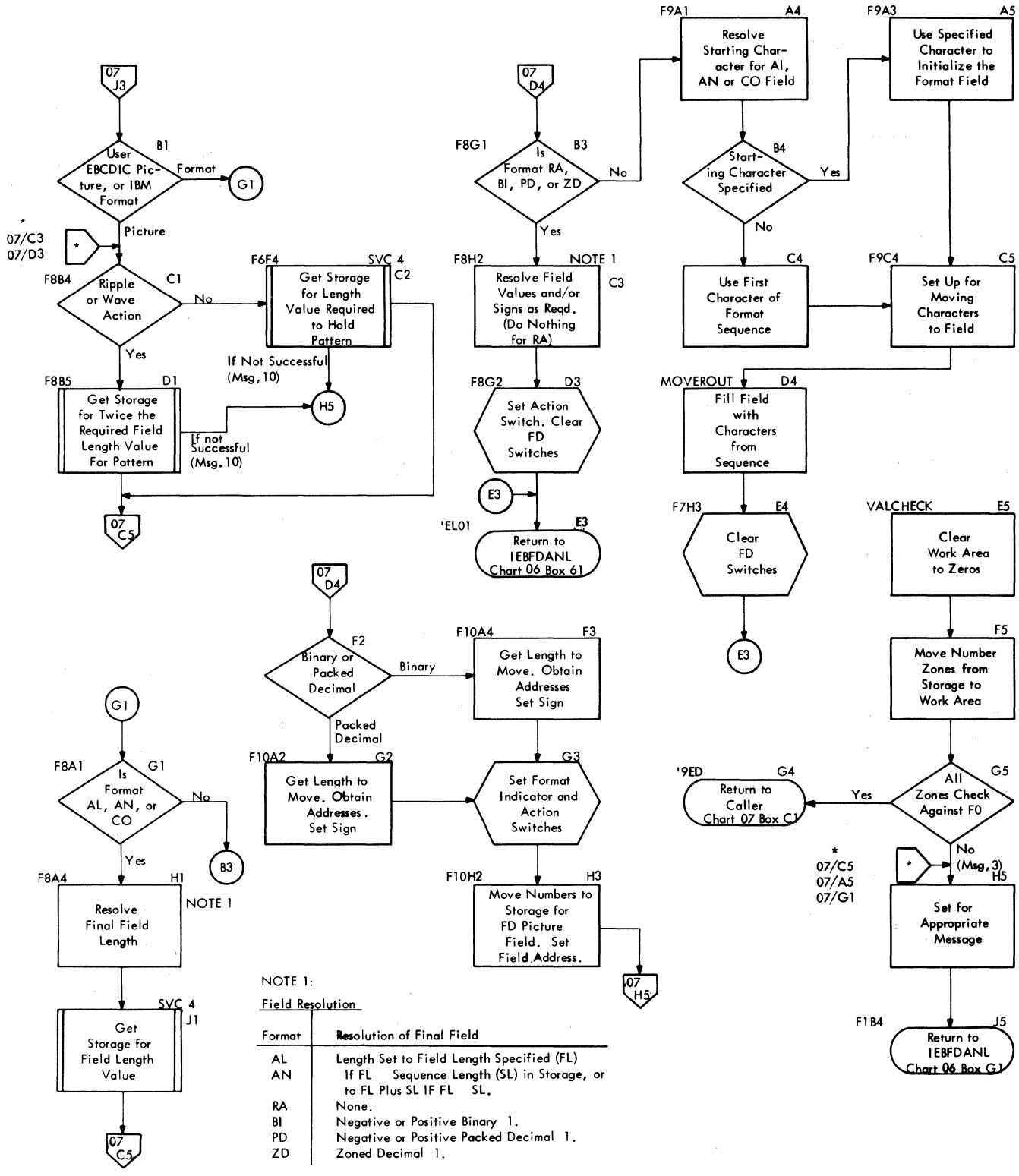
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 07. FD-TABLE MODULE, IEBFDTBL (PART 1 OF 2)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 08. FD-TABLE MODULE, IEBFDTBL (PART 2 OF 2)



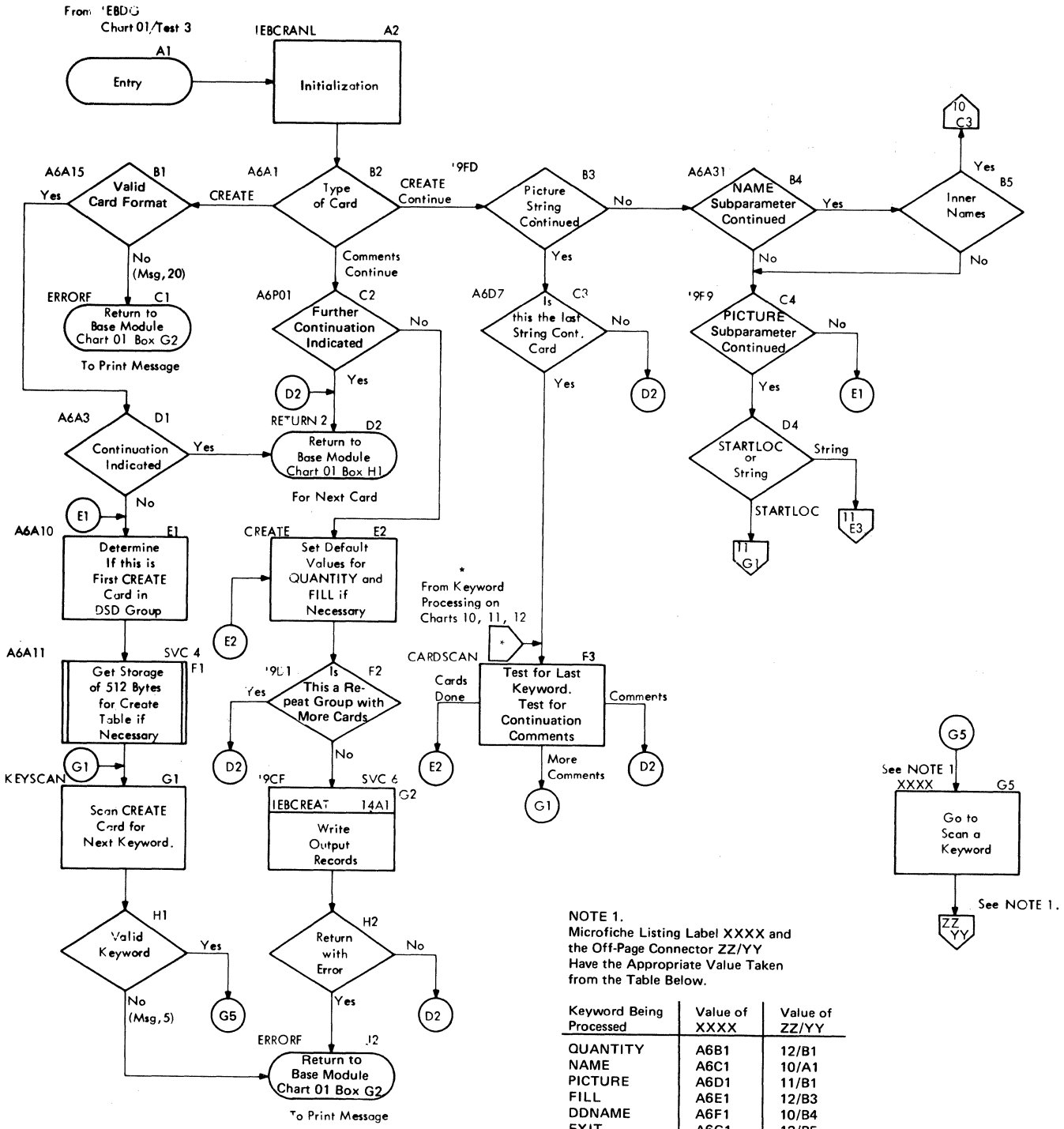
NOTE 1:

Field Resolution

Format	Resolution of Final Field
AL	Length Set to Field Length Specified (FL)
AN	If FL Sequence Length (SL) in Storage, or to FL Plus SL IF FL SL.
RA	None.
BI	Negative or Positive Binary 1.
PD	Negative or Positive Packed Decimal 1.
ZD	Zoned Decimal 1.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 09. CREATE ANALYSIS MODULE, IEBCRANL (PART 1 OF 5)

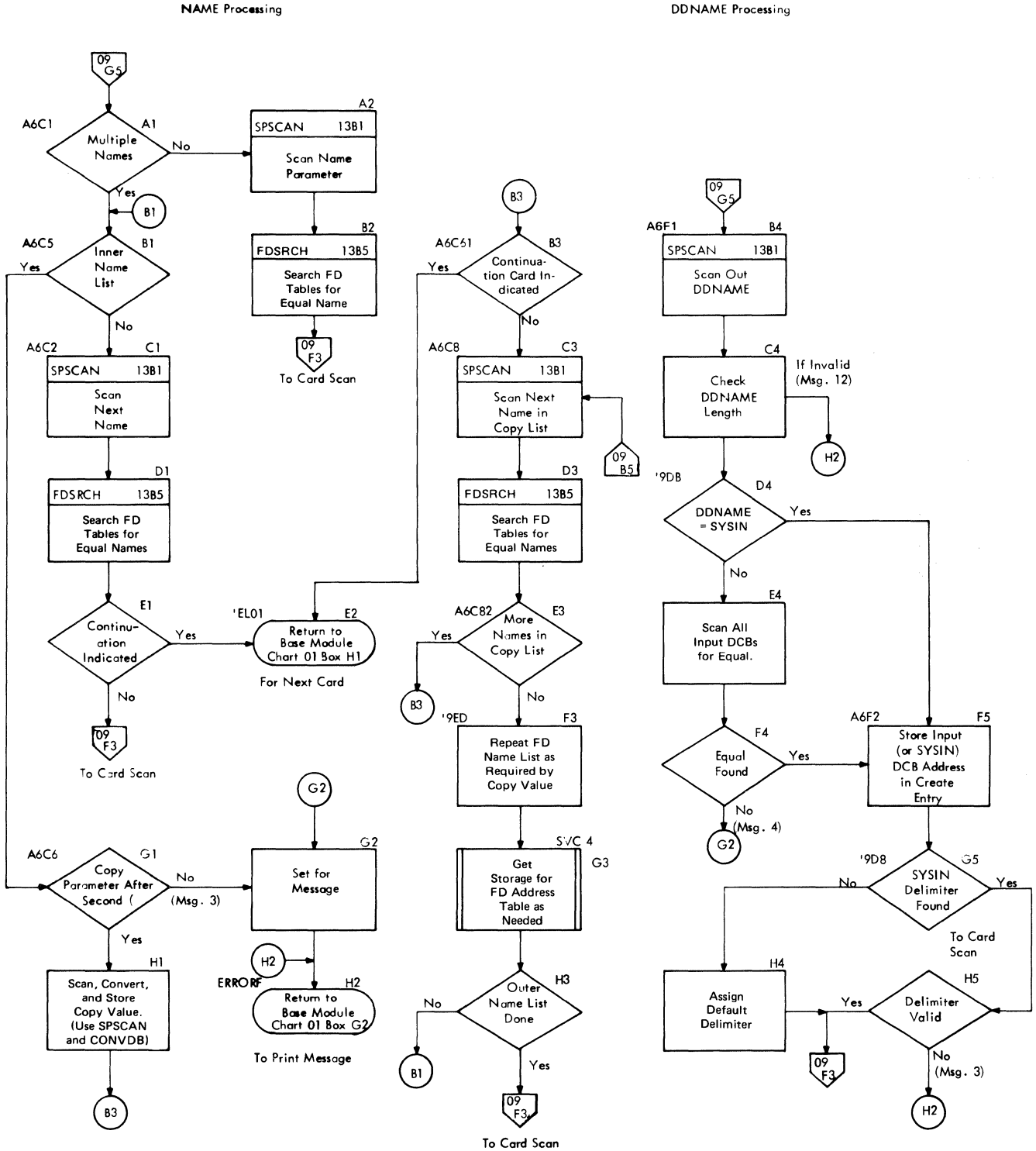


NOTE 1.
Microfiche Listing Label XXXX and the Off-Page Connector ZZ/YY Have the Appropriate Value Taken from the Table Below.

Keyword Being Processed	Value of XXXX	Value of ZZ/YY
QUANTITY	A6B1	12/B1
NAME	A6C1	10/A1
PICTURE	A6D1	11/B1
FILL	A6E1	12/B3
DDNAME	A6F1	10/B4
EXIT	A6G1	12/B5

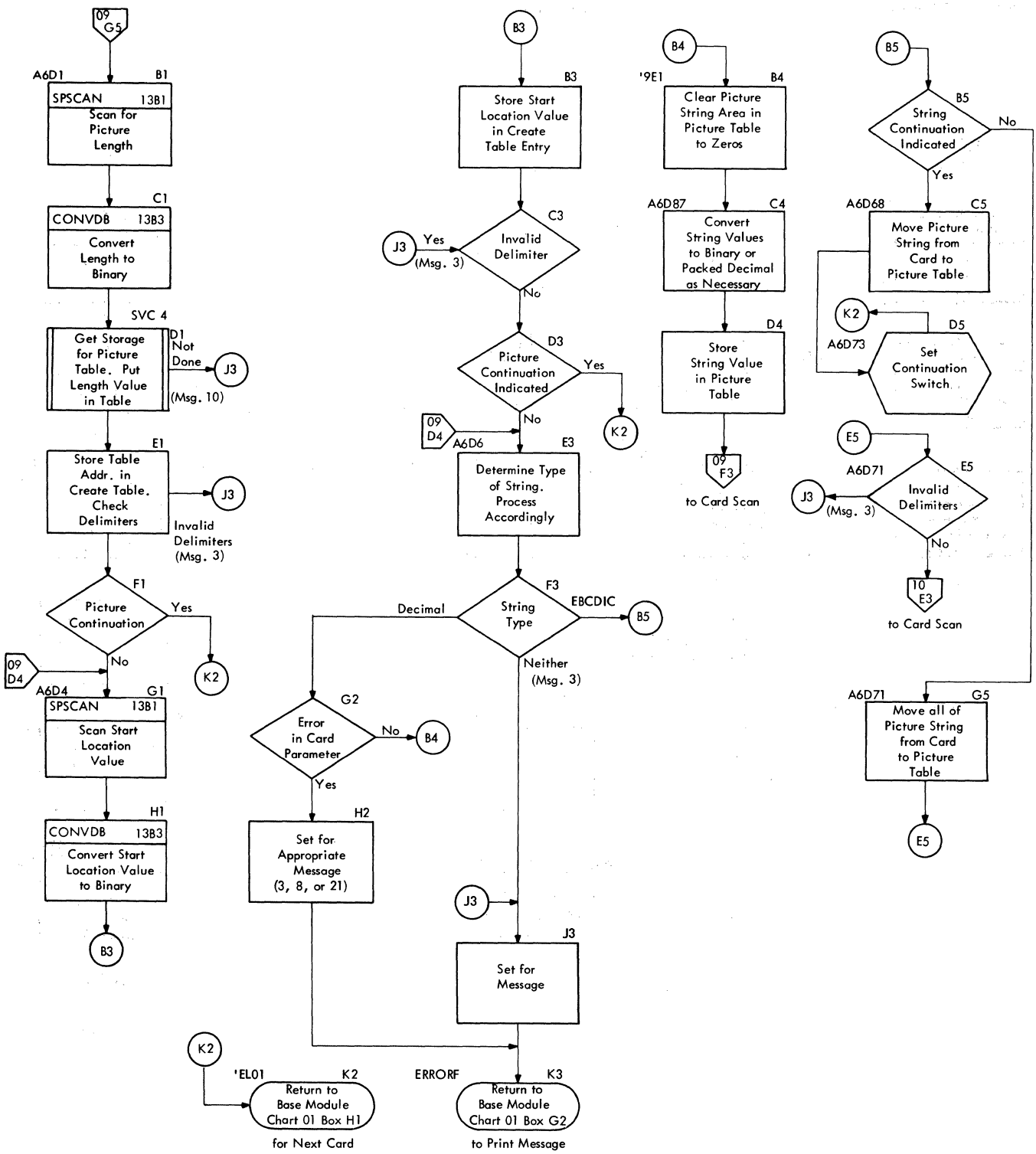
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 10. CREATE ANALYSIS MODULE, IEBCRANL (PART 2 OF 5)



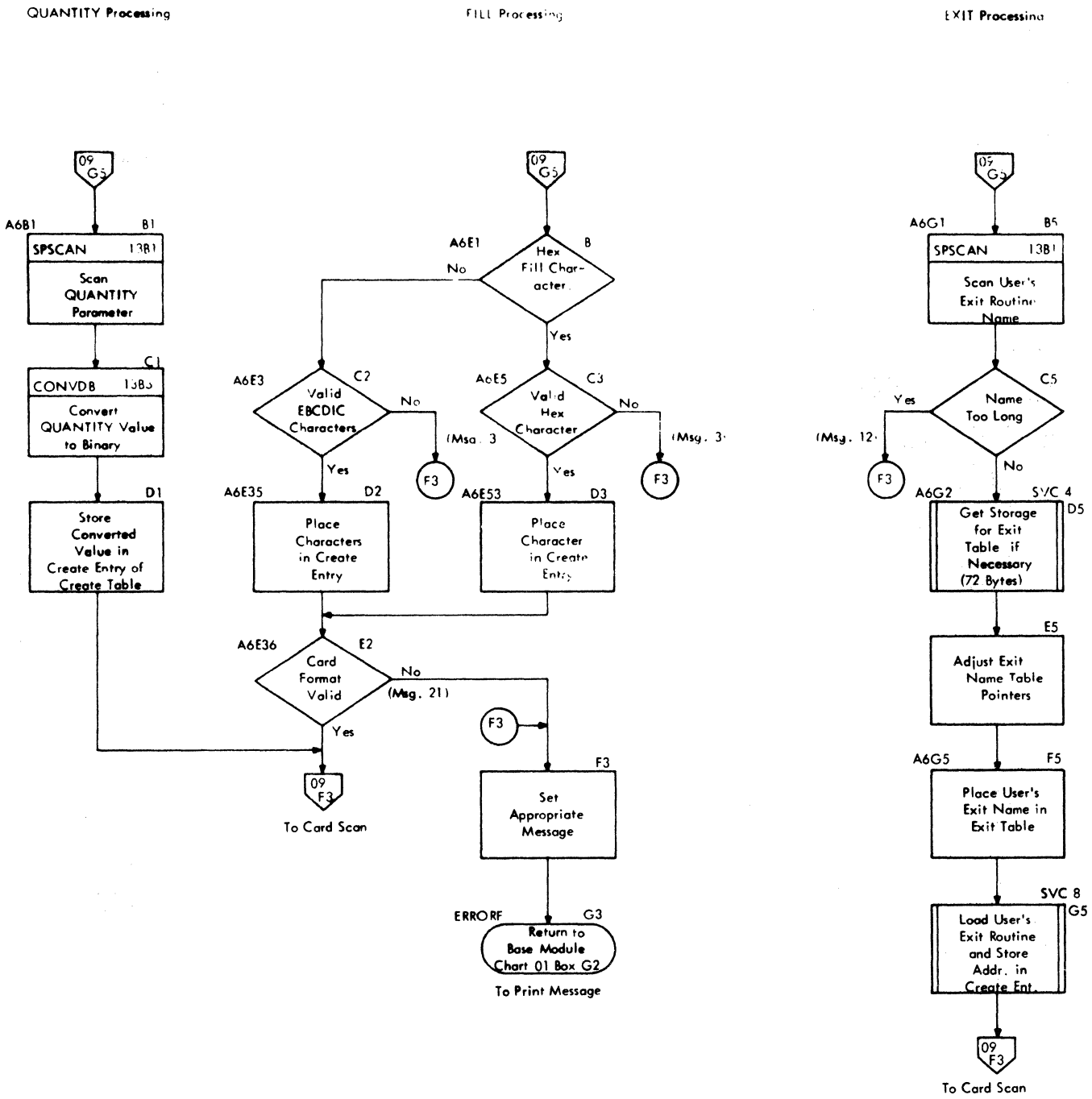
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 11. CREATE ANALYSIS MODULE, IEBCRANL (PART 3 OF 5)



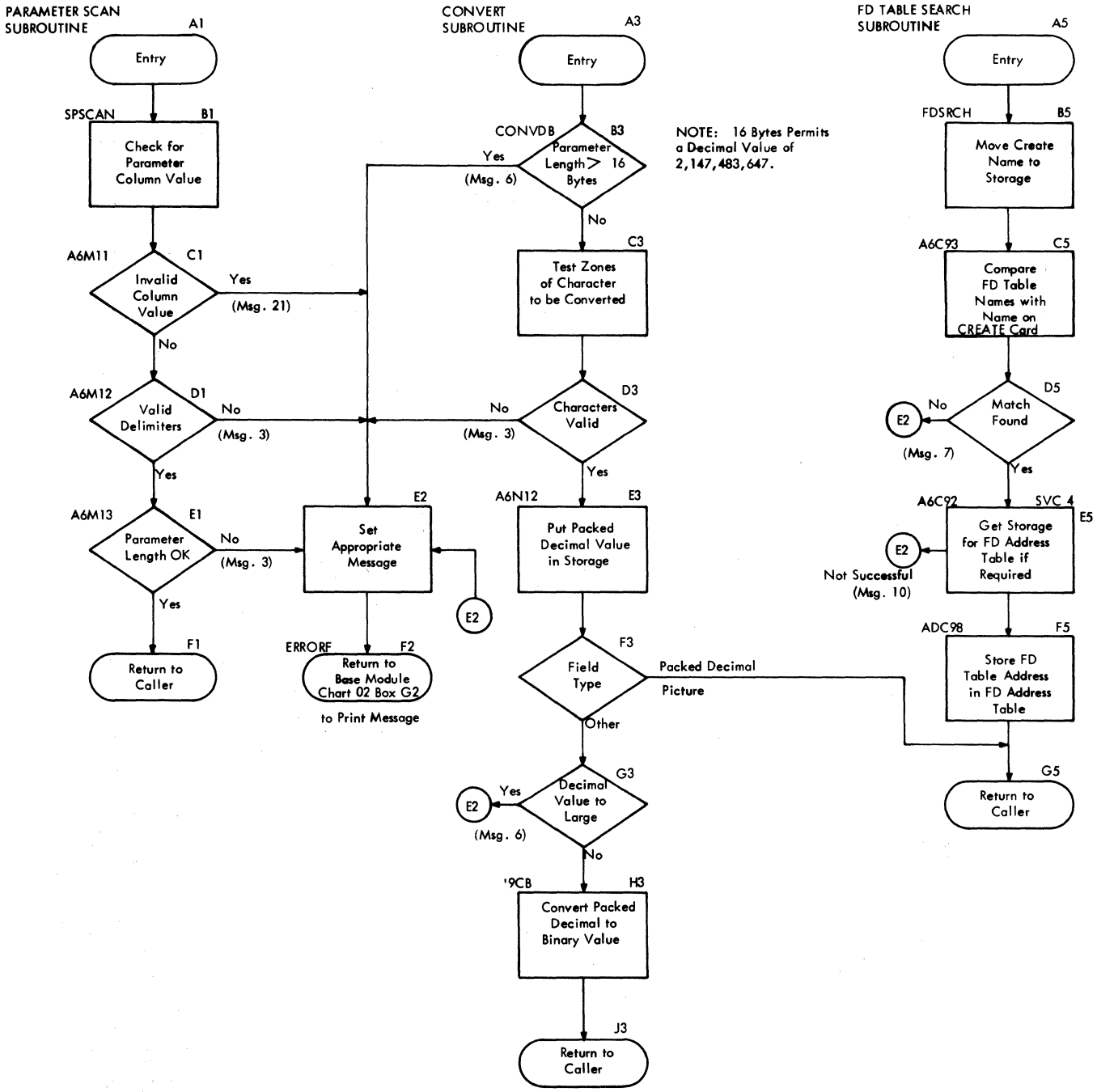
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 12. CREATE ANALYSIS MODULE, IEBCRANL (PART 4 OF 5)



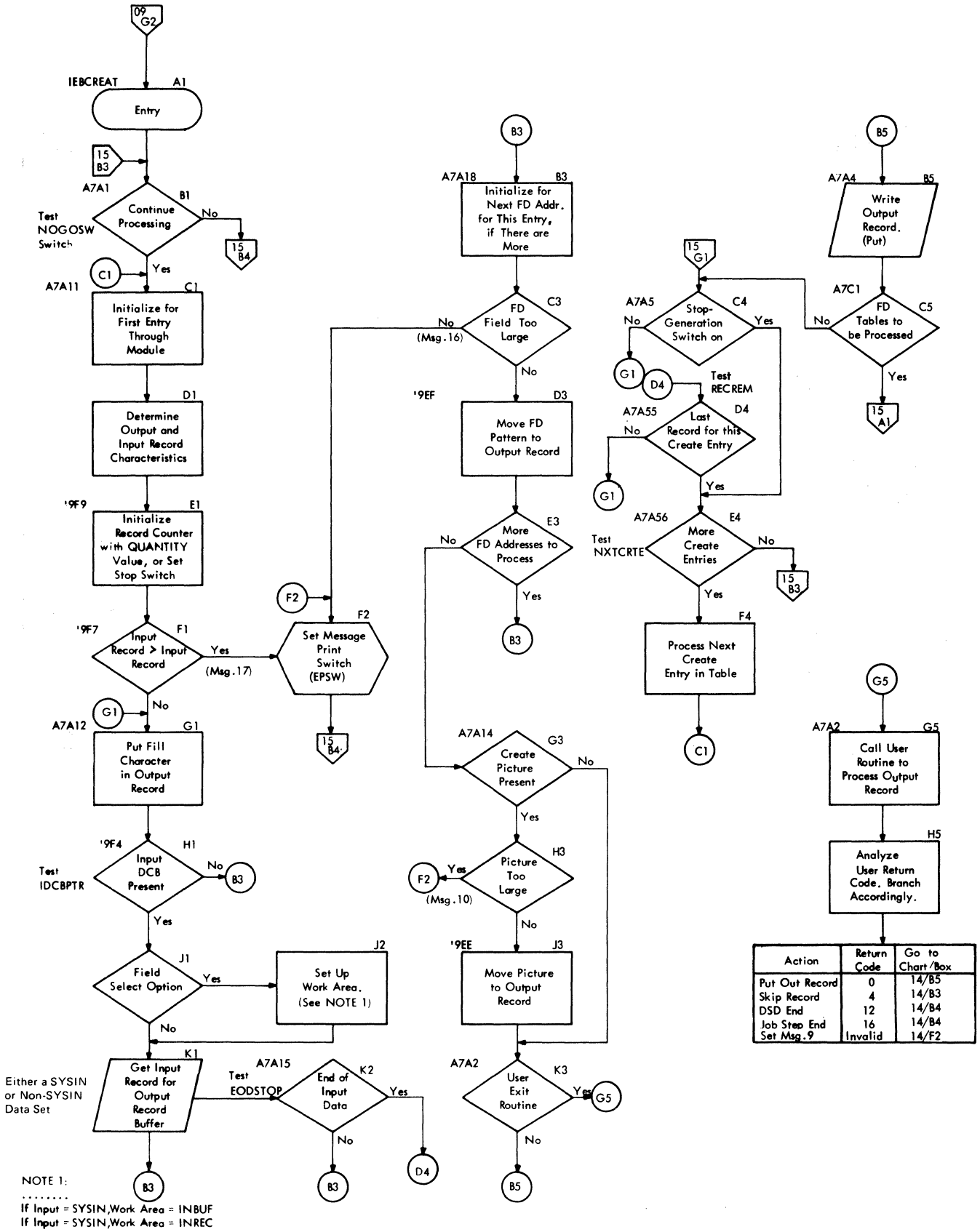
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 13. CREATE ANALYSIS MODULE, IEBCRANL (PART 5 OF 5)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

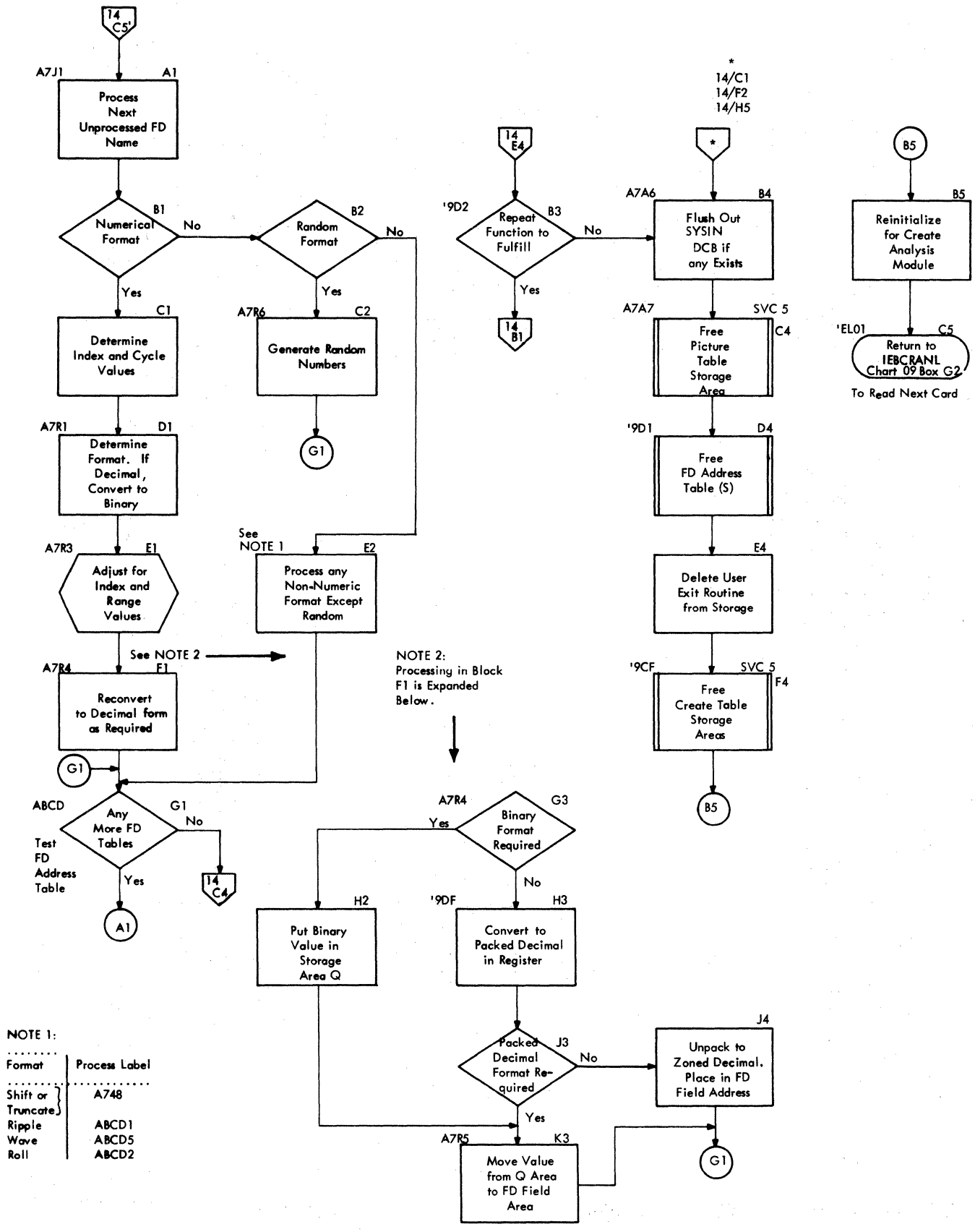
CHART 14. CREATE MODULE, IEBCREAT (PART 1 OF 2)



Action	Return Code	Go to Chart/Box
Put Out Record	0	14/B5
Skip Record	4	14/B3
DSD End	12	14/B4
Job Step End	16	14/B4
Set Msg.9	Invalid	14/F2

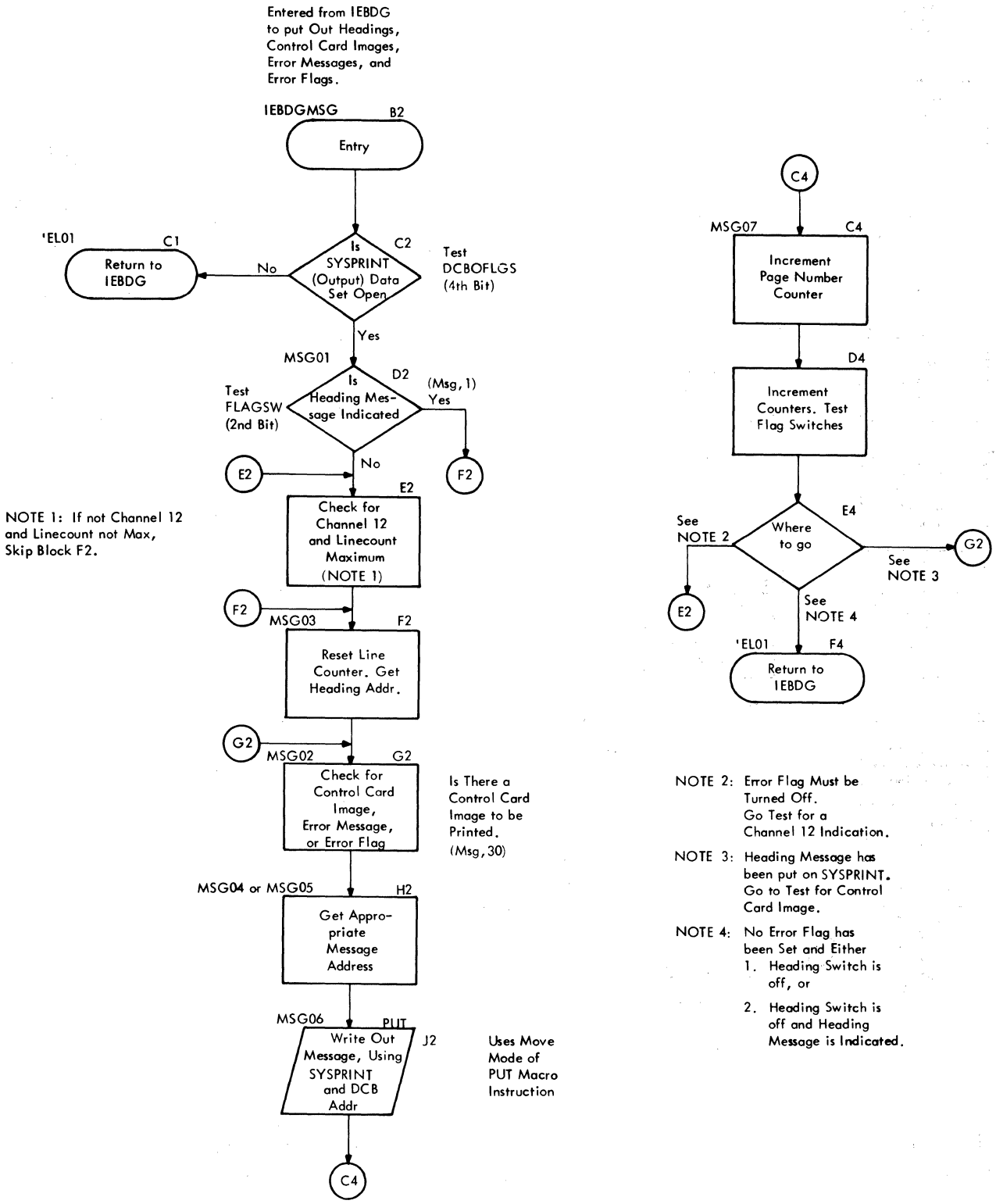
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 15. CREATE MODULE, IEBCREAT (PART 2 OF 2)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 16. MESSAGE MODULE, IEBDGMSG



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

The IEBEDIT program creates a sequential data set containing job control language (JCL) statements and system input data by extracting sets of statements representing jobs or job steps from a master file. The input to the program is in two data sets:

- SYSIN contains control statements that allow the user to control the editing of the master file of JCL statements and data.
- SYSUT1 contains the master file of JCL statements and data.

The output of the program is in two data sets:

- SYSUT2 is the primary output data set. It is composed of 80-character logical records containing the JCL statements and data records extracted from the master file.
- SYSPRINT contains a listing of the control statements and (optionally) a listing of the contents of the SYSUT2 data set.

The IEBEDIT program is executed as a job step; the EXEC statement used to call it specifies the program IEBEDIT.

PROGRAM STRUCTURE

The IEBEDIT program is contained in one load module whose entry point name is IEBEDIT. The module contains three major program sections as well as a number of subroutines. The three major sections of the program are:

- The initializing routine obtains main storage for tables and work areas, initializes them, and opens the program's data sets.
- The main routine passes control among the subroutines to analyze control statements, to inspect master file records, to determine which records should be written out, and to write output records.
- The postprocessing routine stores condition codes, frees main storage, closes the program's data sets, and returns control to the supervisor.

THE INITIALIZING ROUTINE

The entry point for the IEBEDIT program is the initializing routine. When it is entered, the routine obtains storage for an active save area and a work area, and opens the SYSPRINT, SYSUT1, SYSUT2, and SYSIN data sets.

The initializing routine checks the block size specification of each data set except SYSPRINT to ensure that it is a multiple of 80 characters. If the SYSUT2 block size specification is not a multiple of 80 characters, it is changed to match the SYSUT1 specification, and a message is written to SYSPRINT. If the SYSUT1 data set is not a multiple of 80 characters, a message is written to the SYSPRINT data set and the step is terminated.

If any data set cannot be opened, the initializing routine passes control (via a branch instruction) to the postprocessing routine. Otherwise, it uses the GET macro instruction (locate mode) to obtain the first SYSUT1 record, and branches to the main routine.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

THE MAIN ROUTINE

The main routine (Charts 01 and 02) passes control among subroutines that analyze control statements from the SYSIN data set and master file records from the SYSUT1 data set. Based on the specifications in the control statements, the main routine determines which records are to be extracted from the master file and uses the update subroutine to write those records to the SYSUT2 and (optionally) to the SYSPRINT data sets.

When the main routine is entered (via a branch from the initializing routine), the first record from the SYSUT1 data set is in main storage. The main routine uses the scan subroutine to obtain a record from the SYSIN data set, and to analyze the record.

If there are no control statements in the SYSIN data set, the scan subroutine encounters an end-of-data condition, indicating that a total copy of the master file is to be performed. Control is passed to the update subroutine to write the record to the output data sets, and then back to the main routine to get the next master file record. When the master file has been completely copied, the main routine passes control to the postprocessing routine.

If the scan subroutine obtains a control statement, a selective copy is performed, based on the specifications in the control statement. The main routine passes control to the startjob subroutine, which gets master file records until it finds the proper JOB statement:

- If the parameter START=jobname was used in the control statement, the startjob subroutine searches the master file for a JOB statement with the specified name.
- If no job name was specified, the startjob subroutine searches the master file for the next JOB statement.

When the proper JOB statement has been found, the startjob subroutine passes control to the update subroutine, which writes the statement to the SYSUT2 and, optionally, to the SYSPRINT data set. When control is returned to it, the startjob subroutine reads the next record and uses the cardtype subroutine to determine whether the record is a JES2 control statement or a JOBLIB DD statement.

If the record is a JES2 control statement or a JOBLIB DD statement, the update subroutine writes it to the output data sets. The startjob subroutine then obtains another master file record from the SYSUT1 data set and returns control to the main routine.

On return from the cardtype subroutine, the main routine analyzes the switches set by the cardtype subroutine and performs the processing indicated by the record type and control statement specifications.

If the record is an EXEC statement, its disposition depends on the use of the TYPE and STEPNAME parameters in the control statement.

If TYPE=POSITION, and no step name was specified, the main routine passes control to the update subroutine, and the record is written to the output data sets. If a step name was specified, and the corresponding EXEC statement is found, the main routine passes control to the update subroutine, and the record is written to the output data sets.

If TYPE=INCLUDE or EXCLUDE, the main routine must determine whether the current record represents a step within an inclusive set, and, if not, whether it represents a step whose name was specified singly. The routine does this with the aid of two tables (the inclusive step names table and the single step names table) and the inclusive/exclusive switches.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Each entry in the inclusive step names table contains the names of the first and last steps in a set as specified in the STEPNAME parameter; each entry in the single step names table contains the name of a step specified singly. The include/exclude switches indicate whether inclusive or exclusive processing is taking place.

The decisions made in the program, and the resultant processing, are shown in Figure 81 on page 197. The upper section shows the conditions that may exist; the lower section shows the action that is taken as a result of each set of conditions. The action "Write" means that the main routine uses the update routine to write the record containing the EXEC statement and the remaining records representing that step, to the output data sets. The action "No Write" means that the main routine searches for the end of the current step, but does not write the records to the output data sets.

The end of the current step is indicated by the presence of a JOB statement, another EXEC statement, or an end-of-data condition. If a DD DATA statement is encountered, a switch is set; subsequent records, although they may appear to be JCL statements, are treated as data in the input stream. When a delimiter statement is encountered, the DD DATA switch is set off, and, if the other records in the step were written out, so is the delimiter statement.

When a JOB statement is encountered, or when an end-of-data condition exists in the SYSUT1 data set, the main routine scans the list of step names constructed from the control statement. If any of the names in the list were not found, a message containing the step name is written to the SYSPRINT data set for each missing step. If a JOB statement was encountered, the main routine then passes control to the scan subroutine to analyze the next control statement; if there was an end-of-data condition, the main routine passes control to the postprocessing routine.

THE POSTPROCESSING ROUTINE

The postprocessing routine is entered when no more processing is to be performed, at end-of-data in the SYSUT1 data set, when all SYSIN statements have been processed, or when an unrecoverable I/O error occurs. When it is entered, it determines whether an end-of-data condition exists for the SYSIN data set; if not, it uses the scan subroutine to process the remaining control statements.

When all records in the SYSIN data set have been processed, the postprocessing routine uses the update subroutine to write a terminal message (including the condition code) to the SYSPRINT data set. It then closes the program's data sets, frees the main storage that had been obtained, and returns control to the supervisor.

IEBEDIT SUBROUTINES

The IEBEDIT program contains four major subroutines: scan, startjob, cardtype, and update. Linkage to each subroutine is via a BAL instruction; return is via a BR instruction.

The Scan Subroutine

The scan subroutine is entered to obtain and analyze a complete control statement, which is made up of the initial record and any continuation records. When the main routine is first entered, the scan subroutine determines whether a total copy is required; if not, and when a job has been processed, it determines the processing required for the next job. When an end-of-data condition occurs on the SYSUT1 data set, it is entered to scan the remaining SYSIN records.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Include/Exclude Switch is On	X	X	X	X						
Include/Exclude Switch is Off					X	X	X	X		
Match 1st Name in Inclusive Step Names Table							X	X		
Match 2nd Name in Inclusive Step Names Table	X	X								
No Match in Inclusive Step Names Table			X	X	X	X			X	X
Match in Single Step Names Table									X	X
No Match in Single Step Names Table		X	X	X	X	X	X	X		
TYPE=INCLUDE	X		X		X		X		X	
TYPE=EXCLUDE		X		X		X		X		X
Write	X		X			X	X		X	
No Write		X		X	X			X		X
Set Include/Exclude Switch On							X	X		
Set Include/Exclude Switch Off	X	X								

Figure 81. EXEC Statement Include/Exclude Processing

When the scan subroutine is entered, it attempts to obtain a record from the SYSIN data set. If it obtains a record, it scans the record, converting the control statement parameters to switch settings that can be tested by the main routine, and when it has processed the entire statement, returns control to its caller. If it encounters an end-of-data condition, and no statements have previously been processed, the routine sets a switch indicating that a total copy is to be performed, and returns control to its caller. If it encounters an end-of-data condition, and statements have previously been processed, it passes control to the postprocessing routine.

When the routine is entered, it uses a search routine to set pointers to the fields in the statement, then scans the field. The scan routine has five phases: initialization, name/operator handling, operand handling, operand value handling, and scan postprocessing.

The initialization phase clears switches and resets pointers; the search routine finds the name and operator fields, and control passes to the name/operator handling phase.

In the name/operator handling phase of the scan subroutine, the name field of the statement is checked for validity (it must be 8 characters or less). Then, the contents of the operator field is used as a search argument in a search of the operation code table (see Figure 82 on page 198). When a match is found, the Turn-On Box of the table is used to set the appropriate switches in the IEBEDIT work area, and pointers to the operator and to the appropriate parameter table (see Figure 83 on page 198) are placed in the work area.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

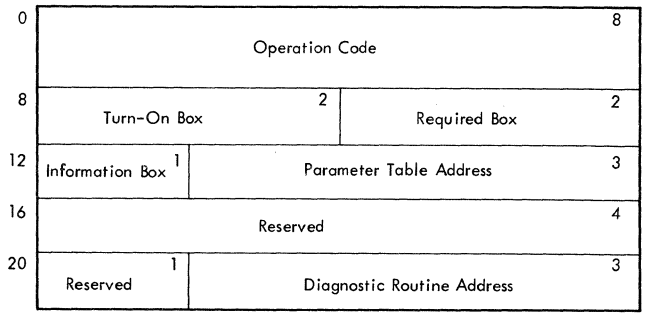


Figure 82. Scan Routine Operation Code Table Entry

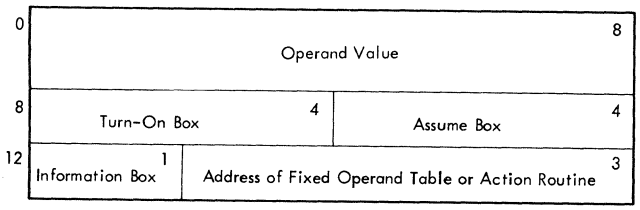


Figure 83. Scan Routine Parameter Table Entry

OPERATION CODE: This field contains the operation code, left aligned, and padded with blanks.

Turn-On Box: This field contains the displacement (byte 1) in the IEBEDIT work area and the bit pattern (byte 2) to be set at that displacement.

Required Box: This field contains a displacement (byte 1) in the IEBEDIT work area, and a bit pattern (byte 2) to be found at that displacement. This bit pattern is required for processing of this statement.

Information Box: If bit 0 of this field is set to 1, this entry is the last entry in the table.

Parameter Table Address: This field contains the address of the parameter table that corresponds to this operation.

Diagnostic Routine Address: This field contains the address of a routine used to perform additional processing on the statement.

Operand Value: This field contains the value of the operand, left aligned, and padded with blanks.

Turn-On Box: Byte 1 of this field contains a displacement in the IEBEDIT work area; byte 2 contains a bit pattern to be set at that displacement as a result of encountering this parameter.

Assume Box: Byte 1 of this field contains a displacement in the IEBEDIT Work Area; byte 2 contains a bit pattern to be set at that displacement if this parameter is omitted.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Information Box: The bits in this field have the following meanings when set to 1:

Bit	Meaning
0	Last entry in table
1	Fixed operand
2	Variable operand
3	Reserved
4	Allow subparameters
5	Keyword-only operand
6-7	Reserved

Address of Fixed Operand Table or Action Routine: If the operand is a fixed operand, this field contains the address of the appropriate fixed operand table entry; if the operand is a variable operand, this field contains the address of the routine that is to process the operand.

Each operand in turn is used as a search argument, in the operand handling phase, to scan the parameter table. When a match is found, the turn-on box of the parameter table is used to set the appropriate switches in the IEBEDIT work area, and a pointer to the parameter table entry is placed in the work area. If the operand is a keyword-only operand and there are additional operand fields, the routine processes the next field. If there are no additional operands, the routine passes control to the scan postprocessing phase.

If there are parameters associated with the keyword, the routine passes control to the operand value handling phase. In this phase, the scan subroutine inspects the parameter table entry to determine whether the parameter has a fixed value, or whether the value may vary. If the parameter is a variable value parameter, the action routine address field of the parameter table entry contains the address of the routine that is to process the parameter, and a branch is executed to give that routine control. If the parameter is a fixed value parameter, the routine uses the value specified as a search argument in a search of the fixed operand table (see Figure 84). When a match is found, the turn-on box field of the table is used to set the appropriate switches in the work area.

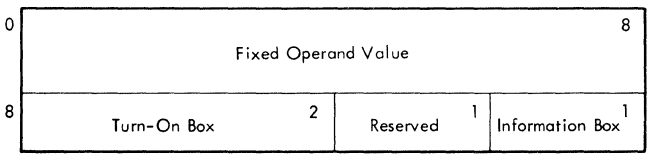


Figure 84. Scan Routine Fixed Operand Table Entry

Fixed Operand Value: This field contains the value of the operand, left aligned, and padded with blanks.

Turn-On Box: Byte 1 of this field contains a displacement in the IEBEDIT work area; byte 2 contains the bit pattern to be set at that displacement when this operand is encountered.

Information Box: If bit 0 of this field is set on, it indicates that this entry is the last entry in the table.

When the parameters associated with a keyword have been processed, control is passed to the operand handling phase to process the next operand; if there are no more operands to process, control is passed to the scan postprocessing phase.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

When a complete statement has been processed, the scan postprocessing phase scans the parameter table for the current operator, and then sets the assumed (default) value switches for any parameters not supplied. The current operation code table entry is then inspected to determine whether any diagnostic routine has been supplied. If so, the diagnostic routine is given control, and, when its processing is complete, the scan routine returns control to its caller.

The Startjob Subroutine

The startjob subroutine is entered from the main routine with the first record of a master file statement in the buffer. It uses the cardtype subroutine to determine the statement type, and it uses the update subroutine to write a JOB statement, a JES2 control statement, and a JOBLIB DD statement to the output data sets.

If the first statement encountered by the startjob subroutine is not a JOB statement, the routine gets records from the master file until it finds a JOB statement. The startjob subroutine then determines whether the START=jobname parameter was used, and, if not, uses the update subroutine to write the statement (including its continuations) to the output data sets.

If START=jobname was specified, the routine compares the specified job name to the name in the JOB statement. If they are not equal, the routine searches the master file until the proper JOB statement is found. In either case, the JOB statement having the specified name is written to the output data sets, and the startjob subroutine reads the next master file record.

Once a JOB statement has been written out, the startjob routine looks for a JOBLIB DD statement. If it encounters one, the routine uses the update subroutine to write the statement to the output data sets; if the next statement is not a JES2 control statement or a JOBLIB DD statement, the startjob subroutine returns control to its caller.

The Cardtype Subroutine

The cardtype subroutine classifies 80-character records by type. It stores a code for each type except system input data records, and, if the record is a JOB or EXEC statement, it stores the statement name. When it has analyzed a record, it returns control to its caller.

The routine first examines the first two positions of the record. The characters "//" indicate that the record is a JCL statement, and the routine performs further analysis. The characters "/*" indicate that the record is a delimiter statement; the routine determines whether the statement is continued by checking for a nonblank character in position 72, and then returns to its caller.

If the statement is a JCL statement, the routine classifies it as one of the following types:

- JOB Statement: The statement is a JOB statement if the operation field contains JOB.
- JOBLIB DD Statement: A statement is a JOBLIB DD statement if the name field contains JOBLIB and the operation field contains DD.
- EXEC Statement: The statement is an EXEC statement if the operation field contains EXEC.
- DD Statement: The statement is an DD statement if the operation field contains DD.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- DD DATA Statement: A statement is a DD DATA statement if it is a DD statement, and the first operand field contains DATA.
- JES2 Control Statement: A statement is a JES2 control statement if it begins with a /* and has a nonblank character in column 3.
- Continued Statement: A statement is a continued statement if it is a JCL statement or a delimiter (/*) statement, and if it has a nonblank character in position 72.

The Update Subroutine

The update subroutine is a control routine for the output functions of the IEBEDIT program. It contains the put routine, which writes records to the SYSUT2 data set, and the print routine, which writes records to the SYSPRINT data set. There are two entry points to the update subroutine:

- UPDATE is the entry point used to write records to the SYSUT1 and, optionally, to the SYSPRINT data set.
- PRINT is the entry point used to write records to the SYSPRINT data set.

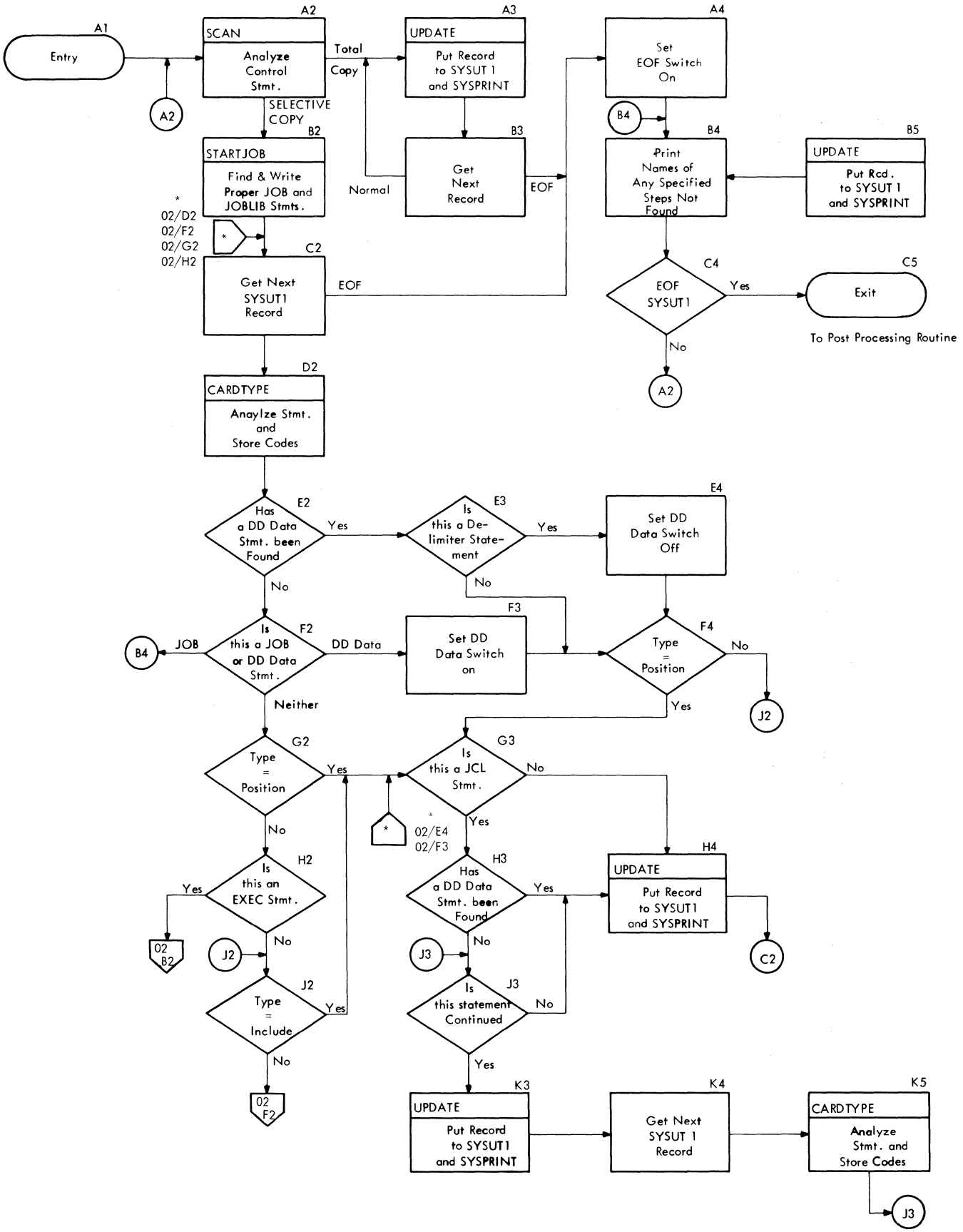
When it is entered at the UPDATE entry point, the routine inspects the first three positions of the record in the buffer. If it finds the characters period, period, asterisk (..*), it substitutes the characters slash, asterisk (/*); in either case, it branches to the put routine.

The put routine contains the PUT macro instruction, which causes the record to be written to the SYSUT2 data set. When the PUT macro instruction has been executed, the routine determines whether NOPRINT was specified, and, if so, returns control to the caller. If NOPRINT was not specified, the routine branches to the PRINT entry point of the routine.

When it is entered at the PRINT entry point, the routine is given the address of a record or a message code. It issues the PUT macro instruction to write the record or message to the SYSPRINT data set, and then returns control to the caller.

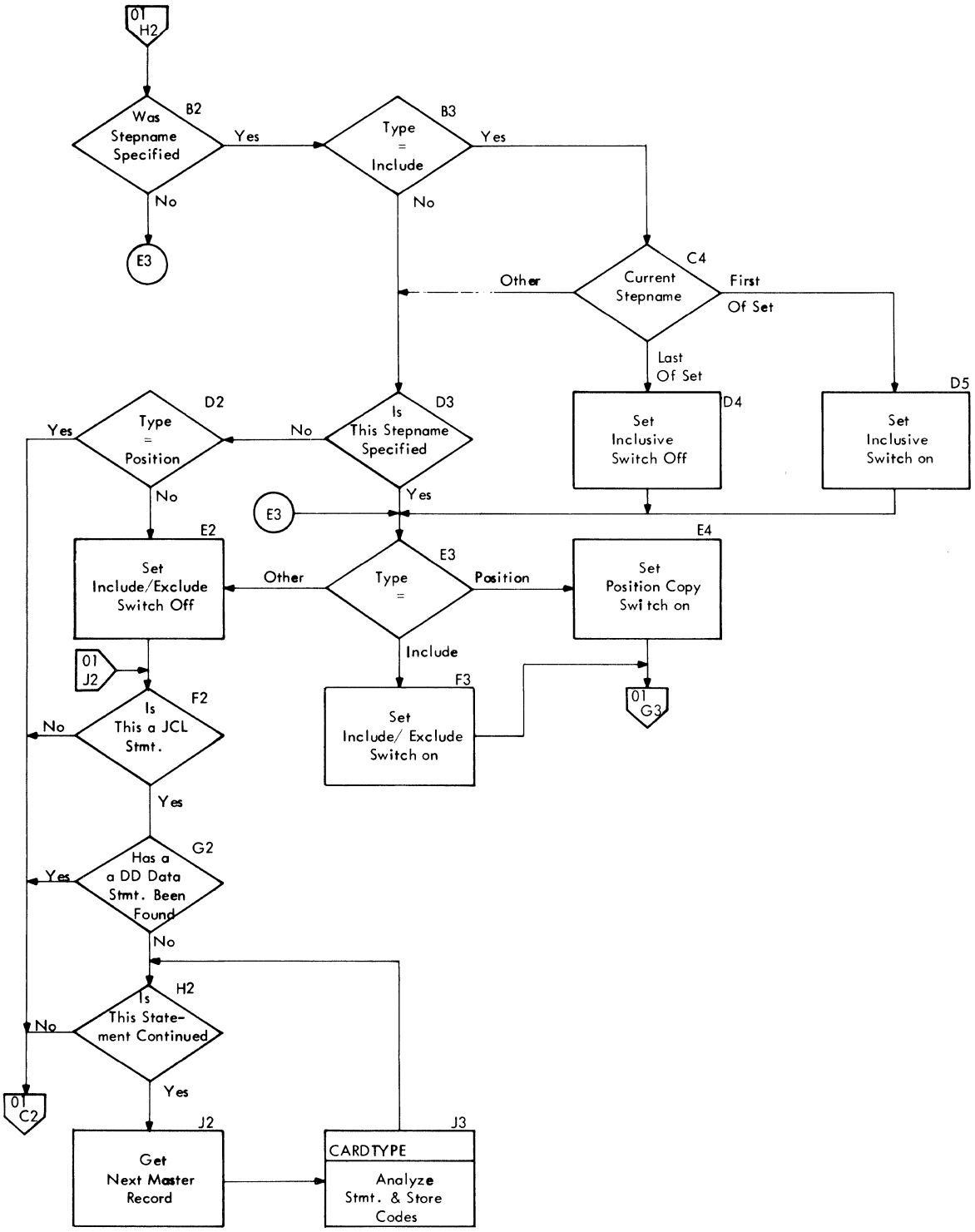
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 01. MAIN ROUTINE (PART 1 OF 2)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 02. MAIN ROUTINE (PART 2 OF 2)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

USING THE IEBGENER DOCUMENTATION

This chapter is in four sections: "The IEBGENER Program," "Program Structure," "Method of Operation," and "Tables, Switches, and Work Areas."

"The IEBGENER Program" provides a general description of the program. This overview concentrates on the uses and purposes of the program, and the inputs and outputs of the program.

"Program Structure" describes the physical organization of the program. It describes the function of each assembly module and shows how assembly modules are combined into a load module.

"Method of Operation" describes the IEBGENER program in terms of the operations it performs. Operation diagrams at two levels of detail show the movement of data through the tables, work areas, parameter lists, and data sets used by the program, and describe the processing performed at each stage. Supporting text expands on the information in each operation diagram and explains the operation in more detail.

Operation diagrams are not flowcharts. (Flowcharts are provided in another section.) An operation diagram shows where information comes from, what is done with it, and where the results are stored. Notes in the diagrams themselves indicate the names of the routines involved with the operation.

If you are already familiar with the program, the operation diagrams will act as a recall mechanism; if you are learning the program, the operation diagrams and the supporting text will explain the whole program (or the part you are interested in) quickly and graphically.

The first operation diagram presents an overview of the IEBGENER program. The diagram is divided into parts, each with an identifying number and a characteristic "shape." In the remainder of the section, each part is described in greater detail and identified by its corresponding number.

"Tables, Switches, and Work Areas" contains detailed descriptions of the format and content of the major tables, switches, and work areas used by the program if they are referred to in more than one place in the text. Other tables, work areas, and parameter lists are described where they are mentioned in the text. An exception is the "Constants and Switches" area of each module, which is documented in the listing. Where fields shown in diagrams or mentioned in the text are in such an area, the area is also mentioned.

The IEBGENER flowcharts consist of a chart of each of the major program modules. Each chart shows program labels where possible, so that you should be able to see the general logic of the module from the chart and find the details in the listing.

Appendix A contains a brief description of each module in the program.

THE IEBGENER PROGRAM

IEBGENER is a data set utility program used to copy either a sequential data set or a member of a partitioned data set, or to expand a partitioned data set by adding members to it. In addition, the program can edit records by inserting literals,

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

rearranging fields, and performing data conversion on specified fields.

The primary input to the IEBGENER program is the data set to be copied. This data set must be sequential and it may consist of fixed-length, variable-length, variable-length spanned, or undefined format records. Logical records may be blocked or unblocked.

The other inputs to the IEBGENER program are the control data set, which contains utility control statements and optional data set label records, and the user parameter lists. If the IEBGENER program is invoked by a user's program, the user parameter lists may specify alternatives for the standard ddnames, and may specify the first page number for the message data set.

The IEBGENER program produces two data sets: the output data set, which may be sequential or partitioned, consisting of records from the input data set as edited by the program; and the message data set, containing warning and error messages and images of the records in the control data set.

PROGRAM STRUCTURE

The IEBGENER program consists of five program segments, which form the structure shown in Figure 85 on page 206. The program is entered at the root segment, which contains the initializer (CSECT IEBGENER) and the message module (CSECT IEBGMESG). The initializer stores user parameters; opens, writes, and closes the message data set; and passes control to the control data set processing segment and the record processing segment. The message module contains all messages issued by the IEBGENER program.

The control data set processing segment is entered from the root segment to process the control data set. It contains the control statement parameter processor (CSECT IEBGSCAN), and the scan routine (CSECT IEBCCS02). The control statement parameter processor obtains main storage for tables and constructs the tables on the basis of control statement parameters. The scan routine is a subroutine of the control statement processor. It opens the input data set, scans control data set records, closes the input data set, and passes pointers to the fields in those records to the control statement parameter processor. The record processing root segment is entered when the control data set has been processed. It contains the record processing initializer (CSECT IEBGENR3). The record processing initializer opens and closes the input and output data sets, loads and executes the user's (optional) label processing routines, and passes control to either the spanned record processing segment or the nonspanned record processing segment.

The spanned record processing segment is entered when either the input or the output data set consists of spanned records. If neither data set consists of spanned records, the nonspanned record processing segment is used.

With the exception of the record processor routines, both the spanned and nonspanned record processing segments contain the same routines. The spanned record processing segment contains the spanned record processor (CSECT IEBGENS3), and the nonspanned record processing segment contains the nonspanned record processor (CSECT IEBGEN03). The record processors read input data set records, execute user exit routines, data movement routines, and conversion routines, and write output data set records.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

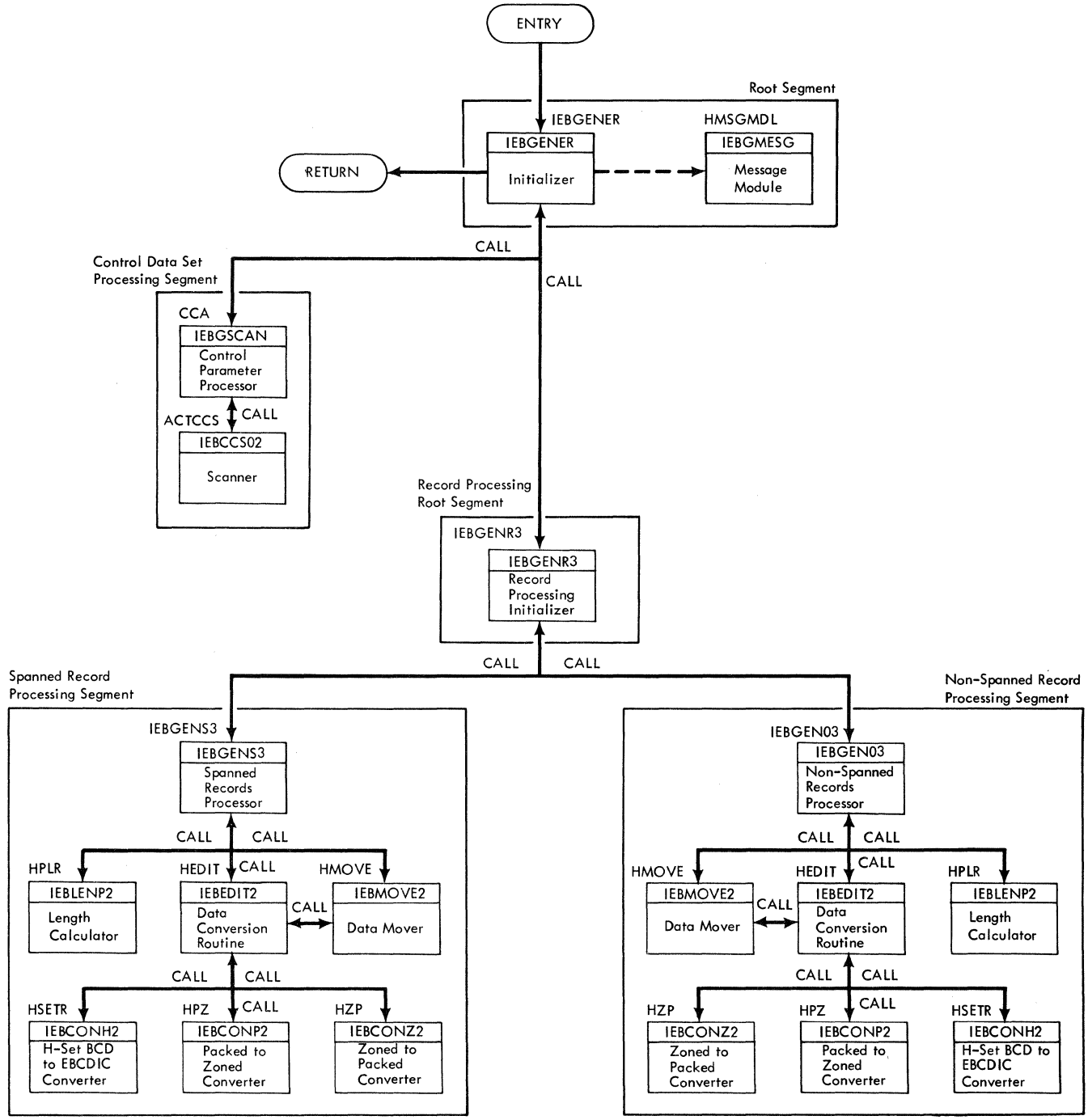


Figure 85. Overlay Structure

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

The common routines in the record processing segments are:

- The length calculator (CSECT IEBLENP2), which is entered at entry point HPRL via a CALL macro instruction issued in the record processor. The routine calculates the length of the output record and returns control to its caller.
- The data mover (CSECT IEBMOVE2), which is entered at entry point HMOVE via a CALL macro instruction issued in the record processor or in the data conversion control routine. It moves a record or a field between internal buffers and returns control to its caller.
- The data conversion control routine CSECT IEBEDIT2), which is entered at entry point HEDIT via a CALL macro instruction in the record processor. The routine uses data conversion routines and the data mover routine to convert the fields in the input record and move them to the output buffer; it stores literals in the output record. When it has processed the entire record, it returns control to its caller.
- The H-set BCD to EBCDIC converter (CSECT IEBCONH2), which is entered at entry point HSETR via a CALL macro instruction in the data conversion control routine. It converts a field from H-set BCD (6-bit format) to EBCDIC (8-bit format), and returns control to its caller.
- The packed to zoned converter (CSECT IEBCONP2), which is entered at entry point HPZ via a CALL macro instruction in the data conversion control routine. It converts a field from packed decimal to zoned decimal, and returns control to its caller.
- The zoned to packed converter (CSECT IEBCONZ2), which is entered at entry point HZP via a CALL macro instruction in the data conversion control routine. It converts a field from zoned decimal to packed decimal and returns control to its caller.

METHOD OF OPERATION

The IEBGENER program performs four basic operations. These operations and the initial invocation of the program are described in this section.

Figure 86 on page 208 provides an overview of the operation of the IEBGENER program. The four basic operations, denoted by the large circled numbers in the diagram, are described under the corresponding numbered headings in this section. The invocation of the program and the basic operations are described in the remainder of the section.

STORING DDNAMES

The initializer (CSECT IEBGENER) stores the ddnames in the alternate ddname list in HDDNAMES; it also stores the alternate for "SYSPRINT" in the message data set DCB. The control parameter processor (CSECT IEBGSCAN) moves the alternate name for "SYSIN" to a field name CSIDDA; the scan routine (IEBCCS02) stores the name in the control data set DCB. The record processing initializer (CSECT IEBGENR3) stores the alternate names for "SYSUT1" and "SYSUT2" in the input and output data set DCBs.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

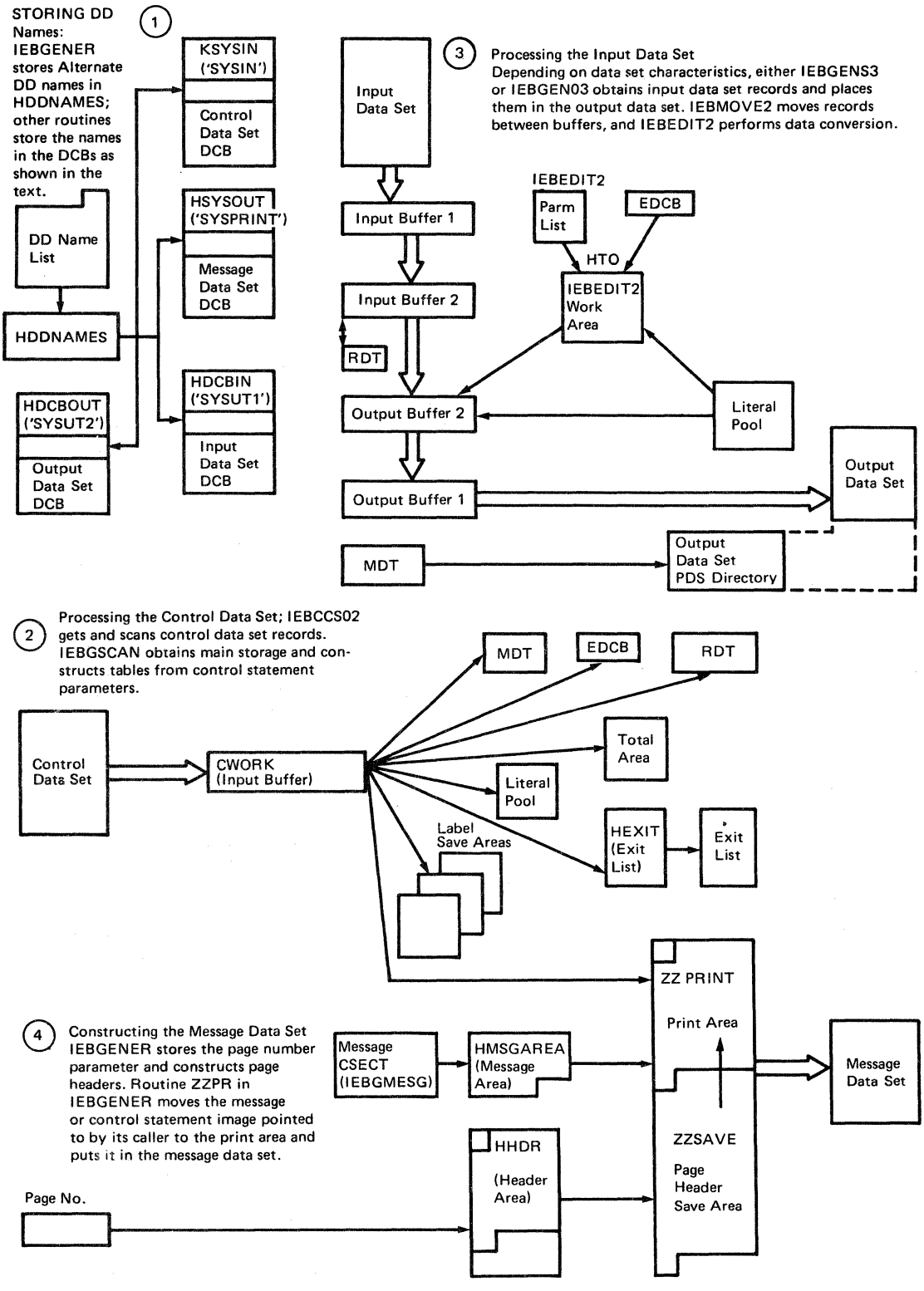


Figure 86. The IEBGENER Program

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

PROCESSING THE CONTROL DATA SET

The utility control statement records, which describe the processing to be performed by the IEBGENER program, and label records, which become labels for the output data set, are in the control data set. The scan routine (CSECT IEBCCS02) opens the control data set (defined by the SYSIN DD statement), reads and scans the records, and passes pointers to the fields to the control statement parameter processor (CSECT IEBGSCAN). The control statement parameter processor obtains main storage for the member description table (MDT), the record description table (RDT), the literal pool, the edit control block (EDCB), the total area, and the label save areas. It stores pointers to these areas in its exit list and stores the control statement parameters in the areas. The initializer (CSECT IEBGENER) obtains main storage for the IEBGSCAN exit list and stores the contents of the list in the area.

PROCESSING THE INPUT DATA SET

The record processing initializer (CSECT IEBGENR3) opens the input and output data sets. The spanned record processor (CSECT IEBGENS3) or the nonspanned record processor (CSECT IEBGEN03) controls the processing, depending on whether spanned records are to be processed.

The record processors use the RDT to identify the last record in a group of records that are to be edited in the same way. The editing (for example, data conversion, rearrangement of fields, and insertion of literals), is controlled by the data conversion control routine (CSECT IEBEDIT2) on the basis of the information in the EDCB. The data mover routine (CSECT IEBMOVE2) moves records and fields between buffers. When a member has been written in the output data set, the record processor uses the names in the MDT to update the partitioned data set directory.

Although the operation diagram shows four buffers, the buffering technique used depends on the characteristics of the data sets and on whether editing is specified. IEBGEN03 uses BSAM and four buffers. IEBGENS3, which is executed when spanned records are to be processed, uses QSAM and two buffers if editing is to be performed, or BSAM and three buffers if no editing is to be performed.

CONSTRUCTING THE MESSAGE DATA SET

If the user supplies a page number parameter, IEBGENER stores the number in the header area. It stores the complete page header in the header save area, and updates the page number each time a page is printed.

All message data set records are written by the routine labeled ZZPR in IEBGENER. When the routine is entered, register 0 contains the message length, and register 1 contains the location of the message. The routine obtains messages from the message module (CSECT IEBGMESG), and control record images from the control data set buffer (CWORK). It uses the page header in the header save area each time it starts a new page.

INVOKING IEBGENER

The IEBGENER program may be invoked either by the job scheduler or by a user's program. In either case, the user must supply job control language (JCL) statements that define the data sets used in the IEBGENER program and that cause the program to be invoked.

If IEBGENER is to be invoked by the job scheduler, the user must use the standard ddnames (SYSIN, SYSUT1, SYSUT2, and SYSPRINT), and must specify IEBGENER (or a procedure that executes IEBGENER) in the EXEC statement. The job scheduler passes

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

control to IEBGENER with register 1 pointing to a 4-byte field containing X'80' in the high-order byte.

If IEBGENER is to be invoked by a user's program, the user may specify alternates for any of the standard ddnames, and may also specify the page number for the first page of the message data set. The user's program passes control to IEBGENER via a LINK or ATTACH macro instruction, after it has constructed the appropriate parameter lists.

For additional information about invoking utility programs, see Utilities.

THE PARAMETER LISTS

There are three kinds of parameter lists that may be constructed by a program before it invokes the IEBGENER program. These parameter lists are shown in Figure 87 on page 211 and described in this section.

The User Parameter List: consists of one, two, or three entries of 4 bytes each. The first must be set to zero, and the last entry must contain X'80' in the high-order byte.

The second entry contains X'00' in the high-order byte and either a pointer to the ddname parameter list or zeros.

The third entry in the user parameter list is present only if the page parameter list is present. If the third entry is present, it contains X'80' in its high-order byte and a pointer to the page header parameter list in its three low-order bytes.

The Option Parameter List: IEBGENER does not use this list but the high-order byte must contain X'00'.

The ddname Parameter List: contains the specified alternates for the standard ddnames. The order in which entries appear is fixed. As shown in Figure 87, the 2-byte (unpacked decimal) length field is followed by 32 bytes of binary zeros. The remainder of the list is a series of 8-byte entries in the following order: the alternate for "SYSIN"; the alternate for "SYSPRINT"; binary zeros; the alternate for "SYSUT1"; and the alternate for "SYSUT2". Entries may be omitted from the end of the list (and the length field adjusted accordingly), but entries within the list must contain either a name or binary zeros.

The Page Header Parameter List: is a 6-byte field containing the length (in unpacked decimal) in the high-order 2 bytes, and the first page number parameter (in unpacked decimal) in the low-order 4 bytes.

Both the ddname parameter list and the page header parameter list are optional. Their presence and location are indicated by the user parameter list.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

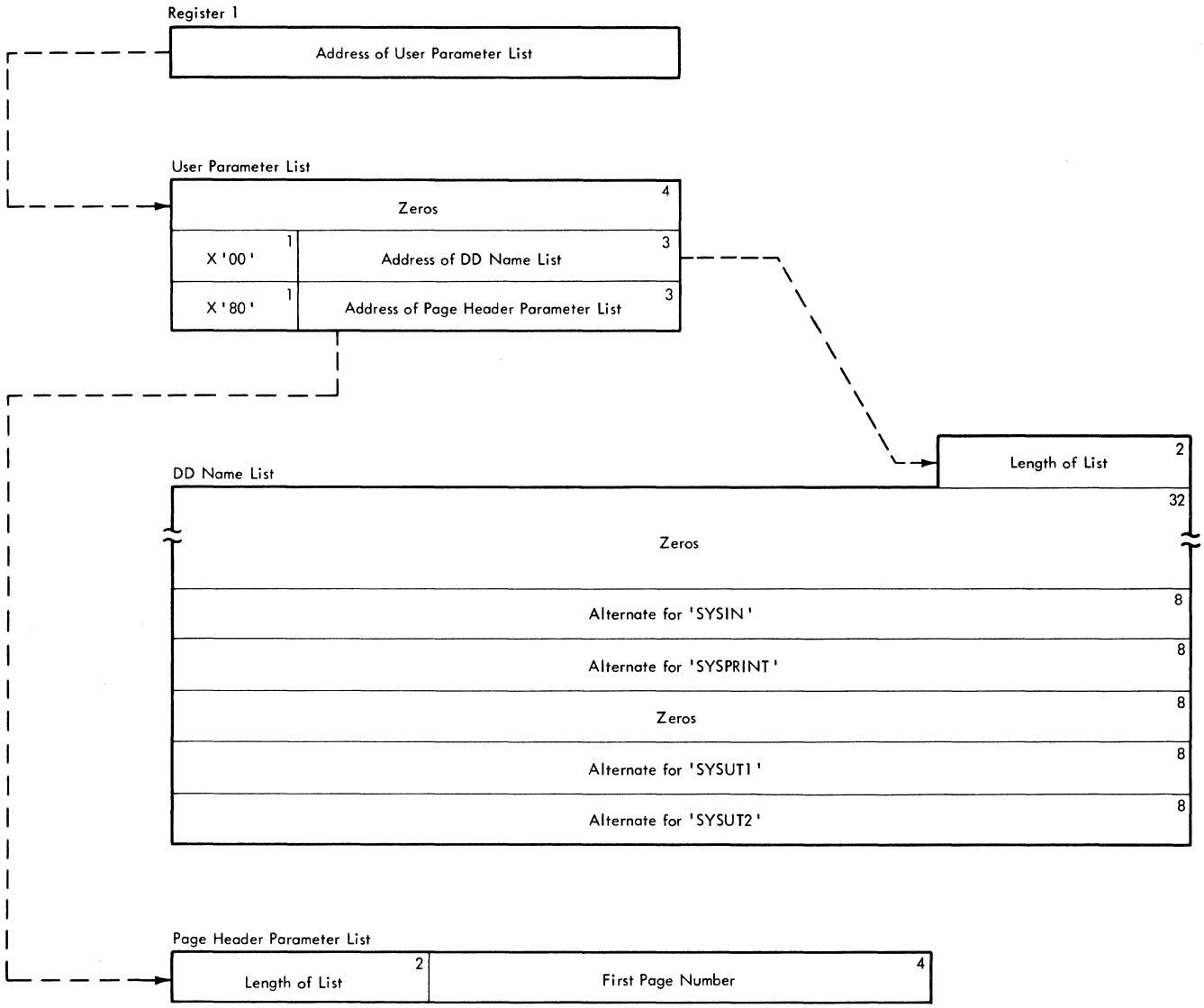


Figure 87. Input Parameter Lists

STORING ALTERNATE DDNAMES

If IEBGENER is invoked by a user's program, the user may supply alternates for the standard ddnames. If alternate ddnames are supplied, they must be in the ddname list, which is pointed to by the second word of the user parameter list. IEBGENER stores any alternate ddnames that are supplied, as shown in Figure 88 on page 212.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

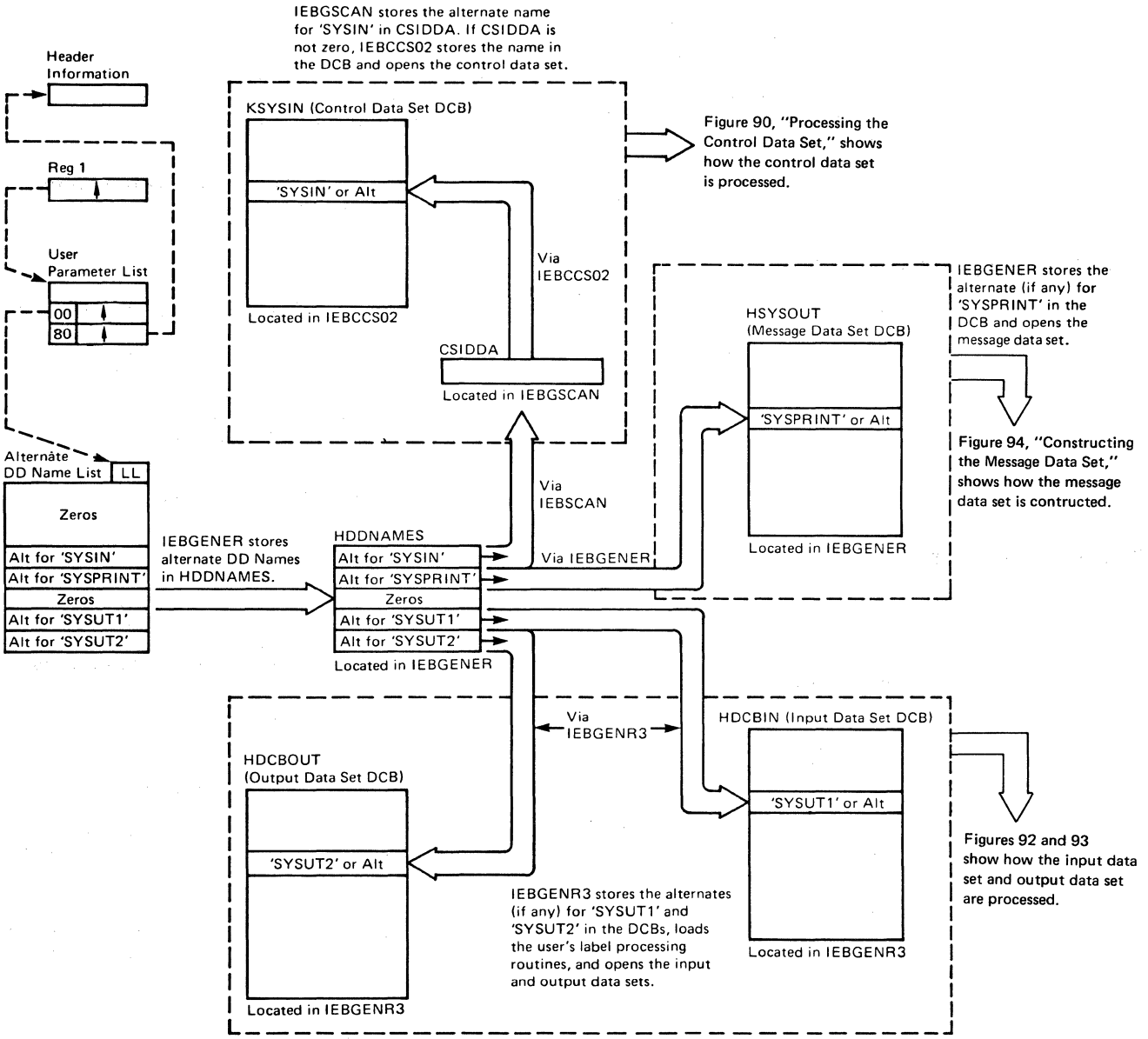


Figure 88. Storing Alternate ddnames

When it is entered, the initializer (CSECT IEBGENER) determines if there is a ddname list. If so, the routine stores the alternate ddname (the nonzero entries) in an area named HDDNAMES. In addition, if an alternate name for SYSPRINT was supplied, the initializer stores the alternate in the message data set DCB, named HSYSOUT, located in CSECT IEBGENER.

The control parameter processor (CSECT IEBSCAN) moves the first 8 bytes of HDDNAMES to a field named CSIDDA in its constants and switches area. The first time it passes control to the scan routine (CSECT IEBCCS02), the scan routine determines whether CSIDDA contains zeros or an alternate for SYSIN. If CSIDDA contains an alternate name, the scan routine stores the name in

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

the control data set DCB. The DCB, named KSYSIN, is located in CSECT.

The record processing initializer (CSECT IEBGENR3) determines whether alternates for SYSUT1 and SYSUT2 have been supplied by testing the appropriate HDDNAMES fields for zeros. If the names are supplied, the routine stores them in the input and output data set DCBs. The input data set DCB is named HDCBIN, and the output data set DCB is named HDCBOUT; both are located in CSECT IEBGENR3.

In all cases, unless replaced by an alternate name, the standard ddnames are compiled into the DCBs to be used.

PROCESSING THE CONTROL DATA SET

Utility control statements in the control data set describe the processing to be performed on the input data set by the IEBGENER program. Label records in the control data set are placed in the output data set as data set labels. This section describes the interface between the control statement parameter processor (CSECT IEBGSCAN) and the scan routine (CSECT IEBCCS02), which is a subroutine of the control statement parameter processor. The scan routine reads and scans control data set records, and the control statement parameter processor stores control statement parameters and label records in various tables and work areas.

THE SCAN/PROCESSOR INTERFACE

The scan routine (CSECT IEBCCS02) is a subroutine of the control statement parameter processor (CSECT IEBGSCAN). Figure 89 on page 214 shows the interface between the two routines. Figure 90 on page 215 shows how the control statement parameter processor stores parameters.

Each time it is ready to process a command (for example, GENERATE, EXITS, RECORD), a keyword, or a parameter, the control statement parameter processor passes control to the scan routine via a CALL macro instruction, along with the parameter list shown in the upper left of Figure 89. The scan routine opens the control data set the first time it is entered, uses the GET macro instruction to obtain the first record, and scans for the command field. On each subsequent entry, the scan routine scans another field, obtaining a new record whenever necessary and closing the control data set at end-of-data. Each time it has scanned a field, the scan routine returns control to the control statement parameter processor and provides the parameter list shown in the upper right of Figure 89.

In addition to the parameter list, the two routines communicate via a work area (named CWORK) also shown in Figure 89. This work area is described in "Tables, Switches, and Work Areas" on page 222.

STORING THE PARAMETERS

As shown in Figure 90, IEBGSCAN obtains main storage for tables, stores their addresses in its list (named HEXIT), and constructs the tables and exit list from the control statement parameters.

When it processes a GENERATE statement, IEBGSCAN obtains main storage for the following:

- A member description table (MDT), if MAXNAME is specified.
- An edit control block (EDCB), if MAXFLDS is specified.
- A record description table (RDT), if MAXGPS is specified.
- A literal pool, if MAXLITS is specified.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

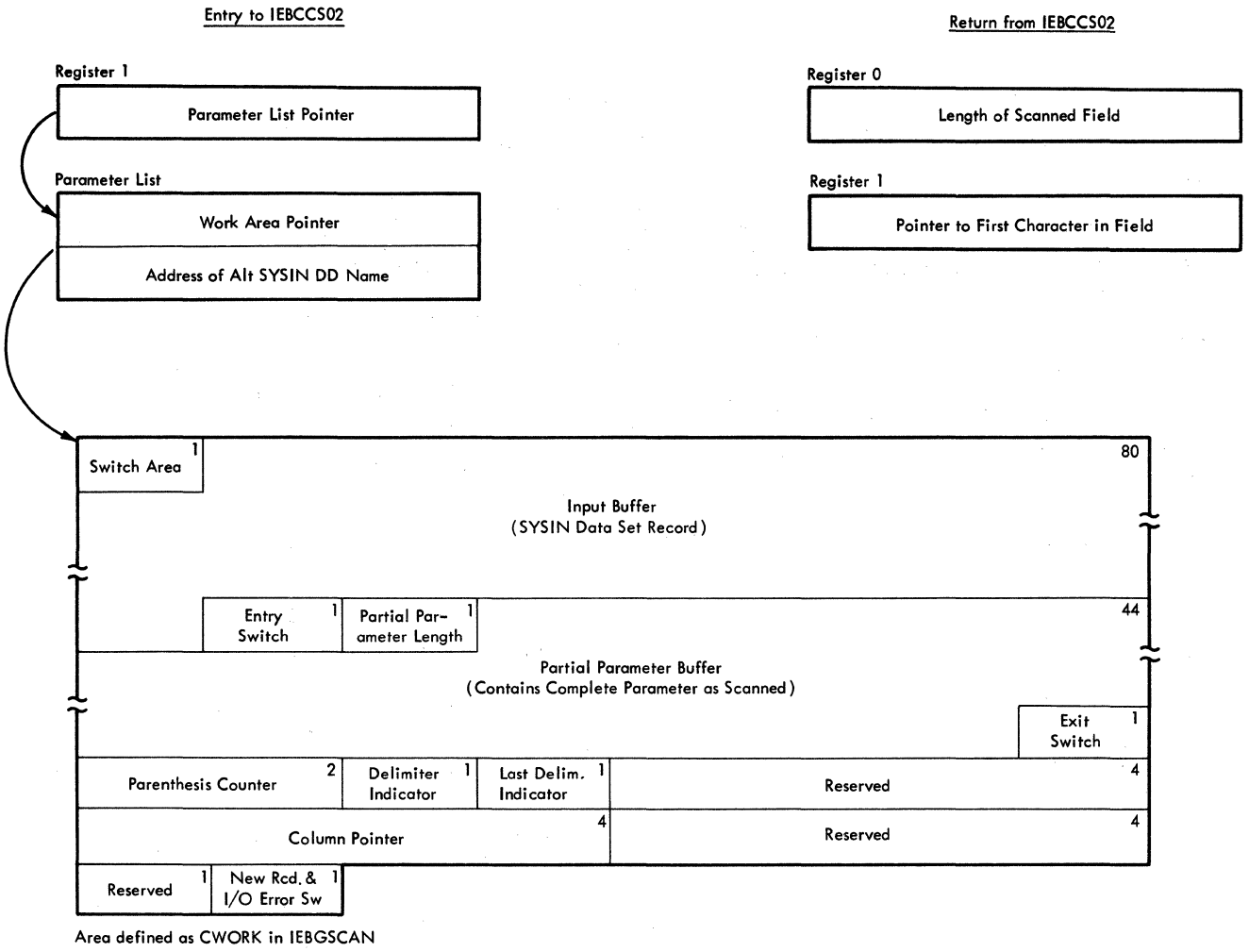


Figure 89. Scan/Processor Interface

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

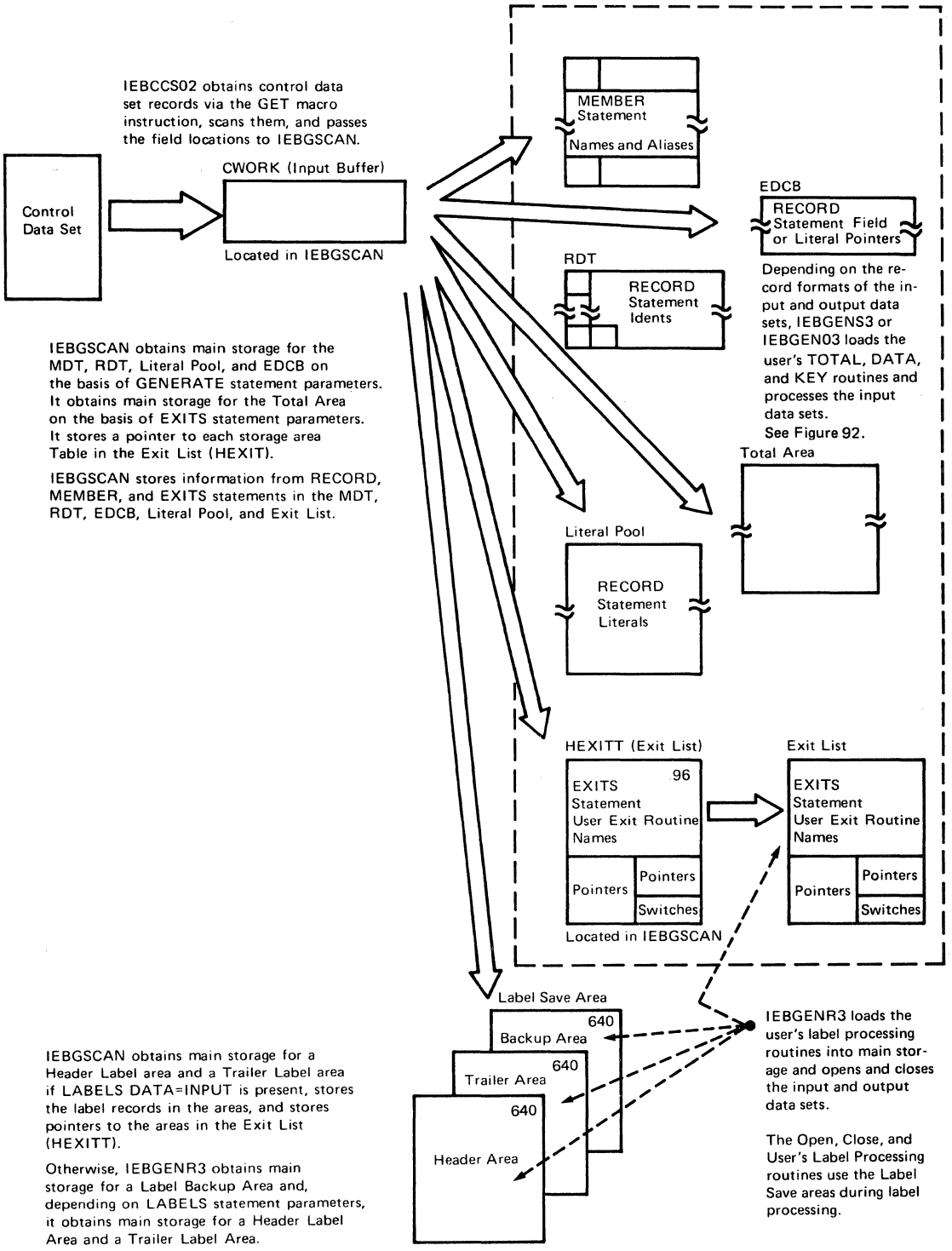


Figure 90. Processing the Control Data Set

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

If it encounters an EXITS statement specifying TOTAL, the routine also obtains main storage for a totaling area.

Main storage for three 640-byte label areas (a header label area, a trailer label area, and a backup area) may be obtained by IEBGSCAN or by the record processing initializer (CSECT IEBGENR3), or by both. The storage is obtained on the basis of the LABELS statement DATA parameter as shown below:

DATA Parameter	Action
INPUT	IEBGENR3 obtains two areas; IEBGENR3 obtains the backup area.
YES	IEBGENR3 obtains three areas.
ALL	IEBGENR3 obtains three areas.
ONLY	IEBGENR3 obtains two areas.
NO	No label areas are obtained.

IEBGENER constructs the tables and its exit list as described below:

- If it encounters an EXITS statement, the routine stores the exit routine names in the exit list.
- If it encounters a MEMBER statement, the routine stores the member names and aliases in the MDT.
- If it encounters a RECORD statement, the routine stores the IDENT parameters in the RDT and the FIELD parameters in the EDCB. If a FIELD parameter contains a literal, the routine stores the literal in the literal pool.
- If it encounters label records (which must follow a RECORD statement specifying LABELS=n), the routine stores the labels in the label area. All label records following the second such RECORD statement are assumed to be trailer labels.

IEBGENER obtains a 96-byte area of main storage, stores a pointer to the area in register 1, and moves the contents of the IEBGSCAN exit list into the area. Thus the user's exit routine names, pointers to the other tables, and switches are available to subsequent processing segments.

IEBGENR3 loads the user label processing routines specified in the exit list, and the record processor (IEBGENS3 or IEBGEN03) loads the remaining routines.

PROCESSING THE INPUT DATA SET

The IEBGENER program can process a sequential data set or a member of a partitioned data set containing fixed-length, variable-length, variable-length spanned, or undefined format records that are either blocked or unblocked. It can copy the input data set, or it can edit the records by converting the data form of selected fields, by rearranging fields or by inserting literal information. This section describes the access methods and the buffering and editing techniques used by the IEBGENER program.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

BUFFERING TECHNIQUES

Depending on the characteristics of the input and output data sets, the IEBGENER program may use BSAM or QSAM, and it may use two, three, or four buffers. Figure 92 on page 218 shows the arrangement of buffers and the access methods used by the program. Figure 91 is a decision table that shows the criteria for selecting the access method and buffering technique, and directs you to the proper section of the operation diagram.

Input data set RECFM=VS	Y	Y	Y	N	Y	N
Output data set RECFM=VS	Y	Y	Y	Y	N	N
Data set characteristics equal	Y	Y	N	-	-	-
Editing specified	N	Y	-	-	-	-
See Figure 92, Part ⟶	B	B	A	A	A	C

Figure 91. Selecting a Buffering Technique

If either or both of the data sets consist of spanned records, and if either the characteristics of the data sets are different or if editing is specified, the spanned record processing segment uses QSAM with two buffers. This is illustrated in part A of Figure 92.

At INOUTSET, the spanned record processor (CSECT IEBGENS3) obtains main storage for an input and an output buffer, each having a length equal to the logical record length (LRECL). The processor uses a GET macro instruction (at HINP1A1) to bring each logical record, one at a time, from the input data set into the input buffer. At label HOUP3B, the processor uses a PUT macro instruction to move each record from the output buffer to the output data set.

If no editing is specified and the characteristics are unequal, IEBMOVE2 moves logical records from the input buffer to the output buffer. If editing is specified, the processing proceeds as described under "Editing the Input Data Set," following.

If the characteristics of both data sets are the same, if the data sets consist of spanned records, and if no editing is specified, the spanned record processing segment uses BSAM with three buffers (see part B of Figure 92).

At HGENP5B, the spanned record processor (CSECT IEBGENS3) obtains main storage for three buffers of BLKSIZE bytes each. It uses a READ macro instruction at HREAD to bring physical records from the input data set into buffer 1, and a WRITE macro instruction at HWRITE to move physical records from buffer 3 to the output data set. IEHMOVE2 moves physical records from buffer 1 to buffer 2, and from buffer 2 to buffer 3.

If neither data set contains spanned records, the nonspanned record processing segment uses BSAM with two input and two output buffers (see part C of Figure 92).

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

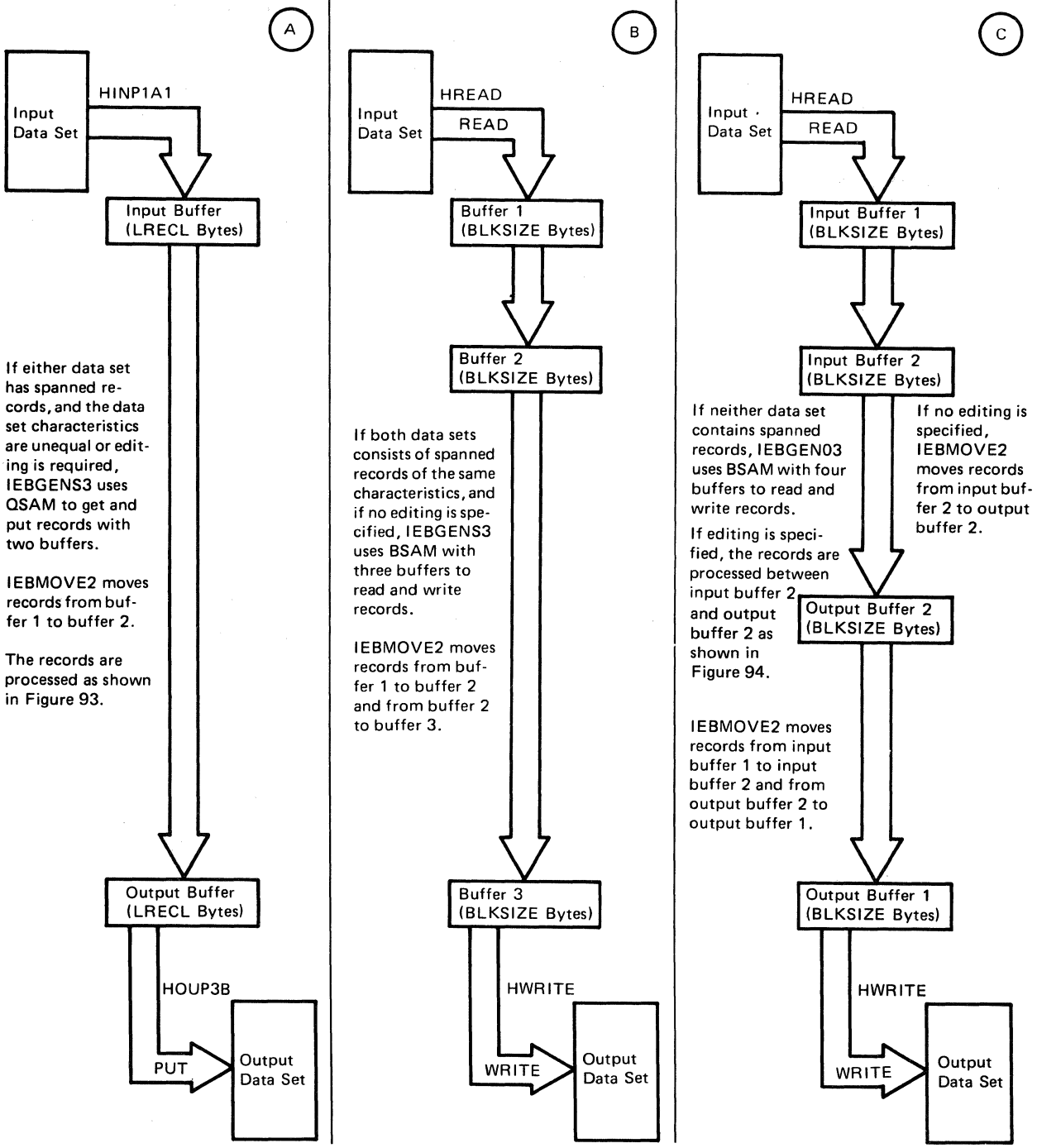


Figure 92. Input and Output Data Set Buffering

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

At INOUTSET, the nonspanned record processor (CSECT IEBGEN03) obtains main storage for four buffers of BLKSIZE bytes each. It uses a READ macro instruction at HREAD to bring physical records from the input data set into buffer 1, and a WRITE macro instruction at HWRITE to move records from output buffer 1 to the output data set. IEHMOVE2 moves physical records from input buffer 1 to input buffer 2, from output buffer 2 to output buffer 1, and if no editing is specified, from input buffer 2 to output buffer 2. If editing is specified, the records are processed as described under "Editing the Input Data Set," following, before they are moved to output buffer 2.

EDITING THE INPUT DATA SET

The IEBGENER program edits the input data set in three ways: It converts fields from one data format to another; it rearranges fields; and it inserts literal data. In addition, because the program can edit different groups of records according to different specifications, it has the ability to identify the records in a group. Finally, because the program can create or add members to a partitioned data set, it also updates the PDS directory. The IEBGENER program performs its editing functions (as shown in Figure 93 on page 220) based on information in tables created from control data set records.

The IEBGENER program identifies the records in a record group by using the record description table (RDT). Each RDT entry contains an identification field and the location and length of the corresponding field in the records of a group. The record processor (IEBGENS3 or IEBGEN03) compares the identification field in each record with the identification field in the current RDT entry. An equal comparison indicates that the record is the last in the group; the routine processes the record (as it does the other records in the group) according to the specifications in the current edit control block entries, and then uses the next RDT to find the last record in the next group.

Each edit control block (EDCB) entry corresponds to a FIELD parameter in a RECORD statement; the group of EDCB entries corresponding to an RDT entry specifies the editing to be performed on the records of the group.

The data conversion routine (IEBEDIT2) uses the information in the current EDCB entry and in its parameter list to compute the address of the input field or literal and the address at which the field or literal is to be stored.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

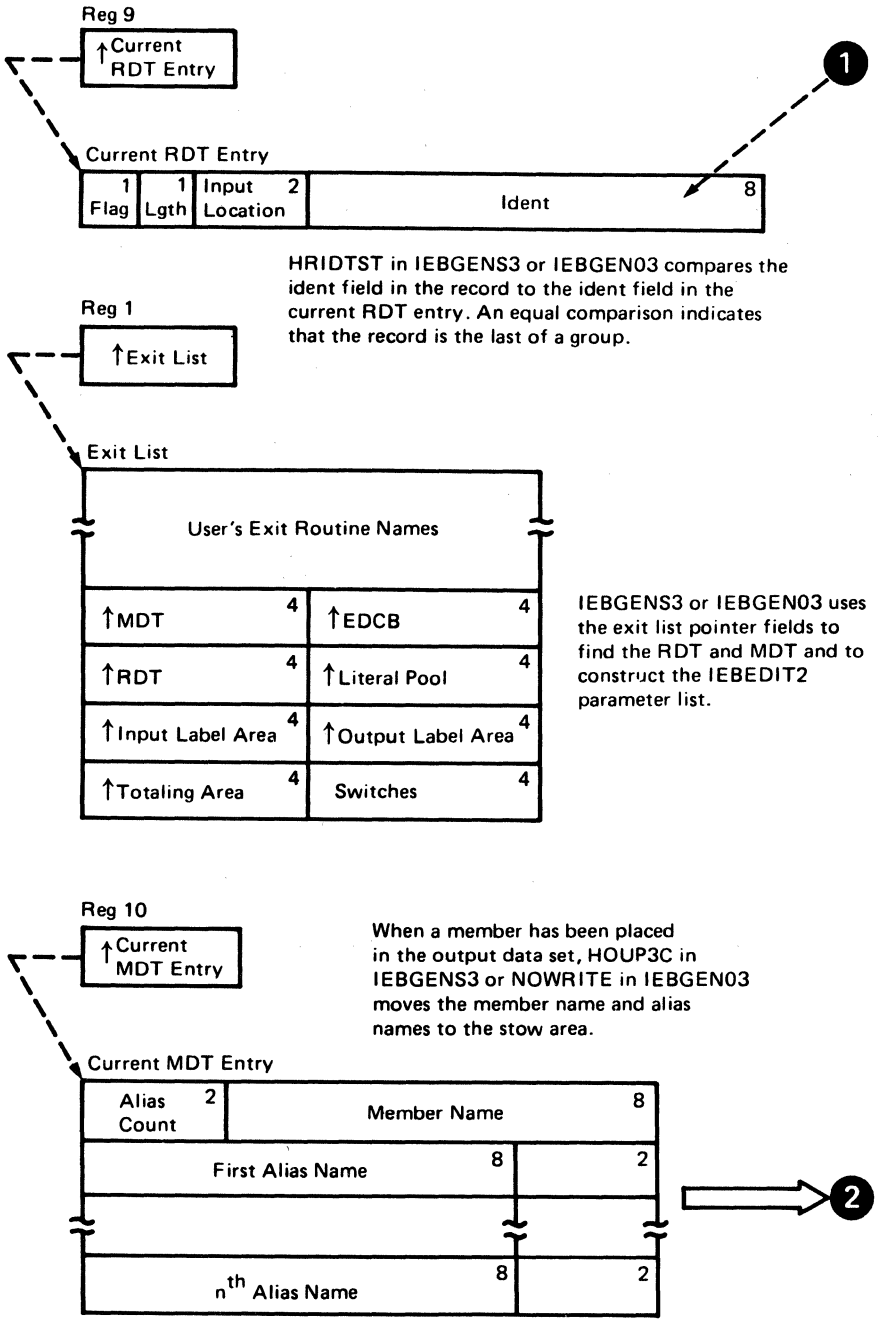


Figure 93 (Part 1 of 2). Editing the Input Data Set

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

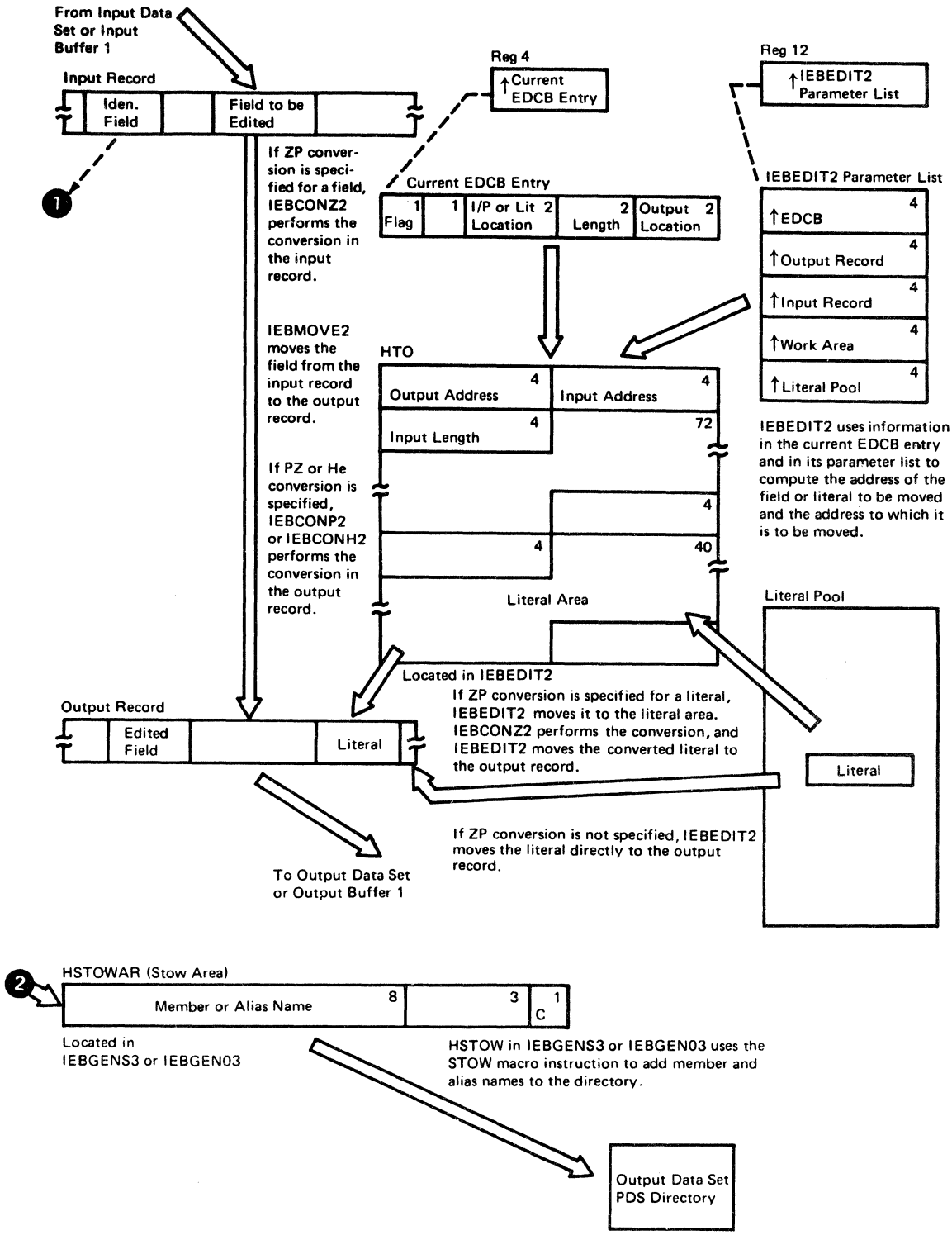


Figure 93 (Part 2 of 2). Editing the Input Data Set

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

If the EDCB entry specifies a literal that is to be converted from zoned to packed decimal (ZP), IEBEDIT2 moves it from the literal pool to a save area. IEBCONZ2 performs the conversion, and IEBEDIT2 moves it from the literal pool directly to the specified location in the output record.

If the EBDC entry specifies a literal that is not to be converted, IEBEDIT2 moves it from the literal pool directly to the specified location in the output record.

CONSTRUCTING THE MESSAGE DATA SET

The message data set (SYSPRINT) contains page headers with ascending sequence numbers, and error and warning messages from the IEBGENER program. All records in the message data set are written (via a PUT macro instruction) by the message writer subroutine. The message writer is named ZZPR; it is located in the root segment, and is used as a subroutine by all segments of the program. The message writing operation is shown in Figure 94 on page 223.

ZZPR is always entered via a BALR instruction specifying register 14. When it is entered, ZZPR stores the contents of registers 14 through 2 in an area called ZZSAVE. The area contains a 1-byte switch, called ZZSW, which is located at the high-order byte of register 0.

ZZPR uses the switch to determine whether it is to print a message or a page header:

- If it is to print a page header, the caller places a 512 in register 0, and the switch therefore contains X'02'.
- If ZZPR is to print a message, the caller places the message length in register 0. Because all messages are 62 bytes long, the switch contains zero.

  If ZZPR is to print a message, the message address is stored in the register 1 portion of ZZSAVE. ZZPR moves the message from the message module (CSECT IEBGMESG) to the print area (ZZPRINT), and then issues the PUT macro instruction.

  If ZZPR is to print a page header, ZZPR moves the header from HHDR to ZZHDSAVE, prints a blank line to cause page ejection, and then moves the header to ZZPRINT and issues the PUT macro instruction.

TABLES, SWITCHES, AND WORK AREAS

This section describes the formats and contents of the major tables and work areas, including switches, used by the IEBGENER utility program. Tables and work areas are drawn 8, 10, or 12 bytes wide; the length of each field is indicated in its upper right corner, and its label is shown in the upper left. The text in the field is descriptive of the contents, but is not necessarily a complete specification; the text that accompanies each figure describes the table or work area and its contents in more detail. For information on areas of main storage not described here or in the text, refer to the appropriate program listing.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

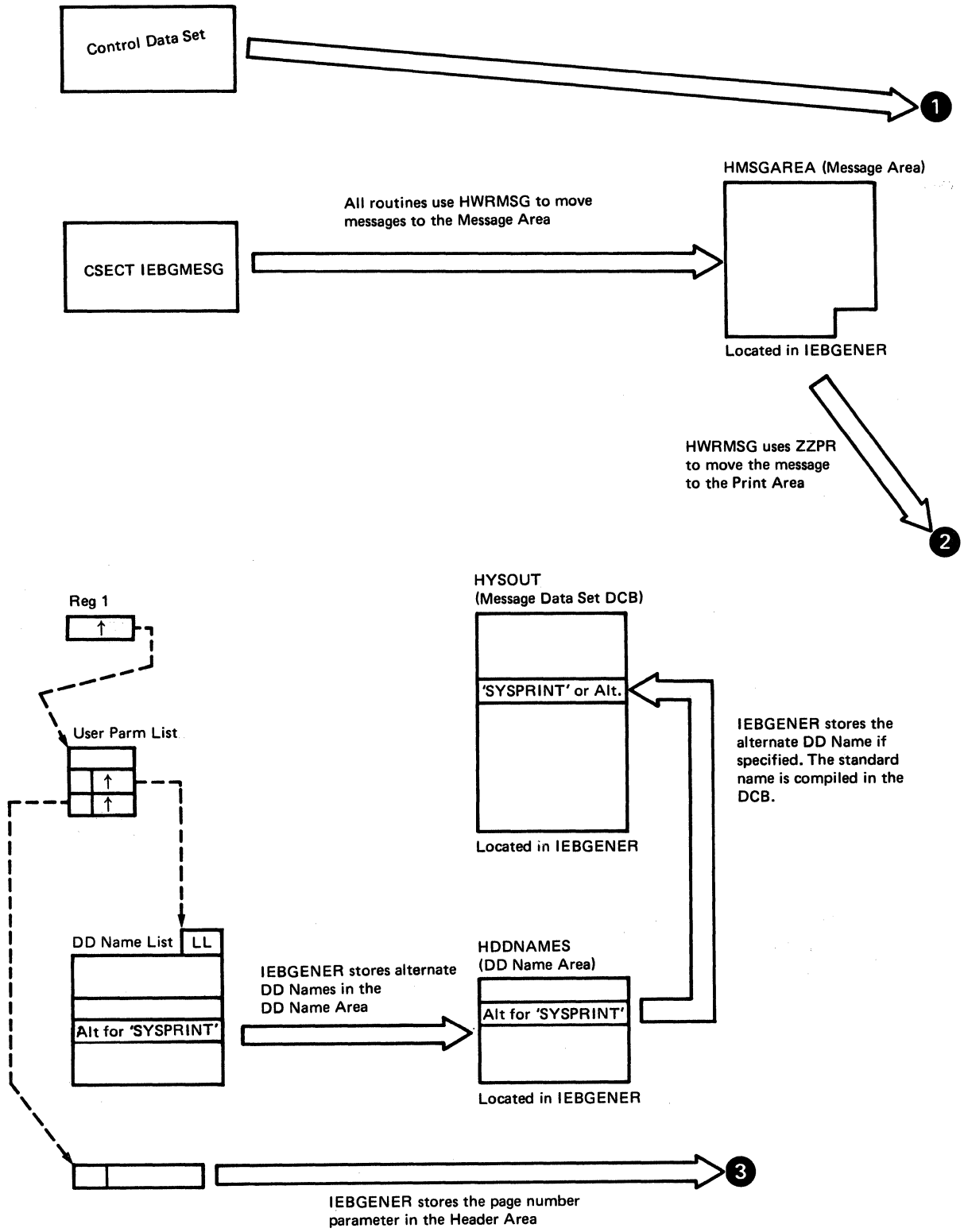


Figure 94 (Part 1 of 2). Constructing the Message Data Set

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

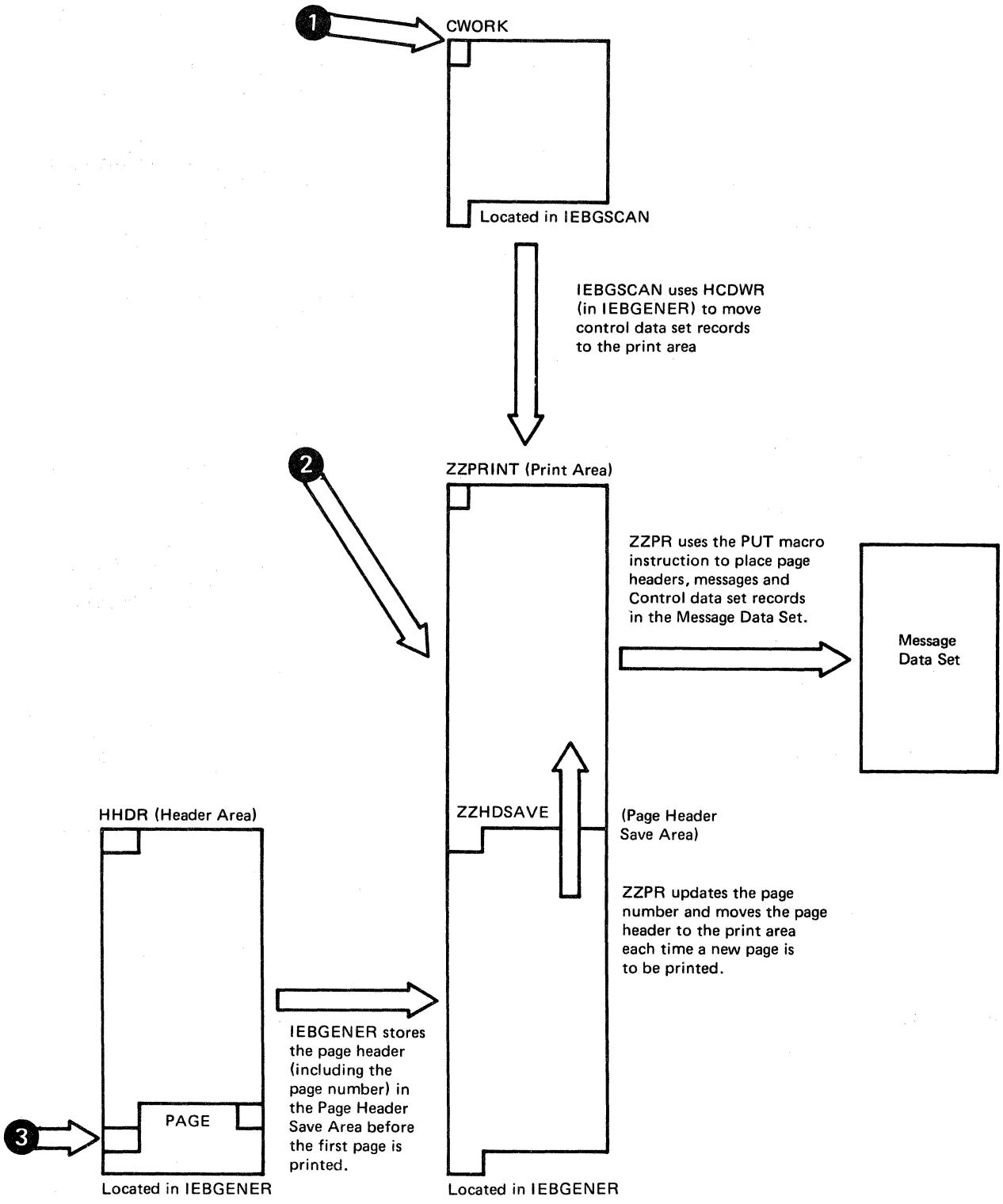


Figure 94 (Part 2 of 2). Constructing the Message Data Set

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

TABLES

The control statement parameter processor (CSECT IEBGSCAN) constructs tables from information contained in the control statement parameters:

Member Description Table (MDT) is created when IEBGSCAN encounters a MEMBER statement. It contains member names and aliases as shown in Figure 95.

Record Description Table (RDT) is created when IEBGSCAN encounters a RECORD statement. It contains record IDENTs, found on the RECORD statement, as shown in Figure 96 on page 226.

Edit Control Block (EDCB) is created when IEBGSCAN encounters a RECORD statement. It contains FIELD data, from the RECORD statement, as shown in Figure 97 on page 226.

Exit List is created when IEBGSCAN encounters an EXITS statement. It contains:

- Names of user routines
- Addresses of tables and blocks
- Switches

The format of the exit list is shown in Figure 98 on page 227.

WORK AREAS AND SWITCHES

In addition to tables, IEBGENER uses work areas, containing switches, to manipulate data. The shaded areas in the work areas are the switches and are described in greater detail following the work areas descriptions.

IEBGSCAN work areas include the CWORK area (Figure 99 on page 227) and a constants and switches area (Figure 100 on page 228).

The CWORK area is used by the scan routine (CSECT IEBCCS02) and the control statement parameter processor (CSECT IEBGSCAN) as a communication area. CWORK is a 146-byte area containing switches, a column pointer, a parenthesis counter, and buffers, in the format shown in Figure 99.

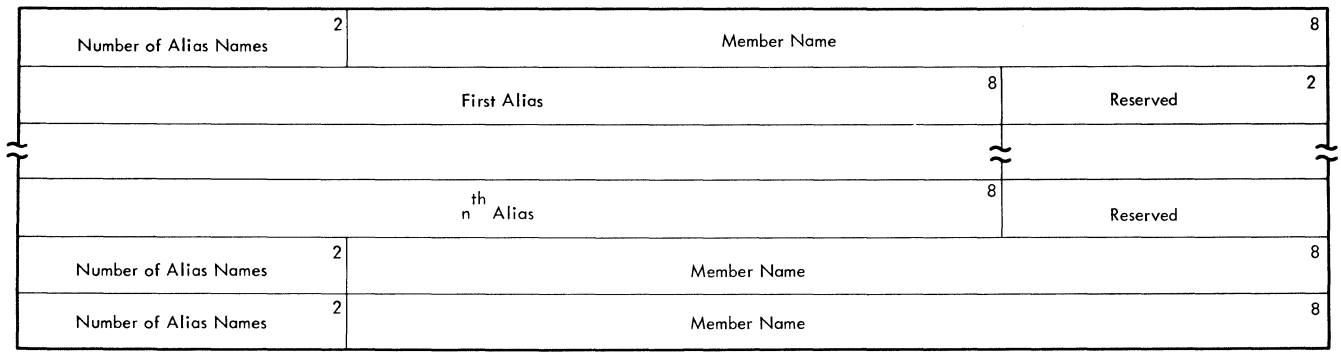
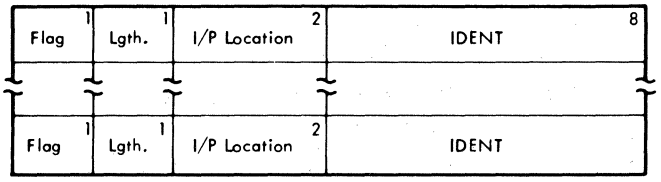


Figure 95. Member Description Table (MDT)

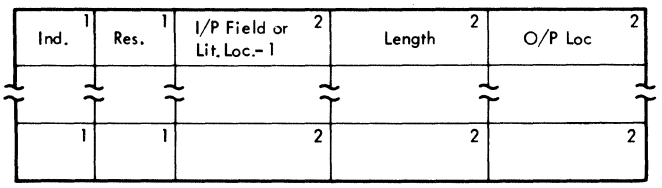
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1



The Flag byte contains the following information:

Hex	Meaning
20	FIELD parameters have been specified for this record group
4F	End of Table
5F	End of Table; process to EOD
8F	End of Table
9F	End of Table; process to EOD

Figure 96. Record Description Table (RDT)



The Indicator contains the following information:

Hex	Meaning
08	Conversion Code = ZP
10	Conversion Code = PZ
20	Conversion Code = HE
40	Specifies a literal
80	Specifies the last FIELD parameter in the RECORD statement
81	Last EDCB entry

Figure 97. The Edit Control Block (EDCB)

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

HEXITT	Name of User TOTAL Routine	8					
	Name of User INHDR Routine	8					
	Name of User OUTHDR Routine	8					
	Name of User INTRLR Routine	8					
	Name of User OUTTRLR Routine	8					
	Name of User DATA Routine	8					
	Name of User KEY Routine	8					
	Name of User IOERROR Routine	8					
HMDTAD	Address of MDT	4	HEDCBAD	Address of EDCB	4		
HRDTAD	Address of RDT	4	HLITTAD	Address of Literal Pool	4		
LABPOLAT	Address of Input Label Area	4		Address of Output Label Area	4		
TOTLAREA	Address of Totaling Area	4	LABPROSW 1 Label Sw.	HCOPYSW 1 Copy Sw.	HPOID 1 POS Sw	CIDSW 1 Member/Id Sw.	

Figure 98. Exit List

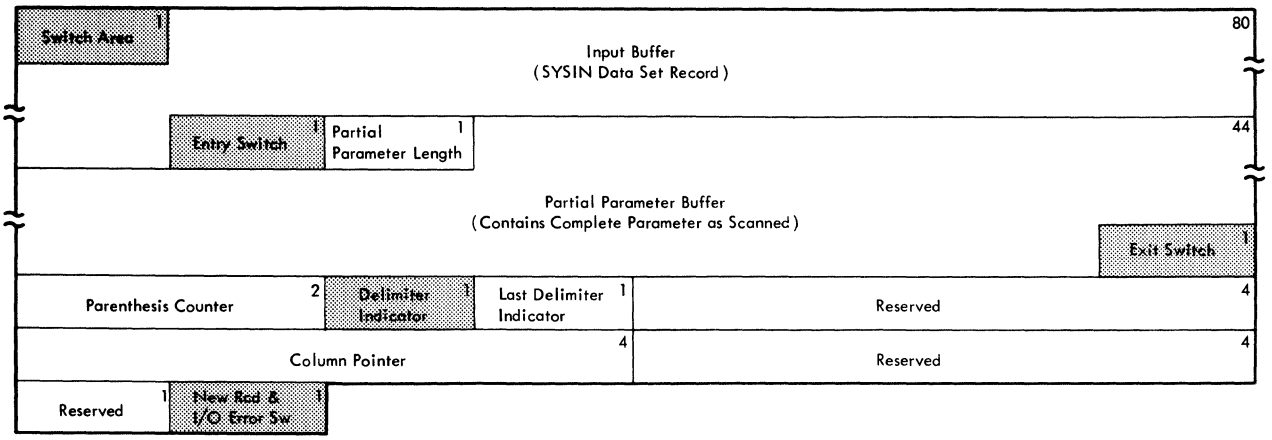


Figure 99. CWORK Communication Area

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

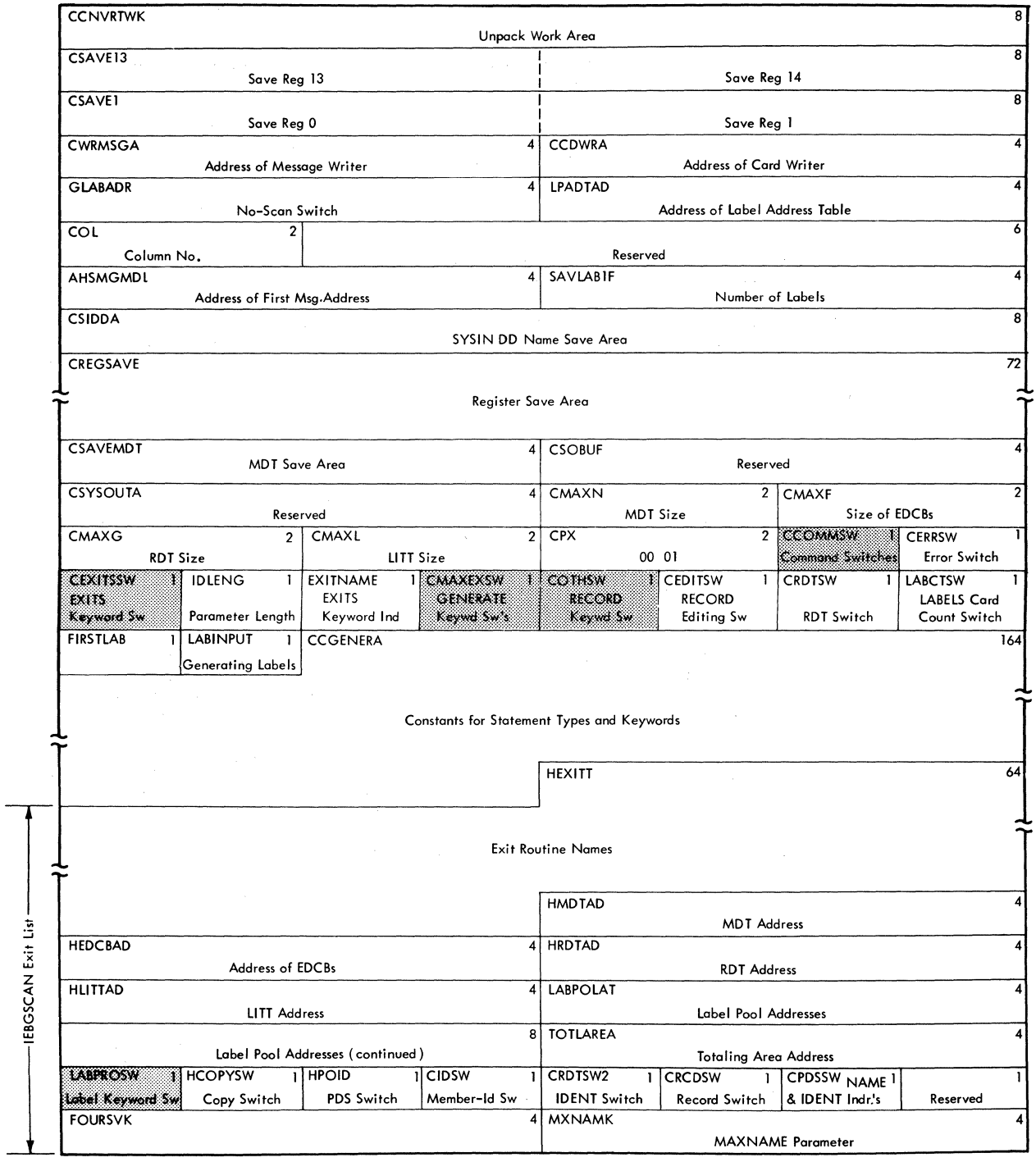


Figure 100. Constants and Switches Area for Module IEBGSCAN

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CWORK contains the following fields:

- Switch Area (1 byte): The bits in this field are set by the IEBCCS02 module and have the following meanings when set to one:

```
.... ...1  Last parameter in record, or subparameter in
           group.
.... ..1.  Scanned field is a parameter field.
.... .1..  Scanned field is a keyword field.
.... 1...  Scanned field is a command field.
...1 ....  Not first entry.
..1. ....  End-of-data.
.1.. ....  No more fields to process in this record.
1... ....  Syntax error.
```

- Input Buffer (80 bytes): This field is the input area for SYSIN data set records.

- Entry Switch (1 byte): Bits 0 to 5 in this field are set by the scan routine; bits 6 and 7 are set by the control statement parameter processor. When set to 1, the bits have the following meanings:

```
.... ...1  Scan a parameter.
.... ..1.  Scan for next keyword.
.... .1..  Scan for command field.
.... 1...  Read a record.
...1 ....  Parameter continued on next record.
..1. ....  Next field is a keyword or parameter field.
.1.. ....  Statement continued.
1... ....  Reserved.
```

  Note: Bits 6 and 7 are turned off (set to 0) by the scan routine.

- Partial Parameter Length (1 byte): If a parameter is continued on a second record, the scan routine counts the number of characters of the parameter that are in the current record, and stores the count in this field.

- Partial Parameter Buffer (44 bytes): If a parameter is continued on a second record, the scan routine stores the first part of the parameter in this field, then reads the next record, and stores the remainder of the parameter in this field. When the continuation statement has been read, this field contains the complete parameter.

- Exit Switch (1 byte): When set to 1, the bits in this field have the following meanings:

```
.... ...1  Length field is 0.
.... ..1.  Field length is greater than 0.
.... .1..  Position 72 is not blank.
.... 1...  Position 72 is blank.
...1 ....  Reserved.
..1. ....  Condition code 2: delimiter; end of record.
.1.. ....  Condition code 1: delimiter; not end of record.
1... ....  Reserved.
```

- Parenthesis Counter (2 bytes): A 1 is added to this field every time a left parenthesis is encountered; a 1 is subtracted every time a right parenthesis is encountered.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- Delimiter Indicator (1 byte): When set to 1, the bits in this field have the following meanings:

```
.... ...1  Reserved.
.... ..1.  Parenthesis not closed.
.... .1..  Delimiter is an apostrophe.
.... 1...  Delimiter is a blank.
...1 ....  Delimiter is a right parenthesis.
..1. ....  Delimiter is a left parenthesis.
.1.. ....  Delimiter is a comma.
1... ....  Delimiter is an equal sign.
```

- Last Delimiter Indicator (1 byte): The bits in this field mean the same as the corresponding bits in the delimiter indicator field. This field, however, refers to the previous delimiter instead of the current delimiter.

- Column Pointer (4 bytes): This field contains a pointer to the beginning of the field to be scanned.

- New Record and I/O Error Switch (1 byte): This field is set to X'01' when a new record has been read, and to X'02' when an I/O error occurs.

The IEBGSCAN module also contains a 442-byte constant and switches area (illustrated in Figure 100). The major switches (shaded areas) are described in detail:

- CCOMMSW (Command Switch): The bits in this 1-byte switch have the following meanings when set to 1:

```
.... ...1  Reserved.
.... ..1.  Reserved.
.... .1..  Reserved.
.... 1...  LABELS statement found.
...1 ....  RECORD statement found.
..1. ....  MEMBER statement found.
.1.. ....  EXITS statement found.
1... ....  GENERATE statement found.
```

- CEXITSSW (EXITS Keyword Switch): The bits in this 1-byte switch have the following meanings when set to 1:

```
.... ...1  KEY keyword specified.
.... ..1.  DATA keyword specified.
.... .1..  OUTTLR keyword specified.
.... 1...  INTLR keyword specified.
...1 ....  OUTHDR keyword specified.
..1. ....  INHDR keyword specified.
.1.. ....  TOTAL keyword specified.
1... ....  IOERROR keyword specified.
```

- CMAXEXSW (GENERATE Keyword Switch): The bits in this 1-byte switch have the following meanings when set to 1:

```
.... ...1  Reserved.
.... ..1.  Reserved.
.... .1..  Reserved.
.... 1...  Reserved.
...1 ....  Reserved.
..1. ....  MCAGPS keyword specified.
.1.. ....  KAXFLDS keyword specified.
1... ....  MAXNAME keyword specified.
```

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- COTHSW (RECORD Keyword Switch): The bits in this 1-byte switch have the following meanings when set to 1:

```
.... ...1  Reserved.
.... ..1.  Reserved.
.... .1..  Reserved.
.... 1...  Reserved.
...1 ....  Reserved.
..1. ....  FIELD keyword specified.
.1.. ....  IDENT keyword specified.
1... ....  NAME keyword specified.
```

- LABPROSW (LABEL Keyword Switch): The bits in this 1-byte switch have the following meanings when set to 1:

```
.... ...1  Reserved.
.... ..1.  Reserved.
.... .1..  Reserved.
.... 1...  DATA=INPUT keyword specified.
...1 ....  DATA=ALL keyword specified.
..1. ....  DATA=ONLY keyword specified.
.1.. ....  DATA=NO keyword specified.
1... ...   DATA=YES keyword specified.
```

IEBGENR3 constants and switches area is a 332-byte area containing save areas, address pointers, and switches (see Figure 101 on page 232). The major switch (shaded area) is described in detail:

- LABPROSW (LABEL Process Switch): The bits in this 1-byte switch have the following meanings when set to 1:

```
.... ...1  Reserved.
.... ..1.  Reserved.
.... .1..  Reserved.
.... 1...  DATA=INPUT keyword specified.
...1 ....  DATA=ALL keyword specified.
..1. ....  DATA=ONLY keyword specified.
.1.. ....  DATA=NO keyword specified.
1... ....  DATA=YES keyword specified.
```

Note: If no other bits are set, DATA=YES is set to 1.

The IEBGENS3 constants and switches area is a 332-byte area containing save areas, work areas, address pointers, and switches (see Figure 102 on page 233). The major switches (shaded areas) are described in detail:

- KEYOUTSW (User Key Exit Switch): The bits in this 1-byte switch have the following meanings when set to 1:

```
.... ...1  Exit key specified.
.... ..1.  SYSUT2 is a direct access device.
.... .1..  SYSUT1 is a direct access device.
.... 1...  SYSUT1 RECFM same as SYSUT2 RECFM.
...1 ....  SYSUT1 BLKSIZE same as SYSUT2 BLKSIZE.
..1. ....  SYSUT1 KEYLEN same as SYSUT2 KEYLEN.
.1.. ....  SYSUT1 key length not 0.
1... ....  SYSUT2 key length is 0.
```

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

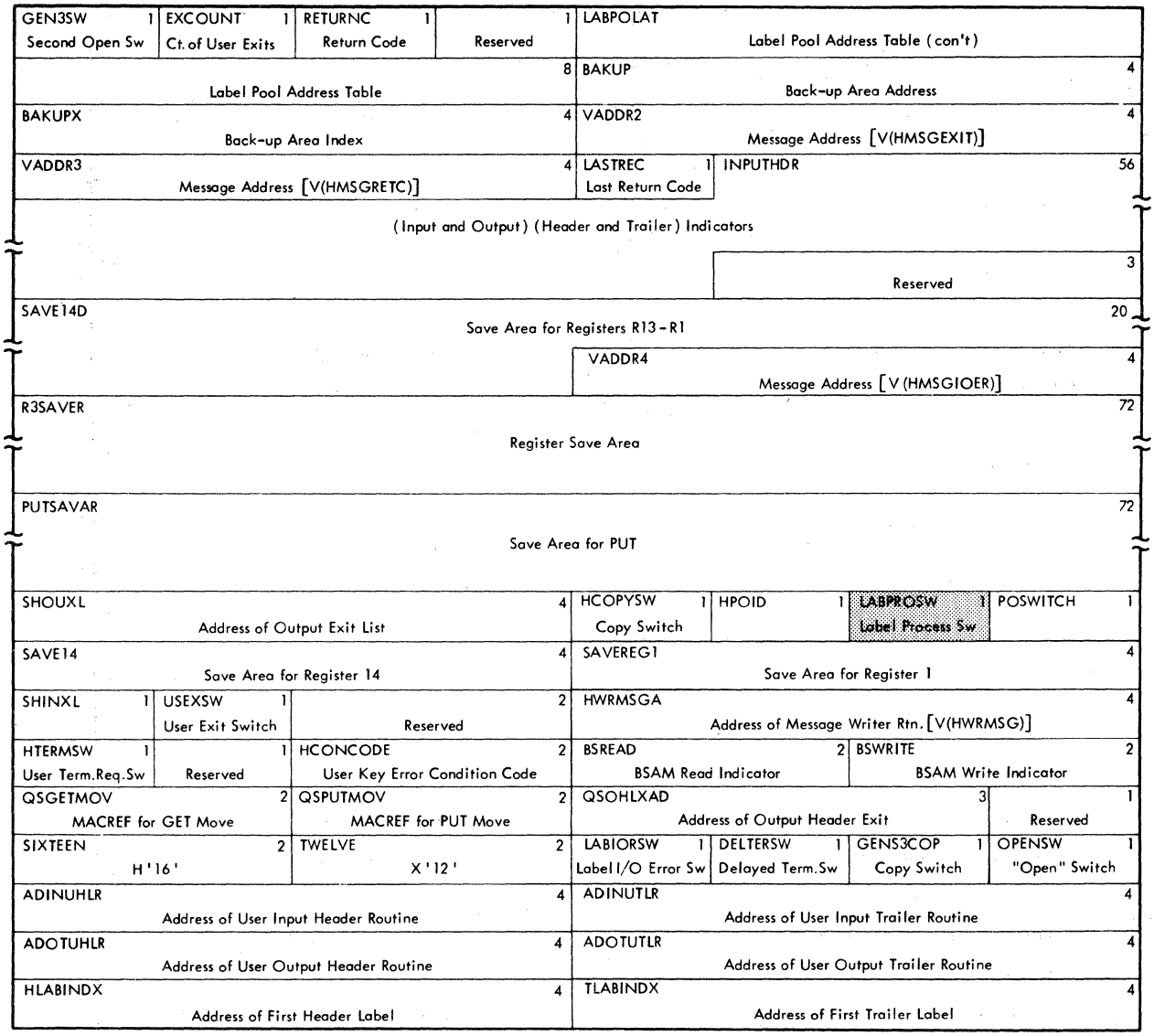


Figure 101. Constants and Switches Area for Module IEBGENR3

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

HINTB, HKEYL Input Key Length	4			HBLIN	4		
HRLIN Record Length	2	HINORG Input Rec Format	2	HOUTB, HKEYLOU Output Key Length	4		
HBLOU Output Buffer Address	4			HRLOU Output Logical Record Length	2	HOUORG Output Rec Format	1	HPOID PDS Indicator	1
HGENPRS Work Area	72						
HSAVE13 Register 13, 14 Save Area	8						
HSAVE14 Register 14 Save Area	4			HSTOWAR	12		
HIN1AD SYSUT1 (Input) Buffer 1 Address Save Area	4			HIN2AD SYSUT1 (Input) Buffer 2 Address Save Area	4		
HOU1AD SYSUT2 (Output) Buffer 1 Address Save Area	4			HOU2AD SYSUT2 (Output) Buffer 2 Address Save Area	4		
HEDSV EDT Save Area	4			HIXIN SYSUT1 Current Record Address	4		
HIXOUT SYSUT2 Current Record Address	4			HBLOUAC Actual Block Length	4		
HWK4	4			HRDTSAV RDT Save Area	4		
HRLPRE Output Record Count	2	HWK2 ARS	2	HKEYA Address of Key Area	4		
LABPOLAT Input Label Area Address	4			Output Label Area Address	4		
TOTLAREA Totaling Parameter List	16						
SAVER01 Save Area for Registers 0 and 1	8						
SYNADSAV Save Area for Registers 10, 3, 4, and 12	16						
HEXITA Address of IEBGSCAN Exit List	4			HMDTCUR Address of Current MDT Entry	4		
HEDCUR Address of Current ECDB Entry	4			HRDTCUR Address of Current RDT Entry	4		
HLITTA Address of Literal Pool	3			HCOPYSW Copy,EOD Switch	1	Reserved	3
HDCBOUTA Address of Output DCB [V(HDCBOUT)]	4			VADDR3 Message Address [V(HMSGRETC)]	4		
HDCBINA Address of Input DCB [V(HDCBIN)]	4			STHEODAD EOD Address	4		
HEODAD Address of EOD Routine	4			HINERAD Address of Input Error Routine	4		
HOUERAD Address of Output Error Routine	4			ADTOTEXR Address of User Totaling Routine	4		
SIXTEEN H'16'	2	FOUR H'4'	2	EIGHT H'8'	2	Reserved	2
R14SAVE Register 14, 15 Save Area	8						
SAVER12 Register 12 Save Area	4			TOTLEXSW Totaling Exit Sw	1	LABPROSW Label Processing Sw	1	KEYOUTSW User Key Exit Sw	1	Reserved	1
THREE H'3'	2	RECFMSW	1	HSTOWSW Stow Sw	1	HRIDSW	1	HTERMSW Termination Sw	1	Reserved	2
HWRMSGA Address of Message Writer	4			HCONCODE Condition Code	2	HH0 H'0'	2				
HH2 H'2'	2	HH4 H'4'	2	TOTEDIT	2	HALFWORK Blocksize	2				
SIXSV Register 6 Save Area	4			REGSAV Length of Buffer to be Freed	4		
AST C'****'	4			BLANKS X'4040'	2	CPYSWSAV Copy Sw	1	TYPMSG Message Indicator	1		

Figure 102. Constants and Switches Area for Module IEBGENS3

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- RECFMSW (Record Format Switch): The bits in this 1-byte switch have the following meanings when set to 1:

```
.... ...1  SYSUT2 has undefined records.
.... ..1.  SYSUT1 has variable records.
.... .1..  Reserved.
.... 1...  Reserved.
...1 ....  Undefined type records.
..1. ....  Variable type records.
.1.. ....  Reserved.
.1.. ....  Reserved.
```

  Note: A combination of bits 2 and 3 indicates variable blocked records.

The IEBGEN03 constants and switches area is a 321-byte area containing save areas, address pointers, and switches (see Figure 103 on page 235). The major switch (shaded area) is described in detail:

- KEYOUTSW (User Key Exit Switch): The bits in this 1-byte switch have the following meanings when set to 1:

```
.... ...1  Reserved.
.... ..1.  SYSUT2 is a direct access device.
.... .1..  SYSUT1 is a direct access device.
.... 1...  SYSUT1 RECFM same as SYSUT2 RECFM.
...1 ....  SYSUT1 BLKSIZE same as SYSUT2 BLKSIZE.
..1. ....  Reserved.
.1.. ....  Reserved.
1... ....  Reserved.
```

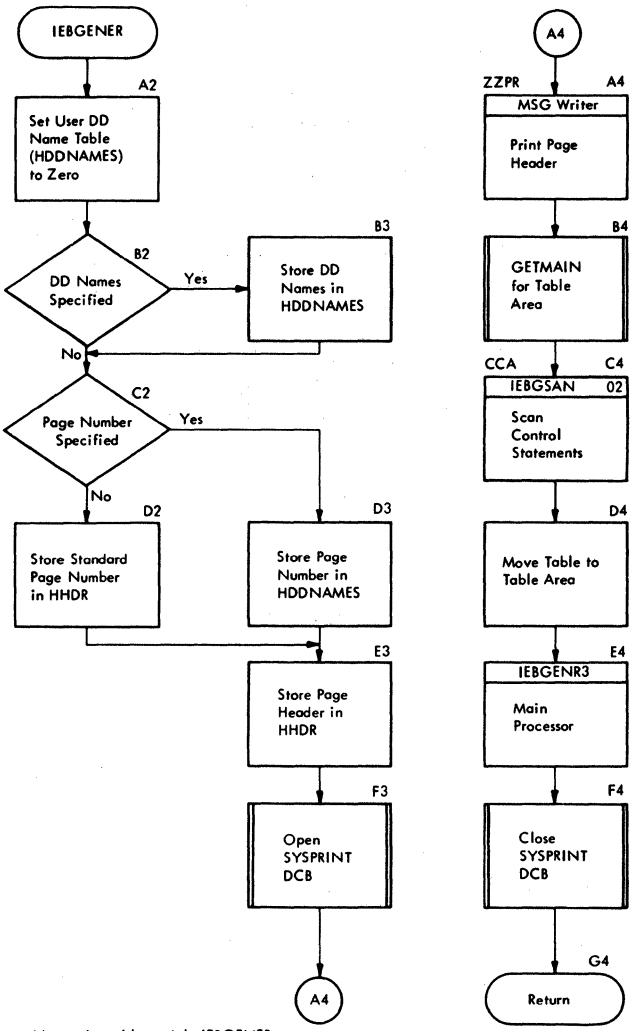
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

HINTB, HKEYL Input Key Length	4			HBLIN	4		
HRLIN Record Length	2	HINORG Input Rec Format	2	HOUTB, HKEYLOU Output Key Length	4		
HBLOU Output Buffer Address	4			HRLOU Output Logical Record Length	2	HOUORG Output Rec.Format	1	HPOID PDS Indicator	1
HGENPRS	0						72
Work Area							
HSAVE13 Register 13, 14 Save Area	8						
HSAVE14 Register 14 Save Area	4			HSTOWAR	12		
HIN1AD SYSUT1 (Input) Buffer 1 Address Save Area	4			HIN2AD SYSUT1 (Input) Buffer 2 Address Save Area	4		
HOU1AD SYSUT2 (Output) Buffer 1 Address Save Area	4			HOU2AD SYSUT2 (Output) Buffer 2 Address Save Area	4		
HEDSV EDT Save Area	4			HIXIN SYSUT1 Current Record Address	4		
HIXOUT SYSUT2 Current Record Address	4			HBLOUAC Actual Block Length	4		
HWK4	4			HRDTSAV RDT Save Area	4		
HRLPRE Output Record Count	2	HWK2 ARS	2	HKEYA Address of Key Area	4		
LABPOLAT Input Label Area Address	4			Output Label Area Address	4		
TOTLAREA Totaling Parameter List	16						
SAVER01 Save Area for Registers 0 and 1	8						
SYNADSAV Save Area for Registers 10, 3, 4, and 12	16						
HEXITTA Address of IEBGSCAN Exit List	4			HMDTCUR Address of Current MDT Entry	4		
HEDCUR Address of Current ECDB Entry	4			HRDTCUR Address of Current RDT Entry	4		
HLITTA Address of Literal Pool	4			HCOPYSW Copy,EOD Switch	1	Reserved	3
HDCBOUTA Address of Output DCB [V(HDCBOUT)]	4			VADDR3 Message Address [V(HMSGRETC)]	4		
HDCBINA Address of Input DCB [V(HDCBIN)]	4			ADTOTEXR Address of User Totaling Routine	4		
SIXTEEN H'16'	2	FOUR H'4'	2	EIGHT H'8'	2	Reserved	2
TOTLEXSW Totaling Exit Sw	1	LABPROSW Label Proc.Sw	1	KEYOUTSW User Key Exit Sw	1	Reserved	1	THREE H'3'	2	RECFMSW Record Format Sw	1	HSTOWSW Stow Sw	1
HRIOSW Last Record Sw	1	HTERMSW Termination Sw	1	Reserved	2			HEODAD Address of End-of-Data Routine	4				
HINERAD Address of Input Error (SYNAD) Routine	8							HOUERAD Address of Output Error (SYNAD) Routine	4				
HWRMSGA Address of Message Writer Routine	4							HCDWRA Alternate Entry Point to Message Writer	4				
HCONCODE Condition Code Save Area	2	HH0 H'0'	2	HH2 H'2'	2	HH4 H'4'	2						
TOTEDIT	2	HALFWORK Blocksize	2	SIXSV Register 6 Save Area	4								
REGSAV Length of Buffer to be Freed	4			AST C'****'	4								
BLANKS X'4040'	2	CPYSWSAV Copy Switch	1	TYPMSG Message Indicator	1	REDANY SYSUT1 Rec.Indr.	1						

Figure 103. Constants and Switches Area for Module IEBGEN03

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

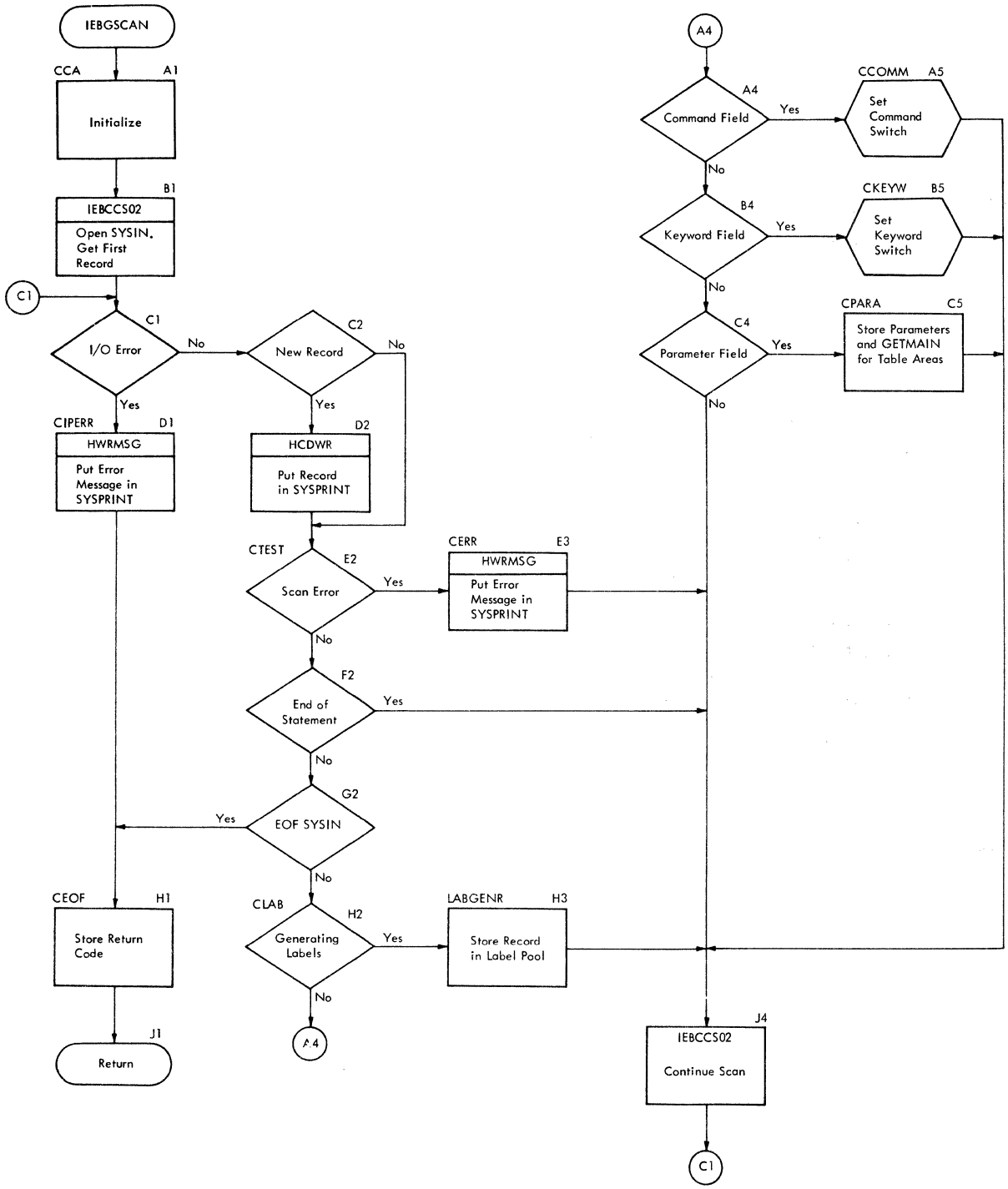
CHART 01. ROOT SEGMENT



Note: Assembly module IEBGENER is referred to as IEBGENRT in microfiche.

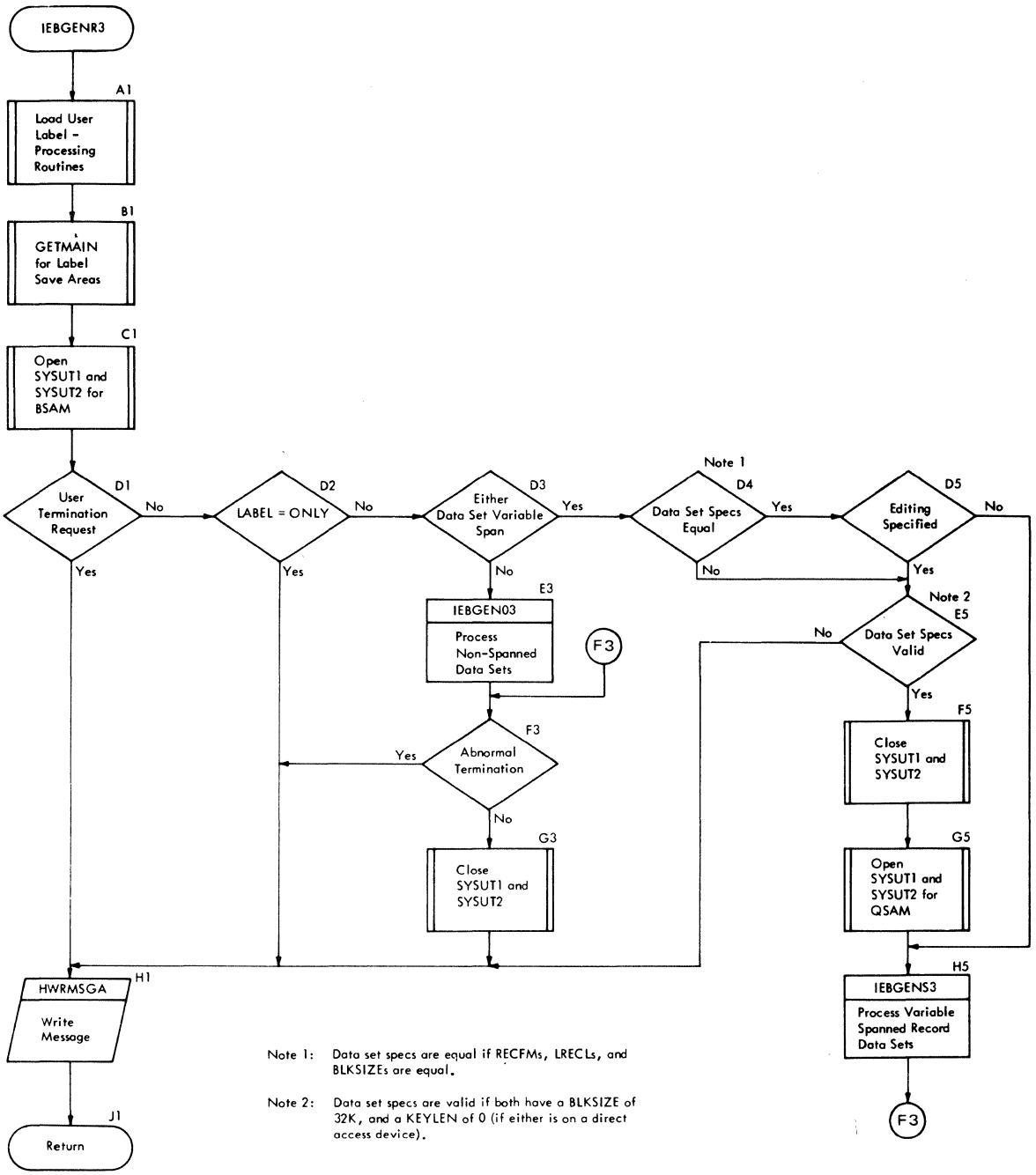
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 02. CONTROL STATEMENT ANALYSIS ROUTINE



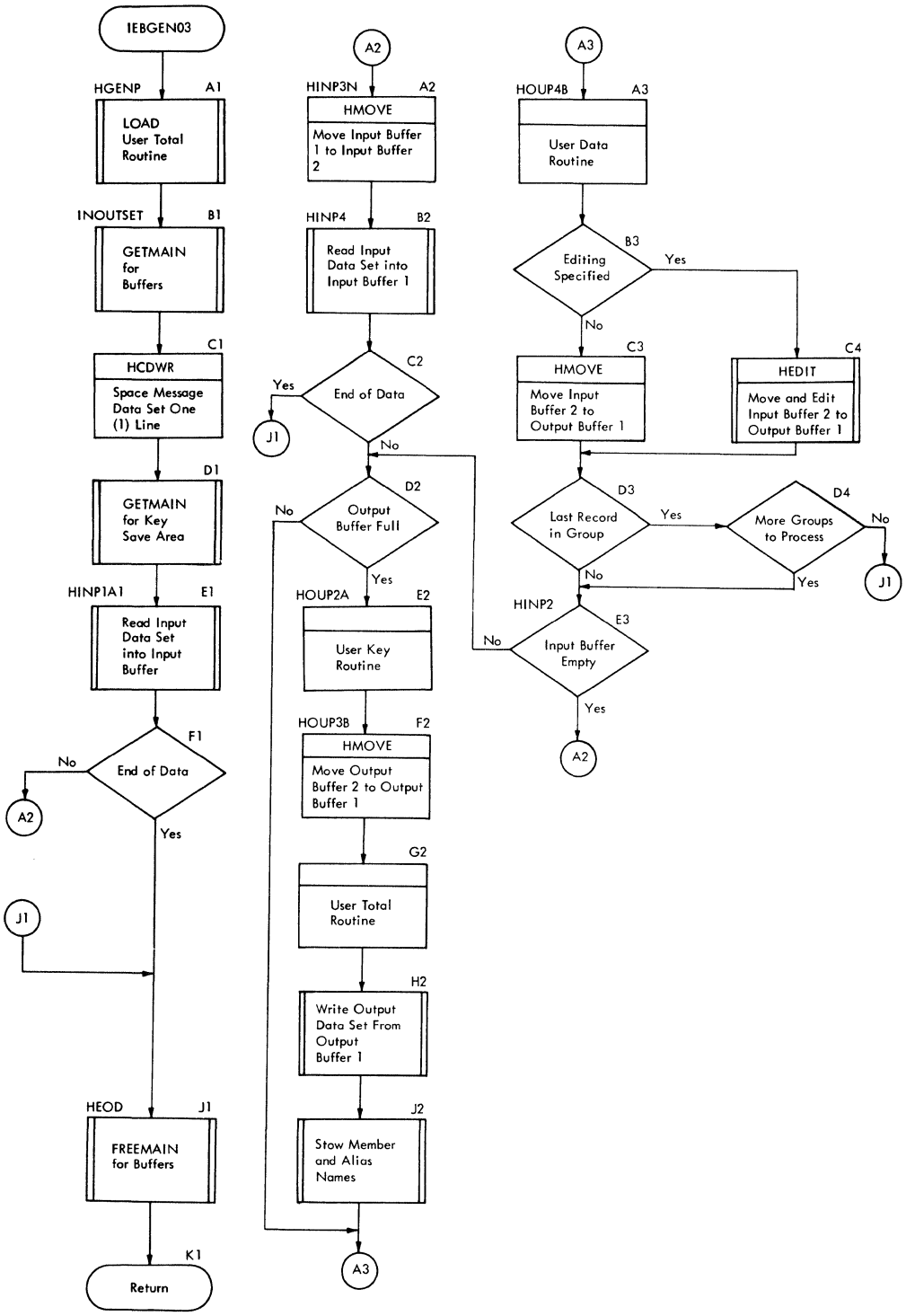
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 03. MAIN PROCESSING SEGMENT



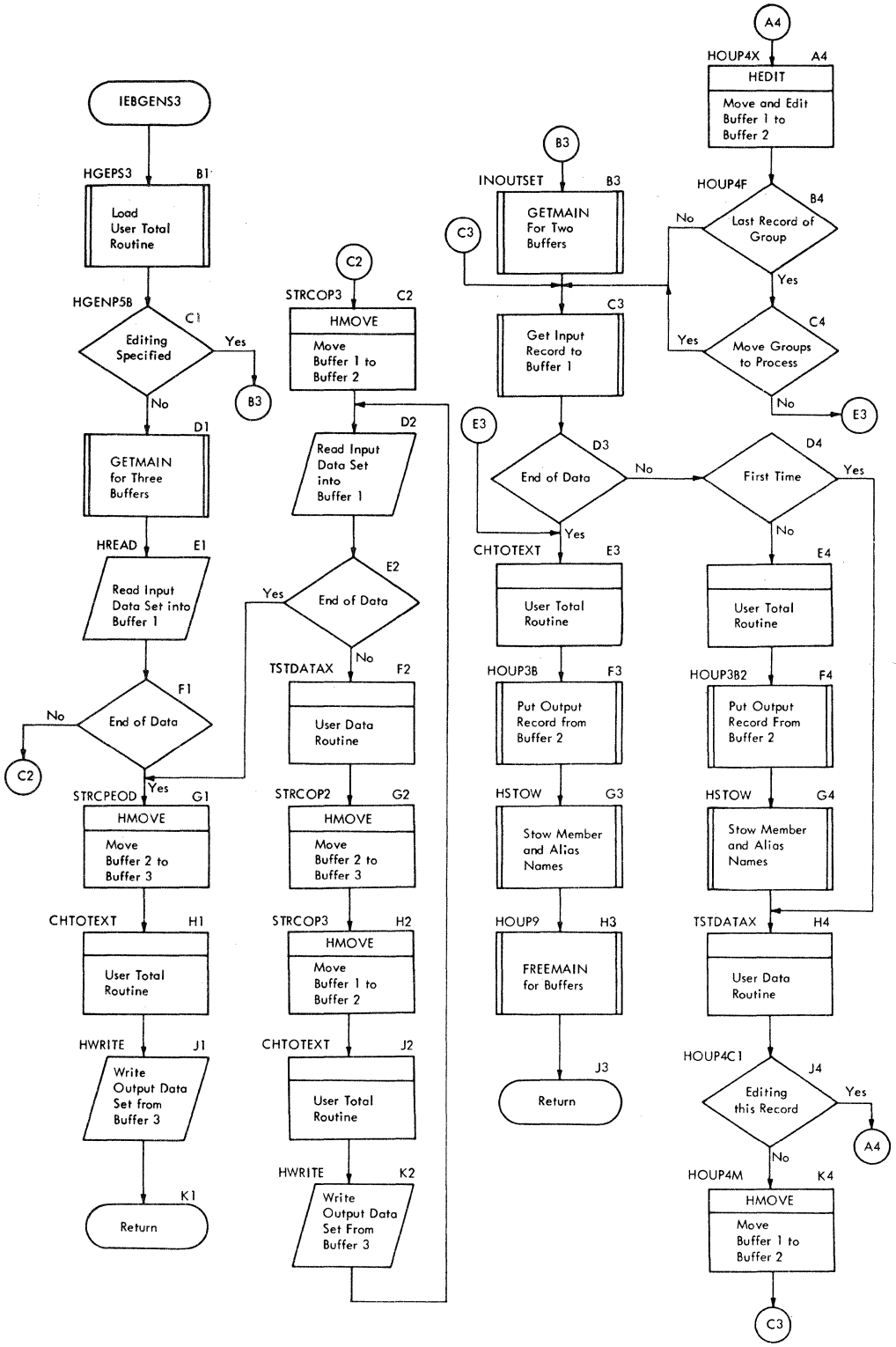
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 04. PROCESSING NONSPANNED RECORD DATA SETS



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 05. PROCESSING VARIABLE SPANNED RECORD DATA SETS



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEBIMAGE PROGRAM

INTRODUCTION

This utility provides the means by which the printer image library (SYS1.IMAGELIB or a user library) is maintained for the IBM 3800 Printing Subsystem. The utility consists of one mapping macro, IEBIBLKS, and nine modules:

- IEBIMAGE
- IEBIMCHA
- IEBIMCOP
- IEBIMFCB
- IEBIMGRA
- IEBIMTAB
- IEBIMWRT
- IEBIMSGS
- IEBISCAN

Models 1 and 3 of the 3800 printer are supported.

IEBIMAGE MODULE

This module is the main procedure, which contains the utility initialization and common, local procedures to process the supported operations. The supported operations are FCB, COPYMOD, GRAPHIC, CHARSET, and TABLE. The local procedures used by other modules are as follows:

Procedures	Meaning
ABEND	traps 213-04, D37-04, and 713-08 abends
BKSZCK	ensures correct block size on SYSPRINT and SYSIN
BKSZFIX	determines block size for library
BLDL	locates members on the library
CONVERT	binary-to-decimal conversion
DELCK	checks success of DELETE requests
EODAD	SYSIN end-of-file routine
LOADLIB	loads text records into storage to the library
PRINTQ	prints the print queue to SYSOUT
PRTCARD	prints control statements
READLIB	reads text records into buffer
SYNAD	obtains information about library errors
WRBLOCK	performs blocking of module

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Procedures	Meaning
WRITE	writes records to the library
WRTEXT	prepares one text block for writing to library

The utility obtains storage for the work area and buffer, opens the system data sets, and calls IEBISCAN to read control statements and to determine the requested operation. IEBIMAGE control statements are:

- FCB—causes a forms control buffer image to be created, altered, and/or printed.
- TABLE—causes a character arrangement table to be created or altered and/or printed.
- COPYMOD—causes a copy modification module to be created or altered and/or printed.
- GRAPHIC—causes a graphic character modification module to be created or altered and/or printed.
- CHARSET—causes a library character set module to be created, altered, and/or printed.
- INCLUDE—is a modifying control statement that causes the subsequent control statement to bring into storage a module to be altered.
- NAME—is a delimiting control statement that must end each unique operation (COPYMOD, CHARSET, and GRAPHIC may have multiple control statements for one operation).
- OPTION—is a modifying control statement that specifies environmental options for a module. The options that may be specified are line-mode compatibility for the 3800 Model 3 printer, or line overrun value for a COPYMOD module.

When control is returned from IEBISCAN, the appropriate procedure is called to build the module requested or, if "print only" has been requested, to prepare the utility for subsequent printing of the module.

A NAME control statement must delimit each operation. When the NAME statement is encountered, control is passed to the NAME procedure. If print only has been previously requested, the appropriate module will be brought into storage, validity checked, and printed. If a create or alter request was specified, the module in the buffer is checked for validity, written to the library, and printed on the SYSOUT data set.

IEBIMCHA MODULE

This module is invoked when a CHARSET operation has been requested. It contains all the logic necessary to create, check, and/or print library character sets for either the Model 1 or Model 3 3800 printer.

IEBIMCOP MODULE

This module is invoked when a COPYMOD operation has been requested. A table-driven algorithm contained in this module assesses the possibility of the COPYMOD causing a line overrun condition. Unique timings for Models 1 and 3 of the 3800 printer are calculated as the COPYMOD is being built.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEBIMFCB MODULE

This module is invoked when an FCB operation has been requested. FCBs of 6, 8, 10, and 12 lines per inch are built, checked, and/or printed by this module.

IEBIMGRA MODULE

This module is invoked when a GRAPHIC operation has been requested. It contains all the logic necessary to create, check, and/or print graphic character modification modules for either the Model 1 or Model 3 3800 printer.

IEBIMTAB MODULE

This module is invoked when a TABLE operation has been requested. Character arrangement table modules are created, modified, and/or printed by this module.

IEBIMWRT MODULE

This module, when invoked, will write a newly created or modified object module to the library.

IEBISCAN MODULE

This procedure handles reading and scanning of all control statements. It determines control statement types and keyword types by referencing the operation table (OPTAB) and keyword table (KEYTAB) respectively.

Reading of control statements is done by the internal procedure GETCARDN.

IEBIMSGS MODULE

This procedure contains all the messages printed by the utility and the code by which the messages are selected. A message number is accepted in the MSGN field and possible data to be edited is accepted through the MSGEDIT list in the work area. The list contains four possible entries, each with the following format:

Byte 0 length of data
Bytes 1-3 address of data

This module selects the message and inserts variable data if needed. The message is written to the SYSPRINT data set.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

METHOD OF OPERATION

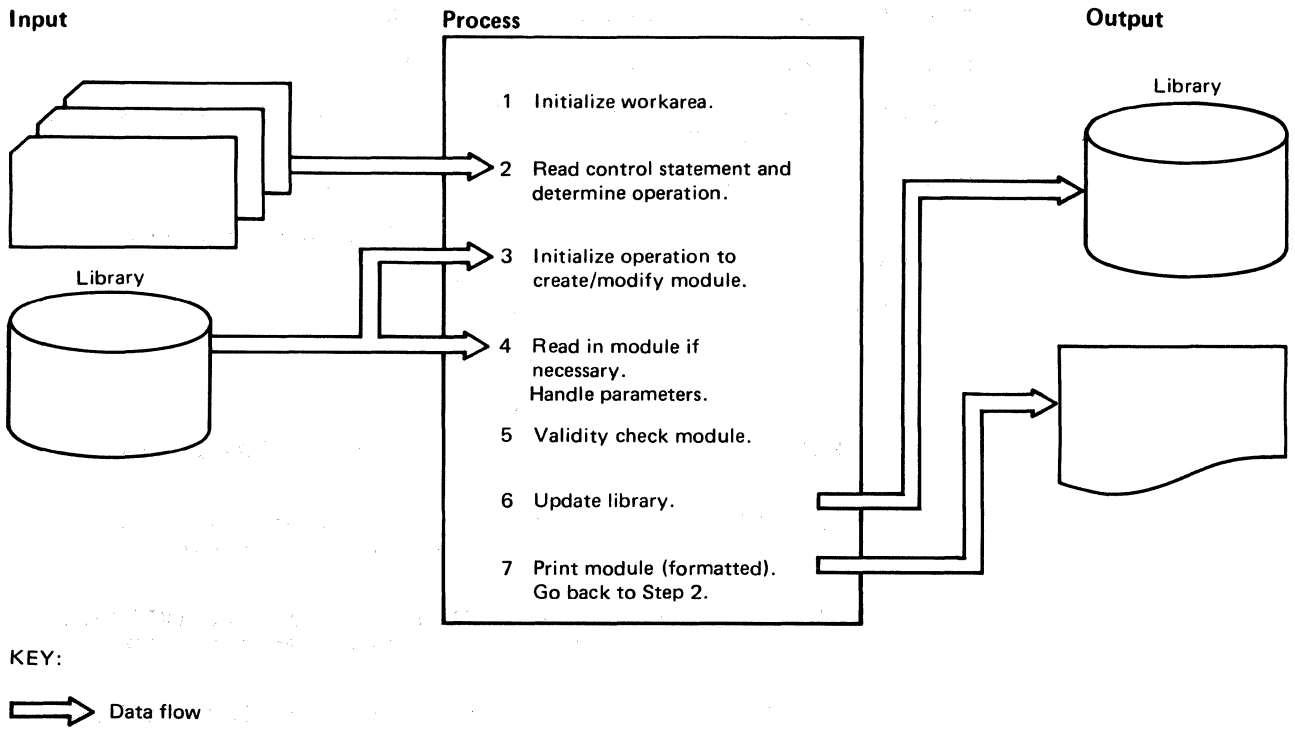


Figure 104. Overview—IEBIMAGE Operation

Notes for Figure 104:

IEBIMAGE

Procedure: IEBIMAGE

1 Clear work area and buffer to zero. Process parameters if attached or linked to. Initialize work area with DCB addresses, ddnames, and operation table characteristics. Open input data sets and system output data set.

IEBISCAN

Procedure: IEBISCAN

2 Call IEBISCAN to read a control statement and to determine the operation requested by table lookup (operation table).

IEBIMAGE, IEBISCAN, IEBIMCHA, IEBIMCOP, IEBIMFCB, IEBIMGRA, IEBIMTAB

Procedures: IEBIMAGE, FCB, TABLE, COPYMOD, GRAPHIC, CHARSET, INCLUDE, NAME, IEBISCAN (keyword).

3 Start requested operation (routine address in operation table). The following unique initialization is performed for each operation:

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

FCB Set module prefix FCB3

TABLE Set module prefix XTB1

COPYMOD Set module prefix MOD1

GRAPHIC Set module prefix GRAF or GRF2

CHARSET Set module prefix LCS1 or LCS2

INCLUDE Save member name suffix and DELSEG information; go to step 2.

NAME Go to step 4.

- If no keywords are specified on the operation (read only), go to step 2.
- If keywords are specified on the operation:
  - Unique initialization continues.

    FCB Set defaults.

    TABLE Set defaults or read in INCLUDED member. [TABLINIT]

    COPYMOD Read in INCLUDED member, or LOAD INCLUDED member and delete specified statements. [COPYINCL]

    GRAPHIC Read in INCLUDED member or LOAD INCLUDED member and delete specified segments. [GRAFINCL]

    CHARSET Read in INCLUDED member or LOAD INCLUDED member and delete specified segments. [CHARINCL]

  - Call IEBISCAN to identify keywords and their parameters.
  - Build or modify module, using specified keyword information.

```
IEBIMFCB:FCB [FCBBLD]
IEBIMTAB:TABLE [SETLOC]
IEBIMCOP:COPYMOD [TEXTCOMP,SETTEXT]
IEBIMGRA:GRAPHIC REF=[SETREF]
IEBIMGRA:GRAPHIC ASSIGN=[GRAFBLD,GRAFBLD3]
IEBIMCHA:CHARSET REF=[SETCREF]
IEBIMCHA:CHARSET ASSIGN=[CHARBLD,CHARBLD3]
```

- Go to step 2.

IEBISCAN, IEBIMAGE

Procedures: IEBISCAN, NAME

4 Call IEBISCAN to identify name of member to be added or read in. If READONLY, read in module specified. [READONLY]

IEBIMAGE, IEBIMTAB, IEBIMCOP, IEBIMGRA, IEBIMCHA, IEBIMFCB

Procedures: TABCHK, COPYCHK, GRFCHK, GRFCHK3, CHARCHK, CHARCHK3, FCBCHECK

5 Check module for validity (routine address in operation table).

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEBIMAGE

Procedure: NAME

6 Write module to library as partitioned data set member. [WRITELIB]

IEBIMAGE, IEBIMFCB, IEBIMTAB, IEBIMCOP, IEBIMGRA, IEBIMCHA

Procedures: FCBPRT, TABLPRT, COPYPRT, GCMPRT, CHARPRT

7 Print module formatted (routine address in operation table).

FCB
TABLE
COPYMOD
GRAPHIC
CHARSET

Routine names are indicated in brackets [ ].

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEBIMAGE PROGRAM MODULES, SUBROUTINES AND SEGMENTS

Module	Subroutines	Segments
IEBIMAGE	ABEND, BKSZCK, BKSZFIX, BLDL, CONVERT, DELCHK, EODAD, INCLUDE, INVOP, LOADLIB, OPTIONX, NAME, PRINTQ, PRTCARD, READLIB, RESET, SYNAD	INITAREA, OPEN, READONLY, USEPARM, UT1OPEN
IEBIMCHA	CHARCHK, CHARSET, CHARPRT, CHARCHK3	CHARINCL, SETCREF, GCMPRTCH, GCMPRTHD, CHARBLD, BLDCHAR, BLDCHAR3, BLDUNDEF, SEGMSG
IEBIMCOP	COPYCHK, COPYMOD, COPYPRT	COPYINCL, TEXTCOMP, SETTEXT, CSEGPRT, TIMECK, OVERRUN
IEBIMFCB	FCB, FCBCHECK, FCBPRT	FCBBLD, FCBCHECK, FCBMARG, FCBSTAT, INCLUDE
IEBIMGRA	GRAPHIC, GRFCHK, GCMPRT, GRFCHK3	GRAFINCL, SETREF, GRAFBLD, GRAFBLD3, GCMPRTCH, GCMPRTHD, SEGMSG, BLDCHAR, BLDCHAR3
IEBIMTAB	TABLE, TABCHK, TABLPRT	TABLINIT, SETLOC, TBLDUPRF
IEBIMWRT	WRITE, WRBLOCK, WRTEXT	WRITELIB, WRBLDL, WRTCESD, STOWMSG, EXTENT
IEBIMSGS		
IEBISCAN	GETCARDN, SCAN, OPSCAN, KEYWORD, REPSCAN, KWDEXCEP	SAMPKWD, FINDPARM

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

PROGRAM ORGANIZATION

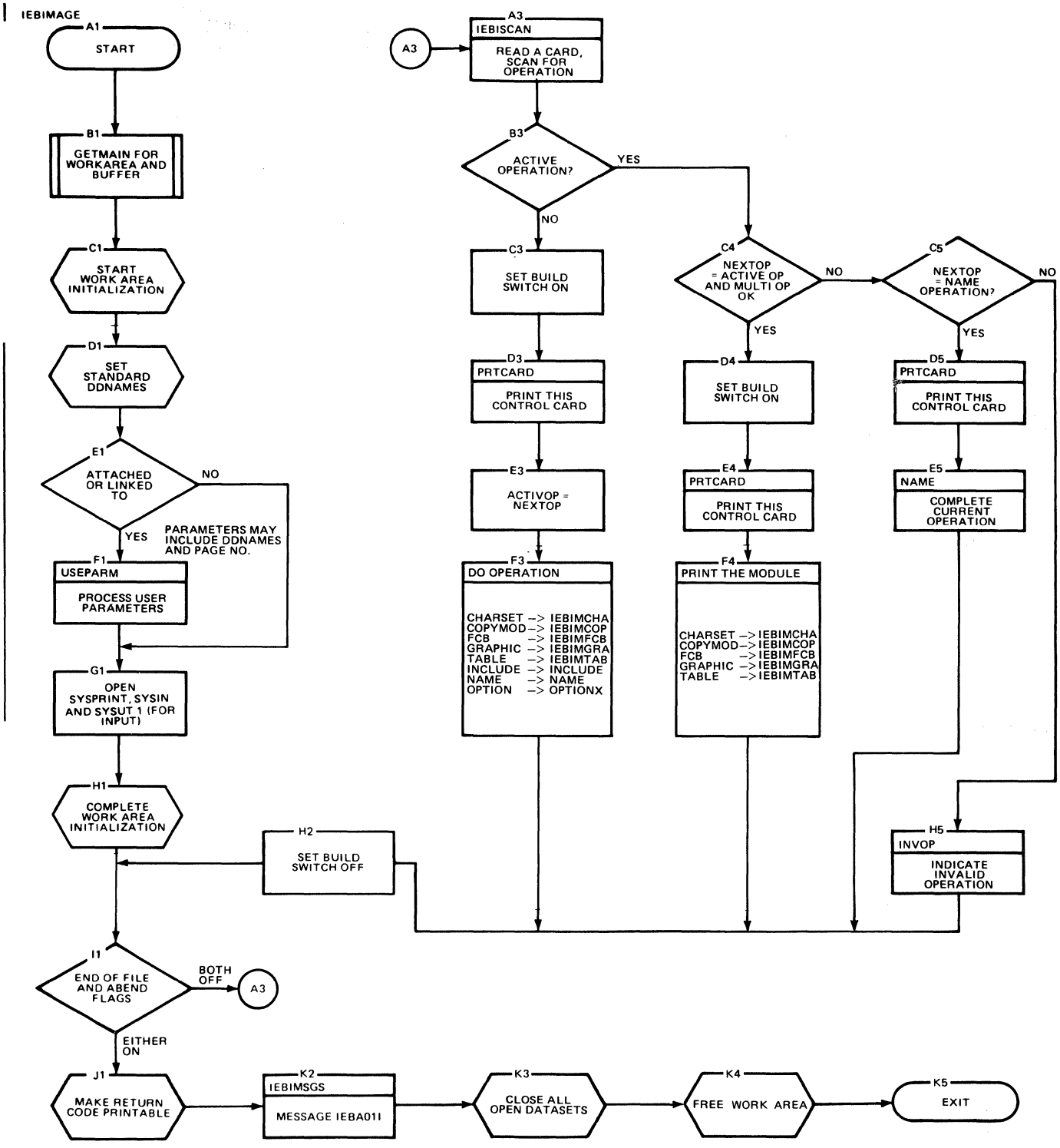
This section contains flowcharts for the printer utility module IEBIMAGE, the modules IEBIMCHA, IEBIMCOP, IEBIMFCB, IEBIMGRA, IEBIMTAB, IEBIMWRT, the card scan module IEBISCAN, and the message print module IEBIMSGS.

The following table lists the various procedures used in the above modules and identifies the chart on which they are described:

Procedure	Chart ID
ABEND	05
BKSZCK	02
BKSZFIX	05
BLDCHAR	13
BLDCHAR3	14
BLDL	06
CHARBLD	12
CHARCHK	15
CHARCHK3	16
CHARINCL	11
CHARPRT	15
CHARSET	10
CONVERT	08
CONVERTX	08
CONVRTER	07
COPCHK	18
COPYINCL	18
COPYMOD	17
COPYMSG	19
COPYPRT	17
EODAD	02
FCB	21
FCBBLD	22
FCBCHECK	22
FCBMARG	23
FCBPRT	23
FCBSTAT	23
FINDPARM	37
GCMPRT	25
GETCARDN	34
GRAPHIC	24
GRFCHK	25
GRFCHK3	26
IEBIMAGE	01
IEBIMCHA	09
IEBIMCOP	17
IEBIMFCB	21
IEBIMGRA	24
IEBIMTAB	27
IEBIMWRT	31
IEBIMSGS	33
IEBISCAN	34
INCLUDE	04
INVOP	04
KEYWORD	35
KWDEXCEP	36
NAME	03
OPSCAN	35
OPTIONX	02
OVERRUN	19
PRINTQ	02
PRTCARD	02
READLIB	06
READONLY	06
REPSCAN	35
SAMPKWD	37
SCAN	37
SEGMSG	16
SETLOC	30
SETCREF	11
SETREF	24
SETTEXT	20
STOWMSG	32
SYNAD	05
TABCHK	28
TABLE	27
TABLINIT	30
TABLPRT	29
WRBLDL	31
WRBLOCK	32
WRITE	02
WRTEXT	32

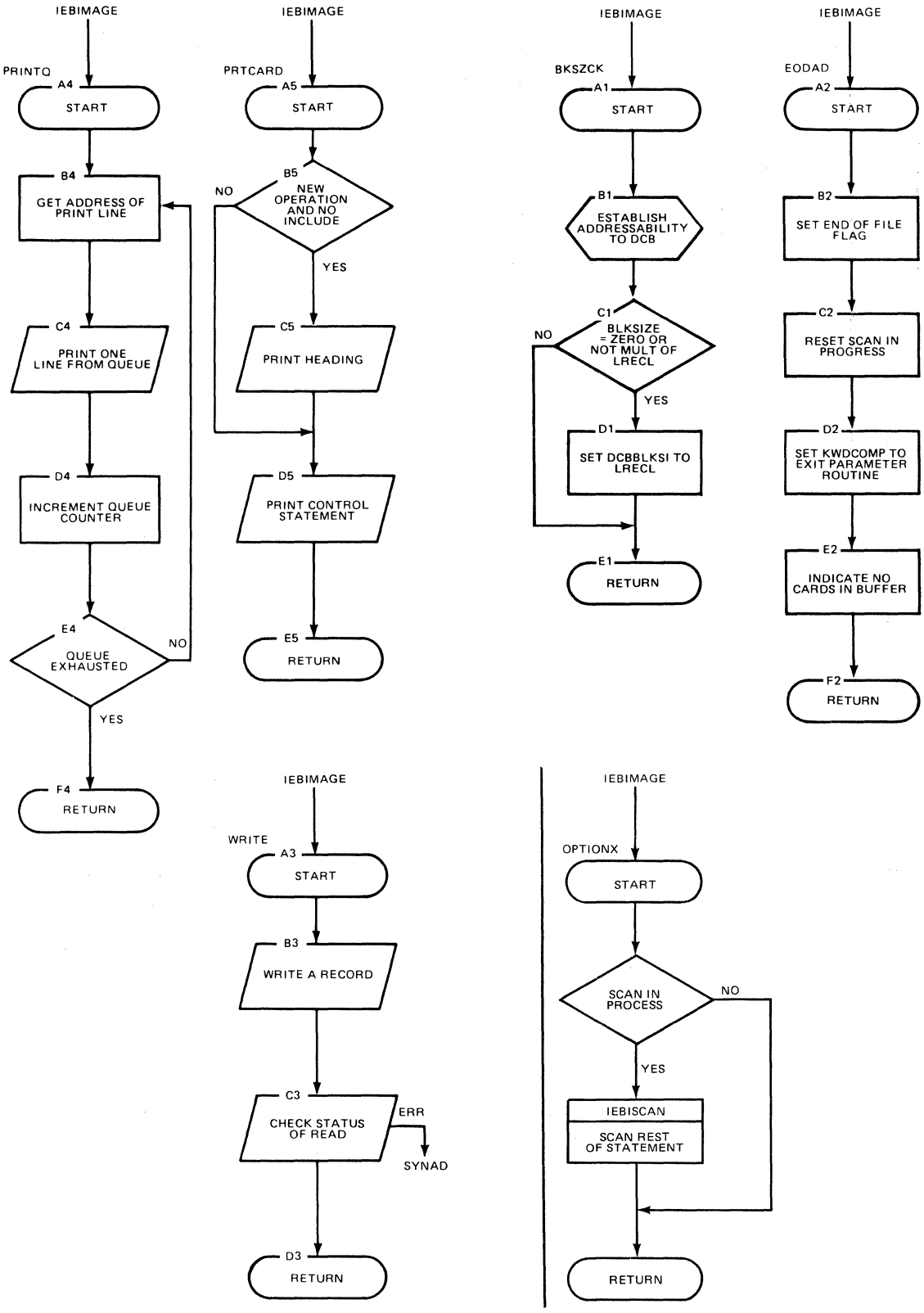
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 01. IEBIMAGE MODULE (PART 1 OF 8): MAIN FLOW



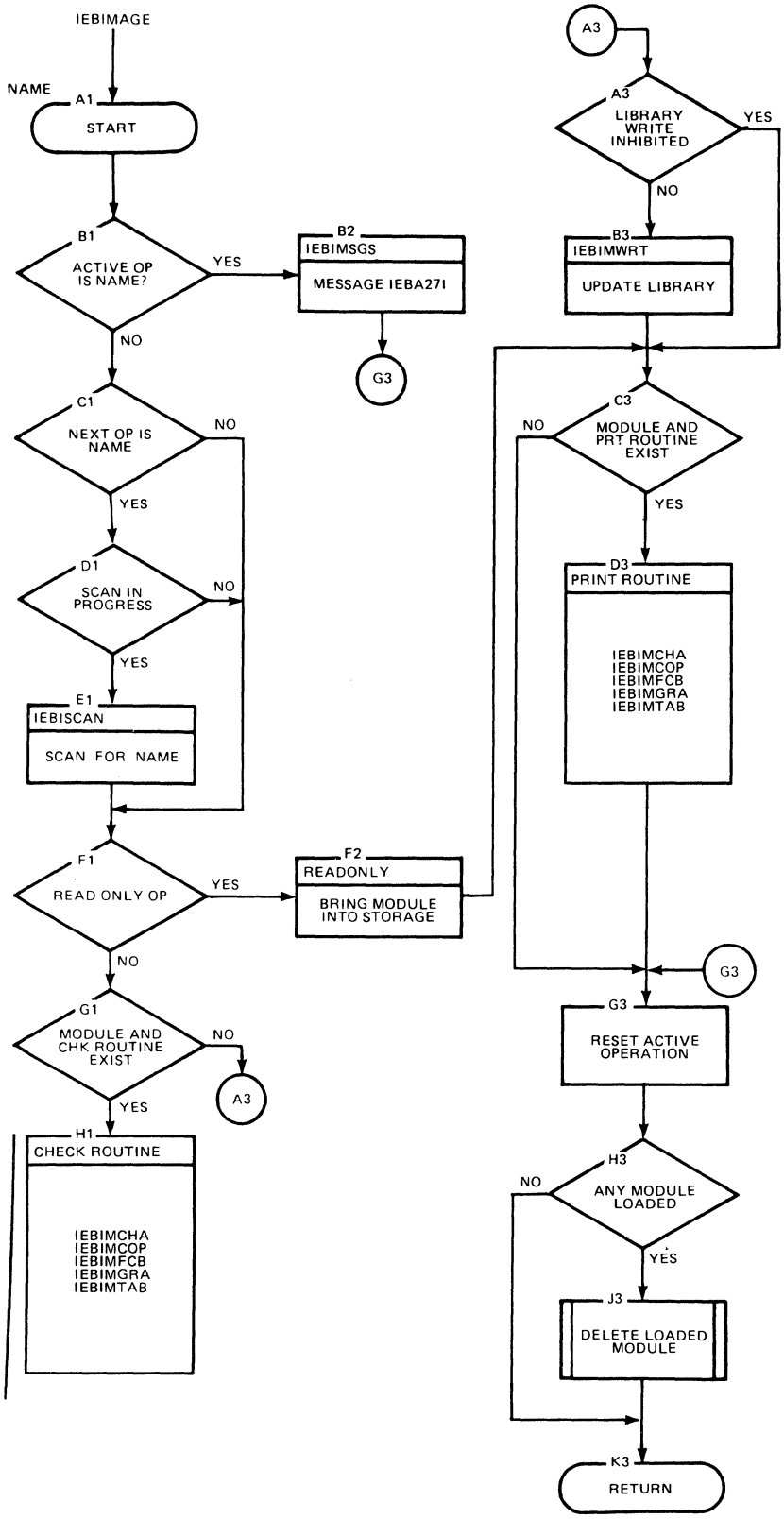
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 02. IEBIMAGE MODULE (PART 2 OF 8): PRINTQ, PRTCARD, BKSZCK, EODAD, WRITE, OPTIONX



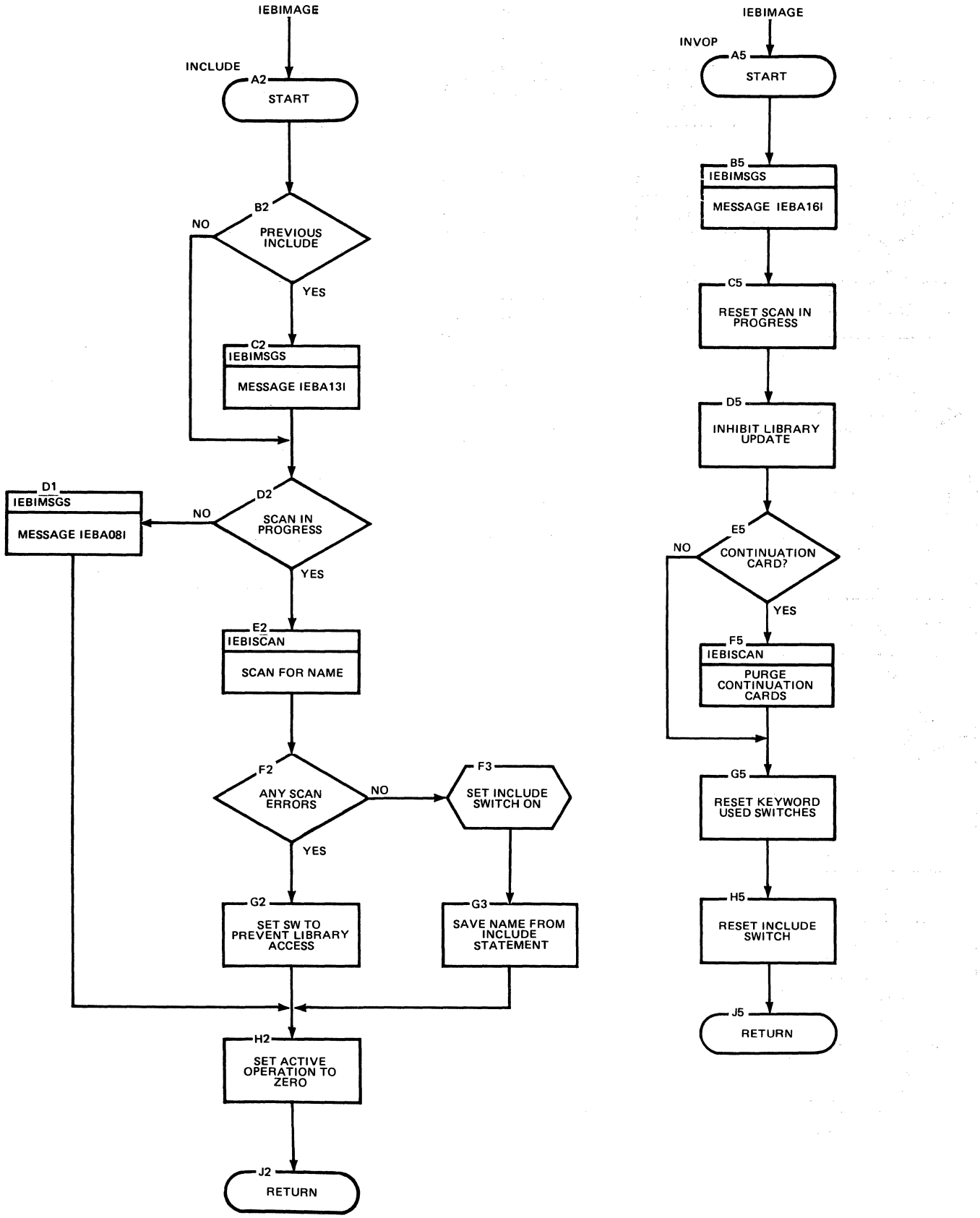
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 03. IEBIMAGE MODULE (PART 3 OF 8): NAME



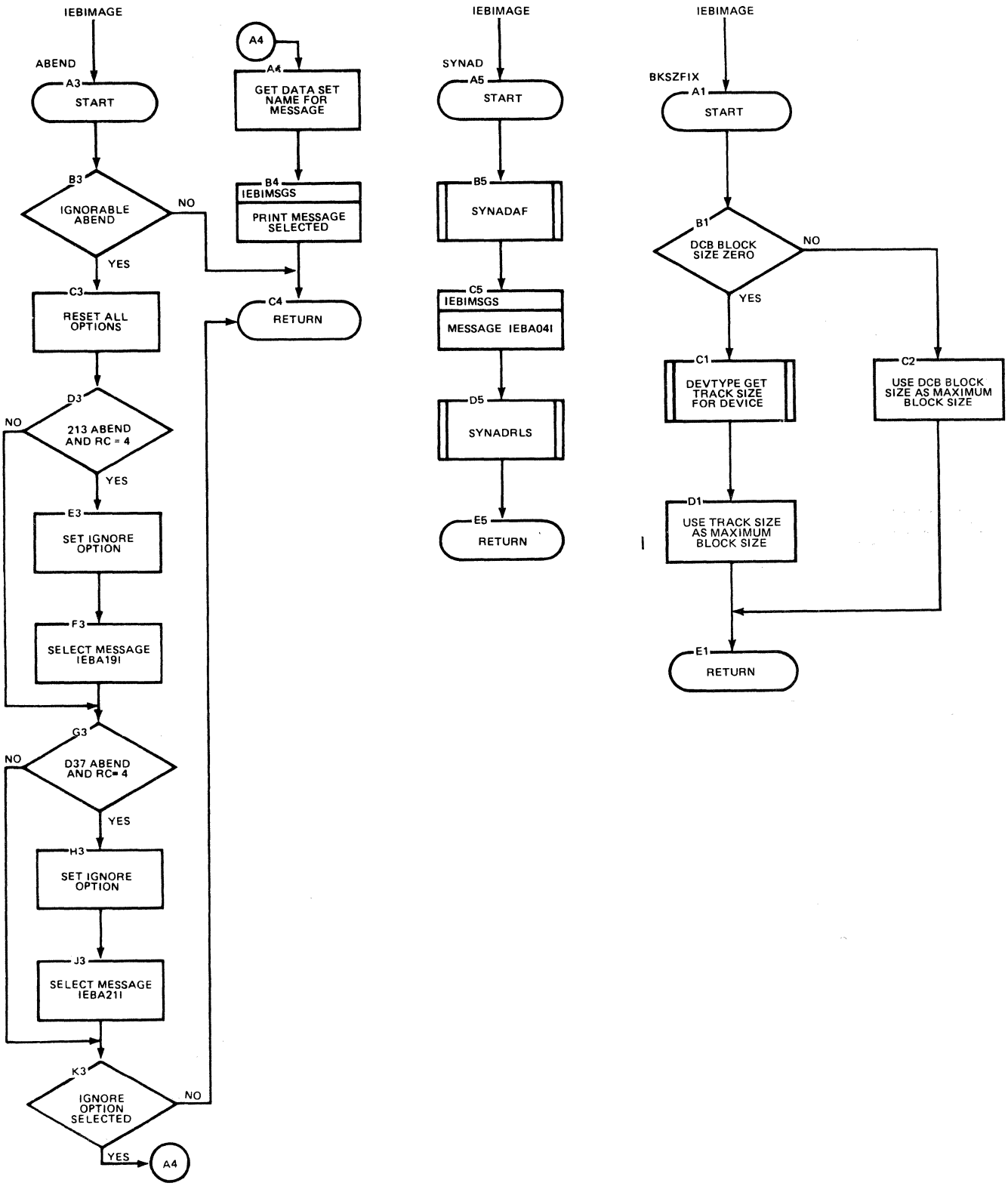
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 04. IEBIMAGE MODULE (PART 4 OF 8): INCLUDE, INVOP



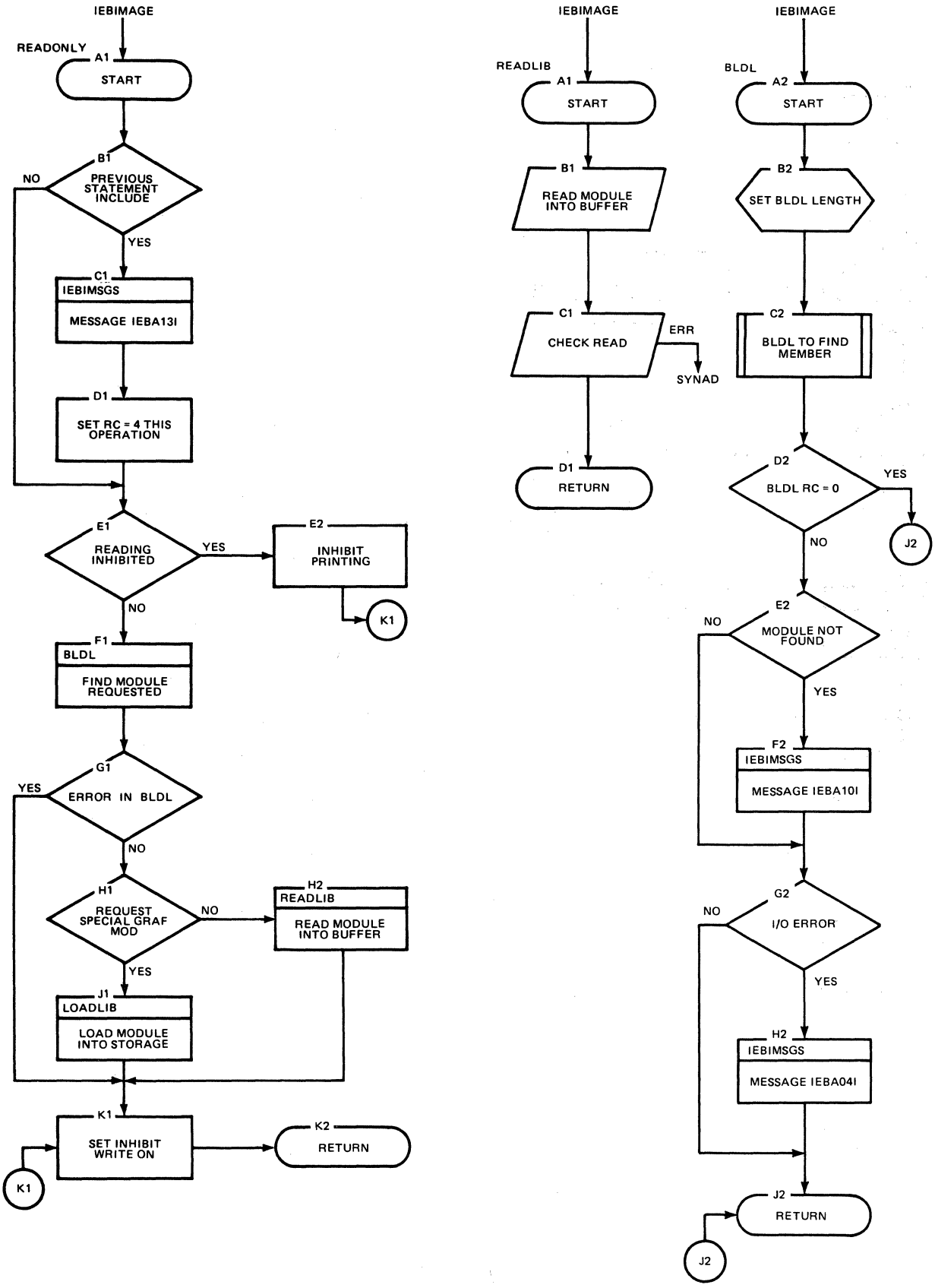
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 05. IEBIMAGE MODULE (PART 5 OF 8): ABEND, SYNAD, BKSZFIX



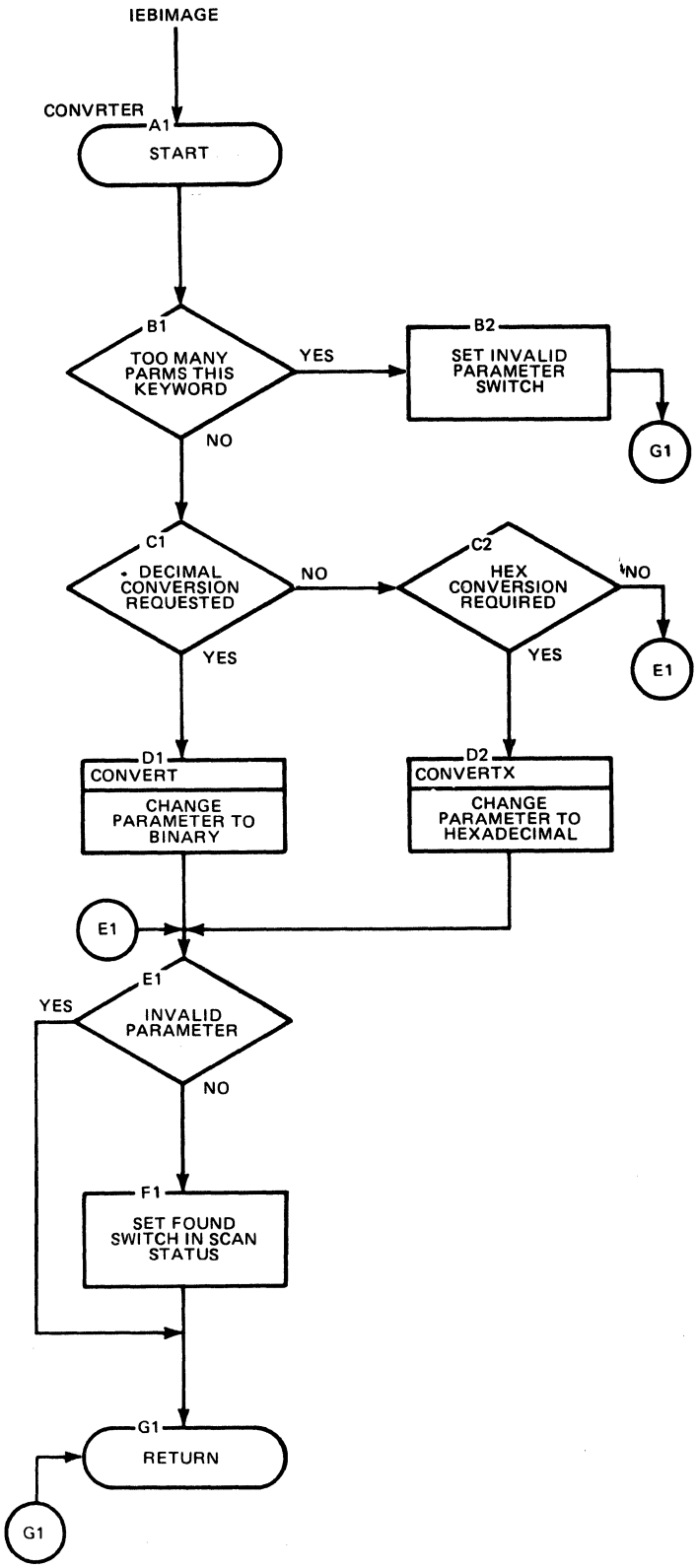
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 06. IEBIMAGE MODULE (PART 6 OF 8): READONLY, READLIB, BLDL



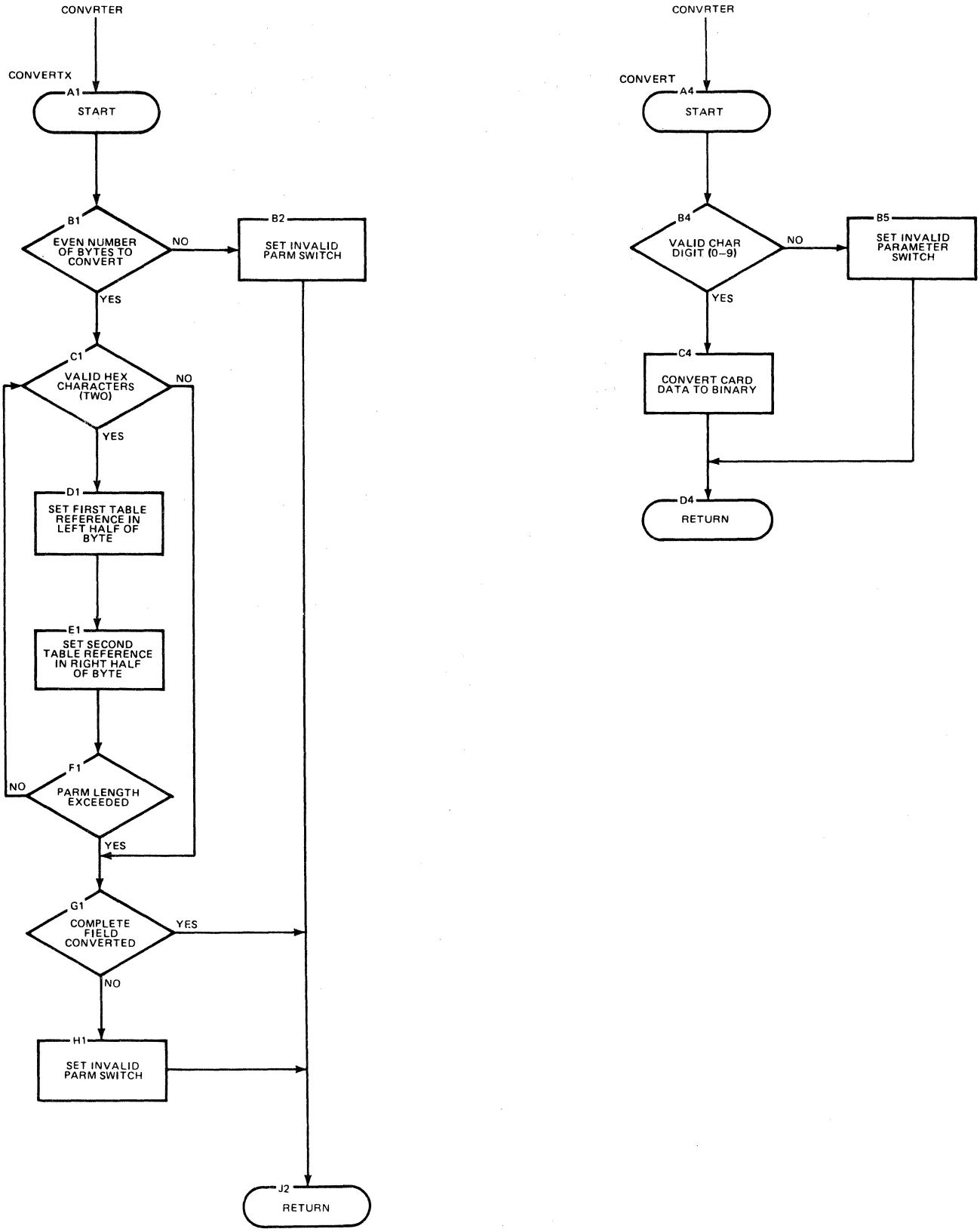
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 07. IEBIMAGE MODULE (PART 7 OF 8): CONVRTER



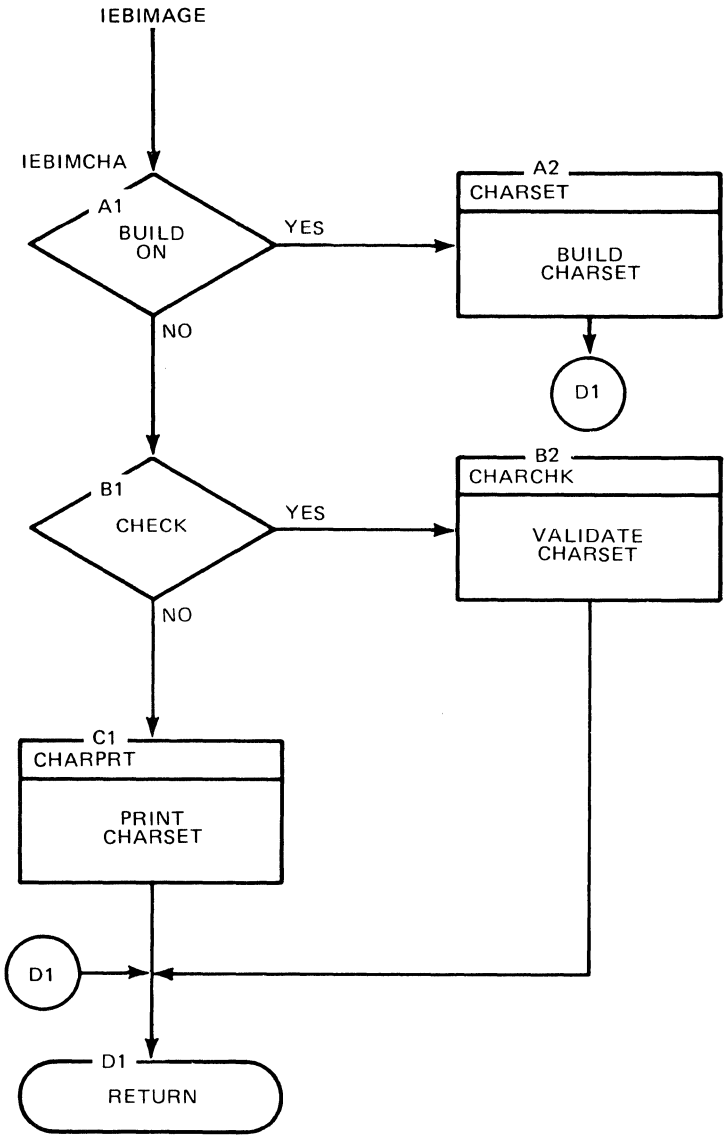
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 08. IEBIMAGE MODULE (PART 8 OF 8): CONVERTX, CONVERT



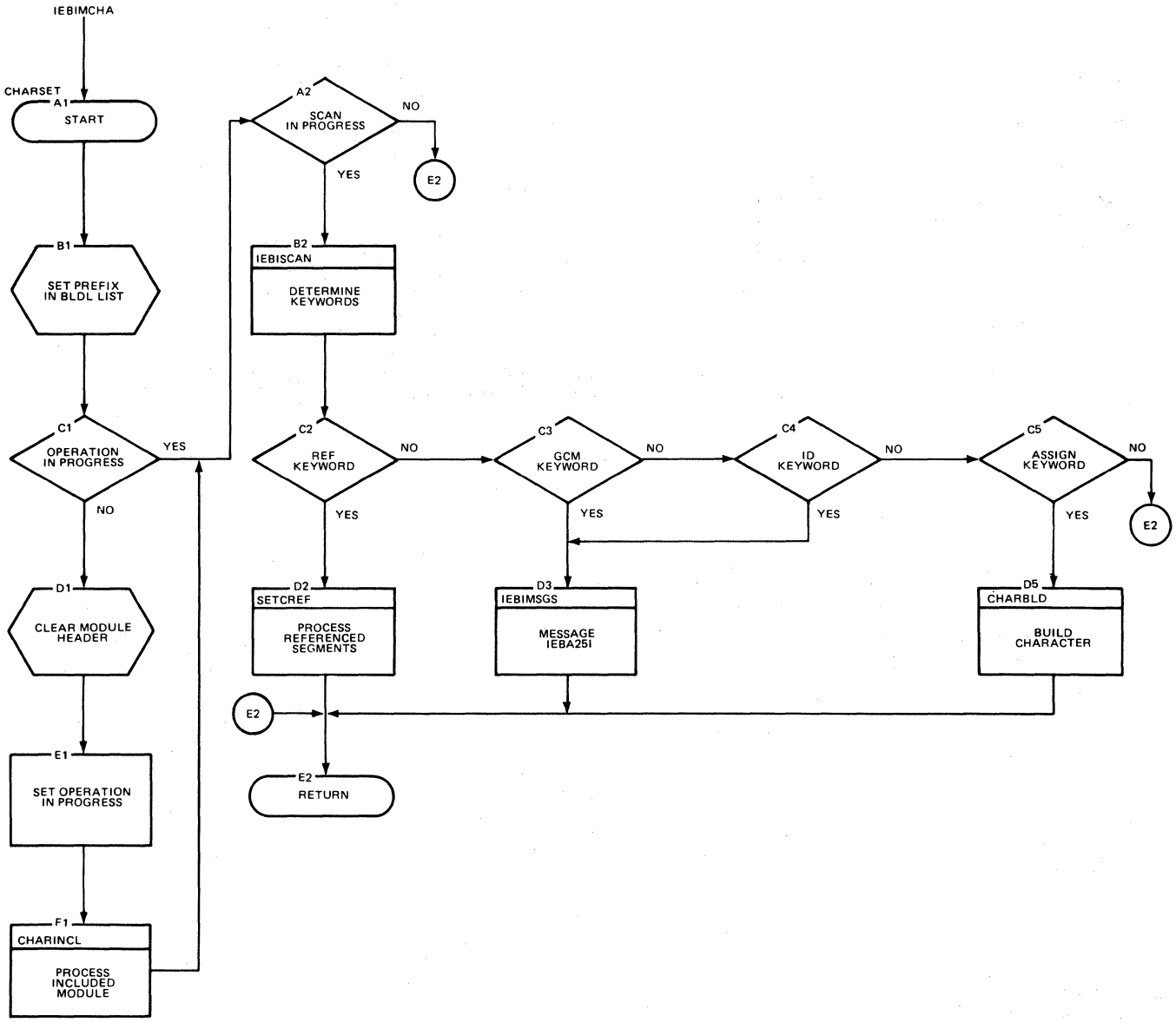
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 09. IEBIMCHA MODULE (PART 1 OF 8): MAIN FLOW



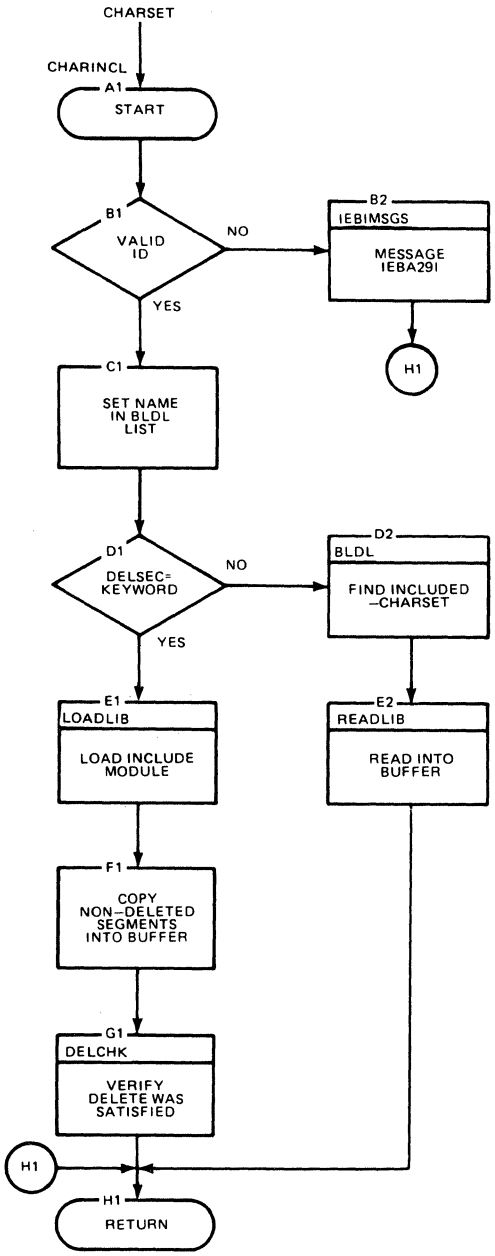
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

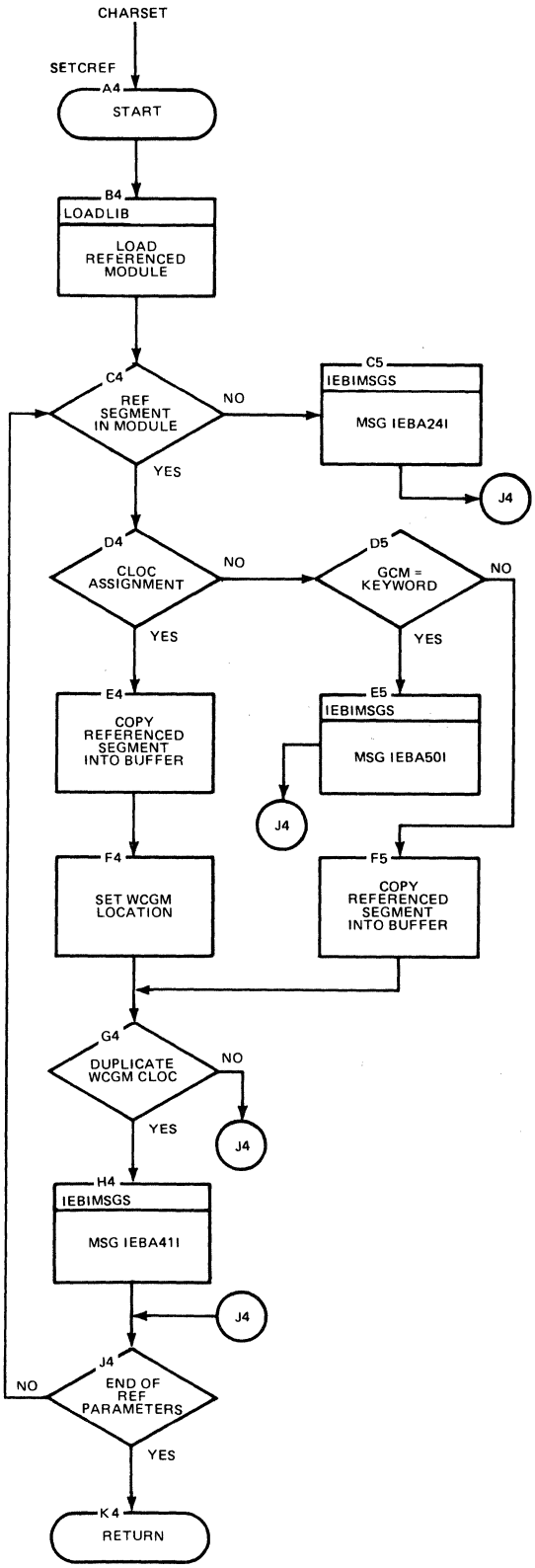
CHART 10. IEBIMCHA MODULE (PART 2 OF 8): CHARSET



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

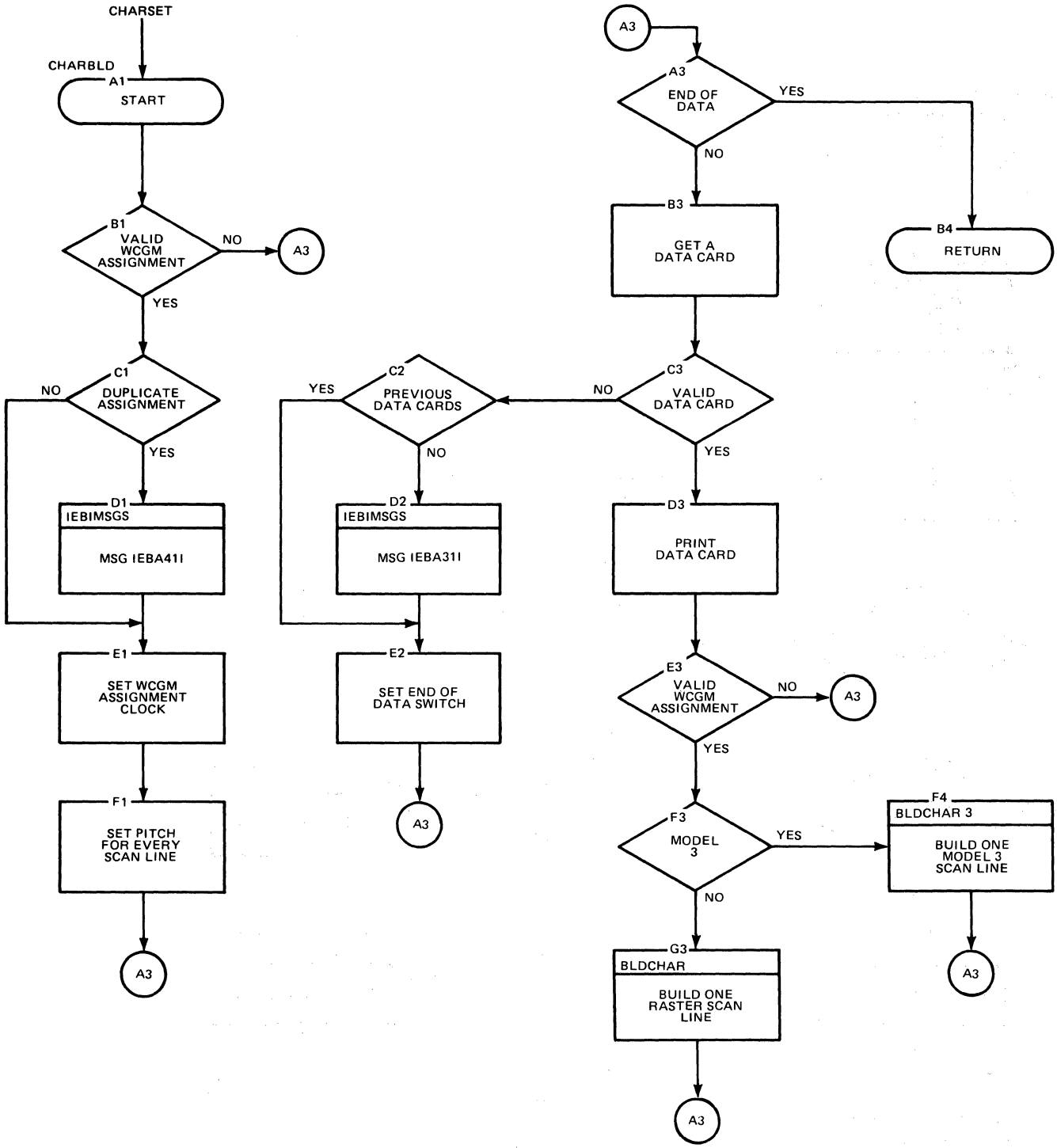
CHART 11. IEBIMCHA MODULE (PART 3 OF 8): CHARINCL, SETCREF





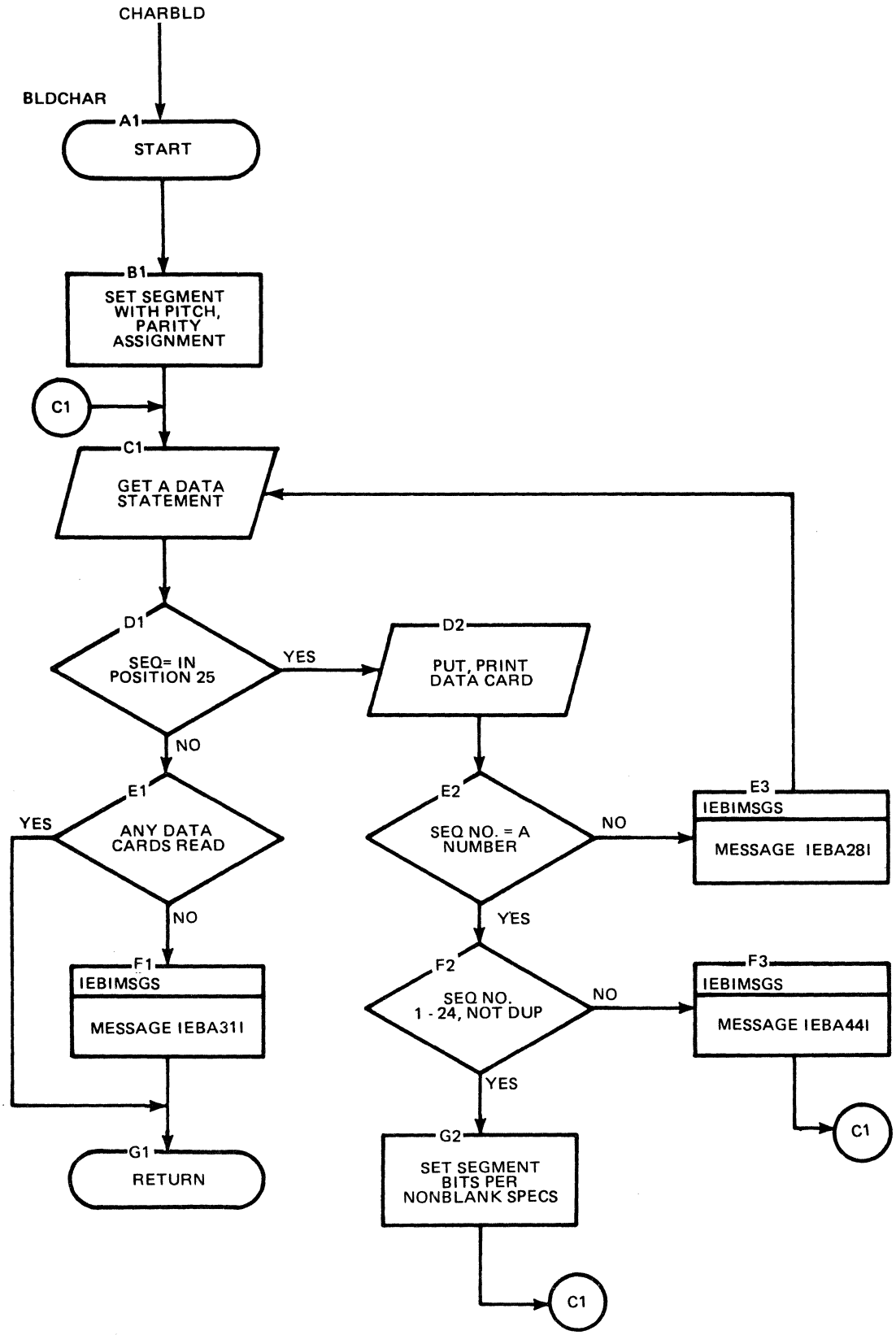
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 12. IEBIMCHA MODULE (PART 4 OF 8): CHARBLD



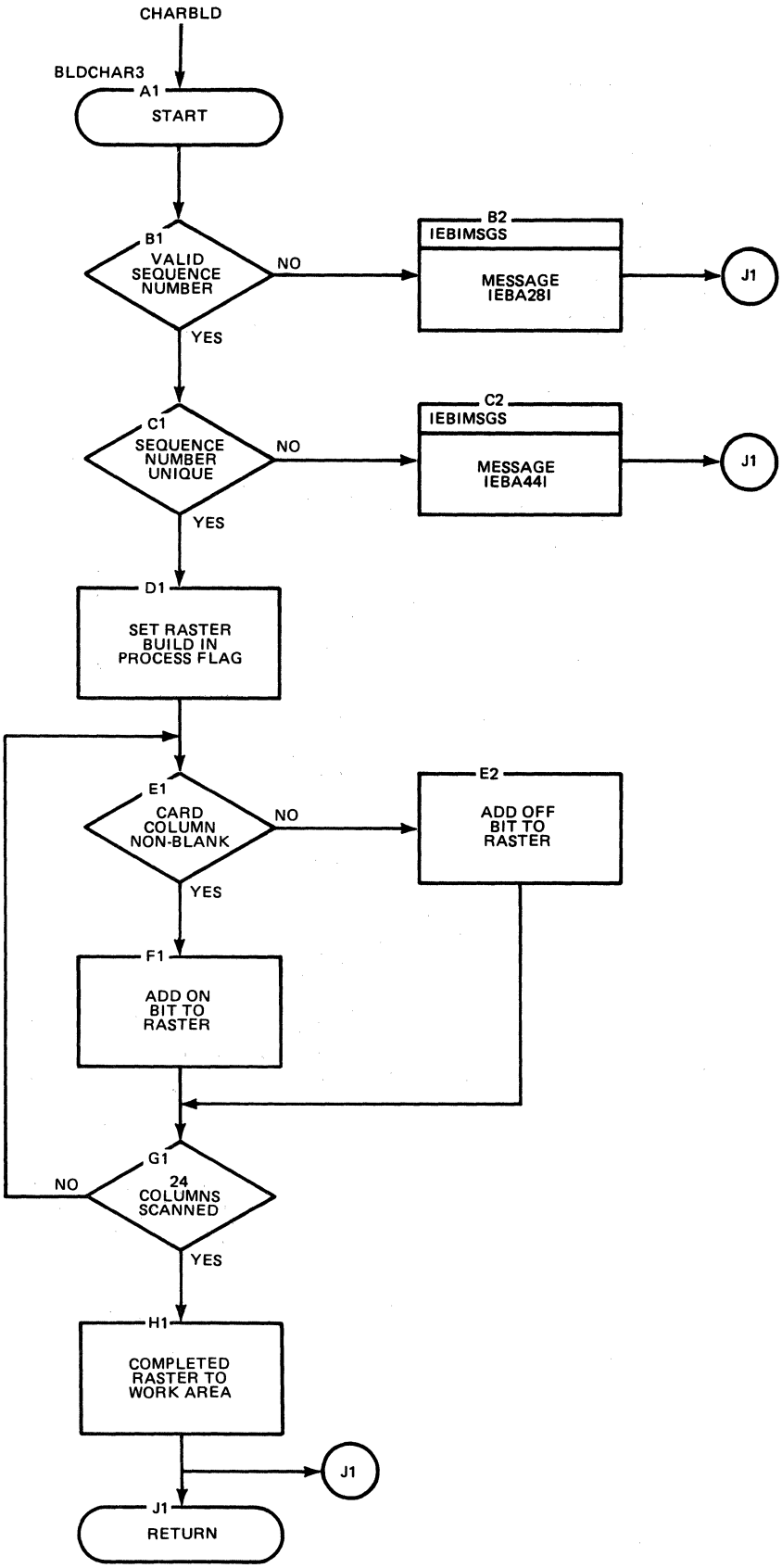
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 13. IEBIMCHA MODULE (PART 5 OF 8): BLDCHAR



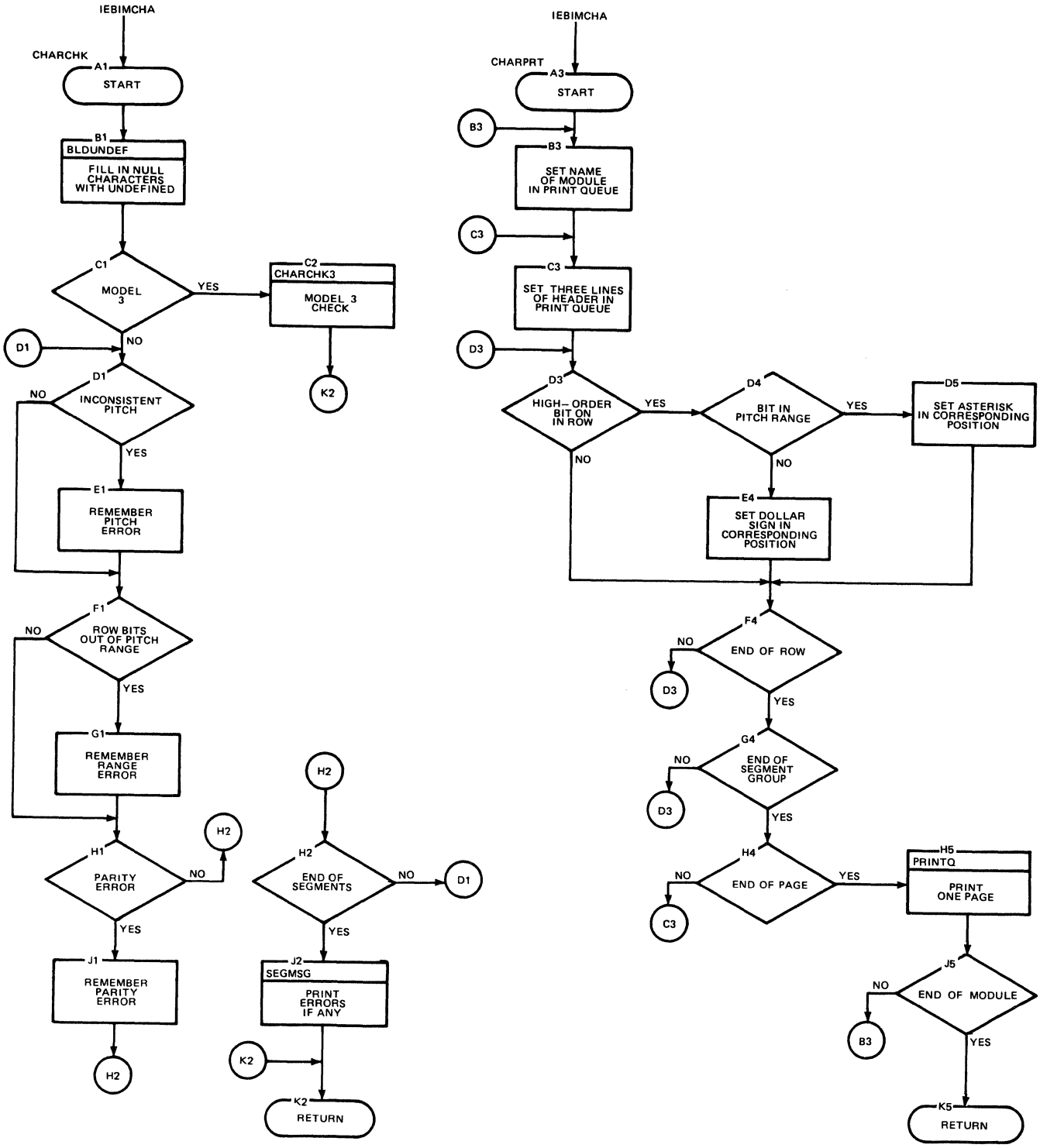
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 14. IEBIMCHA MODULE (PART 6 OF 8): BLDCHAR3



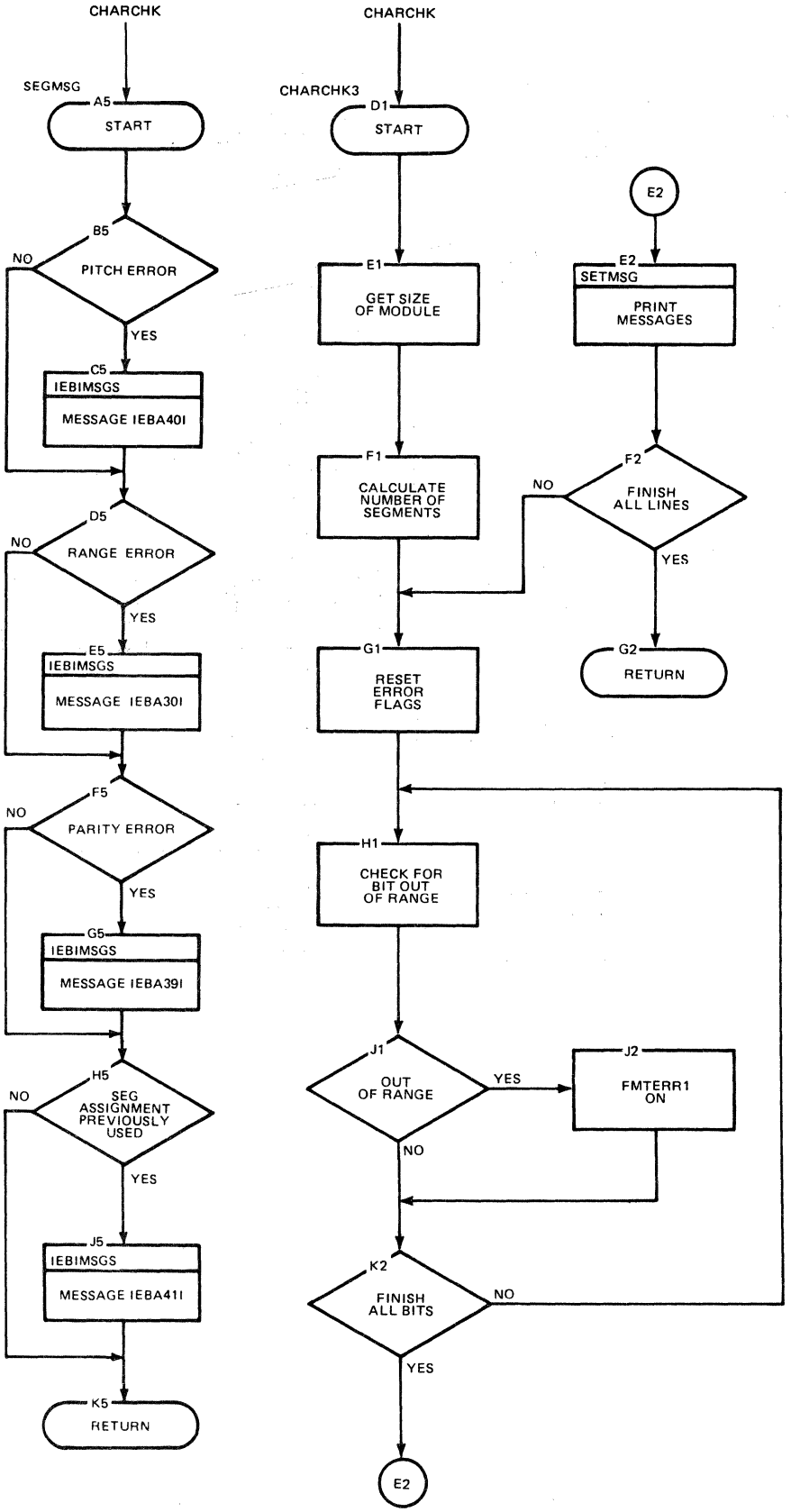
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 15. IEBIMCHA MODULE (PART 7 OF 8): CHARCHK, CHARPRT



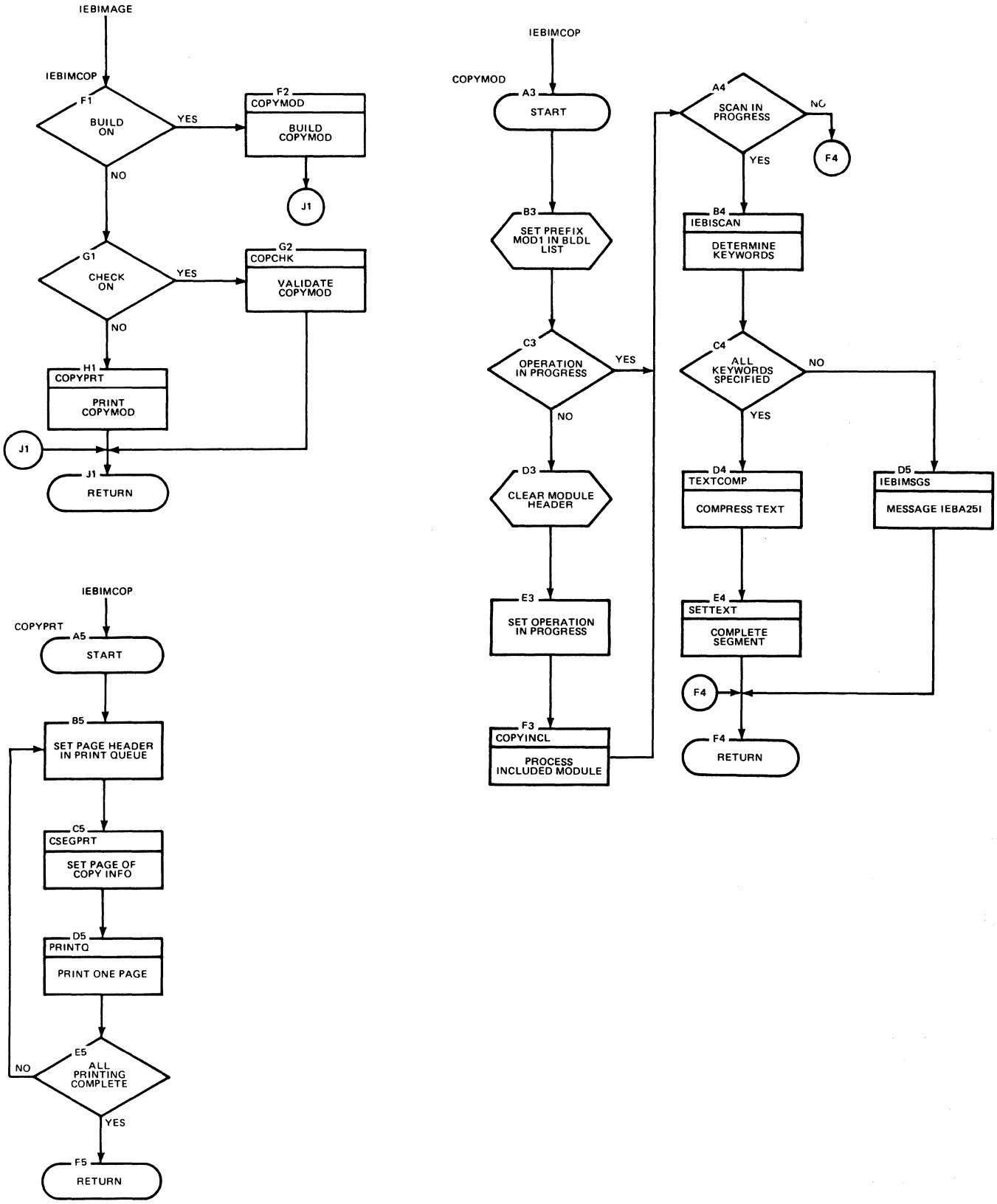
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 16. IEBIMCHA MODULE (PART 8 OF 8): SEGMSG, CHARCHK3



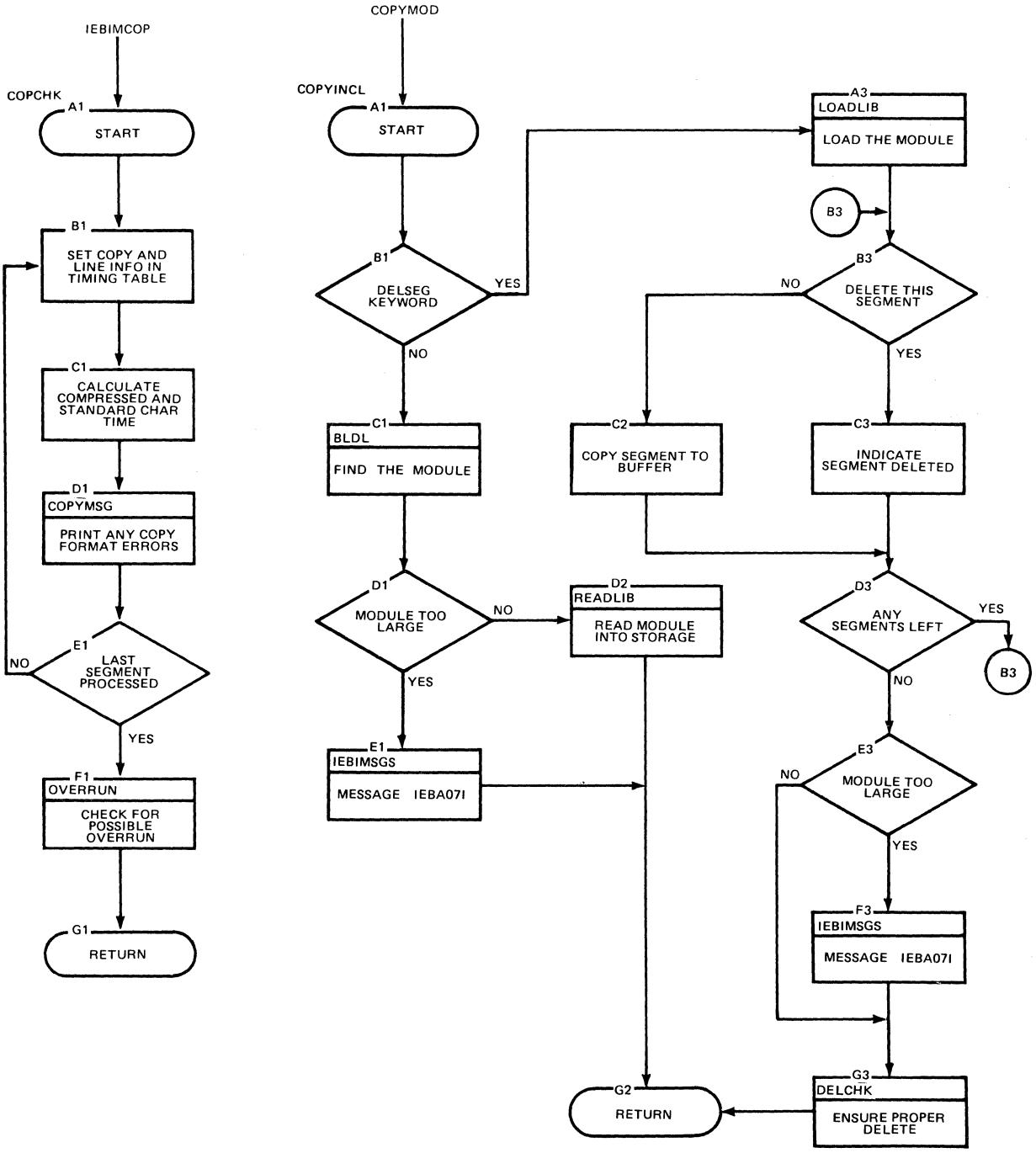
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 17. IEBIMCOP MODULE (PART 1 OF 4): MAIN FLOW, COPYMOD, COPYPRT



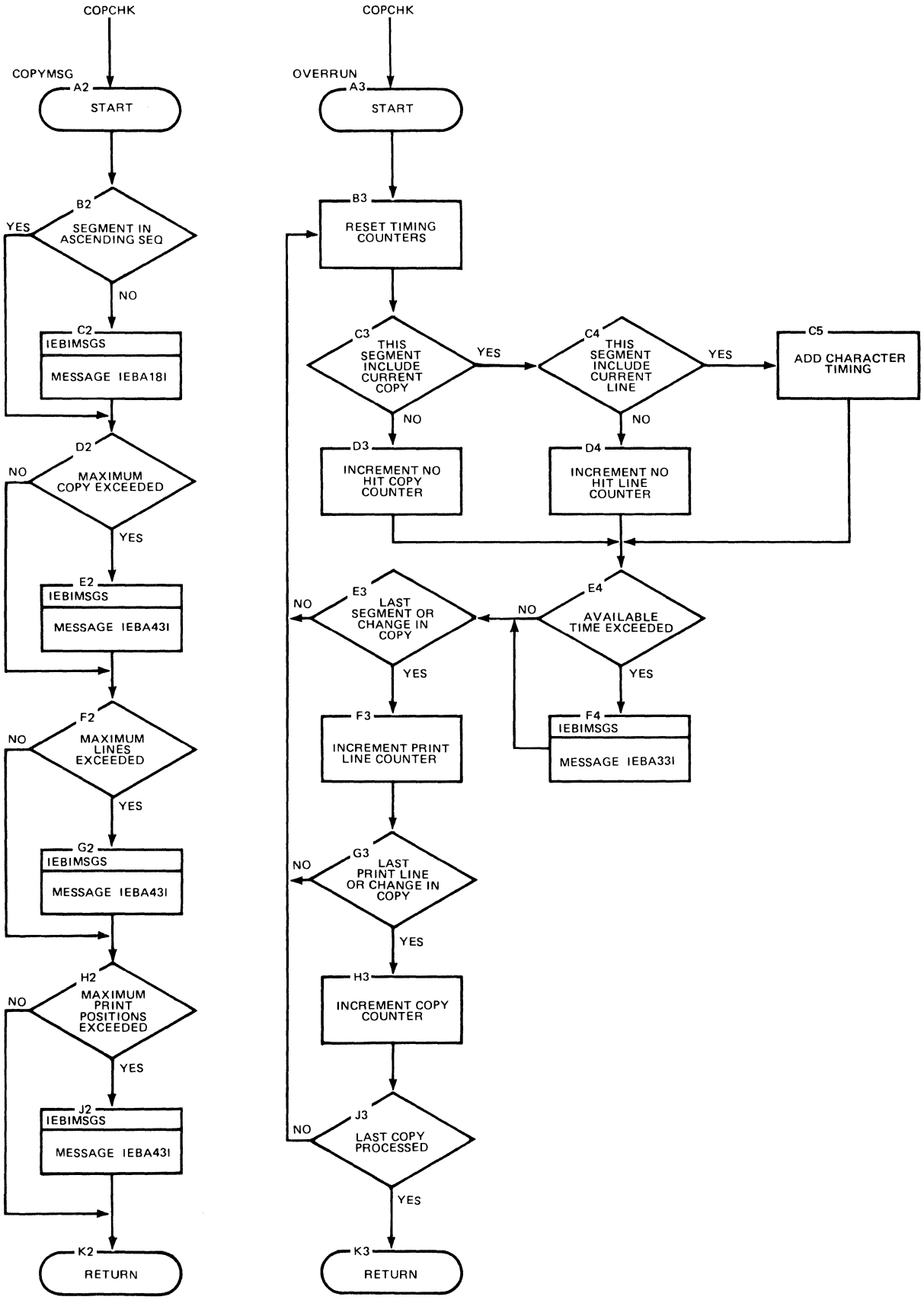
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 18. IEBIMCOP MODULE (PART 2 OF 4): COPCHK, COPYINCL



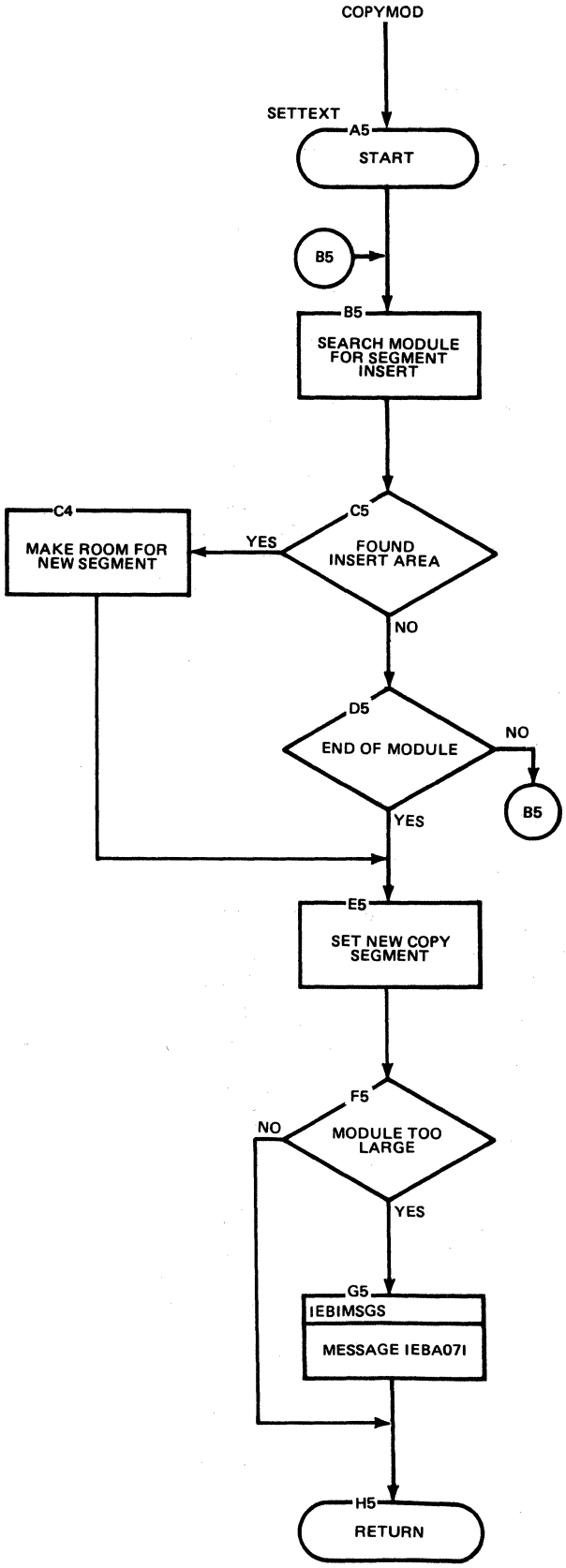
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 19. IEBIMCOP MODULE (PART 3 OF 4): COPYMSG, OVERRUN



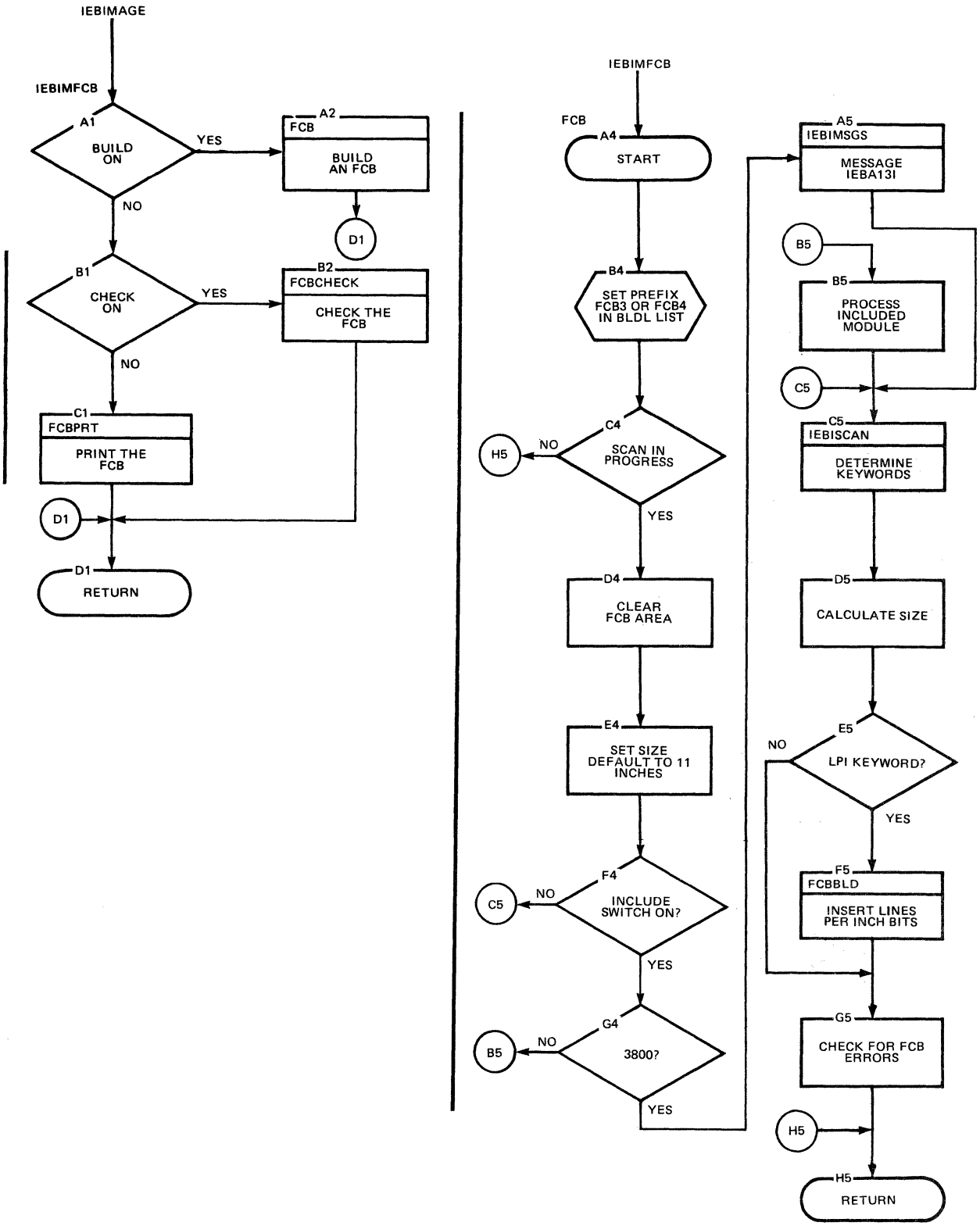
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 20. IEBIMCOP MODULE (PART 4 OF 4): SETTEXT



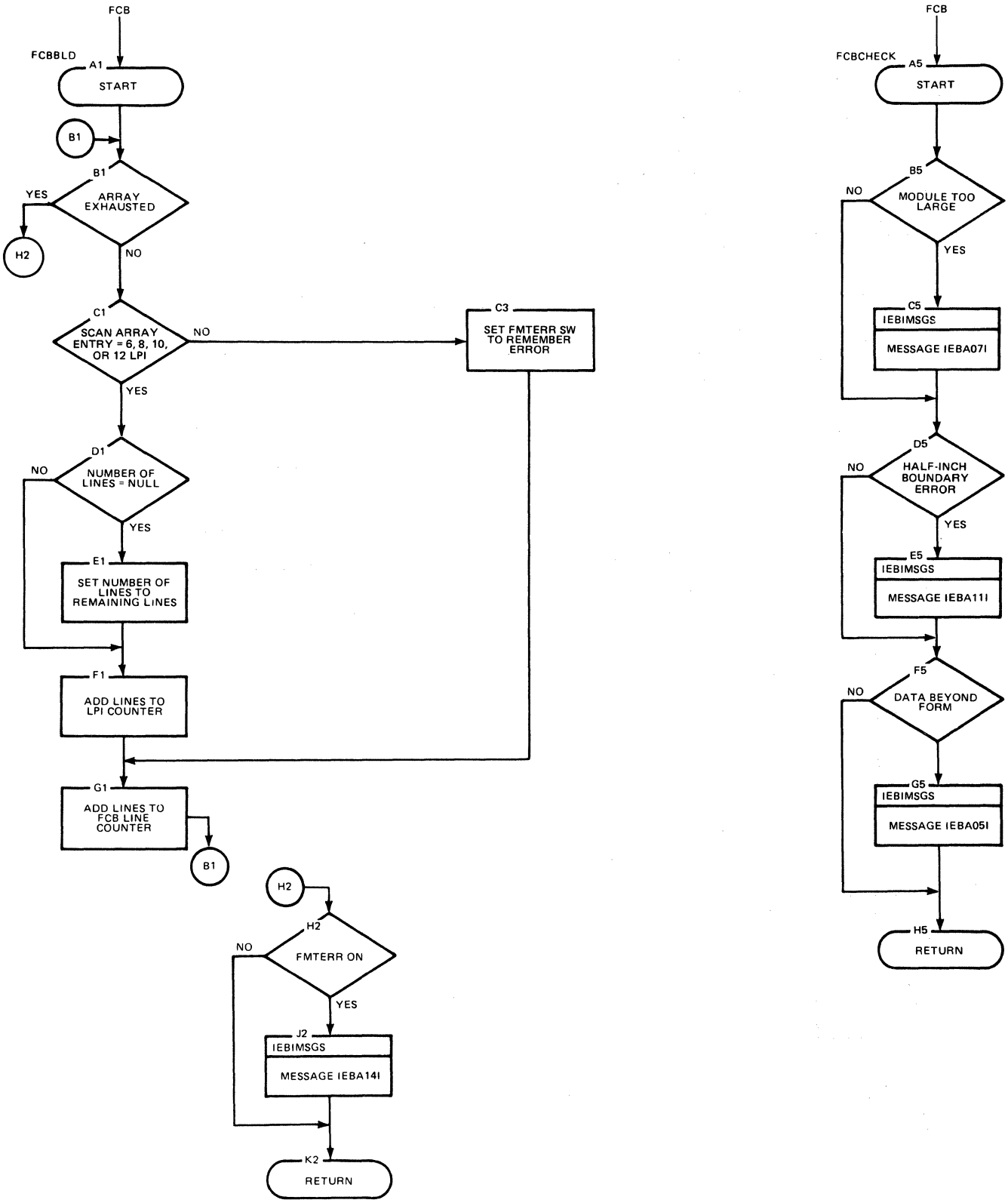
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 21. IEBIMFCB MODULE (PART 1 OF 3): MAIN FLOW, FCB



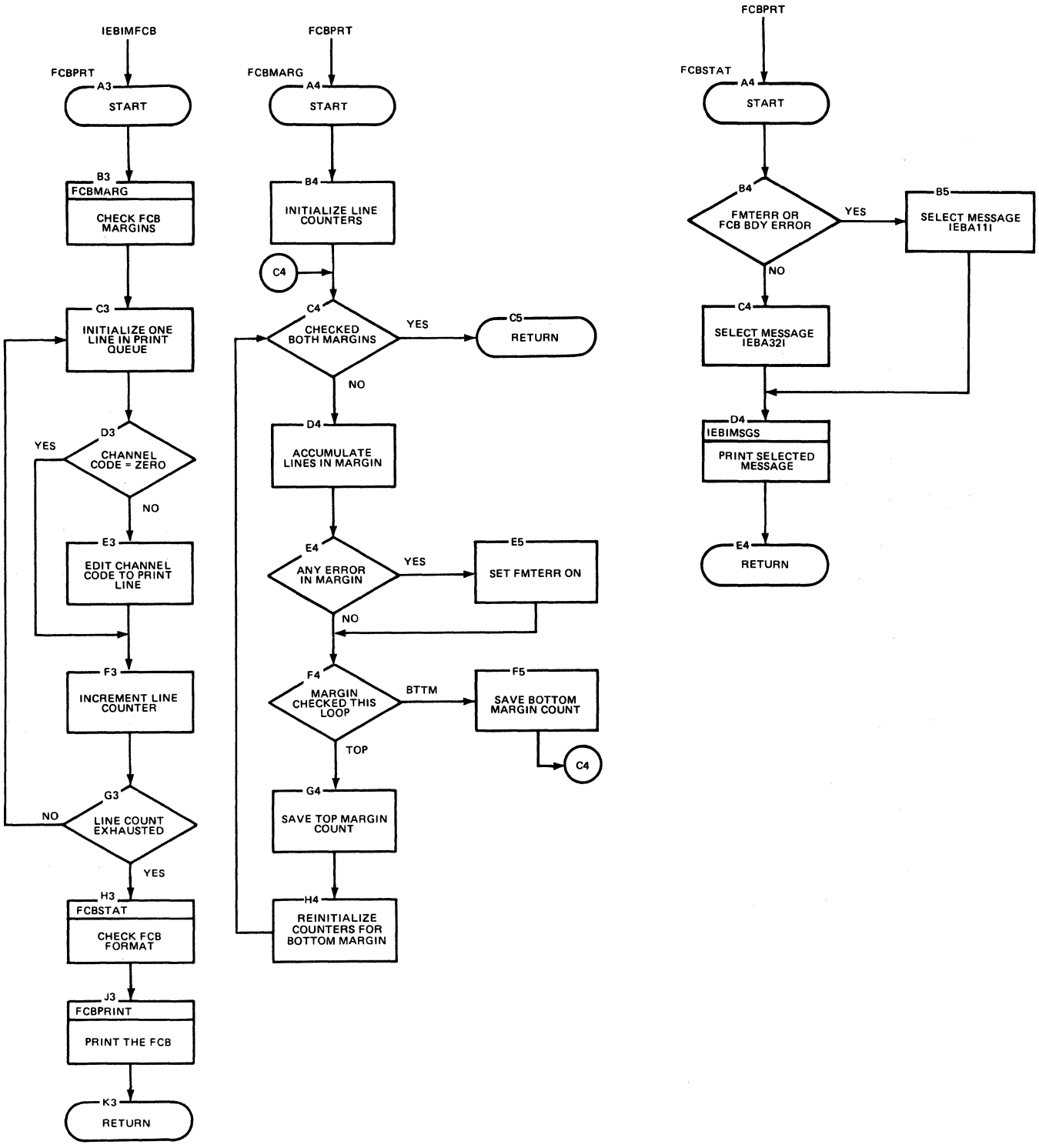
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 22. IEBIMFCB MODULE (PART 2 OF 3): FCBBLD, FCBCHECK



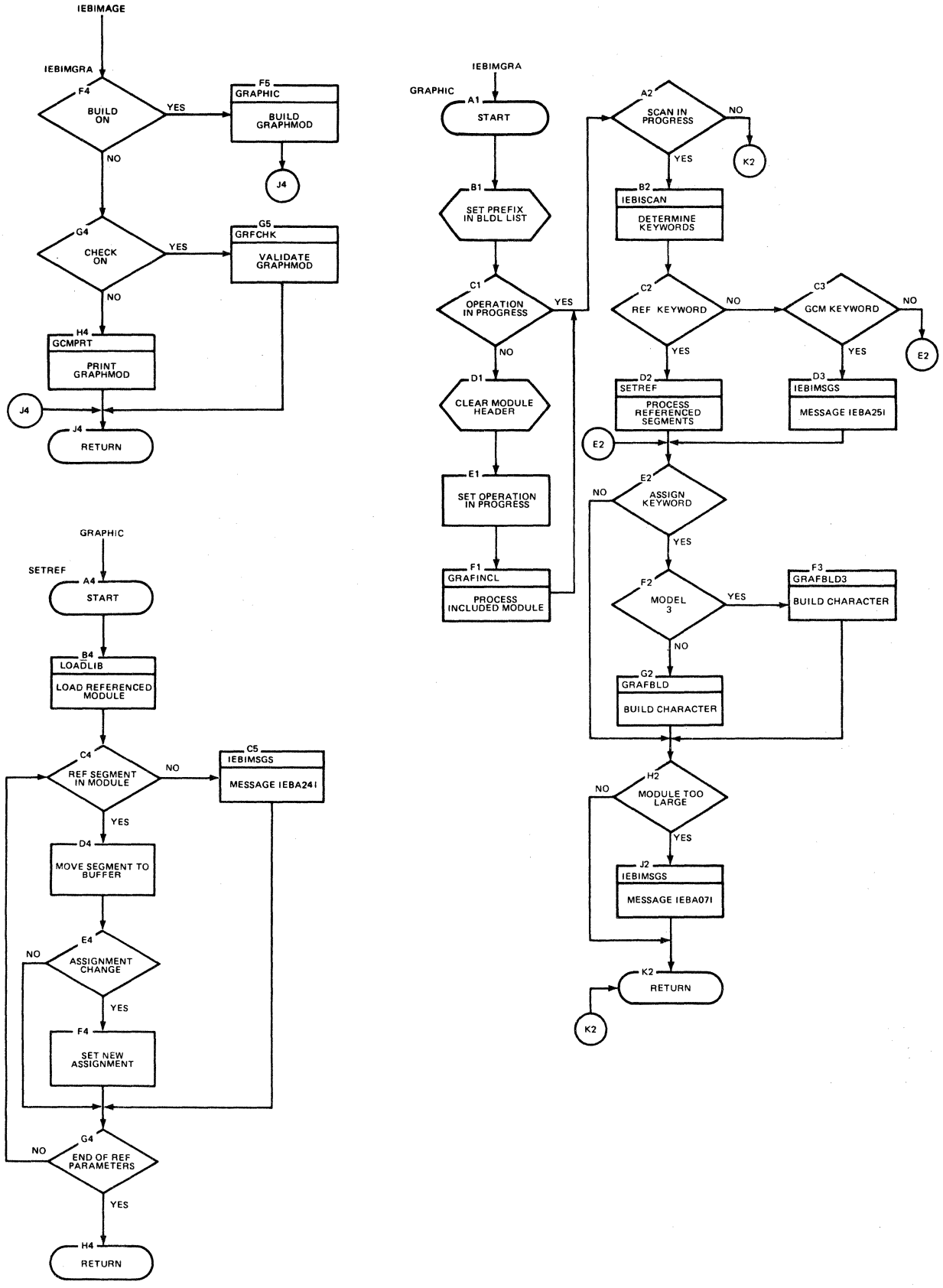
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 23. IEBIMFCB MODULE (PART 3 OF 3): FCBPRT, FCBMARG, FCBSTAT



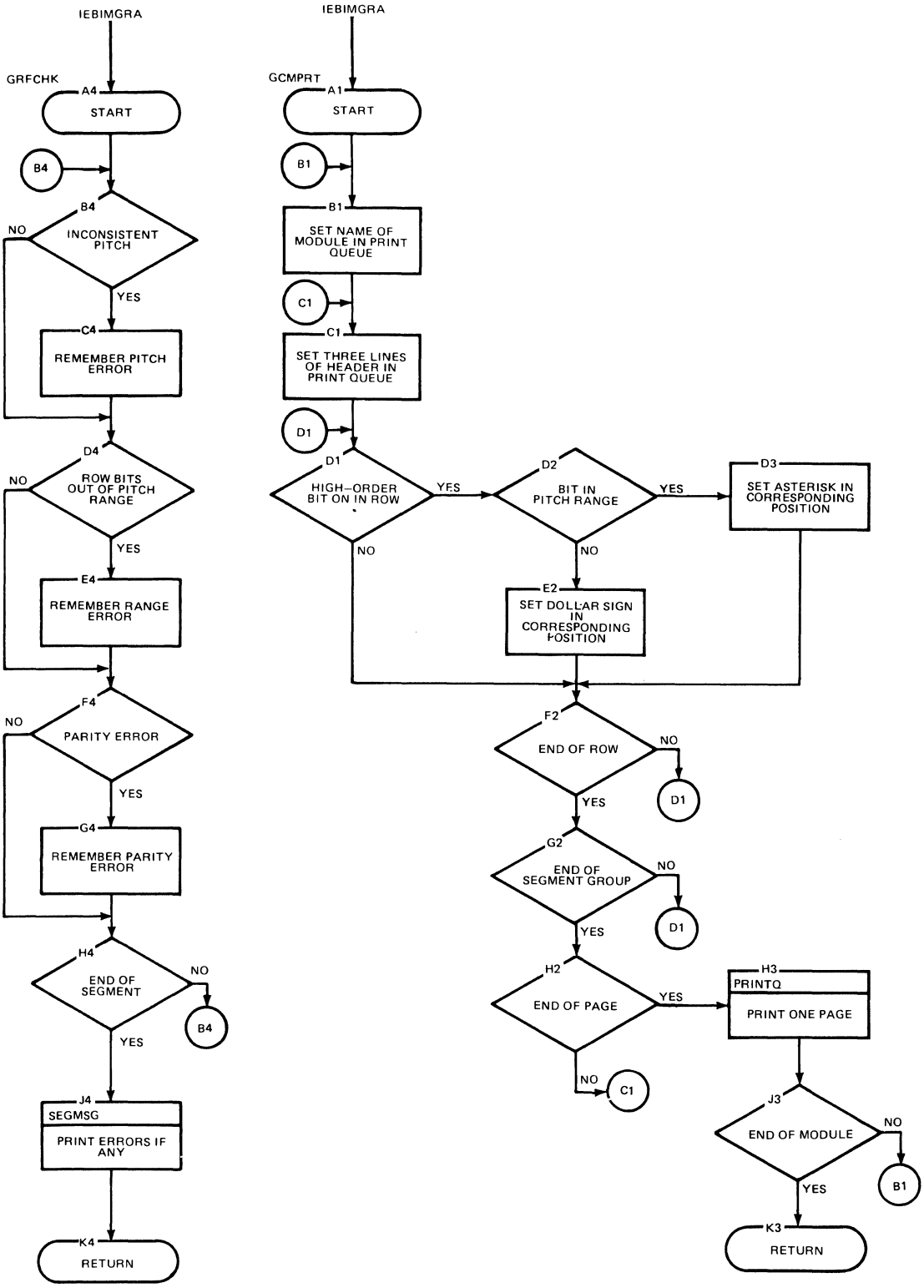
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 24. IEBIMGRA MODULE (PART 1 OF 3): MAIN FLOW, GRAPHIC, SETREF



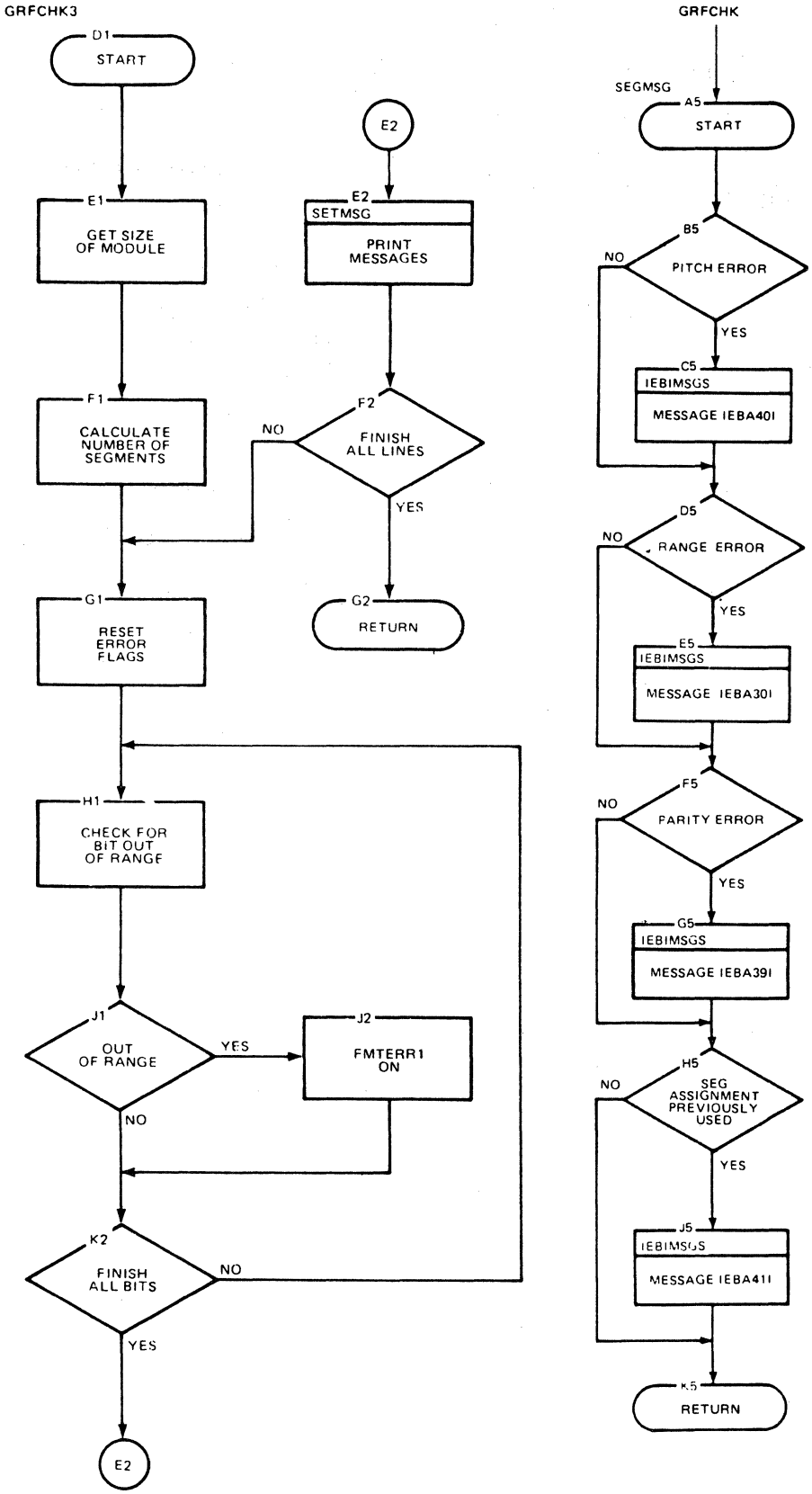
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 25. IEBIMGRA MODULE (PART 2 OF 3): GRFCHK, GCMPRT



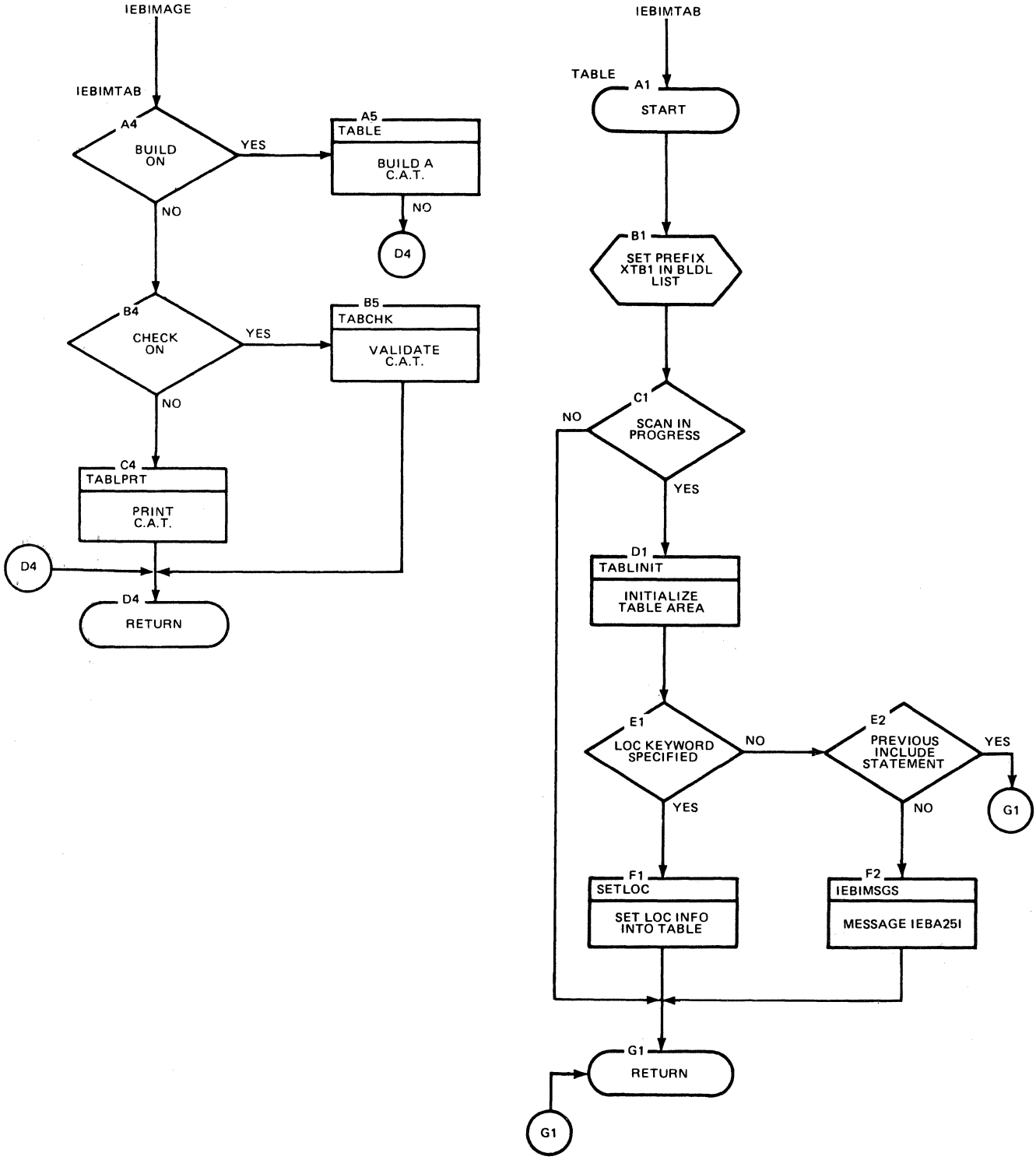
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 26. IEBIMGRA MODULE (PART 3 OF 3): GRFCHK3, SEGMSG



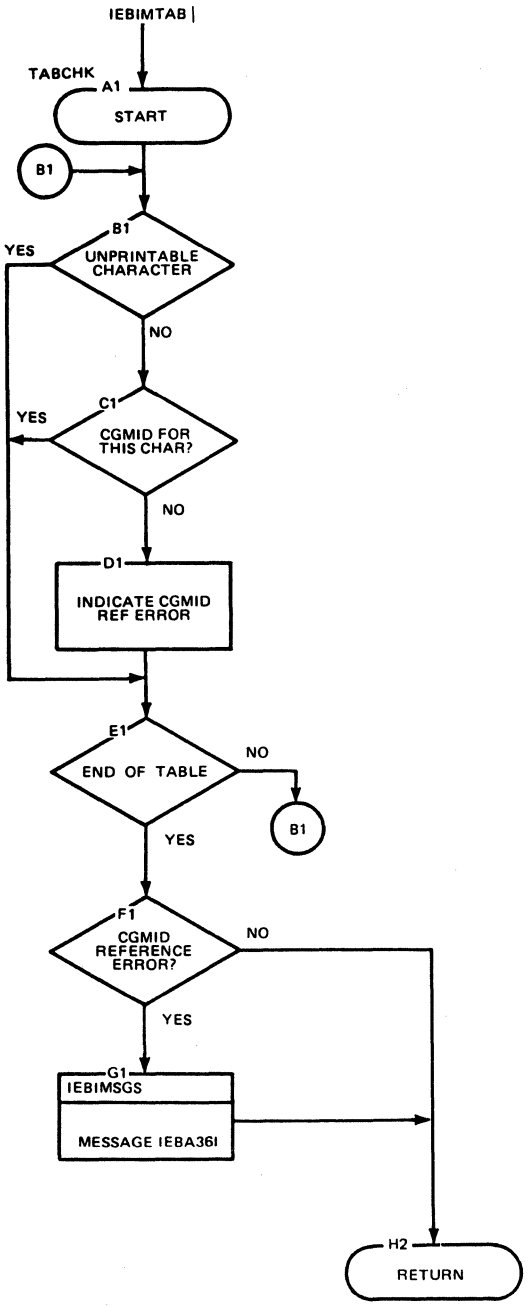
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 27. IEBIMTAB MODULE (PART 1 OF 4): MAIN FLOW, TABLE



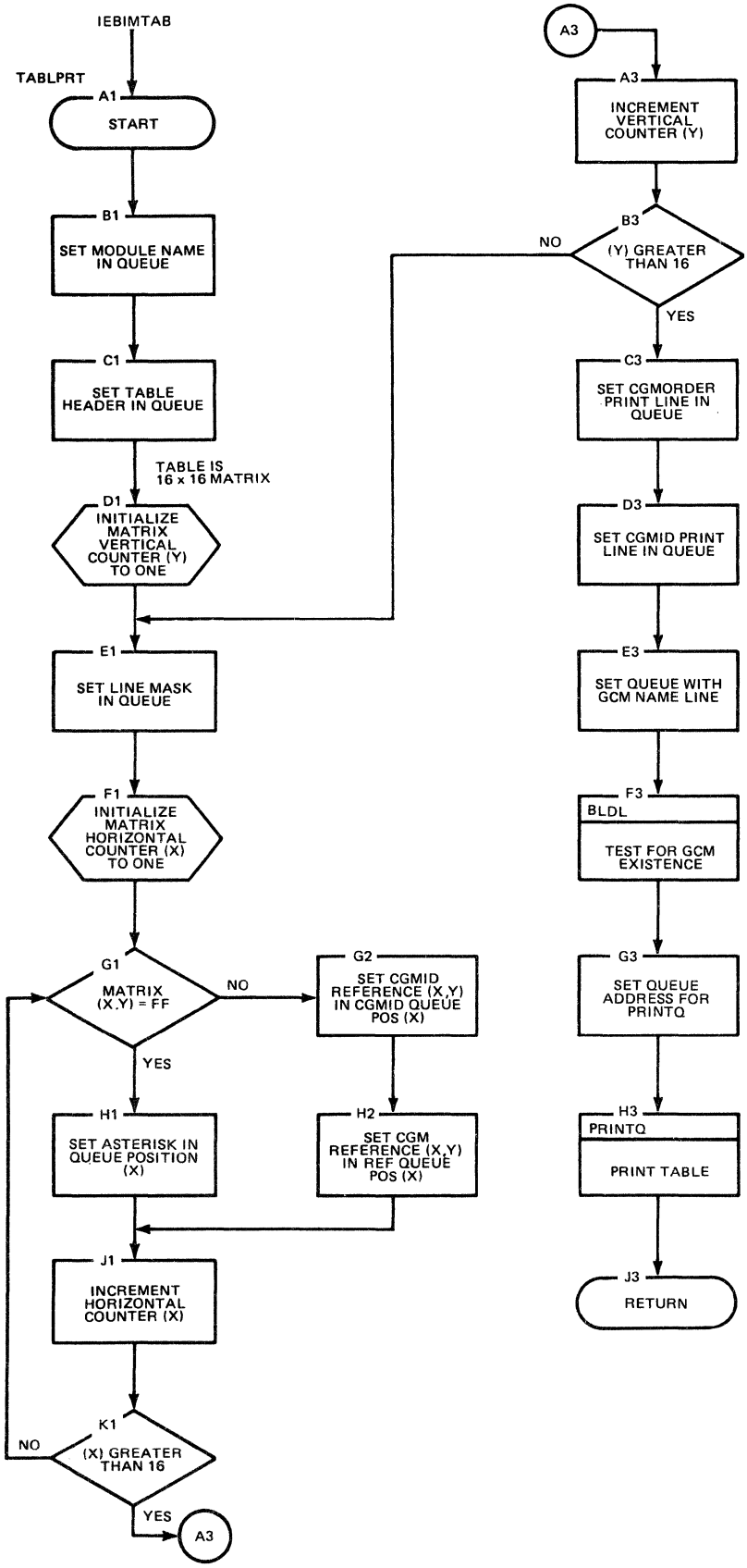
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 28. IEBIMTAB MODULE (PART 2 OF 4): TABCHK



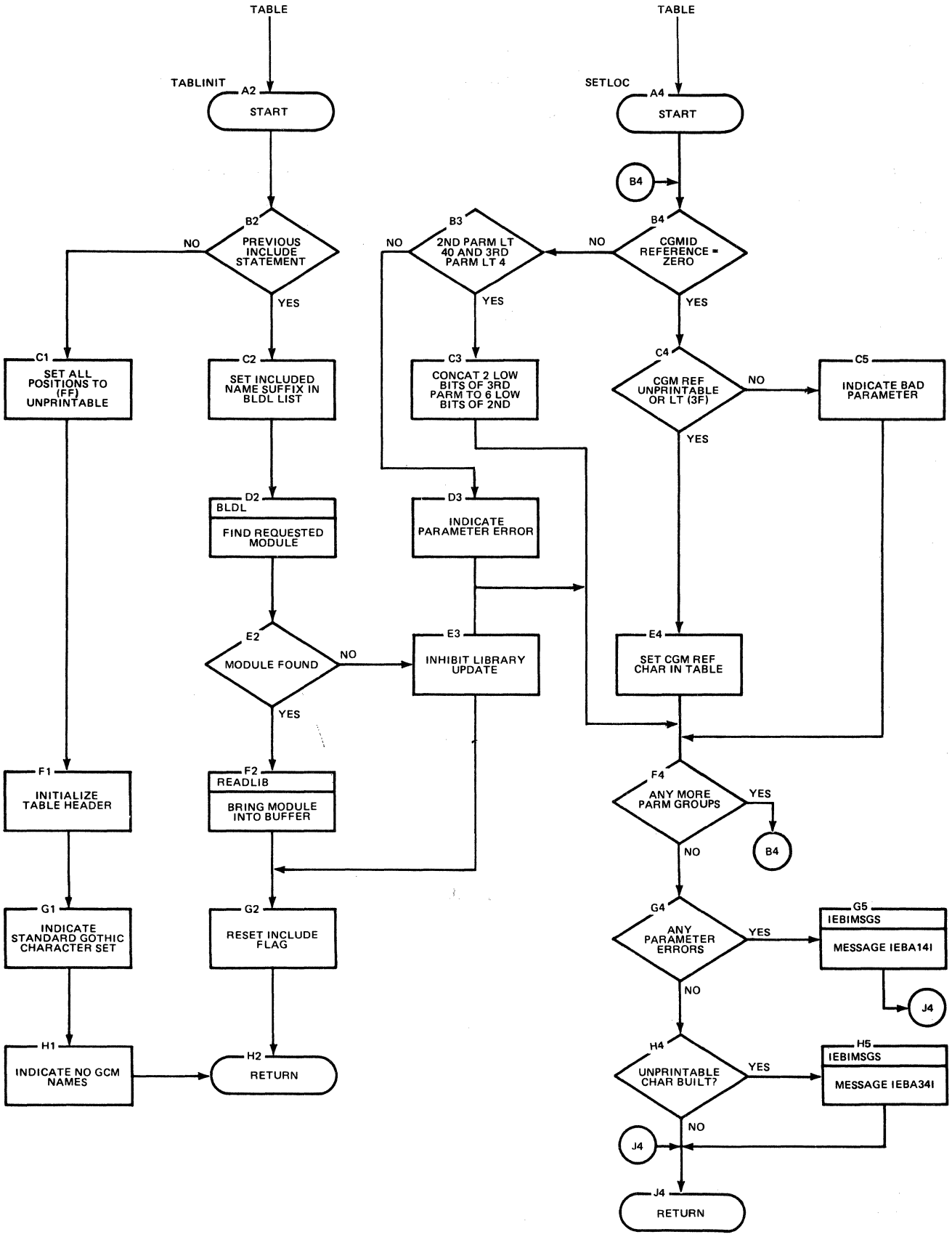
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 29. IEBIMTAB MODULE (PART 3 OF 4): TABLPRT



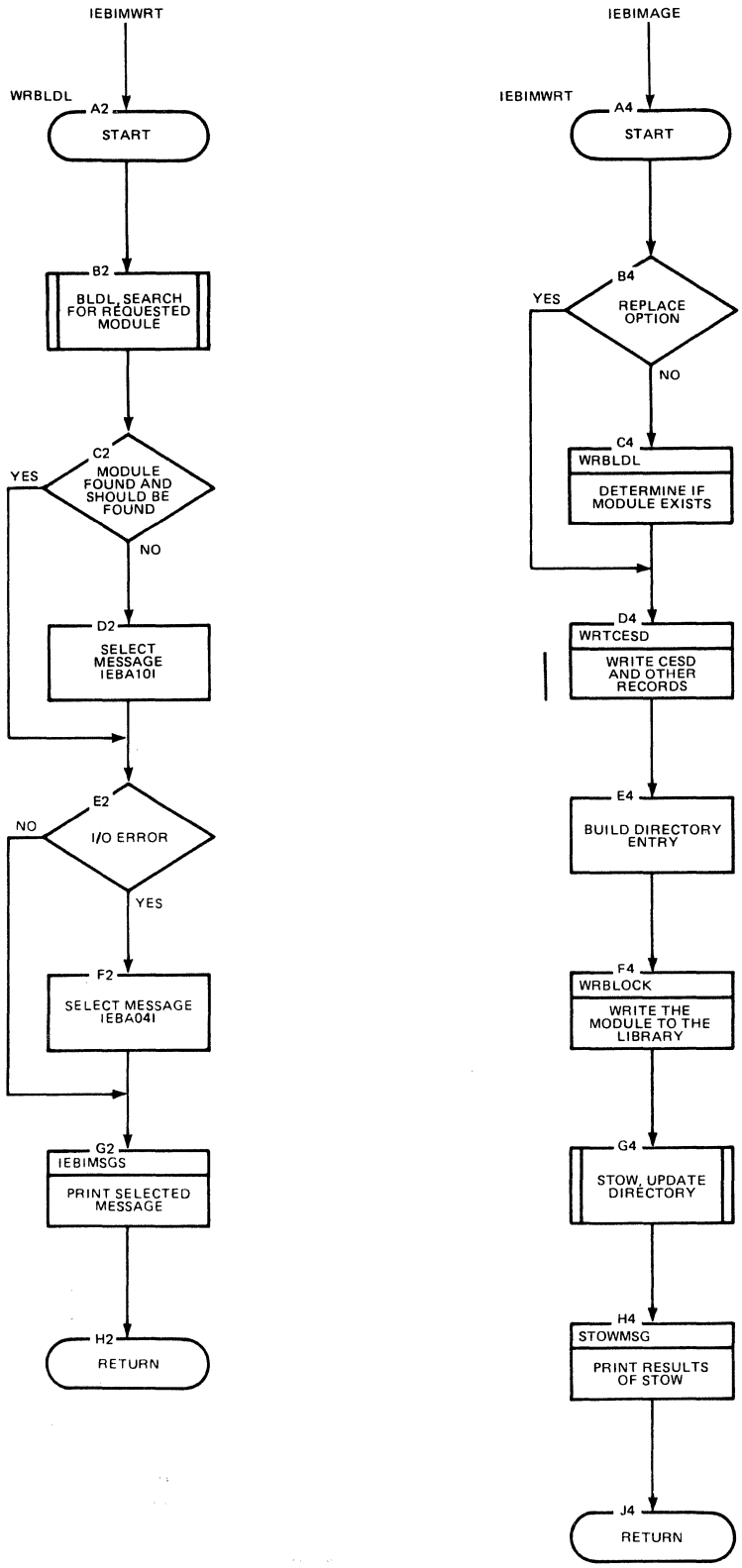
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 30. IEBIMTAB MODULE (PART 4 OF 4): TABLINIT, SETLOC



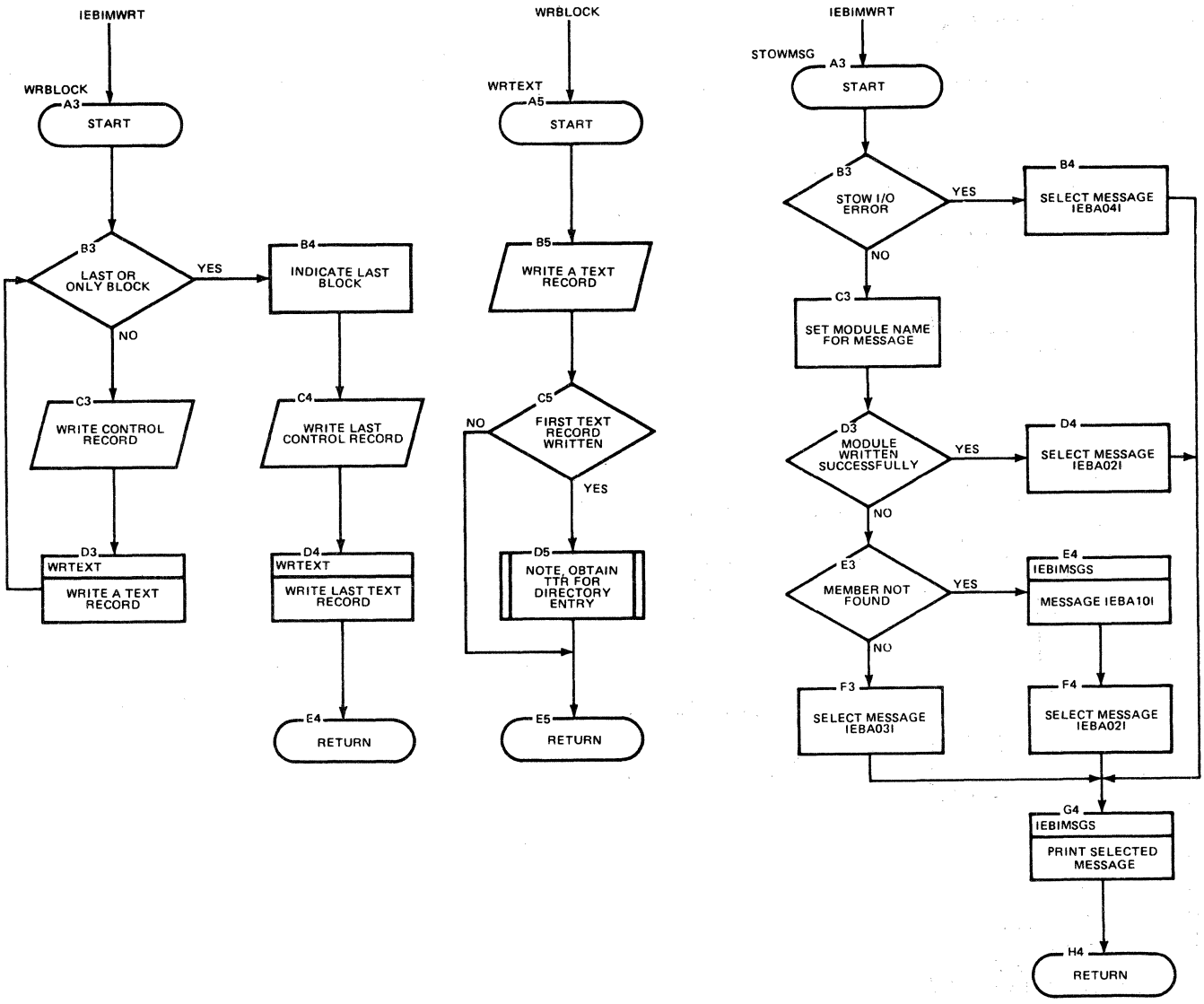
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 31. IEBIMWRT MODULE (PART 1 OF 2): MAIN FLOW, WRBLDL



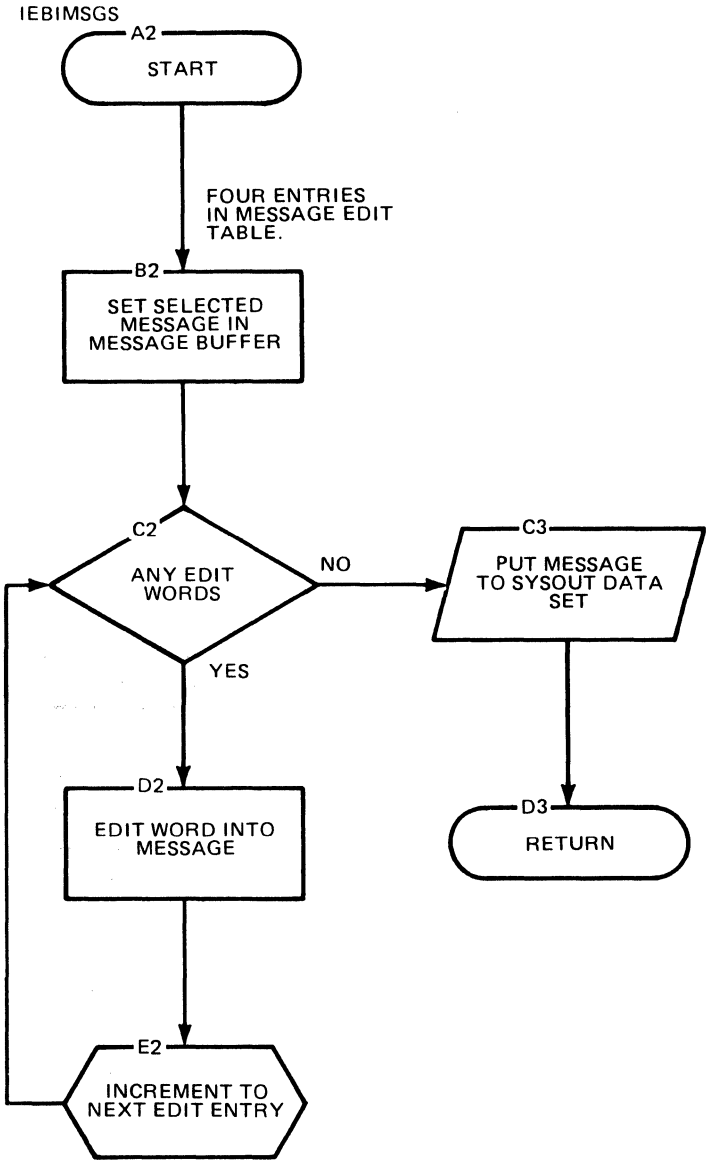
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 32. IEBIMWRT MODULE (PART 2 OF 2): WRBLOCK, WRTEXT, STOWMSG



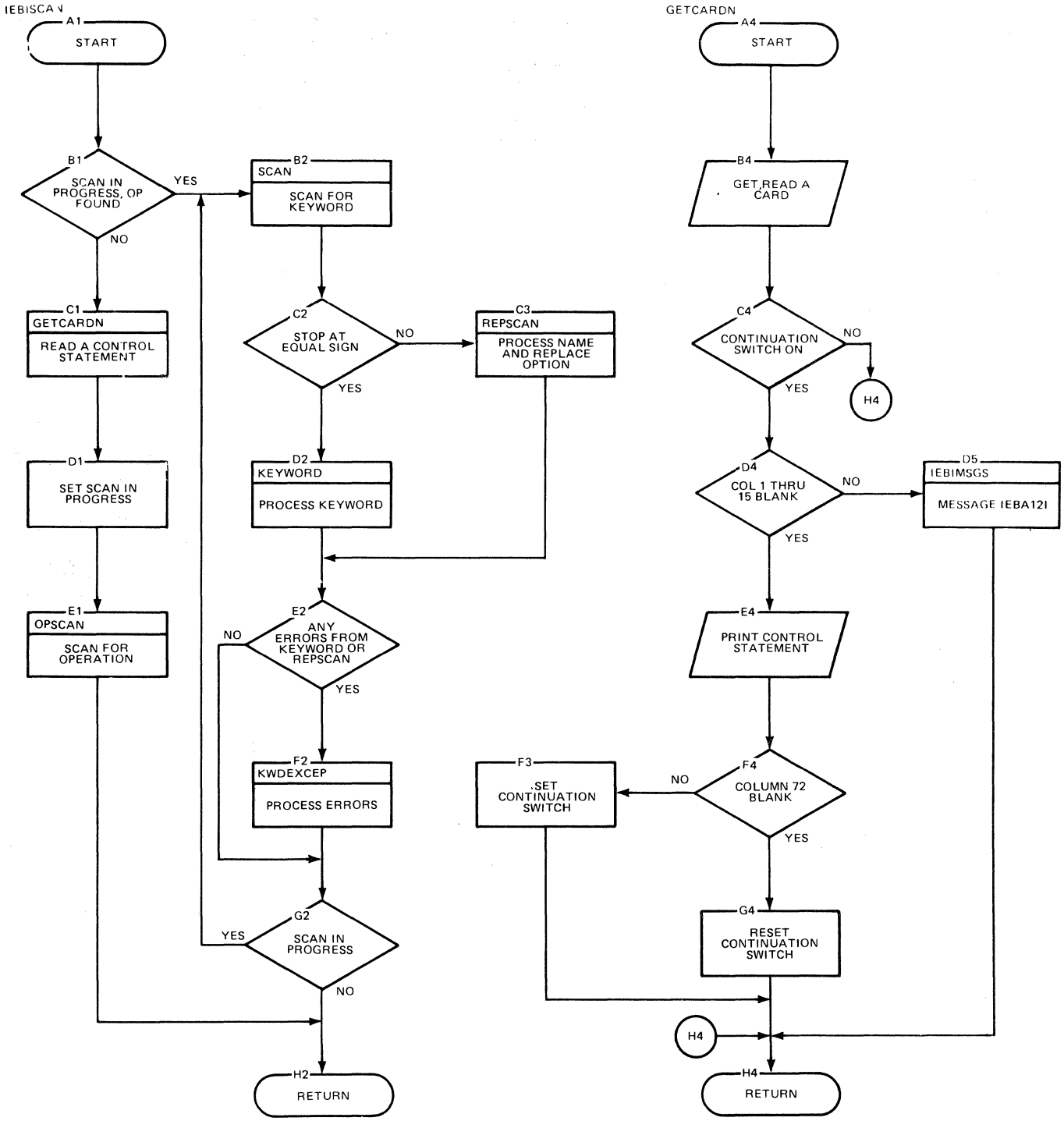
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 33. IEBIMSGS MODULE



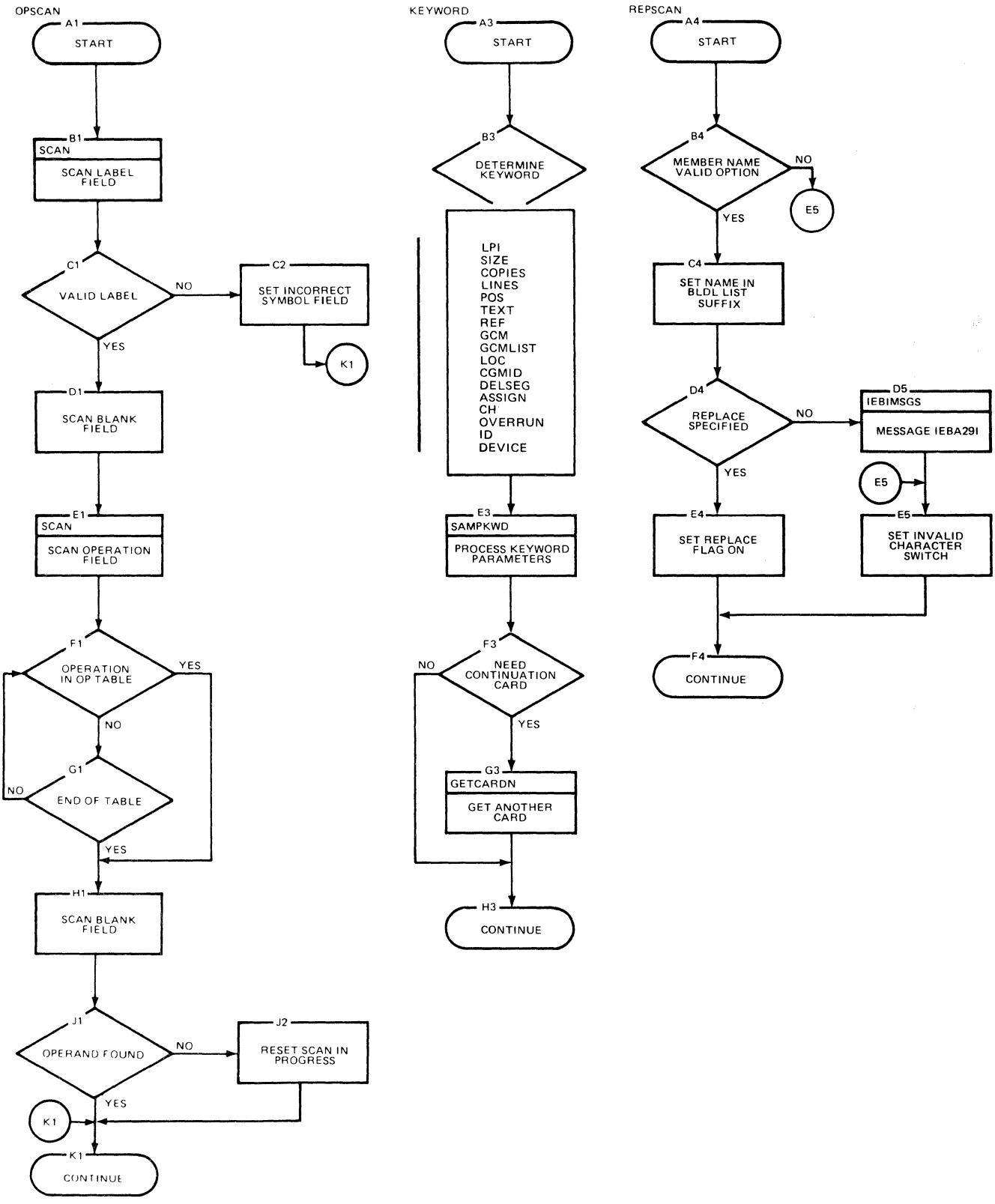
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 34. IEBISCAN MODULE (PART 1 OF 4): MAIN FLOW, GETCARDN



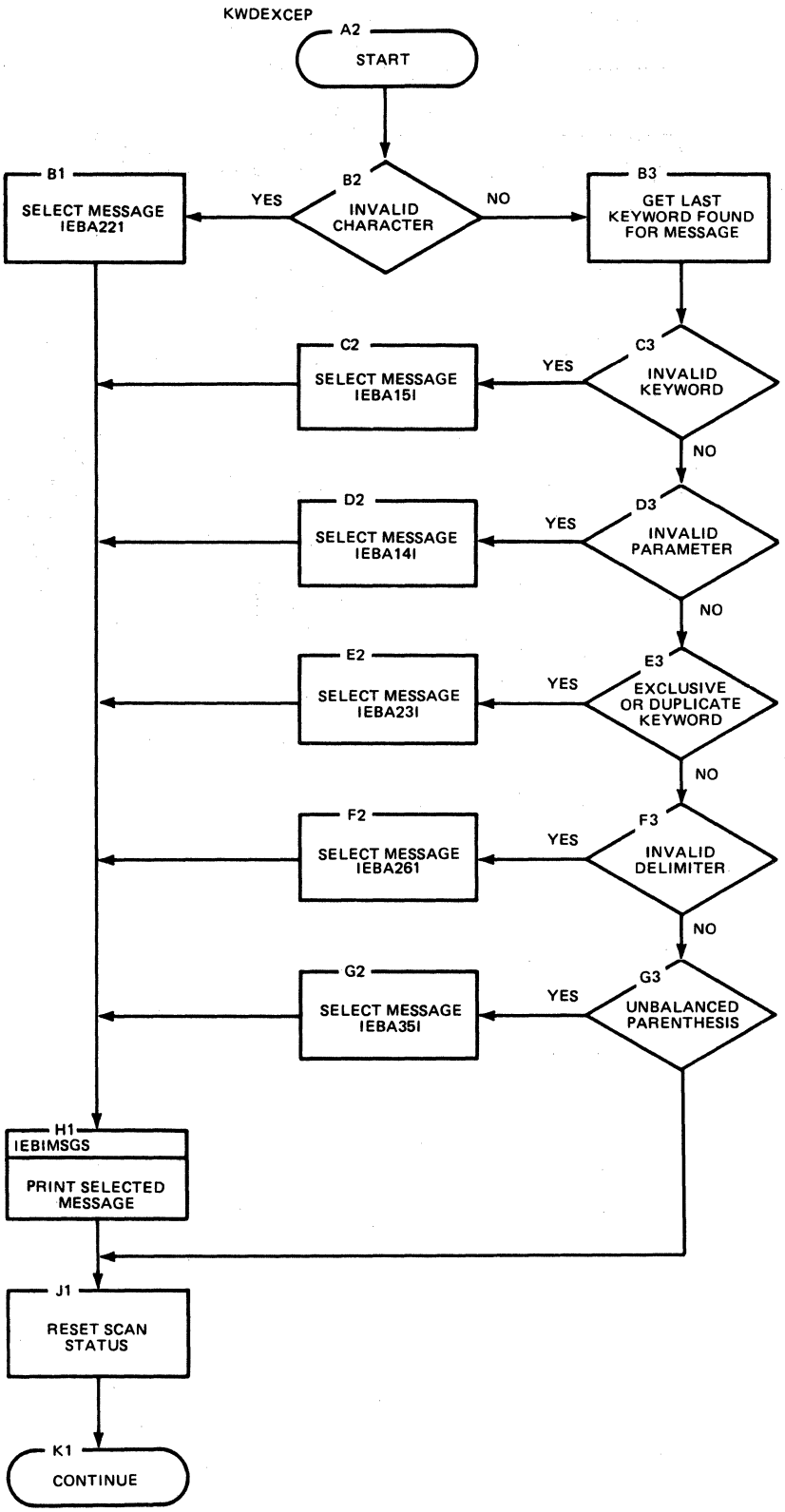
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 35. IEBISCAN MODULE (PART 2 OF 4): OPSCAN, KEYWORD, REPSCAN



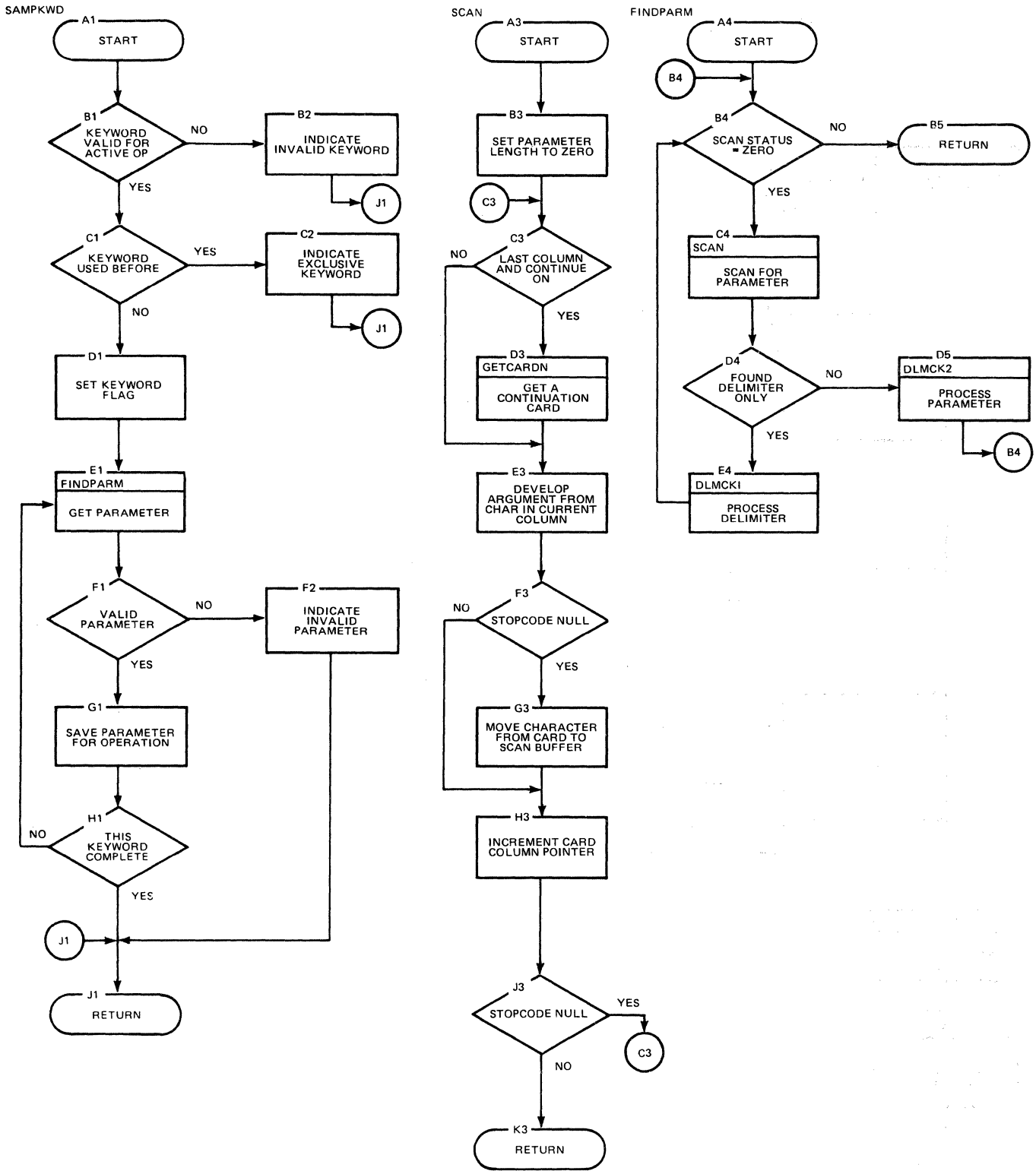
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 36. IEBISCAN MODULE (PART 3 OF 4): KWDEXCEP



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 37. IEBISCAN MODULE (PART 4 OF 4): SAMPKWD, SCAN, FINDPARM



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEBIBLKS—IEBIMAGE WORK AREA

Purpose: Contains work area for the IEBIMAGE program

Built by: IEBIMAGE

Pointed to by: Register 8

Offset	Bytes/Bit Pattern	Name	Description
0 (0)	8	DWORK	Doubleword work area
8 (8)	4	FWORK	Singleword work area
12 (C)	1	STATUS1	Utility status
	1...	ABNDFLG	Abnormal end pending
	.1..	EOFFLG	SYSIN end-of-file pending
	..1.	CARDIN	Card in buffer has not been processed
	...1	INHIBLIB	Inhibit library access
	.... 1...	CONTINUE	Continuation statement to be processed
	.... .1..	NOWRITE	Prevent all WRITEs to library
	.... ..1.	NOTED	NOTE macro issued
	x	—	Reserved
13 (D)	1	STATUS2	Operation status
	1...	REPUSE	Replace option
	.1..	INHIBITW	Prevent library write for this operation
	..1.	INHIBITR	Prevent library read for this operation
	...1	INHIBITP	Prevent printing module for this operation
	.... 1...	FMTERR	Module format error
	.... .1..	FMTERR1	Module format error
	.... ..1.	PRTYERR	GRAPHIC parity error
	1	READMOD	Read only operation
14 (E)	1	STATUS3	Scan status
	1...	INVDLM	Invalid delimiter
	.1..	FOUND	Parameter found
	..1.	INVKWD	Invalid keyword
	...1	EXCLKWD	Exclusive/duplicate keyword
	.... 1...	INVPARM	Invalid parameter
	.... .1..	INVCHAR	Invalid character
	.... ..1.	UNBAL	Unbalanced parenthesis
	1	KWDCOMP	Keyword processing complete
15 (F)	1	STATUS4	Miscellaneous status
	1...	KEYIN	Keyword has not been processed
	.1..	SCANINP	Scan in process
	..1.	INCLSW	INCLUDE statement processed
	...1	HEXERR	Hexadecimal error during convert
	.... 1...	FLDSW	Within parameter group
	.... .1..	PARMGRP	At least one parameter group
	.... ..1.	OPINPRG	Operation in progress
	1	LASTQ	Last quote or graphic data card

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Offset	Bytes/Bit Pattern	Name	Description
16 (10)	2	KWDUSED	Keywords used this operation
	1...	NAMESET	Name found
	.1..	NOKEY	No keywords in this operation
	..1.	DELSET	DELSEG keyword
			For FCB operation:
	...1	LPISET	LPI keyword
	.... 1...	LINESET	LINES keyword
	.... .1..	SIZESET	SIZE keyword
	.... ..1.	CHSET	CHx keyword
	x	—	Reserved
			For TABLE operation:
	...1	LOCSET	LOC keyword
	.... 1...	CGMIDSET	CGMID keyword
	.... .1..	GCMLSET	GCMLIST keyword
			For COPYMOD operation:
	...1	COPYSET	COPIES keyword
	.... 1...	LINESET	LINES keyword
	.... .1..	POSSET	POS keyword
	.... ..1.	TEXTSET	TEXT keyword
	1	ORUNSET	OVERRUN found
			For GRAPHIC operation:
	...1	REFSET	REF keyword
	.... 1...	GCMSET	GCM keyword
	.... .1..	ASGNSET	ASSIGN keyword
			For CHARSET operation:
	...1	REFSET	REF keyword
	.... 1...	GCMSET	GCM keyword
	.... .1..	ASGNSET	ASSIGN keyword
	.... ..1.	IDSET	ID keyword
18 (12)	2	—	Reserved
20 (14)	1	ACTIVOP	Index into operation table for current operation
21 (15)	1	NEXTOP	Index into operation table for next operation
22 (16)	1	ACTIVKEY	Index into keyword table for current keyword
23 (17)	1	OPTABLEN	Operation table length
24 (18)	4	OPTABAD	Operation table address
28 (1C)	16	—	Reserved
			PDS Directory Entry
44 (2C)	8	DNAME	Name of module to write
52 (34)	3	DTTRP	Relative address of first block (CESD records)

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Offset	Bytes/Bit Pattern	Name	Description
55 (37)	1	DINDCT	Number of TTRs and user field length
56 (38)	3	DTTRT	Relative address of text record
59 (3B)	5	—	0
64 (40)	2	DATTRI	Module attributes
66 (42)	3	DSTOR	Main storage required for module
69 (45)	2	DLEN	Length of text record
71 (47)	9	—	0
80 (50)	2	LIBBLKSZ	Library blocking factor
			BLDL Directory Entry
82 (52)	4	BFFLL	Number of entries (1) and length (36)
86 (56)	8	BNAME	Name of module to read/load
94 (5E)	3	BTTRP	Relative address of first block
97 (61)	2	—	Not referenced
99 (63)	1	BINDCT	Number of TTRs and user field length
100 (64)	3	BTTRT	Relative address of text record
103 (67)	5	—	0 (not referenced)
108 (6C)	2	BATTRI	Module attributes
110 (6E)	3	BSTOR	Main storage required for module
113 (71)	2	BLEN	Length of text record
115 (73)	9	—	Not referenced
124 (7C)	1	LIBEXT	Last library extent known
125 (7D)	1	PRECOL	Previous column information for scan
126 (7E)	2	NEXT	Printer queue index
128 (80)	1	RETSAVE	Return code save area
129 (81)	3	CC	Current column pointer
132 (84)	4	CARDEND	Last column pointer
136 (88)	4	SYSPRTAD	Pointer to system output DCB
140 (8C)	4	SYSINAD	Pointer to system input DCB
144 (90)	4	UTINAD	Pointer to library input DCB
148 (94)	4	UTOUTAD	Pointer to library output DCB

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Offset	Bytes/Bit Pattern	Name	Description
152 (98)	16	MSGEDIT	Four 4-byte entries for editing data into messages Byte 0: length of data Byte 1-3: address of data
168 (A8)	2	PRNCNT	Parenthesis count
170 (AA)	2	PAGE	Page count
172 (AC)	1	ARGMT	Scan argument
173 (AD)	3	TODAY	Current date
176 (B0)	1	LOADFLAG	Module loaded flag X'FF'
177 (B1)	3	LOADMOD	Pointer to loaded module
180 (B4)	8	LOADNAME	Name of loaded module
188 (BC)	2	SEGCNT	Segment count
190 (BE)	2	TESTP	Work area for pitch and parity test
192 (C0)	72	SAVE1	Save area level #1
264 (108)	72	SAVE2	Save area level #2
336 (150)	72	SAVE3	Save area level #3
408 (198)	72	SAVE4	Save area level #4
480 (1E0)	72	SAVE5	Save area level #5
552 (228)	80	PRIMCARD	Primary card buffer
632 (278)	80	SDRYCARD	Secondary card buffer
712 (2C8)	1	MSGN	Current message number
713 (2C9)	121	MSG	Message buffer
834 (342)	2	FCBSIZE	FCB size value
836 (344)	1	ASGNMENT	EBCDIC assignment
837 (345)	1	PITCH	Character pitch
838 (346)	2	—	Reserved
840 (348)	1	MARGN	Margin line count
841 (349)	1	*	Switches
842 (34A)	2	MAXFCB	Maximum FCB lines for device
844 (34C)	2	BOTTOM	Bottom margin
846 (34E)	2	LOOP	Loop counter
848 (350)	32	DDNAMES	ddname list
880 (370)	8	LASTKEY	Save area for last keyword
888 (378)	2	PARMSIZE	Length of last scanned string

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Offset	Bytes/Bit Pattern	Name	Description
890 (37A)	1	PARMNUM	Current parameter number
891 (37B)	1	OPTSWS	Option switches
	1...	OPTBIT	At least one option
	.1..	CKN	No overrun message
	..1.	CK6	Check 6 LPI
	...1	CK8	Check 6/8 LPI
	.... 1...	CK10	Check 10 LPI
	.... .1..	BUILD	Build request
	.... ..1.	CHECK	Check request
	1	PRINT	Print request
892 (37C)	12	REBASE	IEBIMAGE base regs saved
904 (388)	8	LPICNT	Lines per inch counters
912 (390)	24	—	Reserved
936 (3A8)	24	SUBRADS	Addresses of subroutines
960 (3C0)	60	*	Various work counters
1020 (3FC)	1	DEVMODL	Model indicator
1021 (3FD)	3	XSWITCHS	24-bit switches
1024 (400)	8	—	Reserved
1032 (408)	1440	SCANWORK	Work area for IEBISCAN
2472 (9A8)	8	MODHEAD	Module header
2480 (980)	8192	MODBOD	Module body
10672 (29B0)	212	OVERFLO	Module overflow area
10884 (2A84)	7260	MSGSAVE	Message buffer queue

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

KEYTAB—IEBIMAGE KEYWORD TABLE (ONE ENTRY)

Purpose: Defines keyword attributes and procedures

Located in: Module IEBISCAN

Offset	Bytes/Bit Pattern	Name	Description
0 (0)	1	KLEN	Number of characters in keyword name
1 (1)	3	PRMPTR	Pointer to keyword routine
4 (4)	1	KFLAGS	Keyword attribute flags
	1...	KMULT	Multiple parameters allowed
	.1..	KPARN1	Single parenthesis allowed
	..1.	KPARN2	Double parenthesis allowed
	...1	VARI	Variable-length keyword
	 xxxx	—	
5 (5)	1	KCNT	Maximum parameters allowed
6 (6)	6	KPARM	Parameter attribute flags (2 bytes per parameter, three possible parameters)
6 (6)	1	KPFLAGS	Byte 0 of parameter attributes
	1...	KDEC	Decimal data
	.1..	KHEX	Hexadecimal data
	..1.	KCHAR	Character data
	...1	KSPEC	Special data
	 xxxx	—	Reserved
7 (7)	1	KPLEN	Maximum length of parameter
12 (C)	8	KEYWD	Keyword name

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

OPTAB—IEBIMAGE OPERATION TABLE (ONE ENTRY)

Purpose: Defines operation attributes and procedures

Resides in: Module IEBIMAGE

Pointed to by: OPTABAD field in IEBIBLKS

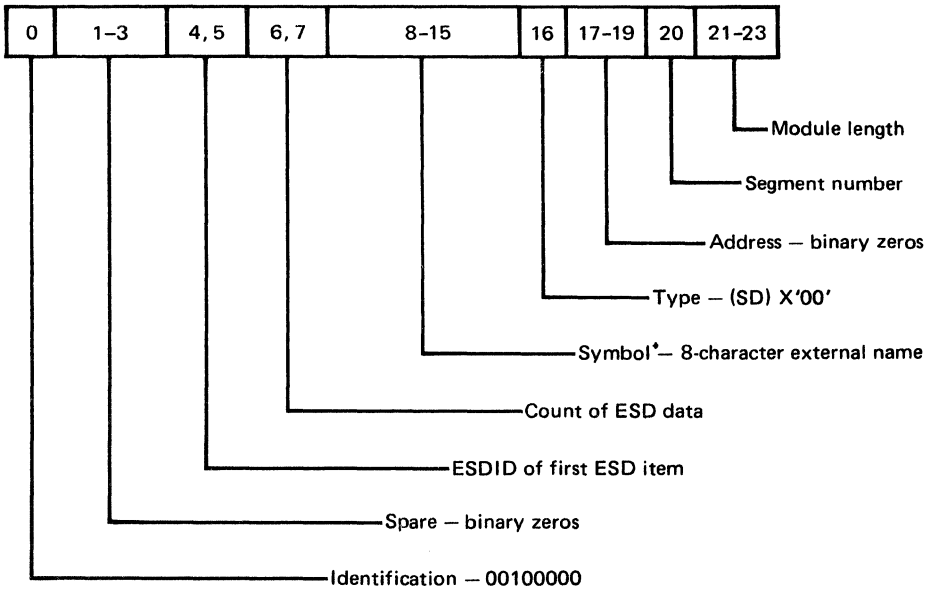
Offset	Bytes/Bit Pattern	Name	Description
0 (0)	1	OPNUM	Operation number
1 (1)	4	OPFLAGS	Operational keyword flags
	byte 0		
	1...	MLTFLAG	Multi control statements allowed
	.1..	LPIFLAG	Allow LPI keyword
	..1.	CHFLAG	Allow CH1–CH12 keyword
	...1	SIZEFLAG	Allow SIZE keyword
	.... 1...	COPIFLAG	Allow COPIES keyword
	.... .1..	LINEFLAG	Allow LINES keyword
	.... ..1.	POSFLAG	Allow POS keyword
	1	TEXTFLAG	Allow TEXT keyword
	byte 1		
	1...	NAMEFLAG	NAME option allowed
	.1..	ASGNFLAG	Allow ASSIGN keyword
	..1.	DELFLAG	Allow DELSEG keyword
	...1	REFFLAG	Allow REF keyword
	.... 1...	GCMFLAG	Allow GCM keyword
	.... .1..	GCMLFLAG	Allow GCMLIST keyword
	.... ..1.	LOCFLAG	Allow LOC keyword
	1	CGMIDFLG	Allow CGMID keyword
	byte 2		
	1...	NOFLAG	Allow no keywords
	.1..	ASTRFLAG	Allow asterisk for NAME
	..1.	REPFLAG	Allow (R) replace for NAME
	...1	ORUNFLAG	Allow OVERRUN option
	.... 1...	IDFLAG	Allow ID keyword
	xxx	—	Reserved
	byte 3		
	1...	DFLTFLAG	Allow DEFAULT keyword
	.xxx xxxx	—	Reserved
5 (5)	3	—	Reserved
8 (8)	1	NOB	Number of characters in operation name
9 (9)	3	BLDPTR	Pointer to build routine
12 (C)	1	OPDEVMSK	Valid devices mask
	1...	3800-1	
	.1..	3800-3	
	..xx xxxx	Reserved	
13 (D)	3	CHKPTR	Pointer to check routine
16 (10)	1	—	Reserved byte
17 (11)	3	PRTPTR	Pointer to print routine
20 (14)	8	RTNAME	Operation name

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

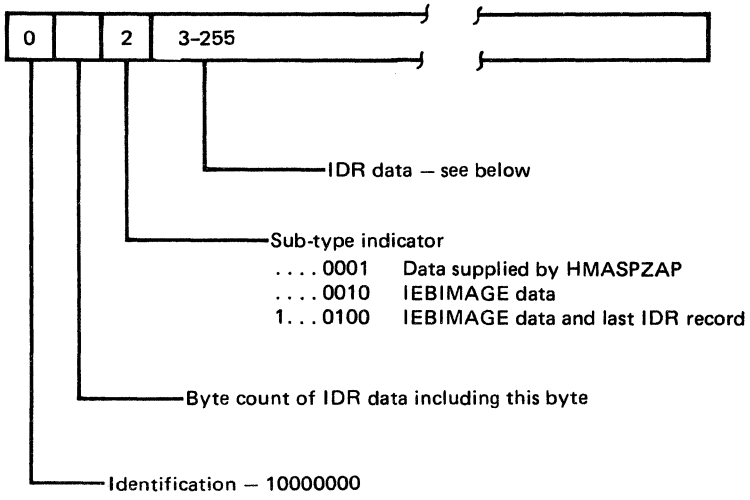
LOAD MODULE RECORDS

IEBIMAGE writes five records at the beginning of each module in the library to maintain compatibility with the linkage editor and other utility programs.

CESD (Composite External Symbol Dictionary)

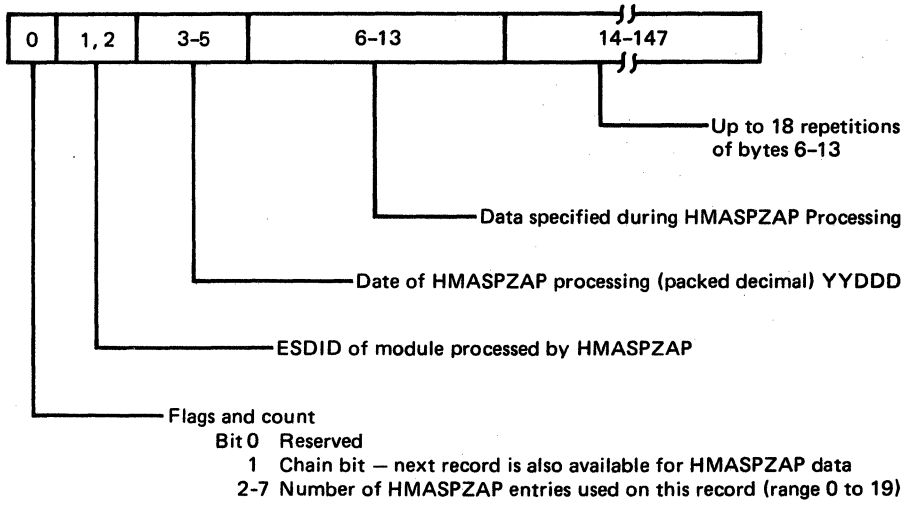


Identification Record

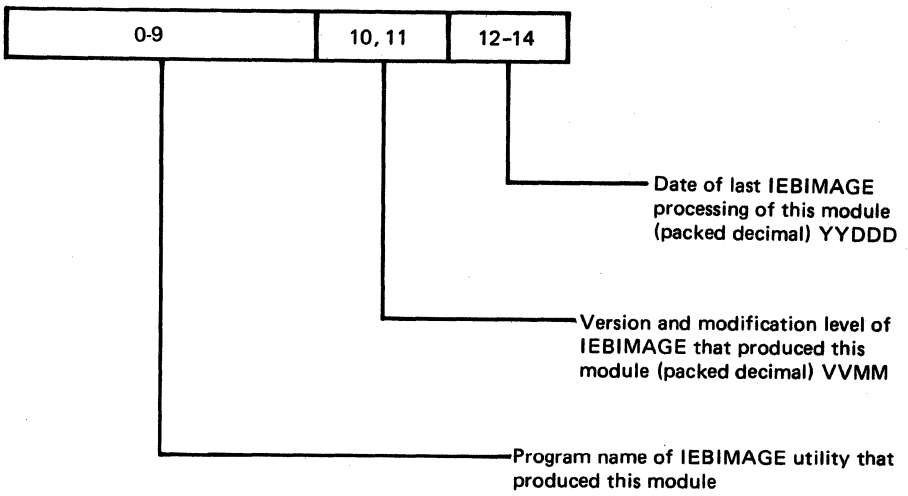


This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

HMASPZAP Data

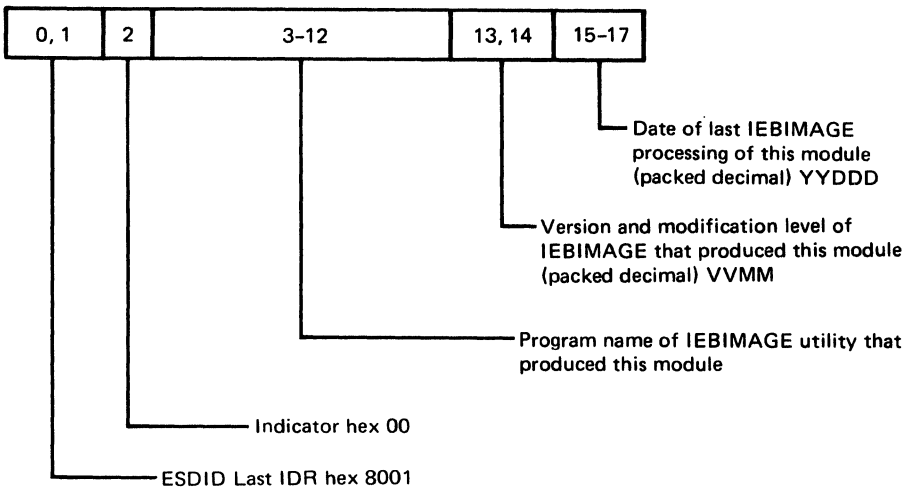


Linkage Editor Data (IEBIMAGE Acts as a Linkage Editor)

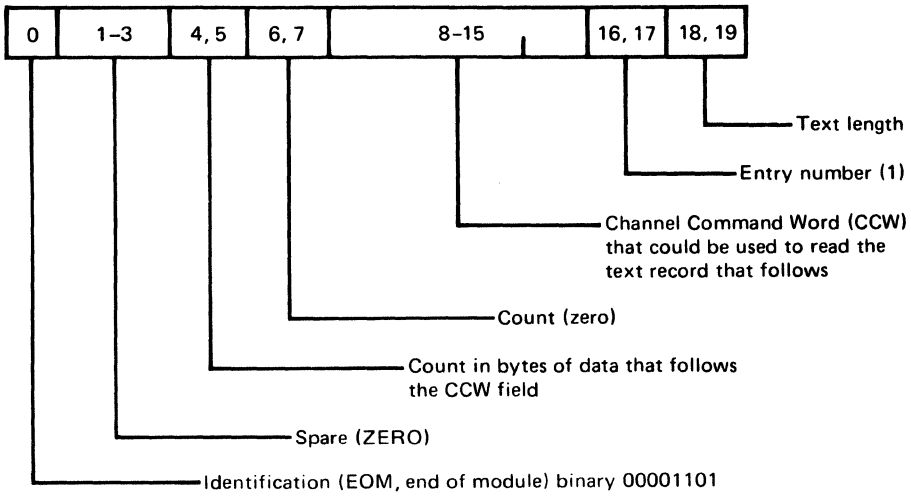


This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Translator Data (IEBIMAGE Acts as a Language Translator)



Control Record



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEBISAM PROGRAM

The IEBISAM program is executed under the operating system to copy, unload, load, or print an indexed sequential data set. As examples, this program can be used to create a backup copy of a data set, or to improve the accessibility of a data set by eliminating wasted track space and overflow areas. The program, which may be either executed as a job step or called by an executing program, consists of six load modules (see "Chart 01. Overall Flow" on page 303) that reside in the linkage library, SYS1.LINKLIB.

The initializing routine determines which function has been specified, and then passes control to one of four functional (or processing) modules. The selected processing module performs its specified function, and then passes control to the terminating routine, which writes messages, terminates processing, and returns control to the calling routine.

Note: If invalid specifications or parameters have been specified, the initializing routine sets the appropriate message and completion code indications and gives control directly to the terminating routine.

The IEBISAM program may be executed as a job step, or it may be called by a program executing a job step. If it is to be executed as a job step, the step's EXEC statement specifies the program IEBISAM, and the EXEC statement's PARM field specifies the function to be performed as a parameter (COPY, UNLOAD, LOAD, or PRINTL). If the IEBISAM program is to be called by a program executing a job step, the calling program must specify the function by providing "EXEC" statement parameters and ddnames as shown in "Auxiliary Parameters," earlier. In either case, the job control language statements that describe the step during which IEBISAM is to be executed must include DD statements to define the input, output, and message data sets.

Figure 105 on page 297 gives a module directory and summary for the IEBISAM program. The figures, from "Chart 02. Initialize IEBISAM Program" on page 304 through "Chart 09. Terminate IEBISAM Program (IEBISF)" on page 311, outline the individual routines of the program. For more information regarding the use of the program, refer to Utilities.

INITIALIZING IEBISAM

The initializing routine is in the load module (IEBISAM) that is entered whenever the IEBISAM program is requested. This routine (Chart 02) obtains storage for a work area, and then inspects the specifications under which the program is to run.

If no options have been specified in the PARM field of the EXEC statement, the program assumes the default option to unload the data set. If a function specification is not valid, this routine stores a completion code, assembles a message, and uses the XCTL macro instruction to pass control to the terminating routine, IEBISF.

In all other situations (that is, those in which correct procedures have been followed), the initializing routine assembles the necessary information and gives control to the appropriate module.

COPYING AN INDEXED SEQUENTIAL DATA SET

If the copy function was specified, control is passed to the copy routine in module IEBISC. This routine creates an output data set containing the same records as the input data set, but

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Module ID	CSECT	Summary	Chart ID
IEBISAM	IEBISAM	Receives control from calling routine. Gets work area used by processing modules. Gets program parameters, alternate ddnames, page number. Passes control to appropriate module.	02
IEBISC (Copy)	IEBISC	Produces copy of input data set with new indexes. Uses PUT and GET macro instructions.	03
IEBISU (Unload)	IEBISU	Retrieves an indexed sequential record and passes its length and address to IBISSO. Analyzes return code from IEBISSO and sets success or error indication for IEBISF.	04
	IEBISSO	Unloads the indexed sequential record(s) into physically sequential 80-byte card images.	05
IEBISL (Load)	IEBISL	Reconstructs an indexed sequential record from "unloaded" data passed by IEBISSI. Checks DCB parameters OPTCD, RECFM, LRECL, BLKSIZE, RKP, NTM, KEYLEN, and CYLOFL against corresponding DD statement information.	06
	IEBISSI	Retrieves unloaded (80-byte card images) records. Maintains pointer to current input area. Maintains number of bytes remaining to be processed on a given card image. Checks each card image for proper sequence.	07
IEBISPL (Print)	IEBISPL	Produces printed copy of input data set. Provides for user exit and/or suppression of data conversion.	08
IEBISF	IEBISF	Receives control from initializing or processing module. Prints appropriate message and returns completion code to calling routine.	

Figure 105. Module Directory, Summary, and Chart IDs for IEBISAM Program

with newly built indexes and empty overflow areas. The copy routine (Chart 03) opens the input data set (SYSUT1) and the output data set (SYSUT2) for use by QISAM and checks the DCB parameters:

- The DCBLRECL parameters must be the same for both the input and the output data sets.
- The DCBRECFM parameters must be the same (F or V) for both the input and the output data sets.
- For the output data set, the DCBBLKSI parameter must be a multiple of the DCBLRECL parameter if the record format (RECFM) is fixed (F). For variable-length records (RECFM=V), the DCBBLKSI parameter must be equal to or greater than the logical record length (LRECL) plus 4.
- The DCBRKP parameter must be smaller than the DCBRECL parameter minus the DCBKEYLE parameters.

If the input and output data sets are opened successfully and the DCB parameters are valid, the copy routine determines what method of processing is to be used. For fixed-length records, the copy routine uses the PUT (locate mode) and the GET (move mode) macro instructions to read the logical records in sequence from the input data set and write them onto the output data set. For variable-length records, the copy routine changes MACRF so that a GET LOCATE is done, followed by a PUT MOVE. Now the real length of the next record is found in the ll bb field before moving is done.

If the data sets are not opened successfully, if the DCB parameters are not valid, or if an unrecoverable input/output error is encountered, the routine stores both a completion code and a message code. Processing on the data set is terminated, the data sets are closed, and control is given to the terminating routine.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

UNLOADING AN INDEXED SEQUENTIAL DATA SET

If the unload function was specified, control is passed from the initializing routine to the unload routine (in module IEBISU to create a physical sequential data set containing the information from the input (indexed sequential) data set. This information is put in 80-byte card images on either a magnetic tape volume or a direct access volume. Figure 106 on page 299 illustrates the data flow and format during unloading operations.

Module IEBISU contains two control sections (CSECTs): IEBISU (Chart 04), which reads records in logical sequence from an indexed sequential data set; and IEBISSO (Chart 05), which reblocks the records and writes them into a physical sequential data set.

OBTAINING INDEXED SEQUENTIAL RECORDS

After module IEBISU is entered at CSECT IEBISU, the unload routine opens the input data set and determines the format of the records. (If the relative key position is the high-order byte of the record (that is, DKBRKP equals zero), the key field is treated separately when the records are put into the output data set.)

CSECT IEBISU uses CSECT IEBISSO as a subroutine; initially, IEBISU passes control to IEBISSO to open the output (unloaded) data set, and then again to write the input (indexed sequential) DCB into the output data set. After the DCB has been written, IEBISU uses the GET (locate mode) macro instruction to obtain a record from the input data set, and then passes control to CSECT IEBISSO.

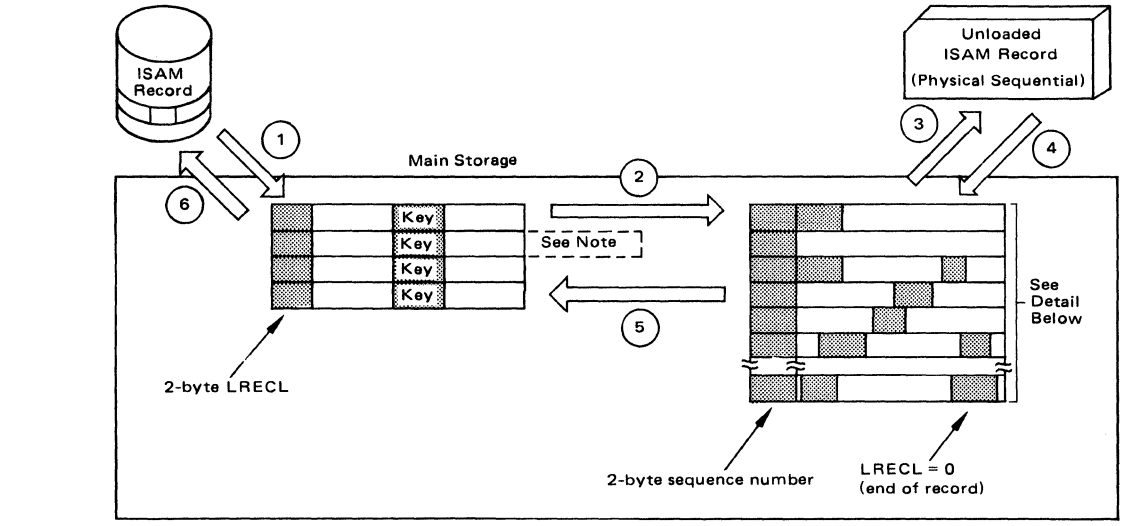
BUILDING THE OUTPUT DATA SET

CSECT IEBISSO performs the reblocking and writing of the input indexed sequential records into the output data set. The output data set is a physical sequential data set consisting of 80-byte logical records. The 80-byte records contain the key and data fields of the indexed sequential data set, together with length indicators and sequence numbers (see Figure 106 on page 299).

The first 154 bytes of DCB information for the indexed sequential data set are written in the first two physically sequential records (those with sequence numbers zero and one) of the output data set. The first 80-byte logical record contains the physical sequence number zero, followed by the length indicator 154. (The length indicator represents the number of bytes between one length indicator field and the following length indicator field.) The first 76 bytes of the DCB for the input data set complete the first logical record. The second 80-byte logical record contains the sequence number one, followed by the next 78 bytes of the input data set's DCB. The information in the first 154 bytes of the input DCB includes the following fields: OPTCD, RECFM, LRECL, BLKSIZE, RKP, NTM, KEYLEN, and CYLOFL. (See "Data Control Block—ISAM" in the Data Areas microfiche.) The remaining 80-byte logical records (beginning with sequence number two) contain the images of the records in the input data set. The last 80-byte logical record of the unloaded (physical sequential) data set contains from 0 to 2 bytes of zeros following the last byte of input record data.

At the first entry to CSECT IEBISSO, the unload routine opens the output DCB and checks the DSORG and BLKSIZE parameters: the DSORG parameter must be PS, and the BLKSIZE parameter must be multiple of 80. If the opening is successful and the parameters are valid, the routine issues a PUT (locate mode) macro instruction to write the DCB in the output data set. This information, along with control of the unload routine, is then returned to CSECT IEBISU.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1



1. Using successive GETs, an indexed sequential record is read. Each logical record is prefixed with a 2-byte LRECL (Logical Record Length)
2. The prefixed ISAM logical records are reblocked into 78-byte logical record images, each of which is prefixed with a 2-byte physical sequence number
3. The 80-byte unloaded records are written using successive PUTs
4. During a subsequent LOAD execution, unloaded records are read (using GET) in sequence until an ISAM record can be rebuilt
5. The sequence numbers are stripped and the record is rebuilt
6. The record is written using successive PUTs

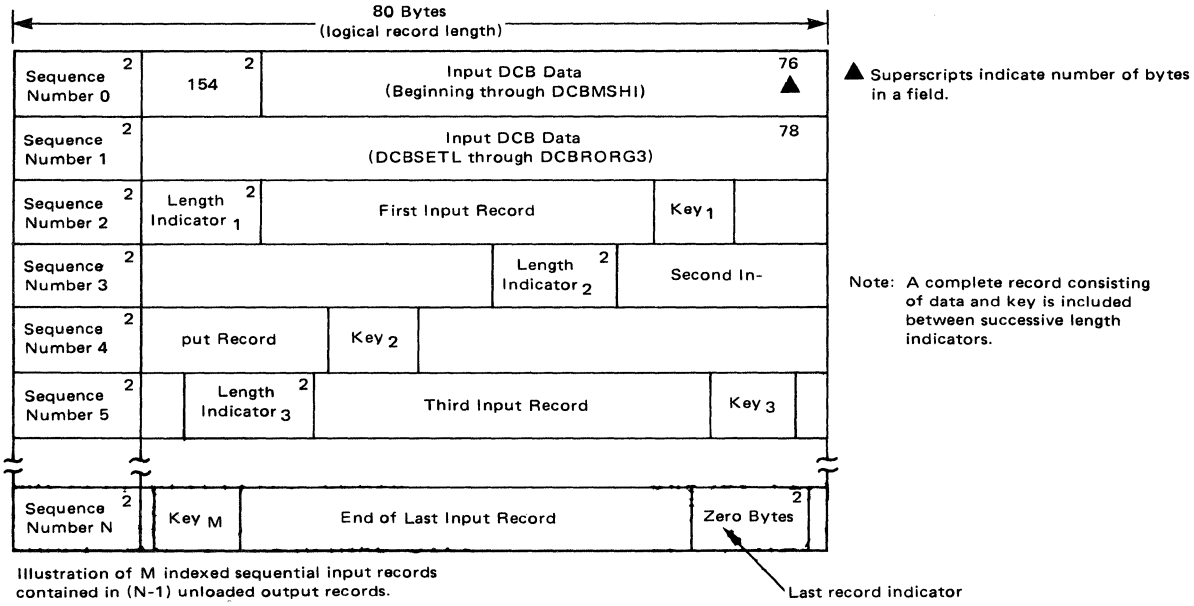


Figure 106. Unloading and Loading an Indexed Sequential Data Set

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

On subsequent entries to CSECT IEBISSO, the output buffer is filled with indexed sequential records obtained by CSECT IEBISU. The routine stores the record length indicator first, and then stores the record key and data fields. When the routine finds the end of an input record, it returns control to CSECT IEBISU to obtain another record; when it has filled the input buffer, the routine issues a PUT (locate mode) macro instruction to write the contents of the buffer onto the output data set. The physical sequence number for the output data set records is then updated.

If CSECT IEBISSO encounters an error condition (for example, unsuccessful open, invalid DCB parameters, or an uncorrectable I/O error), it closes the output data set, sets the appropriate return code (see Chart 04), and returns control to CSECT IEBISU. IEBISU then sets both a message and a completion code, closes the input data set, and passes control to the terminating routine.

LOADING AN INDEXED SEQUENTIAL DATA SET

The load routine is used to reconstruct an indexed sequential data set from an unloaded copy of the physical sequential data set. The output data set resulting from the loading function is placed on a direct access volume. If the original indexed sequential data set contained records in an overflow area, these records will appear sequentially arranged with the records from the original primary area when the unloaded data set is reloaded.

To perform the load function, the initializing routine gives control to CSECT IEBISL of module IEBISL (see Chart 06). The load routine performs its own initializing functions, and then branches to CSECT IEHISSI of the same load module to get the length and address of an input record from the unloaded data set. If the return to CSECT IEBISL from CSECT IEHISSI indicates a return code other than zero, the appropriate message number and/or completion code is established, the output data set (if it was opened as described later) is closed, and control is given to the terminating routine.

If CSECT IEHISSI returns the requested information and a return code of zero when it gives control back to CSECT IEBISL, CSECT IEBISL opens the output data set and checks the validity of the DCB fields. An inconsistency (or error) detected during either of the latter operations leads to procedures for closing the data set as previously described. Otherwise, if no error is detected, control is given to IEBISSI, which moves the length; a PUT LOCATE is issued.

When all records from the unloaded (old) data set have been transferred to the new data set, the old data set is closed and control is given to the terminating routine.

In reconstructing the new data set, the information in the first two logical records of the unloaded data set is used in establishing the DCB for the new data set. The last 78 bytes of each subsequent 80-byte logical record are used to build the records of the new data set.

CSECT IEBISSI (Chart 07) opens the input (unloaded) data set and checks for the validity of the DCB parameters for that data set. Should either the opening be unsuccessful or a DCB parameter be invalid, the data set is closed and return is made to CSECT IEBISL. Otherwise, CSECT IEBISSI gets information from the logical records of the unloaded data set and to transmit it to CSECT IEBISL so that it may be placed in the new indexed sequential data set. The GET and PUT macro instructions are used for these operations. The preceding procedures continue until either the end of the input data set or a terminating error condition is reached. For both situations, the input data set is then closed, and control is returned to CSECT IEBISL.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

PRINTING LOGICAL RECORDS OF AN INDEXED SEQUENTIAL DATA SET

In order to obtain a printed copy of an indexed sequential data set, a user specifies the keyword PRINTL in the PARM field of an EXEC statement. The queued indexed sequential access method (QISAM) is used to obtain the records from the input data set. The records are selected in logical sequence from both the prime and the overflow areas of the input data set. To write the records, the queued sequential access method (QSAM) uses a PUT macro instruction. Record conversion (to hexadecimal notation) and/or user exits before record printing may be specified as options.

After module IEBISAM gives control to the print module IEBISPL (Chart 08), both the input and the output data sets are opened, the success of the openings is determined, and the DCB parameters are checked for validity. If an error is encountered in any of the preceding operations, steps are taken to close the data sets and give control to the terminating routine.

If the data sets have been opened successfully and the DCB parameters are valid, the print routine proceeds to place a record in a buffer area prior to printing it. At this point, a user's routine may gain access to the record if the proper specification has been given on the EXEC statement. Upon return from the user's routine with a return code of either 0 or 4 (see the return code table on Chart 08), or, if no user exit was taken, the data in the buffer is converted to hexadecimal notation unless the no-conversion option has been specified. The PUT macro instruction is then issued to print the record on a SYSOUT device. After all input data records have been printed, or if the routine encounters an unrecoverable error, the input and output data sets are closed and the terminating routine is given control.

Note: A more complete interpretation of the codes returned to the print module IEBISPL by a user's exit routine is:

Code 0: The record currently in the buffer is to be printed, and processing of the input data set is to continue.

Code 4: The record currently in the buffer is to be printed, but processing of the input data set is to be terminated after the printing.

Code 8: The record currently in the buffer is not to be printed. Processing of the input data set is to continue.

Code 12: The record currently in the buffer is not to be printed, and processing of the input data set is to be terminated.

TERMINATING THE IEBISAM PROGRAM

Each of the other routines of the IEBISAM program may give control and a completion code to the terminating routine (in module IEBISF). The basic function of the terminating routine is to write an appropriate message on the SYSPRINT data set. This message indicates the result of the use of the IEBISAM program.

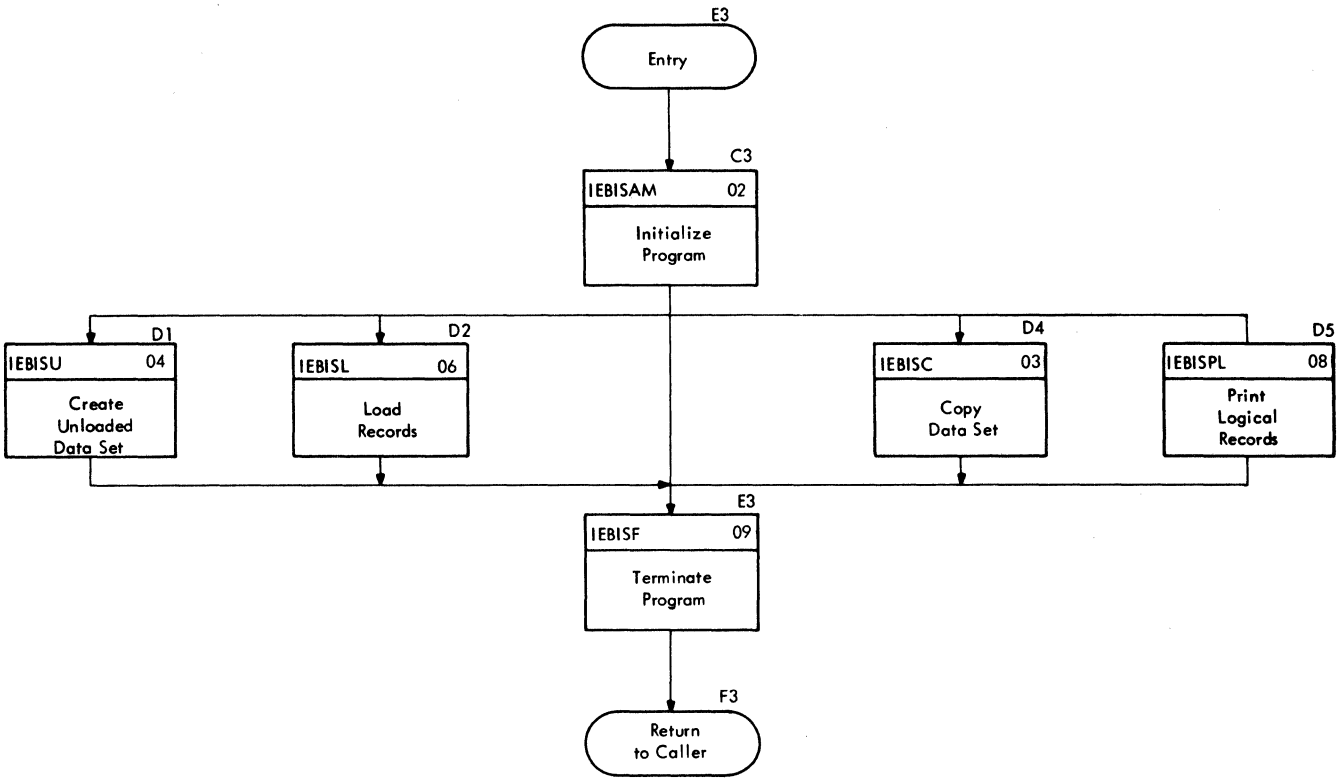
When module IEBISF (Chart 09) gains control, it opens the output (SYSPRINT) data set. If the opening is unsuccessful, the appropriate completion code (16) is set, the SYSPRINT data set is closed, and control is returned to the source from which the IEBISAM program was initially given control.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

After a successful opening of the output data set, the PUT macro instruction is used to write the message concerning the program's result. If an error is encountered during writing, a completion code of 8 is set and returned to the caller of the IEBISAM program. (The completion codes shown on Chart 09 are those resulting from processing activity by module IEBISF.) If no error was encountered during writing, the terminating routine establishes a completion code based upon the results of the routine from which the terminating routine received control. This code, and program control, are then given to the caller of the IEBISAM program.

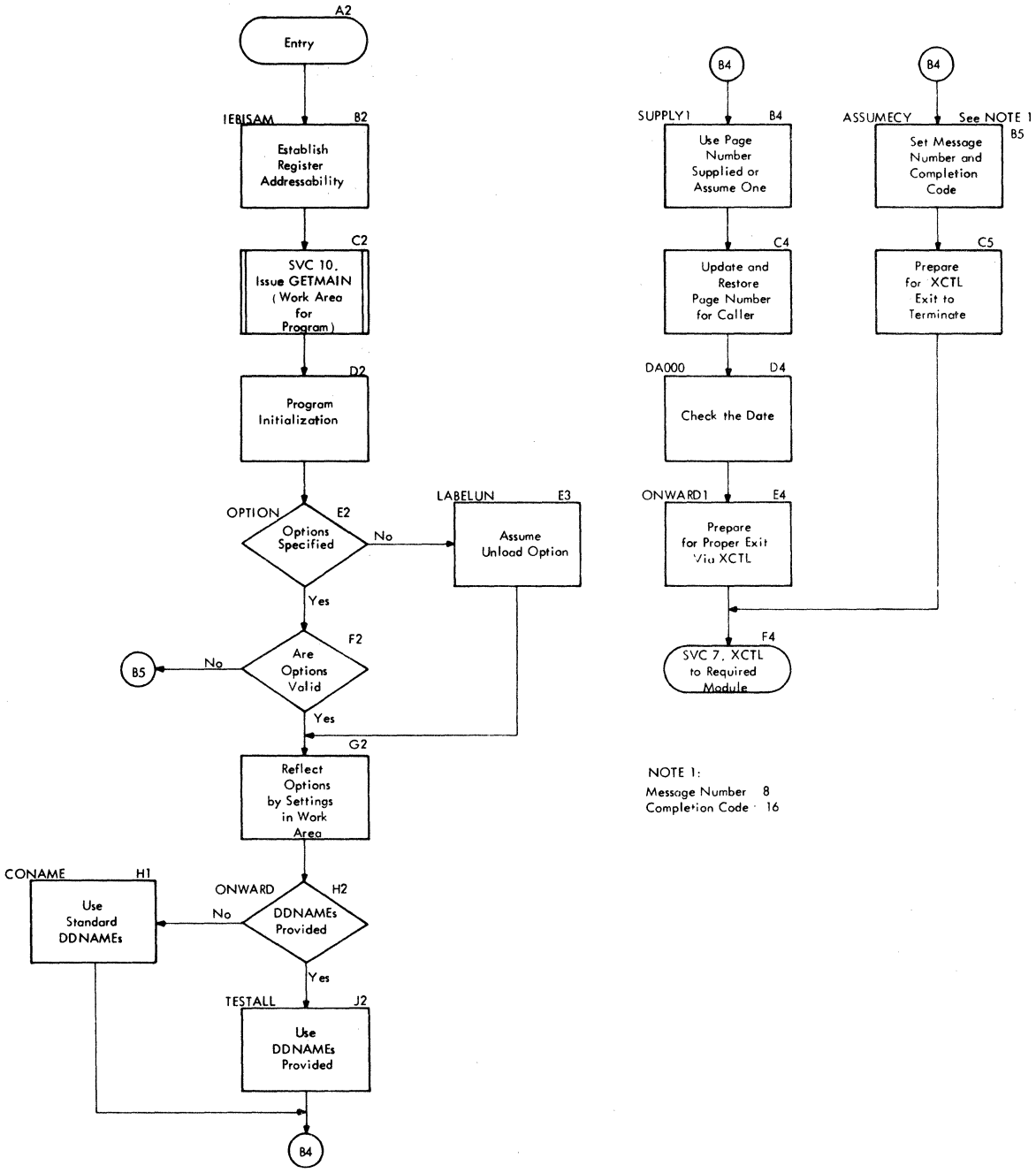
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 01. OVERALL FLOW



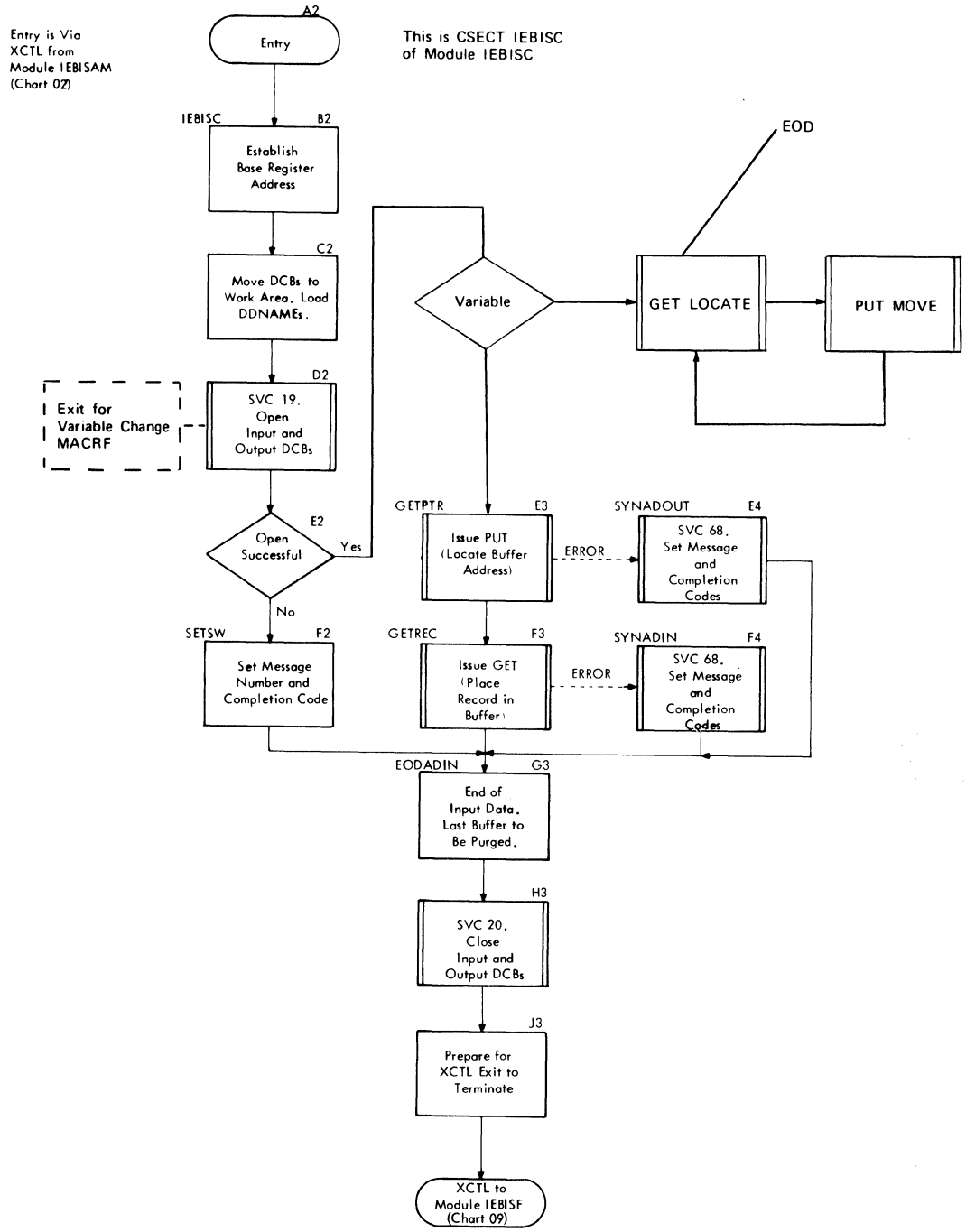
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 02. INITIALIZE IEBISAM PROGRAM



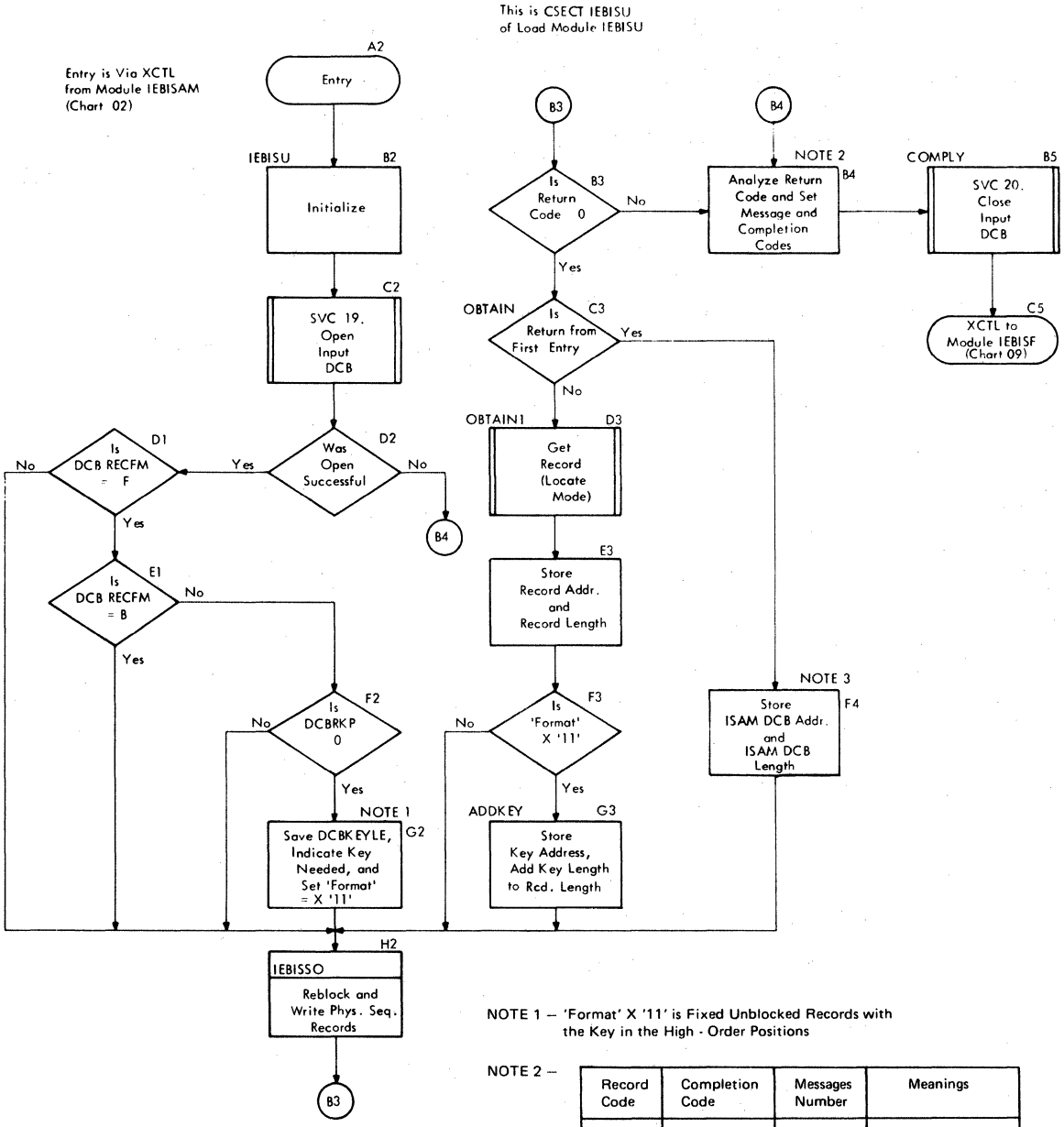
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 03. COPY INDEXED SEQUENTIAL RECORDS (IEBISC)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 04. RETRIEVE INDEXED SEQUENTIAL RECORDS (IEBISU)



NOTE 1 – 'Format' X '11' is Fixed Unblocked Records with the Key in the High - Order Positions

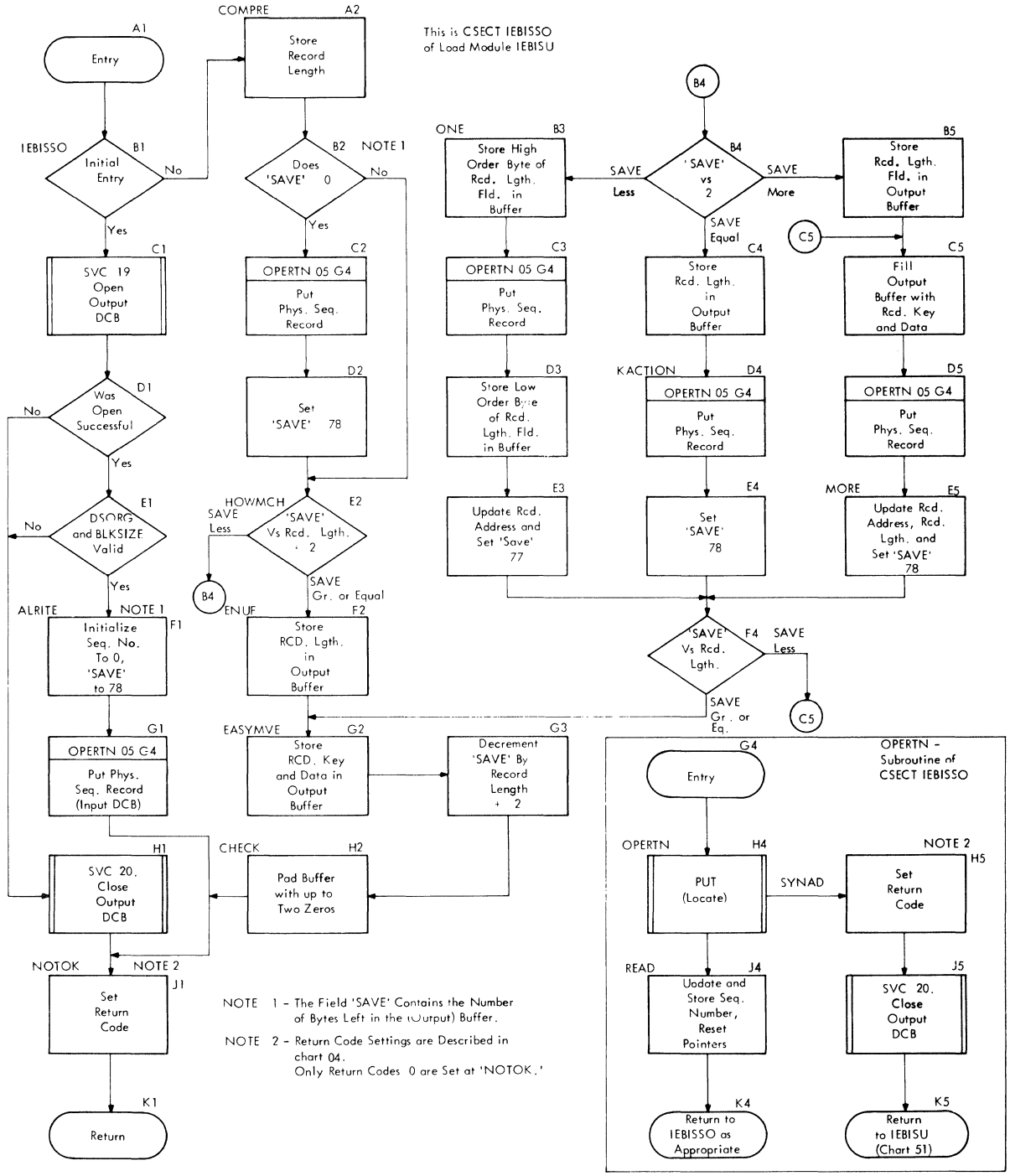
NOTE 2 –

Record Code	Completion Code	Messages Number	Meanings
4	8	2	I/O Error
8	16	7	Bad Open
12	8	1	Bad DCB Params.

NOTE 3 – The First Two (Phys. Seq.) Records Written Contain the DCB for the Input (ISAM) Data Set

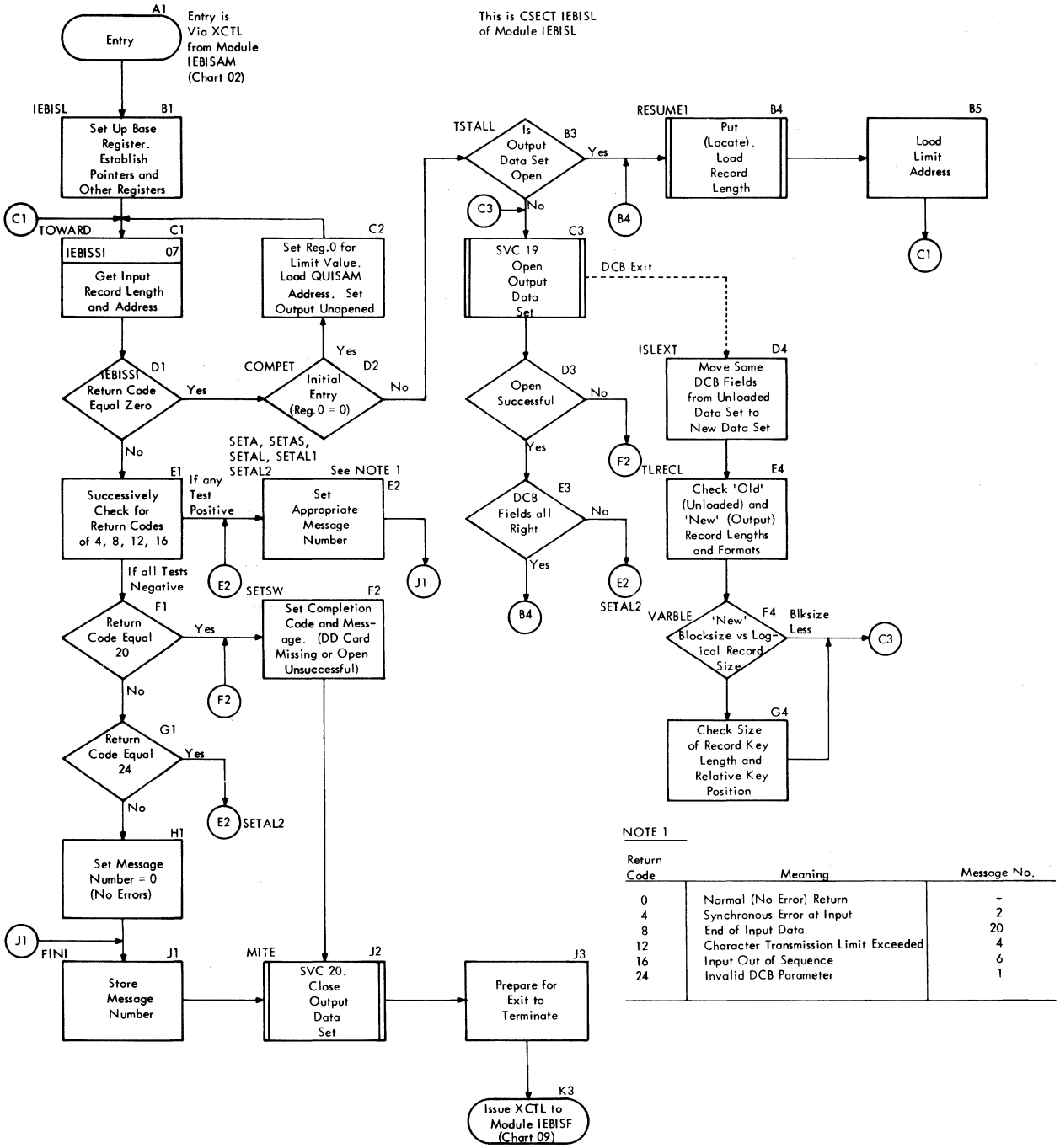
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 05. UNLOAD PHYSICAL SEQUENTIAL RECORDS (IEBISSO)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 06. RECONSTRUCT INDEXED SEQUENTIAL RECORDS (IEBISL)

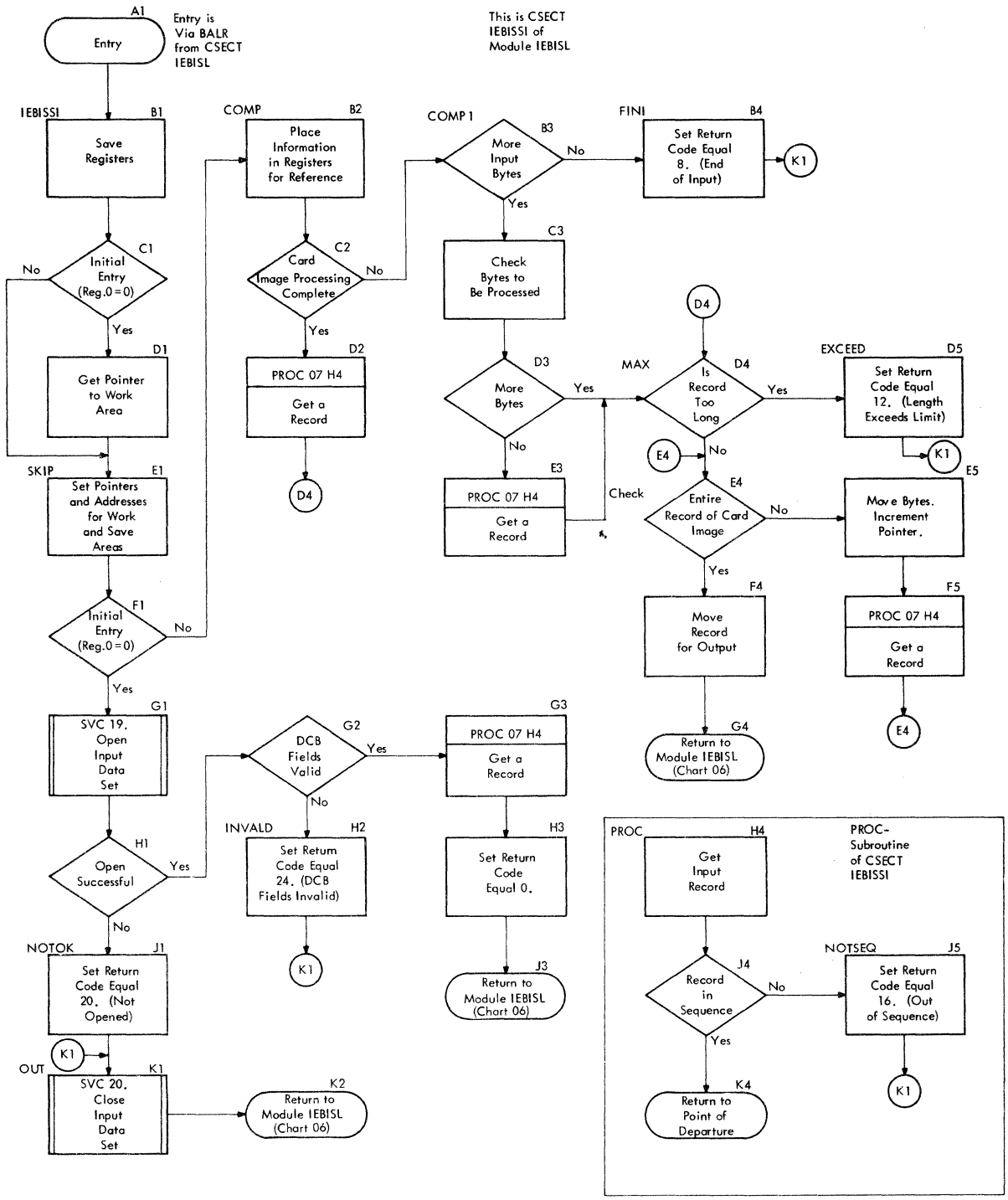


NOTE 1

Return Code	Meaning	Message No.
0	Normal (No Error) Return	-
4	Synchronous Error at Input	2
8	End of Input Data	20
12	Character Transmission Limit Exceeded	4
16	Input Out of Sequence	6
24	Invalid DCB Parameter	1

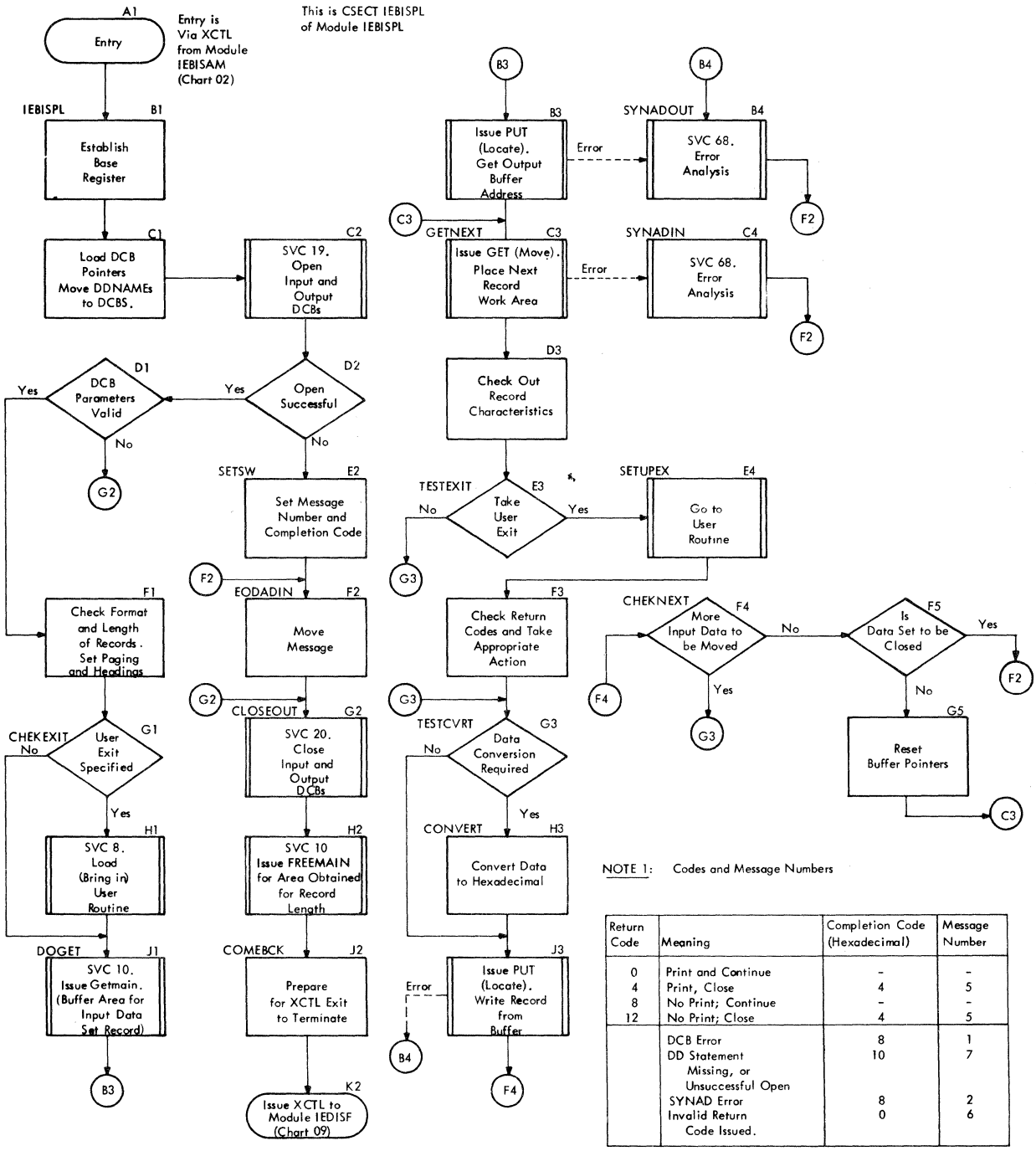
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 07. RETRIEVE PHYSICAL SEQUENTIAL RECORDS (IEBISSI)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 08. PRINT LOGICAL RECORDS (IEBISPL)

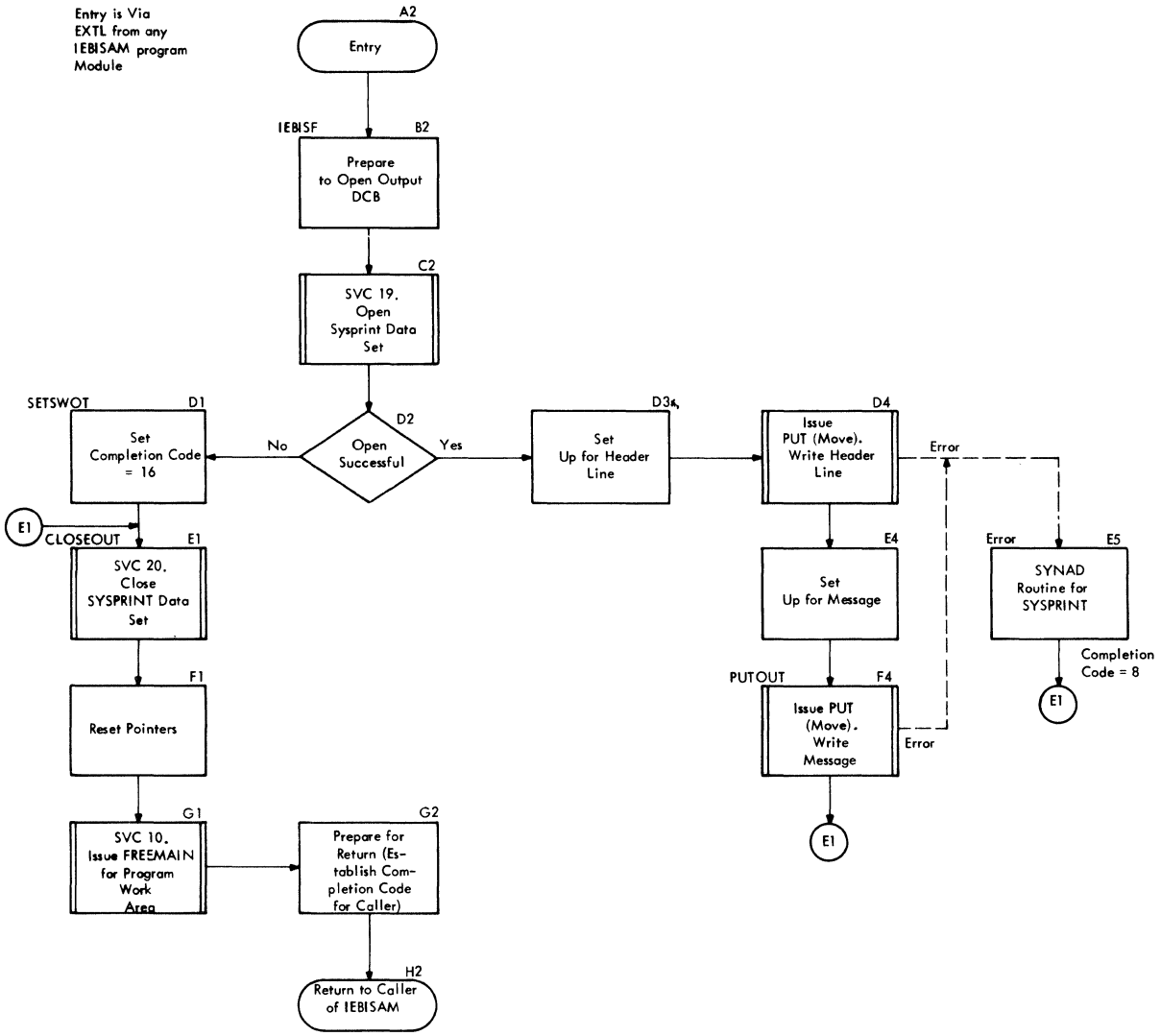


NOTE 1: Codes and Message Numbers

Return Code	Meaning	Completion Code (Hexadecimal)	Message Number
0	Print and Continue	-	-
4	Print, Close	4	5
8	No Print; Continue	-	-
12	No Print; Close	4	5
	DCB Error	8	1
	DD Statement Missing, or Unsuccessful Open	10	7
	SYNAD Error	8	2
	Invalid Return Code Issued.	0	6

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 09. TERMINATE IEBISAM PROGRAM (IEBISF)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

USING THE IEBPTPCH DOCUMENTATION

This chapter is divided into the following sections: "The IEBPTPCH Program," "Program Structure," "Method of Operation," "Tables and Work Areas," and "IEBPTPCH Flowcharts."

"The IEBPTPCH Program" provides a general description of the program. This overview concentrates on the uses and purposes of the program, and the inputs and outputs of the program.

"Program Structure" describes the physical organization of the program. It describes the function of each assembly module and shows how assembly modules are combined into load modules.

"Method of Operation" describes the IEBPTPCH program in terms of the operations it performs. Operation diagrams at two levels of detail show the movement of data through the tables, work areas, parameter lists, and data sets used by the program, and describe the processing performed at each stage. Supporting text expands on the information in each operation diagram and explains the operation in more detail.

Operation diagrams are not flowcharts. (Flowcharts are provided in another section.) An operation diagram shows where information comes from, what is done with it, and where the results are stored. Notes in the diagrams themselves indicate the names of the routines involved with the operation.

If you are already familiar with the program, the operation diagrams will act as a recall mechanism; if you are learning the program, the operation diagrams and the supporting text will explain the whole program (or the part you are interested in) quickly and graphically.

The first operation diagram presents an overview of the IEBPTPCH program. The diagram is divided into parts, each with an identifying number and a characteristic "shape." In the remainder of the section, each part is described in greater detail and identified by its corresponding number.

"Tables and Work Areas" contains detailed descriptions of the format and content of the major tables used by the program if they are referred to in more than one place in the text. Other tables, work areas, parameter lists, etc., are documented where they are referred to. An exception is the "constants and switches" area of each module, which is documented thoroughly in the listing. Where fields shown in diagrams or mentioned in the text are in such areas, the area is also mentioned.

"IEBPTPCH Flowcharts" contains a chart of each of the major program modules. Each chart shows program labels where possible, so that you should be able to see the general logic of the module from the chart and find the details in the listing.

Appendix A contains a description of each module of the program.

THE IEBPTPCH PROGRAM

The IEBPTPCH program is a data set utility program used to print or punch all or selected portions of a data set. The program may be invoked by the job scheduler, or it may be invoked by a user's program.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

The primary input to the IEBPTPCH program is the data set to be printed or punched. This data set may be a sequential or a partitioned data set. It may consist of fixed-length records, variable-length records, or undefined format records. If the input data set is sequential, it may consist of variable-length spanned records. In any case, logical records may be blocked or unblocked.

The other inputs to the IEBPTPCH program are: the control data set, which consists of utility control statements; and the user parameter lists, which may be used (if the program is invoked by a user's program) to specify alternates for the standard ddnames and to specify the first page number for the message data set.

The IEBPTPCH program produces two data sets: the printed or punched output data set, which contains records from the input data set edited as specified by the utility control statements; and the message data set, which contains warning and error messages.

PROGRAM STRUCTURE

The IEBPTPCH program consists of three program segments, which form the structure shown in Figure 107. The root segment, which contains the initializing routine (assembly module IEBPPUN1) and the message module (assembly module IEBPPMSG), initializes the program, stores the parameters passed via the user parameter list, and writes messages to the SYSPRINT data set.

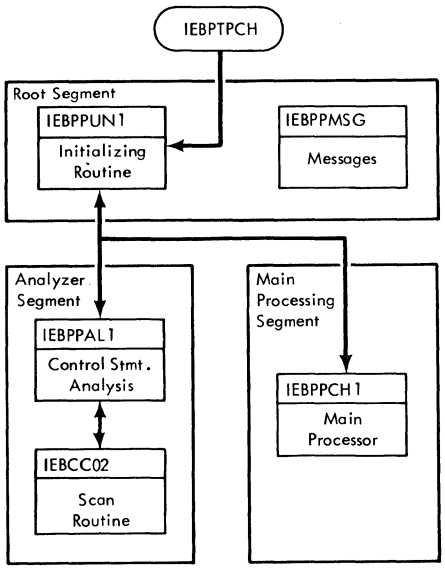


Figure 107. Program Structure

The analyzer segment consists of the control statement analysis routine (assembly module IEBPPAL1) and the scan routine (assembly module IEBCCS02). The scan routine opens, closes, and reads records from the SYSIN data set; the analysis routine analyzes and stores the utility control statement parameters.

The main processing segment contains the main processing routine (assembly module IEBPPCH1). This segment opens and closes the SYSUT1 and SYSUT2 data sets, and locates and processes SYSUT1 records as specified by the user.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

METHOD OF OPERATION

The IEBPTPCH program performs six basic functions. These functions, and the initial step of invoking the program, are described in this section.

Figure 108 on page 315 provides an overview of the operation of the IEBPTPCH program. The six basic operations, denoted by the large circled numbers in the diagram, are described under the corresponding numbers below. The invocation of the program and the basic operations are described in detail in the remainder of this section.

STORING INPUT PARAMETERS—(1)

The IEBPTPCH program may be invoked by the job scheduler or by a user's program. If it is invoked by a user's program, the user may construct a parameter list and pass alternate ddnames and a first page number for the message data set to the program. If a parameter list has been passed, the initializing routine (assembly module IEBPPUN1) stores the alternate ddnames in PPDDNAME and in PPOUT (the SYSPRINT DCB), and it stores the page number in PPHDRBL. The control statement analysis routine (assembly module IEBPPAL1) stores the alternate names for SYSIN, SYSUT1, and SYSUT2 in DTABLE.

PROCESSING CONTROL STATEMENTS—(2)

The utility control statement records, which describe the processing to be performed by the IEBPTPCH program, are in the SYSIN data set. The scan routine (assembly module IEBCCS02) reads and scans the statements. The control statement analysis routine (assembly module IEBPPAL1) obtains main storage for the required tables and stores the parameters in the member name table (MNT), the literal pool, the group control block (GCB), the title and subtitle areas, and the DTABLE.

SORTING MEMBER NAMES—(3)

The input data set, SYSUT1, may be a sequential or a partitioned data set (PDS). If it is a PDS, the main processing routine (assembly module IEBPPCH1) reads the PDS directory records into the directory area. It extracts the member name from each entry, moves them to the member area, and sorts them by TTR.

INITIALIZING THE MAIN PROCESSOR—(4)

The main processor (assembly module IEBPPCH1) moves the operating parameters stored in DTABLE to TABLE, stores the alternate names for SYSUT1 and SYSUT2 in the appropriate DCBs, and opens the input and the output data sets.

If the input data set is a PDS, the main processor uses a BLDL macro instruction and a FIND macro instruction to get the address of the member to be processed. If all members are to be processed, the routine uses the sorted member names in the member area; if the user has specified selected member names, the routine uses the names in the MNT.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

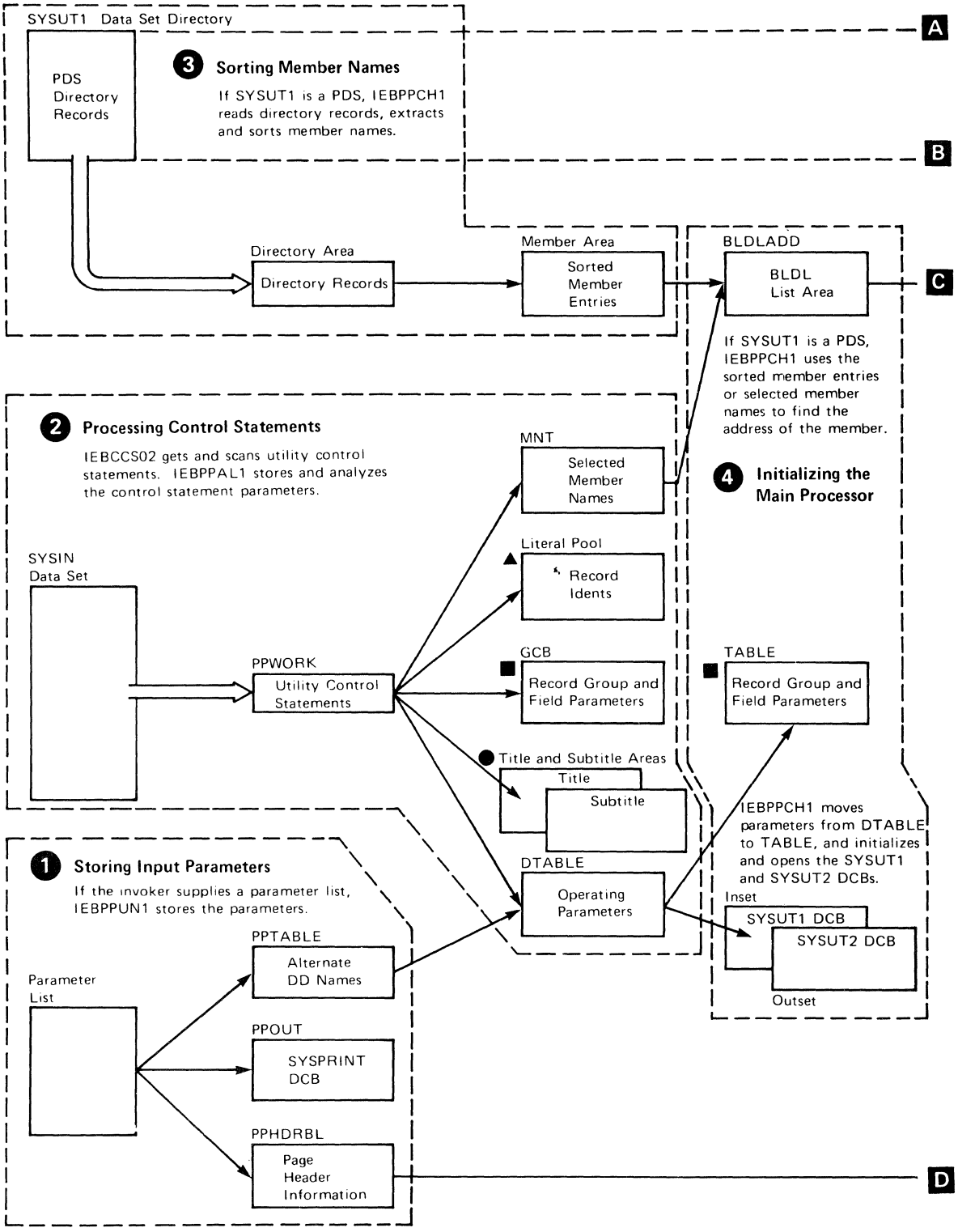


Figure 108 (Part 1 of 2). The IEBPTPCH Program

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

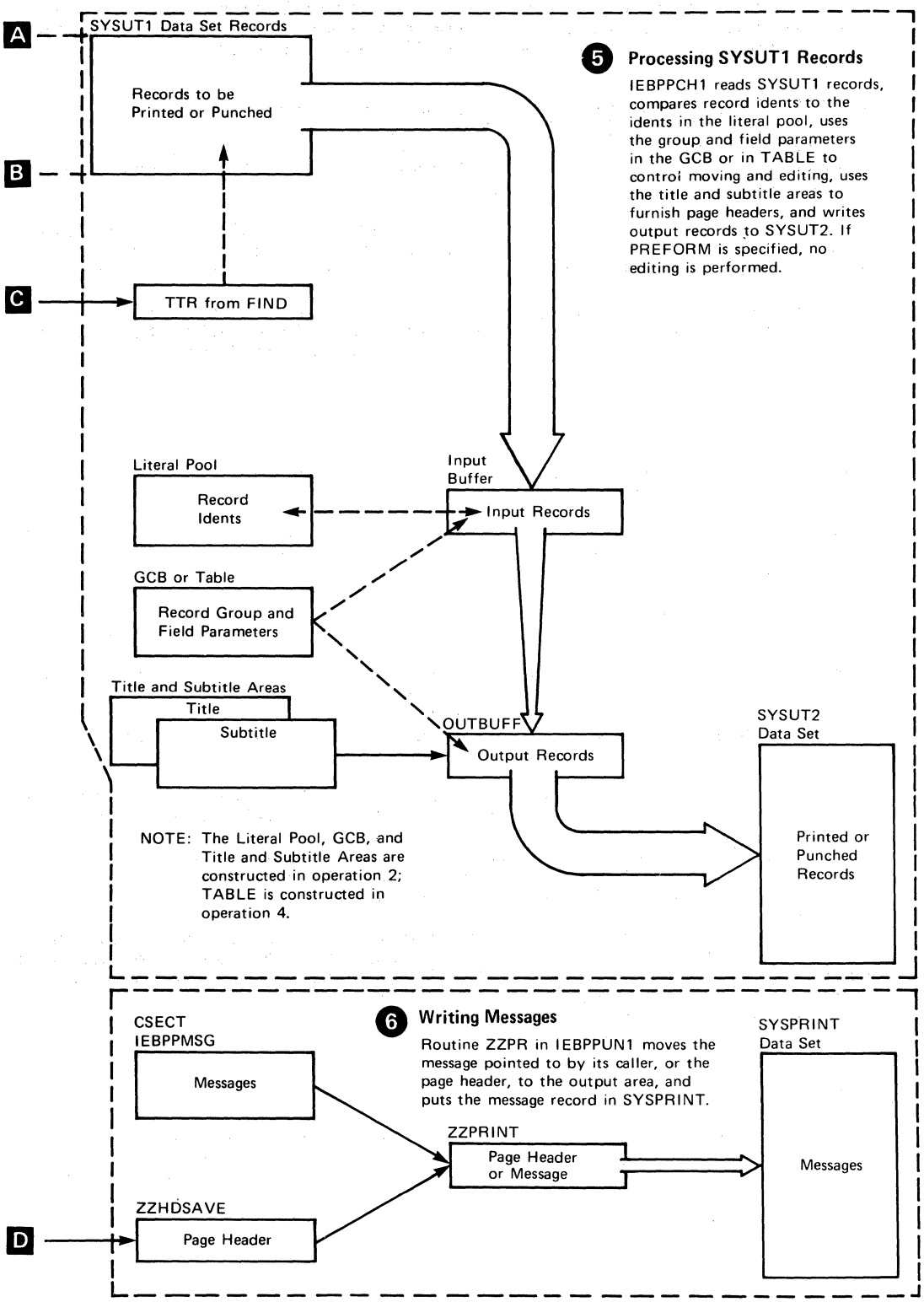


Figure 108 (Part 2 of 2). The IEBPTPCH Program

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

PROCESSING SYSUT1 RECORDS—(5)

The main processor reads records from the input data set into the input buffer, moves them to the output buffer, and writes them in the output data set. The movement of data between the input and output buffer is accomplished in one of the following ways:

If PREFORM processing is specified, the main processor performs no editing. The PREFORM routine moves the contents of the input buffer to the output buffer, segmenting input records as necessary, and uses the control characters in the input records to control printing or punching.

If standard editing is to be performed, the FORMS routine uses the parameter in TABLE to control data conversion and the movement of data characters from the input to the output buffer.

If user-specified editing is to be performed, the FORMU routine uses the parameters in the group control block (GCB) to control data conversion and the movement of data characters from the input to the output buffer.

WRITING MESSAGES—(6)

Messages from IEBPTPCH are written in the SYSPRINT data set by routine ZZPR in the root segment (assembly module IEBPPUN1). When the routine is entered, register 0 contains the message length and, with register 1, points to the message. The routine obtains messages from CSECT IEBPPMSG, and obtains page headers as necessary from ZZHDSAVE.

INVOKING IEBPTPCH

The IEBPTPCH program may be invoked by either the job scheduler or a user's program. In either case, the user must supply job control language (JCL) statements that define the data sets used in the IEBPTPCH program and that cause the program to be invoked.

If IEBPTPCH is to be invoked by the job scheduler, the user must use the standard ddnames (SYSIN, SYSUT1, SYSUT2, and SYSPRINT), and must specify IEBPTPCH (or a procedure that executes IEBPTPCH) in the EXEC statement. The job scheduler passes control to IEBPTPCH with register 1 pointing to a 4-byte field containing X'80' in the high-order byte.

If IEBPTPCH is to be invoked by a user's program, the user may specify alternates for any of the standard ddnames, and may also specify the page number for the first page of the message data set. The user's program passes control to IEBPTPCH via a LINK or ATTACH macro instruction, after it has constructed the appropriate parameter lists.

STORING INPUT PARAMETERS—(1)

Parameters may be passed to the IEBPTPCH program when it is invoked by a user's program. The parameter lists, and the operation of storing the parameters, are shown in Figure 109 on page 318.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

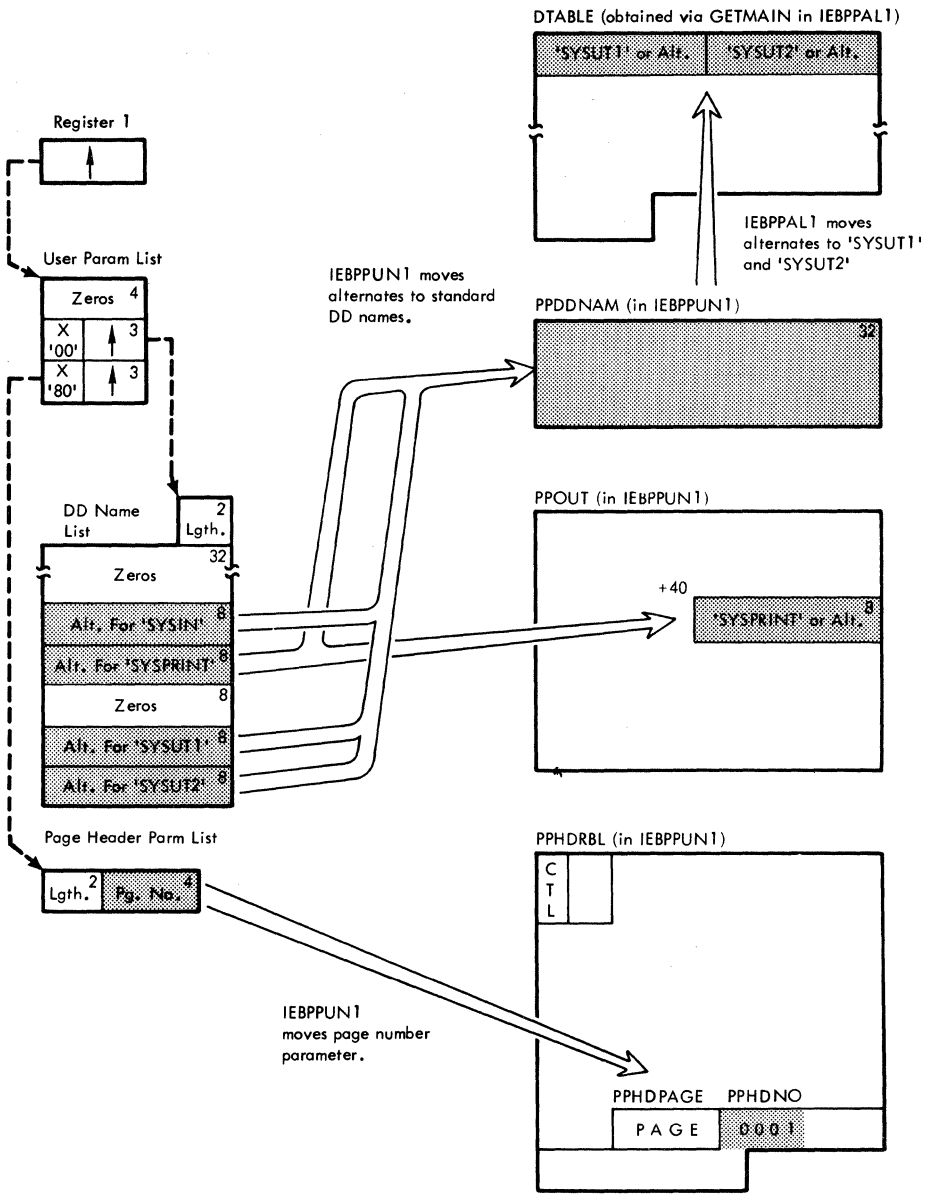


Figure 109. Storing Input Parameters

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

THE PARAMETER LISTS

The ddname parameter list contains the specified alternates for the standard ddnames. The order in which entries appear is fixed. As shown in the operation diagram, the 2-byte (unpacked decimal) length field is followed by 32 bytes of binary zeros. The remainder of the list is a series of 8-byte entries in the following order: the alternate for "SYSIN"; the alternate for "SYSPRINT"; binary zeros; the alternate for "SYSUT1"; and the alternate for "SYSUT2". Entries may be omitted from the end of the list (and the length field adjusted accordingly), but entries within the list must contain either a name or binary zeros.

The page header parameter list is a 6-byte field containing the length (in unpacked decimal) in the high-order 2 bytes and the first page number parameter (in unpacked decimal) in the low-order 4 bytes.

Both the ddname parameter list and the page header parameter list are optional. Their presence and locations are indicated by the user parameter list.

The user parameter list consists of one, two, or three entries of 4 bytes each. The first entry must be set to 0, and the last entry must contain X'80' in the high-order byte.

The second entry contains zeros in the high-order byte and either a pointer to the ddname list or zeros. If neither of the optional parameter lists is present, the second and third entries of the user parameter list are omitted.

The third entry in the user parameter list is present only if the page header parameter list is present. If the third entry is present, it contains X'80' in its high-order byte and a pointer to the page header parameter list in its 3 low-order bytes.

STORING THE PARAMETERS

When the IEBPTPCH program is entered, the initializing routine (assembly module IEBPPUN1) stores any specified alternate ddnames in the ddname list area, labeled PPDDNAM. If an alternative for "SYSPRINT" is specified, the routine stores the alternate name over the standard name in the ddname field of the DCB for the message data set, as well as in the ddname list area. The control statement analysis routine (assembly module IEBPPAL1) moves the alternates for "SYSUT1" and "SYSUT2" from PPDDNAM to DTABLE.

If a page number parameter was specified, the initializing routine stores the number in the area labeled PPHDRBL.

PROCESSING CONTROL STATEMENTS—(2)

Utility control statements in the control data set (SYSIN) describe the processing to be performed on the input data set (SYSUT1) by the IEBPTPCH program. This section describes the interface between the control statement analysis routine and the scan routine, which is a subroutine of the control statement analysis routine. The scan routine reads and scans control statements, and the analysis routine stores control statement parameters in various tables and work areas.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

THE SCAN/ANALYSIS INTERFACE

The scan routine (assembly module IEBCCS02) is a subroutine of the analysis routine (assembly module IEBPPAL1). Figure 110 shows the interface between the two routines, and Figure 111 on page 321 shows how control statement parameters are stored.

Each time it is ready to process a command (for example, PRINT, MEMBER, EXITS), a keyword, or a parameter, the analysis routine passes control to the scan routine via a CALL macro instruction, with the parameter list shown at the upper left in Figure 110. The scan routine opens the SYSIN data set the first time it is entered, uses the GET macro instruction to obtain the first record, and scans for the command field. On each subsequent entry, the scan routine scans another field, obtaining a new record whenever necessary, and closing the SYSIN data set at end-of-data. Each time it has scanned a field, the scan routine returns control to the analysis routine, and provides the parameter list shown at the upper right of the figure.

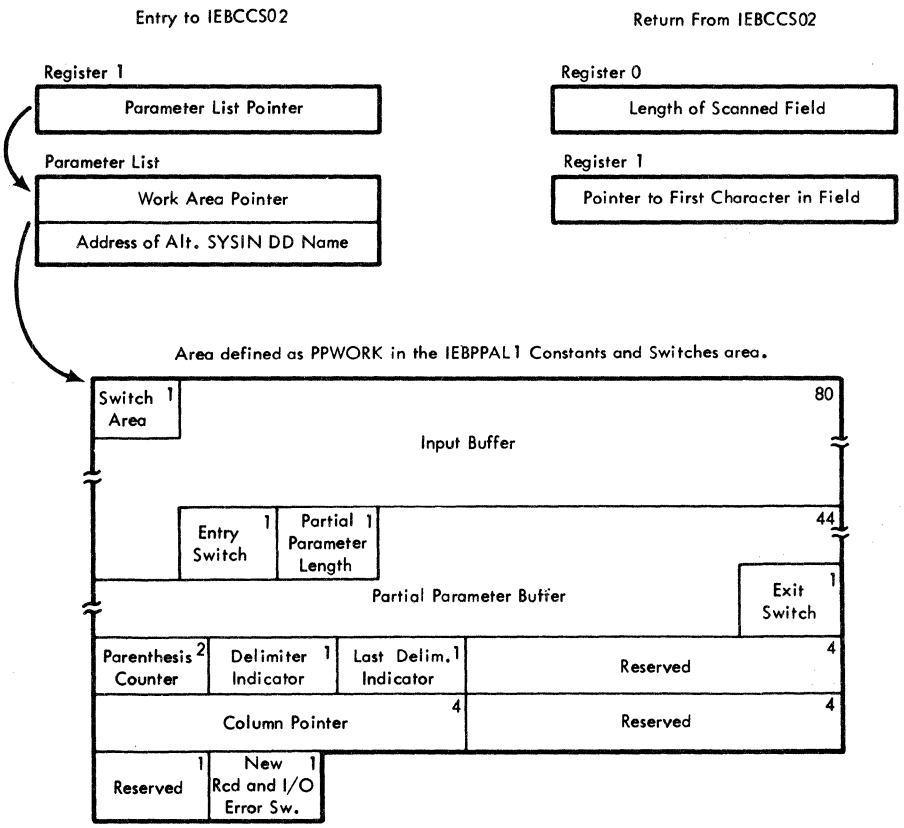


Figure 110. IEBPPAL1/IEBCCS02 Interface

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

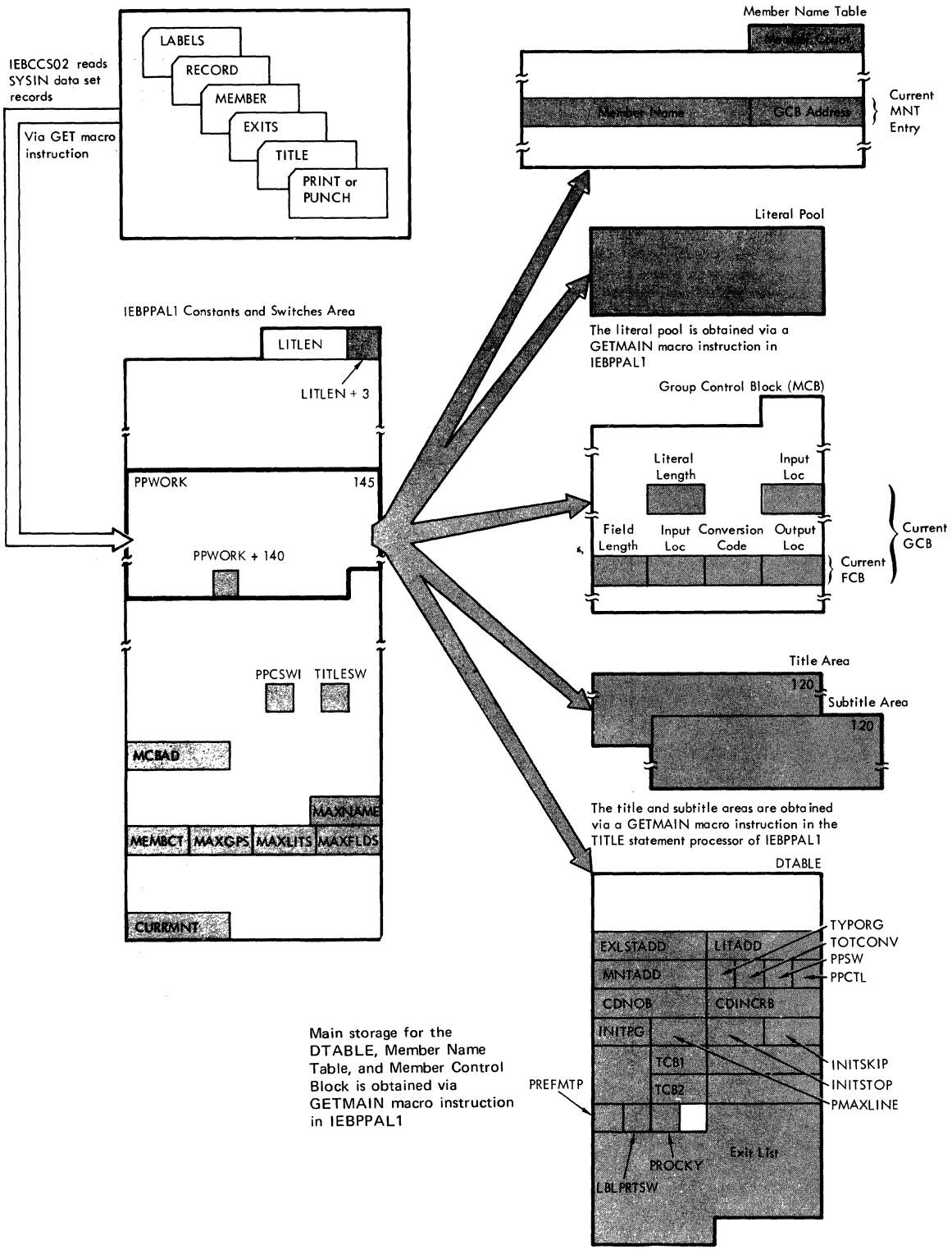


Figure 111. Storing Control Statement Parameters

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

In addition to the parameter lists, the two routines communicate via the work area shown in the figure. The work area, named PPWORK, is defined in the analysis routine constants and switches area. It contains the following fields:

- Switch Area (1 byte): The bits in this field are set by the scan routine. When set to 1, they have the following meanings:

Bit	Meaning
0	Syntax error.
1	No more fields to process in this record.
2	End-of-data.
3	Not first entry (this bit is off for first entry only).
4	Scanned field is a command field.
5	Scanned field is a keyword field.
6	Scanned field is a parameter field.
7	Last parameter in record, or last subparameter of group.

- Input Buffer (80 bytes): This field is the input area for SYSIN data set records.

- Entry Switch (1 byte): Bits 0 to 5 in this field are set by the scan routine; bits 6 and 7 are set by the analysis routine. When set to 1, the bits have the following meanings:

Bit	Meaning
0	Reserved.
1	Statement continued.
2	Next field is a keyword or parameter field.
3	Parameter continued on next record.
4	Read a record.
5	Scan for the command field.
6	Scan for the next keyword.
7	Scan a parameter.

- Partial Parameter Length (1 byte): If a parameter is continued on a second record, the scan routine counts the number of characters of the parameter that are in the current record, and stores the count in this field.

- Partial Parameter Buffer (44 bytes): If a parameter is continued on a second record, the scan routine stores the first part of the parameter in this field, then reads the next record, and stores the remainder of the parameter in this field. When the continuation statement has been read, this field contains the complete parameter.

- Exit Switch (1 byte): When set to 1, the bits in this field have the following meanings:

Bit	Meaning
0	Reserved.
1	Condition code 1: delimiter; not end of record.
2	Condition code 2: delimited; end of record.
3	Reserved.
4	Position 72 is blank.
5	Position 72 is not blank.
6	Field length is greater than 0.
7	Field length is 0.

- Parenthesis Counter (2 bytes): A 1 is added to this field each time a left parenthesis is encountered; a 1 is subtracted each time a right parenthesis is encountered.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- Delimiter Indicator (1 byte): When set to 1, the bits in this field have the following meanings:

Bit	Meaning
0	Delimiter is an equal sign.
1	Delimiter is a comma.
2	Delimiter is a left parenthesis.
3	Delimiter is a right parenthesis.
4	Delimiter is blank.
5	Delimiter is an apostrophe.
6	Parenthesis not closed.
7	Reserved.

- Last Delimiter Indicator (1 byte): The bits in this field have the same meaning as the corresponding bits in the delimiter indicator field. This field, however, refers to the previous delimiter instead of the current delimiter.

- Column Pointer (4 bytes): This field contains a pointer to the beginning of the field to be scanned.

- New Record and I/O Error Switch (1 byte): This field is set to X'01' when a new record has been read, and to X'02' if an I/O error occurs.

CONSTRUCTING TABLES

The analysis routine (assembly module IEBPPAL1) initializes its work area, and then uses the GETMAIN macro instruction to obtain main storage for the DTABLE (which is defined by a DSECT). After it has processed the PRINT or PUNCH statement, the routine calculates the amount of storage required for the member name table (MNT) and group control block (GCB), and uses GETMAIN macro instructions to obtain the storage. If the keyword MAXLITS was specified in the PRINT or PUNCH statement, the analysis routine also calculates the amount of storage required for the literal pool, and uses the GETMAIN macro instruction to obtain the storage. The formats of the DTABLE, the GCB, and the MNT are shown in Figure 122 on page 339, Figure 123 on page 342, and Figure 124 on page 343, respectively; the literal pool is an area big enough to contain all the identifications specified in RECORD statements.

In addition to the tables discussed above, the analysis routine obtains storage for a title area and a subtitle area when the TITLE statements are encountered.

STORING CONTROL STATEMENT PARAMETERS

The first utility control statement in the SYSIN data set must be either a PRINT or PUNCH statement. The control statement analysis routine (assembly module IEBPPAL1) validates and stores each parameter, either in the IEBPPAL1 constants and switches area, or in DTABLE.

The scan routine reads subsequent control statements and passes the fields to the analysis routine. When the scan routine passes a command code, the analysis routine compares it with each of the valid commands. If the command is valid, the analysis routine stores the keywords and parameters.

- If the statement is a TITLE statement, the analysis routine uses the GETMAIN macro instruction to obtain 120 bytes of storage and stores the title at the offset specified in the "location" subparameter. If two TITLE statements are included, the second is considered to be a subtitle statement. No more than two TITLE statements are permitted.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- If the statement is an EXIT statement, the analysis routine stores the names of the user's label and record processing routines, and the address of the list of names, in DTABLE.
- If the statement is a MEMBER statement, the analysis routine constructs an MNT entry and stores a pointer to the entry in the constants and switches area.
- If the statement is a RECORD statement, the analysis routine stores the "name" subparameter in the literal pool and constructs a group control block (GCB) entry and a field control block (FCB). The format of an FCB is shown in Figure 123 on page 342.
- If the statement is a LABELS statement, the analysis routine sets the appropriate switches in PPTABLE.

SORTING MEMBER NAMES—(3)

If the SYSUT1 data set is a partitioned data set (PDS), the main processor (assembly module IEBPPCH1) sorts the PDS directory entries by member name and TTR. The operations of reading, counting, and extracting records from the PDS are shown in Figure 112 on page 325; the sort itself is shown in more detail in Figure 113 on page 326.

The main processor's total routine uses the GETMAIN macro instruction to obtain main storage for a 264-byte directory area. The routine uses the READ macro instruction to read directory records into the area, counts the number of directory entries, and stores the count in MEMBCT.

The total routine uses the count to calculate the size of the member area (12 bytes per member) and obtains the storage via a GETMAIN macro instruction. It uses the READ macro instruction to reread the directory records (into the directory area), and moves the name, TTR, and indicator fields of each entry into the member area.

When it has stored the entries in the member area, the routine releases the main storage obtained for the directory area.

The DSORT1 routine of the main processor sorts the entries in the member area using the sorting procedure shown in Figure 113. As a result of the sort, member area entries are in order by TTR, and the entry containing the true name of a member appears before the entries containing its aliases.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

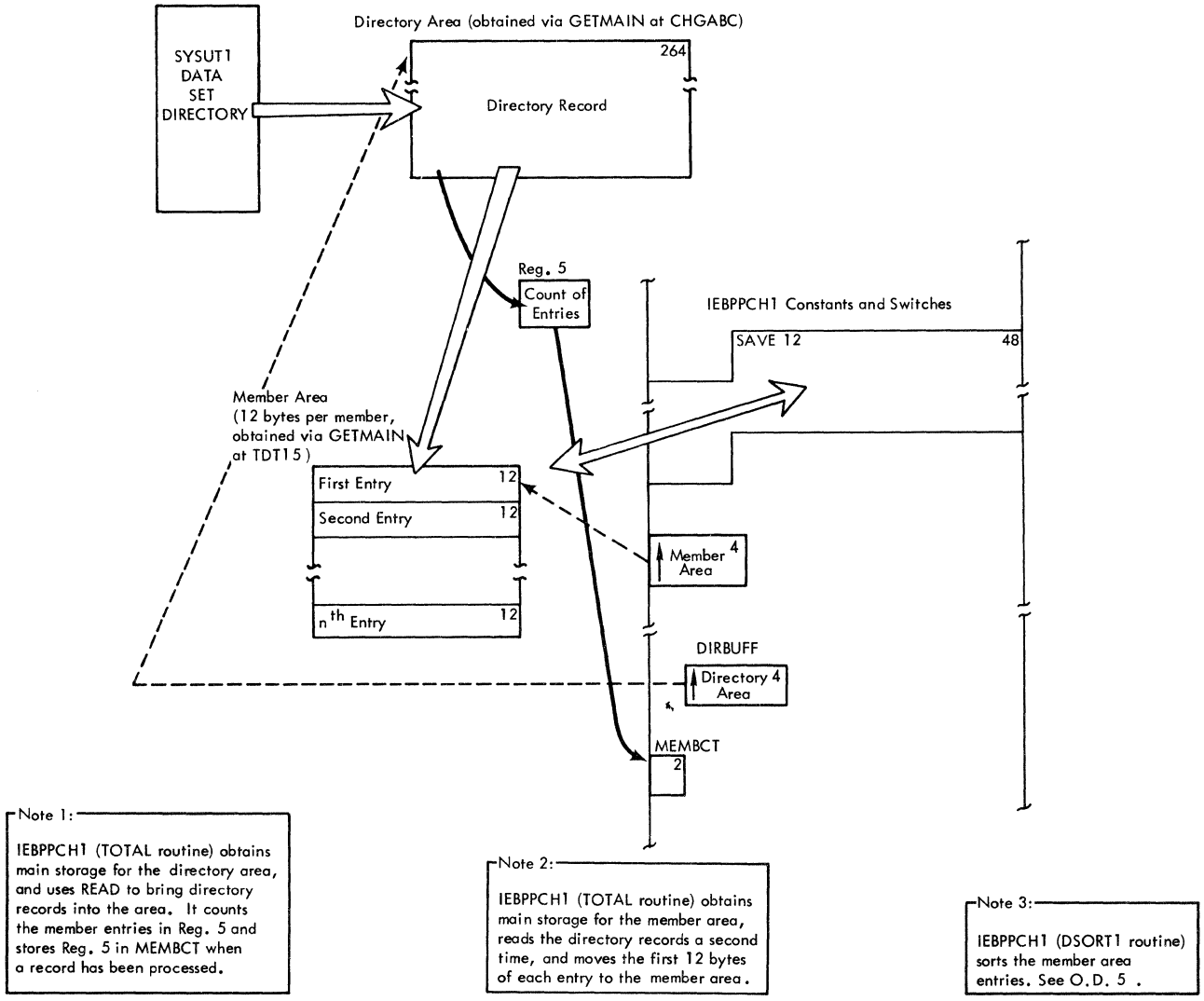


Figure 112. Reading, Counting, and Extracting Directory Records

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

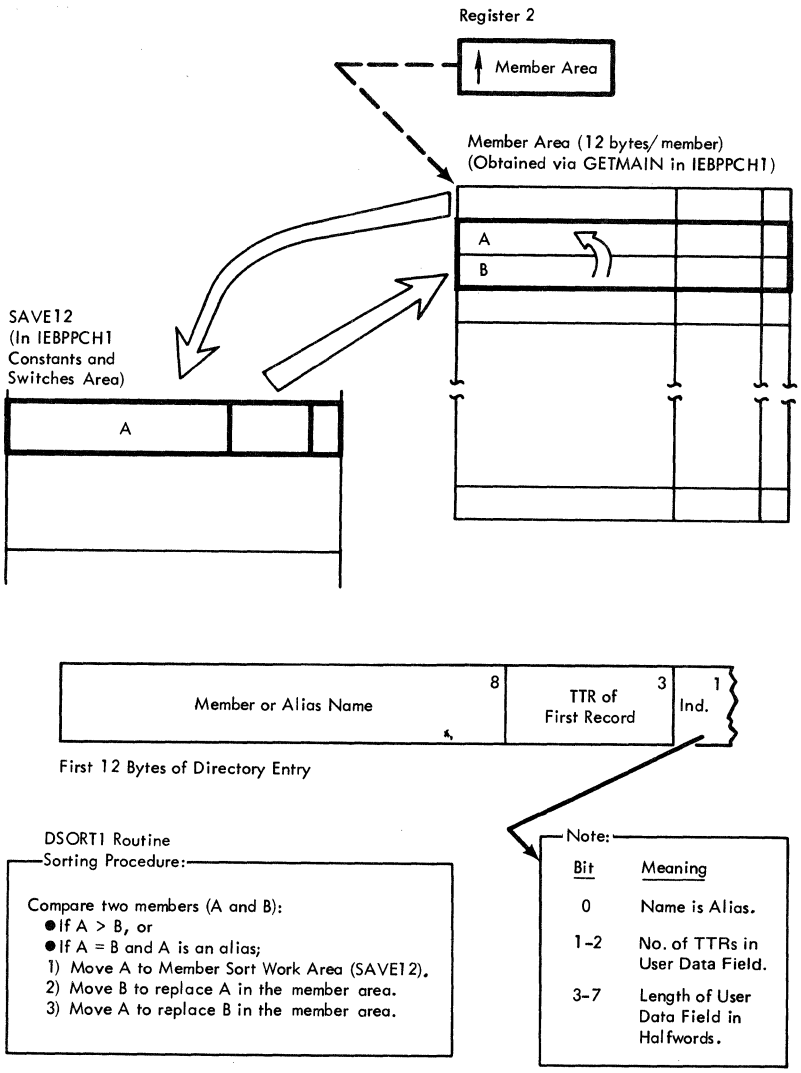


Figure 113. IEBPPCH1 Member Entry Sort

INITIALIZING THE MAIN PROCESSOR—(4)

In order to process the input data set (SYSUT1), the main processor (assembly module IEBPPCH1) needs access to the processing parameters passed via user parameter lists and the control statement parameters. If the input data set is a partitioned data set (PDS), the main processor also needs the address of the first record of each member to be processed. Figure 114 on page 327 shows how the main processor obtains the processing parameters; Figure 115 on page 328 shows how it finds the address of the first record of the member to be processed.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

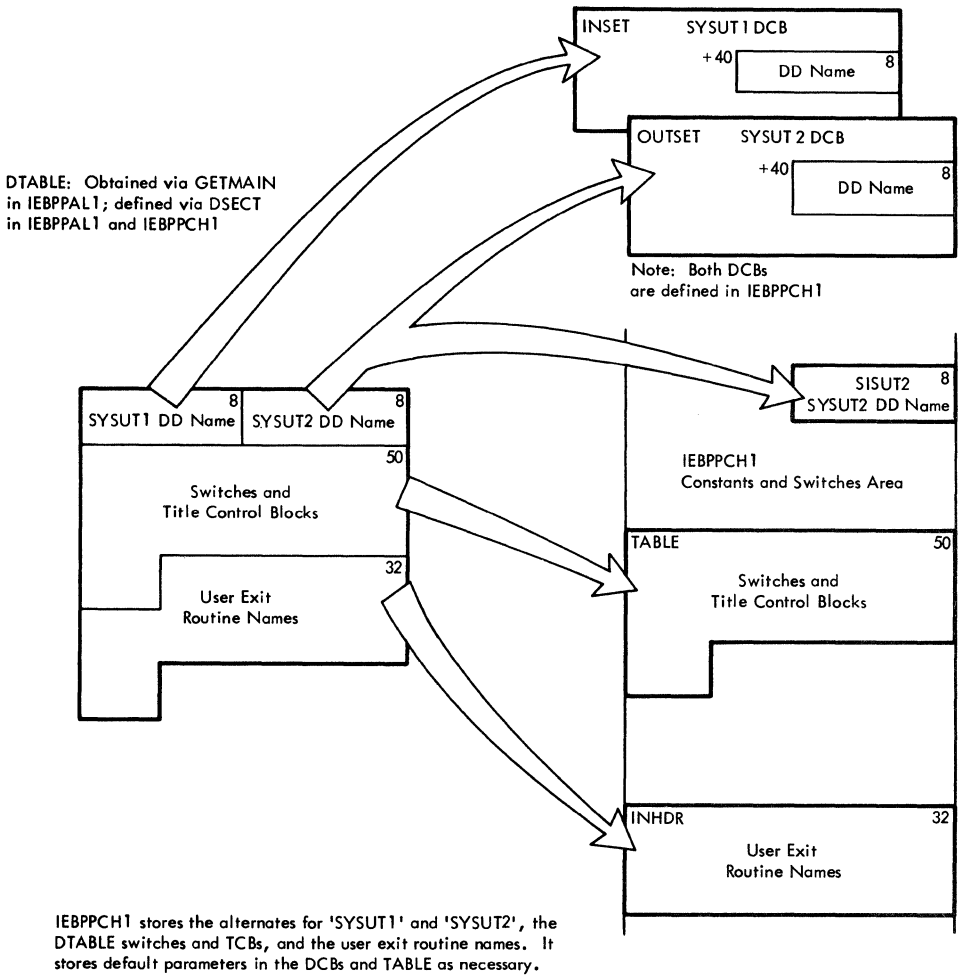


Figure 114. Initializing the Main Processing Routine (IEBPPCH1)

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

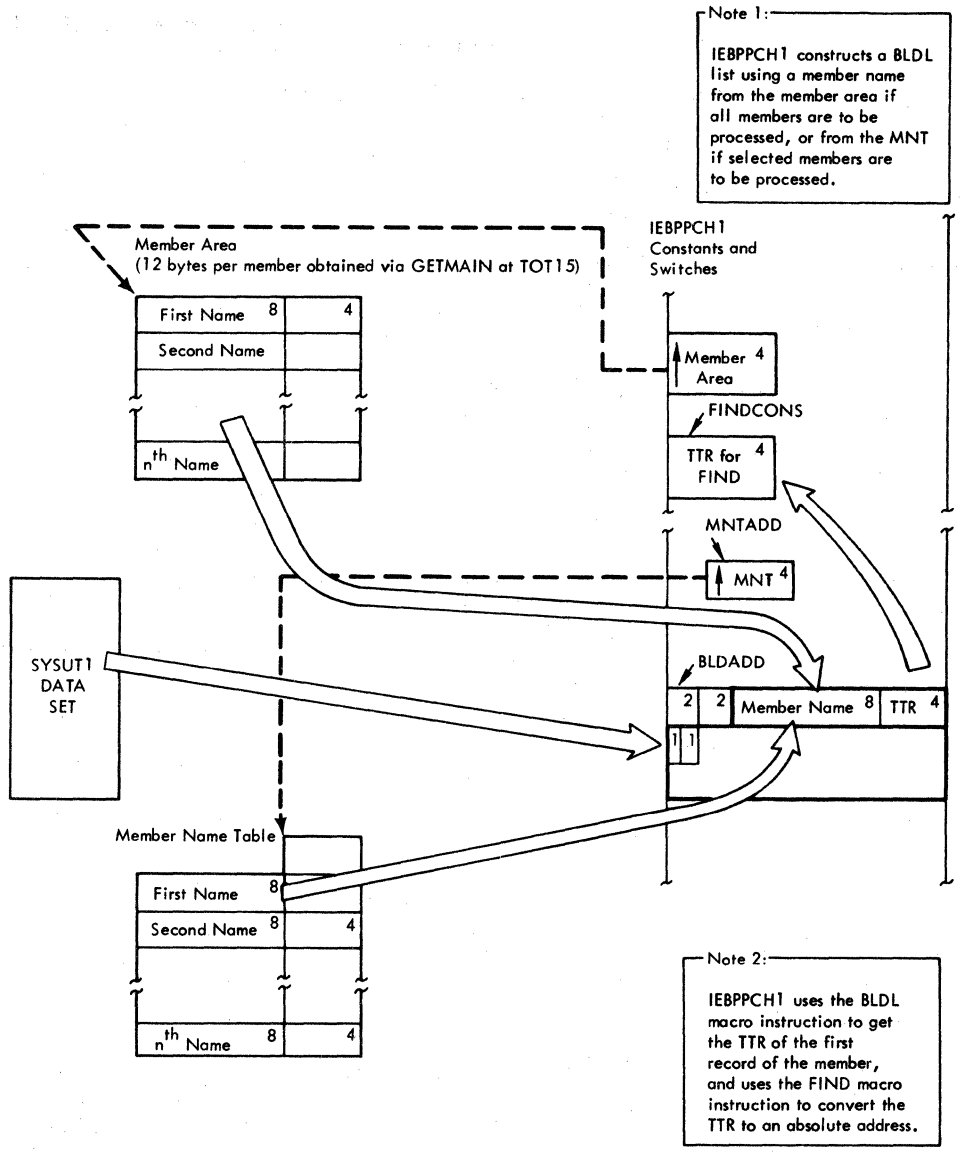


Figure 115. Locating a Member

OBTAINING PROCESSING PARAMETERS

Many of the processing parameters and pointers to the others are stored in an area of main storage obtained in the control statement analysis routine. When the main processor is entered, register 1 contains a pointer to the area.

As shown in Figure 122 on page 339, the area is defined twice. The analysis routine, which obtains main storage for the area, defines it in detail, using a DSECT named DTABLE. The main processor defines the area generally, using a DSECT also named DTABLE.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

As shown in Figure 114, the main processing routine moves blocks of information from DTABLE to other areas:

- It overlays the standard ddnames in the SYSUT1 and SYSUT2 DCBs with the specified alternates.
- It stores switches, title control information, and subtitle control information in the TABLE portion of its constants and switches area.
- It stores the names of the user's exit routines in the INHDR portion of its constants and switches area, and then uses the LOAD macro instruction to bring the routines into main storage.

LOCATING MEMBERS

If the input data set is a PDS, the main processor finds the name of the next member to be processed and uses the BLDL and FIND macro instructions to find the address of the first record of the member. The process is shown in Figure 115 on page 328.

The user may specify the names of the members that are to be processed. If these names are specified, the control statement analysis routine constructs a member name table (MNT), as described earlier under "Storing Control Statement Parameters"; the main processor has the address of the MNT in the field MNTADD in the constants and switches area. If MNTADD is nonzero, the main processor uses the names in the MNT, placing the next unused name in the BLDL list.

If the user does not specify names of members tc be processed, MNTADD is zero, and all members are to be processed. In this case, the main processor uses the names in the member area, described earlier under "Sorting Member Names." The main processor places the next unused member name (it skips aliases) in the BLDL list.

When it has obtained a member name for the BLDL list, the main processor issues the BLDL macro instruction to obtain the TTR of the first record, and then uses the FIND macro instruction to convert the TTR to an absolute address.

PROCESSING SYSUT1 RECORDS—(5)

The main processor (assembly module IEBPPCH1) brings records from the input (SYSUT1) data set into the input buffer, moves them to the output buffer, and puts them in the output (SYSUT2) data set. The processing is shown in Figure 116 on page 330 through Figure 119 on page 335.

If the input data set consists of variable-length spanned (VS or VBS) records with a block size of less than 32K bytes, the main processor QSGETREC routine uses the GET macro instruction to obtain the records. Otherwise, the main processor RDCH routine obtains records via the READ macro instruction. If the records are VS or VBS records with a block size greater than 32K bytes, the main processor routines process physical records. Otherwise, they deblock the physical records and process logical records.

If the user has specified PREFORM, the main processor assumes that the records are formatted and completely processed. It moves each input record to the output buffer (in segments if necessary) and supplies control characters only to segmented records.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

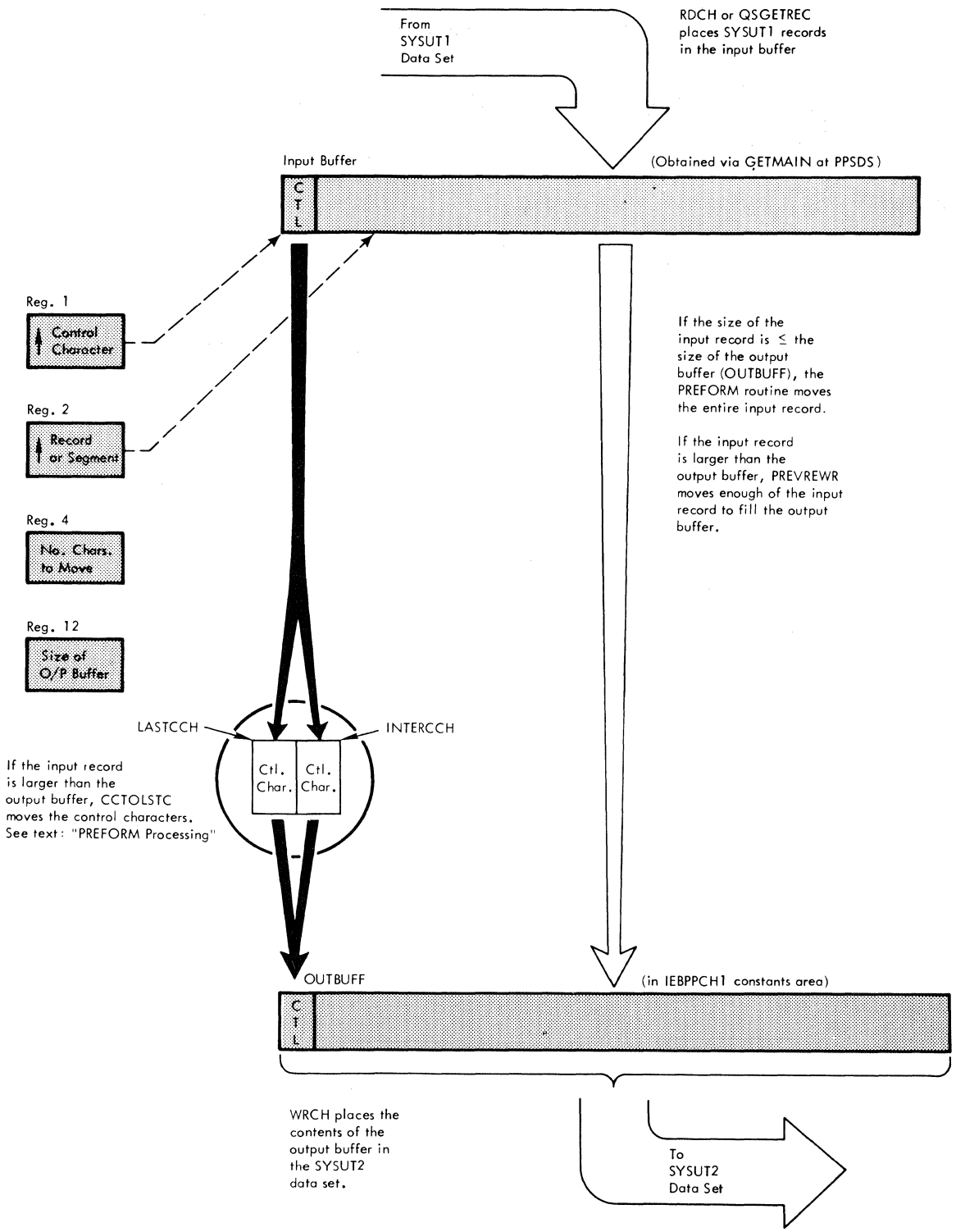


Figure 116. PREFORM Processing

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

If the user has not specified PREFORM, the main processor uses the IDENT (or STOPAFT) and SKIP parameters to identify the records to be processed. If a record is to be processed, the main processor passes control to a user exit routine (if specified), and then edits the record according to user or standard specifications.

When the record has been edited, the main processor passes control to a user exit routine (if specified), and then places the record in the output data set, supplying card sequence numbers or page headers as required.

PROCESSING FORMATTED RECORDS (PREFORM SPECIFIED)

If the user specifies PREFORM, the main processor performs no editing functions, and provides no user processing exits, page headers, or card sequence numbers. As shown in Figure 116, the RDCH or QSGETREC routine obtains a record from the input data set and places it in the input buffer. If the input record is no larger than the output buffer, the PREFORM routine moves the record and its control character to the output buffer in one operation. The WRCH routine uses the PUT macro instruction to place the record in the output (SYSUT2) data set.

If the input record is larger than the output buffer, the PREVREWR routine moves the record to the output buffer in segments. The CCTOLSTC routine performs the required control character processing:

- If the output is being punched, each segment of the record needs the same control character. CCTOLSTC stores the control character from the input record in LASTCCH and INTERCCH.
- If the output is being printed and the input data set uses ASA control characters (PREFORM=A), CCTOLSTC stores a "single space" control character (a blank) in LASTCCH and INTERCCH.
- If the output is being printed and the input data set uses machine control characters (PREFORM=M), CCTOLSTC stores the control character from the record in LASTCCH. It stores a "print and skip one line" control character (X'09') in INTERCCH and in the control character byte of the input record.

The control character stored in the control character byte of the input record is used in the first output segment. INTERCCH furnishes the control character for all subsequent segments except the last, and LASTCCH furnishes the control character for the last segment. The WRCH routine uses the PUT macro instruction to place the record in the output data set.

PROCESSING UNFORMATTED RECORDS (PREFORM NOT SPECIFIED)

When the input data set consists of unformatted records (PREFORM is not specified), the main processor identifies the records to be processed. If a record is to be processed, the routine passes control to the user input record exit routine, then performs either the standard editing procedure or the user-specified editing procedure. When the record has been edited, the main processor passes control to the user output record exit routine, then puts the edited record in the output data set. Figure 117 on page 332 shows how the routine identifies the records to be processed, and Figure 118 and Figure 119 show how the editing is performed.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

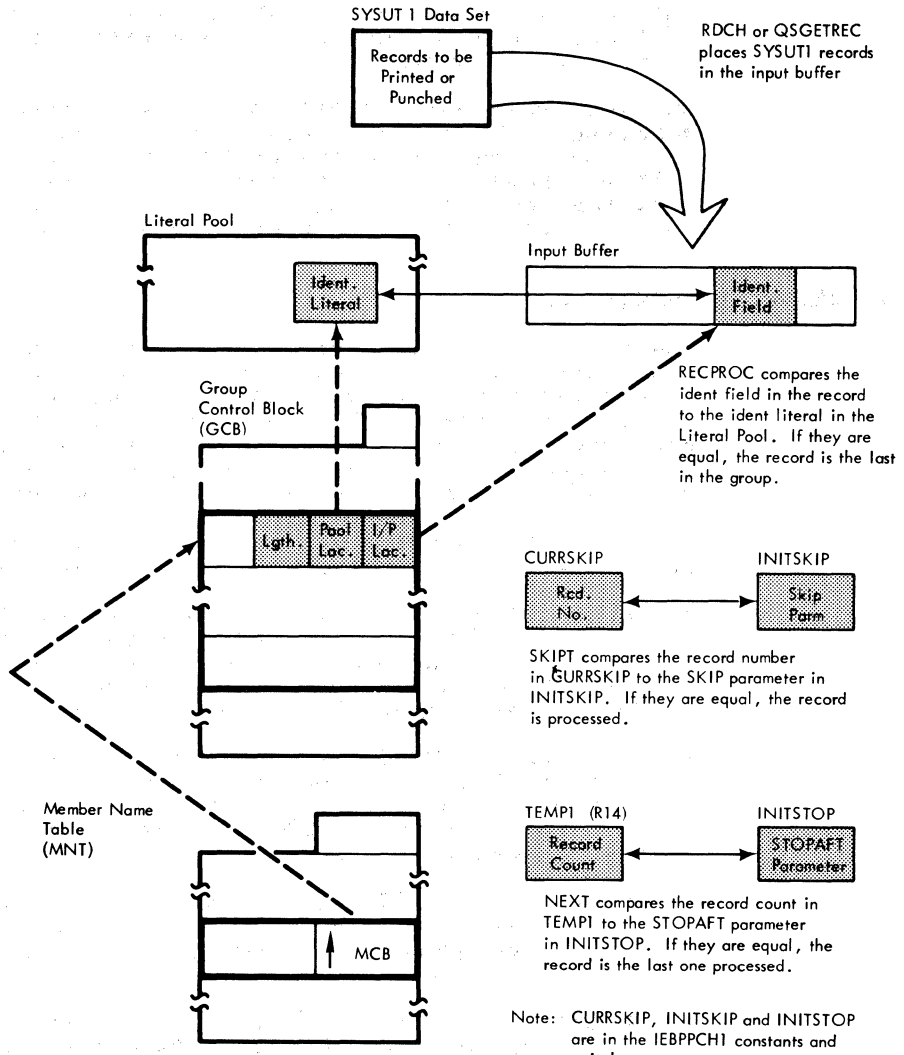


Figure 117. Identifying Records

IDENTIFYING RECORDS

When a record has been brought into the input buffer, the main processor determines whether it is to be processed, using the STOPAFT or IDENT parameter and the SKIP parameter. The process is shown in Figure 117.

The STOPAFT parameter specifies the number of records in the data set or PDS member to be processed. Each time a record is placed in the input buffer, the main processor increases register 14 (TEMP1) by 1, and compares the contents with the contents of the INITSTOP field in the constants and switches area. INITSTOP contains the STOPAFT parameter, and, if the two quantities are equal, the current record is the last in the data set or PDS member to be processed.

The IDENT parameter specifies the last record in a group of records that are to be similarly processed. The parameter is stored in the literal pool, and is compared with the

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

identification field of each record. When the two fields are equal, it signifies that the record is the last in the group; after the current record has been processed, the routine uses the next group control block (GCP) entry to control processing. If there are no more entries, processing of the data set or member is terminated.

The SKIP parameter specifies that every nth record is to be processed. Each time a record is placed in the input buffer, the main processor increases CURRSKIP (in the constants and switches area) by 1. It compares CURRSKIP with INITSKIP, which contains the SKIP parameter. If the two are equal, the record is processed; if not, the routine obtains another record.

EDITING TO STANDARD OUTPUT FORMAT

The standard printed output format is 12 groups of 8 characters each, with two blanks between groups. The standard punched output format is one group of 80 characters (72 characters if CDSEQ is specified). The main processor forms routine moves enough bytes from the input record to the output buffer to make up one group. It uses the FLDCONV routine to convert data to unpacked decimal or to hexadecimal representation.

As shown in Figure 118 on page 334, the movement of data is controlled by information in four registers:

- Register 3 (FLDCT) is loaded from MAXSEG. It contains the number of character groups per output record (12 or 1). The register is decreased by one each time a character group is stored in the output buffer. When FLDCT reaches zero, the WRCH routine puts the record in the output data set.
- Register 2 (ROUT) is initialized to point to the first data byte in the output buffer. Each time a character group is stored in the output buffer, ROUT is increased by either 10 (8 characters and 2 blanks) for printed output, or by 72 or 80 for punched output.
- Register 7 (RLOC) is initialized with the pointer to the input buffer from RECDLOC, and is increased each time a character group is stored by the number of bytes moved from the input buffer.
- Register 11 (INLEN) controls the number of bytes moved from the input buffer. Unless data conversion is specified, INLEN contains the number of characters per output group (8, 72, or 80). If data conversion is specified, INLEN contains half the number of characters per output group.

When the output buffer is full, the main processor UFEND routine passes control to the user's output record exit routine. The WRCH routine stores the control character and (if CDSEQ was specified) the sequence number, and puts the record in the output data set.

EDITING TO USER SPECIFICATIONS

The user specifies the output format via the FIELD parameters in RECORD statements. Each FIELD parameter results in the construction of a field control block (FCB) in the current GCB entry. The FCB specifies the size of the field (in the input record), points to the field's location in the input and output records, and indicates whether data conversion is to take place.

The main processor FORMU routine first computes the total length of all fields to be processed. If the total length does not exceed the maximum output record size, the FORMU routine processes each FCB in turn.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

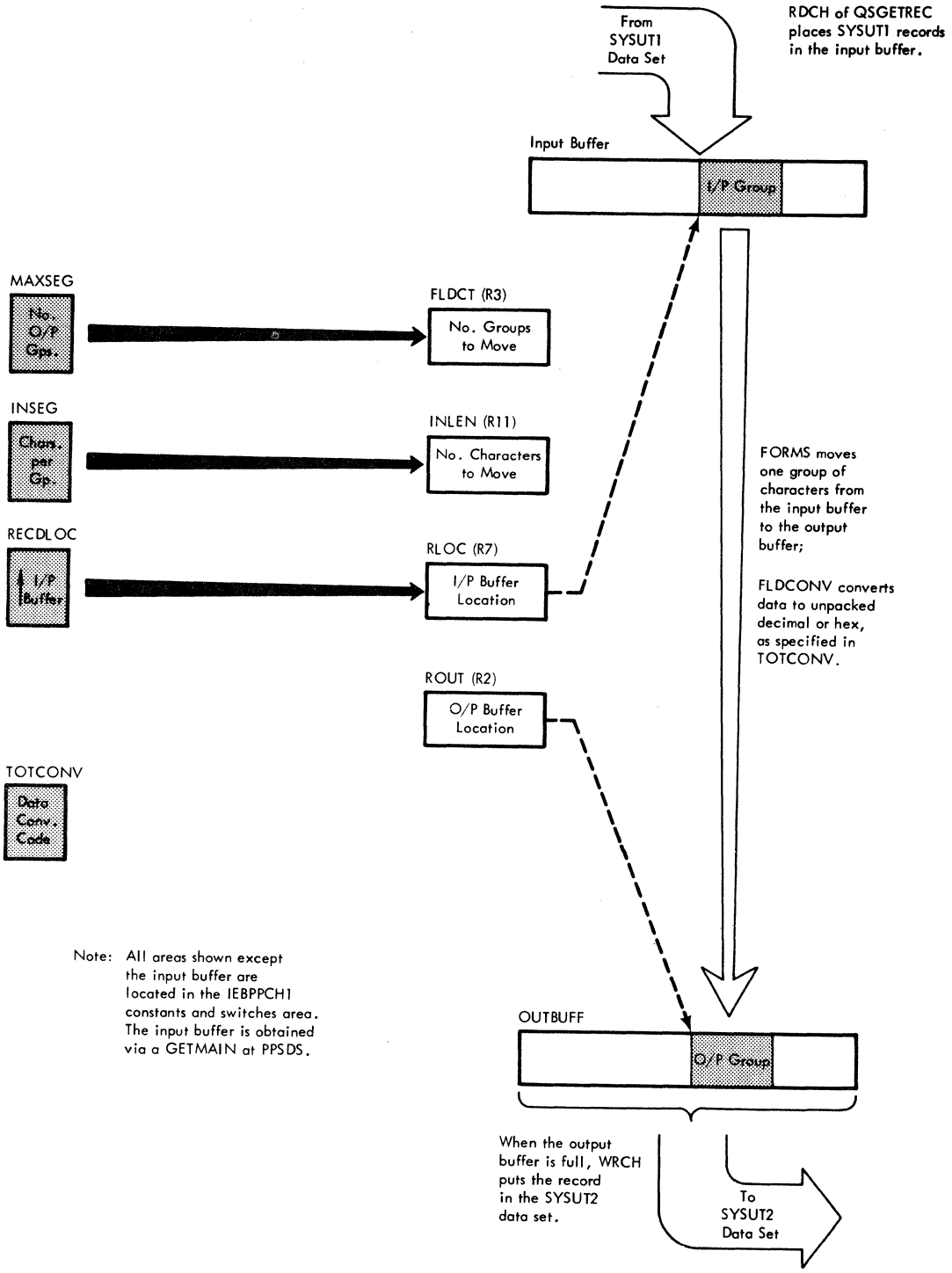


Figure 118. Editing to Standard Format

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

As shown in Figure 119, the routine uses the same registers used by the FORMS routine. However, the registers are initialized from the FCB instead of from specified or default values, and the output buffer pointer in register 2 is the one adjusted to allow for data conversion. In addition, the FORMU routine truncates fields where necessary to avoid moving data from outside the input buffer or storing data outside the output buffer.

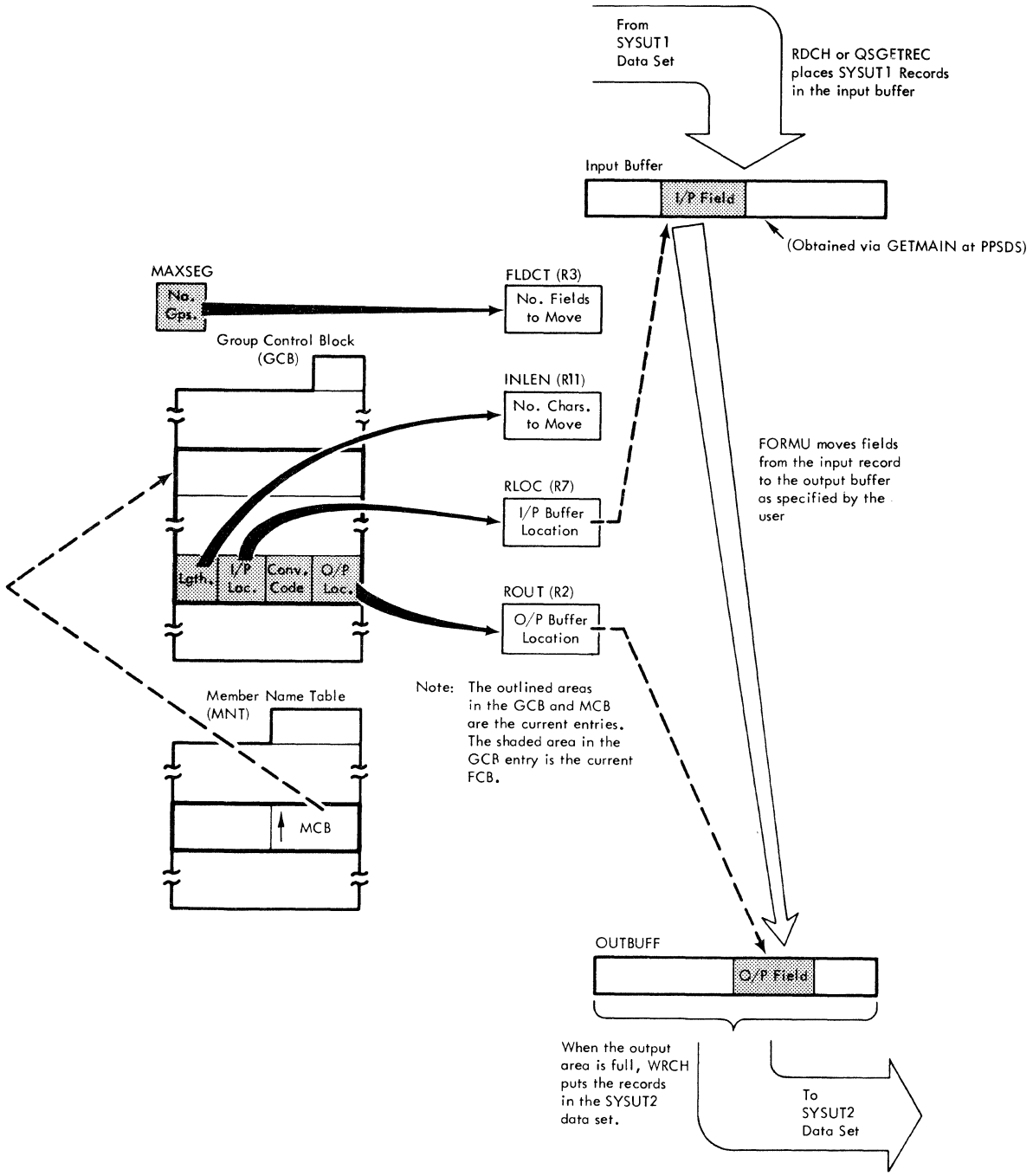


Figure 119. Editing to User Specifications

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

When the FCB has been processed, the main processor UFEND routine passes control to the user's output record exit routine. The WRCH routine stores the control character and (if CDSEQ was specified) the sequence number, and puts the record in the output data set.

HEADERS AND SEQUENCE NUMBERS

The PPHEAD routine of the main processor (assembly module IEBPPCH) places heading information on each page of printed output or in the first cards of punched output. If the user specifies CDSEQ, the WRCH routine places sequence numbers in all subsequent cards. Figure 120 on page 337 shows the data flow involved in producing headers and sequence numbers.

If the output data set (SYSUT2) is to be printed, the PPHEAD routine provides a page number for each page. The user may specify one or two lines of heading information; the title and subtitle are stored in areas of main storage obtained in the control statement analysis routine (assembly module IEBPPAL1), as shown in Figure 111 on page 321.

If the input data set (SYSUT1) is a PDS, the names and aliases of the members to be processed are printed on each page. PPHEAD uses the pointer in DIRADD (in the constants and switches area) to find the names, which are stored in the member area if all members are to be processed, or in the member name table (MNT) if selected members are to be processed.

If the output data set is to be punched, the PPHEAD routine furnishes the sequence number for the first cards. The cards contain the title and subtitle information provided by the user, and the names and aliases of PDS members to be processed. The WRCH routine provides subsequent sequence numbers.

WRITING MESSAGES—(6)

The message data set (SYSPRINT) contains page headers with ascending sequence numbers, and error and warning messages from the IEBPTPCH program. All records in the message data set are written (via the PUT macro instruction) by the message writer subroutine. The message writer (ZZPR) is located in the root segment (in assembly module IEBPPUN1), and is used as a subroutine by all segments of the program. The message writing operation is shown in Figure 121 on page 338.

ZZPR is always entered via a BALR instruction specifying register 14. When it is entered, ZZPR stores registers 14 through 2 in an area called ZZSAVE. The area contains a 1-byte switch, called ZZSW, which is located where the high-order byte of the low-order halfword of register 0 is stored. ZZPR uses the switch to determine whether it is to print a message or a page header:

- If it is to print a page header, the caller places a 512 in register 0, and the switch therefore contains X'02'.
- If ZZPR is to print a message, the caller places the message length in register 0. Because all messages are 32 bytes long, the switch contains zero.

If ZZPR is to print a message, the message address is stored in the register 1 portion of ZZSAVE. ZZPR moves the message from the message module (CSECT IEBPPMSG) to the print area (ZZPRINT), and then issues the PUT macro instruction.

If ZZPR is to print a page header, ZZPR moves the header from PPHDRBL to ZZHDSAVE, prints a blank line to cause page ejection, and then moves the header to ZZPRINT and issues the PUT macro instruction.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

* Note: The areas named INITCDB, DIRADD, TCB1 and TCB2, and the output buffer (OUTBUFF) are located in the IEBPPCH1 constants and switches area.

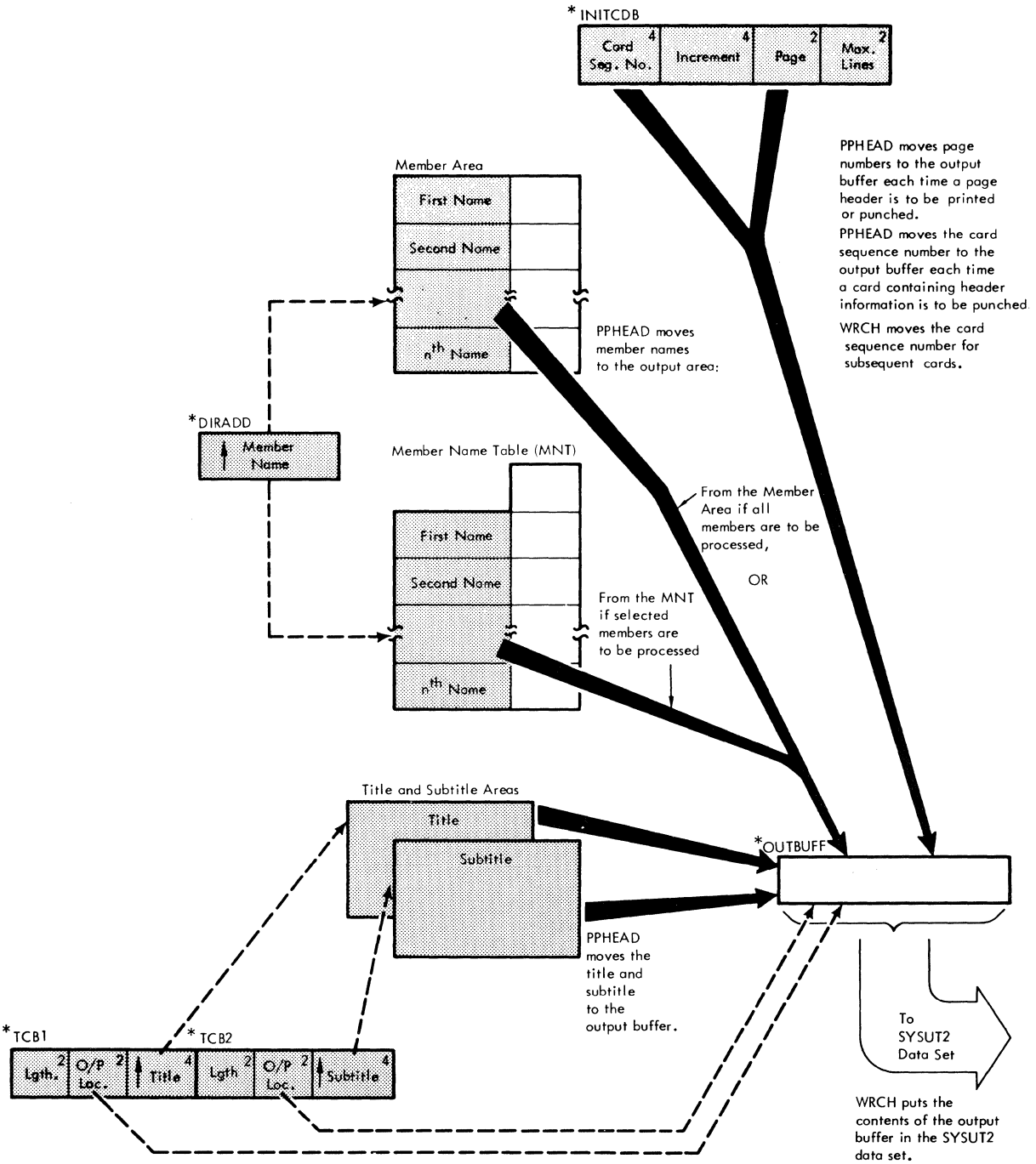


Figure 120. Headers and Sequence Numbers

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

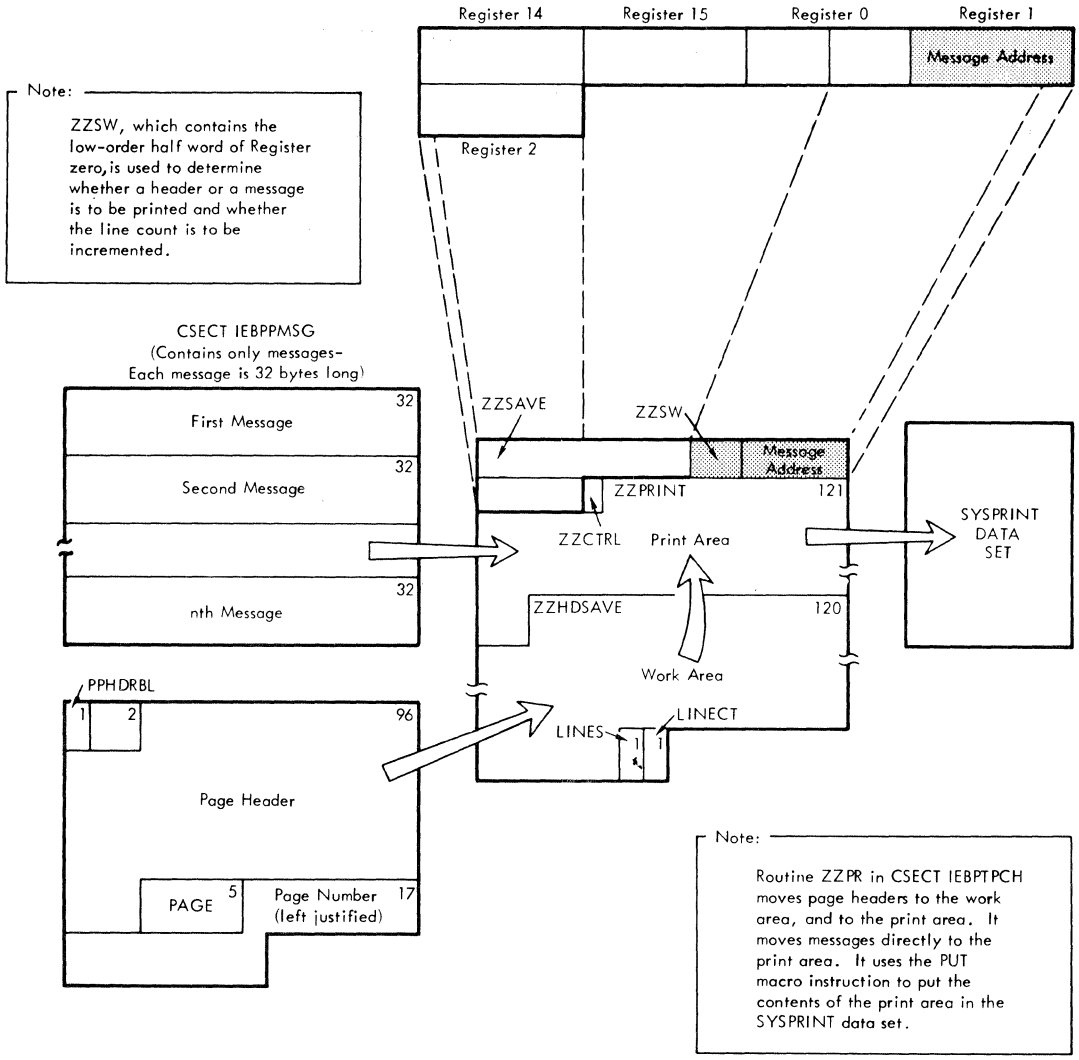


Figure 121. Writing Messages

TABLES AND WORK AREAS

This section describes the formats and contents of the major tables used by the IEBPTPCH program. Tables are drawn 4 or 8 bytes wide; the size of each field is indicated in its upper right corner, and its label is shown in the upper left. The text in the field is descriptive of the contents, but is not necessarily a complete specification. The text that accompanies each figure describes the table and its contents in more detail. Information on areas of main storage not described here or in the text is contained in the appropriate program listing.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

DTABLE as Defined in IEBPPAL1

Offset Dec	Offset Hex				
0	0	DDN1 — SYSUT1 DD Name (8)			
8	8	DDN2 — SYSUT2 DD Name (8)			
16	10	EXLSTADD — Address of Exist List (4)		LITADD — Address of Literal Pool (4)	
24	18	MNTADD — Address of MNT (4)		TYPORG SYSUT1 DSORG (1)	TOTCONV (1) / PPSW1 (1) / PPCTL (1)
32	20	CDNOB (4)		CDINCRB (4)	
40	28	INITPG — First Page No. (2)	PMAXLINE — Max Lines/Page (2)	INITSTOP — STOPAFT Specification (2)	INITSKIP — SKIP = n Param (2)
48	30	TCB 1 — Title Length (2)	O/P Location (2)	Address of Title (4)	
56	38	TCB2 — Subtitle Length (2)	O/P Location (2)	Address of Subtitle (4)	
64	40	PREFMTP — PREFORM Sw. (1) / LBLPRTSW — Label Option (1)	PROCKY — Label Option (1) / LBLCONSW — Label Option (1)	EXLST (32)	
		Names of User Exit Routines			

DTABLE as Defined in IEBPPCH1

0	0	DDN1 — SYSUT1 DD Name			
8	8	DDN2 — SYSUT2 DD Name (8)			
16	10	DEXLSTA — Address of Exit List (4)		DTBLE or DLITA — Address of Literal Pool (4)	
24	18	DMNTA — Address of MNT (4)		DRCB (20)	
32	20	Switch Area			
48	30	DTCB — Title Control Blocks (16)			

Figure 122. DTABLE

Notes to Figure 122:

DTABLE is an area used for communication between the control statement analysis routine and the main processor. The control statement analysis routine (assembly module IEBPPAL1) obtains main storage for DTABLE via a GETMAIN macro instruction, and defines it as shown in the top part of Figure 122 with a DSECT named DTABLE. The main processor (assembly module IEBPPCH1) defines the area as shown in the bottom part of Figure 122 with a DSECT also named DTABLE, and moves the defined fields into its constants and switches area.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

DTABLE contains the following information (field names are those used by IEBPPAL1):

DDN1: This 8-byte field contains the standard or alternate ddname for the SYSUT1 data set.

DDN2: This 8-byte field contains the standard or alternate ddname for the SYSUT2 data set.

EXLSTADD: This 4-byte field contains the address of the field containing the names of the user exit routines.

LITADD: This 4-byte field contains the address of the literal pool.

MNTADD: This 4-byte field contains the address of the member name table (MNT).

TYPORG: This 1-byte field indicates the organization of the input data set:

- If the field contains zero, the data set is sequential.
- If the field contains X'01', the data set is a PDS.

TOTCONV: This 1-byte field specifies the data conversion to be performed:

- If the field contains X'01', data is to be printed or punched in hexadecimal representation.
- If the field contains X'03', data is to be converted from packed decimal to unpacked decimal representation.

Otherwise, data will not be converted unless an overriding RECORD statement is encountered.

PPSW1: This 1-byte field indicates whether a PRINT or PUNCH operation is to be performed:

- If the field contains zero, a print operation is to be performed.
- If the field contains X'01', a punch operation is to be performed.

PPCTL through INITSKIP: These fields contain the binary representations of the parameters whose keywords are shown in the figure.

TCB1: This label refers to an 8-byte area that describes the user-specified title. The area contains the following fields:

- Title Length (2 bytes)—the length of the specified title.
- Output Location (2 bytes)—the offset from the beginning of the output buffer of the first character of the title.
- Address (4 bytes)—The address at which the title is stored (in the literal pool).

TCB2: This label refers to an 8-byte area that describes the user-specified subtitle. It contains fields similar to those in the TCB1 area.

PREFMTP: This 1-byte field contains the user's PREFORM specification:

- If PREFORM was not specified, the field contains zero.
- If PREFORM was specified and the input data set uses ASA control characters, the field contains X'80'.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- If PREFORM was specified and the input data set uses machine control characters, the field contains X'01'.

LBLPRTSW: Reserved.

PROCKY: This 1-byte field corresponds to the DATA parameter in the LABELS statement:

- If DATA=YES, PROCKY=X'80'
- If DATA=NO, PROCKY=X'40'
- If DATA=ALL, PROCKY=X'20'
- If DATA=ONLY, PROCKY=X'10'

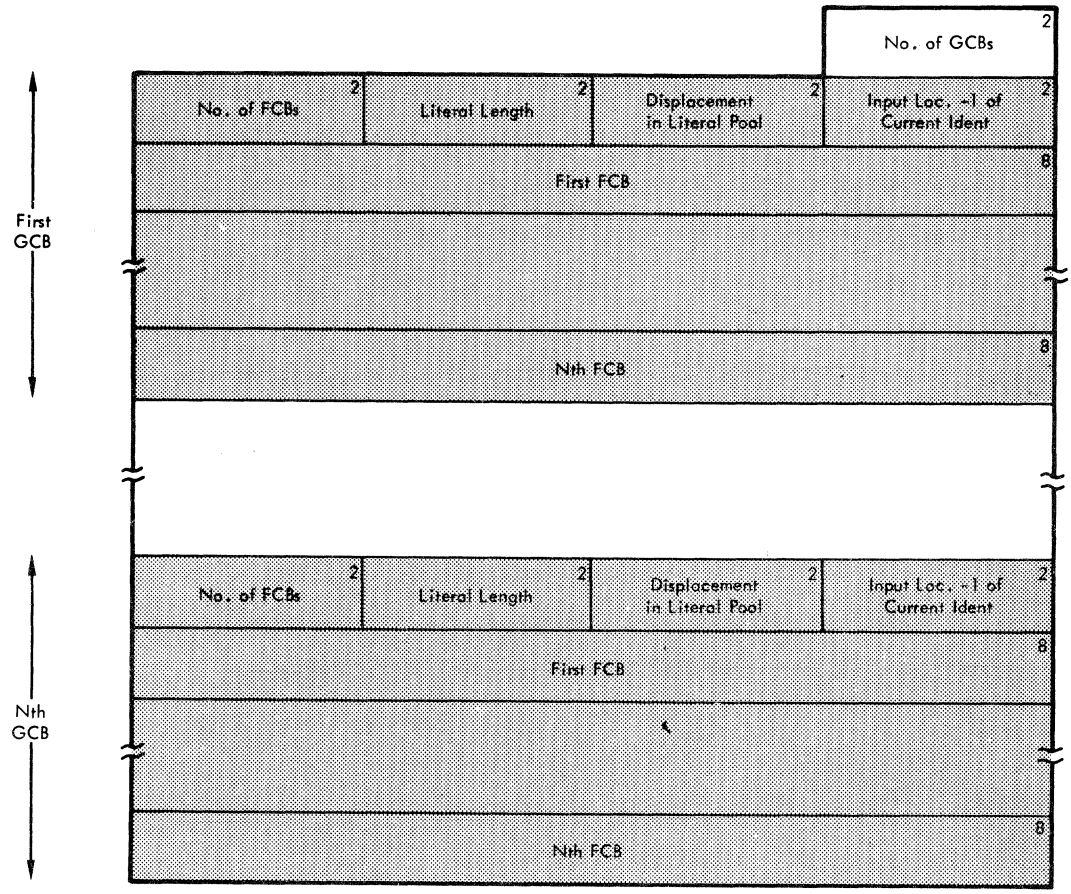
LBLCONSW: Reserved.

EXLST: This 32-byte field contains the names of the user's exit routines. Each name is left-aligned in an 8-byte area, and the names occur in the following order:

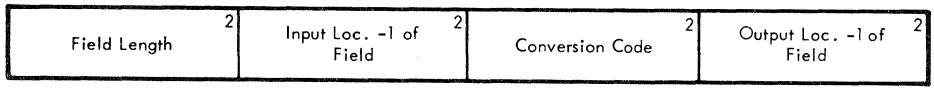
1. EXLIST—Input header label processing routine.
2. EXLIST+8—Input trailer label processing routine.
3. EXLIST+16—Input logical record processing routine.
4. EXLIST+24—Output logical record processing routine.

If a name is omitted, its area is set to zero.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1



Format of Group Control Block (GCB) Area



Format of Typical Field Control Block (FCB)

Note: The Group Control Block (GCB) is also known as the MCB.

Figure 123. Group Control Block and Field Control Block

Notes to Figure 123:

A group control block (GCB) specifies the last record in a record group and contains field control blocks (FCBs) that specify the fields to be processed in each record of the group. The control statement analysis routine (assembly module IEBPPAL1) uses the MAXGPS and MAXFLDS parameters to calculate the size of the area required for GCBs. It uses the GETMAIN macro instruction to obtain the required storage, and then constructs a GCB from each RECORD statement.

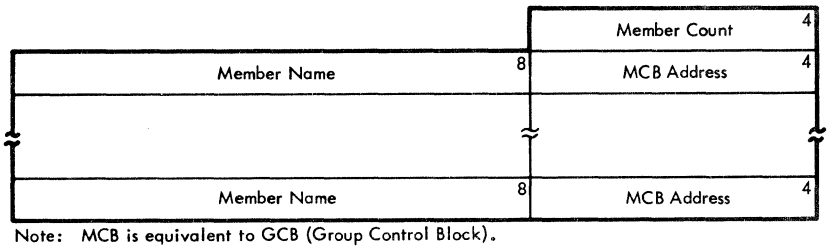
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

The format of the GCB area is shown in the figure. The first 2 bytes of the area indicate the number of GCBs in the area. Each GCB contains the following (2-byte) fields:

- Number of FCBs in the GCB.
- Literal Length—the size of the identification stored in the literal pool.
- The displacement, in the literal pool, of the identification.
- Input Location—the displacement in the input record of the identification field.

In addition, each GCB contains one or more FCBs. Each FCB describes a field to be edited, and contains the following (2-byte) fields:

- Field Length—the length of the field to be edited.
- Input Location minus 1—the displacement of the field from the beginning of the input record.
- Conversion Code—this field is set to X'01' if the user specifies conversion to hexadecimal representation; the field is set to X'03' if the user specifies conversion from packed to unpacked decimal representation.
- Output Location minus 1—the displacement of the field from the beginning of the output record.



Note: MCB is equivalent to GCB (Group Control Block).

Figure 124. Member Name Table (MNT)

Notes to Figure 124:

If selected members of a PDS are to be processed, the member name table (MNT) specifies the names of the members to be processed. If nonstandard editing is specified, the MNT points to the GCB.

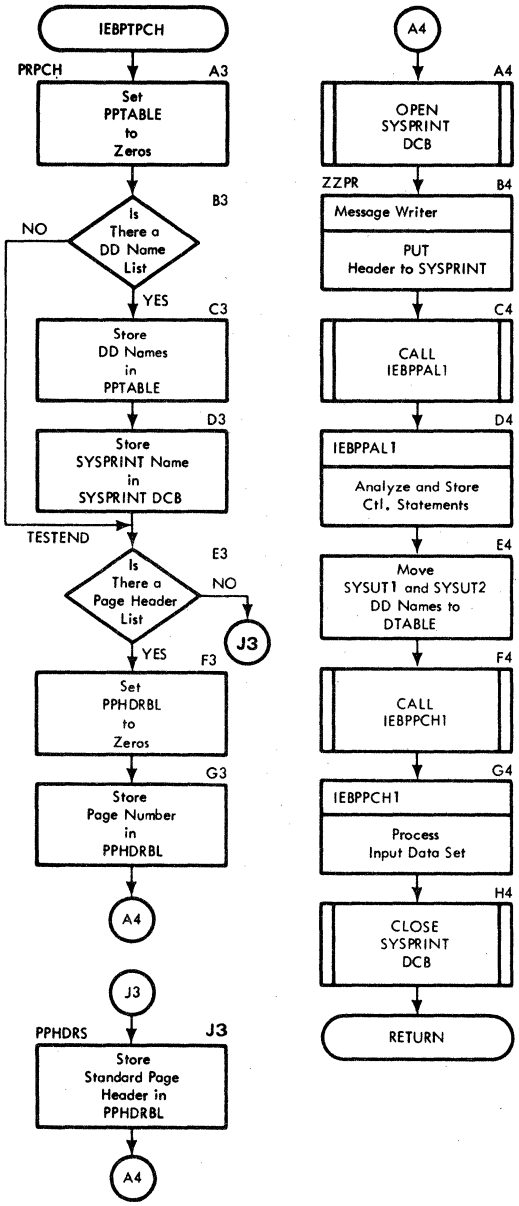
The control statement analysis routine (assembly module IEBPPAL1) uses the MAXNAME parameter to calculate the amount of main storage required for the MNT, and uses the GETMAIN macro instruction to obtain the required storage. It constructs a 12-byte MNT entry from each MEMBER statement, by storing the member name in the first 8 bytes and storing the address of the GCB in the last 4 bytes (the MCB address field) when the GCB is constructed.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEBPTPCH FLOWCHARTS

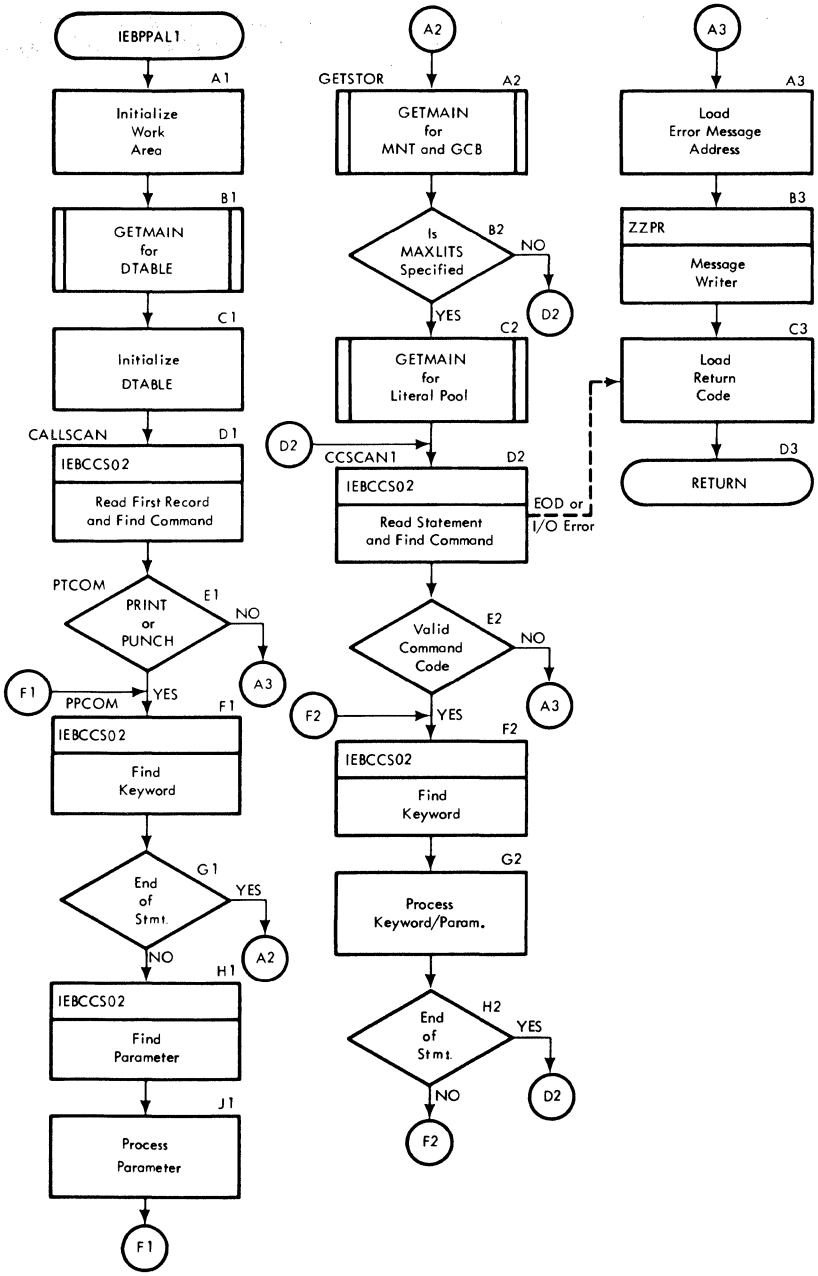
This section contains flowcharts of the major sections of the IEBPTPCH program. The charts are keyed to the program listings and contain program labels where possible.

CHART 01. IEBPPUN1



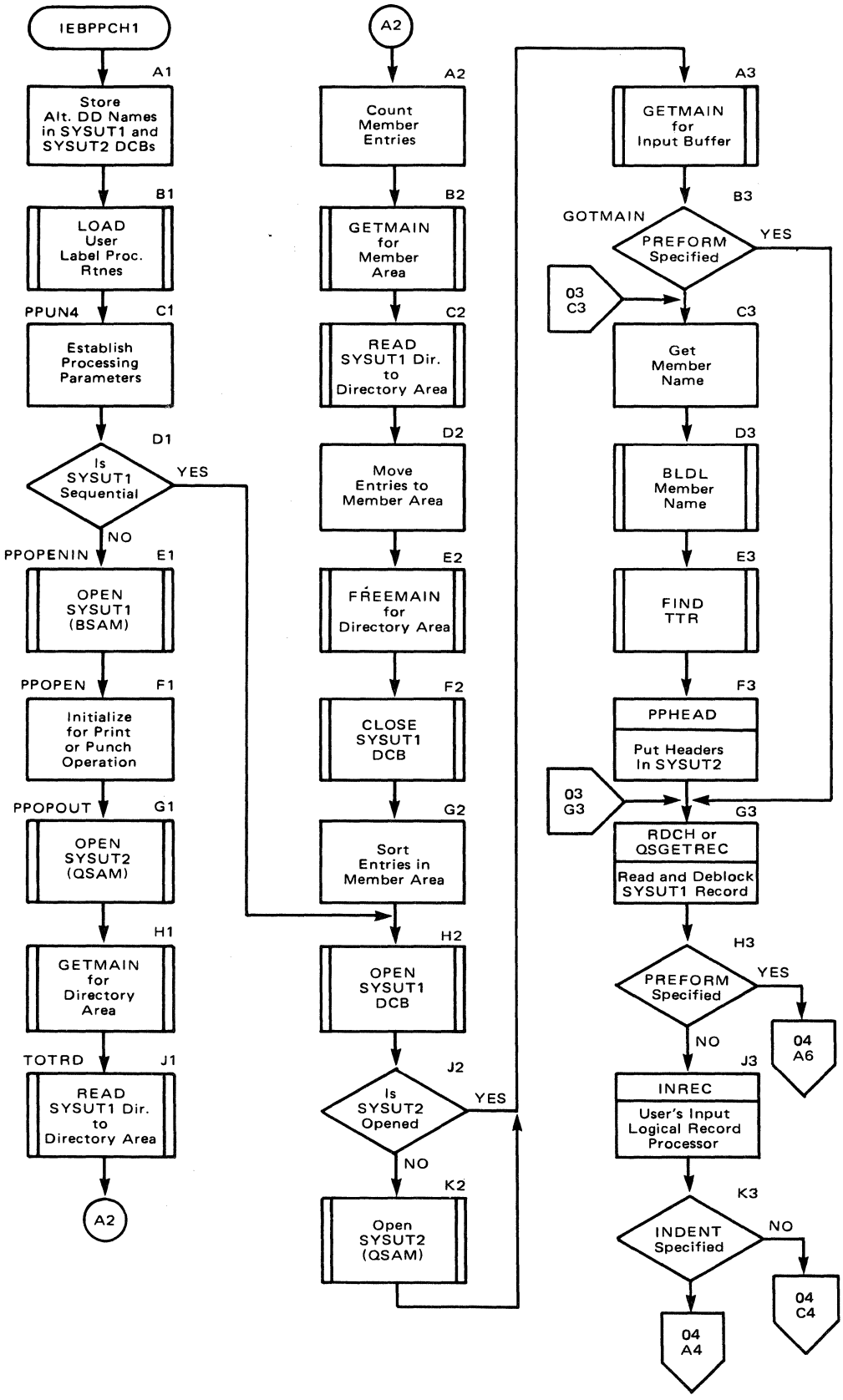
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 02. IEBPPAL1



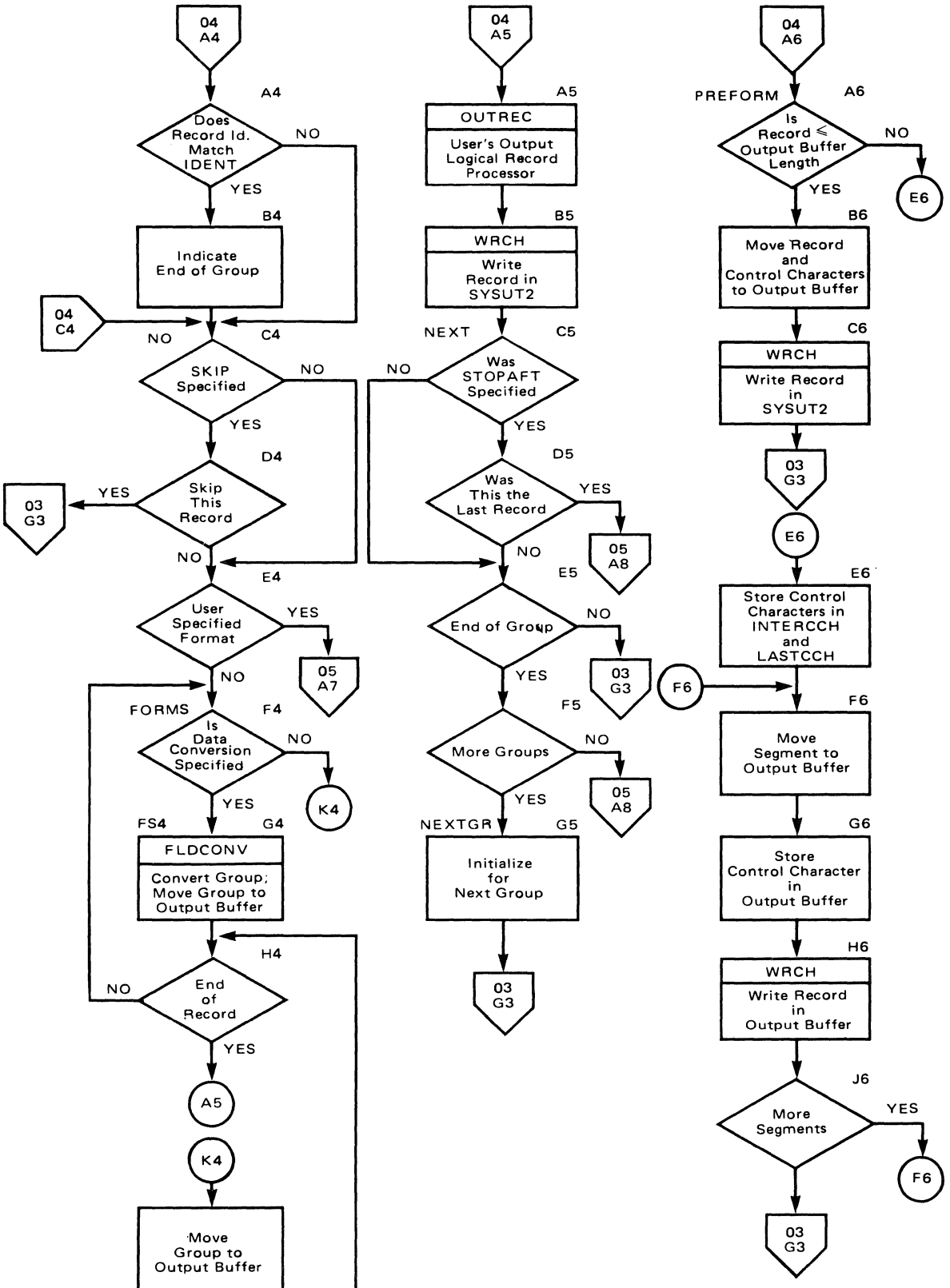
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 03. IEBPPCH1 (PART 1 OF 3)



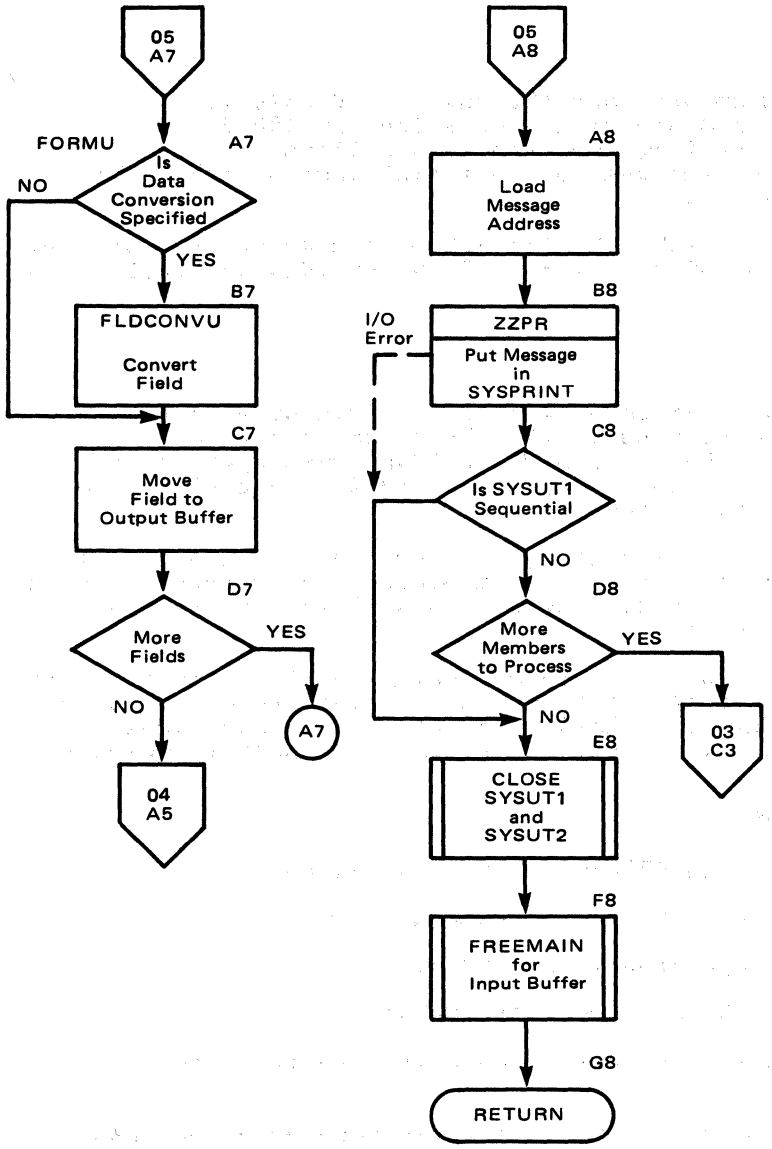
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 04. IEBPPCH1 (PART 2 OF 3)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 05. IEBPPCH1 (PART 3 OF 3)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEBTCRIN PROGRAM

The IEBTCRIN program provides the user of MVS/370 with a means of reading MTDI- or MTST-created data from an IBM 2495 Tape Cartridge Reader, editing the data, and writing the data on any device supported by the queued sequential access method (QSAM) facilities of MVS/370.

The utility can be invoked by the job scheduler or by a problem program through use of the LINK, ATTACH, or LOAD and CALL macro instructions.

PROGRAM STRUCTURE

The program (Figure 125 on page 350) consists of a driver module (IEBTCRIN), a message module (IEBTCR02), and three functional modules (IEBTCR03, IEBTCR04, and IEBTCR05) which perform job file control block (JFCB) analysis, control card analysis, and data read/edit/output processing.

The general operation of each module and the detailed operation of major lower level routines are described on succeeding pages. Charts showing the logical flow of the utility are provided following this narrative.

The narrative description of each module contains references to the code in the form of statement labels (for example, TRCxxx, where xxx is a 3-digit number) associated with each module or routine and to the flowcharts in the form of chart ID numbers.

THE DRIVER MODULE (IEBTCRIN)—CHART 01

The driver module (IEBTCRIN) does the following:

- Processes the user DD name parameter list.
- Loads the message module (IEBTCR02).
- Links to the JFCB analyzer (IEBTCR03).
- Opens the SYSPRINT data set and allocates print area storage.
- Opens the SYSIN data set, if one has been provided, and links to the control card analyzer (IEBTCR04).
- Sets control card defaults if no SYSIN data set was provided.
- Loads the user translate table and exit routines.
- Increments data control block (DCB) defaults by MAXLN.
- Opens the SYSUT1 DCB and issues a mount message.
- Links to the read and edit processor (IEBTCR05).
- Issues SYNAD messages to SYSPRINT for permanent I/O errors.
- Puts out all messages.
- Returns a completion code to the calling task.

Upon entry to this module, register 1 contains the address of a standard parameter list that complies with OS conventions.

The driver module consists of a first level (supervisory) routine and a number of lower level routines.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

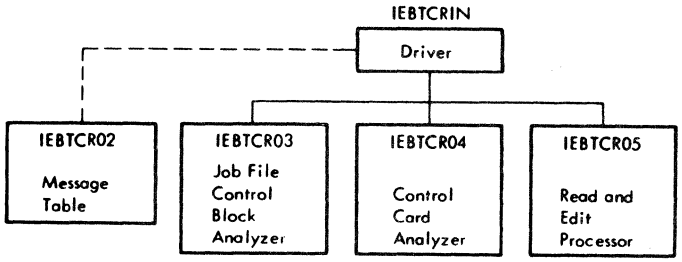


Figure 125. IEBTCRIN Overlay Structure

THE COMMON COMMUNICATION REGION

The driver module contains a common communication region consisting of the following tables, indicators, addresses, and constant data areas that are used by two or more modules:

- Control Card Option Table (Figure 126 on page 351)—Is built by the analyzer module and used by the read/edit module to determine the options that should be used in processing the data.
- Module Address Table (Figure 127 on page 351)—Contains addresses of those modules that are brought into storage by means of a LOAD macro instruction.
- Name Table (Figure 128 on page 351)—Contains DD statement names, user exit routine names, and a user translate table name. If an alternative ddname list is provided by the user at entry, the standard names included in the table will be replaced by the alternate names. The table is used for insertion of names in messages and data control blocks (DCBs).
- Message Pointer Table-Starting at Byte 136 (Figure 129 on page 352)—Contains one entry for each message in the message module. It is used by various program segments to indicate to the display routine the messages to be written and any inserts required. The request code and insert code are set by the program segment requesting that the message be written. The insert location and output location are not altered by the program.
- Request Code:—X'FF' if message is to be written; X'00' if message is not to be written.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

BYTES	0								1								2								3	4 - 5	6 - 7
OPERANDS	EXITS								TYPE			TRANS					EDIT			VER CHK		ERR OPT			REPLACE	MINLN	MAXLN
KEYWORDS	OUTHDR2	OUTTLR2	OUTHDR3	OUTTLR3	OUTREC	ERROR	USERTAB		MTST	MTDI		STDUC	STDLC	USERNAME	NOTRAN		EDITD	EDITR	NOEDIT	VOKCHK	NOCHK	NORMAL	NOERR					
DEFAULT	X'00'								X'40'								X'8C'								X'19'	X'0000'	X'0078'

Figure 126. Control Card Option Table (OPTABLE)

BYTES	8 - 11				12 - 15				16 - 19				20 - 23				24 - 27				28 - 31				32 - 35				36 - 39			
	USER EXIT ROUTINE ADDRESSES																								USER TRANS TABLE ADDRESS				IEBTCR02 MSG MODULE TABLE ADDRESS			
	OUTHDR2				OUTTLR2				OUTHDR3				OUTTLR3				OUTREC				ERROR											
DEFAULT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 127. Module Address Table (ADDRTAB)

BYTES	NAME TABLE							
40 - 47	S	Y	S	I	N			
48 - 55	S	Y	S	P	R	I	N	T
56 - 63	S	Y	S	U	T	1		
64 - 71	S	Y	S	U	T	2		
72 - 79	S	Y	S	U	T	3		
80 - 87	O	U	T	H	D	R	2	
88 - 95	O	U	T	T	L	R	2	
96 - 103	O	U	T	H	D	R	3	
104 - 111	O	U	T	T	L	R	3	
112 - 119	O	U	T	R	E	C		
120 - 127	E	R	R	O	R			
128 - 135	T	R	A	N	S			

Figure 128. Name Table (NAMETAB)

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

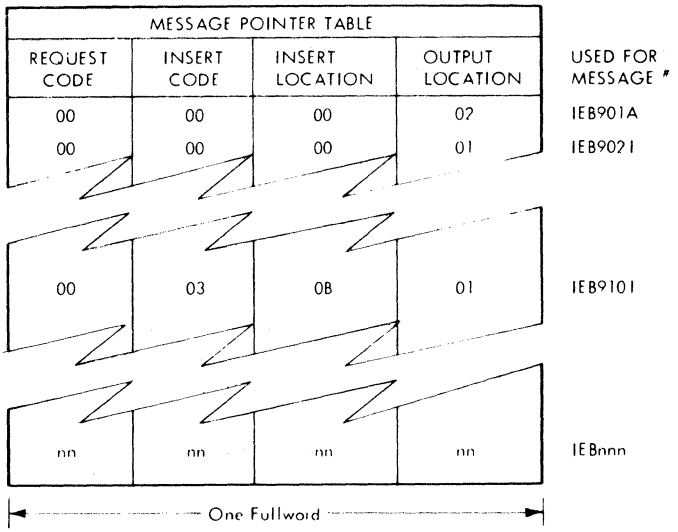


Figure 129. Message Pointer Table (MSGPTRS)

- Insert Code:—References the entries from the common communication region name table (NAMETAB) that are to be inserted into the message.

  X'00' (no insert required)

  X'01' SYSIN

  X'02' SYSPRINT

  X'03' SYSUT1

  X'04' SYSUT2

  X'05' SYSUT3

  X'06' OUTHDR2

  X'07' OUTTLR2

  X'08' OUTHDR3

  X'09' OUTTLR3

  X'0A' OUTREC

  X'0B' ERROR

  X'0C' TRANS

  X'54' SYSUT2 and SYSUT3.

  The message is put out twice, once with SYSUT2 and once with SYSUT3.

- Insert Location: Displacement within the message at which the insert is to be placed (X'00', if none required).

- Output Location:

  X'01' SYSPRINT

  X'02' System console

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

X'03' Both SYSPRINT and console

X'04' SYSPRINT if opened; otherwise, console

Additional communication region components:

1. DCBs
2. DCB exit lists
3. DCB parameter defaults
4. Error indication byte (byte 226)
5. Dummy data set indicators
6. Storage addresses obtained via GETMAIN macro instruction

IEBTCRIN ORGANIZATION

All routines and modules that are given control by the driver module's first level routine must return control to it upon completion of execution and must provide an indication (in the error indication byte located in the common communication region) whether the program should continue or abnormally terminate. Normal termination of the utility is performed by the driver module's first level routine.

If a request for termination is presented to the driver module by a lower level routine or external module, the first level routine gives control to the message writer and terminates processing after return from the message writer. Figure 130 on page 354 shows the structural flow of the driver module.

The driver module's first level routine gives control to each of the lower level routines described in the following text and also to all external modules (IEBTCR02, IEBTCR03, IEBTCR04, and IEBTCR05):

- Parameter List Processor routine (TCR120)—Processes the optional ddname list that is supplied when the utility is invoked by a problem program. It stores each alternative name in the name table (NAMETAB) (Figure 128 on page 351) in the common communication region which is used in message construction and data control block (DCB) processing.
- Message Module Loader routine (TCR130)—Loads the message module (IEBTCR02).
- SYSPRINT Open and Storage Allocation routine (TCR140)—Opens the SYSPRINT data set and allocates storage for a work area if the SYSPRINT DD card was provided.
- Control Card Analyzer Interface routine (TCR150)—Checks for the presence of the SYSIN DD statement. If the statement is present, the routine opens the SYSIN data set, calls the message writer routine to write the card images, links to the control card analyzer (IEBTCR04), and, upon return, closes the SYSIN data set and returns to the first level routine of the driver module.

  If the SYSIN DD statement is not present, the routine sets default options and returns to the first level routine.
- User Routine Loader routine (TCR160)—Loads any user exit routines and a translate table module that are specified in utility control statements and stores the corresponding entry point addresses in the address table (ADDRTAB) (Figure 127 on page 351) in the common communication region.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

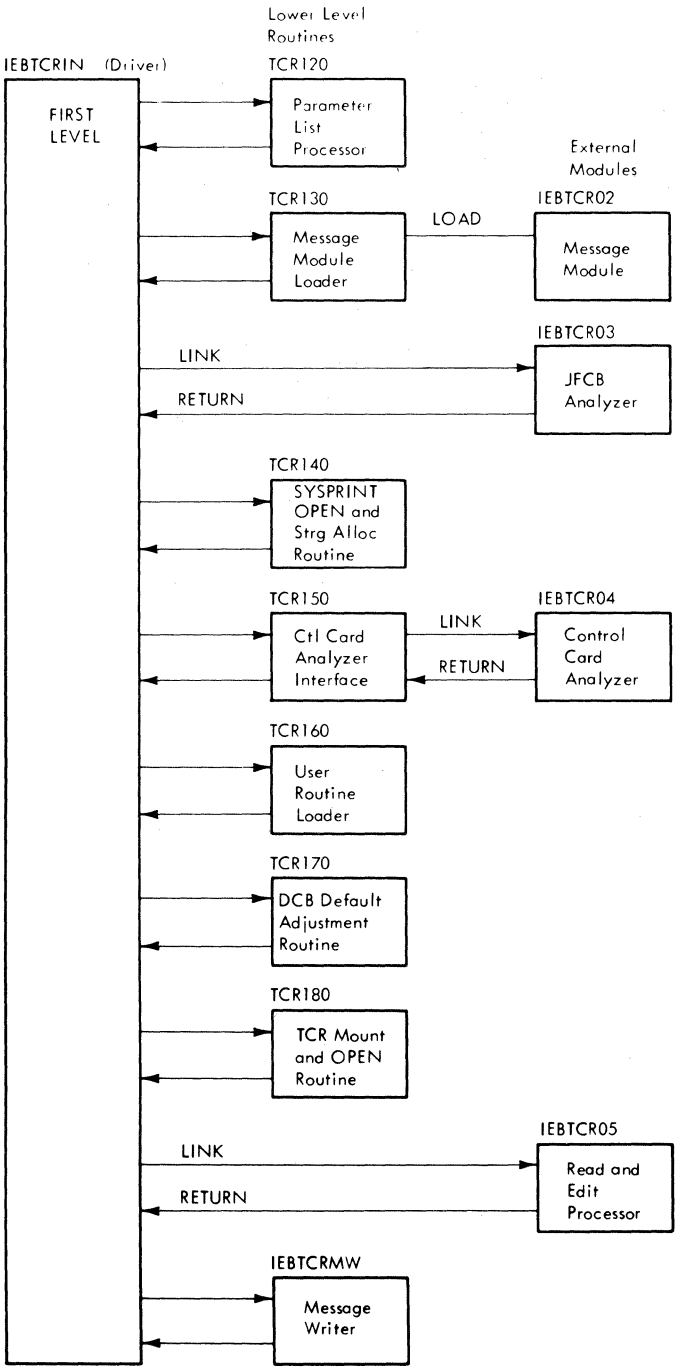


Figure 130. Structural Flow of the Driver

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- DCB Default Adjustment routine (TCR170)—Increments the basic defaults established by the JFCB analyzer by the MAXLN value. If MTDI with editing was specified, and the user omitted the LRECL and/or BLKSIZE DCB parameters of the SYSUT2 and/or SYSUT3 DD statements, then the SYSUT3 LRECL and BLKSIZE values are further increased by 4 to accommodate the 4-byte error description word.

- ICR Cartridge Mounting routine (TCR180)—Opens the SYSUT1 data set, sets up a default buffer length of 2000 bytes if the BUFL parameter was not specified by the user, and constructs and issues the mount message using device, volume, and job identification extracted by the JFCB analyzer.

- SYSUT1 SYNAD routine (TRC190)—Constructs and issues SYNAD messages for permanent I/O errors on SYSUT1.

- Message Writer routine (IEBTCRMW)—Writes all messages for the utility. It consists of two processing functions:

  1. Issuing of all IEB9xxx messages requested by means of the message pointer table (Figure 129 on page 352) in the common communication region

  2. Writing (to SYSPRINT) of control statement images, page headings, SYNAD messages, etc.

  When option 1 is requested, the message writer entry routine checks the request code of each entry of the message pointer table to determine which messages have been requested.

  When it finds one, it sets a pointer to the message text in the message module, inserts any ddname or user routine name that may be required, determines the output location (SYSPRINT or console), and writes the message.

  It sets the request code to zero and proceeds to the next table entry. When all messages have been printed, it returns control to the calling routine.

  If option 2 is requested, the message writer entry routine loads registers with the message address (MSGAD) and message length (MSGLN) supplied via the common communication region and passes control to the SYSPRINT routine (TCR310).

THE MESSAGE MODULE (IEBTCR02)

The message module contains two tables (Figure 131 on page 356):

- A pointer table containing fixed-length entries, each of which points to an entry of the text table

- A message text table containing variable-length entries

THE JOB FILE CONTROL BLOCK ANALYZER MODULE (IEBTCR03)

The job file control block analyzer module (IEBTCR03) does the following:

- Inserts alternative ddnames in DCBs

- Checks the task input/output table (TIOT) for missing DD statements

- Extracts device, volume, and job identification from system control blocks

- Analyzes job file control blocks to determine missing DCB parameters

- Assigns defaults to missing parameters

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

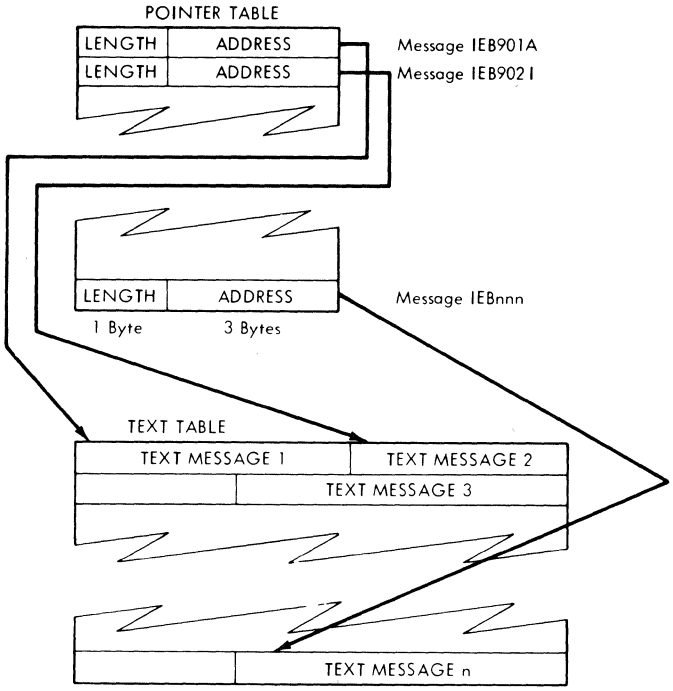


Figure 131. Message Module Tables

The module consists of the IEBTCR03 and TCR210 routines.

Upon entry to this module, register 1 contains the address of a parameter list that contains one entry—a pointer to the common communication region that is located in the driver module.

The analyzer module uses the TIOT and the unit control block (UCB) to determine the device allocated to the task for the SYSUT1 (TCR input) data set. It extracts from the JFCB any external identification associated with the data set by means of the VOLUME=SER=xxxxxx DD statement parameter. It removes job and step identification from the TIOT. All this information is stored in the common communication region for later use by the driver mount routine (TCR180).

If the analyzer module detects a missing DD statement (other than for the SYSUT1 data set), it sets a DUMMY file indicator for this data set, sets an appropriate message indicator, and assigns a return code of 4. If the data set is SYSUT1, the analyzer sets message indicators, establishes a return code of 12, and returns control to the driver module with a request for termination.

If the analyzer module determines that DCB parameters were omitted, it develops complete or partial defaults, depending upon the data set involved. For SYSIN and SYSPRINT data sets, complete defaults are developed as indicated in Figure 132 on page 357 and Figure 133 on page 357.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

DDNAME	LRECL	BLKSIZE	RECFM
SYSIN	80	80	FB
SYSPRINT	121	121	FB A
SYSUT2	"MAXLN+4"	"MAXLN+8"	VB
SYSUT3	"MAXLN+8"	"MAXLN+12"	VB

Figure 132. DCB Parameter Defaults (MTDI with Editing)

DDNAME	LRECL	BLKSIZE	RECFM
SYSIN	80	80	FB
SYSPRINT	121	121	FB A
SYSUT2	"MAXLN+4"	"MAXLN+8"	VB
SYSUT3	"MAXLN+4"	"MAXLN+8"	VB

Figure 133. DCB Parameter Defaults (MTST and MIDI without Editing)

The driver DCB OPEN exit routine (TCR720) inserts the defaults when a data set opens.

If the user includes some parameters (for example, REFCM) but omits others, the JFCB analyzer attempts to set defaults that are consistent with the parameters that have been supplied. For example, if the user supplies the SYSPRINT parameter RECFM=VB, the JFCB analyzer will set the BLKSIZE default to 129. If the user provides a parameter that is not consistent with defaults established for missing keywords, the JFCB analyzer will issue a request for termination through the error indication byte.

THE CONTROL CARD ANALYZER MODULE (IEBTCR04)—CHART 03

The control card analyzer module (IEBTCR04) does the following:

- Reads utility control statements from the SYSIN data set
- Scans the control cards for format errors
- Validates the name, operation, and operand fields of each statement
- Sets options including defaults for missing parameters
- Tests the consistency of the options that have just been set

Upon entry to this module, register 1 contains the address of a parameter list that contains one entry—a pointer to the common communication region that is located in the driver module.

The control card analyzer module processes each control statement a character at a time. It assembles each operator, keyword, or parameter in an 8-position hold area.

When a complete word has been assembled, the control card analyzer module performs one of the following operations, depending upon the class in which the word falls:

- Name field—Checks that the first character is alphabetic.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- Operator—Compares the word with each operator identifier (TCRGEN or EXITS). If a match is found, pointers are set to each of the tables corresponding to this operator.

- Keyword—Compares the word with each keyword entry of the selected operator table. If a match is found, pointers are set to the list of parameters corresponding to this keyword.

- Parameter—Compares the word with each entry of the parameter list associated with the previously identified operator and keyword. If a match is found, the corresponding data table entries are used to set the proper option bytes.

The control card analyzer module consists of the first level (supervisory) routine and a number of lower level routines that perform various functions.

The structural flow of the control card analyzer module is shown in Figure 134 on page 359. With the exception of the message writer, which is part of the driver module, each block represents a routine that is internal to the control card analyzer module.

When all statements have been processed, the control card analyzer sets defaults for those options that were not specified, and tests the consistency of all applicable options. If any conflicting options are detected, the control card analyzer sets the return code to 12 and requests termination.

The first level routine of the control card analyzer module gives control to each of the lower level routines. If a request for termination is presented to the first level routine by a lower level routine, the first level routine gives control to the message writer and returns control to the driver module after the return from the message writer.

The following text describes the lower level routines and their functions:

- The New Statement Fetch routine (TCR230)—Calls the SYSIN data set read routine (TCR710) to read a new card and initializes hold areas and pointers.

- The Name Validation routine (TCR240)—Checks for the presence of a name field and, if one is present, checks that it is eight characters or less in length. It also checks that the first character is alphabetic.

- The Operator Validation routine (TCR250)—Checks that the operator is either TCRGEN or EXITS. If it is one of these, the routine verifies that the same operator has not appeared previously.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

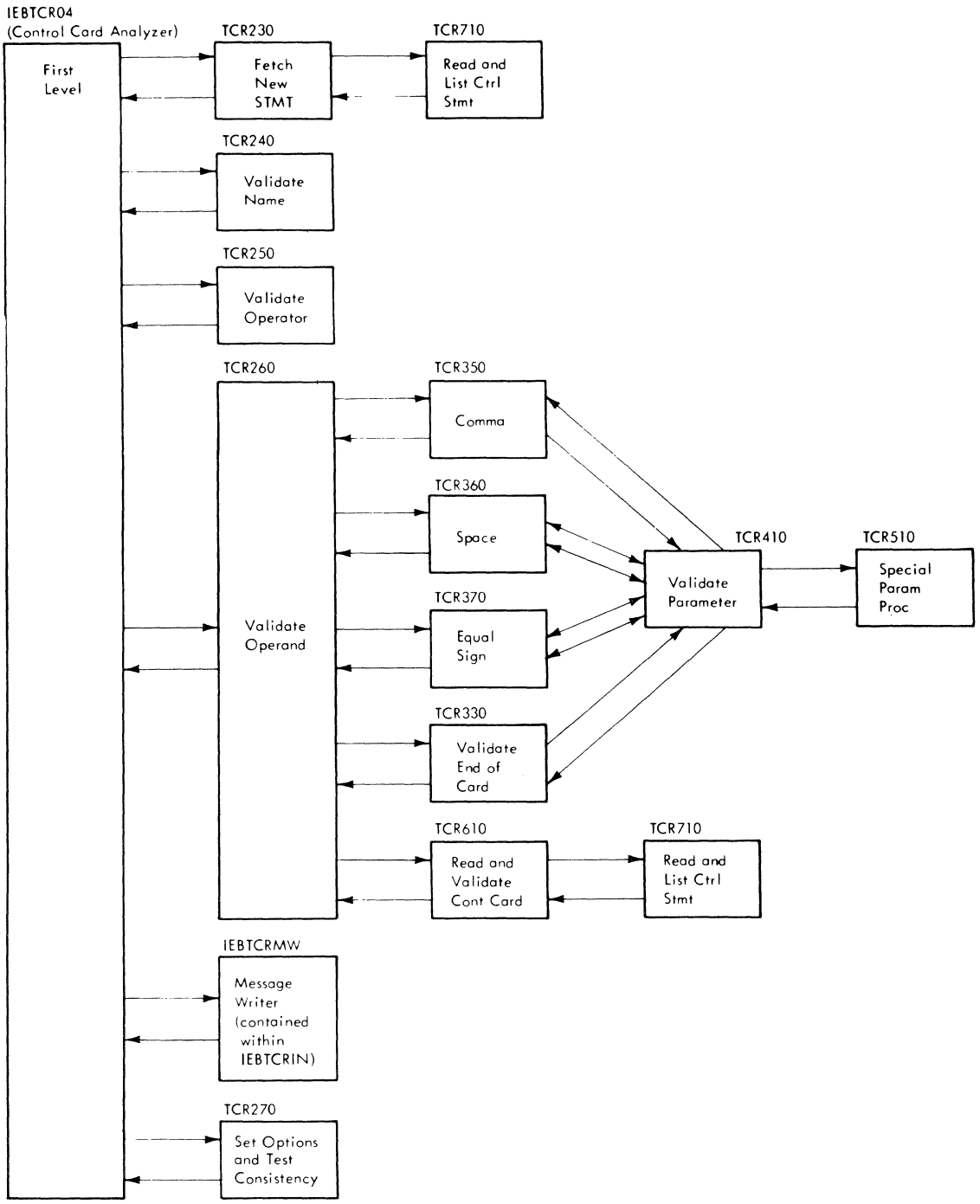


Figure 134. Structural Flow of the Control Card Analyzer

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- The Operand Validation routine (TCR260)—Assembles each keyword or parameter in the 8-position hold area. It tests each position of the input card for a valid delimiter (Figure 135).

Delimiter	Meaning	Action
Space	End of Statement	Fetch new Statement
Comma	End of Parameter	Call Comma Routine
Equal Sign	End of Keyword	Call Equal Sign Routine
Parenthesis		Bypass

Figure 135. Control Card Delimiters

If a delimiter is not present, the routine tests whether or not eight characters have previously been assembled for this word. If not, it transfers this character to the hold area, increments to the next character, and repeats the process.

- The Set Options and Test Consistency routine (TCR270)—Uses the indicators that have been set in a temporary option table to set options in a permanent table. It sets defaults for keywords that were omitted. If any inconsistency appears, it sets message and abnormal termination indicators and returns control to the first level routine of the control card analyzer. The control card analyzer then sets the return code and returns control to the driver routine with a request for termination.

- The End of Card Validation routine (TCR330)—Tests for the valid end of statement or valid end of card.

- The Comma routine (TCR350)—Calls the parameter validation routine (TCR410) if the comma was preceded by a parameter.

- The Space routine (TCR360)—Calls the parameter validation routine if the last parameter was not previously validated. When the routine is completed, the space routine sets the next input character pointer to column 72 to bypass any intervening comments.

- The Equal Sign routine (TCR370)—Sets pointers to both tables (TCRGEN and EXITS) associated with the previously identified operator. It steps through both tables, searching for a matching keyword entry. If a match is found, it sets both table pointers to this entry, reinitializes the hold area pointer, and returns to the first level routine of the control card analyzer. If a match is not found, the appropriate message indicators are set, the remaining characters of the parameter are flushed, and control is returned to the first level routine.

- The Parameter Validation routine (TCR410)—Checks the parameter table for either a matching parameter or an indication of a special validation routine. If a match is found, it sets the option in the temporary option table work area and returns to the first level routine. If a special validation routine is indicated, it branches to one of the special routines (TCR510, TCR520, or TCR530), and then returns to the first level routine. If no routine is present and no match is found, it sets the invalid parameter message indicator and returns to the first level routine.

- The Special Processing routines (TCR510, TCR520, and TCR530)—Validates the user translate table name and the MAXLN, MINLN, and REPLACE parameters.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- The Continuation Card Validation routine (TCR610)—Calls the SYSIN data set read routine (TCR710) to read a card. Then it determines whether columns 1 through 3 of the new card are blank. If comments were not indicated by the previous card, it also verifies that columns 4 through 16 are not blank. If comments were indicated by the previous card, scanning continues at column 72.

THE READ/EDIT MODULE (IEBTCR05)—CHARTS 04 THROUGH 13

The read/edit module reads input data from the IBM 2495 Tape Cartridge Reader, edits or translates it if requested, moves it to an output work area, and puts it out (using QSAM) to one of two data sets if requested. Upon entry to this module, register 1 contains the address of a parameter list that contains one entry, a pointer to the common communication region that is located in the driver module.

The read/edit module consists of five routines:

- Entry routine
- Read routine
- Scan routine
- Move routine
- Output processor routine

The structural flow of the read/edit module is shown in Figure 136 on page 362.

The following text describes the lower level routines and their functions:

- The Entry routine (IEBTCR05 Chart 04)—is executed only once, upon entry to the read/edit module, and does the following:

  1. Establishes base registers.
  2. Completes the DCB exit lists for header and trailer exits.
  3. Gets storage for two input buffers (using BUFL from SYSUT1) and the record construction work area (using the largest value among MAXLN, LRECL on SYSUT2, or LRECL on SYSUT3, plus adding 8 to the largest of these).
  4. Initializes the work area pointers (TCR930).
  5. Determines the translate and test table, TRTMTDI or TRTMTST, to be used if requested.
  6. Builds the CCWs that make up the channel program for the Read routine.
  7. Sets all the NOP switches according to the contents of the OPTABLE (utility control statement options).

- The Read routine (TSW100) Charts 05, 06—Issues the EXCP macro instruction to read data from the Tape Cartridge Reader. Control is passed to this routine whenever the current input buffer is emptied. If the input buffer was emptied properly, control is passed to a WAIT macro instruction (TCR103) to wait for the completion of the overlapped I/O operation (read or rewind).

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

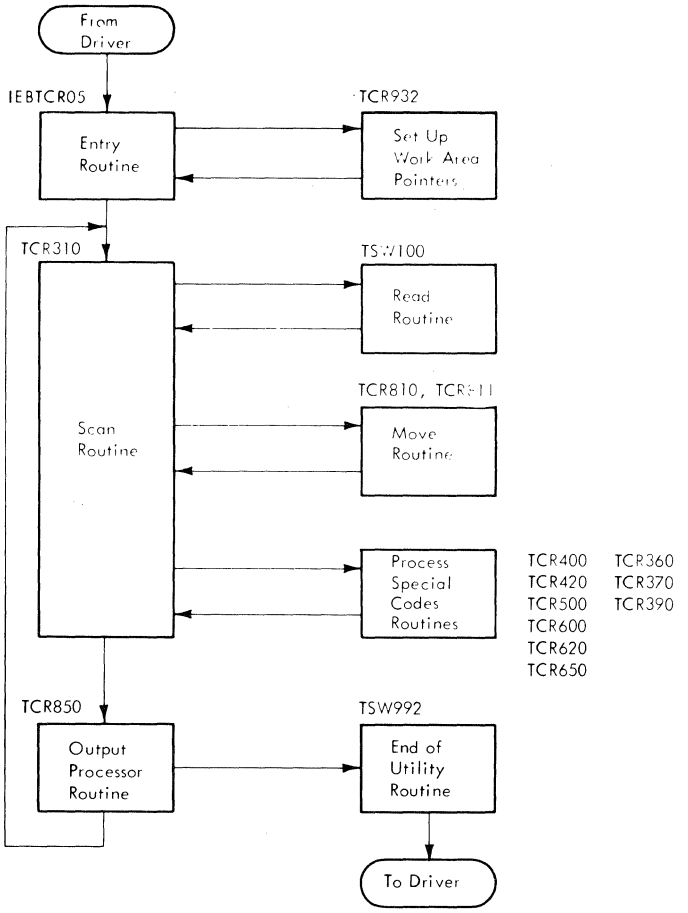


Figure 136. Structural Flow of Read/Edit Module

Upon completion of the I/O operation, a check is made for one of the following conditions listed with the action taken:

- Permanent error or intercept condition—An error switch is set in the error indication byte in the common communication region and the utility is terminated.
- Unit Exception—Because the error recovery procedure flags this condition as a permanent error, the I/O flags in the IOB and in the DCB are reset (TCR107). If the unit exception was posted for a read command, one of two actions is taken:

  1. If a group separator (GS) code on an MTDI tape caused the unit exception, the GS (X'1D') code is deleted for EDIT=EDITR or EDIT=EDITD and retained for EDIT=NOEDIT.

  2. If an end data (ED) stop code (X'19') for MTDI or a stop (st or ST) code (st equals X'1D' and ST equals X'19' for MTST) caused the unit exception (end-of-tape condition), a rewind and unload command is issued (TCR150), after which the utility waits for completion of this I/O operation.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

If the unit exception was posted for a rewind and unload command (end-of-file condition), control is passed to the output processor routine to put out the final record in the output work area. Then the utility is terminated normally.

- Data Check—The I/O flags in the IOB and in the DCB are reset (TCR104) because the standard error recovery procedure flags this condition as a permanent error; the REPLACE character is stored in the data check location; a data check switch is set (TCR128); and a double data check switch is set if two successive data checks were encountered. The data check switch is turned off by the output processor routine after the record is processed. Normal processing then continues.

- Normal Interrupt—Normal processing continues. Normal processing (TCR120) consists of clearing the "complete" bit in the ECB, determining the number of bytes read, swapping CCW addresses in the IOB (to point at the other input buffer), and issuing another read command.

  For the first read command issued to an MTDI tape (TSW106), the program level length indicators are reset to zero, and the DUP switch is set to prevent duplicating.

- The Scan routine (TCR310) Charts 07 and 08—Are used to detect and process special characters in the input buffer, depending upon whether MTDI with editing, MTST with translation, or neither is specified.

- MTDI with Editing—The input buffer is scanned, using the translate and test table (TRTMTDI) to detect codes that require special action. A branch table is utilized to transfer control to the routine that will perform the required action. The codes that require special action are:

  - A cancel code (CAN) detected (TCR620) in a normal start-of-record (SOR) location causes a delete switch to be turned on. A cancel code detected at other than start-of-record positions causes all bytes up to the code to be sent to the output routine and an end-of-record (EOR) to be forced. The cancel code will then be rescanned and treated as in a start-of-record location.

    In either case, all the ensuing bytes will be deleted until a valid EOR or SOR is detected. All codes requiring action that are detected within a canceled record are ignored. The delete switch is turned off whenever a valid end of record or end of tape (EOT) is detected.

  - A left zero start (LZS) code indicates the start of a left zero field and the left zero start position is saved (TCR400) for later field justification.

  - Left zero fill (LZF) codes are summed (TCR420) to calculate the left zero field length. When the end of the left zero field is detected, the digits in the field are right adjusted within the field and leading zeros are inserted to complete the field.

  - Replace characters are inserted for the LZF codes prior to completing the left zero adjustments. If adjustment fails, the replace characters remain.

  - A duplication (DUP) code causes (TCR500) the corresponding byte from the previous record to be entered into the present record if duplication is allowed. The REPLACE character is inserted into the record when duplication is not allowed.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- Program level (P1-P8) codes cause (TCR600) the SOR switch to be set on, and a check to be made for a valid EOR on the previous record. If the SOR switch (TSW322) is on when the SCAN routine is entered, the current byte is checked to see if it is a valid SOR byte.

- Record mark (RM) or verify OK (VOK) codes cause (TCR650) the SOR switch to be set on. When verification (VERCHK-VOKCHK) is expected (TSW658), the record will be considered to be in error if a VOK code is not detected.

- If no special codes are detected by the translate and test (TRT) instruction, the Scan routine (TCR360) passes the scanned bytes to the output routine via the Move routine.

If MTDI with editing is specified and an error is detected, the error description word is properly updated.

- MTST with Translation—The buffer is scanned, with the translate and test table (TRTMTST) to detect codes that require special action before translation can take place. A branch table is utilized to transfer control to the routine that will perform the required action. Codes that require special action are:

  1. Feed codes (FD and fd), switch codes (SW and sw), and autosearch codes (AS and as) are deleted (TCR390) from the input buffer.

  2. Reference codes (which consist of 15 autosearch codes, 15 search codes; 30 parity-bit-only codes, which are identical to lowercase "z" and a carrier return code) are reduced to a single search code (SRC). This is accomplished (TCR370) by deleting all autosearch codes, defining a reference code only when a search code is found, and deleting all codes up to and including the carrier return code that defines the end of the reference code.

  If no special codes are detected by the TRT instruction, the scan routine (TCR360) passes the scanned bytes to the output routines via the move routine.

- MTST with NOTRAN or MTDI with NOEDIT—The scan routine (TCR360) merely passes all data from the input buffer to the output routines via the move routine.

- The Move routine (TCR810, TCR811) Chart 09—Is entered any time a special character or the end of the scan area is detected. The move routine moves bytes from the input buffer to the output work area and adjusts the appropriate pointers.

  If an EOR was forced, control passes directly to the output processor routine. If EOR was not forced, the number of bytes to move and the number of bytes to put out are calculated. If there are bytes to move and enough space is available, the bytes are moved.

  If there is insufficient space to receive all bytes, only the number of bytes that can be accepted are moved. The remaining bytes are moved to the output work area during the next entry to the move routine. Whether or not there are bytes to put out, if SOR is not expected, control returns to the caller.

  If there are bytes to put out and SOR is expected, control is passed to the output processor routine.

  If MTDI with editing is specified and the record to be written has an incorrect length or contains a data check, the error description word is updated before the output processor routine is called.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

If the move routine is entered at TCR810, it always returns control to the scan routine. If the move routine is entered at TCR811, it returns control to the calling routine.

- The Output Processor routine (TCR850) Charts 10, 11, and 12—are called by the move routine any time the output work area becomes full or an EOR is detected in the output work area. If MTDI with editing is specified, the current record length is compared (TCR856) with the record length for the first valid record processed (from the current cartridge) that has the same program level code as the current record (TCR856).

  If they are not equal, a type status error is recorded unless the current record is the first valid record on the cartridge with the current program level code (in which case, the length is saved for future comparisons). This routine also sets up the parameter lists and branches to the ERROR (TSW862) and OUTREC (TSW880) Exit routines, if requested.

  Figure 137 on page 366 describes the actions taken by the output processor routine, depending upon the status of the record, the user exits specified, the user return codes from these exits, and the ERROPT option.

The SYSUT2 and SYSUT3 data sets are written using the standard QSAM PUT (move mode) macro instruction (TCR922) with DCB exits for permanent errors and header/trailer processing. These data sets are not opened (TCR903) until the first record is to be sent out to the data set.

At open time, if LRECL from an existing data set label for SYSUT2 or SYSUT3 is larger than the present output work area, a new work area must be obtained via a GETMAIN macro instruction and the old work area released via a FREEMAIN macro instruction (TCR923).

If MTST with translate is specified, the current record will be translated just prior to being written (TCR940). If MTDI with editing is specified, the error description word (Figure 138 on page 367) is appended to SYSUT3 records (TSW912).

For variable-length output (TCR917), a 4-byte record descriptor word is computed and appended to the front of the record (SYSUT2 and/or SYSUT3). For fixed-length records, the remainder of the output work area (that which exceeds the logical EOR) is blanked out (TCR936).

Upon completion of output (TSW927), work area pointers and program switches are reset. For MTDI with EDIT, control returns to the caller of the move routine. Otherwise, control is returned to the move routine to move any remaining bytes from the scanned area.

The read/edit module provides six user exits:

- The ERROR (TSW862) exit is taken after editing has determined that a logical error exists in the record. For MTST or for MTDI with NOEDIT, the only allowable type of error record is one with a data check. (For other types of error records, see "Error Description for MTDI with Editing" later in this section.) This exit routine may be used to analyze and, if possible, correct the error record.

- The OUTREC (TSW880) exit is taken just prior to passing a valid record to the normal output data set (SYSUT2). This exit routine may be used to modify the record just prior to output and may also perform the actual writing of the record.

- The OUTHDR2 (TCR950) exit is taken during the process of opening the data set for SYSUT2. It may be used to create up to eight user output header labels.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Record Status	Options and Exits				Record Destination
Error Record	ERROR Exit Specified	Return Code = 0			No place
		Return Code = 4			SYSUT3
		Return Code = 8	OUTREC Exit Specified	Return Code = 0	No place
				Return Code = 4	SYSUT2
			OUTREC Exit Not Specified		SYSUT2
	ERROR Exit Not Specified	ERROPT = NOERR	OUTREC Exit Specified	Return Code = 0	No place
				Return Code = 4	SYSUT2
			OUTREC Exit Not Specified		SYSUT2
		ERROPT = NORMAL			SYSUT3
Valid Record	OUTREC Exit Specified	Return Code = 0			No place
		Return Code = 4			SYSUT2
	OUTREC Exit Not Specified				SYSUT2

Figure 137. Record Destination as Determined in the Output Processor Routine

- The OUTHDR3 (TCR960) exit is taken during the process of opening the data set for SYSUT3. It may be used to create up to eight user output header labels.
- The OUTTLR2 (TCR955) exit is taken during the process of closing the data set for SYSUT2. It may be used to create up to eight user output trailer labels.
- The OUTTLR3 (TCR965) exit is taken during the process of closing the data set for SYSUT3. It may be used to create up to eight user output trailer labels.

PROGRAM SWITCHES

In order to properly perform MTDI editing and record error checking, several program switches are necessary, the most important of which are described in the following text:

- Data Check Switch (Bit 4 of switch labeled SWITCH1)—Is set equal to 1 any time a data check terminates a READ command. If this switch is 1, the SCAN routine signals the move routine that a replace character is to be moved to the output area. It is also used by the read routine to determine whether two data checks occur back-to-back in the input stream, and to relay this information to the output processor routine. The data check switch is set to 0 after the record is processed by the output processor routine.
- Delete Switch (Bit 0 of switch labeled SWITCH1)—Is set to 1 whenever a cancel (CAN) code is detected by the scan routine. This switch is used by the scan routine to decide whether to send a string of data to the move routine or to continue scanning. This switch is set to 0 whenever a valid end-of-record or end-of-tape condition is detected.
- Dup Not-Allowed Switch (Bit 6 of switch labeled SWITCH1)—Is used by the duplication section of the scan routine to control whether data duplication is to be performed. This switch is initially set to 1 (to indicate duplication not allowed) and is again set to 1 for the first record of each tape, and for each record that follows a canceled record. Otherwise, it is set to 0 (to indicate duplication allowed) after each record has been processed by the output processor routine.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

ERROR DESCRIPTION WORD			
Byte 0	Byte 1	Byte 2	Byte 3
Level Status	Type Status	Program Level	Record Status
(0) = For any Error Record which will not Cause Questionable Data to be in the Following Record(s). (1) = For any Error which may Cause Questionable Data in the Following Record(s). (2) = For any Error Which Contains Questionable Data due to a non Zero Error Level of Preceding Record(s) and may Cause Questionable Data to be in the Following Record(s).	(0)= No Identifiable Error. (1)= Start-of-Record (SOR) or End-of-Record (EOR) in Error. (2)= Length Error. (4)= Field Error. (8)= Data Check Error. NOTE: Error Records may have hex Combinations of the above Error Types e.g., a C Would Indicate a Data Check and a Field Error.	(1) = P1 (2) = P2 (3) = P3 (4) = P4 (5) = P5 (5) = P5 (6) = P6 (7) = P7 (8) = P8 (E) = None of the Above. Start-of-Record (SOR) in Error.	(U)=Unverified Record. (V)=Verified Record. (E)=Neither of the Above. End-of-Record (EOR) in Error.
The Error Description Word is in EBCDIC Format, e.g., a 2 is Represented as X'F2' ; a C is Represented as X'C3'.			

Figure 138. Error Description Word

- Start-of-Record (SOR) Switch (NOP switch labeled TSW315)—Is initially set to NOP (SOR expected) and is again set to NOP whenever a program level (P1-P8) code, a cancel (CAN) code, a verify OK (VOK) code, or a record mark (RM) code is detected by the scan routine; or whenever it is necessary to force an end-of-record condition due to the fact that an end data (ED) code or a group separator (GS) code has been detected by the read routine, or a record is longer than MAXLN. This switch is used by the scan routine to verify the start-of-record code. It is also used by the move routine to indicate that an end-of-record condition has been found before the output area has been filled. This switch is set to BR (SOR not expected) after the SOR character has been processed.

- End-of-Tape (EOT) Switch (NOP switch labeled TSW113)—Is set to NOP (EOT found) whenever an ED code on an MTDI tape or a stop (ST or st) code on an MTST tape is detected. It is used by the read routine to initialize a REWIND and UNLOAD command to discard the current tape and attempt to load a new one. This switch is set to BR (EOT not found) when the next command is issued to the input device.

- Read/Edit Error Switch (Bit 4 of switch labeled ERRIND)—Is set to 1 whenever one of the output data sets (SYSUT2 or SYSUT3) cannot be opened properly, whenever the return code

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

from a user exit routine indicates an error, or whenever a permanent error occurs on SYSUT1. The proper entry in the message table (in the common communication region) is flagged, and control is returned to the driver routine, which writes the message and terminates the utility.

SYSTEM CONTROL BLOCK USAGE

Because the EXCP macro instruction is used to read in the data from SYSUT1, a channel program (which consists of two channel command words) and three system control blocks (DCB, ECB, and IOB) must be constructed and used.

- Channel Program—Two channel command words (CCWs) are used for double buffering purposes. Each CCW contains an I/O command, a buffer address, a flag byte, and a buffer byte count. The I/O command is modified during execution of the utility to be either a READ command or a REWIND and UNLOAD command. The buffer address and byte count are determined and set when the read/edit module is entered. They remain fixed throughout execution of the utility. The flag byte is set to suppress incorrect length indication. It is not altered during execution of the utility.

- DCB (Data Control Block)—Is located in the common communication region. It is set up as a skeleton DCB containing only a code indicating that it is a DCB used in conjunction with the EXCP macro instruction, a ddname to indicate the data set to process, and an exit list address. The address of the DCB must be specified in the OPEN macro instruction to make the data set available for processing, and in the IOB that is used by the EXCP macro instruction to access the data set. If the BUFL parameter was specified by the user on the DD statement, this value is used to provide buffers and to construct the CCWs. The exit list consists of a single address constant pointing to a 176-byte area. This area is used as a work area for reading the job file control block (JFCB). The JFCB is read by means of the RDJFCB macro instruction prior to opening the data set. This allows extraction of any volume serial number that may have been supplied by the user.

- ECB (Event Control Block)—Is used by the operating system to indicate whether an I/O operation is in process or has completed. The read/edit module tests the ECB via a WAIT macro instruction after issuing an I/O request. When the I/O operation is complete, the completion code which the operating system posts in the ECB is tested to determine the appropriate action to be taken. Then the ECB is reset to allow another I/O operation to be posted.

- IOB (Input/Output Block)—Initially contains the addresses of the ECB, DCB, and channel program. The channel program address is modified each time the EXCP macro instruction is issued to permit use of two different channel programs to effect double buffering. Upon completion of an I/O operation, the read/edit module uses this information to determine the appropriate action to be taken. The IOB is reset after each permanent data check to permit further processing.

Because QSAM is used to send output to SYSUT2 and SYSUT3, it is necessary only to provide a DCB for each data set and issue a PUT (move mode) macro instruction to transfer the data. The DCBs for both SYSUT2 and SYSUT3 are constructed to use simple buffering with the PUT-MOVE mode of operation to transfer data to any device that can accept a sequential data set.

Addresses are provided for a synchronous error analysis routine (SYNAD) and for a list (EXLST) of addresses of routines to handle exceptional conditions. The exceptional conditions to be handled are output header label processing, output trailer label processing, DCB modification, and JFCB processing.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Header labels for SYSUT2 and SYSUT3 can be created by a user exit routine during the opening of these data sets. Likewise, trailer labels for SYSUT2 and SYSUT3 can be created by a user exit routine during the closing of these data sets.

The DCB is modified only if it is incomplete when opening the data set, in which case predetermined default values are inserted by the DCB exit routine in IEBTCRIN (Figure 132 on page 357 and Figure 133 on page 357).

ERROR DESCRIPTION FOR MTDI WITH EDITING

Each error record as determined by the utility will have a 4-byte error description word (EDW) appended to the front of the record. The EDW will consist of a level status byte, a type status byte, a program level byte, and a record status byte (Figure 138 on page 367).

An error record is passed to the ERROR exit routine when either level status or type status has a value other than zero.

Unless either ERROPT=NOERR or the user specifies a return code of 0 or 8 from the ERROR exit routine, error records are written to SYSUT3.

CHARACTER CODE DEPENDENCY

The operation of the IEBTCRIN utility program is dependent upon an internal representation of the external character set that is equivalent to the one used at assembly time. Two modules, control card analyzer and read/edit, are affected by this dependency.

The control card analyzer has two keyword tables—a 264-byte TCRGEN table and a 40-byte EXITS table—that have to be reassembled for a code reassignment. In addition to the tables, a single constant labeled "X" will also have to be reassembled.

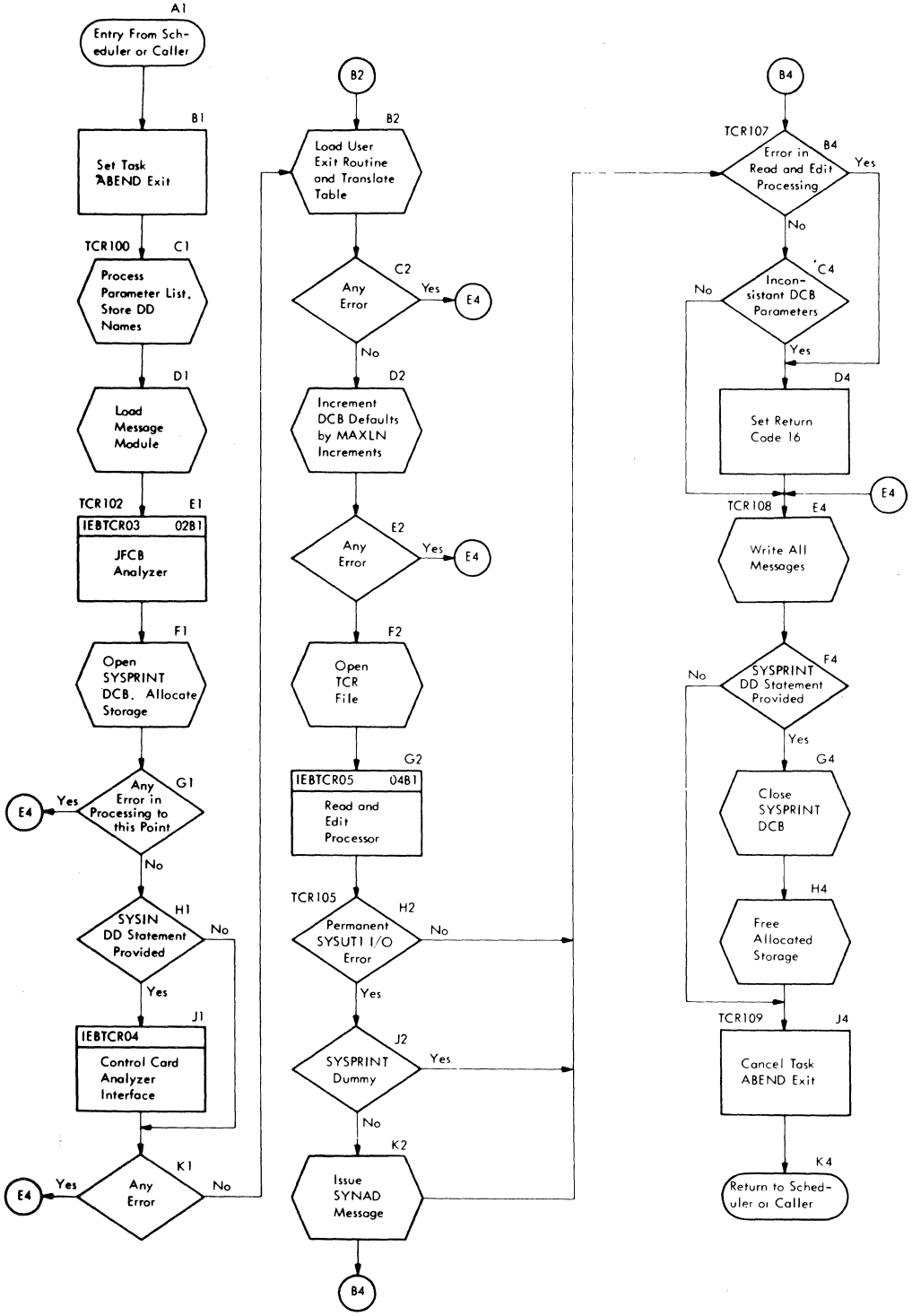
Conversion of the REPLACE character to hexadecimal representation is dependent upon the internal representation of the EBCDIC characters 0 through 9 and A through F. Redefinition of these characters will necessitate recoding of the control card analyzer conversion routine TCR530.

The read/edit module uses two MTST-to-EBCDIC translate tables, an EDW-to-EBCDIC translate table, and a series of EQU constants that must be reconstructed for a code reassignment. The specific instruction sequences, tables, and constants that require modification for a code reassignment are identified as follows:

- The instruction sequence from TCR319 to TsW328 (scan routine), which does a program level search, assumes that the program level codes are sequentially and successively assigned.
- The instruction sequences from TCR610 to TCR612 (scan routine for program level codes) and at TCR860 (output processor routine), which convert the program level codes to the form '01' through '08,' also assume that the program level codes are sequentially and successively assigned.
- The translate tables labeled "EDWTT", "UPPER", and "LOWER" contain entries in hexadecimal notation to convert data to the EBCDIC character set.
- The constant labeled "HEXEOR" and the set of EQU instructions from "MTDILZF" to "MTSTUSRC" define the special characters (in character form) for the input data set.

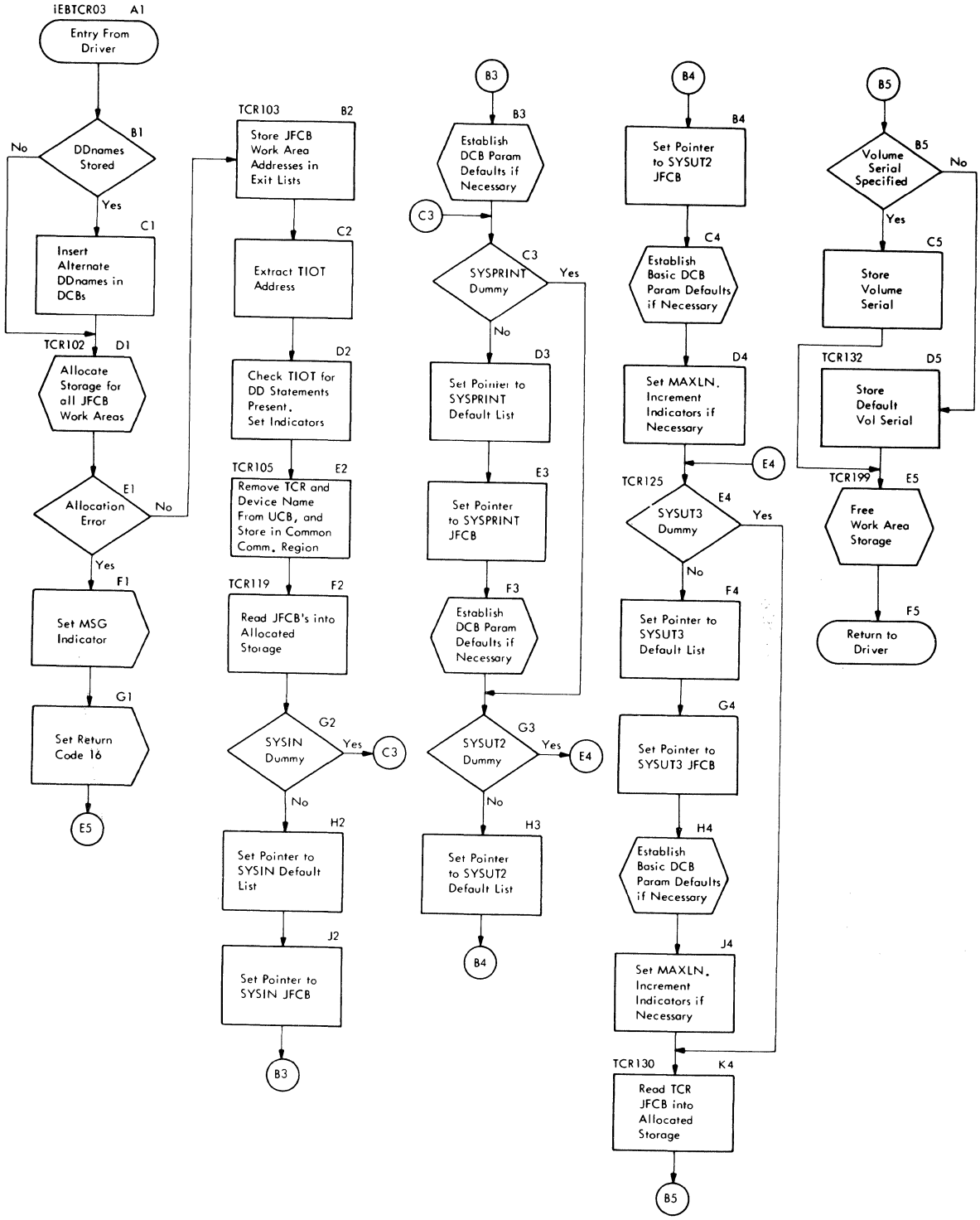
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 01. DRIVER FIRST LEVEL ROUTINE



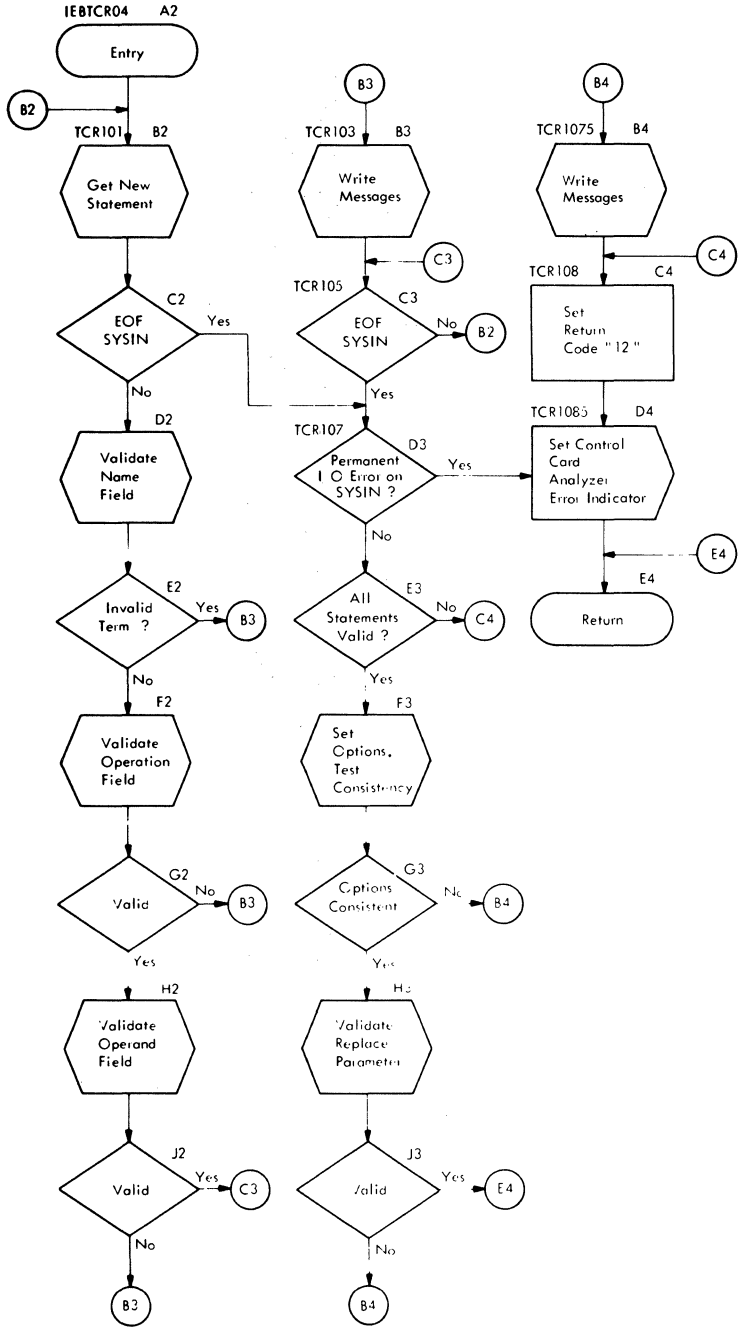
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 02. JOB FILE CONTROL BLOCK (JFCB) ANALYZER, IEBTCR03



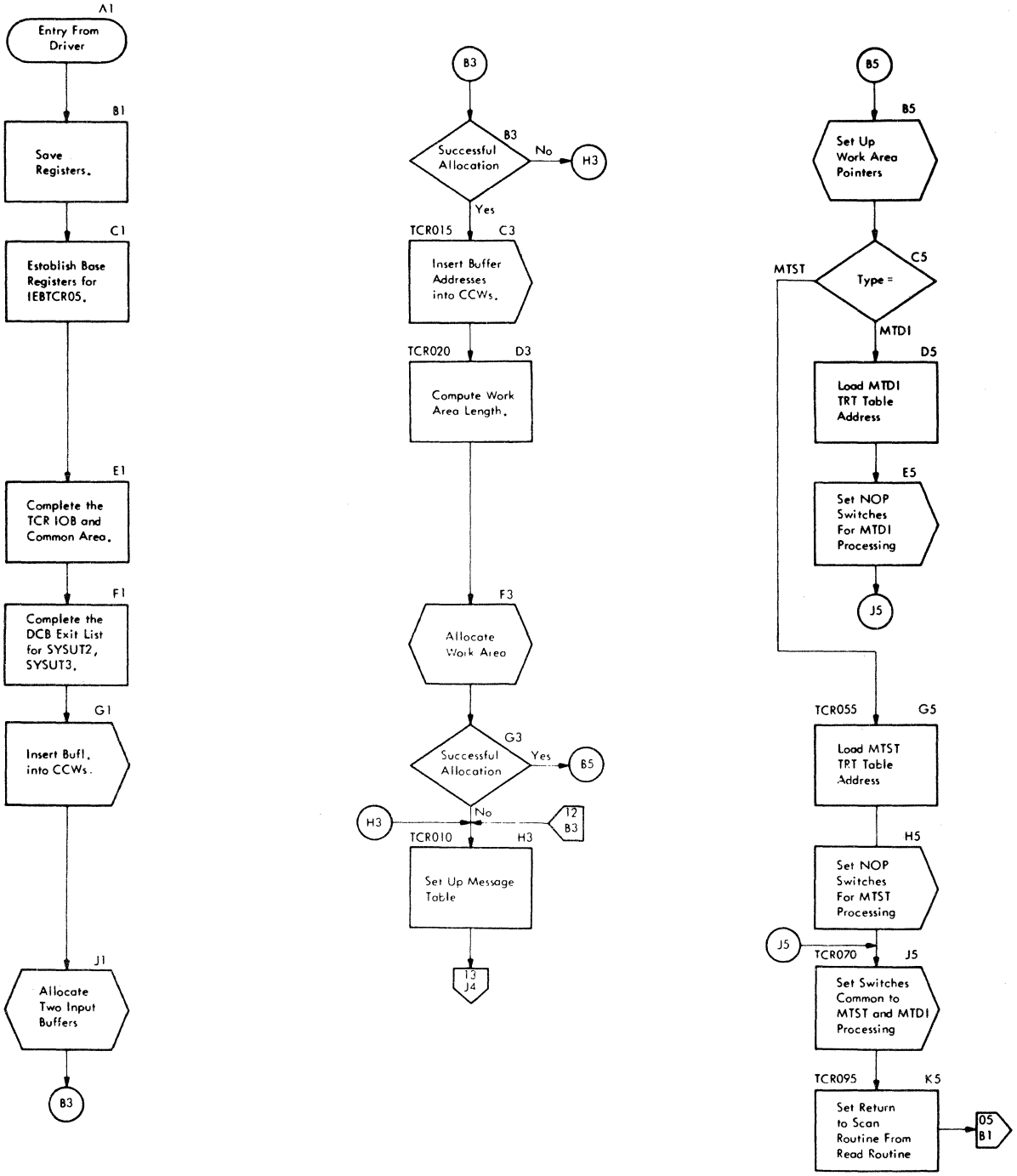
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 03. CONTROL CARD ANALYZER—FIRST LEVEL ROUTINE, IEBTCR04



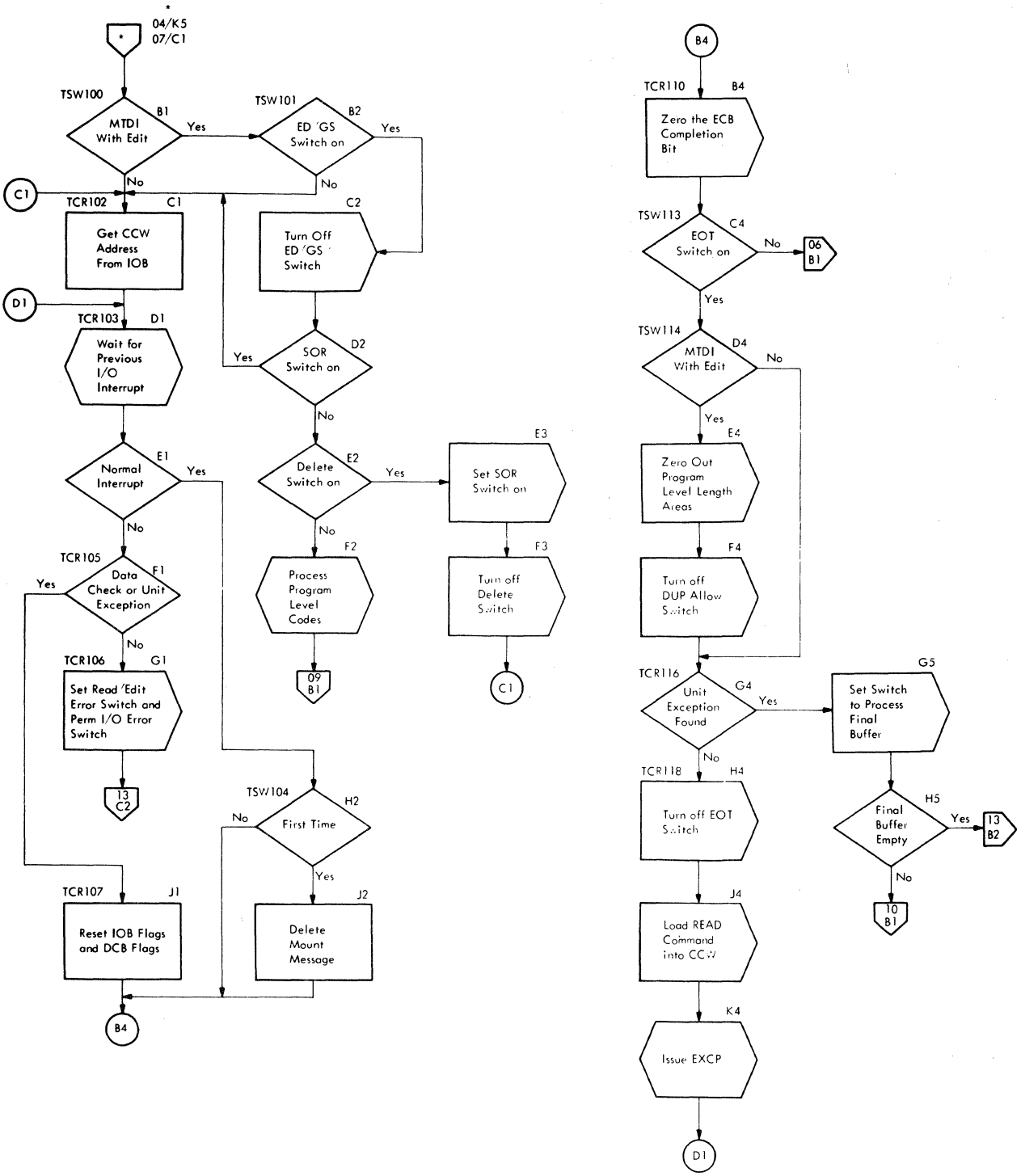
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 04. READ/EDIT ENTRY ROUTINE OF IEBTCR05



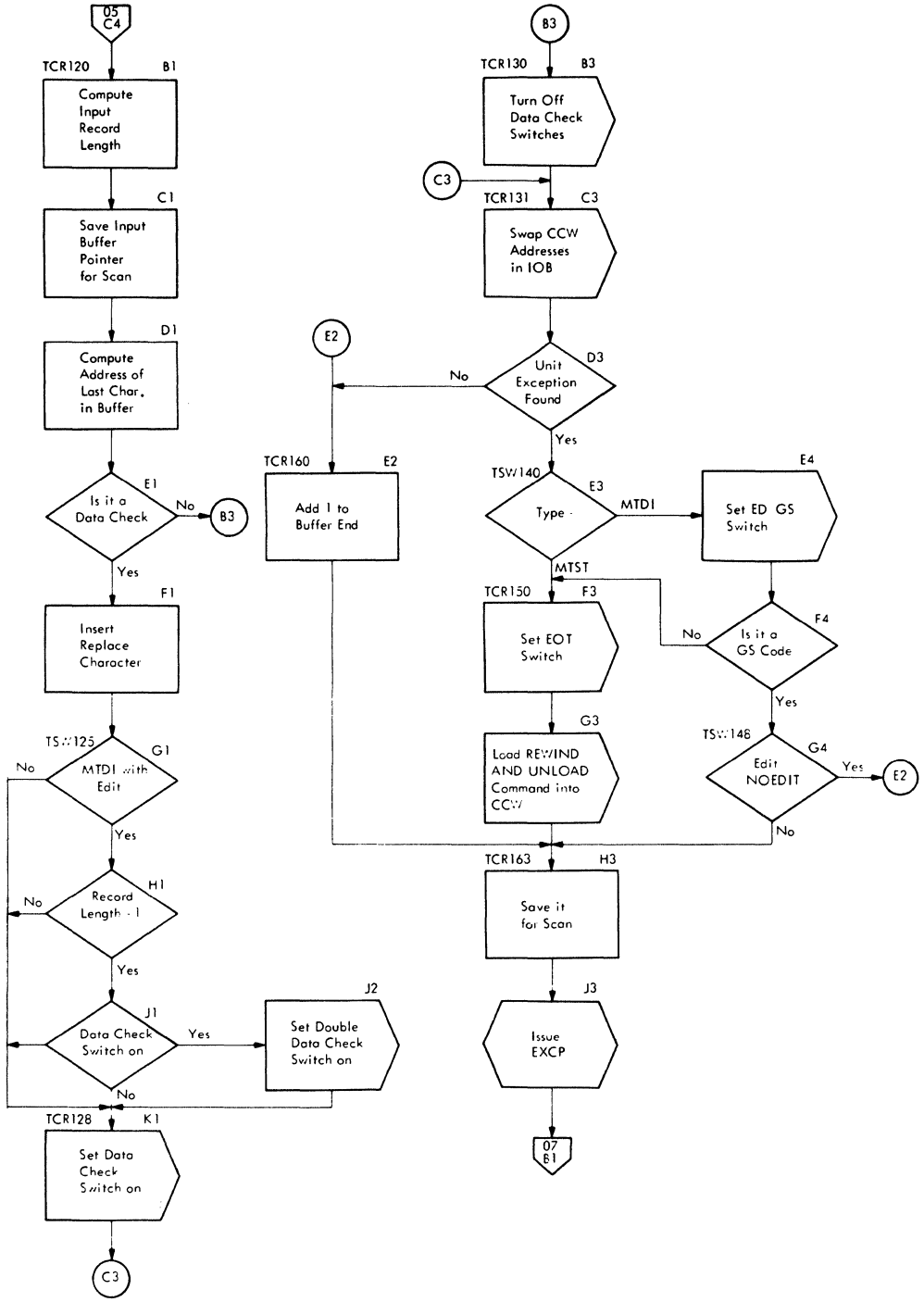
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 05. READ/EDIT READ ROUTINE OF IEBTCR05 (PART 1 OF 2)



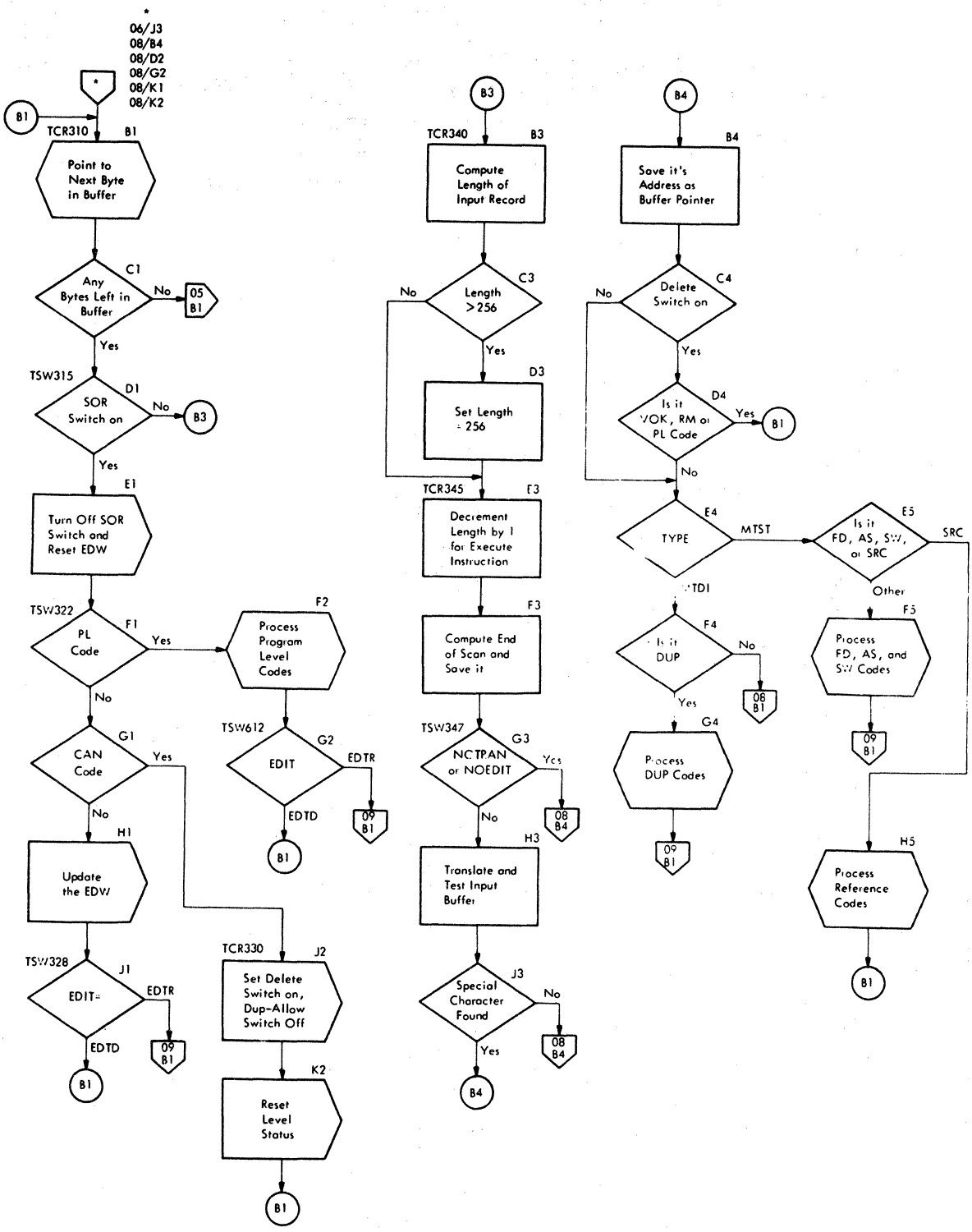
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 06. READ/EDIT READ ROUTINE OF IEBTCR05 (PART 2 OF 2)



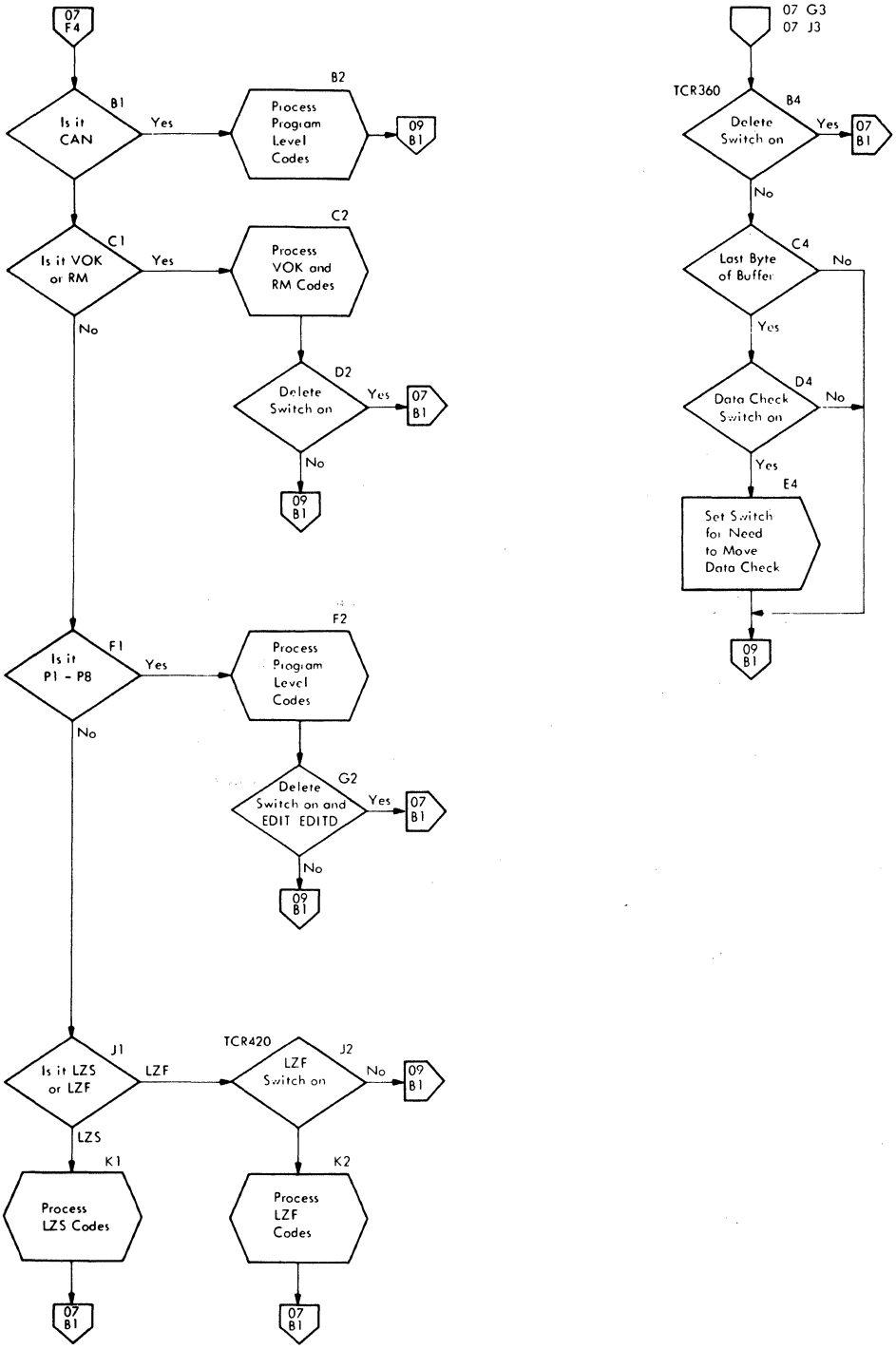
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 07. READ/EDIT SCAN ROUTINE OF IEBTCR05 (PART 1 OF 2)



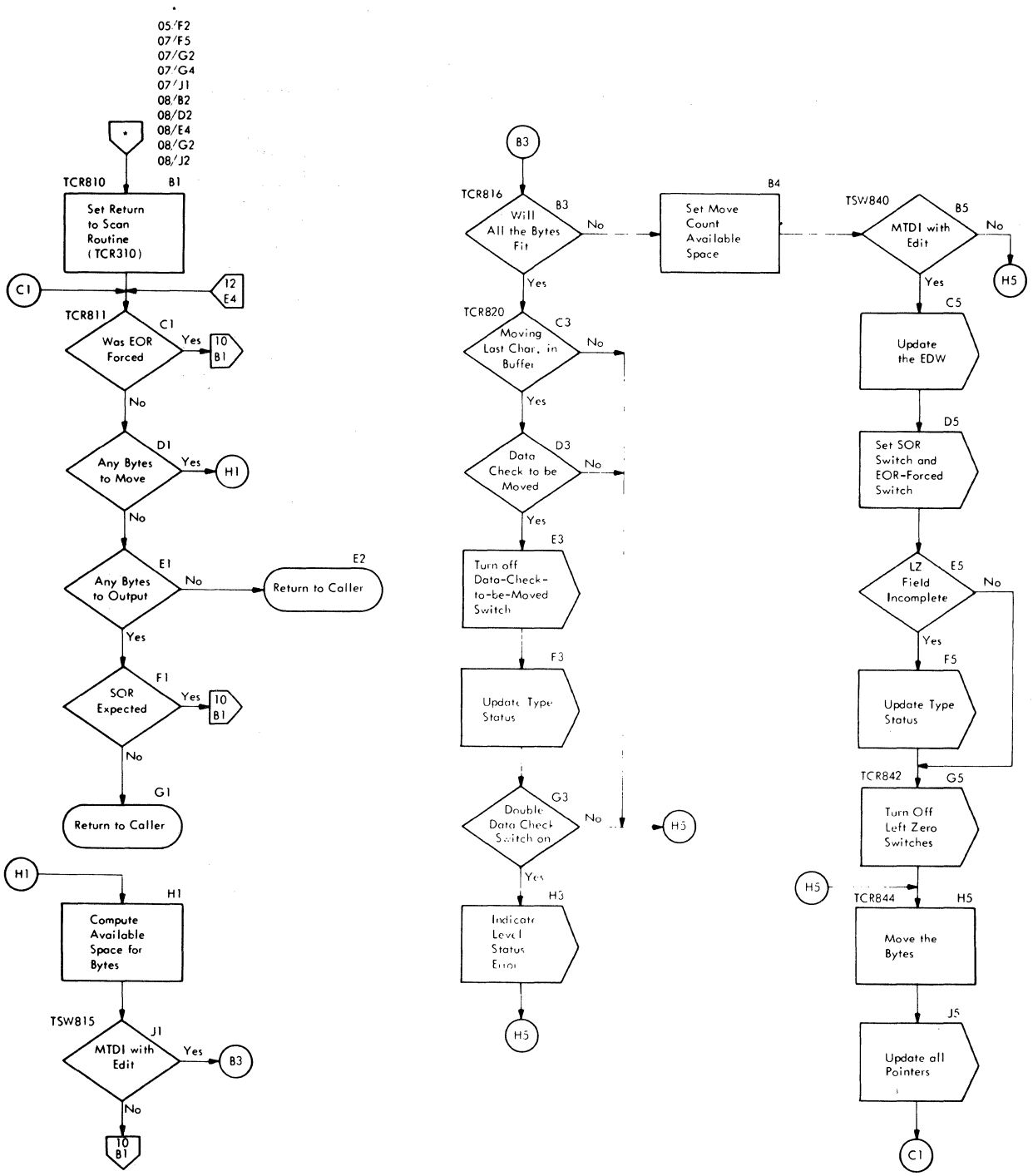
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 08. READ/EDIT SCAN ROUTINE OF IEBTCR05 (PART 2 OF 2)



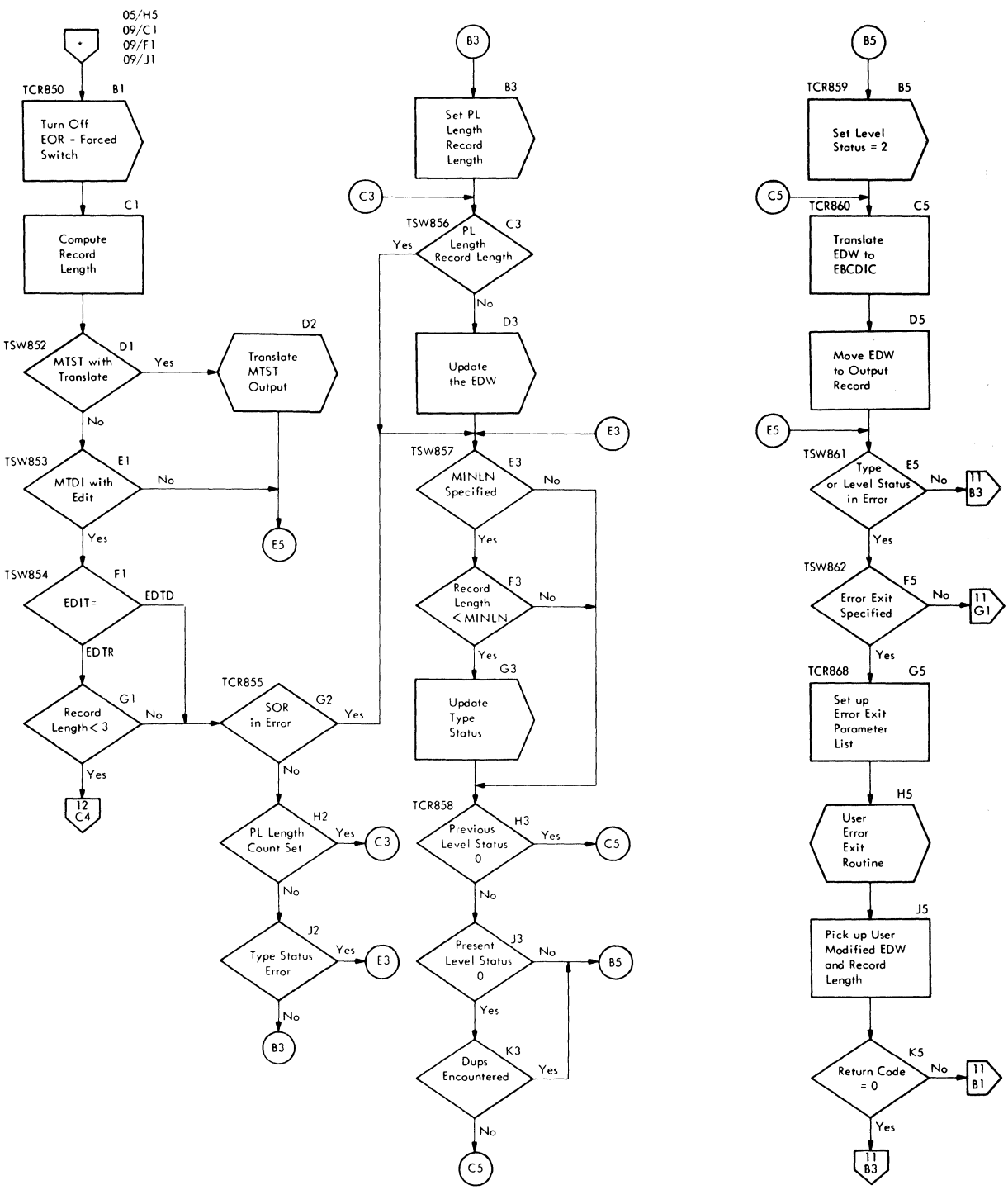
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 09. READ/EDIT MOVE ROUTINE OF IEBTCR05



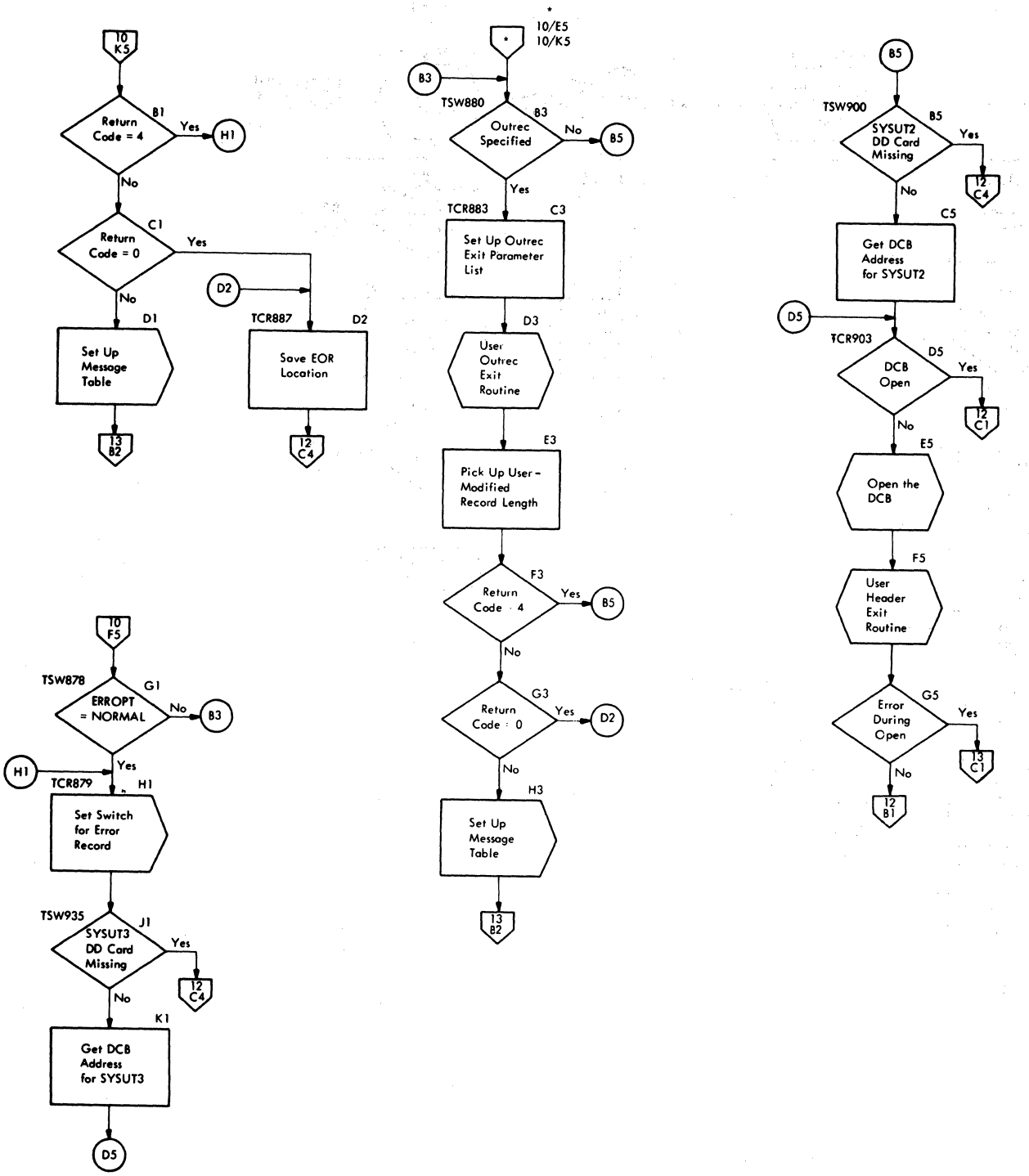
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 10. READ/EDIT OUTPUT PROCESSOR ROUTINE OF IEBTCR05 (PART 1 OF 3)



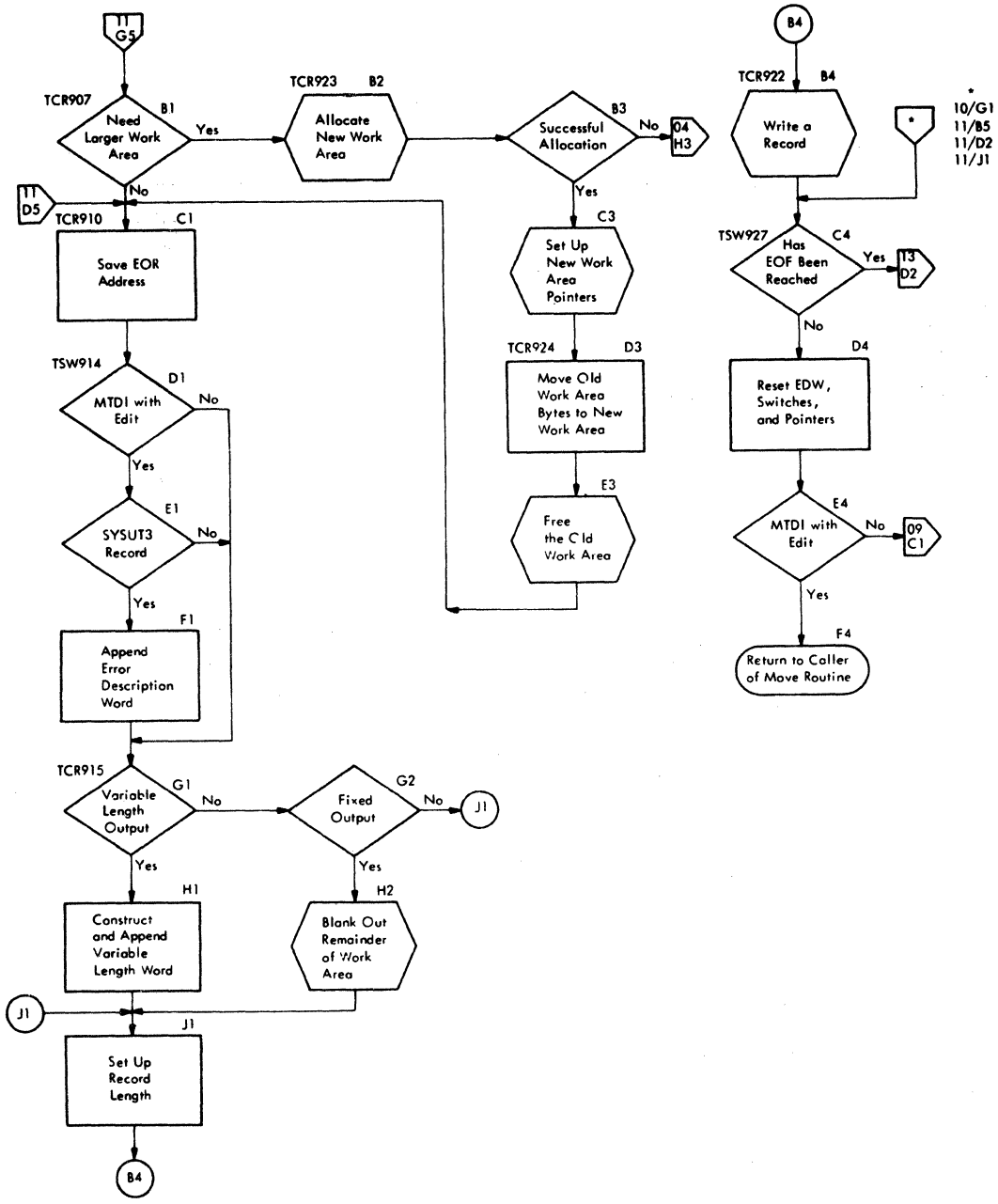
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 11. READ/EDIT OUTPUT PROCESSOR ROUTINE OF IEBTCR05 (PART 2 OF 3)



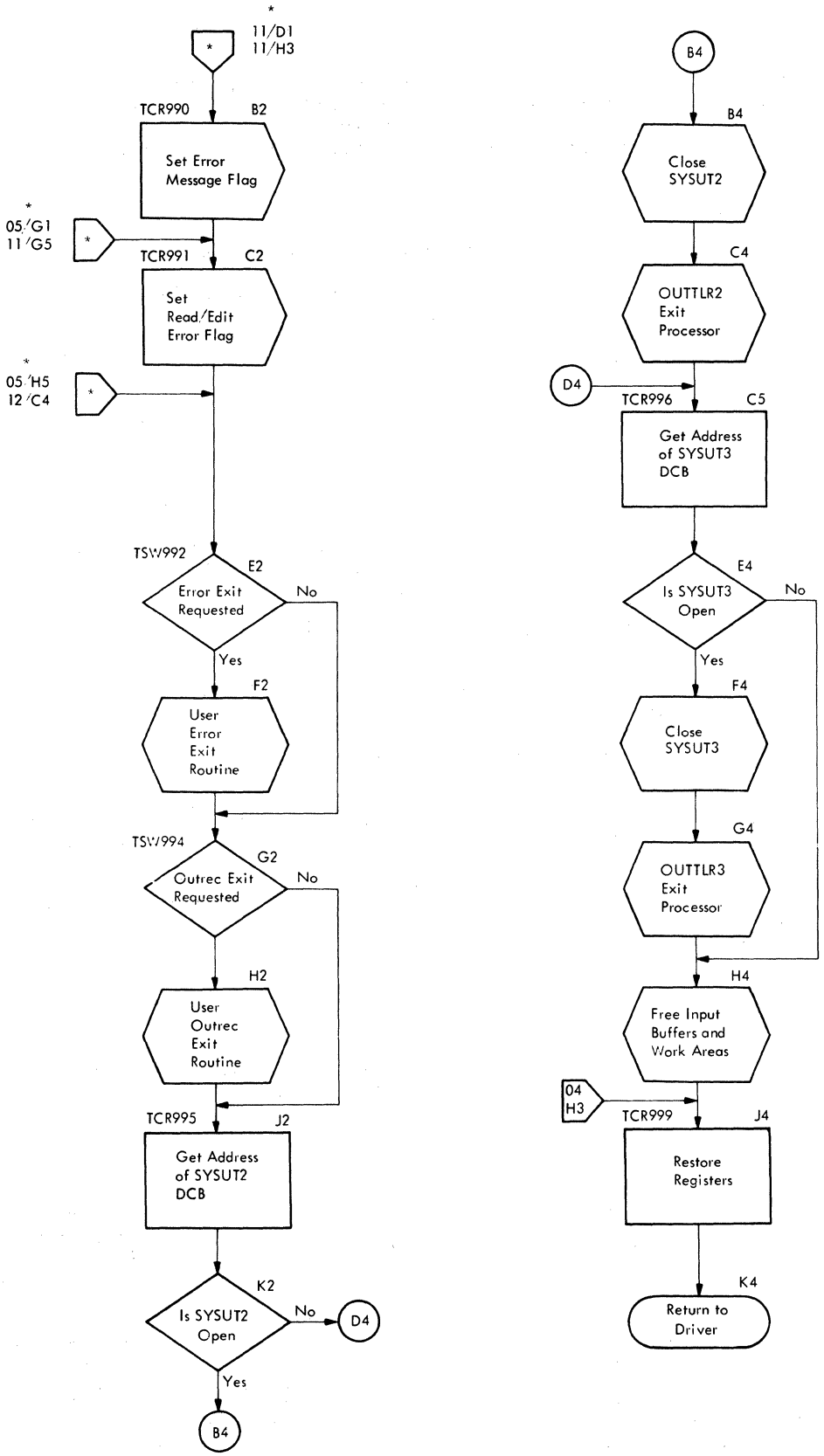
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 12. READ/EDIT OUTPUT PROCESSOR ROUTINE OF IEBTCR05 (PART 3 OF 3)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 13. READ/EDIT END OF UTILITY PROCESSOR ROUTINE OF IEBTCR05



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEBUPDTE PROGRAM

The IEBUPDTE utility program incorporates both IBM-supplied and user-generated source language modifications into sequential data sets or into partitioned data sets. The input and output data sets contain blocked or unblocked logical records of 80 bytes or less each.

The program can:

- Add, copy, and replace members of data sets
- Add, delete, replace, and renumber the records within an existing member or data set
- Assign sequence numbers to the records of a member or data set
- Create a sequential master data set from an input partitioned data set, and vice versa

At the completion or termination of the program, the highest return code encountered within the program is passed to the calling program. The utility program can also produce a message data set containing a listing of the contents of the output data set, the control statements submitted to the utility program, and, if applicable, error messages.

Data definition (DD) statements needed to run the program are as follows:

- SYSUT1, which defines the old master data set (sequential or partitioned)
- SYSUT2, which defines the new (updated) master data set (sequential or partitioned)
- SYSIN, which defines a sequential data set containing the transactions to be applied to the old master data set
- SYSPRINT, which defines a sequential data set containing either changes to the old master or contents of the new master, as well as utility control statements used and any error messages generated

PROGRAM STRUCTURE

The program consists of three segments (one load module): the root segment (IEBUPDTE), the control card analyzer segment (IEBASCAN and IEBBSCAN), and the initialization module (IEBUPNIT).

THE ROOT SEGMENT

The main functions of the root segment are the processing of records and the printing of messages. The segment contains four control sections (CSECTs): IEBUPDTE, IEBUPLOG, IEBUPDT2 and IEBUPXIT.

- IEBUPDTE receives initial control from the supervisor, obtains storage for the communication region IEBUPCON, opens the SYSIN data set, and passes control to module IEBUPNIT for initialization. For writing a header message on the SYSPRINT output device, CSECT IEBUPDTE passes control to CSECT IEBUPLOG. After the return (of control) from CSECT IEBUPLOG, CSECT IEBUPDTE gives control to CSECT IEBUPDT2 to begin the actual processing of records.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- IEBUPLOG is a closed subroutine that writes messages and records on SYSPRINT. The first execution of IEBUPLOG opens SYSPRINT and the last closes it and returns control to the supervisor.

- IEBUPDT2, the heart of the program, opens the old and new master data sets, reads, processes, formats, and writes the output records, and stows member names if the new master is partitioned.

Note: Prior to each writing of a record on SYSUT2, module IEBUPDT2 checks the TOTALSW field in the IEBUPCON region to determine if a user-totaling exit is to be taken. If so, a parameter list address is loaded into register 1, and an exit is made to the user routine. When the user routine returns control to module IEBUPDT2, a return code established by the user routine is checked, and the action taken is as described in Appendix B. If user labels are to be processed on either SYSUT1 or SYSUT2, CSECT IEBUPDT2 passes control to module IEBUPXIT through data management routines during open, close, or end-of-volume processing. CSECT IEBUPDT2 passes control to the IEBASCAN segment to scan and analyze control statements, and to CSECT IEBUPLOG to log messages and records on SYSPRINT.

- IEBUPXIT is the module containing the program's exit routines. For each of the three DCBs (SYSIN, SYSUT1, and SYSUT2), this module contains an entry point for each of the following closed subroutines: DCB exit, header-label exit, trailer-label exit, SYNAD exit, and end-of-data exit (there is no end-of-data exit for the SYSUT2 data set). For the SYSPRINT DCB, this module contains an entry point for the DCB exit and SYNAD exit closed subroutines.

THE CONTROL CARD ANALYZER SEGMENT

The main functions in analyzing control cards are performed by two CSECTs in this segment: IEBASCAN and IEBBSCAN.

- IEBASCAN scans and analyzes control statements and sets appropriate flags in the IEBUPCON region. CSECT IEBASCAN gives control to CSECT IEBUPLOG to print a copy of the control statement. To scan the individual parameters of each control statement, CSECT IEBASCAN gives control to CSECT IEBBSCAN with a doubleword parameter list, located at address STOREG in the communication region. For the reading of a control statement continuation card, CSECT IEBBSCAN gives control to CSECT IEBUPDT2. The first word in the list is the length of the last parameter analyzed by CSECT IEBBSCAN. The last word contains a pointer to the same parameter's location in a buffer area, SWITCHRD.

- IEBBSCAN, which is a closed subroutine, receives control from CSECT IEBASCAN and returns control either to CSECT IEBASCAN or to CSECT IEBUPDT2 (for reading another control card). CSECT IEBBSCAN scans the individual parameters on each control statement. If diagnostic messages are required as a result of the scanning, control is given to CSECT IEBUPLOG.

INITIALIZATION MODULE

There is one module in this group: IEBUPNIT.

- IEBUPNIT, which is a closed subroutine, receives control from CSECT IEBUPDTE for the purpose of initializing the region IEBUPCON, and analyzing the parameters on the EXEC card.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

PROGRAM FLOW

Figure 139 shows the overall flow of the program; more detailed flow is shown in Charts 01, 02, and 03.

PROCESSOR DATA FLOW

Figure 140 on page 386 indicates the paths taken by data from SYSUT1 and SYSIN. The following is a description of data flow within the processor (IEBUPDT2) according to the type of run: NEW or MOD.

NEW: This type of transaction involves reading data from SYSIN and writing it on SYSUT2 and (if specified) on SYSPRINT. Logical records are read in turn from SYSIN into the input buffer at SWITCHRD + 1; records are then stacked in the SYSUT2 output buffer at NMWRITEP until the desired blocking factor is reached. A physical record is then written on the new master (SYSUT2). This process is repeated until SYSIN has been exhausted. If the SYSUT2 data set is partitioned (as indicated by the NAME keyword), the member name is stowed.

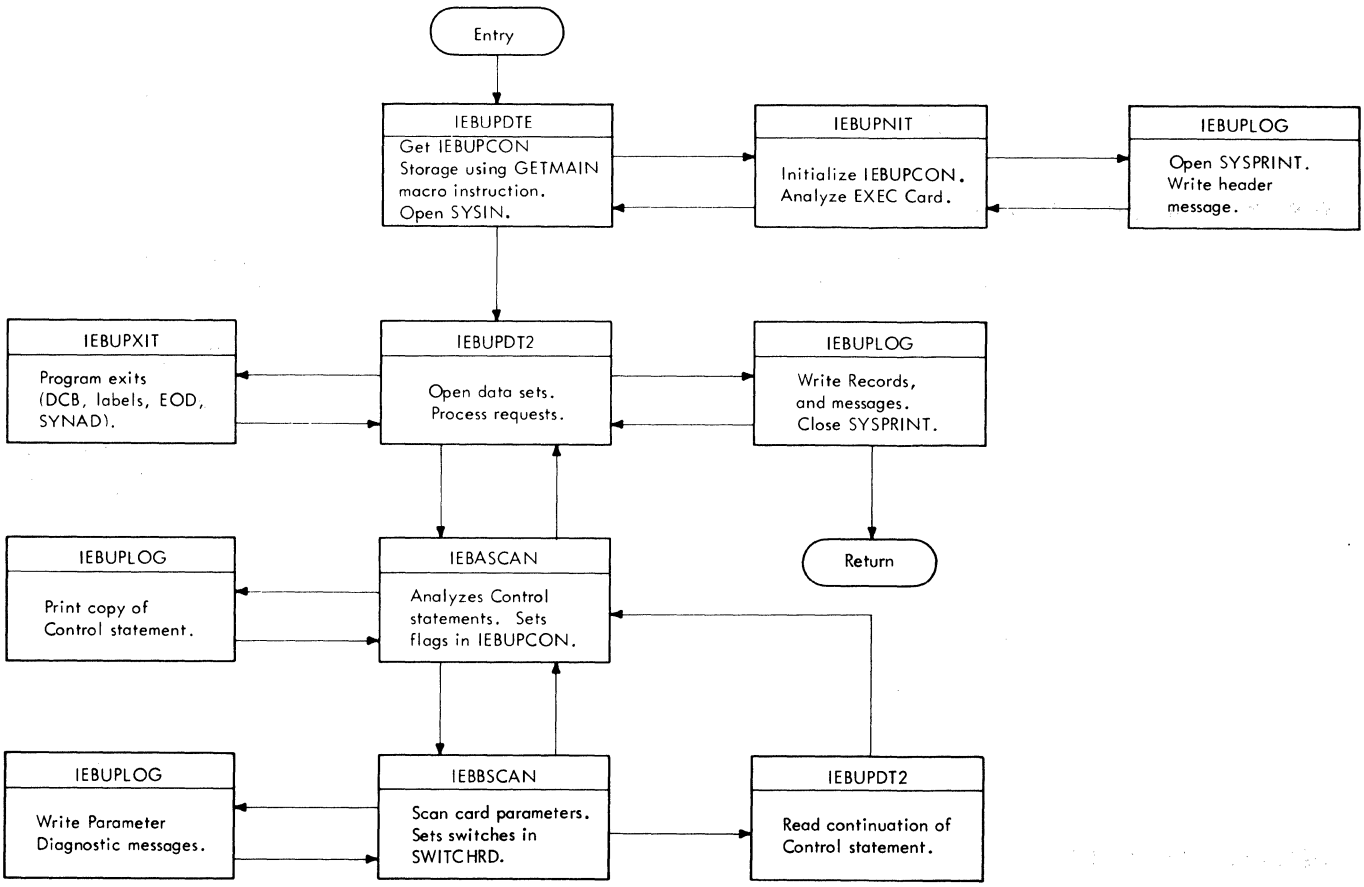


Figure 139. Overall Flow

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

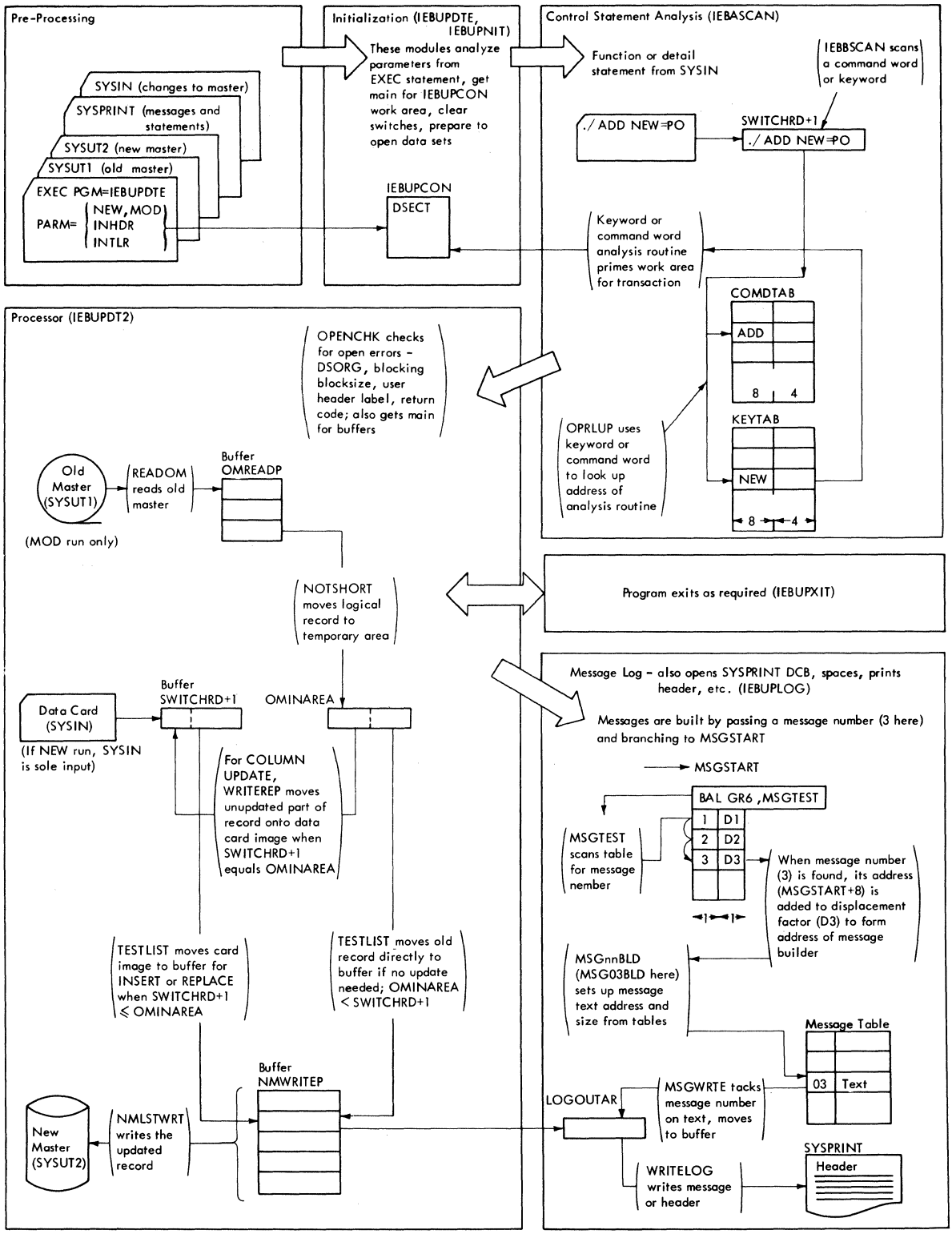


Figure 140. Principle of Operation

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

MOD: This type of transaction involves reading data from SYSUT1 (the old master data set) and SYSIN, merging them as indicated on function and detail statements, and writing the resultant data on SYSUT2. The updated master is written on SYSUT1 only when UPDATE=INPLACE is specified.

- For a REPRO run, a physical record is read from SYSUT1 into the buffer OMREADP, and logical records are then moved individually to OMINAREA for inspection and then to the output buffer NMWRITEP until the desired output blocking factor is reached; the output record is then written on SYSUT2.

- For an ADD run, records are read from SYSIN and stacked in the output buffer NMWRITEP until the desired blocking factor is reached. The output record is then written on SYSUT2. If the data set is partitioned, the member name specified is stowed in the directory of the new master. If numbering of records is specified, it is performed in the input buffer, SWITCHRD + 1.

- For a REPL run, the flow is similar to the ADD run, except that the new data from SYSIN replaces the member specified. Numbering, if specified, is performed in the input buffer SWITCHRD + 1.

- For a CHANGE run, which operates within a data set or member of a partitioned data set, records may be changed, deleted, numbered, or added, depending on the detail statements and data cards following the change statement. When one of these types of statements (DELETE, NUMBER, or data) has to be read from SYSIN into the buffer SWITCHRD + 1, a record is read from the old master (SYSUT1) into the buffer OMREADP and is processed as follows (see also Figure 140):

  1. A logical old master record is moved from buffer OMREADP to OMINAREA, and if the number or delete is in effect, its sequence number (OM) is compared with SEQ1; otherwise, it is compared with the SYSIN record sequence number.

  2. If OM is less than SEQ1, the old master logical record is moved to the output buffer NMWRITEP, and the next old master record is moved into OMINAREA.

  3. If SEQ1 is less than or equal to OM, and OM is less than or equal to SEQ2, the old master is updated:

     - If the old master logical record is to be deleted, the next old master logical record is moved to overlay it in OMINAREA.

     - If data is to be inserted (if the SYSIN sequence number is less than OM), the SYSIN data statement in SWITCHRD + 1 is renumbered if necessary and moved to the output buffer NMWRITEP, and the next SYSIN record is read. If OM equals the SYSIN sequence number, the SYSIN record replaces the old master record.

     - If the SYSIN sequence number equals OM and COLUMN UPDATE was specified, the portion of the record in OMINAREA not to be updated is moved to its corresponding relative position in the buffer SWITCHRD + 1. This updated record is renumbered if necessary and then moved to the output buffer NMWRITEP, and the next SYSIN and old master records are read.

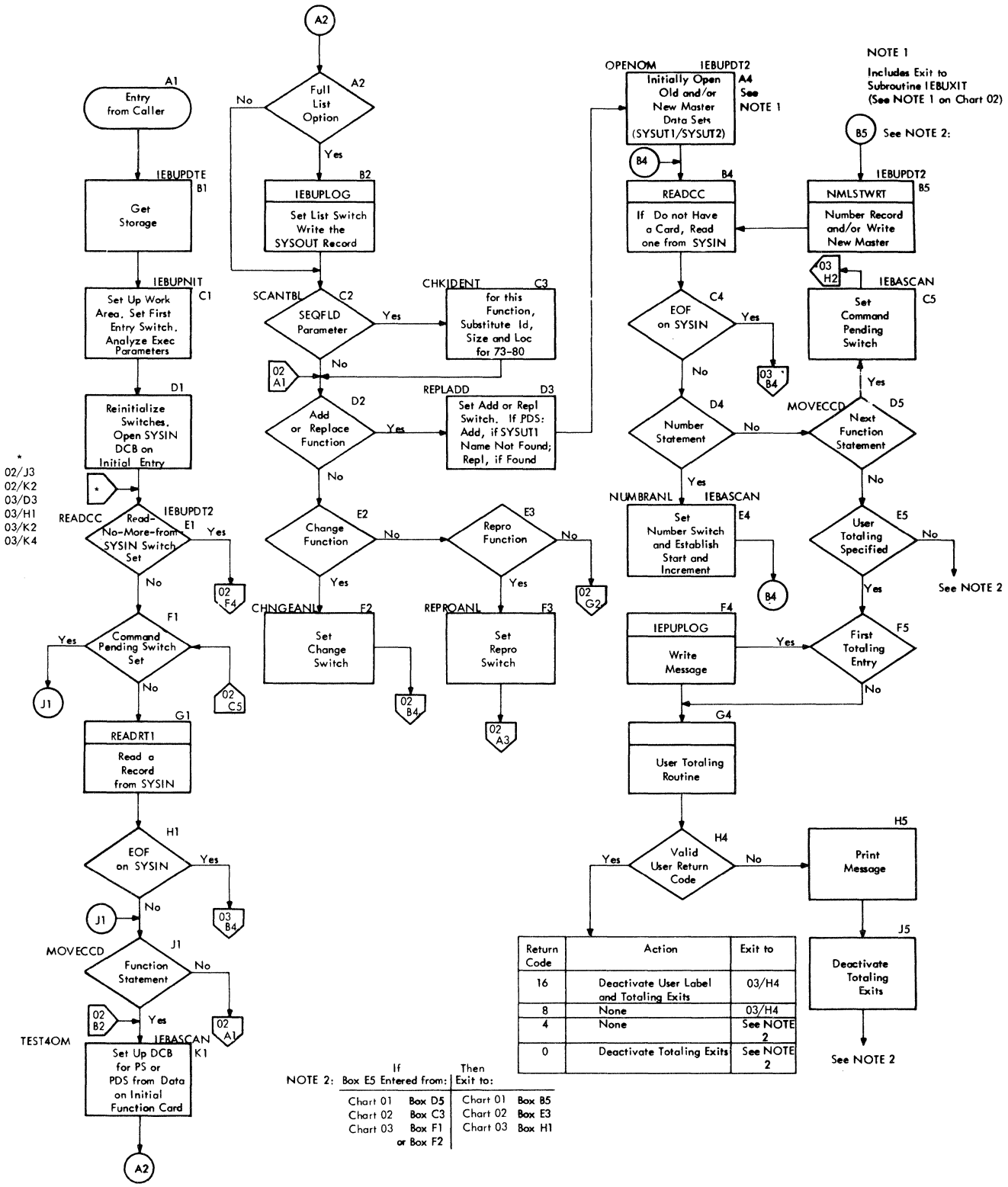
     - If the old master record is to be numbered, the indicated sequence number is stored in it, and the updated master record is moved to the output buffer NMWRITEP. The next old master logical record is then moved into OMINAREA.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

4. When OM is greater than SEQ2, IEBUPDTE checks to see that record SEQ1 was actually processed (deleted or numbered). If it was not, an error message is written and the member update terminates. If it was, the next record from the old master is moved into OMINAREA, and the next record from SYSIN is read into SWITCHRD + 1. Processing of the previous number or delete statement is considered completed.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 01. OVERALL FLOW (PART 1 OF 3)



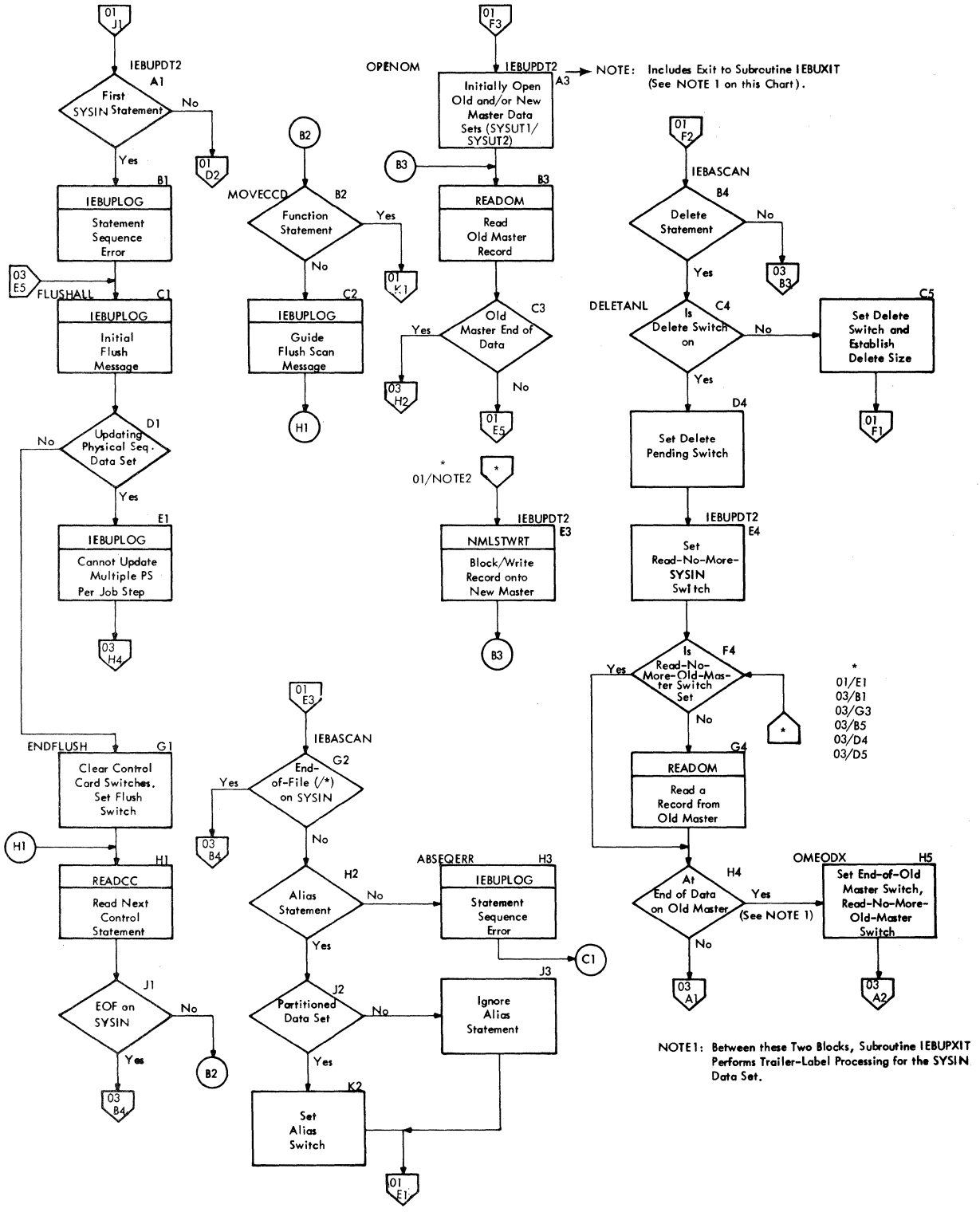
Return Code	Action	Exit to
16	Deactivate User Label and Totaling Exits	03/H4
8	None	03/H4
4	None	See NOTE 2
0	Deactivate Totaling Exits	See NOTE 2

NOTE 2:

If Box E5 Entered from:	Then Exit to:
Chart 01 Box D5	Chart 01 Box B5
Chart 02 Box C3	Chart 02 Box E3
Chart 03 Box F1 or Box F2	Chart 03 Box H1

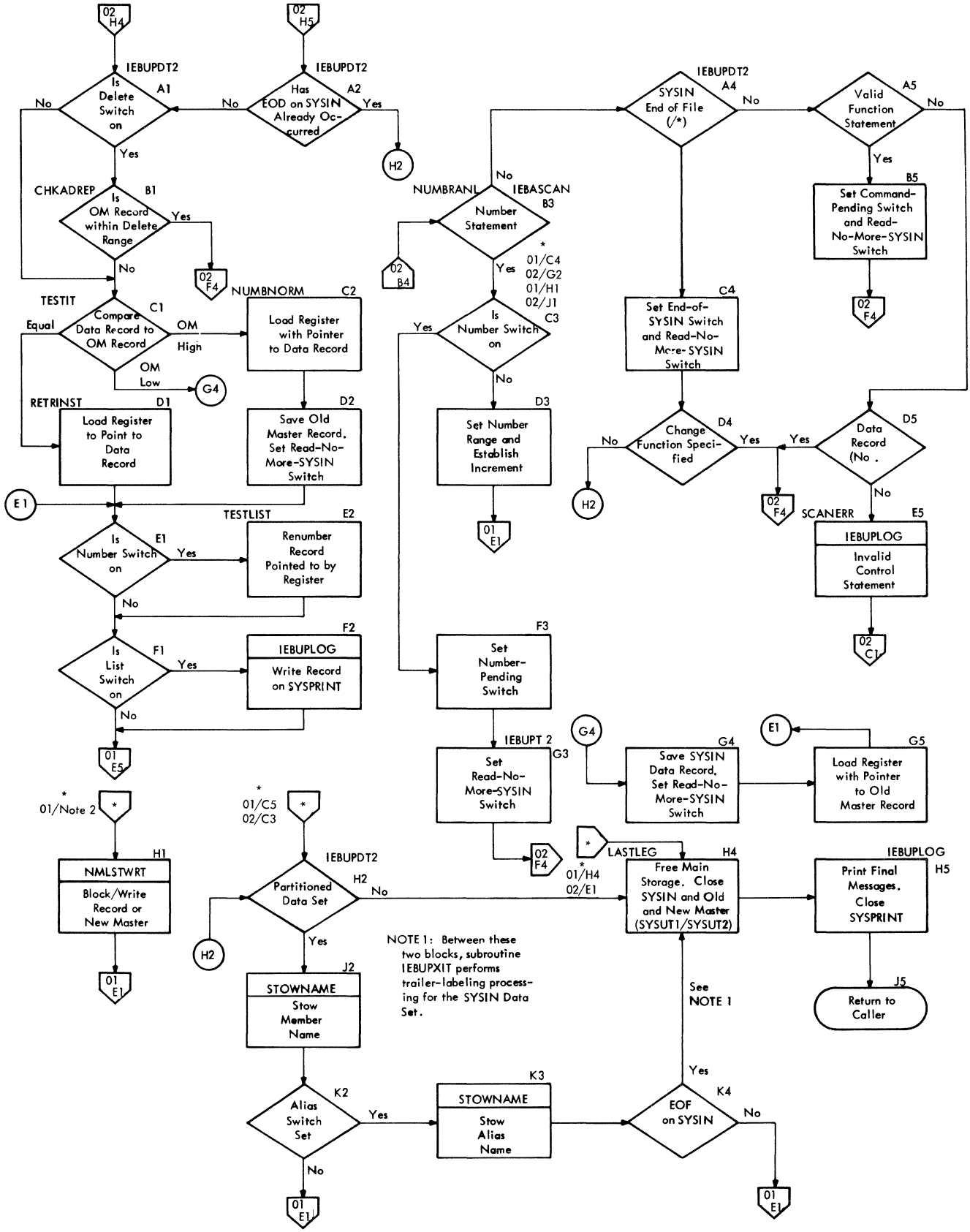
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 02. OVERALL FLOW (PART 2 OF 3)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 03. OVERALL FLOW (PART 3 OF 3)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

The IEHATLAS utility program and the ATLAS (alternate track location and assignment) routine attempt to recover data from direct access devices that have software-assignable alternate tracks. This data would ordinarily be lost during input/output (I/O) operations that result in data check errors or missing address markers. The IBM direct access device types from which data can be recovered are:

IBM 3330 Disk Storage, Models 1 and 11
IBM 3333 Disk Storage and Control, Models 1 and 11
IBM 3340 Direct Access Storage Facility
IBM 3344 Direct Access Storage
IBM 3350 Direct Access Storage
IBM 3375 Direct Access Storage
IBM 3380 Direct Access Storage

CODING NOTE: The 3330 and 3333, Model 1, should always be specified as 3330. The 3330 and 3333, Model 11, should always be specified as 3330-1. The 3330 V should never be specified.

The IEHATLAS utility program attempts to perform an "update-in-place" procedure; that is, it tries to rewrite an "error record" back onto the track where it was intended to be placed when the error occurred. For this program, an error record is defined as that record that was not read or written correctly during an I/O operation. The track on which this record belongs is referred to as the defective or error track.

The ATLAS routine has four primary functions:

- It locates an alternate track upon which the data from a defective track may be placed.
- It copies the data from the defective track onto the selected alternate track.
- It logically chains the alternate track to the original defective track it replaces.
- It updates the VTOC (Format-4) DSCB in the volume table of contents (VTOC) to reflect the availability of alternate tracks on the device.

The user program may invoke the ATLAS routine via either an SVC 86 instruction or an ATLAS macro instruction. ATLAS can rewrite the indicated error record if the channel program in which the error occurred was a "WRITE ..." channel program that can be reexecuted. Otherwise, the user can execute the IEHATLAS utility program as a separate job step to write the error record (in its correct form) on the chosen alternate track.

Only IEHATLAS can attempt the recovery and the replacement of records lost because of errors that occur during execution of the following two channel commands:

READ RECORD ZERO
WRITE RECORD ZERO

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

However, either the IEHATLAS utility program or the ATLAS routine may recover from errors occurring during execution of the following channel commands:

WRITE COUNT, KEY AND DATA
WRITE DATA
SEARCH IDENTITY EQUAL
SEARCH IDENTITY EQUAL OR HIGH
WRITE SPECIAL COUNT, KEY AND DATA
WRITE KEY AND DATA
SEARCH IDENTITY HIGH
READ COUNT

In order for errors to be detected during the "WRITE ..." commands, the read-back check feature must be in effect.

If it is not possible for a user's program to dynamically replace an error record occurring, for instance, during execution of a READ DATA or a READ KEY AND DATA command, it may be possible for ATLAS to assign an alternate track. In such a case, the IEHATLAS utility program can be executed as a job step and will issue the SVC 86 instruction to have the ATLAS routine assign an alternate track. The utility's channel program will be used to replace the error record. (This procedure can be followed if a user can reconstruct the error record and present it as input to the utility program.)

In general, whenever the utility program cannot successfully perform an "update-in-place," it will invoke ATLAS to assign an alternate track and take responsibility for data transfer to that track.

Figure 141 on page 394 indicates the structure, overall flow of control, and the information flow of the IEHATLAS utility program and ATLAS. "Chart 01. Overall Logic Flow" on page 418 shows the overall logic of the recover/replace function. The channel programs that are used by the utility program and ATLAS are shown, functionally, in Figure 152 on page 417, which precedes the group of flowcharts for the recover/replace function. The following discussion of the load modules for the utility program and ATLAS goes into more detail for some of these channel programs.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

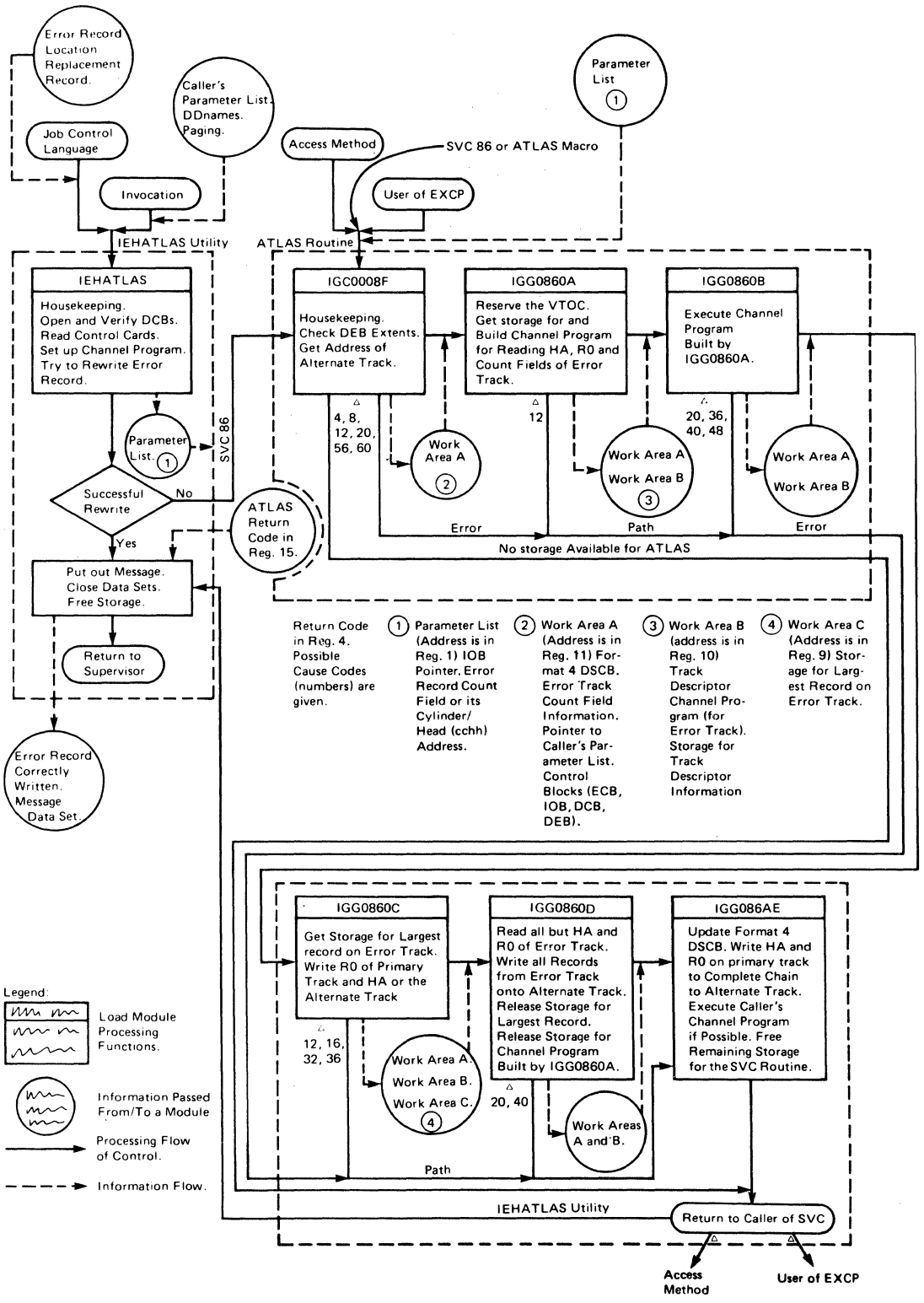


Figure 141. Overview of IEHATLAS Utility Program and ATLAS Routine

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

THE IEHATLAS UTILITY PROGRAM

The utility program tries to perform an update-in-place function. (See "Chart 02. Utility Logic Flow" on page 419.) The program may be executed either as a job step (by using the name IEHATLAS as the program name parameter in an EXEC statement) or as a part of a program that is performing a job step. In the latter case, the program is invoked by using the name IEHATLAS in the operand of a LINK, LOAD, XCTL, or an ATTACH macro instruction.

When used as a job step, the IEHATLAS program requires input in the form of control cards containing data set definition statements, error track and record information (for example, record type, location, length) and replacement data cards (in hexadecimal) for the error record. When invoked as part of an existing job step, the input consists of a parameter list (pointed to by general register 1) containing a starting page number and possible overriding names for the input, output, and control data sets.

The utility program consists of two control sections (CSECTs): IEHMESS, which contains the messages for the program, and IEHATLAS, which contains the functional instructions for the program. The IEHATLAS CSECT processes the job control language (JCL) cards, opens (and later closes) data sets containing information required during processing, builds and uses its own channel program to try to rewrite (with read-back checking) the error record, and provides an interface with the ATLAS routine if required. In addition, the IEHATLAS CSECT provides linkage with the IEHMESS CSECT, through use of the PUT macro instruction, to place a message on the output device.

When the utility program uses the ATLAS routine, it receives a return code from ATLAS indicating which message is to be put out to signify the type of completion for the program. When ATLAS returns the parameter list that identifies records not correctly written on the alternate track because of an abnormal termination, IEHATLAS prints the list with the completion message.

PROGRAM FLOW

Upon entry to the IEHATLAS utility program, standard linkage conventions are used to permit the program to return to the supervisor when it is done. The program establishes a work area in main storage to contain parameters from the input control cards or from an input parameter list if the program is invoked. The work area also contains the following:

- The data control blocks (DCBs) for the input, output, and control data sets
- The job file control block (JFCB) for the input data set
- The event control block (ECB) and the input/output block (IOB) for the utility
- The channel program that the utility builds to rewrite the error record
- The parameter list that the utility gives to ATLAS if the utility issues the SVC 86 instruction

Figure 142 on page 396 contains a layout of the utility program's main work area. In this figure, the fields previously listed and several others have been indicated.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

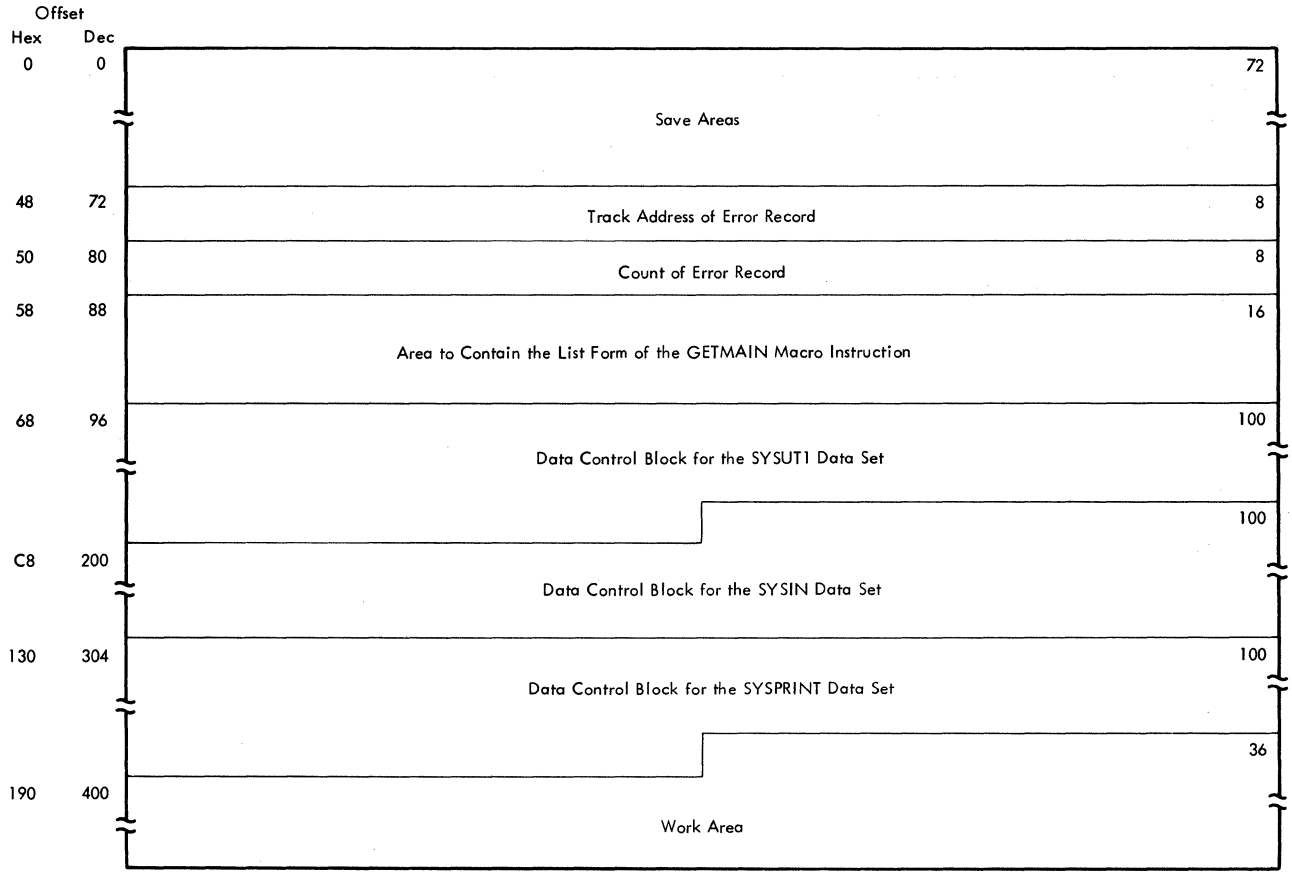


Figure 142 (Part 1 of 3). Main Work Area for IEHATLAS Utility Program

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

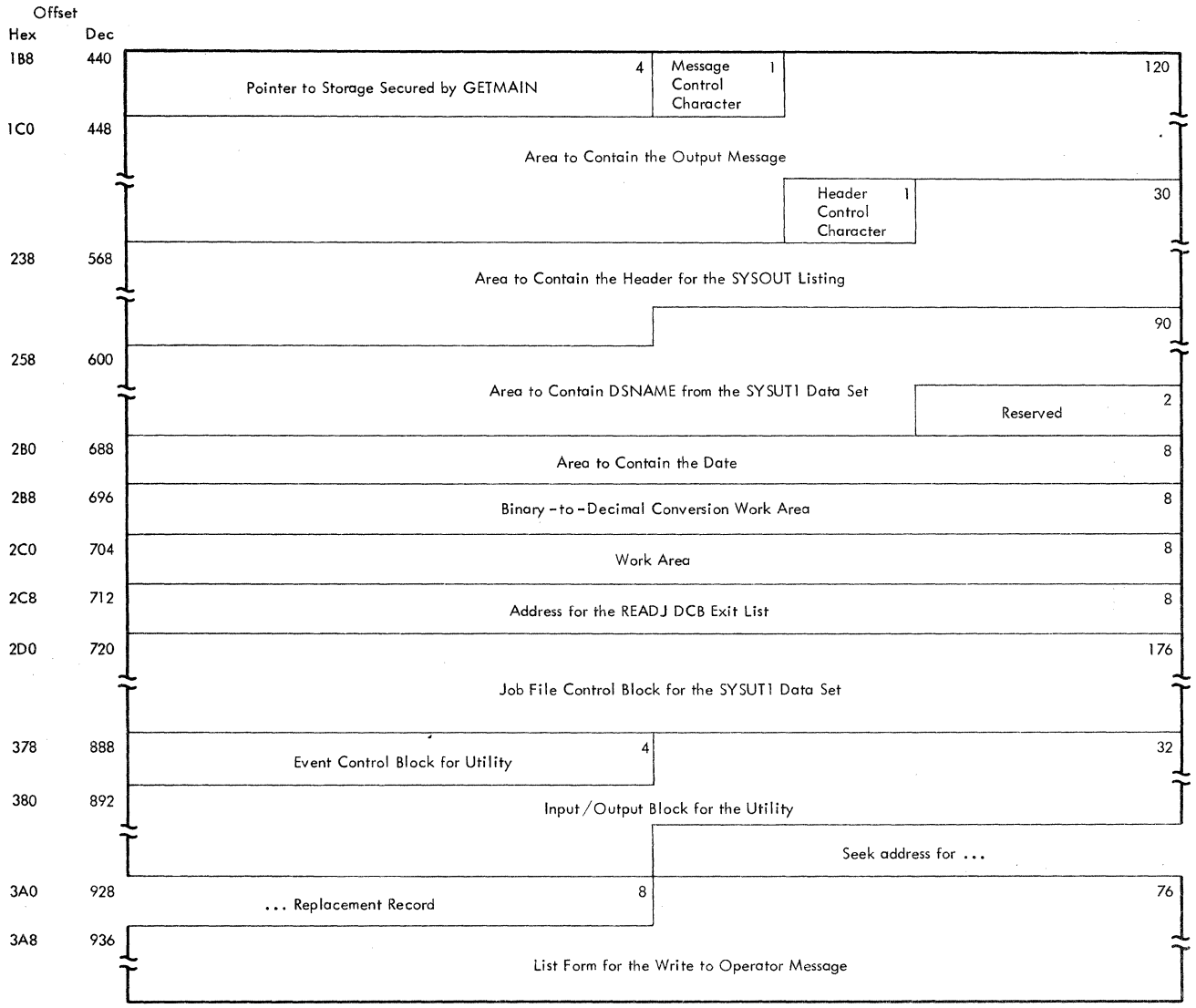


Figure 142 (Part 2 of 3). Main Work Area for IEHATLAS Utility Program

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

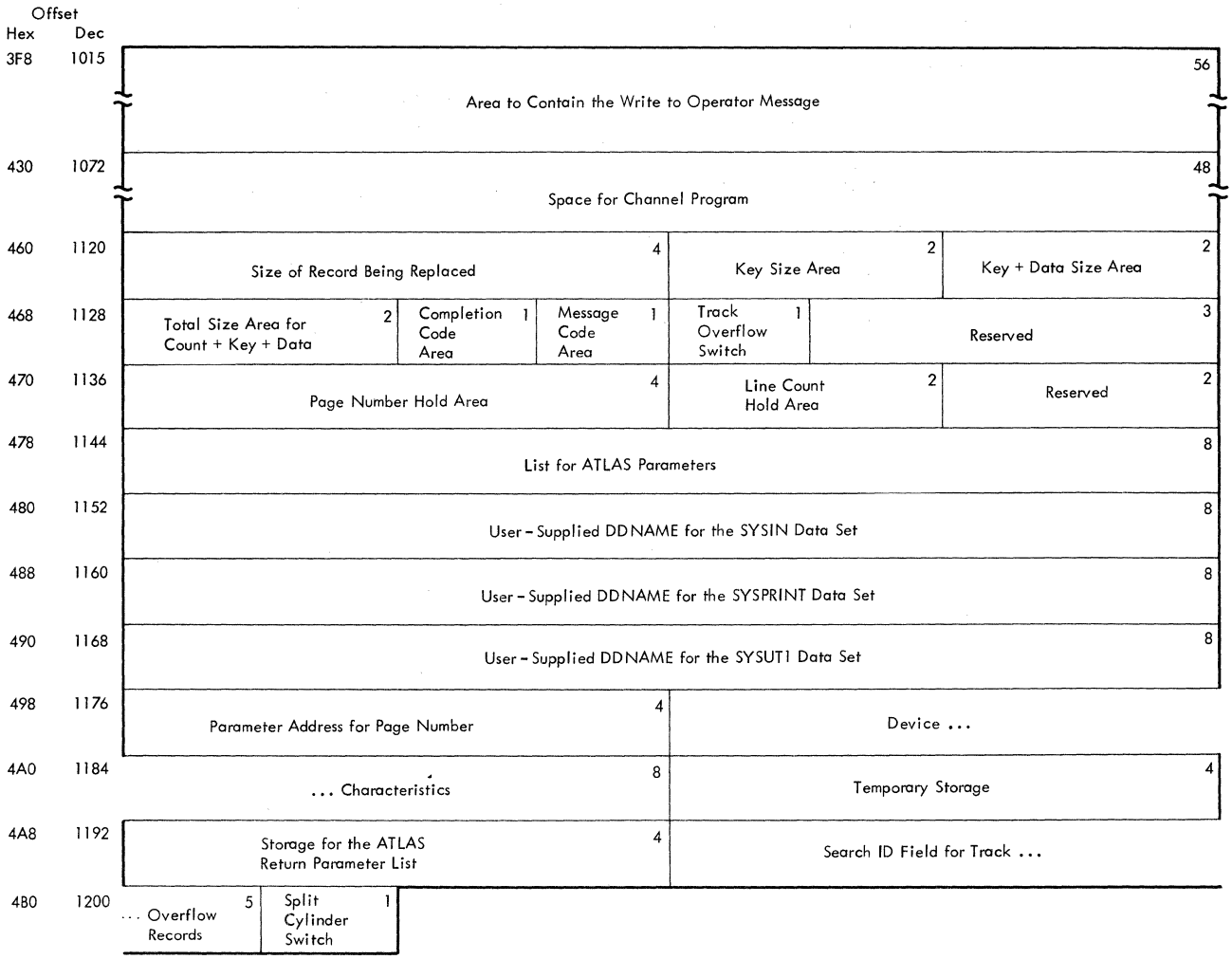


Figure 142 (Part 3 of 3). Main Work Area for IEHATLAS Utility Program

If the utility program has been invoked, it determines if any ddnames (SYSPRINT, SYSUT1, SYSIN) have been overridden. If so, it places the specified names in the work area. If a page number has been specified, it converts it to a hexadecimal value and places it in the work area. (This information is in the parameter list that the caller supplies at the time the utility program is invoked.)

The IEHATLAS utility program requests the opening of the SYSPRINT data set for QSAM by using the list and execute forms of the OPEN macro instruction. The open routine takes an exit (a DCB exit routine) to check on the validity of the DCB parameters. The block size parameter must be a multiple of the logical record length parameter if the utility program is to continue.

If the data set is not opened successfully, the program sets a completion code, releases the storage obtained for the work area, and returns control to the supervisor.

If the data set is opened successfully and the block size and logical record length DCB parameters are valid, the utility constructs page headers, determines the date and time, and

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

constructs the header record for the output. (The header record for each page contains the name of the utility, the name of the SYSUT1 data set [that is, the DS name], the current date and time, and the page number.)

The utility program issues another OPEN macro instruction for the input data set (which either is named SYSIN or has a user-supplied name), and a DCB exit routine checks the validity of parameters in the same manner as for the SYSPRINT data set. (Because the logical record length of the input data set is 80, the block size parameter must be a multiple of 80.)

After opening the input data set, the utility issues the RDJFCB macro instruction for the SYSUT1 data set, which contains the error record. This macro instruction always moves the JFCB for the SYSUT1 data set from auxiliary storage to main storage and is used for subsequent opening of the data set if needed—it is needed if the SYSUT1 data set is the volume table of contents (VTOC). IEHATLAS takes the SYSUT1 data set name from the JFCB and places it in the work area as a part of the header record that it builds there. When the header record is complete, the program places it on the SYSPRINT (output) data set.

If there is an error caused by a missing input (SYSIN) DD card, or if an error is encountered when checking the validity of the DCB parameters during the DCB exit routine, the utility program places the appropriate message on the SYSPRINT data set and terminates processing. Termination procedures consist of:

- Closing the input (SYSIN and SYSUT1) data sets (if they have been opened)
- Releasing buffer storage obtained for the error record
- Closing and releasing storage for the output (SYSPRINT) data set
- Releasing storage assigned to the main work area for the utility

After successfully opening the SYSIN data set, the module reads a control card from SYSIN into storage, and checks for a valid card format and keyword. The module then determines and obtains the storage space required to contain the replacement record, and moves the record into the area. (If the control card was invalid in any way, the module flushes the input data set cards through the input device to the end-of-data condition. It writes a message and terminates the processing.)

The utility program converts the data on each of the input cards from 80 bytes of card data to 40 bytes of internal machine data, and checks the validity of the input data characters. The data cards are read in and checked one by one. If the key and data lengths given on the control card do not agree with the lengths of the key and data information representing the replacement record, or if there are too many control cards, the appropriate message is given and the module terminates the processing.

Figure 143 on page 400 contains the symbolic form of the channel program used by the IEHATLAS module for writing the replacement record on the defective track. This channel program is a part of the IEHATLAS CSECT that is loaded into storage before the utility program receives control.

The utility program places the addresses of the first channel command word, the ECB, and the DCB (for SYSUT1) into IOB and opens the SYSUT1 data set.

The UCB is checked for a virtual volume or a virtual device. If either is found, the job step is terminated, and message IEH924I is issued with a return code of 16.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Channel Command Word	Action Performed
SEARCH ID EQUAL	Find record to be replaced on track.
TIC (Transfer-in-channel)	Loop to search until record is found.
WRITE KEY DATA	Write record from storage onto defective track.
SEARCH ID EQUAL	Find the record just written.
TIC	Loop to search until record is found.
READ KEY DATA	Read record just written. Do not place data in storage (i.e., perform a read-back check function).

Figure 143. Channel Program to Write Replacement Record

IEHATLAS then attempts to write the replacement record onto the defective track. The program indicates a successful write operation by placing the appropriate completion message on the SYSPRINT device and terminating the processing.

Note: If the user has indicated that the error record is the VTOC, the utility program attempts to read the record. For a successful read operation, a message is written informing the user that there is no error, and the utility program terminates processing. When the "read" operation results in an error, the VTOC record is handled as any other indicated error record.

If the indicated error record is an end-of-file record (has a key and data length of zero), the utility program tries only to read the record. The system does not process a write key and data command when the key and data lengths are zero.

If the operand for the TRACK keyword contains the write special indication, IEHATLAS uses a write special count, key and data command to rewrite the record.

When the channel program cannot successfully rewrite the replacement record, IEHATLAS constructs a parameter list (see Figure 144 on page 401) containing the following information for the alternate track assignment (ATLAS) routine:

- A pointer to the IOB (from which the beginning of the channel program may be found)
- A pointer to the count field of the error record

The utility program then issues the SVC 86 instruction to obtain an alternate track and have all records from the error track written on the alternate track.

Upon return from ATLAS, the IEHATLAS utility uses the completion code and message code sent by ATLAS to indicate the outcome of the alternate track assignment function. IEHATLAS prints the appropriate message and, if present, the ATLAS return parameter list. Then it terminates processing and returns control to the supervisor.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

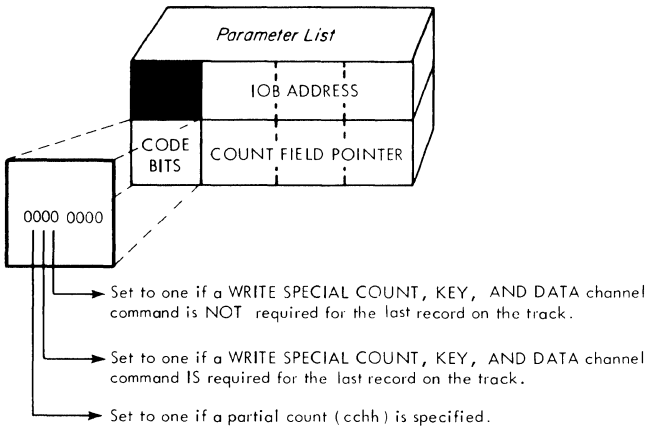


Figure 144. Parameter List for ATLAS (SVC 86)

THE ALTERNATE TRACK LOCATION ASSIGNMENT (ATLAS) ROUTINE

A calling program issues the SVC 86 instruction or the ATLAS macro instruction to assign an alternate track upon which to write a record that has been unsuccessfully read or written by that calling program. Both of these instructions result in control being given to the ATLAS routine. This routine consists of six load modules that are loaded into storage one at a time and given control in a sequential manner. (See Charts 03, 04, 05, 06, and 07.) Exceptions to this sequence are the following:

- When an error occurs in the processing by one of the first five load modules, the module may transfer control to the sixth load module. The sixth load module will give control to the calling program (or caller).
- The fourth and fifth load modules may transfer control between each other in the search for an alternate track.

ATLAS is normally used by a caller's SYNAD routine when an error occurs during an I/O operation. ATLAS requires input from the caller that gives both the status of the caller's channel program and an auxiliary storage address. The two-word parameter list that includes this information points, as Figure 4 on page 5 shows, to the IOB for that channel program and also indicates its status by the sign of the word containing the IOB address. When the user sets the high-order bit to 0, the pointer is plus, and the channel program can be reexecuted. The address included in the parameter list may point to the whole count field (CCHHRKDD) of the error record, or it may be merely the cylinder and track (CCHH) address that designates the track on which the error record is located on the direct-access device. (The use of the code bits shown in Figure 144 is explained in the description of load module IGG0860C.)

ATLAS performs its processing in slightly different ways depending on the state of the caller's channel program being executed when the initial read/write error occurred. If the channel program can be reexecuted, ATLAS assigns an alternate track and writes everything from the defective track to the alternate track. The caller's channel program is then reexecuted to write the error record.

If the channel program cannot be reexecuted, ATLAS assigns the alternate track and writes all records it is able to. When the caller's channel program has passed ATLAS the full count field

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

(the full CCHHRKDD) of the defective record, ATLAS can successfully transfer that count field to the alternate track on the initial try. Otherwise, ATLAS attempts recovery from count field errors and writes the data onto the alternate track exactly as it read it from the defective track. If ATLAS is unable to complete recovery successfully, the caller has the option of using the IEHATLAS utility program as a separate job step to write the error record correctly onto the alternate track that ATLAS assigned.

In certain cases of abnormal termination, ATLAS returns a parameter list containing the record numbers of the error records that cannot be written because of the abnormal termination. This parameter list, shown in Figure 145, consists of 4 bytes returned in general register 0 to identify a maximum of three such error records. Because ATLAS stores the numbers of error records it encounters while reading a track as it encounters them, the numbers fall in left-to-right order.

Subsequently, they appear in the register in the same order. Therefore, the rightmost R entry in the register denotes the error record that caused the abnormal termination, provided that ATLAS encountered no more than three records having errors.

This parameter list may also denote records that ATLAS did write on the alternate track. In this special case, where the parameter list accompanies a return code of 40, there was no abnormal termination, but the records listed had errors in their key-data fields.

In any case, the parameter list, if present, is printed with the ATLAS return code by the IEHATLAS utility program.

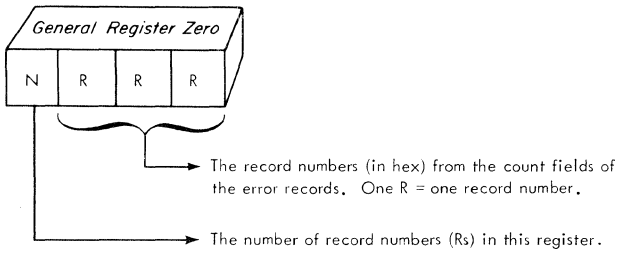


Figure 145. Return Parameter Register 0 from ATLAS (SVC 86)

LOAD MODULE IGC0008F

All entries to ATLAS are made to this, the first, load module (see "Chart 03. ATLAS Modules IGC0008F and IGG0860A" on page 420). The prime functions of the module are to determine if the caller's channel program is reexecutable and to ascertain whether ATLAS can attempt to correct the indicated error. The module also reads in the VTOC (Format-4) DSCB of the volume's table of contents (VTOC) and acquires information regarding the availability of alternate tracks for the volume.

When module IGC0008F is entered, it obtains a work area. The work area contains information that will be used by this and subsequent modules. It is passed from one module to the next throughout ATLAS's six load modules and includes the following information:

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- The volume's VTOC (Format-4) DSCB
- The address of the caller's parameter list
- The maximum number of count fields on a track for the device being used
- The control blocks (ECB, IOB, and DCB) constructed by this module

After determining the status of the caller's channel program, the module compares the cylinder and track (CCHH) address pointed to by the parameter list with the DEB extents. It does this to verify that a track was specified within the extent of the user's own data set. The module then checks the sense bytes in the IOB to determine the type of error that led to the calling of ATLAS.

The following contains the byte and bit values of the IOB sense bytes, which are indicative of errors that ATLAS may be able to correct:

Sense Byte	Bit No.	Indication (When bit is '1')
0	4	Data check in other than count field of record
1	0	Data check in count field of record
1	6	Address marker is missing

Either of the following indications occurring with bit 6 of byte 1 indicates an unacceptable error.

Sense Byte	Bit No.	Indication (When bit is '1')
0	7	Unable to complete a seek.
1	4	No data (record) has been found on the track in question.

If the error is a type that ATLAS may be able to correct, the module checks the device on which the error occurred to see if it is one (for example, a 3330) that contains software-assignable alternate tracks. It then determines the maximum number of records on a track and the original number of alternate tracks available for the device. The control blocks (DEB, DCB, IOB, and ECB) are initialized for the channel program to read the VTOC (Format-4) DSCB for the volume containing the error record.

The module then gets a conversion routine address from the communication vector table and uses the routine to convert the relative track address of the VTOC (found in the UCB) to an absolute address. Module IGG0008F then gives control to the second load module, IGG0860A.

LOAD MODULE IGG0860A

This module (see Chart 03) receives control from IGC0008F. The main functions of the module are the following:

- It reserves the VTOC for the exclusive use of the ATLAS routine.
- It reads the VTOC (Format-4) DSCB for the volume containing the error record into main storage.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- It computes the amount of storage needed for a track descriptor channel program and for the information that the channel program gathers.
- It generates the channel program itself.

ATLAS uses the information from the VTOC (Format-4) DSCB to identify the first available alternate track and to determine how many alternate tracks are available. Then it uses the channel program to gather descriptive information about the track containing the error record.

To secure the VTOC (Format-4) DSCB until it can be read and updated, and to have exclusive use of the VTOC, the module first issues a RESERVE macro instruction to enqueue the VTOC. (In a multiprogramming environment, the module must be able to rely on and to modify, without interference, the contents of the VTOC as it transfers control from one load module to another. ATLAS dequeues the VTOC only after having prepared to return control to the caller in the sixth, and last, load module.)

After enqueuing the VTOC, IGG0860A reads the VTOC (Format-4) DSCB into storage by using a following channel program consisting of the following channel command words (CCWs):

READ HOME ADDRESS	of the VTOC track containing the Format-4 DSCB
READ COUNT	of the Format-4 DSCB
READ DATA	of the Format-4 DSCB

If the caller's indicated error record happens to be the VTOC (Format-4) DSCB, its information about the availability of alternate tracks cannot be reliably obtained. In this case, the module sets an error return code and gives control to the sixth load module. Otherwise, it makes a determination of the number of alternate tracks available for ATLAS and stores the number in the work area.

The information gathered by the track- descriptor channel program that IGG0860A generates includes the home address (HA) and record zero (R0) of the track and all the count fields for the records on the track. Collectively, this information constitutes a "track-descriptor" field, shown in Figure 146.

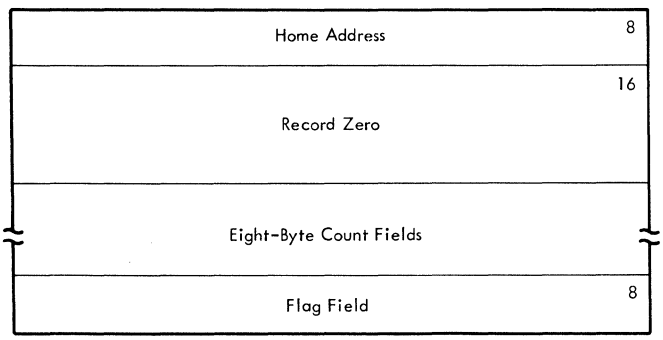


Figure 146. Track-Descriptor Field

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

The module computes the storage required for the track-descriptor channel program on the basis of 8 bytes for each CCW plus space for the track-descriptor field. After obtaining the storage area, the module initializes it by clearing all its fields to zero.

If the error record is record R0 and the caller (of ATLAS) is other than the IEHATLAS utility program, ATLAS is unable to carry out its function, because the data field of that record must be reliable should it be needed. If the utility program is the caller of ATLAS, the replacement record (in this case, R0) is available for a rewriting attempt by ATLAS in a later module.

In building the channel program for the track-descriptor record, the module loads into the work area the channel command words for reading the HA and R0 and places the address of the first CCW in the program in the IOBSTART field of the IOB. The CCWs for the read count commands are then set up to follow the read R0 command.

Note: The channel program contains read count commands, with the multitrack bit set to '1', for one more than the maximum number of records on the track. This ensures that an out-of-extent condition will be indicated when the channel program executes the last read count command for the track. The storage areas for the actual count fields follow the read count commands in the work area.

Thus, in determining the address of the area into which the first count field is to be read, the number of CCWs required to read all count fields on the track is multiplied by eight so as to skip the space required for the CCWs. For all but the last read count command, the command chaining bit is set to '1' to cause the count fields to be read in sequence. For the last read count command, this bit is turned off to signal the end of the channel program. The module then gives control to the third load module.

LOAD MODULE IGG0860B

This module (see Chart 04) receives control from module IGG0860A. Its primary function is to execute the channel program constructed by module IGG0860A to read in the track-descriptor record. If, in executing this channel program, an error occurs in reading any count field, the channel program is modified to read the actual key and data of the record for which the count field reading error occurred. The count field of a record contains information about the key and data field lengths of the record.

If the channel program comes to an end because of an end-of-extent condition, which signifies that all the count fields have been read successfully, this module gives control to the fourth load module. However, because ATLAS's channel program is but a part of the overall I/O program that is executed when the EXCP macro instruction is issued, it may be that the channel program has come to a stop at some CCW that does not "belong" to ATLAS. The storage limits of the space needed for ATLAS's channel program are the address of the first CCW given in the IOBSTART field and the storage address reserved for the first count field. Module IGG0860B checks these addresses against the address of the CCW during which the error occurred. If ATLAS's original channel program has been modified, the module also checks the limits of the modification to the channel program.)

If the I/O program has come to a stop because of an error occurring outside of ATLAS's part of the channel program, the ATLAS routine is terminated by taking an exit to the sixth load module, which performs cleanup procedures and returns control to the caller of ATLAS.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

If the primary track is defective and already chained to an alternate track, the address of the alternate track is moved to the seek address for the channel program, and the extents in the DEB are changed to reflect the alternate track. The module then reexecutes (via an EXCP macro instruction) the channel program to obtain a description of the HA, R0, and count fields of the defective track.

If the channel program stopped because of a data check error or a missing address marker and the error occurred in a read R0 command for an error record indicated by the IEHATLAS utility program, ATLAS replaces the read R0 channel command with a read HA command and reexecutes the channel program to skip over the record R0. In this case, the track-descriptor record will be lacking the R0 information, but, because the utility program requires both the count field (either in a parameter list or on a control card) and the replacement record as input, the recovery function may still be completed.

If an error is detected on reading one of the count fields of the error track, a test is made to determine if the track record causing the channel program to stop was an end-of-file (EOF) record as indicated by a data-length value of zero. If it is, and if either

- The record is not the error record indicated by the user, or
- The full count field (CCHHRKDD) is not available to SVC 86,

then ATLAS will terminate processing. If the IEHATLAS utility program has issued the SVC 86, the key and data length values are also available from the replacement record specifications. For an error record other than the one specified by the user, the data that has been read into storage is unreliable as a result of the data check or missing address marker condition for which the channel program has been terminated.

If the record is not an EOF record or if ATLAS does not terminate the processing at this point, module IGG0860B builds a short modification to the channel program to bypass the count field in question. The module also saves the record number for possible later uses in the return parameter list (Figure 145 on page 402) provided in case of an abnormal termination. The channel program (with the "bypass" modification) is then reexecuted.

In building the channel program modification, there are two situations that may arise: (1) an error-causing read count command may occur between two error-free read count commands and (2) two or more error-causing read count commands may be adjacent to one another. Figure 147 on page 407 illustrates the first situation, and Figure 148 on page 407 illustrates the second.

In Figure 147, the third read count command is indicated as causing an error. This command is replaced by a TIC (transfer-in-channel) command that contains the address of the first command in the first modification to the channel program. In the modification, the address field of the space count command contains the address of the key and data portion of the count field in error. The read key and data command contains the key and data lengths from the count field in error, and the command chaining bit is set to '0' so the channel program will halt after executing that command. The third command (TIC) in the modified channel program is retained for use, by module IGG0860D, as a return pointer in the channel program for reading the record into a buffer in main storage. After executing the modified channel program, module IGG0860B changes the read key and data command to a TIC command (second modification). This TIC command (called a return TIC command) contains the address of the read count command that follows the one being executed when the current error occurred.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

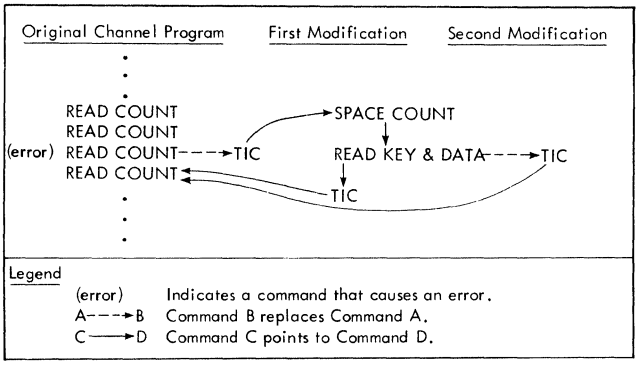


Figure 147. Read-Count Channel Program Modification for a Single Error-Causing Read Count Command

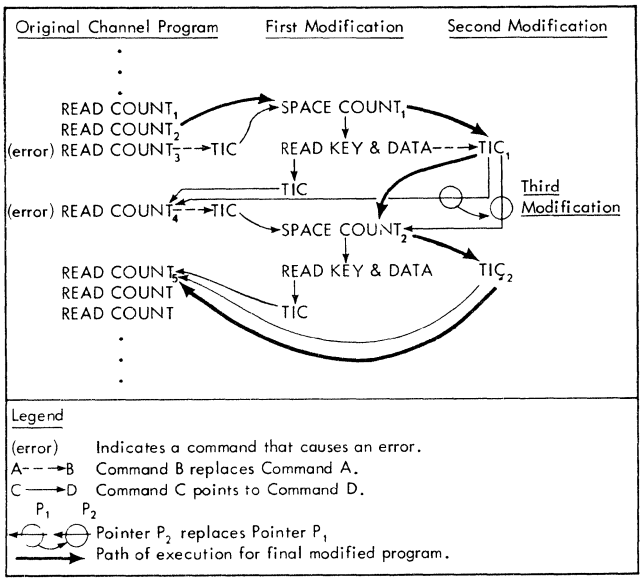


Figure 148. Read-Count Channel Program Modification for Two Adjacent Error-Causing Read Count Commands

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

When module IGG0860B encounters an error in executing a read count command, it determines if a modified channel program has been constructed to replace the preceding read count command in the original channel program. If a modified program has been constructed, module IGG0860B makes another modification as indicated in the following text.

In Figure 148, the execution of both the third and fourth read count commands has resulted in errors. The first and second modifications shown for these commands have the form illustrated in Figure 147. However, in Figure 148, the address portion of the return TIC command (TIC 1) in the modification for command read count 3 now indicates the address of command space count 2. This address change is indicated as the third modification in Figure 148.

Figure 148 also shows the final composite channel program that is executed after the modifications have been completed for read count command.

If more than three count fields (including that of the error record) from the error track are read with an error, the program will not attempt recovery. In this case, the module sets an error message code and starts termination procedures by giving control to the sixth load module. Otherwise, module IGG0860B gives control to module IGG0860C.

LOAD MODULE IGG0860C

This module (see Chart 05) receives control from either module IGG0860B, in the case of the first entry to the module during normal processing flow, or module IGG0860D, when another alternate track (other than the one already assigned by this module) must be assigned. The main functions of this module are to get sufficient storage to contain the largest record on the defective track and to write the home address and record zero on the selected alternate track.

However, this module transfers control to the sixth load module without executing its main functions if there has been an error in reading the count field of the last record on a track, and if either of the following is true:

- The IEHATLAS utility program was the caller of ATLAS, and the last record on the track is not the error record that the user indicated.
- The DCB indicates that the data set containing the error record contains track overflow records, and the user has not stipulated whether or not ATLAS requires a write special count, key and data channel command.

The user passes ATLAS a parameter list (Figure 144 on page 401) whose code bits indicate whether or not the routine will require the "Write Special ..." channel command. The user can specify one of three options with respect to the "Write Special ..." channel command:

Option One: User specifies that ATLAS will require a "Write Special ..." channel command (by setting code bit 2).

Option Two: User specifies that ATLAS will not require a "Write Special ..." channel command (by setting code bit 3).

Option Three: User does not specify whether or not ATLAS will require a "Write Special ..." channel command (by setting neither code bit 2 nor code bit 3).

In all cases in which track overflow data sets are not involved, the user makes simple use of option three; that is, there is no need to be concerned with whether or not ATLAS will need to use a "Write Special ..." channel command, because the routine cannot encounter any track overflow records.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

However, when the module determines by checking the DCB that the data set contains track overflow records, it looks at the user's parameter list to determine whether or not ATLAS requires a "Write Special ..." channel command to write any last record that overflows to the next track. This check demands that the user explicitly specify either option one or option two by setting a bit in the parameter list. The user must know (for example, by consulting a backup copy such as an earlier listing) where track overflow records occur in order to be able to specify the correct option. Any decision must be based on the consideration that ATLAS can correctly write track overflow records only by using "Write Special ..." channel commands. When the user passes ATLAS the correct code bit pattern, the module pursues its normal course of processing, described below, and ATLAS recovers the error record correctly, regardless of whether or not it is a track overflow record.

The user's failure to be explicit when the module is in a track overflow environment, however, causes ATLAS to bypass further processing if it encounters an error in the count field of the last record on a track. Instead, the module takes the error exit to the sixth load module.

Normally, the module gets storage for the largest record on the error track. The key length and data length fields of the first count field are added together and placed in a register as a "maximum" value. A comparison is made, in turn, with the sum of the key and data lengths for each record on the track. Each time a sum is higher than the current maximum value in the register, it becomes the new maximum value. A GETMAIN macro instruction is issued for space to contain the final maximum value (in bytes) arrived at in the comparison loop, plus 8 bytes for a count field.

The starting cylinder and ending cylinder addresses in the DEB are set to reflect the alternate track. The module then inserts parameters in its channel program to search for the home address and to write (with read-back check) the HA and R0 on the alternate track.

If record R0 is the indicated error record and was passed to ATLAS by the IEHATLAS utility program, this module constructs a temporary record R0, in which the content of the data field is zero, for the alternate track. The actual data for the error record replacement is available to the utility program when it first receives control.

After the alternate track address has been obtained (from the DS4HCCHH field of the format-4 DSCB) and the channel program has been completed, module IGG0860C executes the channel program to partially chain the alternate track to the primary track. (The sixth load module completes the chaining.) The HA and R0 records written on the alternate track contain the chaining information.

If there is an error in this I/O operation, the module determines which CCW was being processed when the error occurred. If the error was not during a search HA command, module IGG0860C attempts to select another alternate track and reexecute the channel program. If another alternate track is available, the module updates the format-4 DSCB (read in by module IGC0008F) accordingly, resets control blocks, and alters the DEB extents and the track seek address to reflect the new alternate track. The channel program is then reexecuted to write the HA and R0 on the next alternate track. When there are no more alternate tracks available, or if there is an I/O error during a search HA command, the module exits to the sixth load module.

If the I/O operation has concluded successfully, module IGG0860C gives control to the fifth load module, IGG0860D.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

LOAD MODULE IGG0860D

This module (see Chart 06) receives control from module IGG0860C after the HA and the R0 have been written on an alternate track. The primary function of module IGG0860D is to transfer data records from the defective track to the chosen alternate track.

A pass through module IGG0860D is made each time a record from the error track is to be placed on the alternate track. The module first reads a record into a buffer area and then writes it, with a read-back check for verification, onto the alternate track. If a data check error or a missing address marker error occurs when reading into the buffer, the module may make a modification to the "read" command in the original channel program before setting up the "Write" channel program.

During a given pass, the module always transfers data to an alternate track exactly as it reads it into main storage from the defective track. When the particular record transferred is the user's indicated error record, it is the user's responsibility to rewrite the error record if the channel program (during which the original I/O error occurred) is not reexecutable. Although the IEHATLAS utility program is always reexecutable, the user's channel program is reexecutable only when it is specified to be so. (See the discussion of the sixth load module, IGG086AE.)

For each pass through this module, the module uses the record's count field to compute the size of the record. (An end-of-file condition is indicated when the record's data length value in the count field is zero.) As each record is being considered, the associated CCW in the read-count channel program (built in the second and third load modules) is inspected and modified as necessary to permit reading the key and data for that record.

As an example, consider a channel program (constructed and modified by modules IGG0860A and IGG0860B) consisting of enough read count commands for a track with six records beyond record R0. Figure 149 on page 411 illustrates how this program and the necessary space to contain information read by the program would appear, symbolically, in storage. The subscripts 1 to 6 refer to the six records on the track. Subscript 7 refers to the additional command to cause an out-of-extent interruption when the channel program attempts to read the count field for a nonexistent seventh record.

Assume the original channel program's read count commands for records 2 and 5 resulted in correctable error conditions. In each of these cases, module IGG0860B replaced the read count command with a TIC command, which pointed to a modification consisting of three channel command words (illustrated as Modification 1). Further, as module IGG0860B successfully executed these commands, the read key and data commands were changed to TIC commands, shown as Modification 2 in the figure.

The occurrence of a TIC command in the read-count channel program indicates to module IGG0860D that a count field error occurred in executing the program in the third load module, and that a modified channel program has replaced the original program.

Module IGG0860D then moves the return TIC command (or, if a count field error occurred during execution by module IGG0860B of the succeeding read count command, the TIC command to the space count command for that count field error) from the second position to the third position in the channel program modification (see Figure 147 or Figure 148 on page 407, respectively). The space count command is considered to be in the first position in the channel program modification.

If the command (for a given record) in the read-count channel program is a read count command, the module replaces it with a read count, key and data command, again with the command chaining bit turned off. A read key and data command with the

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

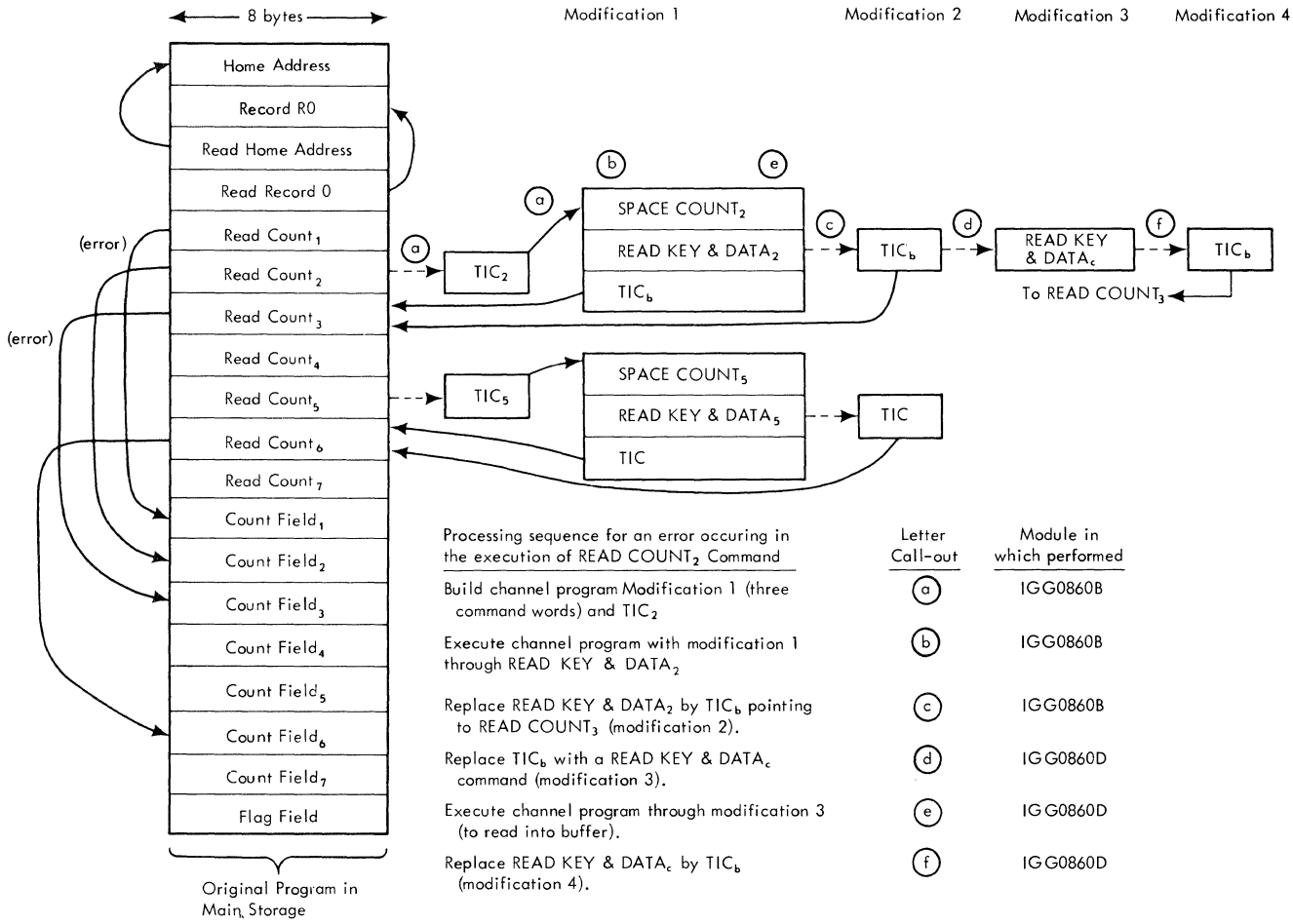


Figure 149. Representation of Read-Count Channel Program and Count Fields in Storage

command chaining bit set to '0' is then placed in the second position. The channel program will stop after the appropriate command (read key and data or read count, key and data) is executed. After placing the starting and ending addresses of the error track in the DEB and clearing fields in control blocks, module IGG0860D executes (via an EXCP macro instruction) the modified read-count channel program to read a record from the error track into a buffer area. Depending on the command, either the count, key, and data portions or just the key and data portions of a record are read into the buffer. If an error occurs while the module reads the key and data portions into the buffer, it saves the error record number for later use in the parameter list that ATLAS prepares.

If a data check or a missing address marker condition causes the channel program to be interrupted and the related "read" command in the original channel program is a read count, key and data

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

command, module IGG0860D rebuilds the channel command word so that it has the read count form it had when the module gained control from module IGG0860C.

If the related "Read ..." command is a read key and data command (in this case, it is part of a modified channel program), module IGG0860D replaces the command with a return TIC command that points to either the next read count command in the original channel program or the space count command of the modification for the next "Read ..." command error.

The count field for the record being written is moved from the count field area of main storage into the buffer area prior to writing the record. If the channel program has used a read count, key, and data command to enter the buffer, and a data check or missing address marker condition occurs, the count field is not in error. (A read count command for the current record had been executed successfully in module IGG0860B.) If the user specified the full count field (CCHHRKDD), but the record in error is not the one so specified, the module sets a switch to indicate that a record other than the indicated error record is incorrect.

If the channel program for reading a record other than an end-of-file record into the buffer area has been interrupted for other than a unit check with a data check or missing address marker condition, this module will give control to the sixth load module, IGG086AE.

IGG0860D checks for an end-of-file (EOF) condition in three situations:

- Upon initial entry to the module, if the key and data lengths are both zero, the module moves the address of the count field, for the current record to be written, from the work area into an output buffer area. The record (containing only a count field) is then written from the buffer onto the alternate track.

- An input/output error may occur when this module reads the current record into an assigned buffer area. If so, the module tests for an EOF condition in which the data length, but not the key length, is zero. If this situation, called a "keyed" EOF condition, exists, the module will write a record consisting of only a count field and a key field.

- After the last record has been successfully written on the alternate track, the module determines if that record was an EOF record (that is, has a data length of zero). If the record is not an EOF record, the module tests for, and if necessary processes for, a record overflow condition as described in "Record Overflow," later in this section.

In order to write a record from the buffer area onto the alternate track, the generalized channel program existing in storage as a part of this load module is adjusted for the record's parameters, which are contained in the work areas pointed to by general registers. The channel program (see Figure 150 on page 413) includes a read-back check feature that is used to verify the results of the write operation. Before using the channel program to write a record onto the alternate track, module IGG0860D sets up the search commands to look for the record number preceding the number of the record that is to be written. (For example, to write record number 3, the search command searches on record 2. To write the first record (record 1), the command searches on record R0.)

Module IGG0860D adjusts the control blocks (inserts the DEB extents, and sets the IOB sense bytes and the ECB to zero) and executes the channel program to write (with read-back verification) the record onto the alternate track. If the channel program is executed successfully, the command-chaining bit and the skip-data-transfer bit in the appropriate "Read ..." command of the original channel program are set to '1'.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

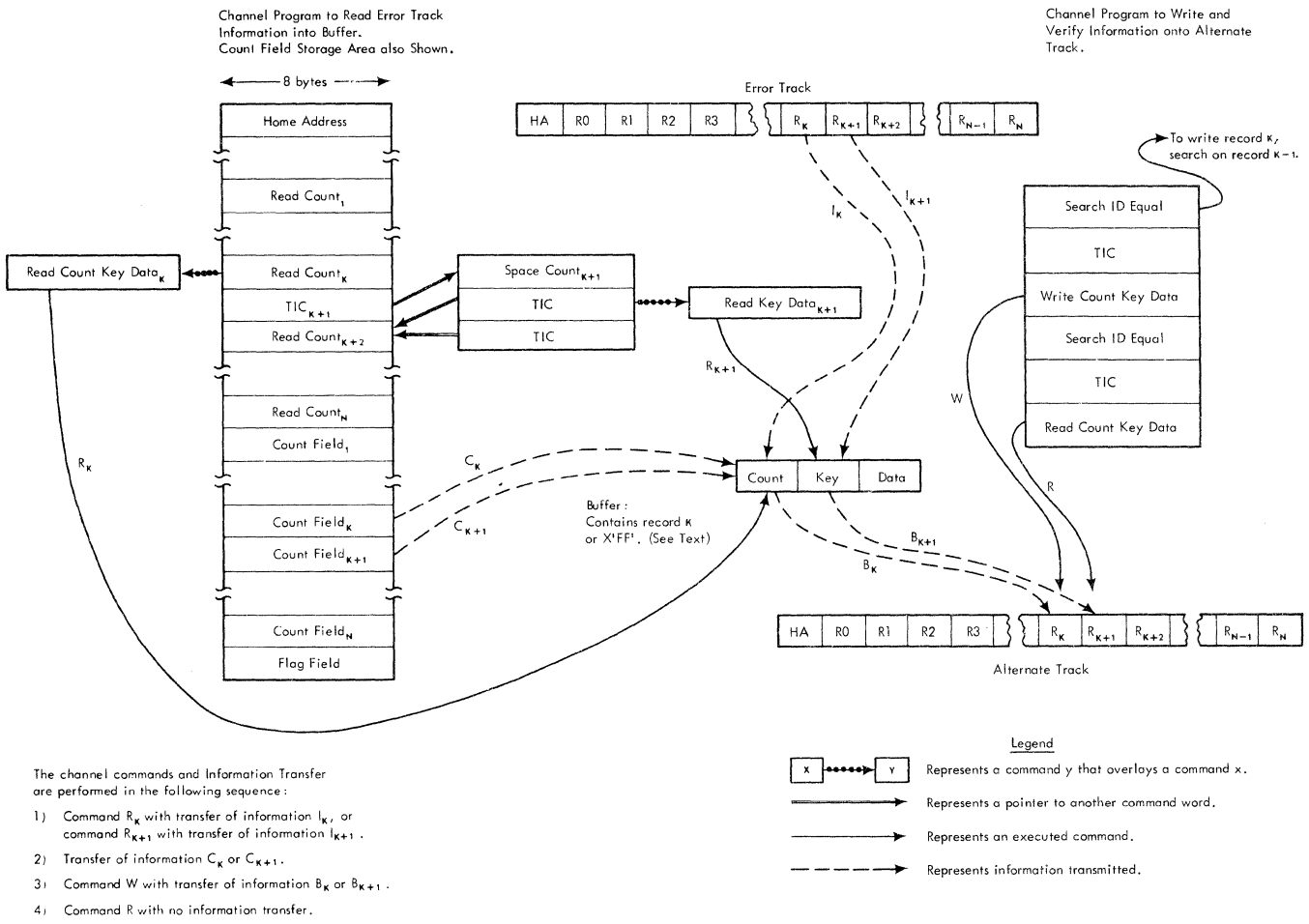


Figure 150. Example of Channel Programs Used by Module IGG0860D to Transfer Information from Error Track to Alternate Track

Note: The command is either a read count, a read count-key-and-data, or a read key-and-data command. This module has used either a read count, key and data, or a read key and data command to attempt to read the current record into the buffer area. However, if an error was encountered in the execution of a read count, key, and data command, the module has changed the command back to a read count command. In this case, module IGG0860B had successfully executed the corresponding read count command (otherwise, a modified channel program would exist), thus giving assurance that the record's count field is available. If an error occurs during the execution of the channel program for writing records onto the alternate track, module IGG0860D returns control to the preceding module to obtain another alternate track.

After successfully writing a record on the alternate track, module IGG0860D determines if the record was the last record on the error track. If it was, the first (or flag) byte in the "count" field following the count field for the record just processed will be X'FF'. (Module IGG0860C has placed the X'FF' in the field as a last-record indicator.) If X'FF' is not in the field, module IGG0860D returns to the beginning of the

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

"Read ..." channel program to obtain the next record. The occurrence of X'FF' causes the module to test the bits previously set for overflow considerations on the basis of the code bits contained in the user's parameter list (Figure 144 on page 401). If these bits indicate that ATLAS will need a write special count, key and data channel command to transfer the track overflow record, then ATLAS can write the record with its track overflow indication onto the alternate track. If these bits do not specify the need for a "Write Special ..." channel command, the module will attempt to force the overflow situation described in "Record Overflow."

If the data length of the last record (that is, the EOF record on the error track) is zero, the record has been written correctly, and the module has finished its record-transferring function.

If the data length of the last record is nonzero, the record may be an overflow record. The module then determines the type of "Read ..." command that was used by the "read" channel program in placing this record in the buffer. If the execution of this "Read ..." command resulted in an error, one of the following actions now occurs before checking for record overflow:

- The associated read count command is replaced with the same read count, key, and data command that was used to read the record into the buffer. (The read count, key, and data command had already been changed back to the original read count command before the possibility of a record-overflow condition arose.)
- The return TIC command that replaced the read key and data command after the execution of a modified channel program for this record is now replaced by the same read key and data command.

If the execution of the read count, key, and data command in the "read" channel program was successful, the command has not been changed.

RECORD OVERFLOW: The writing of any segment except the last segment of an overflow record represents a special situation. The flag byte of the count field of any data record contains information regarding the overflow status of the record. This byte is not read into storage with the rest of the count field; therefore, after reading the last record on the track, and, if the record is not an EOF record, module IGG0860D forces an overflow situation to occur. This situation is necessary because, if the last record on the track is part of an overflow record, the channel program in this module must write the record with the same indication on the alternate track.

Just before forcing the overflow situation, however, the module tests to see if the user stipulated that ATLAS would require a write special count, key, and data channel command to write the current error record. If so, the module skips the code that forces the overflow situation and uses the "Write Special ..." channel command to transfer the error record to the alternate track. Otherwise, it forces the overflow situation as follows:

The key and data length value for the last record is increased by one. The suppress incorrect length (SLI) and the suppress data transfer (SKIP) bits are set to '1'. The module then reexecutes the read count, key, and data channel program for the last record on the error track. Because a larger record is requested than actually exists as the last record on the track, the channel program is requesting the device to switch to the next track.

If the last record actually is an overflow record, bit 1 of the count field's flag byte is set to '1'. The input/output supervisor sets the sense byte IOBSENS1; module IGG0860D then restructures the channel program so that it is a "Write Special ..." program and rewrites the last record with the overflow

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

indicator set on. An I/O error at this point causes a return to the preceding module for another alternate track.

If the last record is not an overflow record, the flag byte's overflow indicator bit is set to '0'. The attempt to switch tracks is therefore not completed. Because the SLI indicator is set to '1', the I/O supervisor does not record an I/O error condition at termination of the channel program.

When the last record on the track has been written successfully, module IGG0860D releases the storage that has been obtained for the record buffer area and for the track-descriptor channel program and count fields. The module sets an indication of any errors in records other than the indicated error record, and gives control to the sixth load module.

When there has not been an I/O error in writing any record but the last record on the track, the command chaining bit and the skip (data transfer) bit for the appropriate read command are set to '1', and the module branches back to get the next record to be placed on the alternate track.

LOAD MODULE IGG086AE

This module (see Chart 07) is the sixth and last module of ATLAS. It is entered either from the fifth module under conditions of normal (error-free) processing or from any one of the first five modules when the particular module encounters an uncorrectable error. The module updates the VTOC (Format-4) DSCB in the VTOC, if necessary, and completes the chaining of the primary and alternate tracks. It then executes the caller's channel program, if possible, to write the error record, releases storage areas used by ATLAS, and returns control to the caller.

If module IGG086AE receives control either normally from module IGG0860D or because of a data transfer error, it must complete the primary-to-alternate track chaining process partially accomplished in module IGG0860C. First, however, it updates the VTOC (Format-4) DSCB so that the VTOC will properly reflect the changes in track availability that result from the chaining process.

To update the VTOC (Format-4) DSCB, the module determines the address of the next available alternate track, based on considerations of cylinder number, and head number. Then it reduces the number of available alternate tracks and completes the chaining process by executing a channel program to write the HA and R0 (containing the alternate track identification) on the original primary track.

If the alternate and primary tracks are successfully chained together, the error record must then be written on the alternate track.

Note: For DOS/VSE compatibility, module IGG086AE sets the key and data length fields of record R0 to 8. The first load module of ATLAS has set a switch (based on the sign of the parameter list pointer to the IOB) to indicate whether the caller's channel program to process the error record is reexecutable. If it is reexecutable (it is always reexecutable if the caller is the IEHATLAS utility program), ATLAS attempts to reexecute it. Otherwise, the caller must take responsibility for rewriting the record.

At this point, the main purpose of ATLAS has been completed, so the module performs the end-of-routine processing.

When this module receives control because ATLAS has unsuccessfully used all the available alternate tracks, it is still necessary to update the VTOC to account for the alternate track defects that ATLAS encountered. Therefore, the module

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

updates the VTOC (Format-4) DSCB and executes a channel program to write it. It then performs the end-of-routine processing.

If this module receives control because of any unrecoverable error, it updates the VTOC, if necessary, and performs the end-of-routine processing.

For end-of-routine processing, module IGG086AE dequeues the VTOC, if necessary, and releases all storage areas that ATLAS has used but has not already released. (For example, the area for the largest record on the defective track may not have been released by the fifth load module, depending on how much of its total processing the module was able to complete.) The module then loads the ATLAS return code (Figure 151) into general register 15 and the ATLAS return parameter list (Figure 145 on page 402) into general register 0, and issues an SVC 3 instruction to return control to the caller by way of the supervisor.

Return Code	Meaning
0	Successful completion; ATLAS has assigned an alternate track.
4	The device does not have software-assignable alternate tracks.
8	All the alternate tracks for the device have been assigned.
12	The requested main storage space is not available.
16	There was an I/O error in the alternate track assignment after N attempts at assignment (where N=10% of the assignable alternate tracks for this device).
20	The error is a condition other than a data check or a missing address marker.
24	There is an error in the Format 4 DSCB that prevents ATLAS from reading it.
28	The user-specified error record is the Format 4 DSCB, which ATLAS cannot handle because the alternate track information is unreliable.
32	ATLAS cannot handle the error found in the count field of the last record on the track.
36	There are errors in the Home Address or in Record Zero.
40	ATLAS found one or more errors in record(s) and assigned an alternate track. 1) There was an error on an end-of-file record. 2) ATLAS encountered an error in the key length-data length portion of the count field. 3) There were errors in more than three count fields.
48	ATLAS found no errors on the track and assigned no alternate track.
52	Because of an I/O error, ATLAS cannot reexecute the user's channel program successfully.
56	The system does not support track overflow.
60	The track address provided does not belong to the indicated data set.

Figure 151. Return Codes from ATLAS

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Main Storage Label	IEHATLAS	IGC0008F	IGG0860A	IGG0860B
CCWAR1	SIDE			
CCWAR2	TIC			
CCWAR3	WKD			
CCWAR4	SIDE△			
CCWAR5	TIC			
CCWAR6	RKD			
CCW1			RHA	
CCW2			RC	
CCW3			RD	
CCW44				
CCW45				
CCW56				
CCW57				SC△
CCW31				RKD, TIC
CCW32				TIC
CCW33				SC
CCW34				RKD, TIC
CCW35				TIC
CCW36				SC
CCW37				RKD, TIC
CCW38				TIC
CCW39				
CCW21			RHA	
CCW22			RR0	
CCW23			RC	
Purpose	To write replacement record on track. △Channel program begins here if record is an EOF record or is the VTOC record.		To read Format 4 DSCB. To get error track description. △See text for description of this program.	

IGG0860C	IGG0860D		IGG086AE	
SHA		SIDE	WHA	SIDE
TIC		TIC	WR0	TIC
RHA		WCKD	RHA	WD
WHA		SIDE	RR0	SIDE
WR0		TIC		TIC
RHA		RCKD		RD
RR0				
	RCKD or RKD			
To partially chain primary track to alternate track. (To write HA & R0 on alternate track.)	To read all records (except HA and R0) from error track.	To write all records on alternate track.	To complete the chaining of primary track to alternate track. (To write HA & R0 on primary track.)	To update Format 4 DSCB and write Format 4 DSCB.

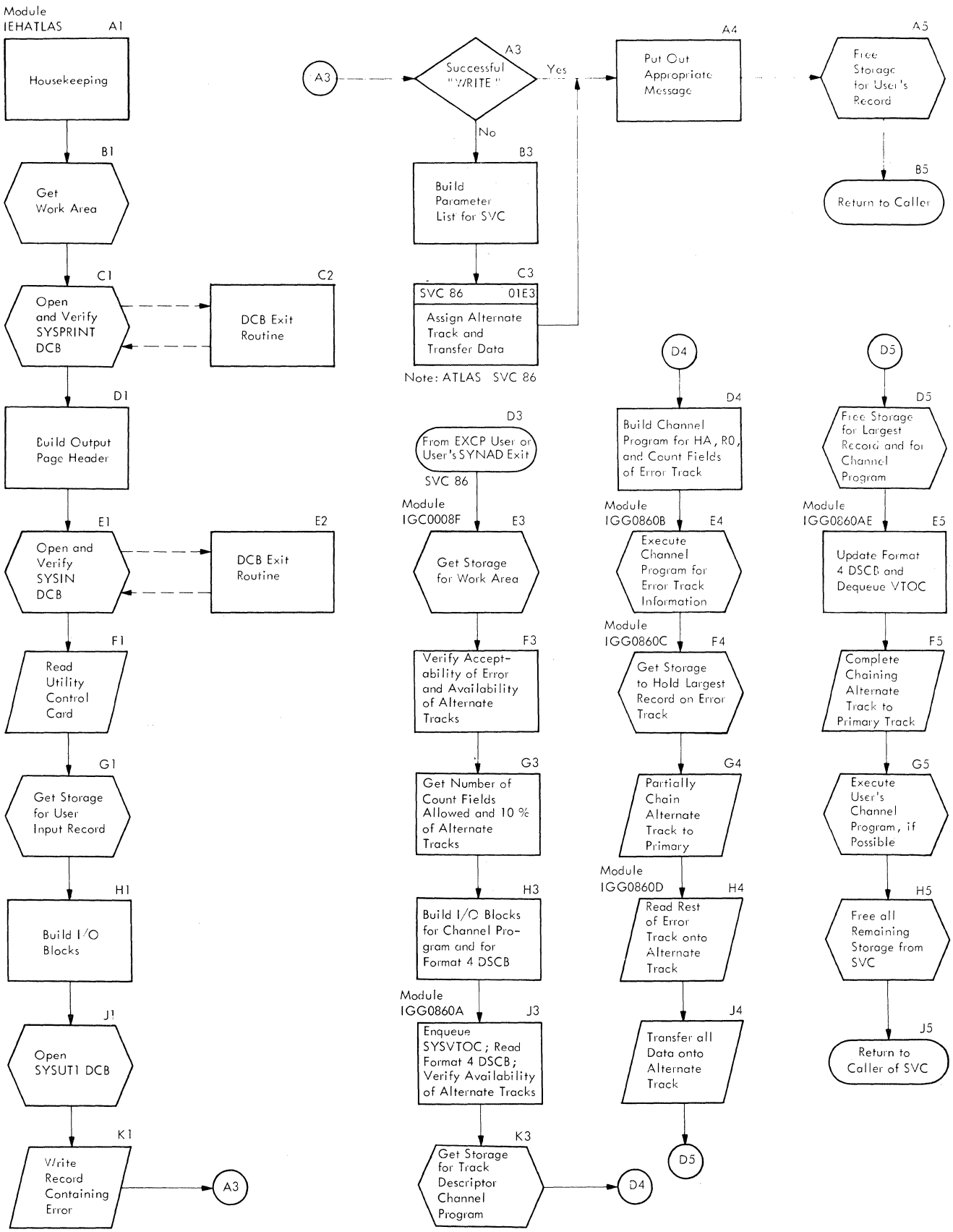
Legend

Channel Command Mneumonic	Meaning
RC	Read Count Field of Record.
RCKD	Read Count, Key, and Data Fields of Record.
RD	Read Data Field of Record.
RHA	Read Home Address of Track.
RKD	Read Key and Data Fields of Record.
RR0	Read Record 0 of Track.
SC	Space Count
SHA	Search on Home Address of Track.
SIDE	Search for an Equal Record Identification (CCHHR)
TIC	Transfer in Channel.
WCKD	Write Count, Key, and Data Fields of Record.
WD	Write Data Field of Record.
WHA	Write Home Address of Track.
WR0	Write Record 0 of Track.

Figure 152. Channel Programs Developed by IEHATLAS Utility Program and ATLAS Routine

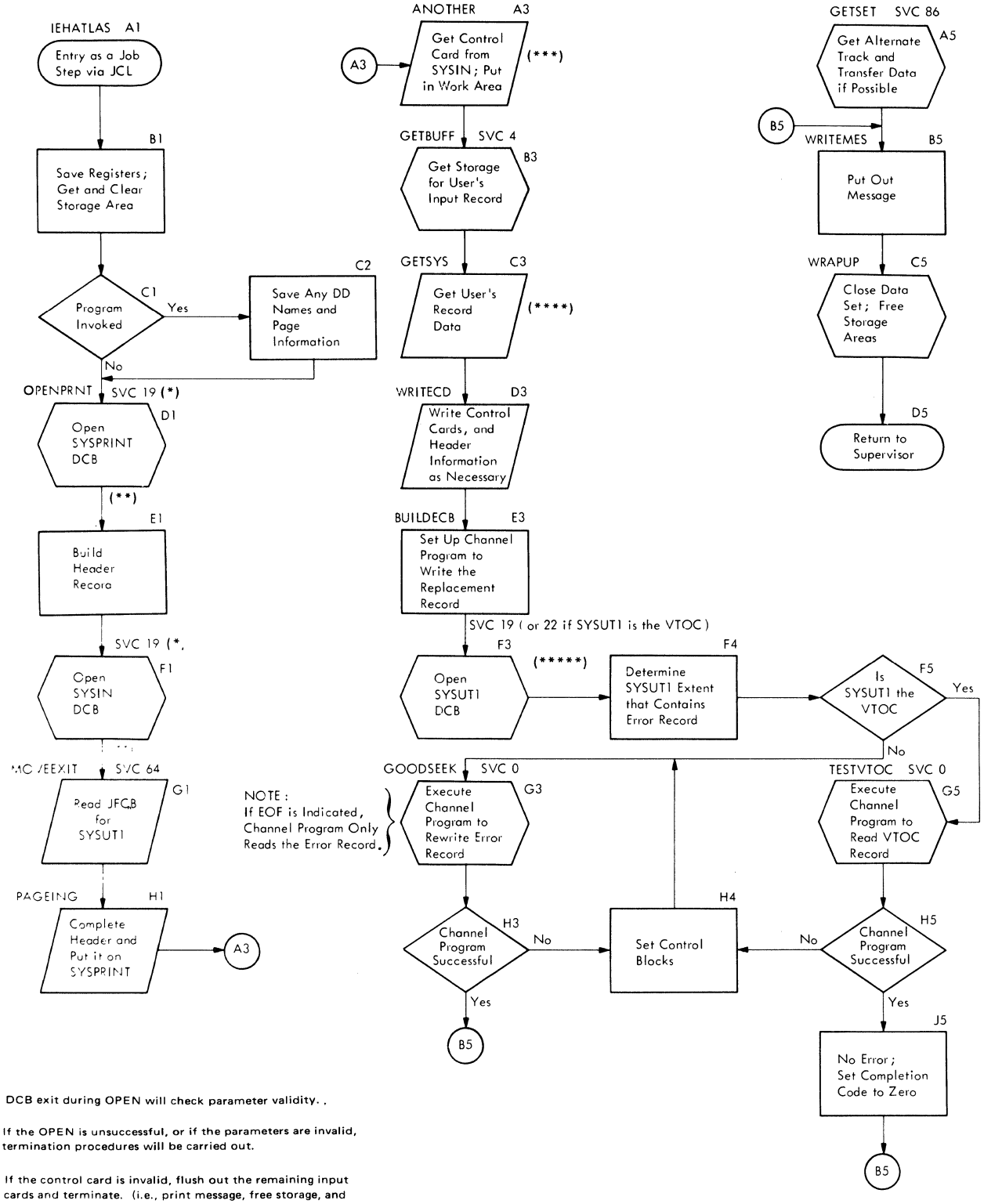
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 01. OVERALL LOGIC FLOW



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 02. UTILITY LOGIC FLOW



* DCB exit during OPEN will check parameter validity. .

** If the OPEN is unsuccessful, or if the parameters are invalid, termination procedures will be carried out.

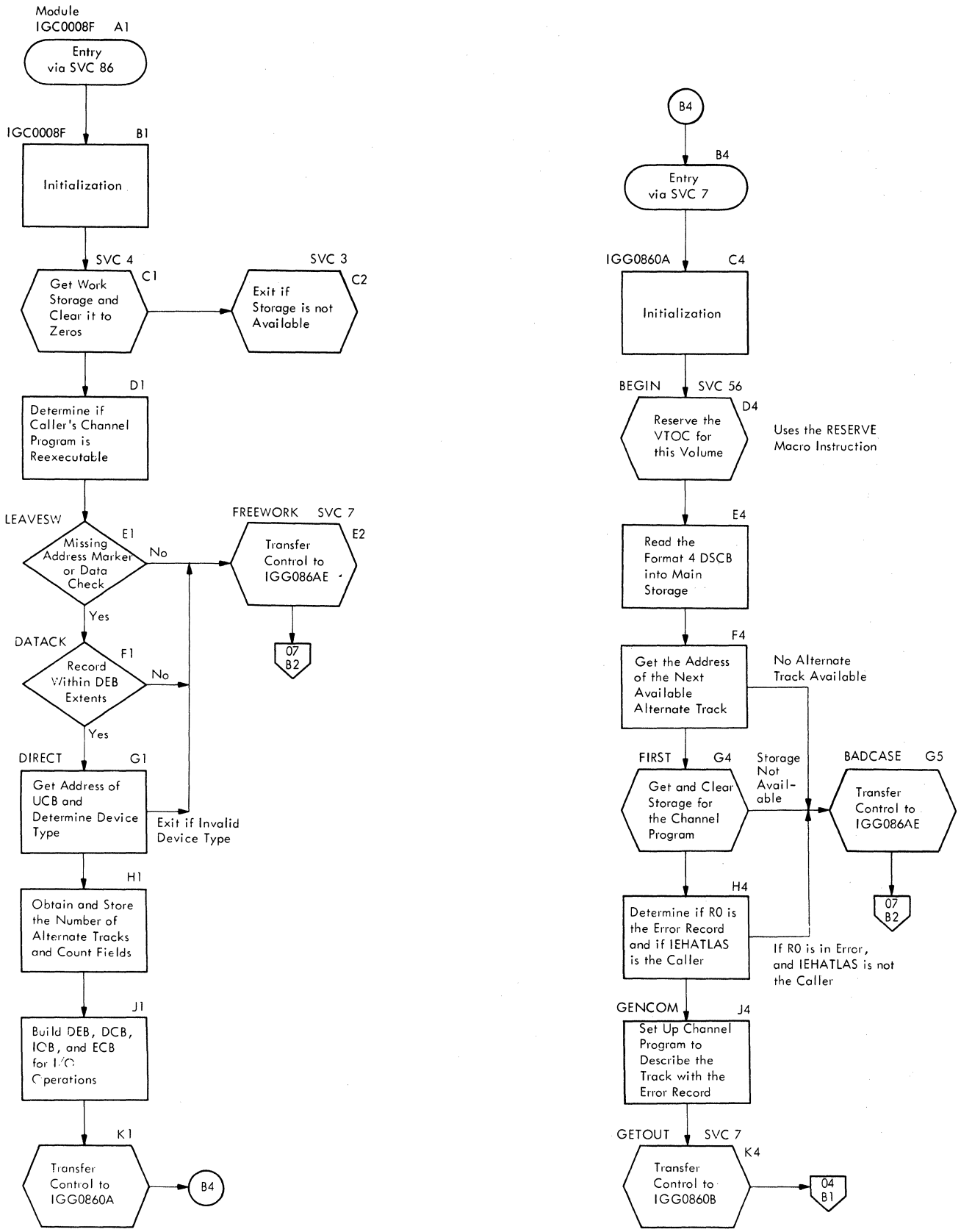
*** If the control card is invalid, flush out the remaining input cards and terminate. (i.e., print message, free storage, and close data sets.)

**** SYNAD exit will be taken if invalid data is encountered. Message printing and termination will occur.

***** If the device is Virtual, message printing and termination will occur.

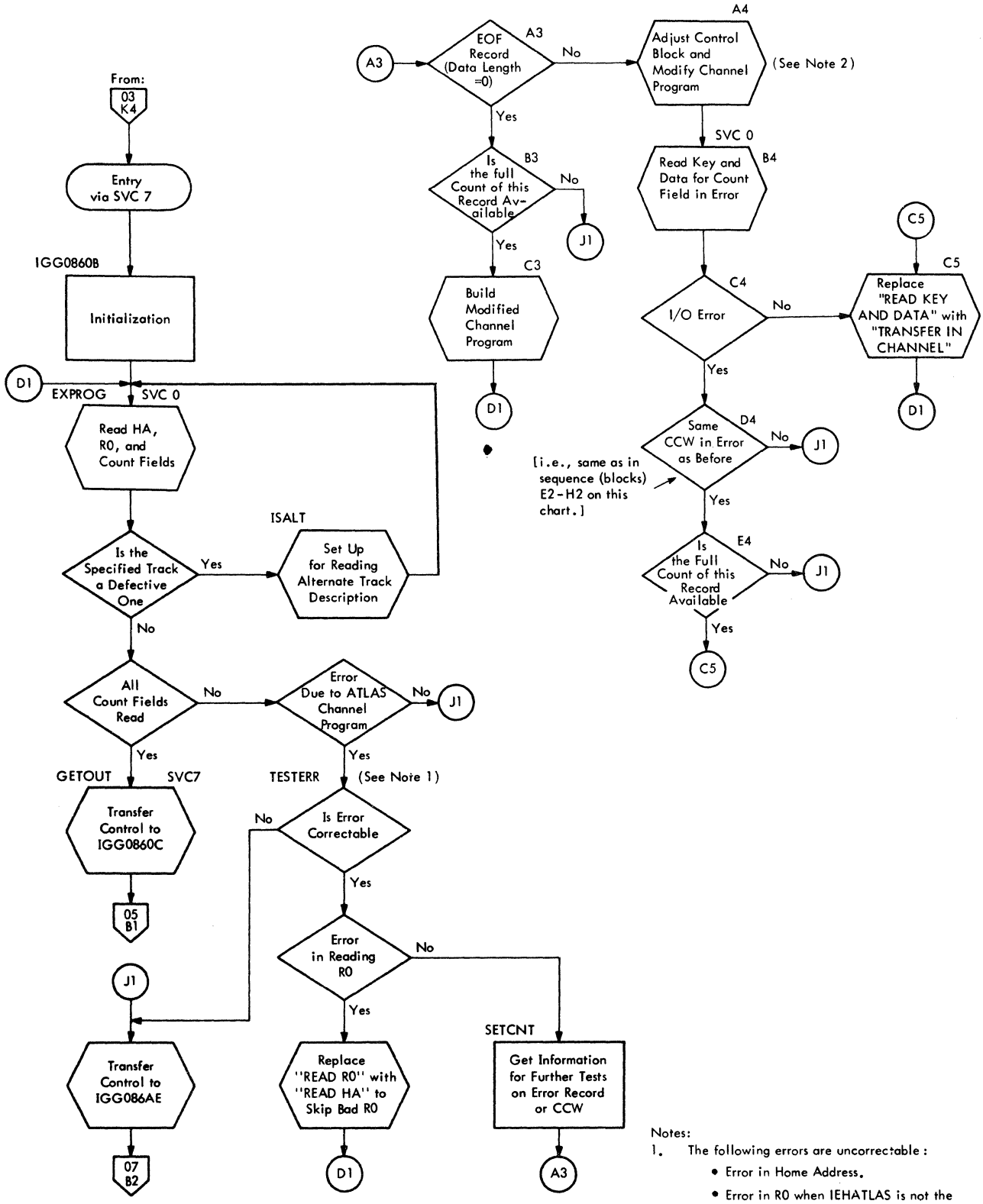
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 03. ATLAS MODULES IGC0008F AND IGG0860A



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 04. ATLAS MODULE IGG0860B

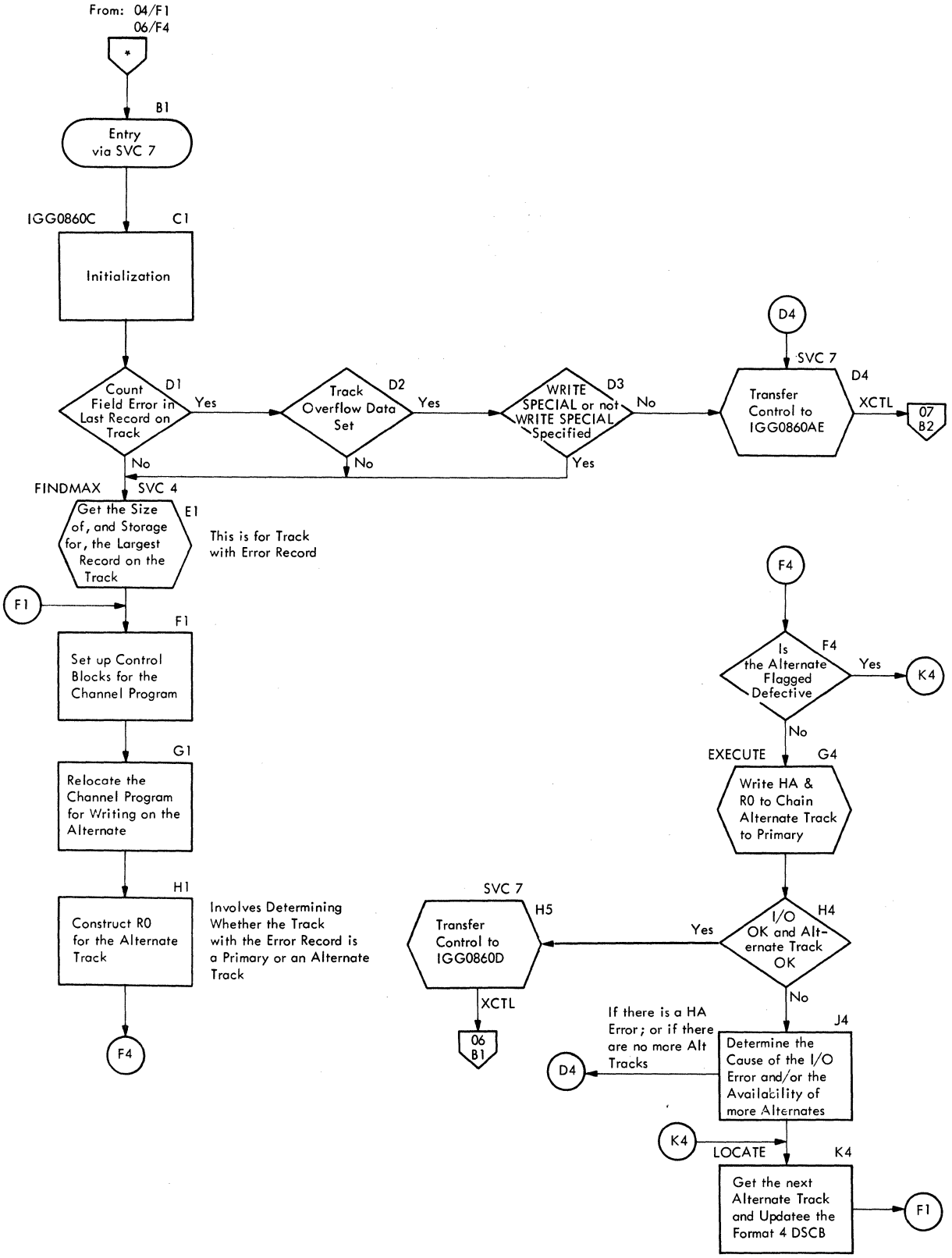


Notes:

1. The following errors are uncorrectable:
   - Error in Home Address.
   - Error in R0 when IEHATLAS is not the caller of ATLAS.
   - Errors in more than three "READ COUNT" commands.
2. Adjust channel program for previous "READ COUNT" error, if necessary.

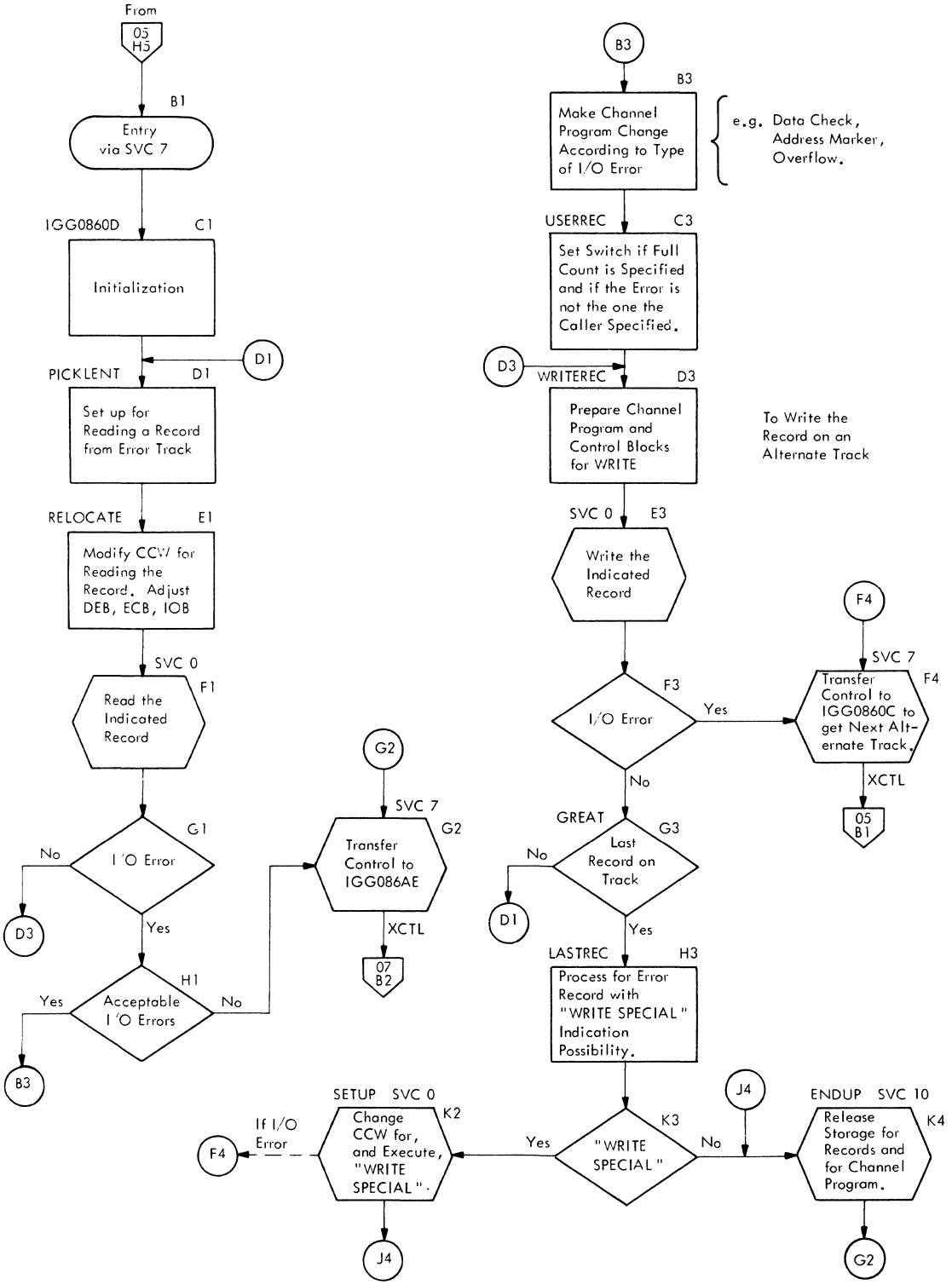
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 05. ATLAS MODULE IGG0860C



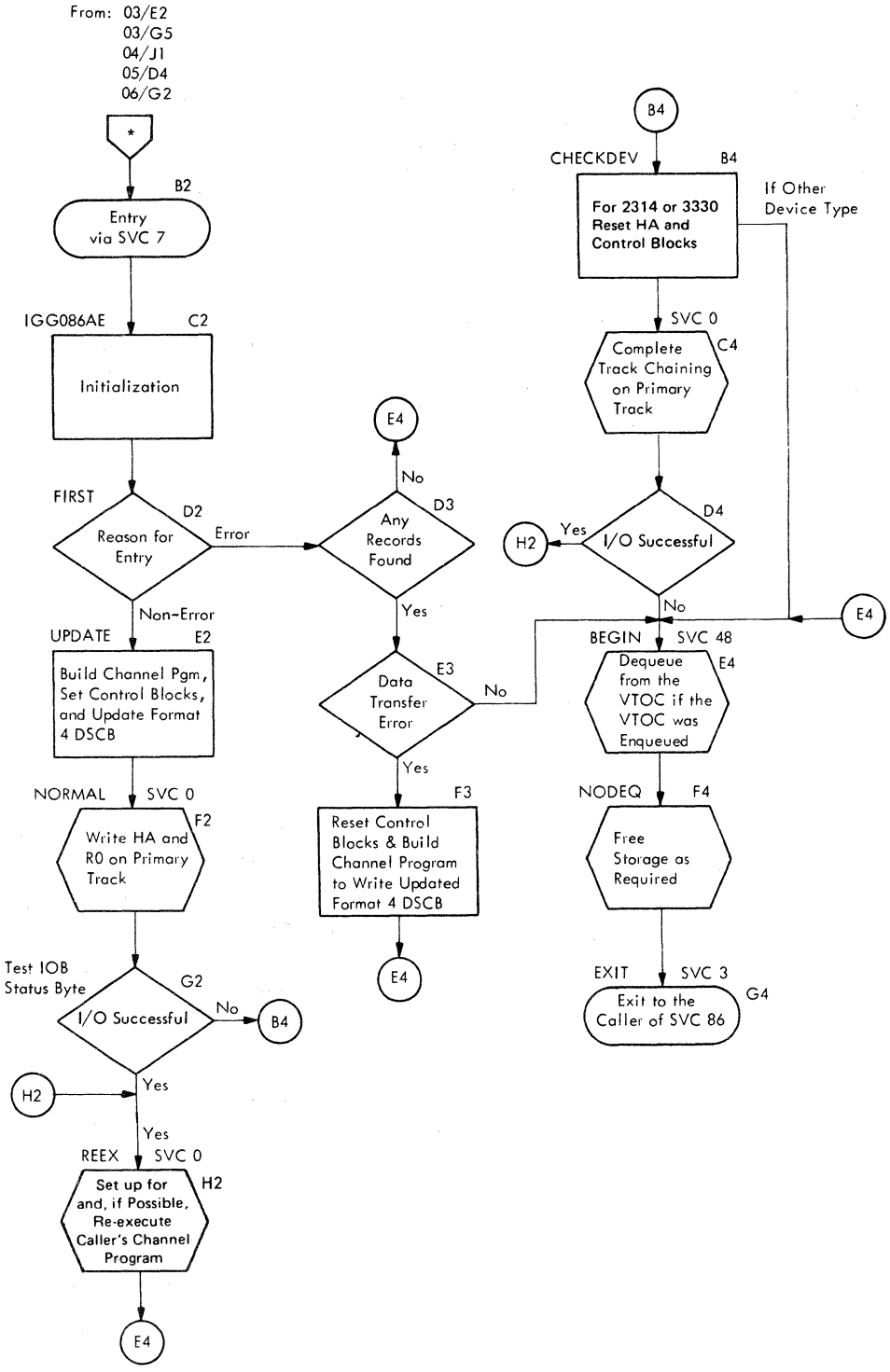
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 06. ATLAS MODULE IGG0860D



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 07. ATLAS MODULE IGG086AE



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEHINITT PROGRAM

The IEHINITT program writes volume label sets on tapes based on information current with ISO/ANSI X3.27 - 1978 specifications for AL tapes. The program reads control cards, builds a parameter list, calls an SVC routine to write a tape volume label set, and informs the user of the result of the labeling attempt.

PROGRAM FLOW

The general flow of the program and its relationship to the operating system are shown in Figure 153 on page 426. Charts 01 and 02 show the control routine, Charts 03 through 08 show the scan routine, and Chart 09 shows the print routine. Chart 10 shows the logic of SVC 39, the tape-labeling SVC routine.

PROGRAM STRUCTURE

The program consists of four modules:

- The control module (IEHINITT)
- The SVC 39 routine (IGC0003I)
- The control statement scan routine (IEHSCAN)
- The message writer (IEHPRNT)

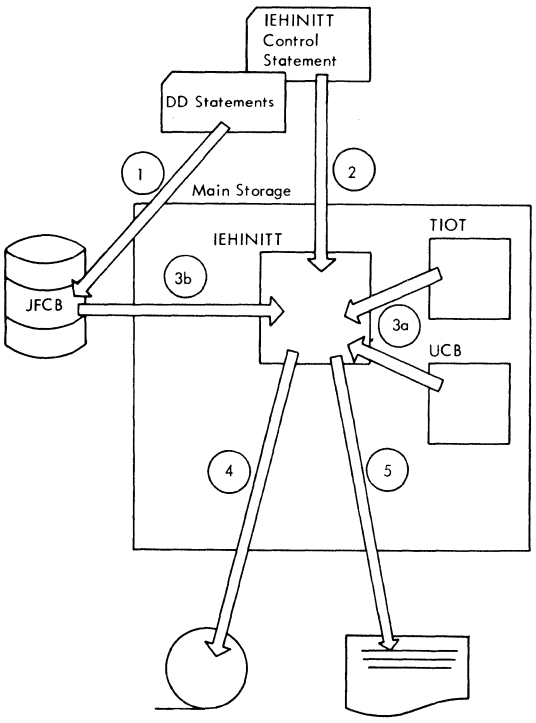
The Control Module (IEHINITT)

The control module (IEHINITT) builds two DCBs (SYSIN and SYSPRINT) for the tape-labeling operation, and moves the DCBs to the work area. By issuing a GETMAIN, the routine obtains 208 + X bytes, where X is the amount by which the volume label exceeds the standard length of 80 bytes. This area is used for building a DCB, DEB, ECB, IOB, and a channel program, and also holds messages and labels. It then links to the message writer (IEHPRNT) to write a header, and links to the control statement scan routine (IEHSCAN) to read a control statement into main storage. IEHPRNT then prints the control statement.

Control cycles between IEHINITT and IEHSCAN until the parameters are analyzed or an error is detected. If there are no errors, IEHINITT builds an image of the tape label and the dummy header in contiguous main storage, and then builds a parameter list for the tape-labeling SVC by referring to the JFCB, TIOT, and UCBs for DD statement information. The symbolic link needed by the program to gain access to this information is the ddname, supplied in both the DD statement and the utility control statement.

IEHINITT determines which type of labels are to be written and invokes the tape-labeling routine SVC 39. When control is returned, IEHINITT analyzes the return code and links to IEHPRNT to print the label or an error message. The process of building the parameter list, issuing the SVC, and interpreting the return code is repeated for each tape to be labeled. When the last tape has been labeled, IEHINITT returns control to the supervisor.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1



(1)

Before IEHINITT has gained control, information from the data definition (DD) statements has been entered in the task I/O table (TIOT) and job file control blocks (JFCB)

(2)

IEHINITT gains control and reads a control statement. The dd name from the control statement indicates which collection of tape drives to use from the TIOT

(3)

A drive is selected and its relative position in the TIOT is described in the parameter list for the tape labeling SVC. The parameter list is built by extracting:

a) the device type (dual density, 7-track, or 9-track) from the UCB
b) the density for dual density or 7-track from the JFCB

(4)

The SVC is issued and the tape label is written

(5)

The return from the SVC is analyzed and the result is logged. If the request being processed shows more tapes to be labeled, go to (3). If another control statement is to be read, go to (2).

Figure 153. Writing Tape Labels

The SVC 39 Routine (IGC0003I)

The SVC routine writes the specified volume label set, which contains:

- A standard volume label with user-specified serial number and owner identification. In addition, an access code may be present for AL tapes.
- An 80-byte dummy header label. For IBM standard labels, this record consists of a HDR1 followed by zeros. For ISO/ANSI labels, this record consists of a HDR1 followed by zeros in the remaining positions, with the exception of:
  - Position 54, which will contain an ISCII/ASCII space.
  - A '1' in the file section, file sequence, and generation number fields.
  - A leading blank (ISCII/ASCII space) in the creation and expiration date fields.
  - A system code of 'IBMZLA' followed by 13 blanks, which identifies the operating system that created the label.
- A tapemark.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Upon entry to the SVC routine, register 1 contains the address of a 5-word parameter list:

Word	Bytes	Contents
0	0-1	X'C000'
	2	X'04' to rewind tape X'06' to unload tape
	3	A binary number from 0 to n-1, where n is the number of UCB addresses in the DD entry portion of the TIOT indicating which device to use for volume mounting
1	0-3	A pointer to an 8-byte area containing the ddname corresponding to the ddname in the DD entry portion of the TIOT; the ddname is left-aligned and padded with blanks
2	0-3	A pointer to one volume label image to be written on the tape
3	0-1	The binary length of a volume label
	2	The number of labels in the volume label set.
	3	Command code for the control CCW to set mode
4	0-3	A pointer to a 10-byte area containing the EBCDIC volume ID and serial number

The SVC routine extracts the UCB address from the DD entry portion of the TIOT. This UCB is checked before the labeling of each tape to verify that the tape is not SYSIN or SYSOUT, that the tape is online and not scheduled to be offline (a VARY OFFLINE request is recognized immediately), that the tape is not reserved, and that the data management count is zero. If a tape is already mounted on the device and its volume serial number is in the UCB, it is unloaded. After the volume label has been written and verified, the dummy header label and tapemark are written. If the tape is not to be unloaded, its volume serial number is left in the UCB. If a nonstandard label was written, the pseudo volume serial number LGL000 is left in the UCB. If an I/O error is encountered in the labeling process, the operator is given one attempt to correct the situation. When a file-protected tape is encountered, the IEHINITT operator is allowed an unlimited number of retries to correct the situation.

When returning control to IEHINITT, the SVC routine issues a FREEMAIN to free the work area, and loads register 15 with one of the following return codes:

Code	Meaning
00	Labeling successful
04	Operator has canceled labeling
08	Unacceptable parameter list
0C	Permanent I/O error

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

The Control Statement Scan Routine (IEHSCAN)

This routine reads control statements using QSAM, checks syntax, and returns to IEHINITT an indication of the item scanned. IEHINITT supplies a work area (on a fullword boundary) containing the DCB for the control statement data set, which is opened by IEHINITT before calling IEHSCAN. IEHSCAN inserts the address of the end-of-file routine, KEOF, in the DCB, and the EOF routine for IEHINITT is restored when control is returned to IEHINITT.

After scanning a field from the control statement, IEHSCAN returns to IEHINITT the following information:

- Register 1 points to the starting address of the field.
- Register 2 contains the length of the field.
- Register 3 points to the starting address of the next field.
- A setting of a byte, SWITCHRD, in the work area, is as follows:

Bit 1	Meaning
0	Syntax error
1	Bypass switch
2	EOF
3	Initial entry
4	Command word
5	Keyword
6	Parameter
7	Comment

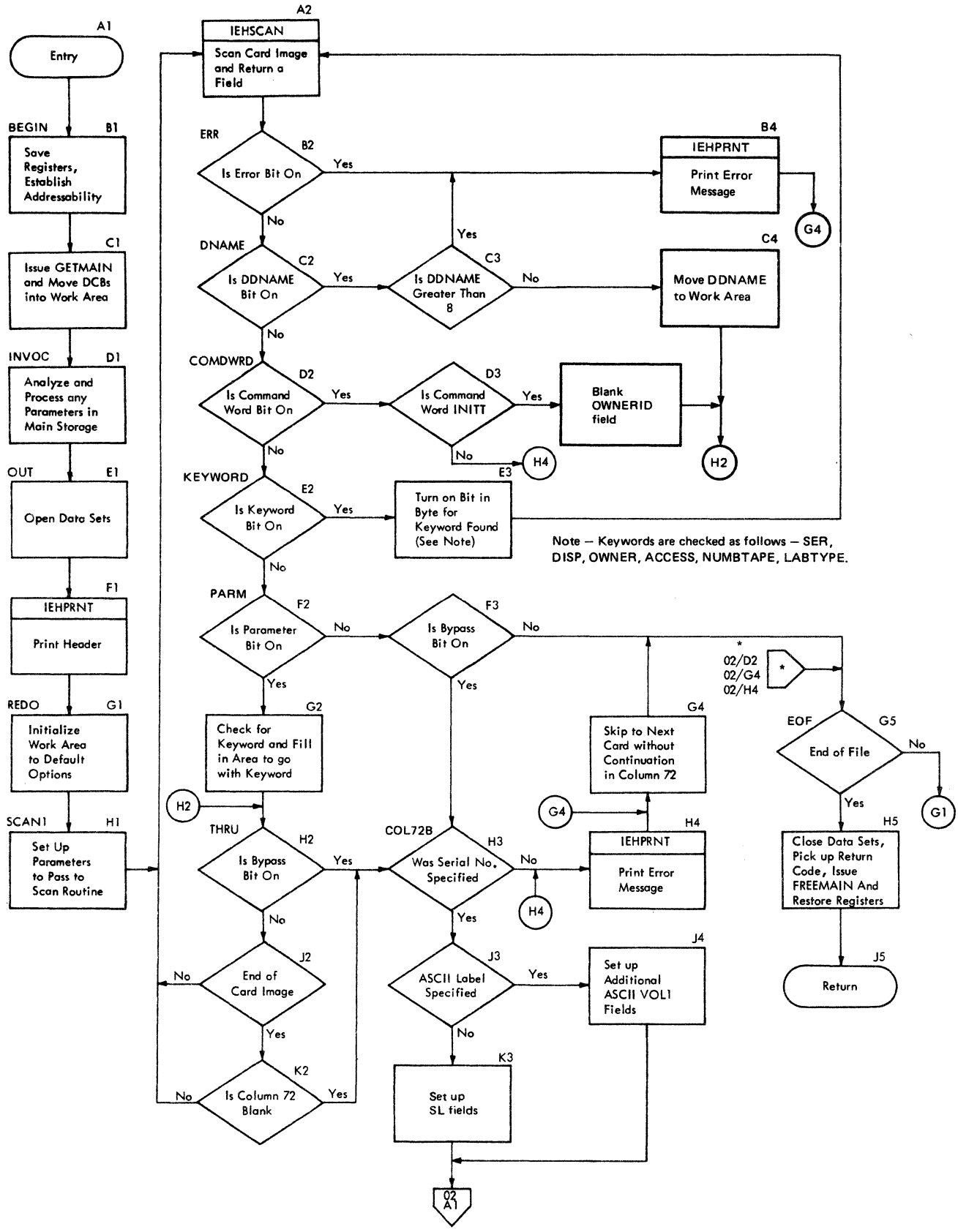
Unlike other control statement fields, the owner name field (when enclosed in apostrophes) is moved from the control statement image to the label image by the control statement scan routine. The owner name is considered to begin at the first byte following the first apostrophe, if present; two consecutive apostrophes are considered a single embedded apostrophe and counted as 1 byte of a maximum of 10 for the field if the tape is to be written in EBCDIC or BCD, or a maximum of 14 if the tape is to be written in ASCII. The scan is terminated when the count is exceeded or when a single (that is, not followed immediately by another) apostrophe is encountered.

The Message Writer (IEHPRNT)

Using QSAM, this routine writes page numbers, headings, and messages. Upon entry to the message writer, register 2 contains the address of the COMPTAB, which contains the message writer area. If a permanent I/O error is detected in writing the message, the 1-byte switch SWITCH2 is set to X'01' before control is returned, and a code of 4 is returned to IEHINITT in register 15.

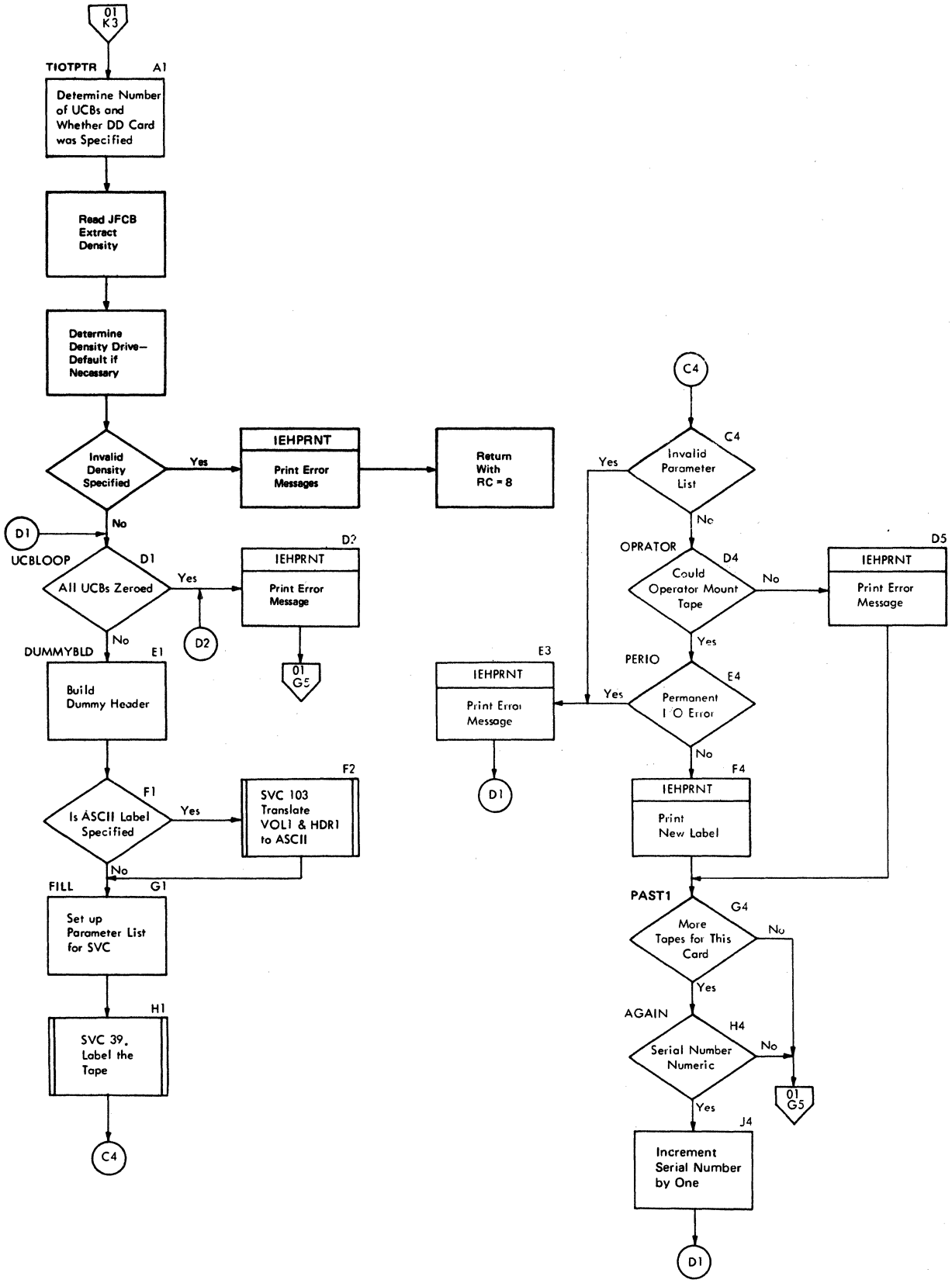
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 01. CONTROL ROUTINE (PART 1 OF 2)



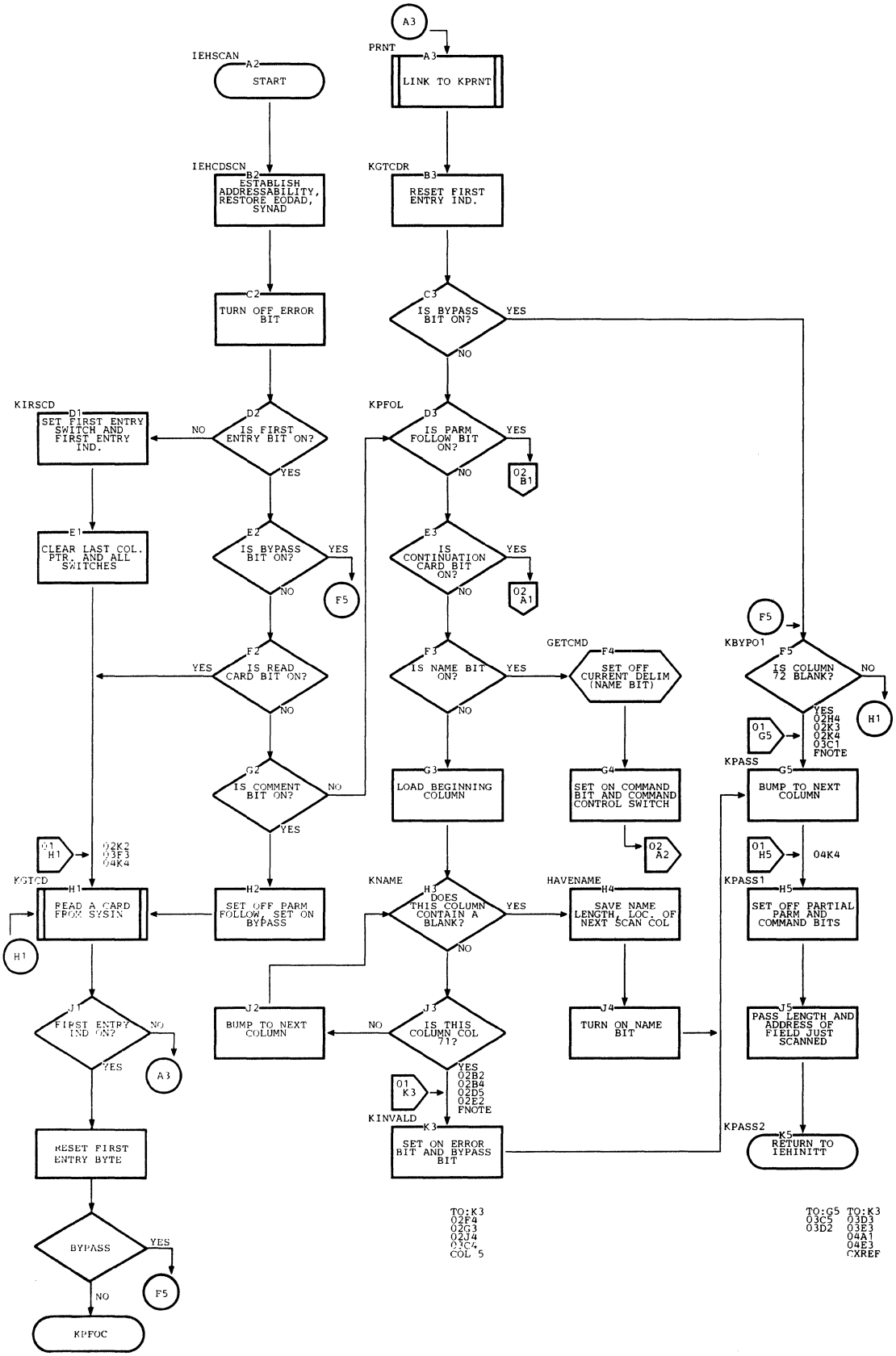
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 02. CONTROL ROUTINE (PART 2 OF 2)



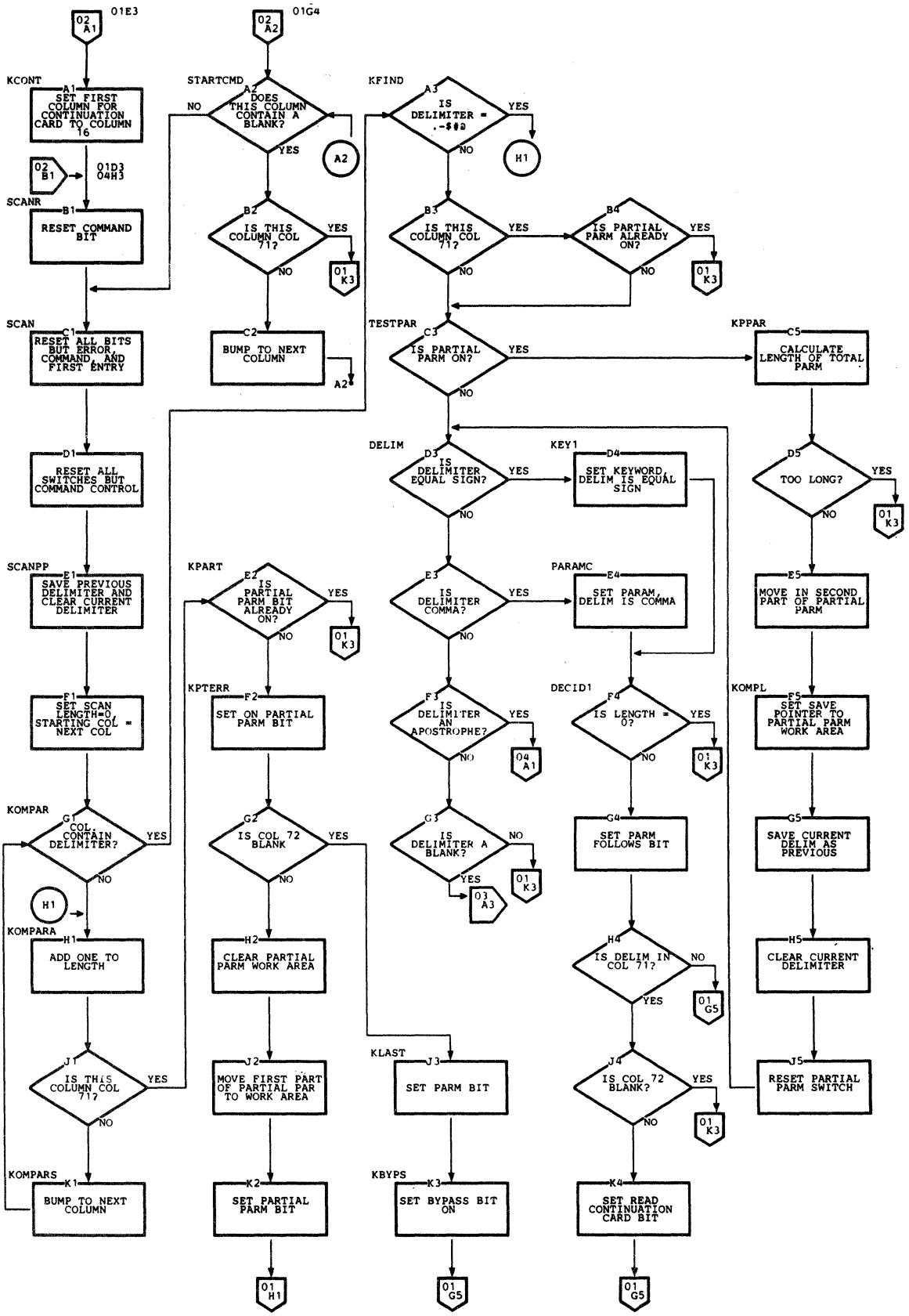
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 03. SCAN ROUTINE (PART 1 OF 6)



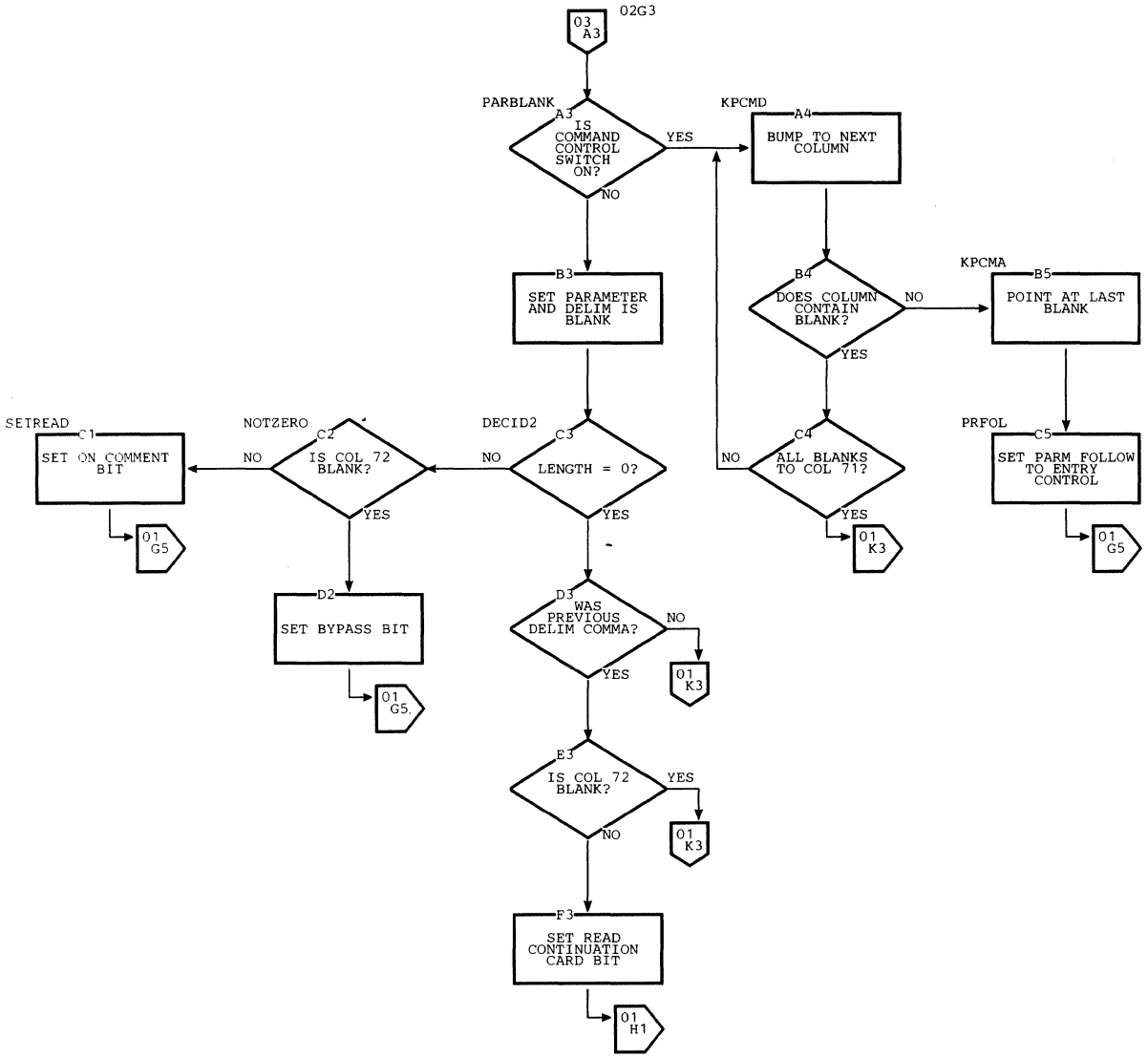
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 04. SCAN ROUTINE (PART 2 OF 6)



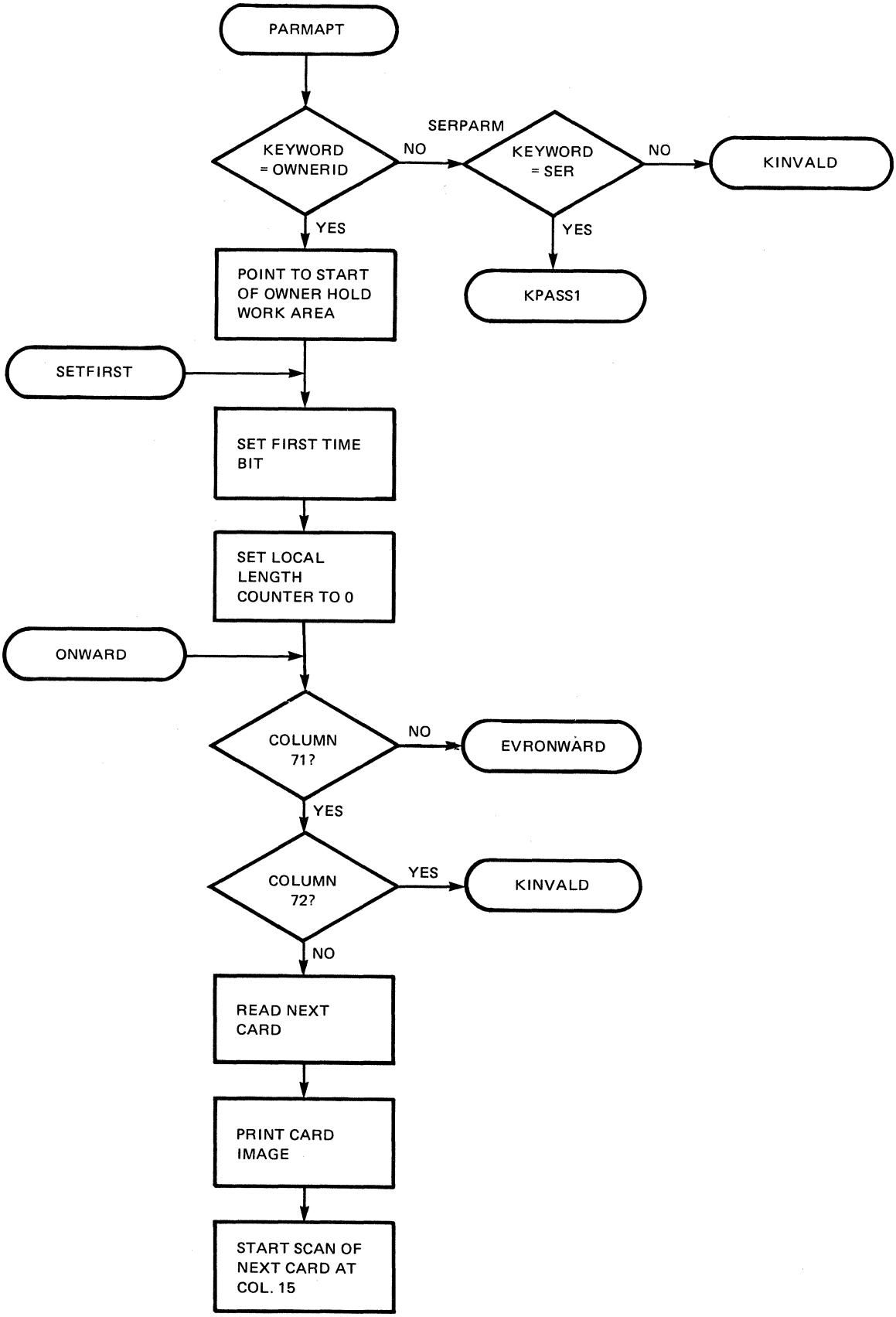
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 05. SCAN ROUTINE (PART 3 OF 6)



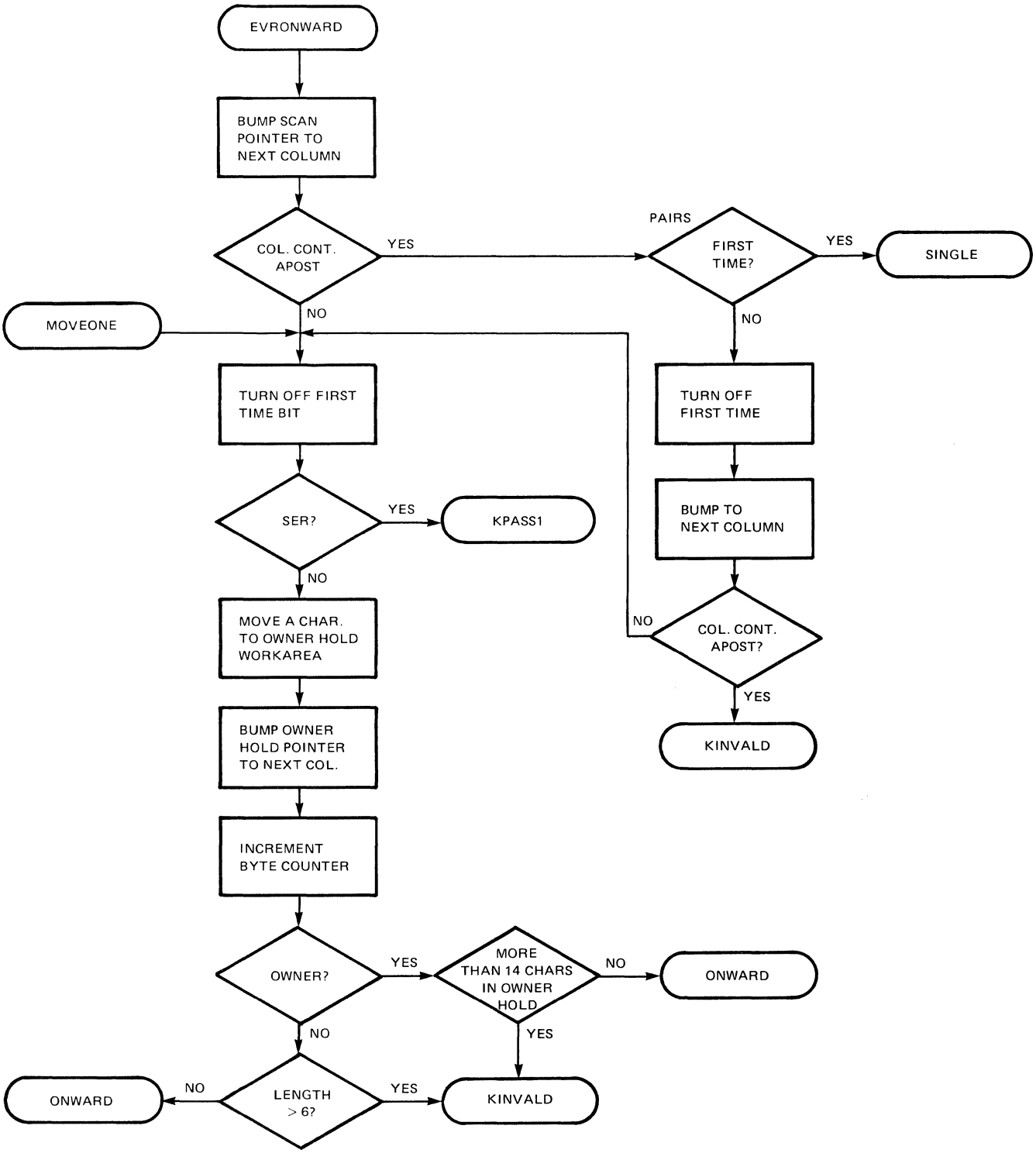
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 06. SCAN ROUTINE (PART 4 OF 6)



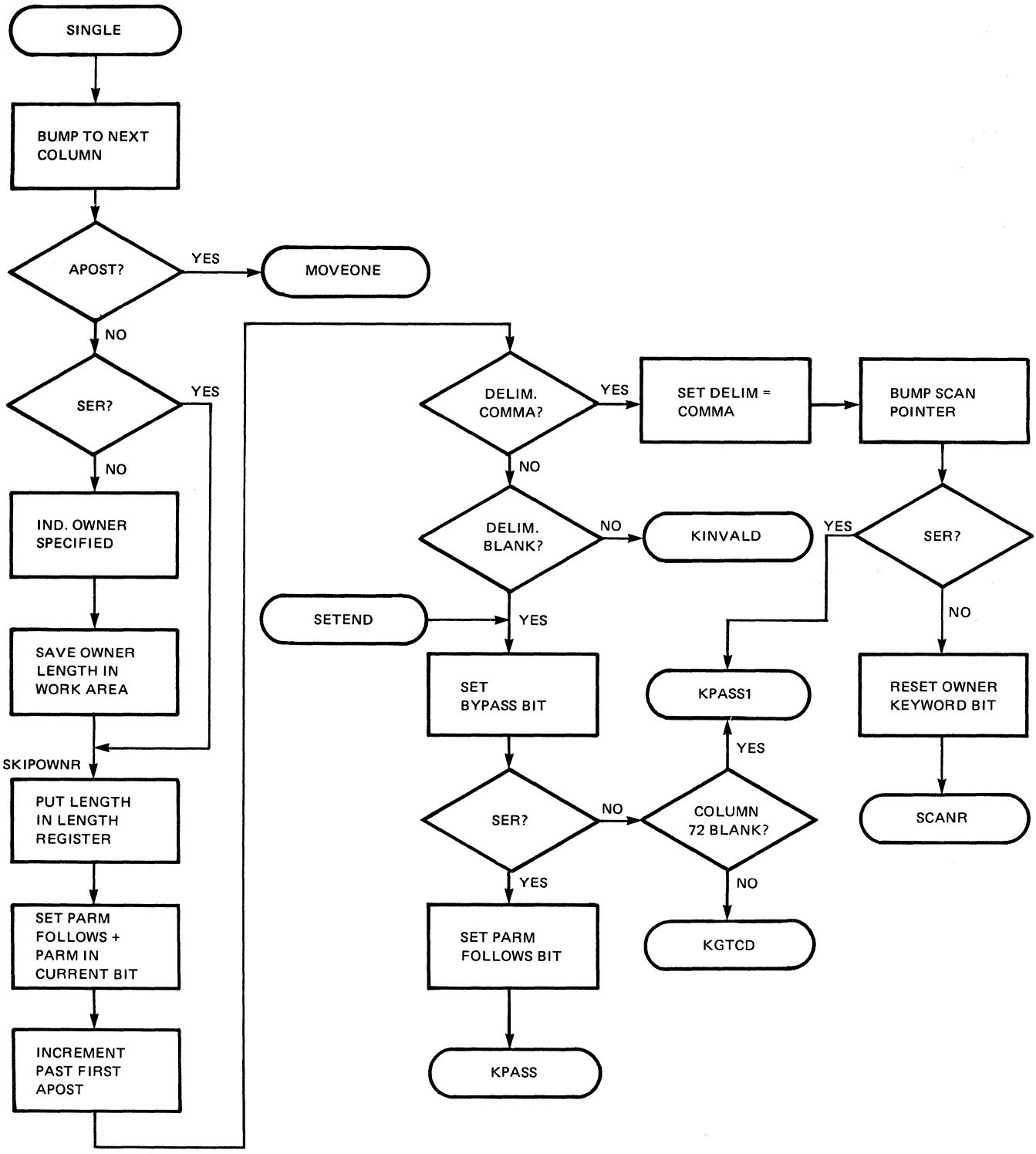
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 07. SCAN ROUTINE (PART 5 OF 6)



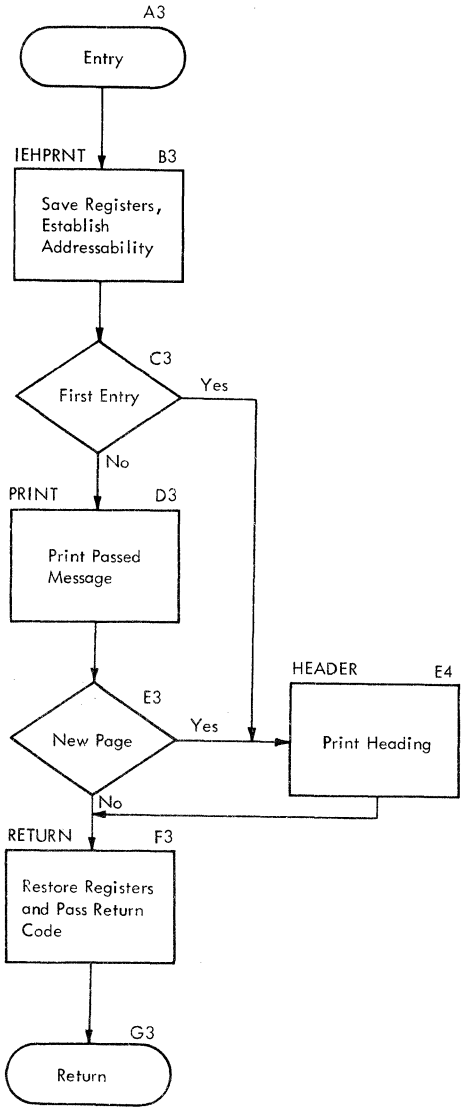
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 08. SCAN ROUTINE (PART 6 OF 6)



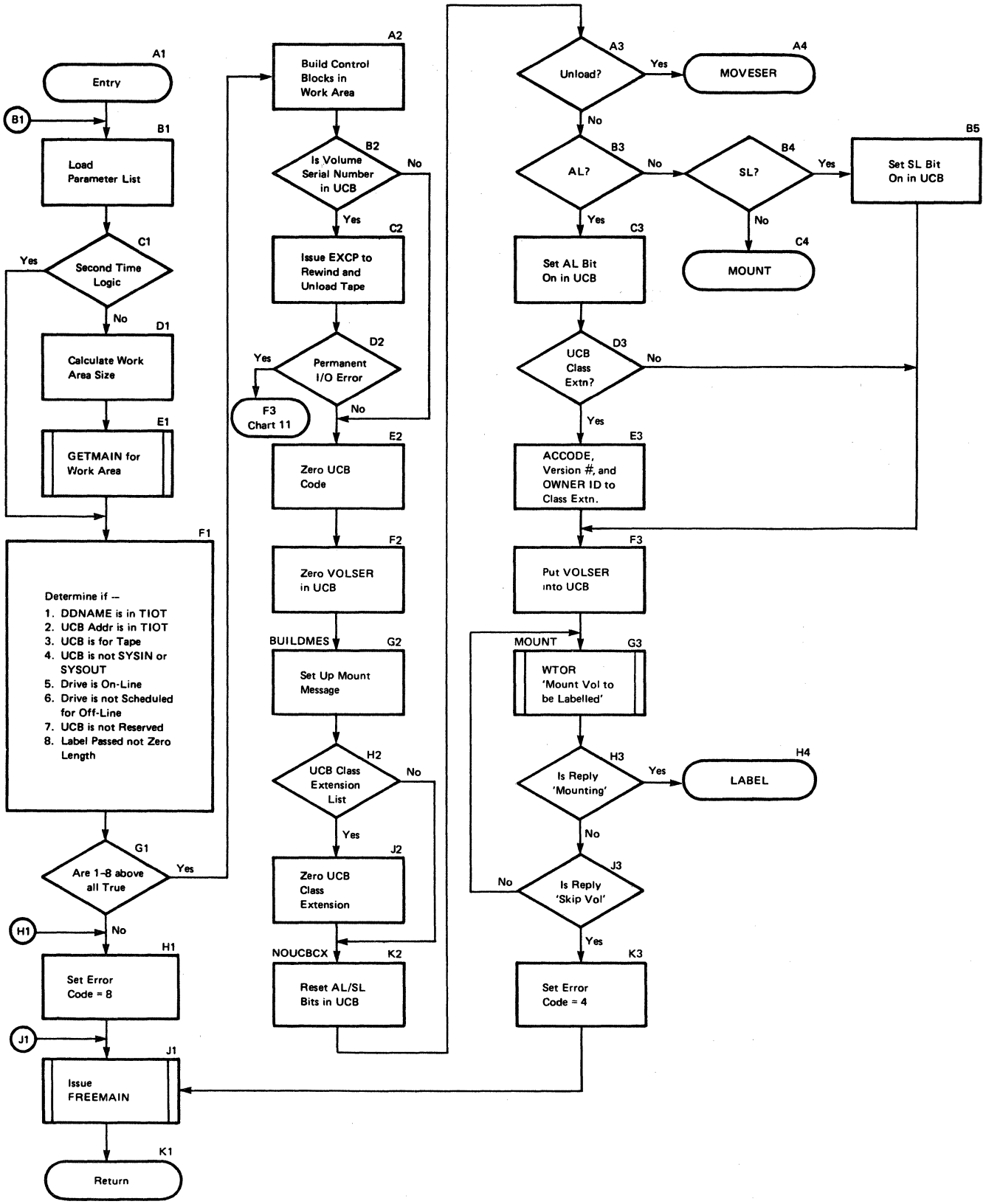
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 09. PRINT ROUTINE



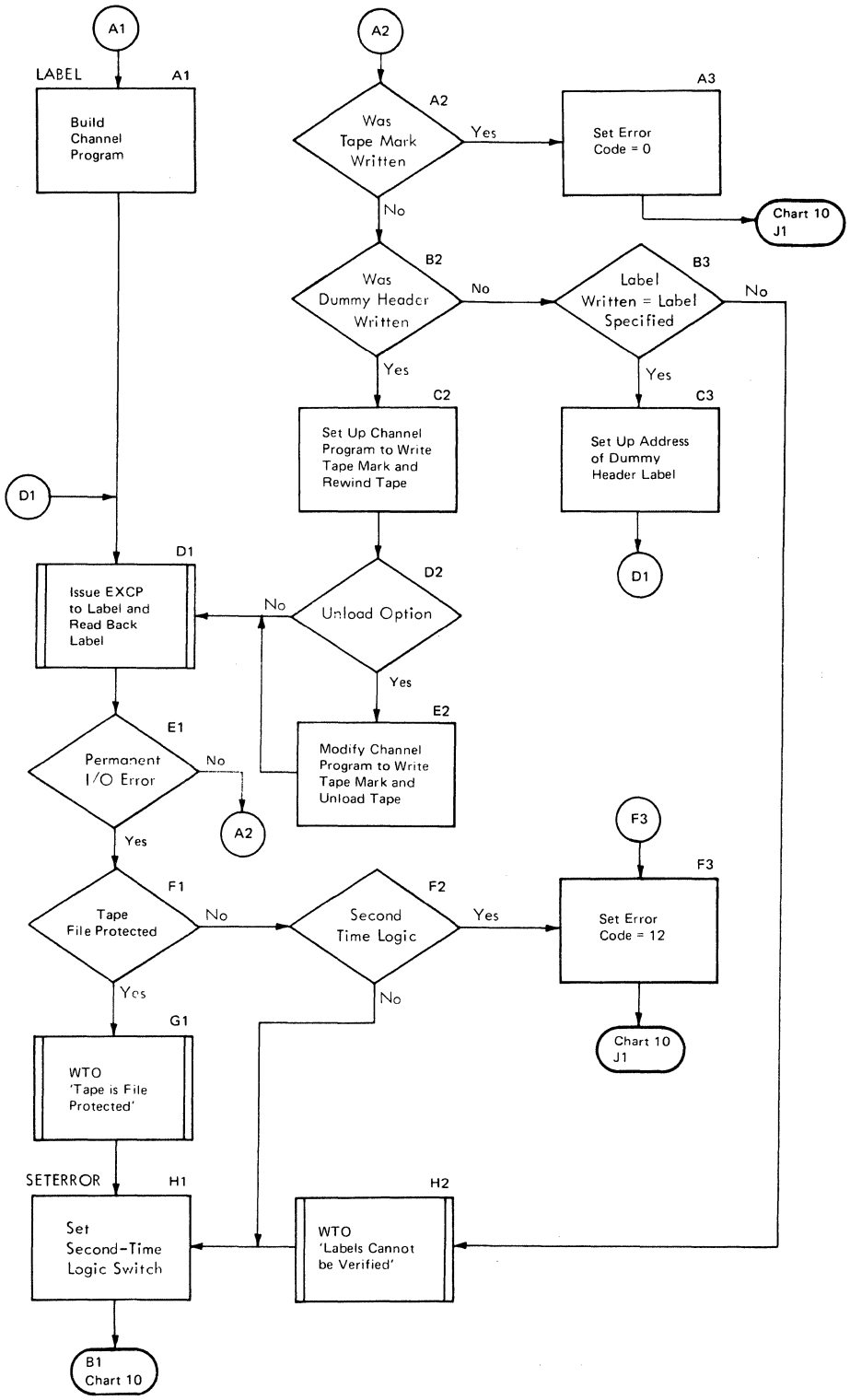
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 10. SVC 39 TAPE LABEL ROUTINE (PART 1 OF 2)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 11. SVC 39 TAPE LABEL ROUTINE (PART 2 OF 2)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEHLIST PROGRAM

The IEHLIST utility program performs four functions:

- It prints a volume table of contents (VTOC).
- It prints an index for the VTOC.
- It prints the directories of up to ten partitioned data sets (PDSs).
- It prints all or part of an OS CVOL.

The program is serially reusable, but not reenterable.

PROGRAM STRUCTURE

The structure of the program is shown in Figure 154 on page 441. The program consists of the following control sections (CSECTs):

- IEHLISTR performs basic program initialization. Unlike the other CSECTs, it is resident in main storage throughout the program's execution. IEHLISTR contains V-type address constants needed by the overlay supervisor.
- IEHLISTM contains only messages.
- IEHLISTP analyzes requests.
- IEHLISTC scans control statements.
- IEHLISTI interprets parameters supplied by a calling program.
- IEHLISTU produces a dump or abridged listing of the DSCBs in the VTOC.
- IEHLISTS scans and prints PDS directory data.
- IEHLISTD produces a formatted listing of the DSCBs in the VTOC.
- IEHLISTX contains the free-space summary statement printer, the VTOC index VIR dump printer, and the VTOC index VIER formatter and printer.
- FMTMAPS contains the VTOC index VPSM, VMDS, and VIXM formatting and printing routines.
- IEHLISTQ scans and prints cataloged data.
- IEHVOLMT mounts necessary volumes. It is described in "Device Allocation and Volume Mounting (IEHMVSSF, IEHMVXSF, and IEHVOLMT)" on page 3.

"Chart 01. Listing System Control Data (Part 1 of 2)" on page 446 shows the logical flow of control through the program. Figure 155 on page 442 shows the structural flow of the program, including the successive phases of the contents of storage during the program's execution. The logic of each control section (CSECT) of the program is described in the following paragraphs:

IEHLISTR
contains miscellaneous routines needed in storage throughout the program's execution (PERRPR, WORKERR, PTERM, LINEPR, DOPOINT, and PBEGIN), together with several communication areas (CARDIN, PRINTOUT, WORKIN, and RONTAB):

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

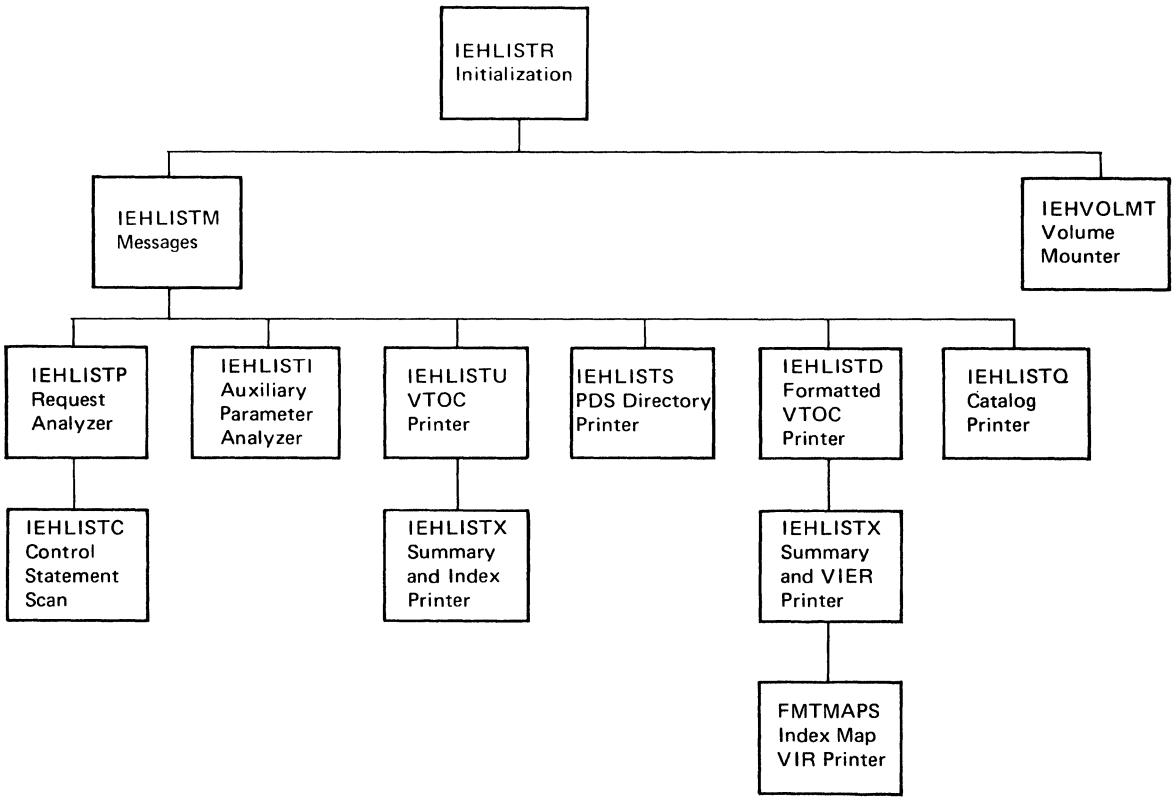


Figure 154. The Structure of the Program

PERRPR
causes any invalid control statements to be printed, and gives control to PTERM or PBEGIN, depending on whether there are any more control statements.

WORKERR
treats all SYNAD exits.

PTERM
receives control when all job requests have been serviced or aborted and ends the job.

LINEPR
prints all program output.

DOPOINT
issues all POINT supervisor calls used by the program.

PBEGIN
directs control to the segments by means of a branch table of V-type address constants.

The following areas are located in the root:

CARDIN
is the DCB for reading control statements.

PRINTOUT
is the DCB for printing the VTOC or PDS.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

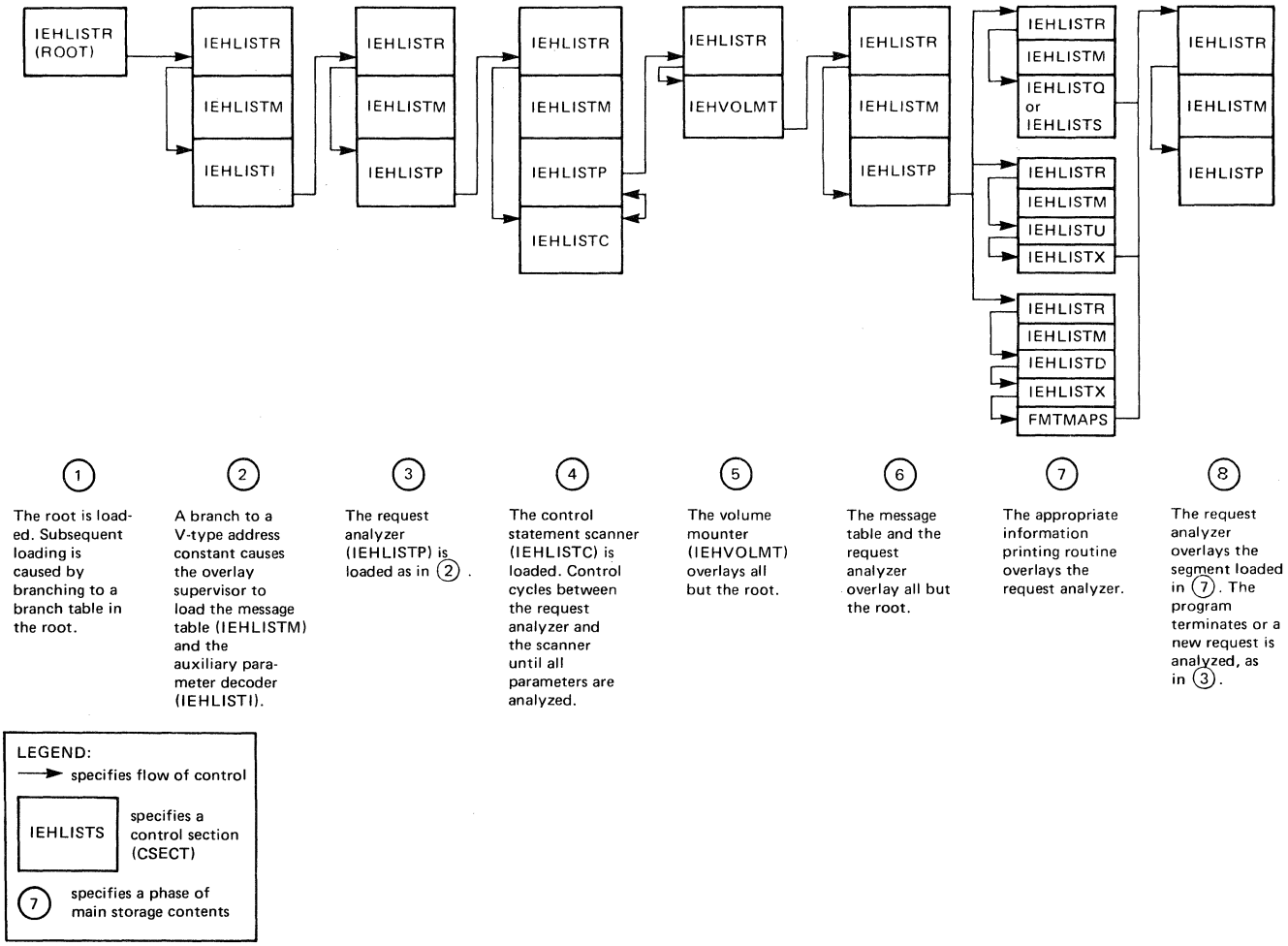


Figure 155. The Structural Flow of the IEHLIST Program

WORKIN
is the DCB for reading the catalog, VTOC, or PDS directory.

RONTAB
is the parameter list for the volume mounting routine.

IEHLISTP
analyzes requests and directs control to the appropriate routine to print the requested information. It contains the following subroutines:

PBEGIN
directs control to IEHLISTI to interpret calling program parameters, and to open SYSIN and SYSPRINT.

PON
directs control to IEHLISTC to obtain control card information.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

PKEY
is given control when a keyword is returned by IEHLISTC.

PPARAM
is given control when a parameter is returned by IEHLISTC. When the parameter supplied to the (optional) VOL keyword is returned, the parameter is used as a search argument to link to IEFAB4UV (at entry point IEFEB4UV). This routine returns device-type information obtained from the device class queue. The value retrieved is used by PWORKIN.

PWORKIN
constructs the calling sequence for IEHVOLMT.

PHEAD
prints a header and gives control to the appropriate routine to print VTOC or PDS directory information.

PTERM
receives control following the printing of VTOC or PDS directory information. If there is another request, PTERM directs control to PON; otherwise, PTERM closes SYSIN and SYSPRINT, and returns control to the supervisor.

IEHLISTC
scans utility control statements. It is described in "Control Card Scanner (RDCDRT)" on page 6.

IEHLISTI
interprets auxiliary parameters supplied by a calling program (these parameters are described in "Auxiliary Parameters" on page 2), and also opens SYSIN and SYSPRINT or their substitutes, as specified in the calling sequence to the program.

IEHLISTQ
prints the catalog. It gains control indirectly from the request analyzer, IEHLISTP, by means of a branch to a branch table in the root. IEHLISTQ contains the following routines:

QCHECK
scans the catalog for general information and prints it. Actual printing is done via a branch-and-link to LINEPR in the root.

QHEAD
prints a catalog header after the general information is printed. Actual printing is done via a branch-and-link to LINEPR.

QALL
scans high-level node points in the catalog and passes them to QLOCATE. QALL is used only for an entire catalog printout.

QLOCATE
scans from a node point to successive index levels until a data set pointer is found. A fully qualified data set name is placed at location INDXNAME for routine LPRDATA.

LPRDATA
prints information pertinent to a data set.

QCATREAD
performs all the reading of a catalog for the catalog function of the program.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEHLISTU
reads a VTOC. Control is gained indirectly from IEHLISTP by means of a branch to a branch table in the root. IEHLISTU contains the following routines:

RPARTIAL
treats requests for partial VTOC printouts. Successive DSCBs are printed by linking to RPRDSCB.

RENTIRE
treats requests for entire VTOC printouts and differs from RPARTIAL in that the VTOC must be opened.

RPRDSCB
handles the formatting of a DSCB in dump or abridged format.

RREAD
reads the VTOC.

IEHLISTS
prints PDS directories. Control is gained indirectly from IEHLISTP by means of a branch to a branch table in the root. IEHLISTS contains the following routines:

SREAD
reads the directory records for a given partitioned data set.

TESTOPT
tests for the DUMP or FORMAT option.

SSCAN
scans and prints the partitioned data set directory entries when the DUMP option is requested.

SECNDENT
scans, edits, and prints the partitioned data set directory entries when the FORMAT option is requested.

SEODAD
prints the number of empty directory records after all the directory entries have been printed.

IEHLISTD
formats the DSCBs in the VTOC (see Appendix C). This CSECT gains control indirectly from CSECT IEHLISTP by branching to a branch table in the root segment. CSECT IEHLISTD contains the following routines:

DPARTIAL
handles requests for a partial VTOC printout. The routine reads the identifier (Format-1) DSCB by using the OBTAIN macro instruction and links to the DFORMAT1 routine for formatting of the DSCB. The DPARTIAL routine then links to the DFORMAT2 and DFORMAT3 routines for formatting, respectively, the index (Format-2) and/or extension (Format-3) DSCBs.

DENTIRE
handles requests for the printout of an entire VTOC. This routine is similar to the DPARTIAL routine. However, for an entire VTOC print-out, the DCB for the VTOC must be opened and all forms of DSCBs are formatted.

DFORMAT1
handles the formatting of the identifier (Format-1) DSCB.

DFORMAT2
handles the formatting of the index (Format-2) DSCB.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

DFORMAT3
handles the formatting of the extension (Format-3) DSCB.

DFORMAT4
handles the formatting of the VTOC (Format-4) DSCB.

DFMT56
handles the formatting of the free space (Format-5) and shared extent (Format-6) DSCBs.

IEHLISTX
receives control following VTOC end of data from IEHLISTU or IEHLISTD. Control is gained indirectly by means of a branch table in the root. IEHLISTX contains the following routines:

DEODADCT
accumulates summary statement data for an OS VTOC.

IXEOD
accumulates summary statement data for an indexed VTOC.

DFINAL
prints the free-space summary statements.

ENIXRD
reads index VIRs from an enabled index to produce a dump or formatted listing of the index.

DISIXRD
reads index VIRs from a disabled index to produce a dump or formatted listing of the index.

DMPVIR
edits and prints index VIRs for a dump listing of the index.

FMTVIER
formats and prints VIERs for a formatted listing of the index.

FMTMAPS
is entered directly from IEHLISTX to format map VIRs for a formatted listing of the VTOC index. FMTMAPS contains the following routines:

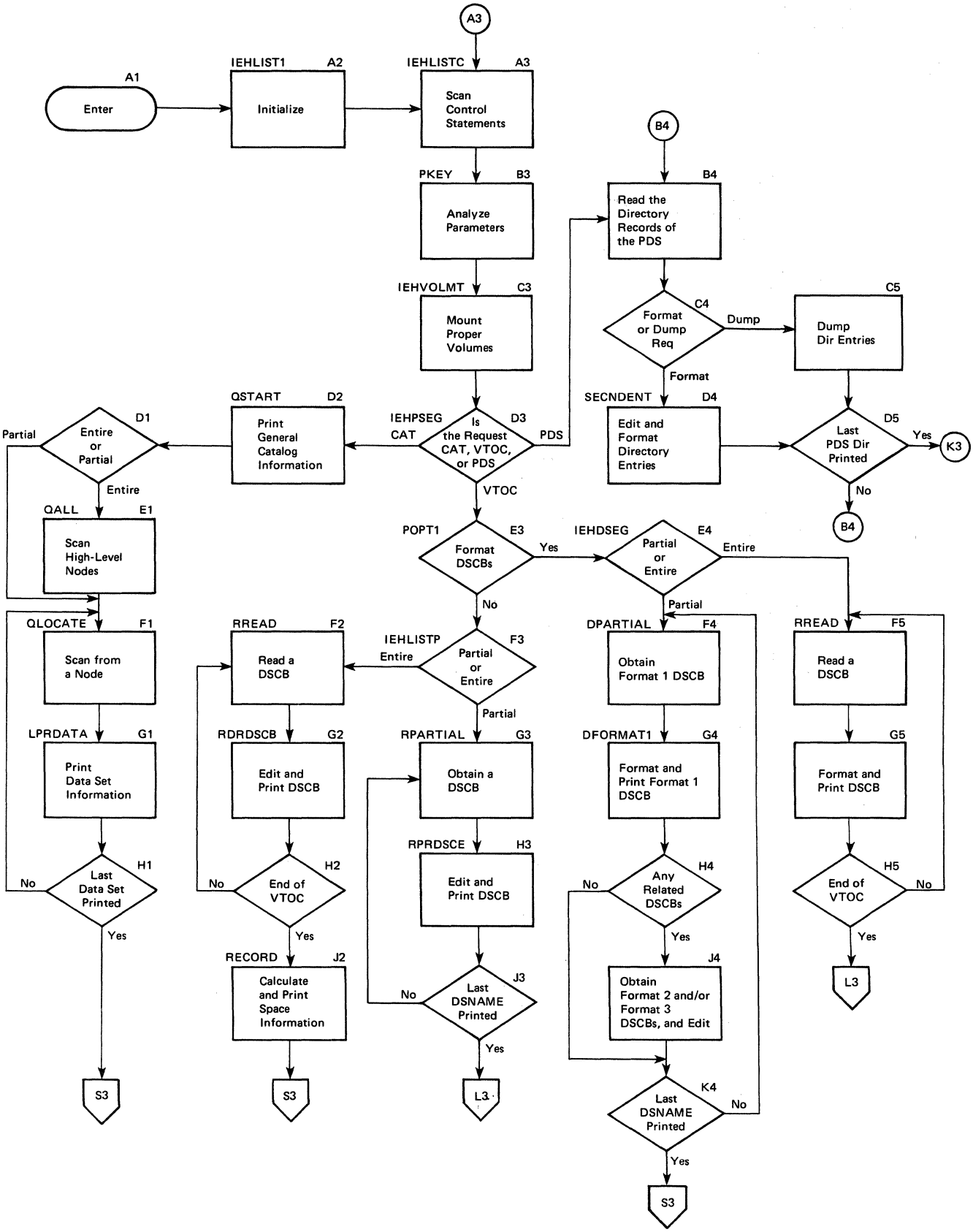
FMTVIXM
formats and prints VIXMs.

FMTVMDS
formats and prints VMDSs.

FMTVPSM
formats and prints VPSMs.

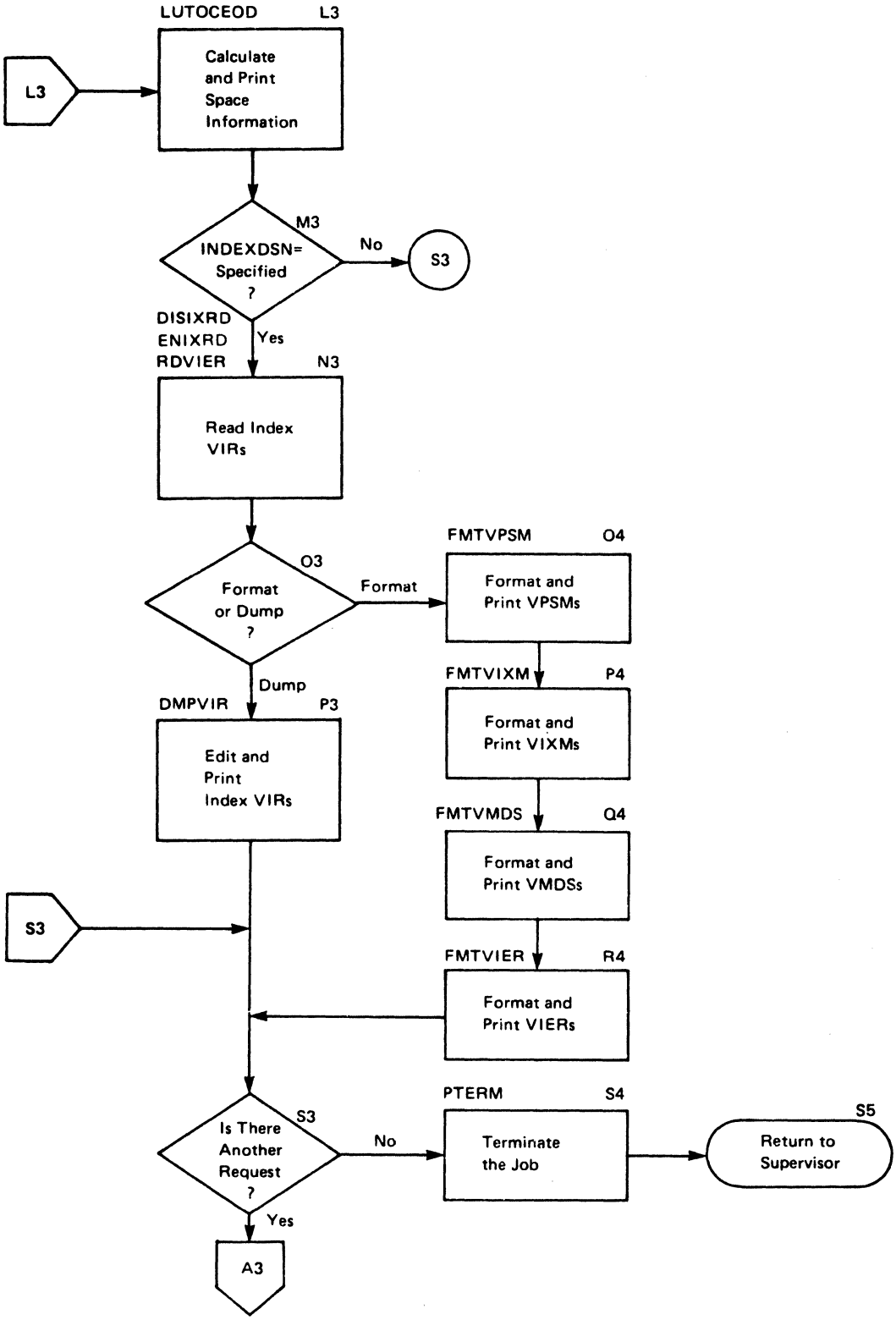
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 01. LISTING SYSTEM CONTROL DATA (PART 1 OF 2)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 02. LISTING SYSTEM CONTROL DATA (PART 2 OF 2)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

The IEHMOVE utility program reproduces one or more data sets. The move operation relocates a collection of data and scratches the source data; the copy operation produces a replica of the source data, and leaves the source data intact. (Throughout this discussion, the word "copy" refers to both the MOVE operation and the COPY operation.)

The program is serially reusable. It copies the following collections of data:

- A data set
- A volume
- A group of data sets related by a catalog
- All or portions of an OS CVOL (SYSCTLG data set)

Note: IEHMOVE does not copy ISAM or VSAM data sets.

Depending on the compatibility of the source and receiving volumes, the movability of the source data set, and the allocation of space on the receiving volume, an attempt to copy a data set may result in an "unloaded" version of the data set. This version of the data set is in a format recognizable to the IEHMOVE program, but is not directly usable by other programs. An attempt to copy an unloaded data set onto a volume that would have originally supported a successful operation results in the "loading" of the unloaded data set, that is, the reconstruction of the original data set.

If a user has requested the processing of input/output header/trailer labels, this program will handle the direct moving or copying of such labels as they exist on the data sets to be moved or copied.

OVERALL FLOW

Figure 156 on page 449 shows the design of the IEHMOVE program; each of the smaller blocks represents a load module, while each of the larger blocks represents a grouping of load modules by function:

- Program Setup
- Request Setup
- Message Writing
- DSGROUP Setup
- Data Set and Volume Setup
- Partitioned Data Set (PDS) Subroutines
- Copying, Unloading, and Loading
- Data Set Wrapup

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

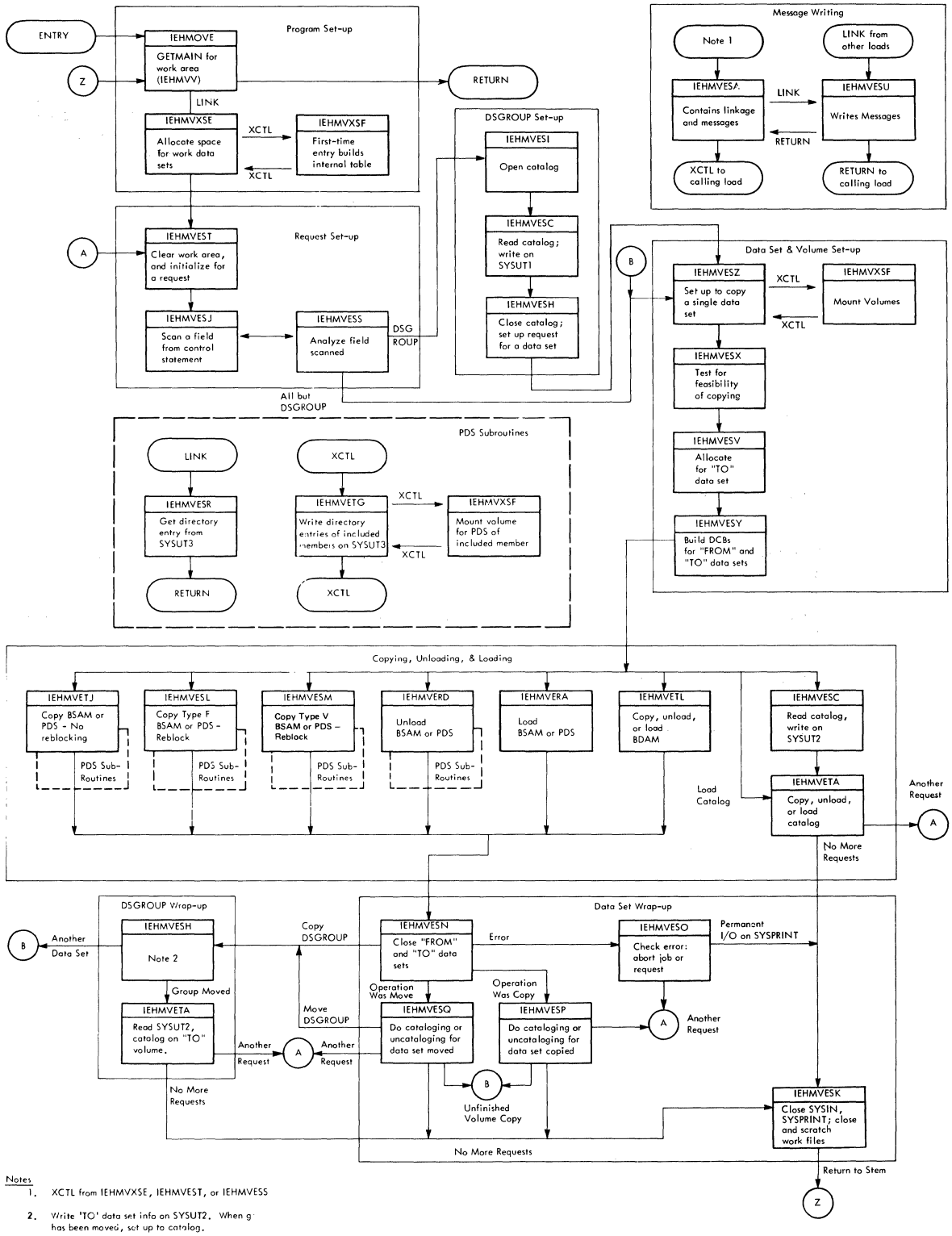


Figure 156. The Design of the IEHMOVE Program

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

"Chart 01. Overall Logic" on page 461 through "Chart 06. CATALOG Logic for CVOL Catalogs" show the logical flow of the program as follows:

Overall Logic	Chart 01
VOLUME Logic	Chart 02
PDS Logic	Chart 03
DSNAME Logic	Chart 04
DSGROUP Logic	Chart 05
CATALOG Logic	Chart 06

Control is passed between loads by means of transfer control (XCTL), except in the following cases:

- The stem, IEHMOVE, links to IEHMVXSE. The corresponding return to the stem is issued at the conclusion of the program by IEHMVESK.
- The message writer, IEHMVESU, is always linked to.
- IEHMVESR, a PDS subroutine that retrieves directory entries from a work data set, is always linked to.

PROGRAM STRUCTURE

PROGRAM SETUP (IEHMOVE, IEHMVXSE, IEHMVXSF)

The function of initializing the program for a job is performed by three loads: IEHMOVE, IEHMVXSE, and IEHMVXSF.

IEHMOVE
is the stem, which is present during the entire execution of the program. It obtains main storage for a work area (IEHMVV) to be used by the rest of the program.

IEHMVXSE
allocates space for the work data sets and opens them, clears the work area, and sets up an initial call to IEHMVXSF.

IEHMVXSF
is the volume mounter, described in "Device Allocation and Volume Mounting (IEHMVSSF, IEHMVXSF, and IEHVOLMT)" on page 3. The first-time entry of this routine builds the internal table used by the volume mounter.

REQUEST SETUP (IEHMVEST, IEHMVESJ, IEHMVESS)

The program is initialized to handle a single request by three loads: IEHMVEST, IEHMVESJ, and IEHMVESS.

IEHMVEST
initializes the work area for a request and sets up an initial call to IEHMVESJ.

IEHMVESJ
is the control statement scanner, described under "Control Card Scanner (RDCDRT)" on page 6.

IEHMVESS
analyzes the information scanned by IEHMVESJ. For a PDS request, IEHMVESS writes the following information:

- Member names to be included or to replace on SYSUT1 (the format is shown in Figure 157 on page 451).
- Member names to be excluded or to be replaced on SYSUT2 (the format is shown in Figure 158 on page 451).

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

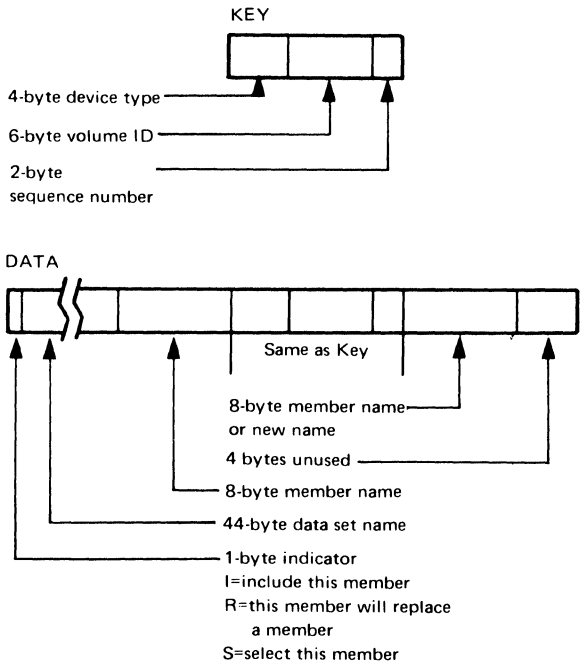


Figure 157. SYSUT1 Record Format (for a PDS Request Only)

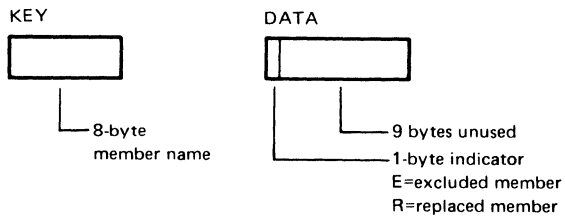


Figure 158. SYSUT2 Record Format (for a PDS Request Only)

MESSAGE WRITING (IEHMVESA, IEHMVESU)

All messages are written by IEHMVESU. Whenever IEHMVESS effects a message, it first interfaces through IEHMVESA, which contains messages and linkage to IEHMVESU.

DSGROUP SETUP (IEHMVESI, IEHMVESC, IEHMVESH)

Preliminary operations needed to copy a group of data sets are performed by three modules: IEHMVESI, IEHMVESC, and IEHMVESH. Modules IEHMVESI and IEHMVESC are bypassed unless a CVOL catalog

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

is specified by the user. See Figure 159 on page 453 for SYSUT1 record formats for CATALOG.

IEHMVESI
opens the CVOL and sets up a call to IEHMVESC.

IEHMVESC
reads the CVOL and writes data set information on SYSUT1.

IEHMVESH
for the case in which a CVOL is not specified, IEHMVESH performs a VSAM generic locate to obtain the names of all cataloged data sets in the group. A regular locate is then done on each name, and information is written on SYSUT1. For both the CVOL and non-CVOL cases, a single data set move/copy operation is set up for each entry on SYSUT1. IEHMVESH is reentered until all items on SYSUT1 have been processed.

DATA SET AND VOLUME SETUP (IEHMVESZ, IEHMVXSF, IEHMVESX, IEHMVESV, IEHMVESY)

Preliminary operations needed to copy a single data set are performed by five loads: IEHMVESZ, IEHMVXSF, IEHMVESX, IEHMVESV, and IEHMVESY.

IEHMVESZ
examines the request and sets up a call to IEHMVXSF. For a VOLUME request, IEHMVESZ reads the VTOC and obtains a DSCB.

If an abnormal termination is indicated as the result of an error in either this module or a called subroutine, this module initiates termination of the program. User label exits are not processed at this time.

IEHMVXSF
is the volume mounter, described in "Device Allocation and Volume Mounting (IEHMVSSF, IEHMVXSF, and IEHVOLMT)" on page 3. This execution of the volume mounter effects volume mounting.

IEHMVESX
performs tests on the data set to be copied. Tests include movability, unload or load, access method type, compatible block size, and compatible receiving device.

When user-label processing has been specified, this module processes the input header labels as it opens the input data set. If storage in which to save the labels is required, it is obtained in this module.

IEHMVESV
causes space to be allocated for the "TO" data set. If user labels exist in the "TO" data set and user-label processing has been requested, an additional track will be allocated for the user labels. The DS1EXT1 field of the DSCB is modified for this purpose. (For a preallocated "TO" data set, one that has been allocated before the execution of the IEHMOVE program, the user must provide a user-label track to permit the labels to be moved.) If the "FROM" data set is a PDS, the directory entries of members to be copied are written on SYSUT3. During abnormal terminations that are handled by this module, no user-label exits are processed. If a user-label track has not been allocated, the message text in this load module informs the user that labels cannot be moved or copied.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

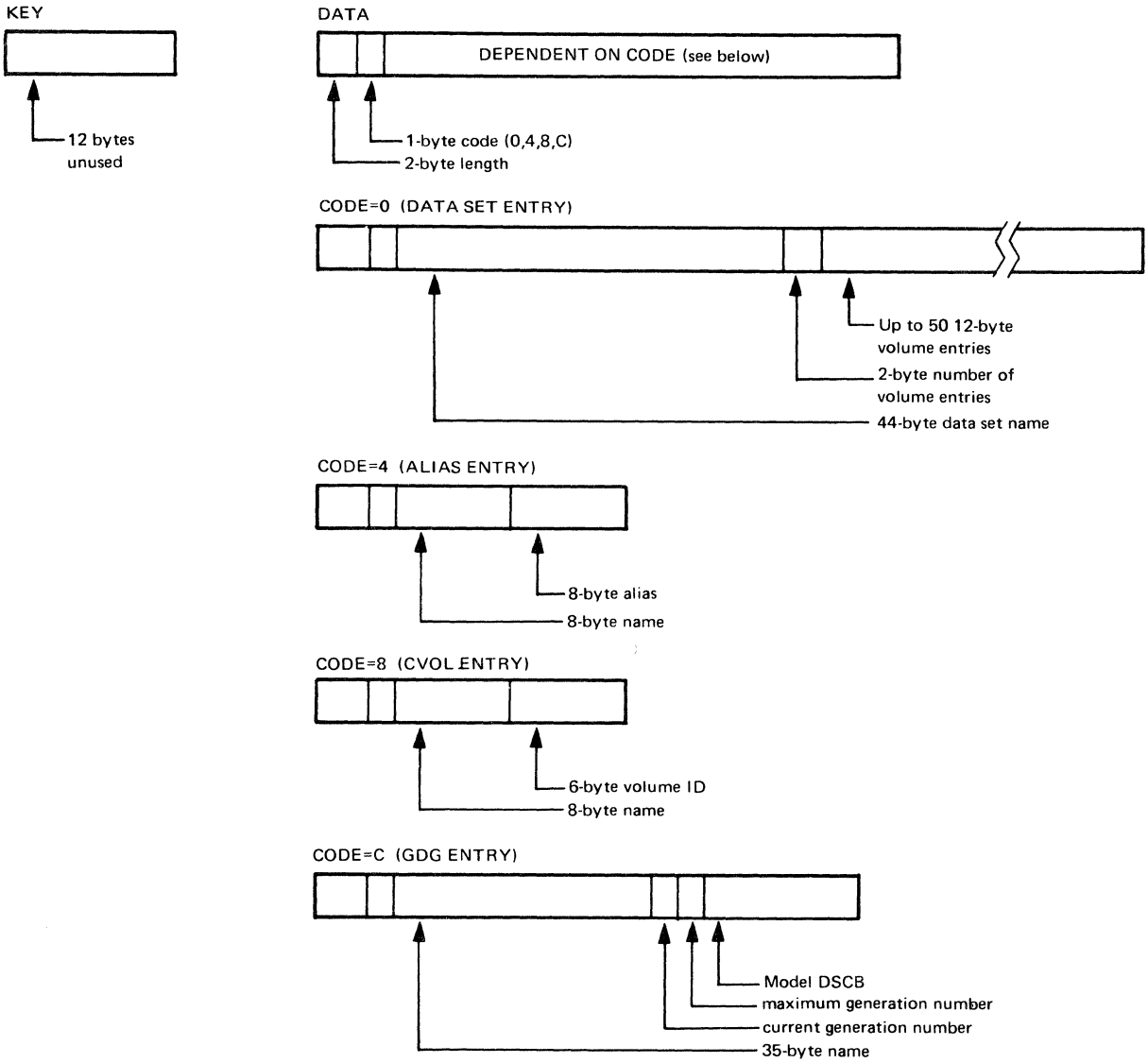


Figure 159. SYSUT1 and SYSUT2 Record Formats for DSGROUP; SYSUT1 Record Formats for CATALOG

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEHMVESV defines a discrete profile of the "TO" data set to RACF (via RACROUTE with REQUEST=DEFINE) under the following conditions:

- COPYAUTH is specified and the user has alter authority to the RACF-defined input data set.
- The user has the automatic data set protection (ADSP) attribute.

If the data set name is listed in a RACF generic profile, a RACROUTE REQUEST=DEFINE is issued to verify that the user has authority to allocate a data set with that name.

IEHMVESY
builds, using DCB and label information specified by FROMDD and TODD (if given in an operation involving 7-track or 9-track unlabeled tape), the DCBs for the "FROM" and "TO" data sets and directs control to the appropriate module to copy, unload, or load the data set.

When user-label processing has been specified, this module processes the output of the user header labels that have been saved by the program, as it opens the output DCB.

If this module encounters errors such that an abnormal termination is indicated, the module initiates termination of the program. If user-label processing has been specified, the processing operations are not performed during the abnormal termination procedures.

If variable spanned records are indicated, this module will identify the record format and determine to which module control is to be given for processing each record. If logical copies involving changes in the record format (RECFM), the block size (BLKSIZE), or the logical record length (LRECL) of data set records are attempted, an error is indicated and a message is printed. The program will then attempt to move the rest of the data sets as requested.

This module also writes the first records of an unloaded data set and determines the module that next receives control to perform the actual move/copy operation.

PDS SUBROUTINES (IEHMVESR, IEHMVETG, IEHMVXSF)

If a partitioned data set is being copied or unloaded, IEHMVESR is always used by the copying module, whereas IEHMVETG (which uses IEHMVXSF, the volume mounter) is used if the request specifies PDS. Figure 156 on page 449 shows which loads use PDS subroutines.

IEHMVESR
is used to obtain a directory entry from the work data set SYSUT3. If the directory entry is from the PDS being copied, it was placed on SYSUT3 by IEHMVESV (data set and volume setup); if the directory entry is from an INCLUDE or REPLACE option, it was placed on SYSUT3 by IEHMVETG. The format of a directory entry on SYSUT3 is shown in Figure 160 on page 455.

IEHMVETG
places directory entries of members from INCLUDE, REPLACE, or SELECT options on SYSUT3. Each execution of IEHMVETG places the directory entries to be included from one PDS. When there are no more members to be included in the copy, IEHMVETG gives control to IEHMVESN. IEHMVXSF, the volume mounter, is used as needed. The logic of IEHMVXSF is described in "Device Allocation and Volume Mounting (IEHMVSSF, IEHMVXSF, and IEHVOLMT)" on page 3.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

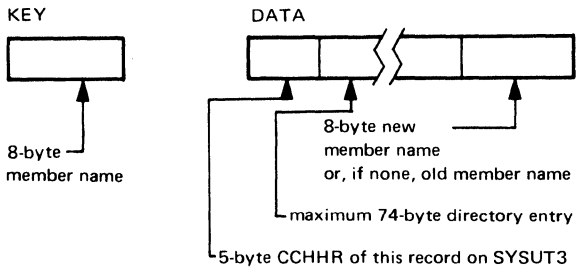


Figure 160. SYSUT3 Record Format

COPYING, UNLOADING, AND LOADING

The load modules used to copy, unload, and load data are grouped according to type of data set and format condition as shown in Figure 161. The DCB attributes for unloaded data sets are RECFM=FB, LRECL=80, and BLKSIZE=800. The modules are described below.

Type of Data Set	Format Condition	Load Modules Used
CVOL	Normal	IEHMVETA with IEHMVESC
CVOL	Previously unloaded	IEHMVETA
BDAM	Any	IEHMVETL
PDS or BSAM	Normal, copyable, no reblocking	IEHMVETJ with IEHMVETG, (PDS only) IEHMVESR IEHMVXSF
PDS or BSAM	Type F, copyable with reblocking	IEHMVESL with IEHMVETG, (PDS only) IEHMVESR, (PDS only) IEHMVXSF
PDS or BSAM	Type V, copyable with reblocking	IEHMVESM with IEHMVETG, (PDS only) IEHMVESR IEHMVXSR
PDS or BSAM	Normal, uncopyable must be unloaded	IEHMVERD with IEHMVESR, IEHMVXSF
PDS or BSAM	Previously unloaded	IEHMVERA

Figure 161. Load Module Grouping for Copying, Unloading, and Loading

IEHMVETA
copies, unloads, or loads a CVOL catalog. If the catalog is to be loaded, it is in the format shown in Figure 159. The entries are then cataloged in a CVOL on the receiving volume. If the CVOL catalog is to be unloaded or copied, IEHMVESC first writes catalog entries on SYSUT1 as shown in Figure 159. IEHMVETA then catalogs the SYSUT1 entries in a CVOL or the receiving volume (for a copy) or writes them in the same format (for an unload).

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEHMVETL
copies, unloads, or loads a BDAM data set. The data set is copied using BDAM-read and BSAM-write (load mode) routines. If the input data set record format is type U, the block length of each physical record is read and calculated and then passed to BSAM write. This calculation is unnecessary for types F and V and is bypassed.

If user-label processing has been specified, this module obtains any necessary storage in which to save the labels. This is done when either the end of a data set has been reached or a switch to another volume is to be made. The saved labels will be passed to the data set wrapup routines.

IEHMVETJ
copies a BSAM data set or a PDS. A BSAM data set is copied using BSAM in a simple read-write loop. When either the end of a data set or the end of a volume has been reached in reading, storage will be obtained, if necessary, for saving any labels for which processing has been requested. The saved labels will be passed to the data set wrapup routines. For a PDS, at the time IEHMVETJ receives control, the directory entries of members (barring any EXCLUDEs or REPLACEs) of the "FROM" PDS are on SYSUT3, where they were written by IEHMVESV (data set and volume setup).

The "FROM" PDS is then copied as follows:

1. IEHMVETJ directs control to IEHMVESR, which reads from SYSUT3 one directory entry of the PDS.

2. If the entry indicates a note list is present, IEHMVETJ reads the note list. Using BSAM, IEHMVETJ then reads and writes member records up to the note list. The note list is then updated and written with the new TTRs of the members.

3. When all note lists and member records have been written, the directory entry is updated with the new note list addresses and then stowed. This process is repeated for each directory entry on SYSUT3. When the "FROM" PDS has been copied, control is given to IEHMVETG to write on SYSUT3 the directory entries for all members to be included, selected, or replaced from another PDS. IEHMVETJ then copies these members as outlined. IEHMVXSF (the volume mounter) is given control by IEHMVETG as needed.

IEHMVESL and IEHMVESM
copy partitioned and BSAM data sets if the "TO" data set has been preallocated (that is, before execution of IEHMOVE) and the "FROM" and "TO" DSCBs indicate that reblocking is necessary in order to perform copying.

IEHMVESL
copies (with reblocking) a PDS or BSAM data set having type F record format. Blocks are read (using BSAM) until the output block size is exceeded, and then logical records are sectioned from the high-order end of the buffer until the output block size is reached. The block is then written, using BSAM. The last block (of a BSAM data set or of a member of a PDS) is written as a truncated block if necessary. For a PDS, any user TTRs are ignored.

If user-label processing has been requested, this module will, when reaching either the end of a data set or the end of a volume, obtain necessary storage in which to save the labels. When the module passes control to the data set wrapup routines, the saved labels are passed to the routine that receives control.

IEHMVESM
copies (with reblocking) a PDS or BSAM data set having type V record format. The operation is similar to that of

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEHMVESL: Blocks are read using BSAM until the maximum output block size is exceeded, and then logical records are sectioned from the high-order end of the buffer until the size of the output buffer is not greater than the maximum block size. As with IEHMVESL, any user TTRs are ignored in a PDS.

If user-label processing has been requested, this module will, when reaching either the end of a data set or the end of a volume, obtain necessary storage in which to save the labels. When the module passes control to the data set wrapup routines, the saved labels are passed to the routine that receives control.

IEHMVERD
unloads a PDS or a BSAM data set. For a BSAM data set, the data set is read one block at a time. After each read, the block is prefixed by 3 to 6 bytes of control information, and then deblocked into 78-byte sections. Each section is then prefixed with a 2-byte physical sequence number. The resultant 80-byte blocks are then written, or, if the receiving device permits, are reblocked in groups of 10 to be written out as 800-byte blocks. The last block written is padded with blanks. For a PDS, the directory entry of a member is first read into the buffer. Each note list is read and followed by member records which precede it in the PDS. Aliases are read last. The buffer is sectioned and control information inserted. The process is repeated for each directory entry and its member blocks. Directory entries are read from SYSUT3 by IEHMVESR. The INCLUDE, REPLACE and SELECT options are ignored in unloading; the EXCLUDE option is not ignored.

If user-label processing has been requested, this module will, when reaching either the end of a data set or the end of a volume, obtain necessary storage in which to save the labels. When the module passes control to the data set wrapup routines, the saved labels are passed to the routine that receives control.

IEHMVERA
loads a PDS or BSAM data set. For a BSAM data set, a block is read, the control information is removed and analyzed, and successive blocks are read until a block from the original data set is reconstructed. The block is then written, and the process is repeated until the original data set is loaded. For a PDS, the process is similar: The directory entry is first reconstructed, but is not stowed until member blocks have been written and note lists updated and written. At the time the data set was unloaded, an entry was written, followed by member blocks (in unloaded format).

If user-label processing has been requested, this module will, when reaching either the end of a data set or the end of a volume, obtain necessary storage in which to save the labels. When the module passes control to the data set wrapup routines, the saved labels are passed to the routine that receives control.

DSGROUP WRAPUP (IEHMVESH, IEHMVETA)

After each data set of a DSGROUP has been copied, control is given to IEHMVESN of the data set wrapup portion of the program. If scratching of the "FROM" data set is necessary (for a MOVE DSGROUP), control is given to IEHMVESQ, which scratches the "FROM" data set and gives control to IEHMVESH. If scratching is not necessary, control goes directly from IEHMVESN to IEHMVESH.

IEHMVESH
If the user specified a CVOL catalog, the SYSUT1 "TO" data set information is written, in the same format, on SYSUT2,

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

so that the catalog can be updated if needed. After all data sets have been copied, IEHMVESH then invokes IEHMVETA to process the entries on SYSUT2.

IEHMVETA
reads SYSUT2 and catalogs the information. The process is the same as that followed by IEHMVETA in copying a CVOL catalog.

DATA SET WRAPUP (IEHMVESN, IEHMVESO, IEHMVESP, IEHMVESQ, IEHMVESK)

Load modules in this group perform terminal operations following the copying, unloading, or loading of a data set processed for a PDS, DATA SET, or VOLUME request. In addition, when all requests have been serviced, control is given to IEHMVESK.

IEHMVESN
completes the moving or copying of a data set and closes the "TO" and "FROM" data sets.

If user labels have been specified, and if output trailer labels have been saved in storage, these labels are written out and the storage area is released. If a user-label track has not been allocated, the message text in this load module informs the user that labels cannot be moved or copied.

IEHMVESO
is entered following an unsuccessful copying, unloading, or loading operation, or following a test (data set and volume setup) indicating a request could not be honored. IEHMVESO prints a diagnostic message and scratches the "TO" data set.

IEHMVESP
performs terminal operations following a COPY request, including any specified or implied cataloging, uncataloging, and scratching.

IEHMVESQ
performs terminal operations following a MOVE request, including any specified or implied cataloging, uncataloging, and scratching.

IEHMVESK
is entered when all requests have been serviced, or when a permanent I/O error has been detected on the printer. IEHMVESK closes SYSIN and SYSPRINT, closes and scratches the work data sets, frees main storage, and returns control to the stem, IEHMOVE. During abnormal termination handled by this module, user-label exits are not processed.

COMMUNICATION AREA (IEHMVV)

The communication area for the program is defined at assembly time by the macro instruction IEHMVV, which is internal to the IEHMOVE program. Register 12 contains a pointer to the communication area whenever a request for a module is issued.

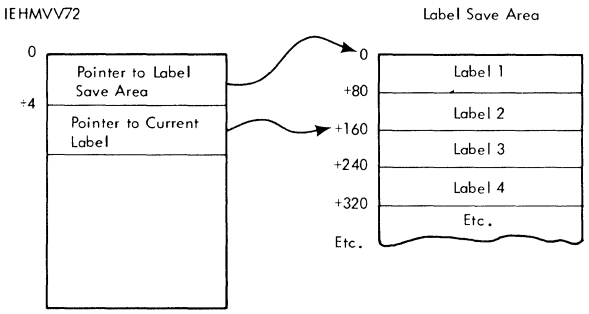
The macro instruction IEHMVV generates a dummy section (also IEHMVV) containing work areas and control data for all object modules of the program. Storage for the dummy section IEHMVV is obtained dynamically by the stem.

The communication area consists of the following parts:

- A work area of 512 bytes (IEHMV00).
- The addresses of the beginning and end of an 800-byte work area (IEHMVV10).

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

- A table of switches controlling the flow of the program (IEHMVV20).
- A control table containing the return codes of the control statement scan routine (IEHMVESJ).
- A table of control data for volume lists and include-exclude-replace lists (IEHMVV21-IEHMVV26).
- A table of addresses of the "FROM" data set's DCB, DSCB, DECB and ddname (IEHMVV30) and the "TO" data set's DCB, DECB, and ddname (IEHMVV31). Each address is stored in a fullword.
- A table of the addresses of the DCBs and DECBs of SYSPRINT (IEHMVV33), SYSIN (IEHMVV32), and SYSLIB (IEHMVV34).
- A table of work data-set control data (IEHMVV39).
- A table of addresses of work areas for loading, unloading, including, replacing, and copying a PDS.
- The DCB exit list (for user-label processing) defined by the macro instruction IEHDCBXL. This list is found in the 40-byte section IEHMVV70 of the communication area. Included in the list are symbolic names for the input and output header-label processing subroutines, the input and output trailer-label processing subroutines, the DCB exit, and the OPENJ JFCB exit.
- An area containing pointers both to the storage area (the label save area) obtained for user labels and to the current label being processed. These pointers are in the 20-byte section IEHMVV72. For the first label being processed, both pointers will indicate the same address. Figure 162 indicates these relationships.



Example of pointers when third label is being processed

Figure 162. Label Save Area Pointers

IEHMOVE WORK DATA-SET RECORD FORMATS

The program uses three work data sets: SYSUT1, SYSUT2, and SYSUT3. How a work data set is used depends on the function being performed by the program.

The device on which the work data sets reside is allocated by job management and is associated with the ddname SYSUT1. The spaces occupied by the work data sets specified by the names SYSUT1, SYSUT2, and SYSUT3 are dynamically allocated.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

OBTAINING SPACE FOR A WORK DATA SET

Space for a work data set (for example, SYSUT3) is obtained from DADSM in the following way:

1. The first time space is requested, it is requested for the data set **SYSUT3.

2. If the return from DADSM indicates that a DSCB for the requested data set space already exists on the VTOC, the name previously submitted to DADSM is qualified by the index name consisting of the single character * and the modified request is submitted to DADSM.

Step 2 is repeated until space is allocated or until 44 bytes have been used with no success. Thus, the first request for space for SYSUT3 either results in space being allocated for the 8-byte name **SYSUT3 or an indication that a data set of the name **SYSUT3 already resides on the subject volume. If the latter, space is requested for the data set **SYSUT3.*. The third request, if necessary, would specify the name **SYSUT3.*.*. A count of the number of times the name has been qualified is maintained in the communication area, IEHMVV.

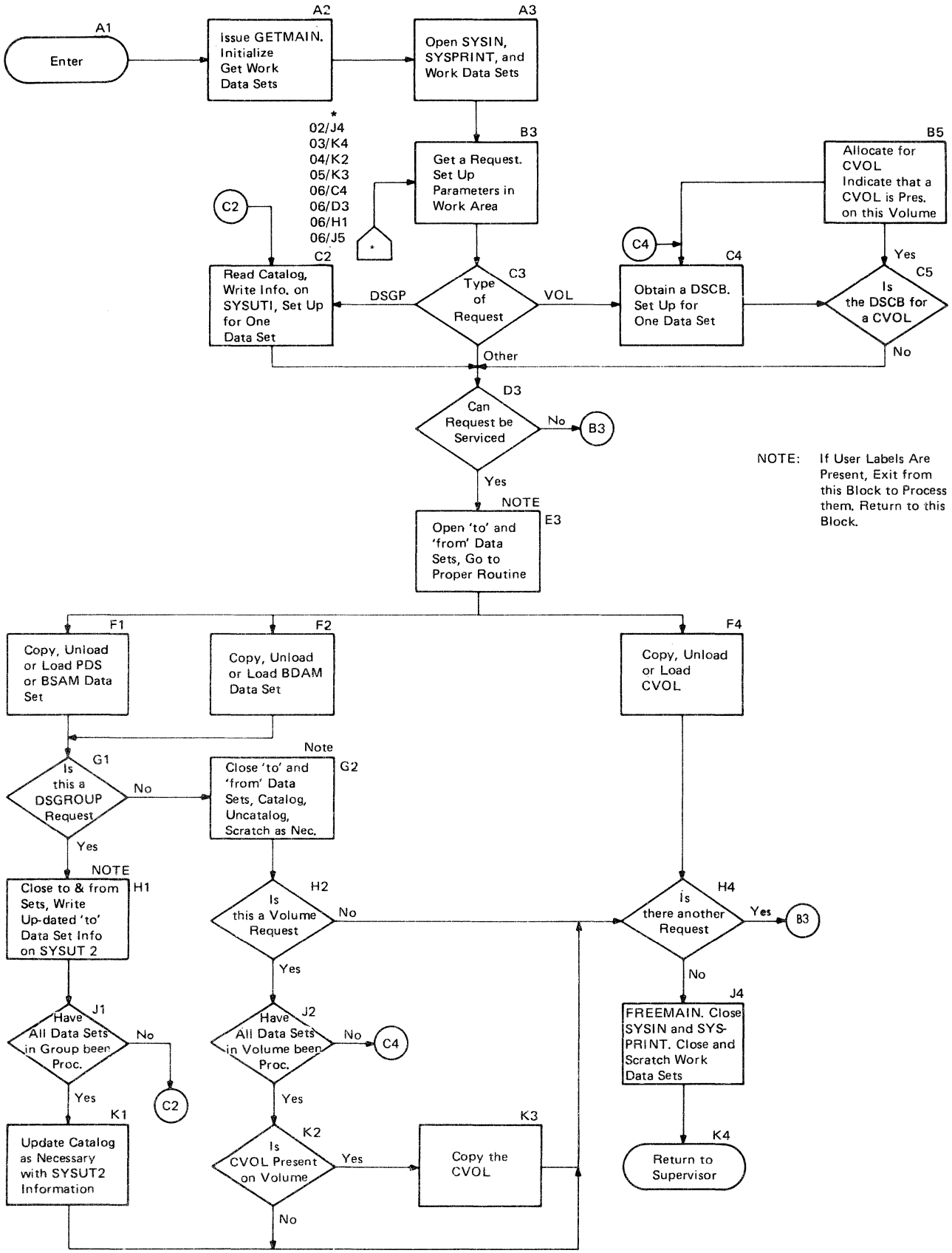
After space has been allocated for the work data sets, they are opened in the order: SYSUT1, SYSUT2, SYSUT3.

RELEASING SPACE USED BY A WORK DATA SET

The work data sets are not closed until final wrapup. At this time, SYSUT3 is first closed and scratched, then SYSUT2 is renamed to SYSUT3 and closed and scratched, and then SYSUT1 is renamed to SYSUT3 and closed and scratched.

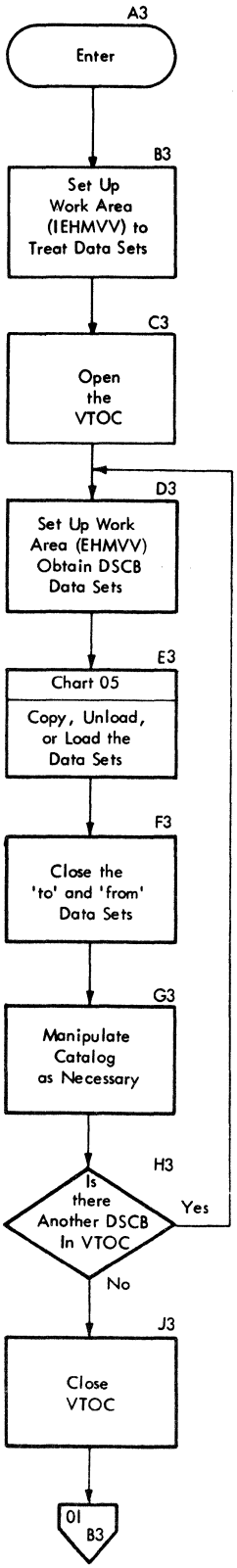
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 01. OVERALL LOGIC



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

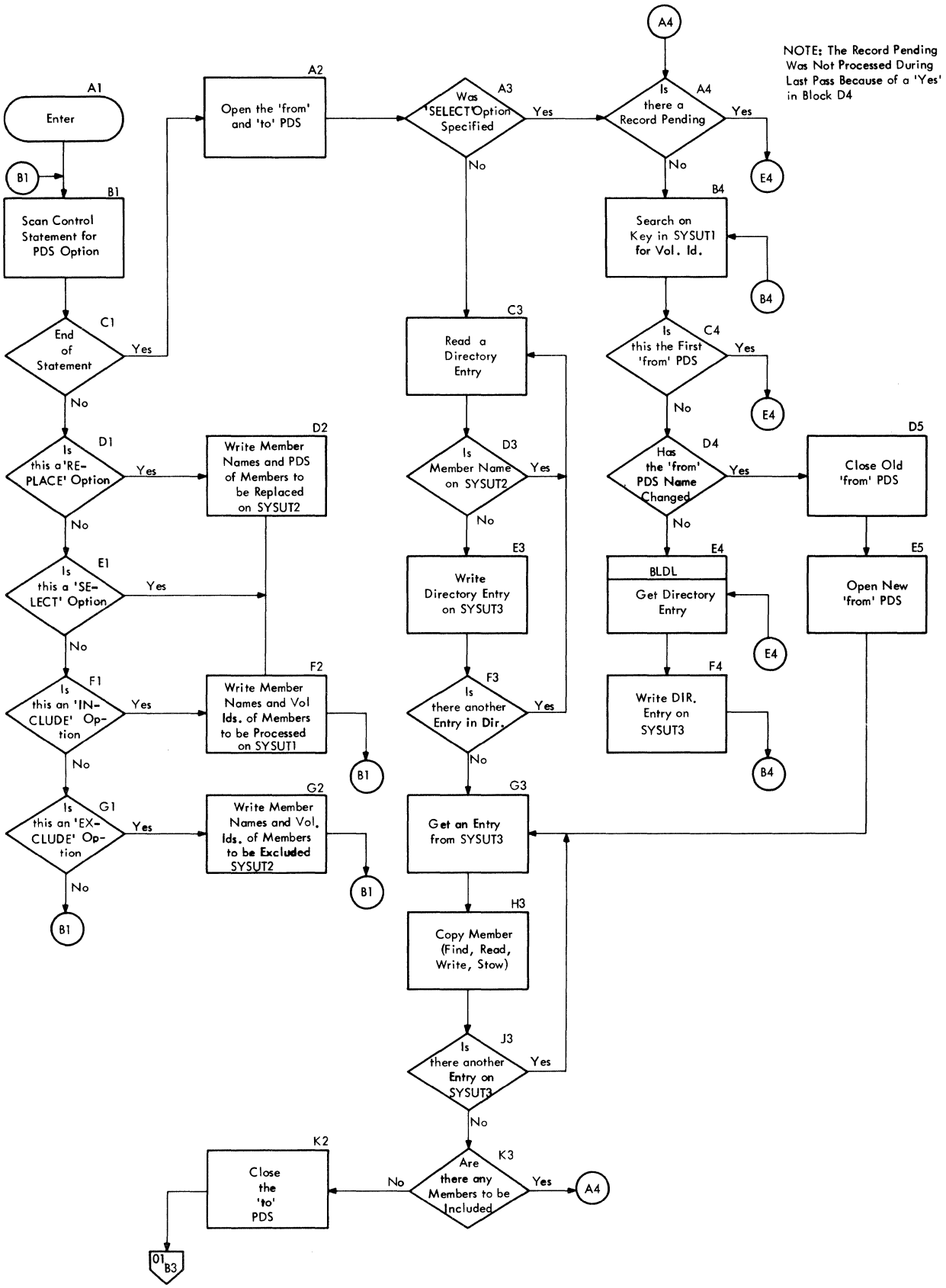
CHART 02. VOLUME LOGIC



NOTE: If user labels are present, exit from this block to process them. Return to this block.

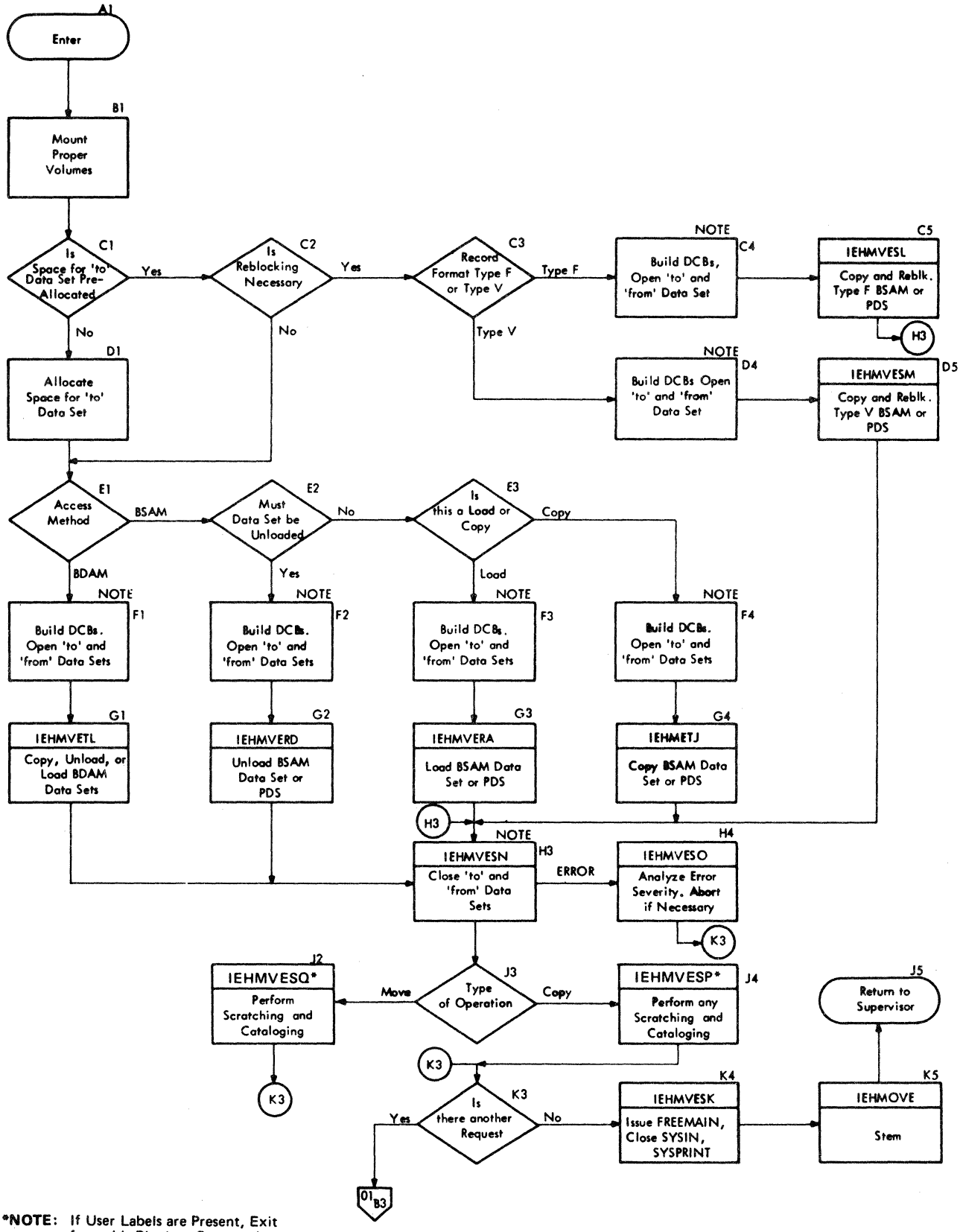
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 03. PDS LOGIC



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

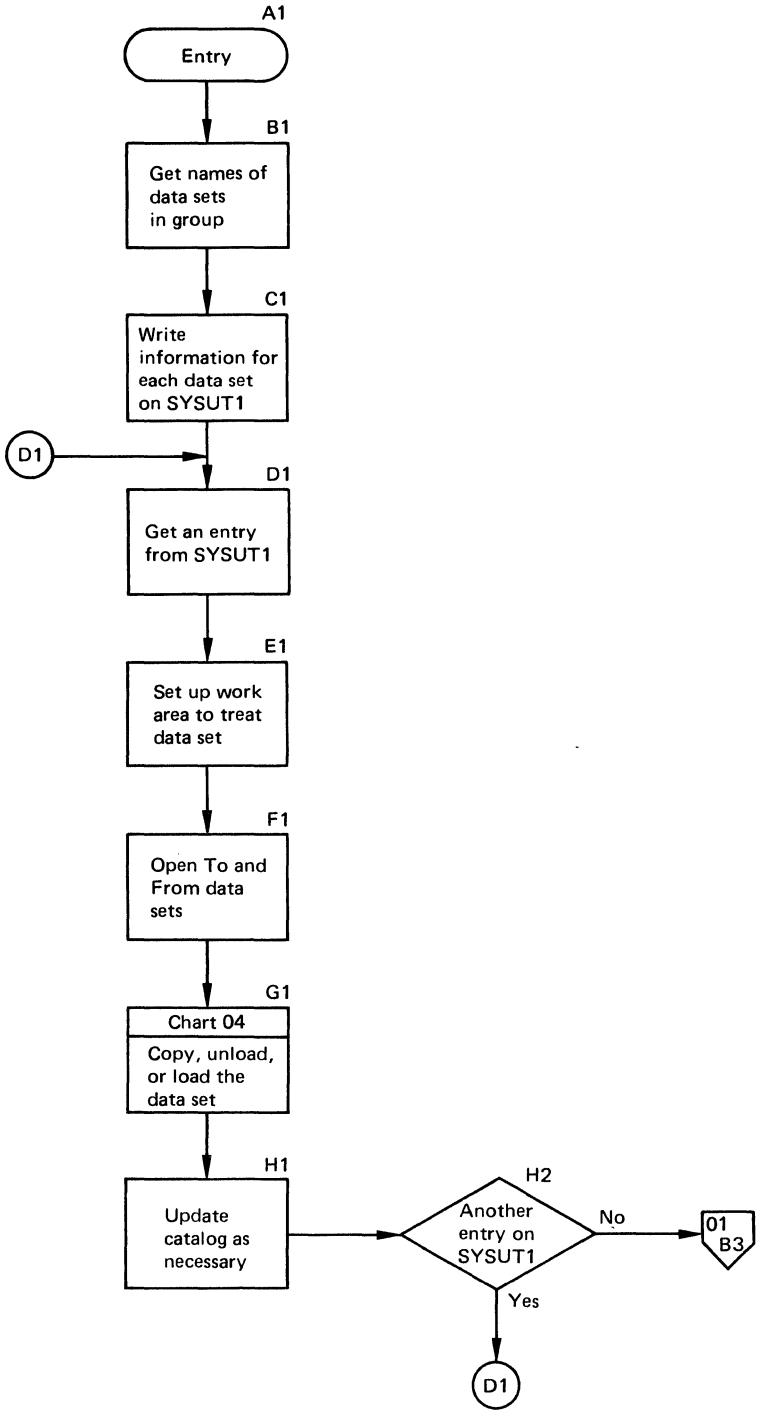
CHART 04. DSNAME LOGIC



*NOTE: If User Labels are Present, Exit from this Block to Process them. Return to this Block.

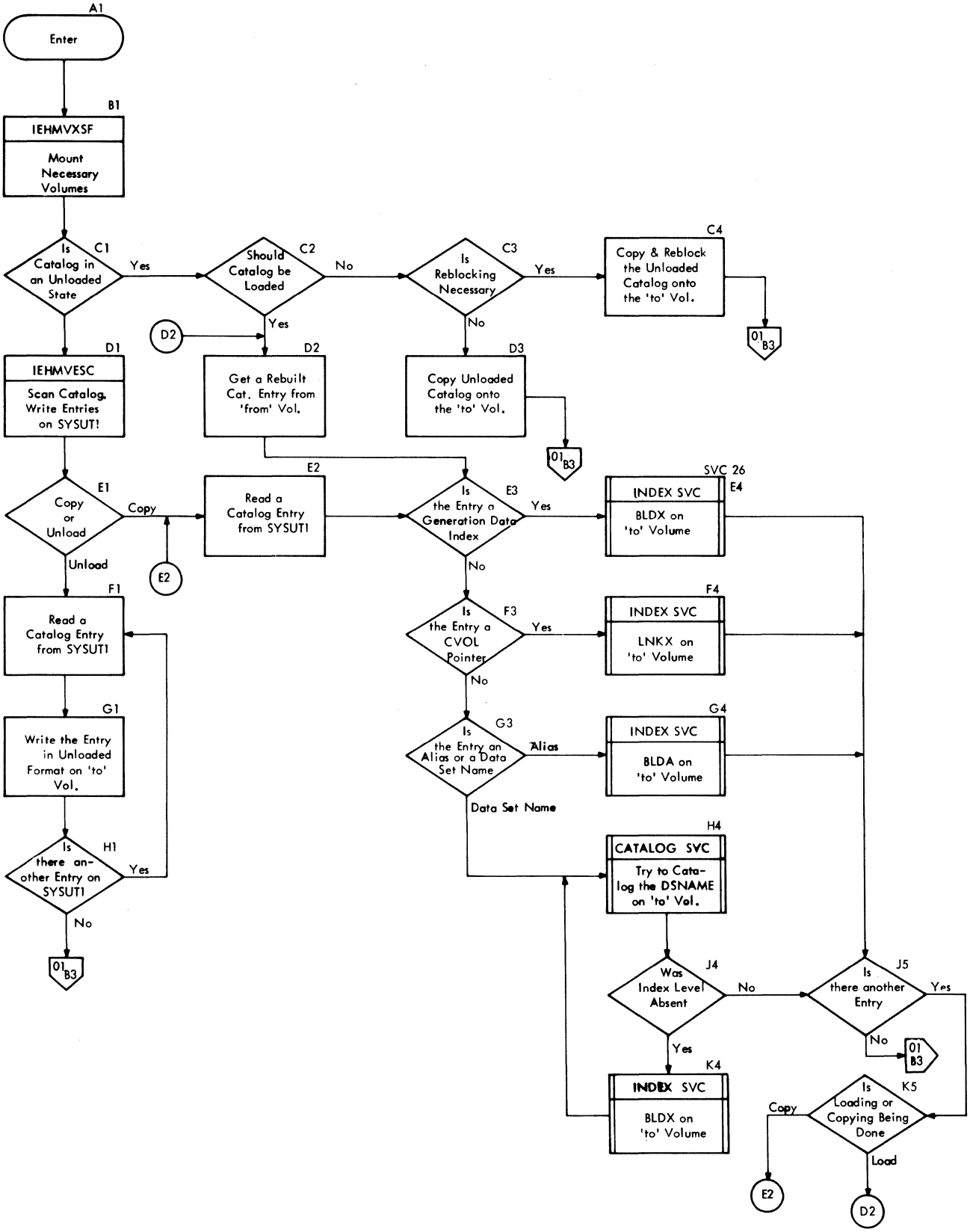
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 05. DSGROUP LOGIC



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 06. CATALOG LOGIC FOR CVOL CATALOGS



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEHPROGM PROGRAM

The IEHPROGM program is an interface between the user and data management routines that modify volume and catalog structures. By means of utility control statements, the user may request the IEHPROGM program to:

- Scratch or rename a data set
- Scratch or rename a PDS member
- Scratch a data set assigned by the operating system
- Build or delete an index level, index alias, or generation data group in a CVOL catalog
- Connect or release two CVOLs
- Catalog or uncatalog a non-VSAM data set
- Add, replace, or delete data set passwords and list password entries in the PASSWORD data set

The general design of the program is shown in Figure 163 on page 468. For each request listed above, the program issues one or more supervisor calls (SVCs) for data management routines that perform the requested service; the IEHPROGM program interfaces with these routines by building parameter lists for them, invoking them, and analyzing their returns.

Following the return from a data management routine, further processing by the IEHPROGM program may include additional calls to data management routines.

PROGRAM STRUCTURE

The program consists of six load modules (a seventh, which is shared, is part of IEHMOVE) and a dynamically obtained work area:

- The Root resides in main storage throughout the program's execution. It contains V-type address constants needed by the overlay supervisor.
- The Parameter List Builder initializes the program, obtains and analyzes requests, and builds a parameter list for the appropriate data management routine.
- The SVC Return Analyzer issues the appropriate SVC and analyzes its return. In some instances, additional SVCs may be issued by this module.
- The Auxiliary Parameter Analyzer analyzes auxiliary parameters supplied to the IEHPROGM program by a calling program, and also opens SYSIN and SYSPRINT.
- The Message Writer writes all diagnostic messages and operator instructions issued by the program.
- The Volume Lookup obtains volume information from supervisor routine IEFEB4UV when the keyword parameter VOL occurs in a request.
- The Volume Mounter ensures that all volumes needed to service the request are mounted or mountable. This load module (IEHMVSSF) is shared by and is part of the IEHMOVE component.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

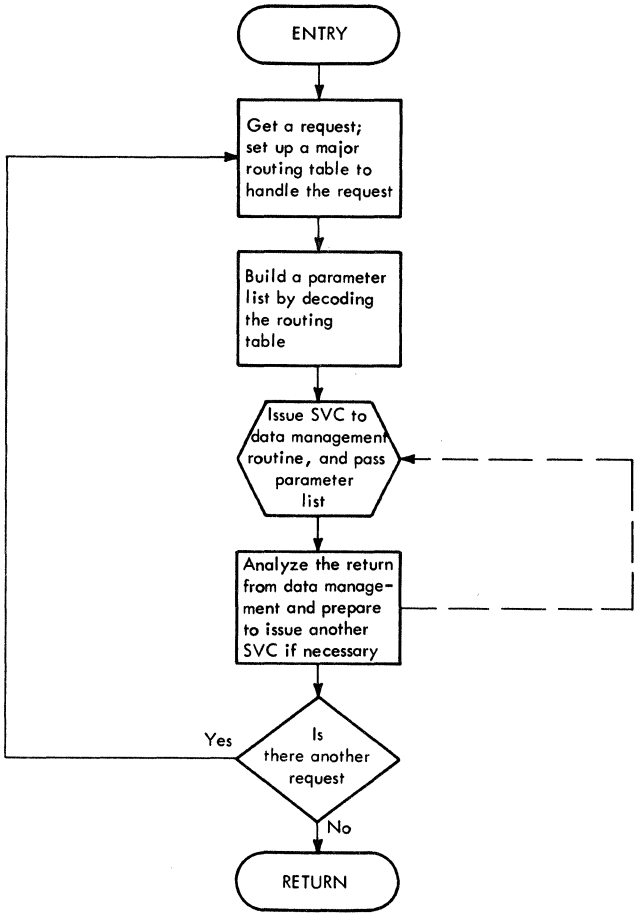


Figure 163. The General Design of the IEHPROGM Program

- The Work Area is obtained dynamically by the parameter list builder.

The structure of the program is shown in Figure 164 on page 469; each block represents a control section (CSECT).

The structural flow of the program is shown in Figure 165 on page 470. The auxiliary parameter analyzer, message writer, and volume lookup are subordinate load modules; the others are control load modules.

CONTROL LOAD MODULES

The root, the parameter list builder, the volume mounter, and the SVC return analyzer are control load modules.

THE ROOT (IEHPROG1)

The root consists of one CSECT, IEHPROG1. It contains V-type address constants and a branch instruction to the parameter list builder.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

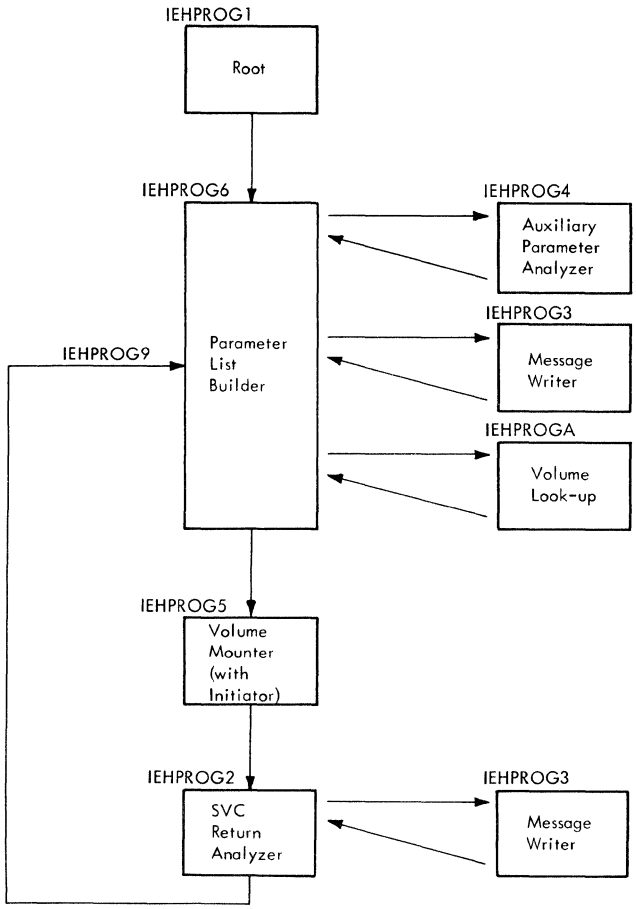


Figure 164. The Structure of the IEHPROGM Program (Each block represents one control section.)

THE PARAMETER LIST BUILDER (IEHPROG6, IEHPROGB, IEHPROGC)

The parameter list builder contains three CSECTs: IEHPROG6, IEHPROGB, and IEHPROGC.

IEHPROG6
builds the parameter list for the initial SVC to the appropriate data management routine. It contains the following routines: COMMENCE, IEHPROG9, FNDECODE, FDLD, KODECODE, IEHESCAN, IEHETLU, DSLNGTH, INIT, PASWORD1, PASWORD2, CPASWORD, DATA, and TYPE. The parameter list is built at location IEHEMAC1 in the work area.

COMMENCE
initializes the program by establishing addressability and obtaining a work area of 5246 bytes.

IEHPROG9
initializes for a new request by resetting switches.

FNDECODE
determines the operation (for example, SCRATCH) requested, and stores the address of its routing table

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

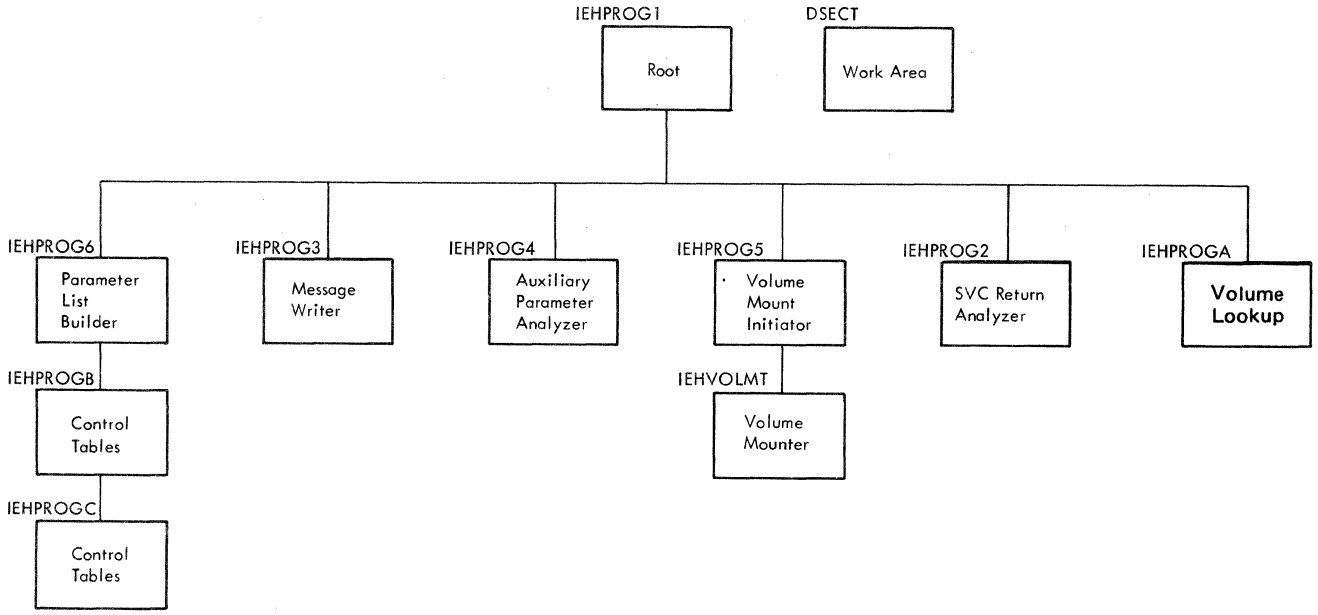


Figure 165. The Structural Flow of IEHPROGM Program (Each block represents one load module.)

at location FINUSE in the work area. The use and format of the routing tables are discussed under "Program Flow."

FDLD
decodes the routing table in use. Each routing table indicates a sequence of operations to be performed; FDLD effects these operations by decoding the routing table.

KODECODE
causes keyword parameters to be scanned from a utility control statement, and successively directs control to subroutines that move parameter data to the parameter list for the data management SVC. (The list is at location IEHMAC1 in the work area.)

IEHESCAN
scans a control statement for an operation or an operand.

IEHETLU
performs a table lookup for the address of a routine or a routing table: If the search argument used is an operation, a routing table address is retrieved; if the search argument used is an operand, a routine address is retrieved.

DSLNGTH
sets the length of the data set name in the SVC 98 parameter list.

INIT
initializes counters that record the number of invalid passwords supplied per password entry on each utility statement and per job step.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

PASWORD1, PASWORD2, CPASWORD, DATA, and TYPE
check for duplicate and required operands on the ADD, DELETEP, REPLACE, and LIST utility statements, and build a portion of the SVC 98 parameter list.

IEHPROGB
contains the following tables: TABLEN3, TABLEN4, TABLEN5, TABLEN6, TABLEN7, TABLEN8, CATALOG, and UNCATLG.

Each of the tables consist of a variable number of entries: The first n bytes of each entry are an operation or keyword operand, and the last 4 bytes of each entry are the address of a routing table (for an operation) or a routine (for a keyword operand). As an example, TABLEN6 contains as typical entries

```
C'RENAME'   AL4(RENAME)
C'DSNAME'   AL4(DSNAME)
```

where RENAME is the symbolic location of the RENAME routing table, and DSNAME is the symbolic location of the routine given control when the data set name is scanned.

The tables CATALOG and UNCATLG are major routing tables for the catalog and uncatalog operations on non-VSAM data sets.

IEHPROGC
contains the major routing tables DELETEX, CONNECT, RELEASE, BUILDA, DELETEA, BLDX, SCRATCH, RENAME, ADD, REPLACE, DETETEP, and LIST. The use and format of all routing tables are discussed under "Program Flow."

THE VOLUME MOUNTER (IEHPROG5, IEHVOLMT)

This function ensures that all volumes needed by the data management routine are mounted or mountable. CSECTs involved are IEHPROG5 and IEHVOLMT (part of the IEHMVSSF load module).

IEHPROG5
is entered from IEHPROG6 when the parameter list for the initial data management SVC has been built. IEHPROG5 then calls IEHVOLMT, if necessary, to ensure that a needed volume is already mounted or is mountable. IEHVOLMT is called in the following cases:

- For a SCRATCH or RENAME request, the volume ID from the control statement is passed to IEHVOLMT, together with an indication that the volume is not to be mounted. (The scratch and rename data management routines themselves perform volume mounting.) The call to IEHVOLMT serves as a check that the volume is mountable.
- For any other request, if CVOL is specified, IEHVOLMT is passed the volume ID, together with an indication that the volume is to be mounted. Otherwise, IEHVOLMT is not called.

For both cases above, IEHVOLMT normally returns a pointer to a ddname associated with a channel-unit on which the volume is mounted or can be mounted. IEHPROG5 then inserts the ddname into the data control block (DCB) needed to perform I/O operations on the volume. Control is then given to IEHPROG2.

If the volume is not mountable, as indicated by the return from IEHVOLMT, IEHPROG5 aborts the request, giving control to IEHPROG6 to honor the next request.

IEHVOLMT is described under "Device Allocation and Volume Mounting (IEHMVSSF, IEHMVXSF, and IEHVOLMT)" on page 3.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

THE SVC RETURN ANALYZER (IEHPROG2)

This function issues and analyzes the returns of all data management SVCs used by IEHPROGM to perform a requested operation. The SVC return analyzer segment consists of a single control section, IEHPROG2, which contains the following routines: LATAB, FDLD, OPENVTOC, GETADSCB, SVCRET, SVC26RET, VTOCRET, WTORPAS1, WTORPAS2, and SVCPRO.

LATAB
stores the address of the routing table in use at location FINUSE in the work area.

FDLD
decodes the routing table, directing control to the routines indicated by the table. (This is the same FDLD present in IEHPROG6, the parameter list builder.)

OPENVTOC
constructs a DCB for the VTOC to be opened for a scratch VTOC request, and then opens the VTOC.

GETADSCB
reads the VTOC and scratches the following DSCBs for a scratch VTOC request:

- If the SYS keyword was specified, each system-assigned data set (a data set having a name beginning either with the 36 characters:

  AAAAAAAA.AAAAAAAA.AAAAAAAA.AAAAAAAA.

  or with the characters SYSnnnnn.T and containing one of the characters P, A, or V in the 19th position) is scratched.

  Note: Each character "n" represents a digit from 0 to 9.

- If the SYS keyword was not specified, each identifier (Format 1) DSCB is scratched.

The VTOC is closed when an EOF is detected in reading it.

SVCRET
scratch (SVC 29) and rename (SVC 30) routines. The return is used as an indexing factor on the branch table, BRANTAB, giving control to an appropriate diagnostic routine.

SVC26RET
interprets returns from the catalog routine. SVC26RET uses BRANTAB in the same manner as SVCRET. The difference between the two routines is that SVC26RET also interprets the return from the locate routine.

INDEX8
gives control to SVC26RET to interpret an error return from the locate routine.

NEEDINDX
is entered the first time SVC26RET detects that an index level supplied in a utility control statement was not found in the catalog. NEEDINDX passes the index name to SCANIT.

SCANIT
constructs a parameter list for an SVC to the index routine each time a return shows that an index level was absent.

VTOCRET
effects the writing of diagnostic messages following a return from the scratch routine on a SCRATCH VTOC request.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

WTORPAS1, WTORPAS2
prepares to request the console operator to enter passwords for both the PASWORD1 and PASWORD2 operands of the ADD, DELETEP, REPLACE, and LIST utility statements. The routines update and check the number of invalid passwords recorded per utility statement and per job step.

SVCPRO
loads register 1 with the address of the SVC 98 parameter list, issues SVC 98, and interprets the SVC return code.

SUBORDINATE LOAD MODULES

The subordinate load modules of the program are the auxiliary parameter analyzer, the message writer, and the volume lookup.

THE AUXILIARY PARAMETER ANALYZER (IEHPROG4)

The auxiliary parameter analyzer (IEHPROG4) analyzes any auxiliary parameters passed to the IEHPROGM program by a calling program and moves the DCBs for data sets SYSIN and SYSPRINT (or their substitutes) to the work area and opens them.

THE MESSAGE WRITER (IEHPROG3)

The message writer (IEHPROG3) writes all messages issued by the program. Unless a calling program specifies otherwise, messages are written on the SYSPRINT data set; console messages are written using the job management WTO routine.

Input to the message writer consists of a full word in register 0:

Byte 0 is unused.

Byte 1 Bits 0 to 5 are unused. Bit 6 is set to 1 if the message to be written is to be placed at the next available location in the output buffer; otherwise, it is set to 0. Bit 7 is set to 0 if the message is to be written during the current execution of the message writer, and to 1 otherwise.

Byte 2 contains the relative position in the buffer of the message.

Byte 3 contains the message number.

VOLUME LOOKUP (IEHPROGA)

This module obtains volume information for use by IEHVOLMT and data management. It is called only when information specified by the VOL keyword is encountered by IEHESCAN. At the time IEHPROGA is entered, register 2 points to a 28-byte field in the work area consisting of the following subfields:

- Bytes 0 to 7 contain the value supplied to the keyword operand VOL (left-aligned and padded with blanks), as scanned from a utility control statement.
- Bytes 8 to 11 are initially blank.
- Bytes 12 to 15 contain the return address (coded as a V-type address constant).
- Bytes 16 to 27 are initially blank.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Using the unit name of the device as input, IEHPROGA links to entry point IEFEB4UV in scheduler routine IEFAB4UV. This routine returns a function lookup value of the following format:

Byte 8 contains a bit configuration used by the I/O supervisor.

Byte 9 contains a device option code.

Byte 10 contains a device class code.

Byte 11 contains a device type code.

PROGRAM FLOW

The logical flow of the program proceeds in three phases:

- Phase 1 (Chart 01), during which a major routing table is established for the operation to be performed.
- Phase 2 (Chart 02), during which the major routing table is decoded, causing the parameter list to be built for the SVC to a data management routine, and causing appropriate volumes to be mounted. The parameter list (Figure 168 on page 477) is built at location IEHEMAC1 in the work area.
- Phase 3 (Chart 03), during which the initial SVC is issued, its return is analyzed, and any additional SVCs are issued in order to complete the request.

PHASE 1

The program receives control of the processor when the keyword operand PGM=IEHPROGM is encountered in an EXEC statement, or when the program is invoked by a calling program. The root segment immediately gives control to the parameter list builder segment. The registers are saved and the work area is obtained. Any auxiliary parameters are analyzed, and SYSIN and SYSPRINT are opened. A new request is initialized for, and then obtained (using BSAM).

FNDECODE scans the name field for syntax errors, and writes an error message if one is encountered. FNDECODE then directs control to IEHESCAN to scan the operation name (for example, SCRATCH) from the utility control statement image. IEHETLU then uses the name thus obtained to look up the address of the major routing table corresponding to the given operation. FNDECODE then places this address at location FINUSE in the work area, for use in Phase 2.

Communication between FNDECODE, IEHESCAN, and IEHETLU is effected through the use of the communication table, IEHECHAR (in the work area), shown in Figure 166 on page 475. IEHECHAR is also used in Phase 2 in scanning keyword operands.

PHASE 2

This phase of the program decodes the major routing table established during Phase 1. Decoding of the routing table results in the scanning of keyword operands, the volume lookup, the building of the parameter list for the data management routine to be used, and the mounting of volumes.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

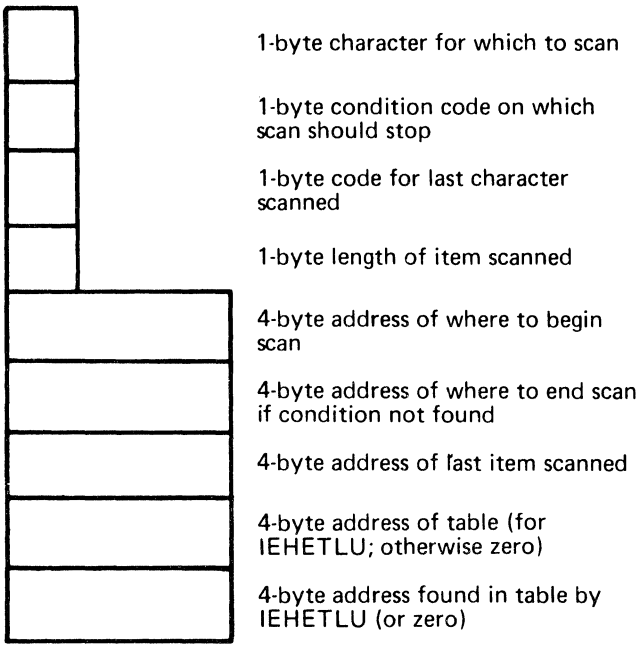


Figure 166. IEHECHAR, the Communication Table for FNDECODE, KODECODE, IEHESCAN, and IEHETLU

Major routing tables appear in CSECTs IEHPROGB and IEHPROG3 in the parameter list builder segment. There is one major routing table for each operation of the IEHPROGM program; the symbolic name of the routing table is the same as the name of the operation it supports. Each major routing table consists of a variable number of entries; the first byte of each entry contains a routine code, and the remaining bytes of the entry contain information useful to the routine. Figure 167 shows the format of the routing table for the catalog operation.

The Catalog Routing Table								Explanation of Table Entries
01*	00	70	00	24	00	00	00	DSNAME and VOL are required parameters.
02*	00	50	0C					
04*	x	x	x					xxx = address of error routine.
08*	y	y	y					yyy is unused.
0A*	00	00	01					The number of this major routing table is 1.

*Routine Code

Figure 167. The CATALOG Routing Table

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

As indicated in "Chart 02. Phase 2—Modifying System Control Data" on page 481, FDLD decodes the successive entries of the major routing table in use, directing control to the indicated routines.

Note: It is possible for the sequence of operations indicated by a routing table to be interrupted, possibly even broken. For example, if a syntax error on a control statement is encountered, the address of the major routing table will be replaced by the address of a routing table which will effect the writing of appropriate messages. FDLD would then decode this table, causing IEHPROG3 to write the selected messages. The last entry in tables of this nature will then either cause the address of the original major routing table to be restored, or will cause the request to be aborted, depending on the severity of the situation.

The scanning of keyword operands takes place when a routine code of hexadecimal 01 is encountered. KODECODE directs control to IEHESCAN to scan a keyword or keyword value. If VOL is present as a keyword, the value supplied to it in the control statement is passed to IEHPROGA, the volume lookup, and the device information retrieved is saved for the volume mounter and data management routines. When a value supplied to another keyword is detected, a lookup is performed by IEHETLU, using the keyword (for example, DSNAME), to retrieve the address of a keyword routine, which is then given control. (The symbolic address of a keyword routine is identical with the keyword name. For example, the symbolic address of the DSNAME routine is DSNAME.) The keyword routine then enters the keyword value information into the parameter list for the data management routine. Control is then given back to IEHESCAN to scan the next keyword parameter, and the cycle continues until an EOF condition is detected in reading SYSIN (or its substitute).

When the control statement has been scanned, control is given back to FDLD to decode the next entry in the major routing table. Successive entries in the major routing table may test for the presence of minimum allowable parameters (TESTDUP), establish a temporary routing table to write messages (LINKSAVE), restore the major routing table (DCRETURN), or read and log the remaining cards (READALL). If an error has been found in the utility control statement, the request is aborted and control is given to IEHPROG9 to honor the next request. If no error has been found, the last entry of the major routing table will have been decoded. The last entry of each major routing table causes FDLD to give control to GETAVOL, which places the number of the major routing table at location FINUSE in the work area, and then gives control to the volume mounter. If any required volume is not mounted or mountable, control is given to IEHPROG9; otherwise, control is given to the SVC return analyzer (IEHPROG2), and Phase 3 is entered.

At the completion of Phase 2, the parameter list for the initial SVC to a data management routine has been built at location IEHEMAC1 in the work area. The formats of the various parameter lists for SVCs 26, 29, and 30 are shown in Figure 168 on page 477. The formats of the various parameter lists for SVC 98 are shown in Figure 169 on page 478.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

SCRATCH (SVC 29)	RENAME (SVC 30)	CATALOG (SVC 26)
SVC bit configuration	SVC bit configuration	SVC bit configuration
↑ DSNAME	↑ DSNAME	↑DSNAME
unused	↑NEWNAME	↑CVOL
↑VOLLIST	↑VOLLIST	↑VOLLIST
UNCATALOG (SVC 26)	**BUILD INDEX (SVC 26)**	**DELETE INDEX (SVC 26)**
SVC bit configuration	SVC bit configuration	SVC bit configuration
↑DSNAME	↑INDEX	↑INDEX
↑CVOL	↑CVOL	↑CVOL
unused	unused	unused
BUILD ALIAS (SVC 26)	**DELETE ALIAS (SVC 26)**	**BUILD GENERATION (SVC 26)**
SVC bit configuration	SVC bit configuration	SVC bit configuration
↑INDEX	↑ALIAS	↑INDEX
↑CVOL	↑CVOL	↑CVOL
↑ALIAS	unused	unused
CONNECT (SVC 26)	**RELEASE (SVC 26)**	
SVC bit configuration	SVC bit configuration	
↑INDEX	↑INDEX	
↑CVOL	↑CVOL	
↑VOL	unused	

Notes:

- At time of SVC, register 1 contains the address of the parameter list.
- Each parameter list is constructed at IEHEMAC1 in DSECT, the work area.
- The addresses in the above parameter lists point to the following items:

DSNAME	44-byte data set name
NEWNAME	44-byte new name of data set
INDEX	44-byte index name
ALIAS	8-byte alias
VOL	4-byte device code followed by a 6-byte volume ID
CVOL	6-byte volume ID
VOLLIST	2-byte count followed by a variable number of fields of the following format: 4-byte table value from DEVNAMET 6-byte volume ID 2-byte sequence number (for taped volumes)

Figure 168. Parameter Lists Built by IEHPROGM for Data Management Routines

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

ADD (SVC 98)		REPLACE (SVC 98)	
01 (ADD)	00 00 00	02 (REPLACE)	00 00 00
data set name length	↑DS NAME	data set name length	↑DS NAME
00	00 00 00	00	↑current password
00	00 00 00	00	↑control password
number of volumes	↑volume list	number of volumes	↑volume list
protection code	↑new password	protection code	↑new password
string length	↑string	string length	↑string
DELETEP (SVC 98)		**LIST (SVC 98)**	
03 (DELETEP)	00 00 00	04 (LIST)	↑80-byte buffer
data set name length	↑DS NAME	data set name length	↑DS NAME
00	↑current password	00	↑current password
00	↑control password		
number of volumes	↑volume list		

Notes:

- At time of SVC, register 1 contains the address of the parameter list.
- Each parameter list is constructed at IEHEMAC1 in DSECT, the word area.
- The addresses in the above parameter lists point to the following items:

DSNAME	44-byte data set name.
control password	the control password of the entry to be added, deleted, or replaced.
volume list	list of six-byte volume IDs.
new password	the password of the entry to be added or to replace the current entry.
string	77-byte user security information.
current password	the password in the entry to be deleted, replaced, or listed.

The protection code indicates the type of protection to be assigned in the PASSWORD data set and in the DSCB, as follows:

X'00'	Indicates default protection. For add, the default is the protection type of the control entry. If this if the first, therefore control, entry, the defult is type X'03'.
X'01'	Indicates that this password is to allow both read and write access to the data set. If a control password is being changed, read/write protection is set in the DSCB.
X'02'	Indicates that this password is to allow only read access to the data set. If a control password is being changed, read/write protection is set in the DSCB.
X'03'	Indicates that this password is to allow both read and write access to the data set. If a control password is being changed, read-without-write protection is set in the DSCB.

Figure 169. Parameter Lists Built by IEHPROGM for Data Management Routines (for SVC 98)

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

PHASE 3

Phase 3 issues, and analyzes the returns of, all SVCs to data management routines used to accomplish IEHPROGM functions. At the time Phase 3 is entered, the parameter list for the initial SVC has been built, and all volumes are mounted or mountable.

Using the routing table number placed at FINUSE (in the work area) during Phase 2 by routine GETAVOL, routine LATAB replaces the routing table number with the address of a carryover routing table, to be decoded by FDLD. If the current request is a SCRATCH VTOC request, the carryover routing table will cause itself to be replaced with the routing table VTOCDCS. Otherwise, the carryover routing table will merely cause the SVC to be issued, by means of the entry

X'00' AL3 (SVC instruction)

FDLD then decodes the carryover routing table, establishing the VTOCDCS routing table only if the request is to scratch a VTOC.

SVCRET and SVC26RET decode the return from the data management routine used, directing control through the branch table, BRANTAB, to diagnostic routines or to message-effecting routines.

Control is directed in the following way:

1. The return code (in register 15) is used by routine SVCRET or SVC26RET (depending on the SVC) as an indexing factor to retrieve a code number from the current entry of the active routing table. (The current entry is the entry following that entry designating the SVC.)

2. The code number thus retrieved is used as an indexing factor to give control to the appropriate IEHPROGM diagnostic routine to treat the type of return. Control is given to a diagnostic routine by means of a branch table, BRANTAB.

EXAMPLE: Figure 170 shows the return-indexing entry of the CATALOG routing table. This entry is at location CATALOG + 4 in IEHPROG2, immediately following the SVC entry address. Following the return of control to the program from the catalog routine (SVC 26), routine SVC26RET would use the contents of register 15 to retrieve a code number from this return-indexing entry. If the return code from the catalog routine were 28, for example, a code number of 8 would be retrieved. Routine SVC26RET would then give control (via the branch table BRANTAB) to the diagnostic routine indicated by a code number of 8.

CATALOG + 4

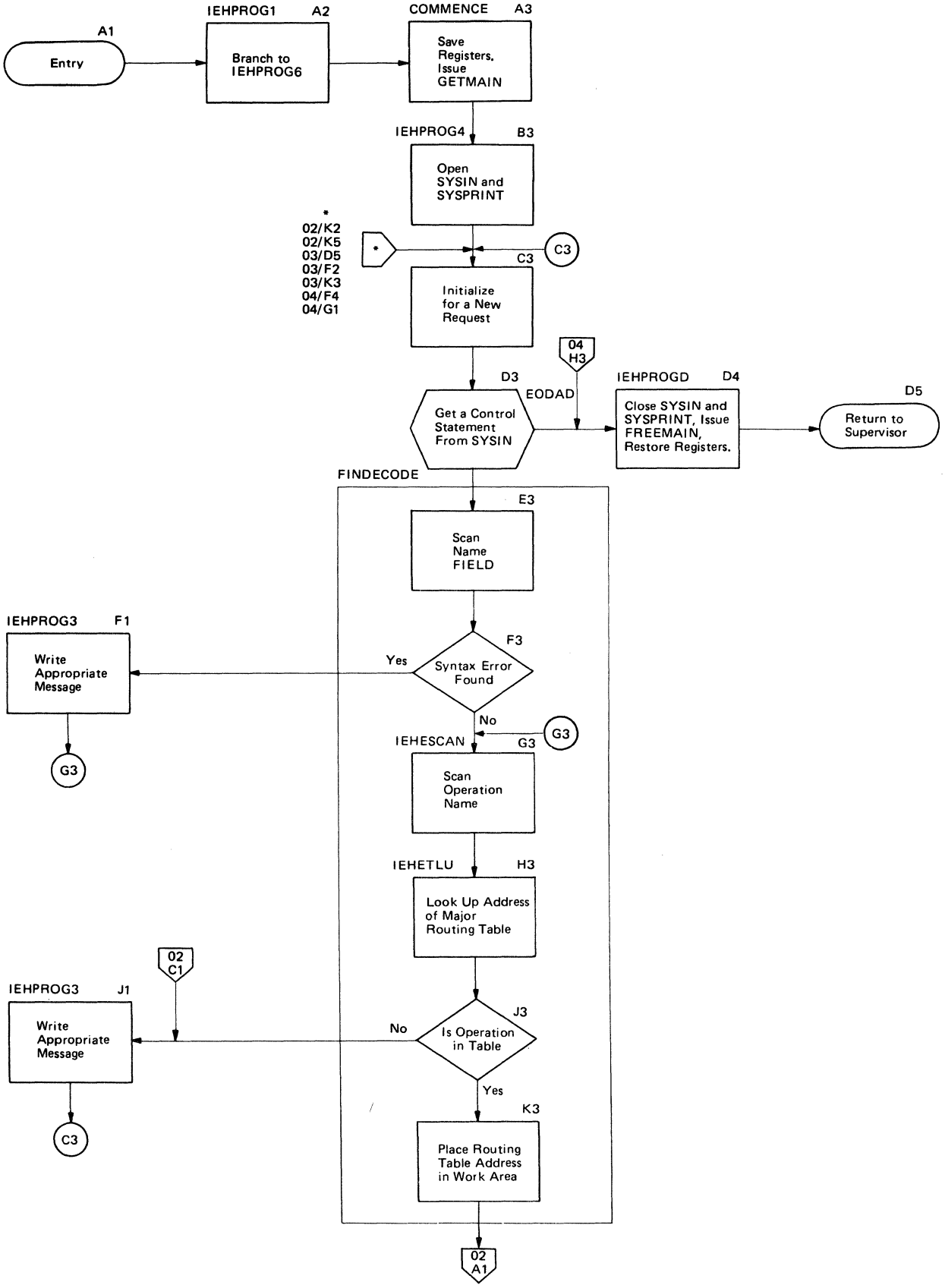
01	02	03	00	05	06	07	08	00	00	00	00
0	4	8	12	16	20	24	28	32	36	40	44

Return code from Catalog (Reg. 15)
Indexing factor for BRANTAB

Figure 170. The Return-Indexing Entry (for the Catalog SVC) of the Catalog Operation

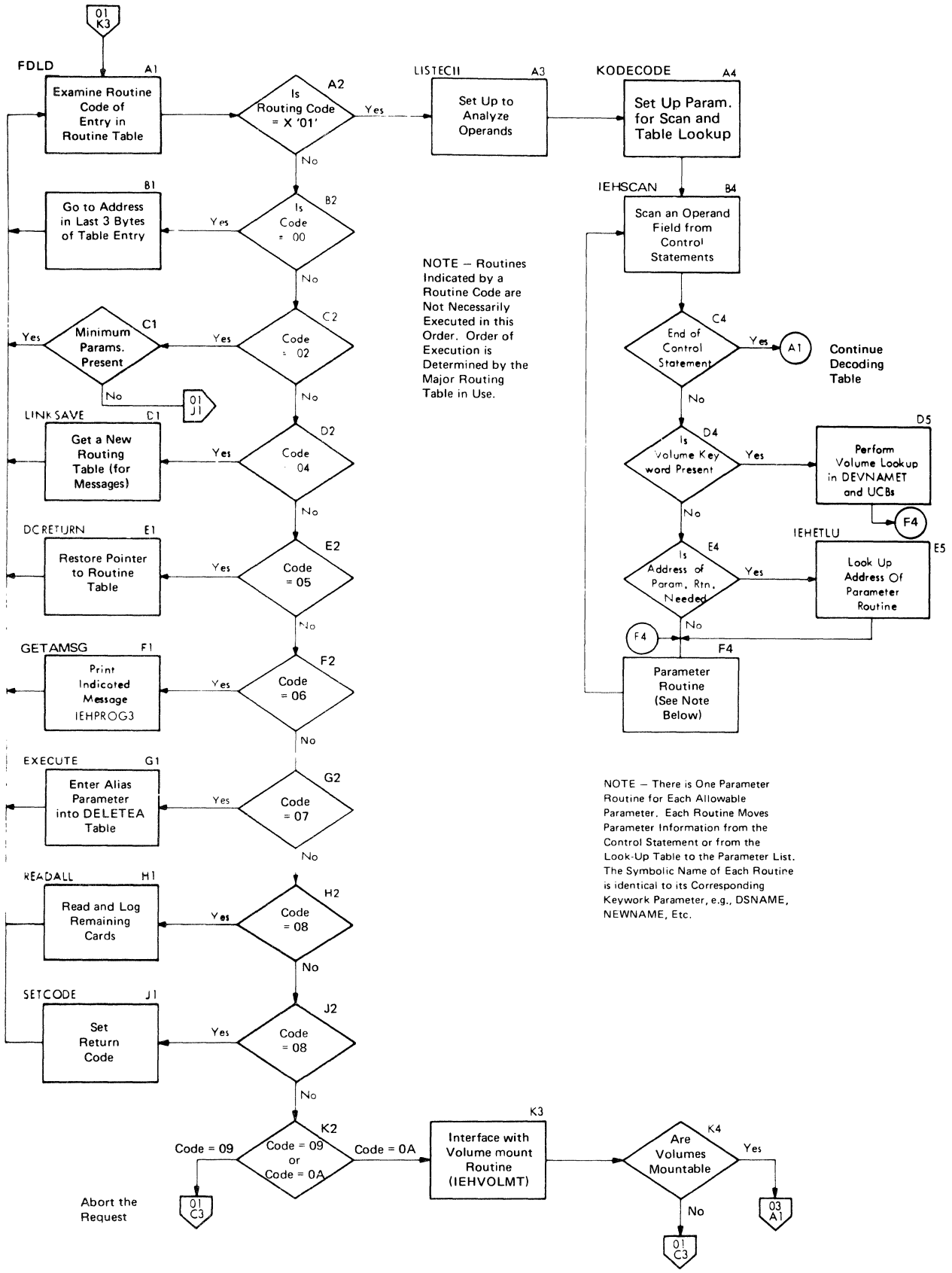
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 01. PHASE 1—MODIFYING SYSTEM CONTROL DATA



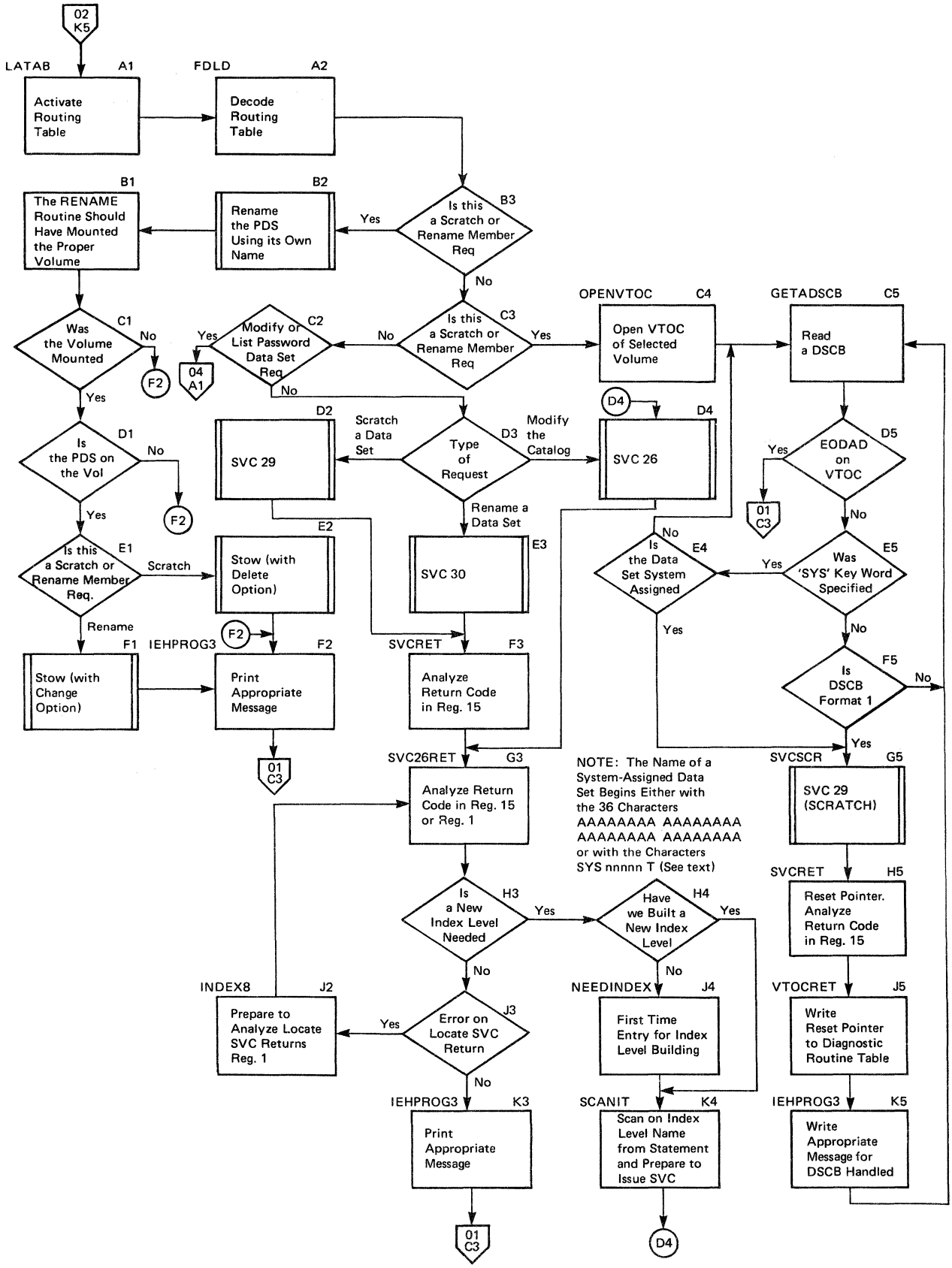
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 02. PHASE 2—MODIFYING SYSTEM CONTROL DATA



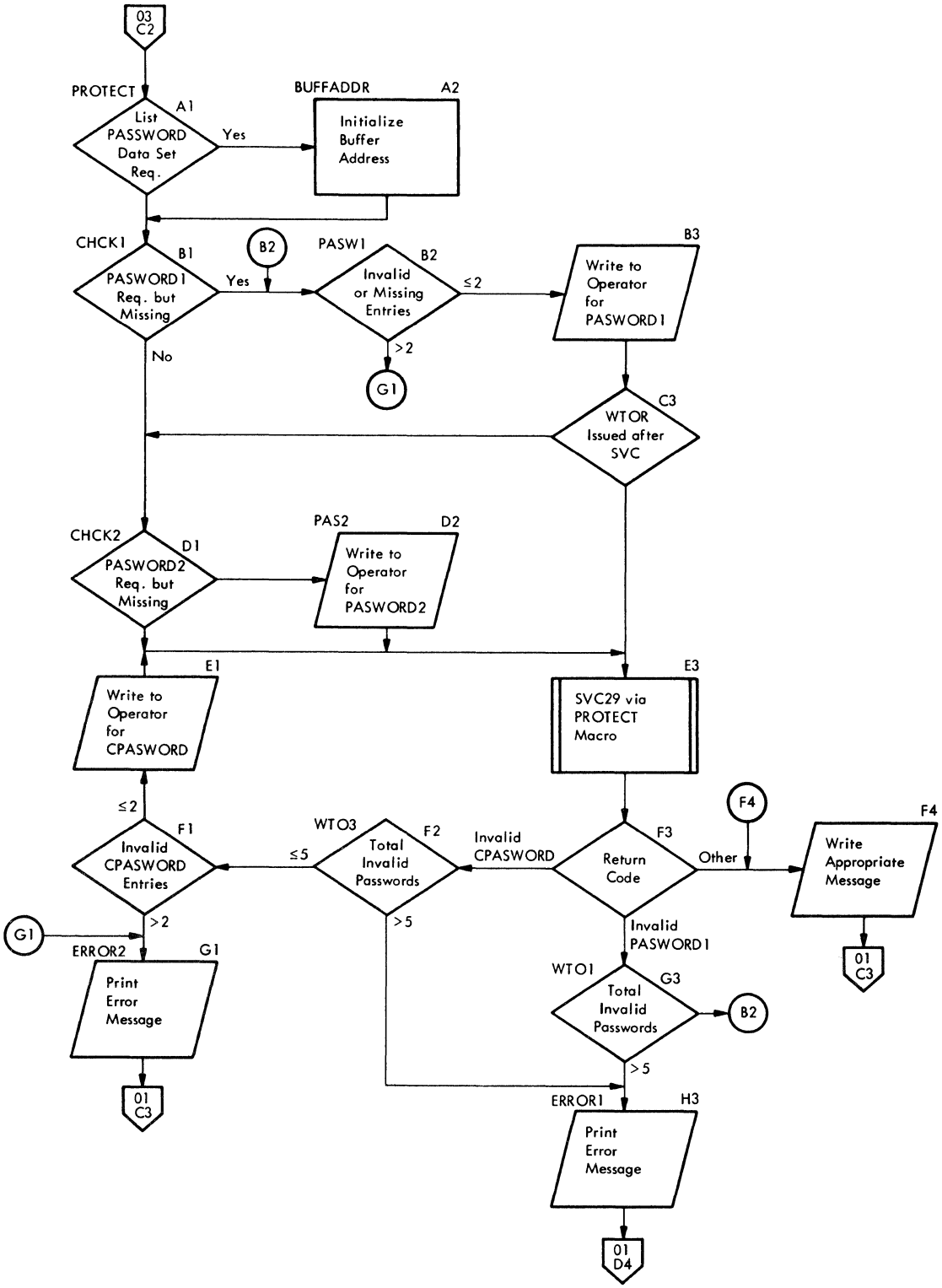
This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 03. PHASE 3—MODIFYING SYSTEM CONTROL DATA (PART 1 OF 2)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 04. PHASE 3—MODIFYING SYSTEM CONTROL DATA (PART 2 OF 2)



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IFHSTATR PROGRAM

The IFHSTATR program enables a user to obtain a volume-statistics report about errors occurring on magnetic tape. The program reads the IFASMFDP tape, selects only the type 21 (ESV) records, extracts and formats information from these records, and writes the report. Figure 171 on page 485 shows the structure of type 21 records.

PROGRAM FLOW

"Chart 01. Program Flow" on page 486 shows the flow of the IFHSTATR program. The program consists of one load module (IFHSTATR), which first initializes the program by saving the contents of registers, opening both the input (SYSUT1) and output (SYSUT2) data sets, and using the TIME macro instruction to get the date from the supervisor.

After initialization is completed, the program reads one record from the input data set and checks the RECORD TYPE field (see Figure 171) for X'15' (type 21). If the record is not a type 21 record, the program gets the next input record.

If the record type is 21, the program prints a page heading if one is required, converts the record to a printable form, and writes it. An indicator (DATAFLAG) is set to show that a type 21 record was found.

After reading the end-of-file record, the program checks the DATAFLAG indicator to see if any type 21 records have been found. It then closes the input and output data sets and uses the RETURN macro instruction to return control to the supervisor.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

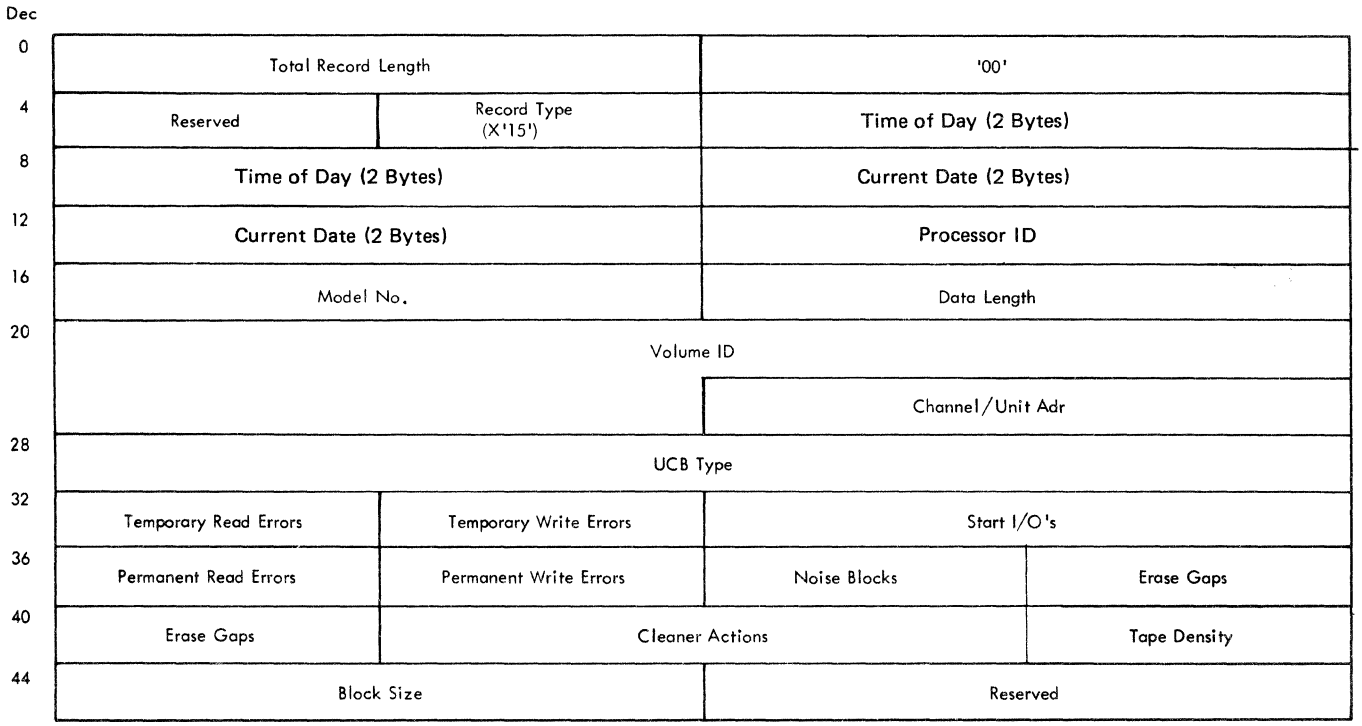
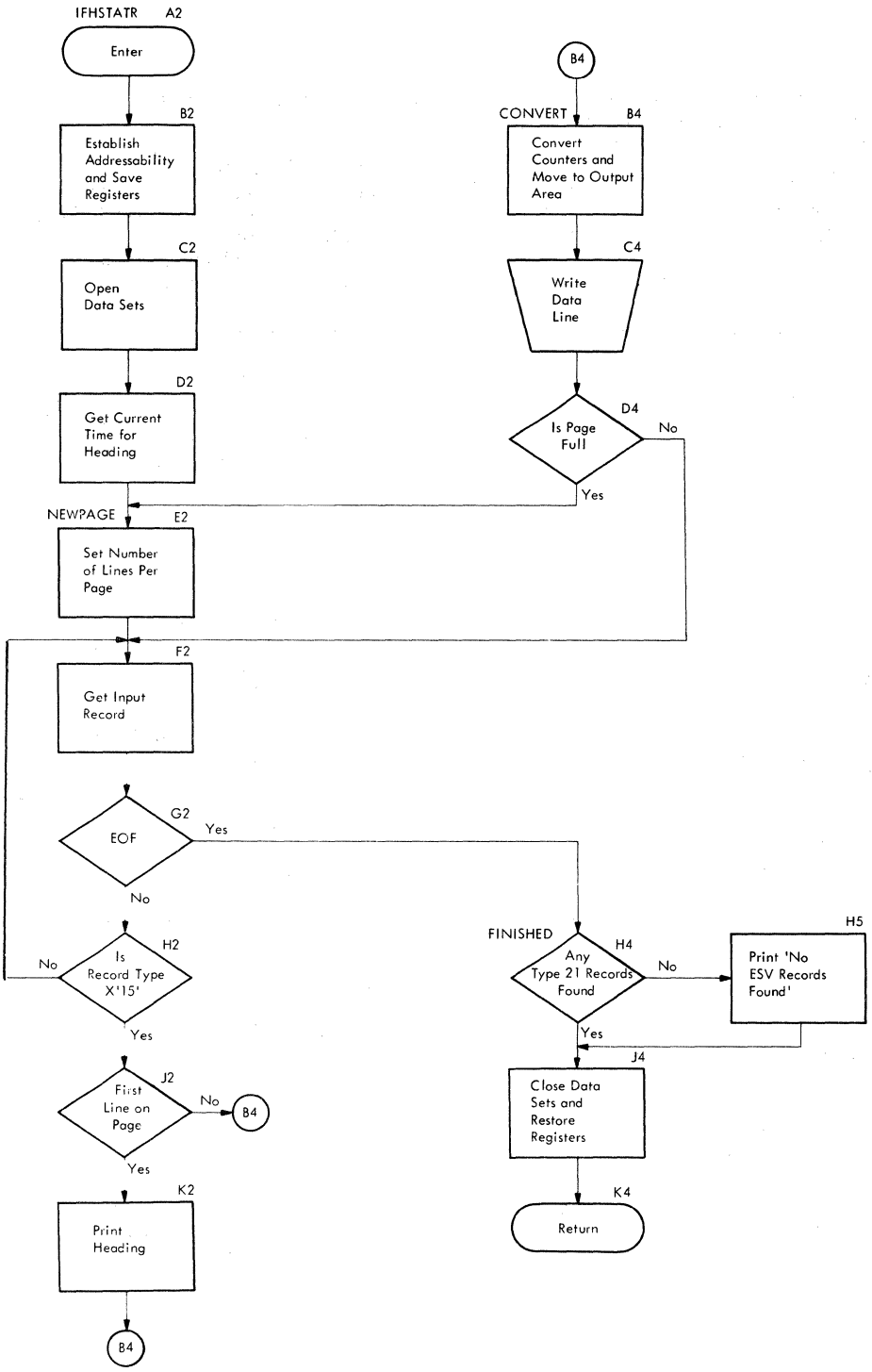


Figure 171. ESV Record Format

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

CHART 01. PROGRAM FLOW



This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

OFFLINE IBM 3800 UTILITY

A new way of printing offline is available when the IBM Tape-to-Printing Subsystem Feature is installed on an IBM 3800 Printing Subsystem. The 3800 Printing Subsystem is a nonimpact, high-speed, general purpose system printer that has a variety of functions and features that provide a wide range of printing applications. With the addition of the IBM 3800 Tape-to-Printing Subsystem Feature and using the Offline IBM 3800 Utility program product, the 3800 with all its functions and features can now be used offline in a stand-alone tape-to-printer operation.

The IBM Tape-to-Printing Subsystem Feature provides the 3800 with the capability of being used as an offline or stand-alone printer. It requires the addition of either an IBM 3410/3411 Model 3 Magnetic Tape or an IBM 3803/3420 Magnetic Tape System or its equivalent. The 3800 with this feature can be controlled from the operator control panel, by defaults you have established, or from control information provided on a tape called a control file, or by any combination of the three. The data to be printed must be on tape. It can be on a separate tape, or on the same tape that contains the control file, provided that it follows the control file.

The Offline IBM 3800 Utility program product is provided to be used with the IBM Tape-to-Printing Subsystem Feature of the IBM 3800 Printing Subsystem. This program product provides the offline IBM 3800 user with the same functional capability that is available to the online IBM 3800 user. The utility provides:

- Utility control statements to achieve the full range of function available to the online IBM 3800 user
- Functions not available through operator control or default selection—graphic character modification, user-designed characters, national use graphics, copy modification, and user-defined character arrangement tables.
- Increased throughput and decreased risk of operator error by reducing operator intervention.
- Flexibility to place control files on the same tape as the print data or on a separate tape volume.
- Operator communication via printed messages on the offline IBM 3800.
- Ability to print the contents of tape control files, either formatted or unformatted, on an online system printer.

The utility, which is executed in an online MVS/370 Data Facility Product environment, can be used to create control information and place it on magnetic tape that can be used to set up the IBM 3800 for the printing of user data.

For further information, see the following publications:

- Offline IBM 3800 Utility
- Offline IBM 3800 Utility Logic

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

APPENDIX A. MODULES OF UTILITY PROGRAMS

This appendix describes the modules of each utility program. The module names given are the SYS1.AOSU0 names. When these names differ from equivalent SYS1.LINKLIB names, the latter are given in parentheses.

ICAPRTBL

The independent utility ICAPRTBL is part of the SYS1.SAMPLIB data set.

ICAPRTBL
loads the UCS buffer and the FCB buffer of the IBM 3203 Model 5 or 3211 Printer.

IEBCOMPR

IEBCROOT
is the root segment; it opens and closes SYSPRINT, writes messages, and calls the proper modules.

IEBCOMPM
is the message module.

IEBCANAL
interprets returns from IEBCCS02.

IEBCMAIN
when the data sets are partitioned, compares directories to determine whether one is a subset of the other; when the data sets are sequential, it compares the data sets.

IEBCCS02
opens/closes SYSIN data set, scans cards, returns data to IEBCANAL. (This module is part of the IEBGENER utility.)

IEBCQSAM
compares spanned records on logical record level.

IEBCULET
processes user labels.

IEBCOPY

IEBDSCPY
is the controlling module for the program; it gives control to the proper modules and contains subroutines to allocate storage, scan directories, construct control tables, and perform address conversion.

IEBDV1
opens and verifies the characteristics of data sets.

IEBSCN
scans control cards, builds the input data set name table, and builds the table of selected or excluded members.

IEBVCT
sorts the table of selected or excluded members and builds the table of pointers to new member names.

IEBDRD
reads directory entries or member data at appropriate points in the program.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEBVTT
sorts the control table for the copy step.

IEBMCM
prepares a data set to be compressed-in-place, if necessary.

IEBDRB
blocks or deblocks records as required during movement between buffers.

IEBVDM
builds the merged output directory to incorporate input directory information into the existing output directory.

IEBWSU
determines the output data set address to which a record will be written from a buffer.

IEBDWR
writes directory entries or member data at appropriate points in the program.

IEBVTM
performs termination procedures for the program.

IEBVMS
prints all program messages and control statements.

IEBIOE
provides I/O-error analysis for the program.

IEBBAM
allocates the initial I/O buffer area and moves the old-name/new-name member pairs from a table of selected members to the I/O buffer.

IEBCNVT
handles all input TTR and MBBCCHHR conversion requests in a load operation.

IEBDSU
handles directory entries in load and unload operations.

IEBLDUL
initiates load and unload operations.

IEBRSAM
reads data from an unloaded data set into buffers.

IEBWSAM
writes data from buffers to an unloaded data set.

IGG019C8
PCI appendage.

IGG019FT
page-fix and SIO appendage.

IEBDG

IEBDG
is the control module that is the interface with a calling program; it opens the input, output, and message data sets, and reads the program's control cards.

IEBFDANL
analyzes the keywords and parameters on an FD card and begins construction of an entry in the FD table.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEBFDTBL
completes the construction of the FD entry that was begun by the FD analysis module; it assigns FD card default values if necessary.

IEBCRANL
analyzes the keywords and parameters on a CREATE card and builds a create table entry, a picture table, an FD address table, and an exit name table.

IEBCREAT
generates output records by using information from (1) input data sets and (2) tables built by previous modules, as required; it permits user modifications before final record output and releases storage obtained for information tables.

IEBDGMSG
is the message module; it controls the paging on a message printer.

IEBDGCUP
is the cleanup module that closes DCBs and frees storage for DCBs and buffer pools.

IEBEDIT

IEBEDIT
extracts records from a master file of JCL statements to create an edited input stream data set.

IEBGENER

IEBGENRT
is the root segment routine; it initializes switches; processes/stores internal parameters; stores overlay parameters; writes the header, control cards, and error messages for analysis and processing segments; governs control of flow between segments; and contains the following subroutines:

- Initialization
- Internal parameter analyzer
- Message writer
- Control card writer

IEBGMESG
contains all error messages and the header for SYSPRINT.

IEBGSCAN
executes, in conjunction with IEBCCS02, the scan routine; it analyzes parameters passed by the scan routine, diagnoses invalid parameters, sets switches and creates tables to be used by IEBGEN03, requires a RECORD statement indicating that label information follows in SYSIN, checks the legality of each statement, and gets a save area for as many labels as indicated in the RECORD card.

INPUT: One card field at a time, supplied by IEBCCS02; a parameter list, pointed to by register 1, contains the parameter to be analyzed and the address of a work area containing switches set by IEBCCS02.

OUTPUT: Tables and switches, the addresses of which are in a list pointed to by register 1 at exit time; these tables and switches are used by processing routines and are provided in a usable format (to represent the control card parameters).

IEBCCS02
executes to obtain information from the control data set; it opens SYSIN, reads control cards, and passes the cards

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

and the location, length, and identification of card fields to its caller.

IEBGENR3
activates the DCB exit lists if user exits or label processing is specified; it opens the SYSUT1 and SYSUT2 DCBs for BSAM, takes label (input/output, header/trailer) exits if user exits and/or label processing is required, compares RECFMs, LRECLs, and BLKSIZEs of both DCBs (depending on the parameter comparisons and on the requirements for editing and/or conversion, DCBs may be closed and reopened for QSAM), and then passes control to either IEBGEN03 or IEBGENS3.

INPUT: Register 1 pointing to a table containing the addresses of tables built by IEBGSCAN.

OUTPUT: Register 1 containing the same contents as at input time; control is given to the appropriate processing routine.

IEBGEN03
(processing for nonspanned data sets) gets control after IEBGENR3 has opened the input and output data sets; it controls linkage to the edit and conversion routines, uses BSAM and double buffering, and gets storage for label processing.

INPUT: Register 1 pointing to a list with addresses of tables built by IEBGSCAN according to control card parameters.

IEBGENS3
processing for spanned data sets [RECFM =VS or VBS] gets control after IEBGENR3 has opened the input and output data sets; it uses BSAM if there is no change in block size, LRECL, or RECFM (otherwise, it uses QSAM), and controls linkage to the edit and conversion routines, to do processing dictated by control cards.

INPUT: Register 1 pointing to a list with addresses of tables built by IEBGSCAN according to control card parameters.

IEBEDIT2
moves logical records from the input buffer to the output buffer; it initiates the requested conversion of each field. Parameters for moving and conversion come from end data control blocks (EDCBs) for the current record group.

IEBLENP2
executes each time a new group of input data set records for which the user supplies new editing parameters is encountered; the total length of the output records is computed on the basis of the length of the fields to be moved to output, and the conversion to be done to each field.

IEBMOVE2
moves any number of bytes from one area to another in main storage; it moves data from one buffer to another (input to input, output to output, input to output), 256 bytes at a time.

IEBCONH2
converts data from H-set BCD to EBCDIC; it converts any length field and translates 256 bytes at a time.

IEBCONP2
converts data from packed to zoned decimal format, proceeding from right to left.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEBCONZ2
converts data from zoned to packed decimal format, proceeding from left to right.

IEBIMAGE

IEBIMAGE
is the main procedure that contains the utility initialization and nested procedures necessary to process the FCB, COPYMOD, GRAPHIC, CHARSET, and TABLE operations.

IEBISCAN
contains the procedure for handling the reading and scanning of all control statements. It determines control statement types by referencing the operation table (OPTAB) and keyword table (KEYTAB), respectively.

IEBIMSGS
contains all the messages printed by the utility and the code by which the messages are selected.

IEBIMCHA
creates, checks, and prints library character set modules.

IEBIMCOP
creates, checks, and prints copy modification modules.

IEBIMFCB
creates, checks, and prints forms control buffer modules.

IEBIMGRA
creates, checks, and prints graphic character modification modules.

IEBIMTAB
creates, checks, and prints character arrangement table modules.

IEBIMWRT
writes new modules to the library.

IEBISAM

IEBISAM
is the root segment; it sets up a common work area, obtains input parameters, sets switches, and passes control to the required module.

IEBISC
copies records of an indexed sequential data set.

IEBISU
retrieves logical records sequentially from an indexed-sequential data set.

IEBISSO (IEBISU)
creates 80-byte logical records with fields as defined for "unloaded" data sets.

IEBISL
reconstructs indexed-sequential data set from "unloaded" data.

IEBISSI (IEBISL)
retrieves logical records from an "unloaded" data set.

IEBISPL
prints logical records of an indexed sequential data set.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEBISF
writes messages, prints error messages if applicable, and returns a completion code to the root segment.

IEBISMES (IEBISF)
message module.

IEBPTPCH

IEBPPUN1
is the root segment; it opens and closes the SYSPRINT data set, calls the proper modules, and prints all messages and control cards.

IEBPPMSG
is the message module.

IEBPPAL1
obtains storage for and then constructs tables and work areas, calls and then interprets returns from IEBCCS02, and checks for valid parameters.

IEBCCS02
opens and closes the SYSIN data set, reads and scans cards, and returns data to IEBPPAL1. (This module is part of the IEBGENER utility.)

IEBPPCH1
is the processor module; it handles sequential and partitioned data sets, opens and closes SYSUT1 and SYSUT2 data sets, checks for valid control cards, and examines tables built by IEBPPAL1.

IEBUPDTE

IEBUPDT2
creates partitioned or sequential data sets; sequences new data sets, resequences old data sets, replaces or reproduces data set members, or adds members to a partitioned data set.

IEBUPLOG
opens SYSPRINT and writes messages.

IEBUPDTE
reads control cards and opens SYSUT1 and SYSUT2.

IEBASCAN
scans and analyzes control statements and sets appropriate flags.

IEBBSCAN
syntax checking, gives controls back to IEBASCAN.

IEBUPNIT
initializes the IEBUPCON region and opens the SYSIN data set.

IEBUPXIT
contains exit routines for the program.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEHATLAS

IEHATLAS
attempts to write a record to replace an error record on a defective track; if not successful, it issues a macro instruction for an alternate track assignment routine (ATLAS).

IEHMESS
contains most of the IEHATLAS messages. There are no executable instructions in this CSECT.

IGC0008F
gets the address of the alternate track.

IGG0860A
reads the Format-4 DSCB and builds the channel program for getting the description of the error track.

IGG0860B
executes the channel program to get the description of the error track.

IGG0860C
obtains storage for the largest record on the error track; it partially chains an alternate track to the primary track.

IGG0860D
transfers all records from the error track to the alternate track.

IGG086AE
completes chaining the alternate track to the primary track; it executes the caller's channel program (if possible) to rewrite the error record onto the alternate track, updates the VTOC (Format 4) DSCB, and releases storage areas.

IEHINITT

IEHINITT
is the root segment; it opens and closes SYSIN and SYSOUT, builds a tape label image in main storage, extracts information from the JFCB, and links to SVC 39 to write the tape label.

IGC0003I (SVC 39)
writes a tape volume label followed by a dummy header label and a tapemark.

IEHSCAN
reads control statements and scans them for an INITT command and for keywords.

IEHPRNT
is the message module.

IEHLIST

IEHLIST1 (IEHLIST)
scans and reads requested data from VTOCs, catalogs, and directories.

IEHLIST2 (IEHLIST)
is the message module.

IEHLIST3 (IEHLIST)
reads control cards.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEHMOVE
is the root segment; it obtains a save area.

IEHMVXSE
gets three work files and a work area.

IEHMVXSF
is the first-time control module for IEHMVSSF.

IEHMVSSF (IEHMVESF)
mounts volumes.

IEHMVEST
clears work areas and initializes for a request.

IEHMVESJ
reads cards.

IEHMVSSS (IEHMVESS)
builds tables and sets switches.

IEHMVESI
opens a CVOL for a data set group operation when a CVOL is specified.

IEHMVESC
reads the CVOL and writes it onto SYSUT1 for a CVOL data set group operation, or writes the CVOL entries on SYSUT2 for a move or copy catalog operation.

IEHMVESH
sets up single data set requests for DSGROUP operations. Also, for DSGROUP when CVOL is not specified, this module performs a VSAM generic locate to determine the members of the group.

IEHMVSSZ (IEHMVESZ)
checks for a volume or a data set.

IEHMVSSV (IEHMVESZ)
obtains the "FROM" DSCB; it links to the module for mounting of the "FROM" volume.

IEHMVMRZ (IEHMVESZ)
writes messages.

IEHMVSRZ (IEHMVESX)
handles routing and errors.

IEHMVSRV (IEHMVESX)
validates data sets.

IEHMVSRK (IEHMVESX)
reads unloaded records.

IEHMVMRZ (IEHMVESX)
writes messages.

IEHMVSRY (IEHMVESV)
handles routing and errors.

IEHMVSSX (IEHMVESV)
allocates "TO" data sets.

IEHMVSTC (IEHMVESV)
reads the "FROM" partitioned data set directory.

IEHMVMRY (IEHMVESV)
writes messages.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEHMVSSY (IEHMVESY)
handles routing and errors.

IEHMVSRM (IEHMVESY)
writes the first unloaded record when applicable.

IEHMVSRX (IEHMVESY)
builds the "TO" and "FROM" DCBs; it handles the "TO" DD and "FROM" DD.

IEHMVMSY (IEHMVESY)
writes messages.

IEHMVSTL (IEHMVETL)
performs the functions of MOVE/COPY/LOAD/UNLOAD for BDAM data sets.

IEHMVSRK (IEHMVETL)
reads unloaded data sets.

IEHMVSRM (IEHMVETL)
writes unloaded data sets.

IEHMVMTL (IEHMVETL)
writes messages.

IEHMVETJ
reads the "FROM" and writes the "TO" sequential or partitioned data sets without performing reblocking.

IEHMVESL
reads the "FROM" and writes the "TO" sequential or partitioned data sets; it reblocks type F records.

IEHMVESM
reads the"FROM" and writes the "TO" sequential or partitioned data sets; it reblocks type V records.

IEHMVSRD (IEHMVERD)
builds unloaded records.

IEHMVSRM (IEHMVERD)
writes unloaded records.

IEHMVSRA (IEHMVERA)
re-creates an unloaded record in its original state.

IEHMVSRK (IEHMVERA)
reads unloaded records.

IEHMVSRM (IEHMVETA)
writes unloaded records.

IEHMVSRK (IEHMVETA)
reads unloaded records.

IEHMVMTA (IEHMVETA)
writes messages.

IEHMVSTA (IEHMVETA)
puts SYSCTLG data on a specified device.

IEHMVESR
gets directory entries from the SYSUT3 work file.

IEHMVETG
gets directory entries from SYSUT1 of includes or selects.

IEHMVESA
contains and selects messages for IEHMVXSE, IEHMVEST, and IEHMVSSS.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

IEHMVESU
writes messages.

IEHMVESN
closes the "TO" and "FROM" data sets; it determines the next module.

IEHMVMSN (IEHMVESN)
writes messages.

IEHMVMSQ (IEHMVESQ)
writes messages.

IEHMVESP
catalogs and uncatalogs copied data sets.

IEHMVESO
checks errors—job abort or request.

IEHMVESK
closes SYSIN; it scratches and closes SYSUT1, SYSUT2, and SYSUT3.

IEHPROGM

IEHPROG1 (IEHPROGM)
gets the work area, reads SYSIN, and mounts volumes if applicable.

IEHPROG2 (IEHPROGM)
issues SVCs for cataloging, uncataloging, deleting, connecting, releasing, BLDA, and DELET.

IEHPROG3 (IEHPROGM)
contains and writes messages.

IEHPROG4 (IEHPROGM)
opens input and output DCBs.

IEHPROG5 (IEHPROGM)
prepares for the volume mounting module IEHMVSSF.

IFHSTATR

IFHSTATR
selects, formats, and writes information about tape errors from the IFASMFDP tape.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

APPENDIX B. USER LABEL PROCESSING AND TOTALING

With respect to the processing of user labels and the totaling functions performed by user routines, Figure 172 on page 499 shows the general logic of the following utility programs: IEBCOMPR, IEBGENER, IEBPTPCH, and IEBUPDTE.

The following discusses parameter information passed from a utility program to a user routine, and return code information passed from a user routine to a utility program.

PARAMETER LIST

When the utility gives control to a user's label processing or totaling routine, general register 1 contains the address of a parameter list whose format is given in Figure 173 on page 500.

A description of the underscored fields indicated by the parameter list in Figure 173 is given below.

- Label Buffer Area: Prior to entering a label routine, user header or trailer labels are read into this area by the operating system. When a user's label routine constructs labels, the labels are placed (one at a time) in this area.

- Status Information Address: If an uncorrectable I/O error occurs during the reading or writing of a user label, the utility routine gives control to a user label routine with bit 0 of the high-order (error flags) byte of this field set to 1. If the totaling facility has been specified and an uncorrectable I/O error occurs as the utility is placing data on an output data set, a user's totaling routine is given control with bit 0 of the high-order (error flags) byte set to 1. The 3 low-order bytes of this field contain the address of the standard status information for SYNAD routines (see Data Management Services for more information).

- Totaling (Image) Area: For a totaling routine, a user may require an area to contain record counters, totals, pointers, etc. A utility supporting the totaling facility obtains this area and places its address in a DCB exit list (for use by data management routines such as the open routine and the end-of-volume routine) and in this field of the parameter list (for use by the totaling routine). The information that the user saves in this area is synchronized with the physical records written prior to placing a given record on an output data set. In an end-of-volume situation, the updated totaling information is kept as an "image" whose address (in the parameter list) is given to a user label routine. This ensures an accurate transfer of totaling information from one volume to another. The image area has meaning only for output data sets and when the user and the utility have called for the totaling facility via an entry in the DCB exit list.

Note: At volume switch time, the utility routines use the information contained in the flag byte of the second word to indicate end-of-volume and end-of-data.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

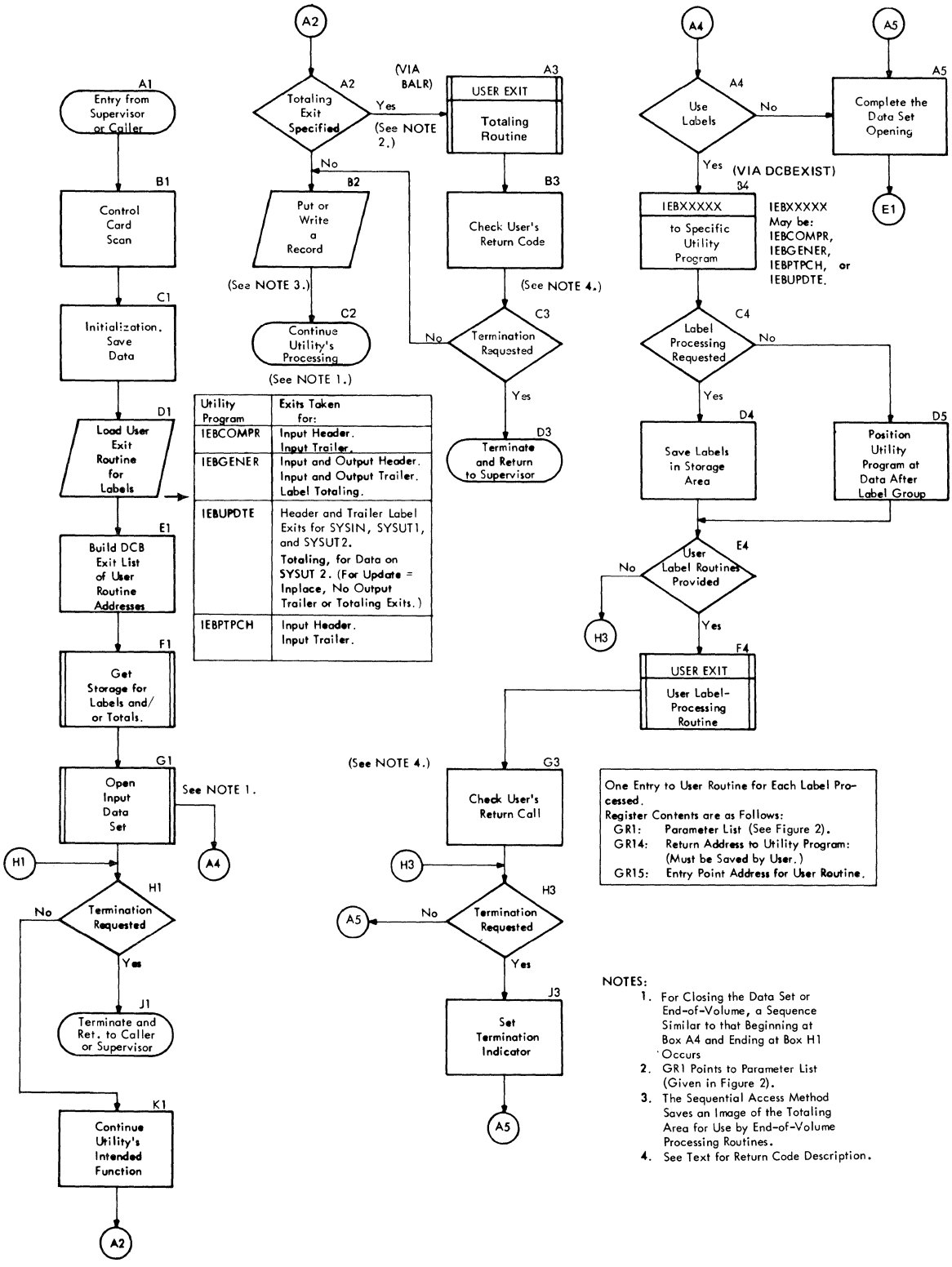


Figure 172. General Logic of Utility Program with User Label Processing Routine Exits

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

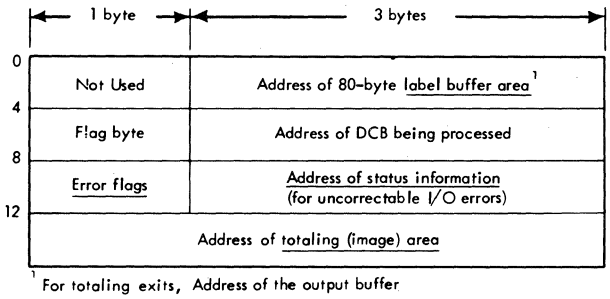


Figure 173. Parameter List Passed to User-Label Exit Routine

PARAMETER LIST MODIFICATIONS

For IEBUPDTE, the following modifications are made to the parameter list:

- When there are user label-processing routines, the first meaningful field of the parameter list passed to the user output-label routine points to the label buffer. This buffer, which contains a label data record from the SYSIN data set, is for the user to inspect before the record is written as a label.
- If the error status information in the parameter list is established as a result of a reading error, the user routine must return one of the return codes (described in the next section), or the program will be terminated.
- If the error status information is established as a result of a recording error, bit 1 (of the error-flags byte) is set to 1 to indicate that the error occurred during an output operation. In this case, the user routine must return a code of either 0 or 4, or the program will be terminated.
- For header labels only, a fifth entry in the parameter list occurs under the conditions given below. The first byte of this entry is meaningless; the last 3 bytes contain the address of the label that has been replaced from the old master data set (SYSUT1). The conditions (all of which must be present) for the occurrence of the entry are:
  - An update of the old master is specified via the keyword UPDATE=INPLACE.
  - A LABEL statement must be specified for header labels in the input data set.
  - A user label routine corresponding to the LABEL statement is specified and user labels are encountered on SYSUT1.

RETURN CODES

One of the following return codes must be placed in general register 15 when a user (label processing) exit routine gives control back to the utility program.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

An incorrect (or no) code results in termination of the program.

Type of Routine	Code	System (Utility) Response
Input header or input trailer label	0	Resume normal processing. Ignore additional labels in the label group.
	4	Read next user label into buffer area. Return control to user exit routine. Resume normal processing if no more labels.
	16	Request termination of label processing. Utility program performs cleanup functions and terminates.
Output header or output trailer label	0	Resume normal processing. No label is written from buffer area.
	4	Write label from buffer area. Resume normal processing.
	8	Write label from buffer area. If fewer than eight labels created, return to exit routine. Otherwise, resume normal processing.
	16	Request termination of label processing. Utility program performs cleanup functions and terminates.

RETURN CODE MODIFICATIONS

1. For IEBUPDTE, the following modifications are made to the return codes when the keyword UPDATE=INPLACE is specified.

Type of Routine	Code	System (Utility) Response
Input header	0	Same as above.
	4	Same as above.
	8	Write label from buffer area. Resume normal processing.
	12	Write label from buffer area. Read next label into buffer area. Return control to user exit routine. Resume normal processing if no more labels.
	16	Request termination of label processing. Utility program performs cleanup functions and terminates.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

2. For IEBCOMPR, the following modifications are made to the return codes, depending on the operand in the LABELS statement (see Figure 174 on page 504):

Type of Routine	User Return Code	LABELS Statement	System (Utility) Response
Input header or trailer labels	16	DATA ALL	Return a code of 4 to open routine. Take no additional label exits.
	16	DATA ALL	Return a code of 0 to open routine. Ignore rest of labels.
	0[1]	DATA ALL	Same as for code 16.
	0[1]	DATA ALL	Same as for code 16.
	0, with a previous code of 16	DATA ALL	Compare the labels, then terminate the processing.
		DATA ALL	Terminate the processing.
	0, with no previous code of 16		Compare available labels. Then check LABELS statement operand as follows:
		DATA ONLY	Terminate the processing.
		DATA ONLY	Compare the data of the appropriate data sets

[1] After SYSUT1 and SYSUT2 have been opened, the value of the LABELS statement is tested and the indicated response is taken.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

3. For totaling exits, valid only for IEBUPDTE and IEBGENER programs, the normal return codes will be processed with the following modifications:

Codes	Utility Response
0	No more exits will be taken. Normal processing is resumed.
4	Continue processing with totaling exits.
8	No more data processing or totaling exits. Data sets will be closed and the requested output trailer label exits (if any) are taken.
16	Program terminates all totaling and data processing and returns control to the supervisor.

Invalid Codes

Under 16 Same as for code 0.

Over 16 Same as for code 16.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

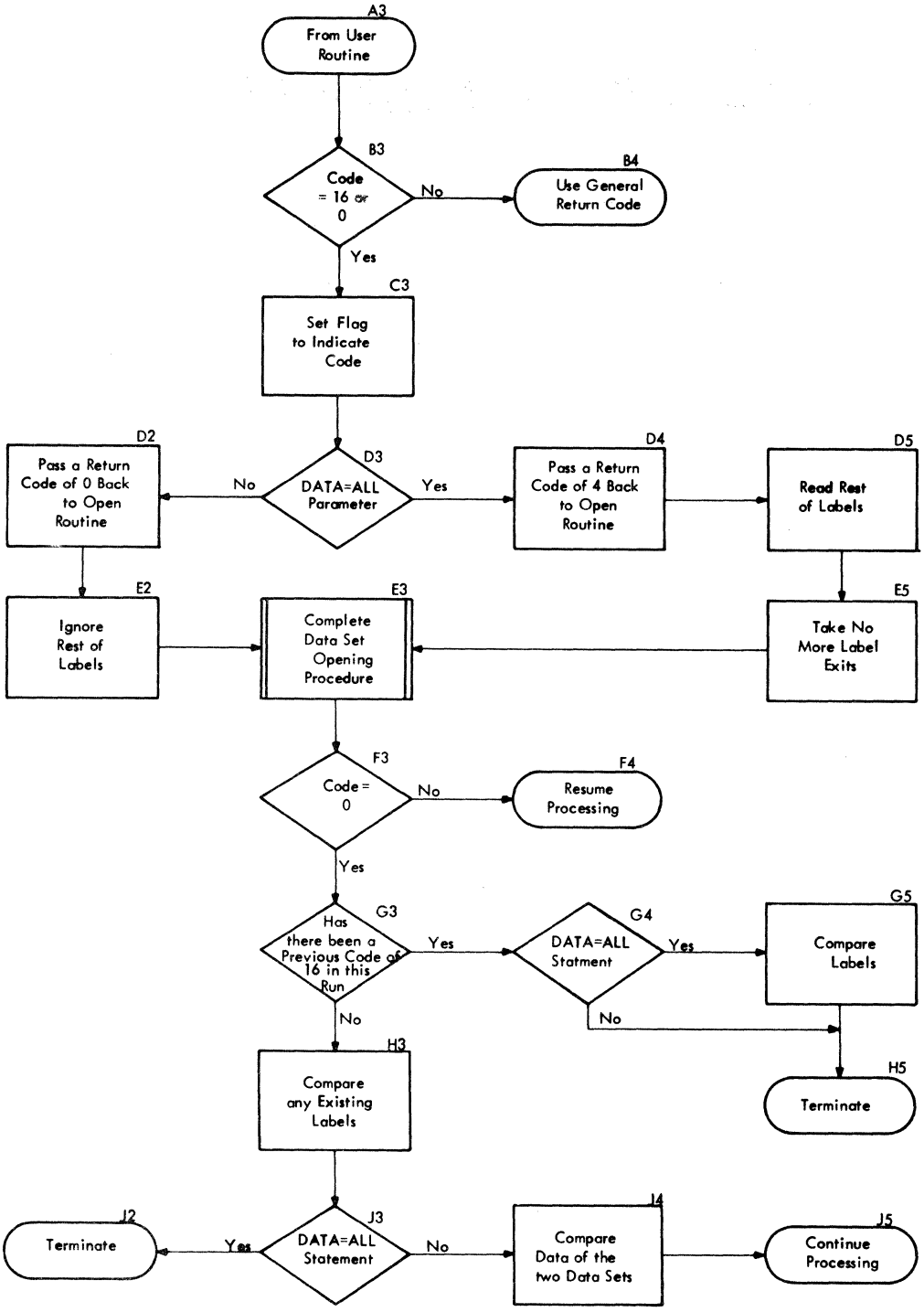


Figure 174. Return Code Modification for IEBCOMPR Program

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

APPENDIX C. DSCB FORMATS FOR THE IEHLIST PROGRAM

The format of the DSCBs as printed by the IEHLIST utility program is given in Figure 175 on page 506.

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

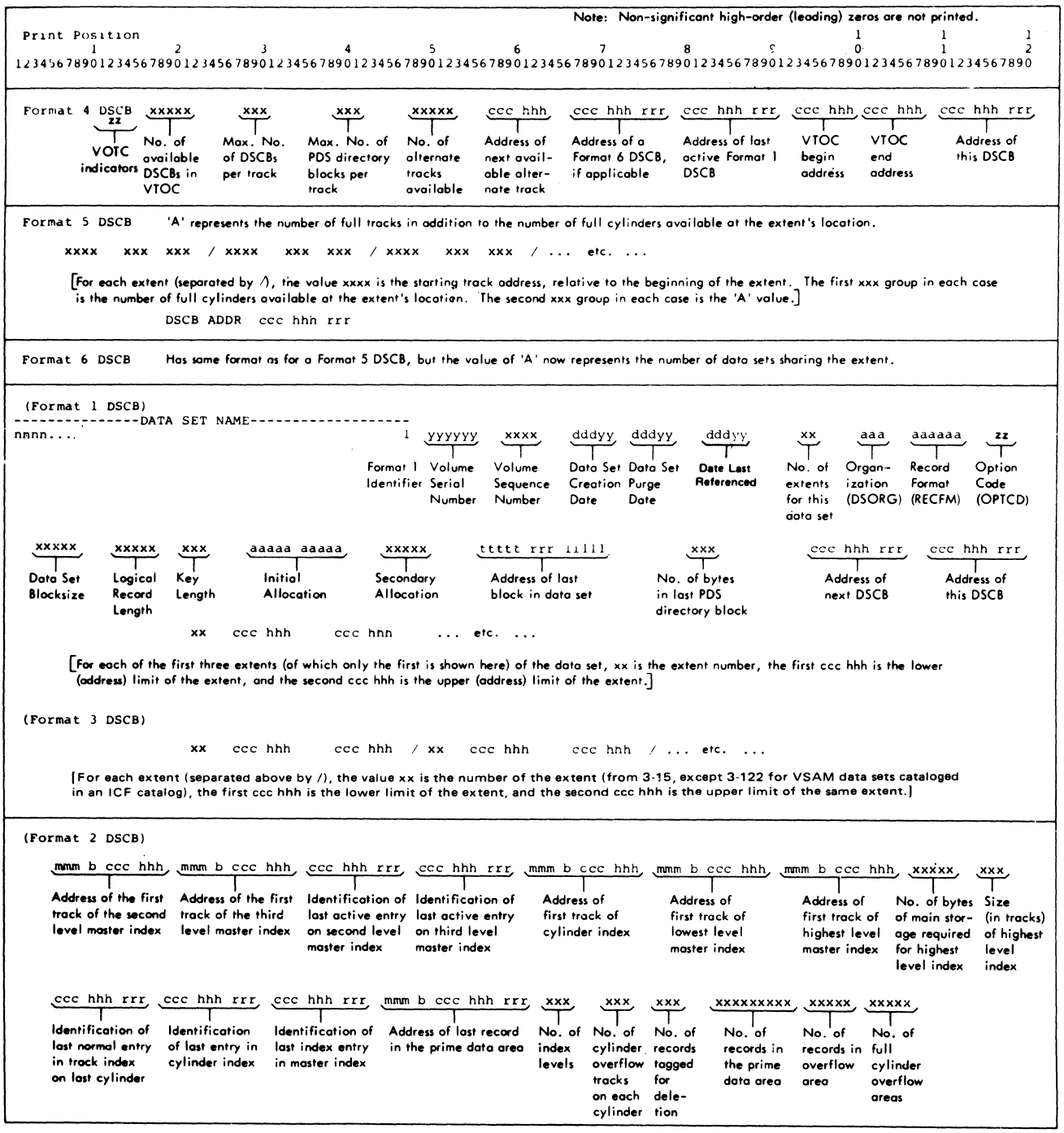


Figure 175 (Part 1 of 2). Description of Fields in Data Set Control Block Formats 1 to 6 on Printout by IEHLIST Program

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Legend			
aa...aa	Refers to Format 1 DSCB abbreviations for option code, record format, and organization given below:		
Name of Format 1 DSCB Printout Field	Printout Position	Abbreviation Appearing on Printout	Meaning of Abbreviation
Initial Allocation			Initial Allocation (on Direct Access)
	29	ABSTR	Absolute track address
		TRKS	Tracks
		CYLS	Cylinders
		RECS	Records
		ROUND	Request (in records) was rounded up to a cylinder boundary.
	35	CONTIG	Contiguous extent requested
		MXIG	Maximum contiguous extent requested
		ALX	Five (or less) of the largest extents (each equal to or greater than the specified minimum) was requested.
DSORG			Data Set Organization
	90	PS	Physical Sequential
		IS	Indexed Sequential
		DA	Direct Access
		PO	Partitioned Organization
		UN	Undefined Organization
	92	U	Unmovable Data Set.
RECFM			Record Format
	96	U	Undefined
		V	Variable
		F	Fixed
		*	Invalid format
	97	B	Blocked
	98	T	Track Overflow
	99	S	If RECFM is F: Standard Blocks (no truncated blocks or unfilled tracks are embedded in the data set)
			If RECFM is V: Spanned Records
	100	A	ASA Control Record
		M	Machine control character

ccc hhh rrr	cylinder, track, and record number.[1]
dddyy	day and year (digits from 00000 to 36699)
mmm b	device identification
nnnn...	DSNAME for Format 1 DSCB
ttttt rrr llllll	digits from 00000 000 00000 to 99999 999 99999.
xx...x	digits from 00...0 to 99...9
yyyyyy	digits from 000000 to 999999 (except for Volume Serial Number in Format 1 DSCB).
zz	hexadecimal digits from 00 to FF

[1] For the 2321 data cell, the ccc hhh part of the identification follows the interpretation given below for addressing:

	Digits	
ccc	1 & 2	Subcell value from 0-19
	3	Strip value from 0-9
hhh	1	Cylinder value from 0-4
	2 & 3	Track value from 0-19

Figure 175 (Part 2 of 2). Description of Fields in Data Set Control Block Formats 1 to 6 on Printout by IEHLIST Program

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

APPENDIX D. MODULE-MICROFICHE CROSS-REFERENCE

Program Name	Entry Name	Module/Microfiche Name
ICAPRTBL	ICAPRTBL	ICAPRTBL
IEBCOMPR	COMPARE	IEBCROOT
		IEBCOMPM
		IEBCULET
		IEBCCS02
		IEBCANAL
		IEBCMAIN
		IEBCQSAM
IEBCOPY	IEBDSCPY	IEBDSCPY
		IEBVMS
		IEBDV1
		IEBLDUL
		IEBSCN
		IEBVCT
		IEBIOE
		IEBRSAM
		IEBCNVT
		IEBDRD
		IEBWSAM
		IEBDSU
		IEBWSU
		IEBDWR
		IEBBAM
		IEBVDM
		IEBCPMOD
		IEBDRB
		IEBMCM
		IEBVTT
		IEBVTM
		IGG019C8
		IGG019FT

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Program Name	Entry Name	Module/Microfiche Name
IEBDG	IEBDG	IEBDG
		IEBCRANL
		IEBCREAT
		IEBDGCUP
		IEBDGMSG
		IEBFDANL
		IEBFDTBL
IEBEDIT	IEBEDIT	IEBEDIT
IEBGENER	IEBGENER	IEBGENRT
		IEBGMESG
		IEBCCS02
		IEBGSCAN
		IEBGENR3
		IEBCONP2
		IEBCONH2
		IEBCONZ2
		IEBEDIT2
		IEBLENP2
		IEBMOVE2
		IEBGENS3
		IEBGEN03
IEBIMAGE	IEBIMAGE	IEBIMAGE
		IEBIMCHA
		IEBIMCOP
		IEBIMFCB
		IEBIMGRA
		IEBIMTAB
		IEBIMWRT
		IEBIMSGS
		IEBISCAN
IEBISAM	IEBISAM	IEBISAM
		IEBISC
		IEBISF
	IEBISF	IEBISMES

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Program Name	Entry Name	Module/Microfiche Name
		IEBISL
	IEBISL	IEBISSI
IEBISPL	IEBISPL	IEBISPL
		IEBISU
	IEBISU	IEBISSO
IEBPTPCH	PRPCH	IEBPPUN1
		IEBPPMSG
		IEBPPAL1
		IEBCCS02
		IEBPPCH1
IEBUPDTE	IEBUPDTE	IEBUPDTE
		IEBUPLOG
		IEBUPDT2
		IEBUPXIT
		IEBUPNIT
		IEBASCAN
		IEBBSCAN
IEHATLAS	IEHATLAS	IEHATLAS
		IEHMESS
		IGC0008F
		IGG0860A
		IGG0860B
		IGG0860C
		IGG0860D
		IGG086AE
IEHINITT	IEHINITT	IEHINITT
		IEHSCAN
		IEHPRNT
		IGC0003I
IEHLIST	IEHLISTR	IEHLIST1
		IEHLIST2
		IEHLIST3
IEHMOVE	IEHMVESD	IEHMOVE
	IEHMVERA	IEHMVSRA
	IEHMVERA	IEHMVSRK

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Program Name	Entry Name	Module/Microfiche Name
	IEHMVERD	IEHMVSRD
	IEHMVERD	IEHMVSRM
		IEHMVESA
		IEHMVESC
		IEHMVESH
		IEHMVESI
		IEHMVESJ
		IEHMVESK
		IEHMVESL
		IEHMVESM
		IEHMVESN
	IEHMVESN	IEHMVMSN
		IEHMVESO
		IEHMVESP
		IEHMVESQ
		IEHMVMSQ
		IEHMVESR
	IEHMVESS	IEHMVSSS
		IEHMVEST
		IEHMVESU
	IEHMVESV	IEHMVSSX
	IEHMVESV	IEHMVSRY
	IEHMVESV	IEHMVSTC
	IEHMVESV	IEHMVMRY
	IEHMVESX	IEHMVSRV
	IEHMVESX	IEHMVSRZ
	IEHMVESX	IEHMVSRK
	IEHMVESX	IEHMVMRZ
	IEHMVESY	IEHMVSRX
	IEHMVESY	IEHMVSRM
	IEHMVESY	IEHMVSSY
	IEHMVESY	IEHMVMSY
	IEHMVESZ	IEHMVSSZ
	IEHMVESZ	IEHMVSSV
	IEHMVESZ	IEHMVMRZ

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

Program Name	Entry Name	Module/Microfiche Name
	IEHMVETA	IEHMVSTA
	IEHMVETA	IEHMVSRK
	IEHMVETA	IEHMVSRM
	IEHMVETA	IEHMVMTA
		IEHMVETG
		IEHMVETJ
	IEHMVETL	IEHMVSTL
	IEHMVETL	IEHMVSRK
	IEHMVETL	IEHMVSRM
	IEHMVETL	IEHMVMTL
	IEHMVESE	IEHMVXSE
		IEHMVXSF
	IEHMVESF	IEHMVSSF
IEHPROGM	IEHEBASE	IEHPROG1
		IEHPROG2
		IEHPROG3
		IEHPROG4
		IEHPROG5
IFHSTATR	IFHSTATR	IFHSTATR

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

INDEX

A

access method for
 buffering—IEBGENER 217
alias names-IEBCOMPR 20
allocation of storage—IEBCOPY 66
alternate ddnames—IEBGENER 211
analysis of CREATE card—IEBDG 158
ATLAS
   general discussion 392
   restriction with 3330V 392
auxiliary parameters 2

B

base module function—IEBDG 150
block size vs. logical record
 length—IEBCOPY 87
blocking records in buffer—IEBCOPY 85
buffer allocation—IEBCOPY 78, 94
buffering techniques—IEBGENER 209, 217
building parameter lists for
 IEHPROGM 469
building table entries in
 storage—IEBCOPY 56

C

calling ATLAS 401, 403
catalog, listing an OS 440
ceiling quotient 79
ceiling—IEBCOPY 81
chaining alternate track—ATLAS 395
chaining create tables—IEBDG 158
chaining FD address tables—IEBDG 160
channel program example—ATLAS 410
channel program for
 "PCI-read"—IEBCOPY 87
channel program modification—ATLAS 405
channel program modification—IEBCOPY
 for writing 98
channel programs—ATLAS 392, 395
common communication area—IEBDG 150
common work area—IEBCOMPR 27
communication area
   IEBCOPY 56
   IEBDG 169
   IEHMOVE 458
communication region—IEBCOMPR 48
comparing header and trailer labels 32
comparing input records—IEBCOMPR 40
comparing labels as data 40
comparing member names—IEBCOPY 60
comparing records 40
compressing in place 108
constants for IEBDG 174
continuation card—IEBDG 149
control card scanner for IEHMOVE and
 IEHLIST 6
control character for punched
 output—IEBPTPCH 319
control statement
 processing—IEBGENER 205, 209
control statement
 processing—IEBPTPCH 314, 319
control table—IEBCOPY 64
copying a BSAM data set 456
copying a PDS—IEHMOVE 456
copying and modifying
 records—IEBGENER 204
cross-reference, module—microfiche 508
CTLTAB entry construction 64
CTLTAB table sorting 125
CVOL
   See OS control volume

D

data block sizes—IEBCOPY 87
data check error—ATLAS 392, 410
data control block—IEBDG 173
data management, used by IEHPROGM 467,
 468
data recovery—IEHATLAS 392
data set
   indexed sequential
      copying 296
      input stream 194
      listing the directory of a
       partitioned 440
      loading 300
      printing 301
      unloading 298
   moving or copying 448
   scratching a 467
data set compression—IEBCOPY 108
data set processing—IEBGENER 209
data set utility programs 7
data transactions—IEBUPDTE 383
DCB exit list—IEHMOVE 459
DCB exit routine—IEBDG 150
DCB name comparison—IEBDG 161
ddname parameter list—IEBGENER 210
ddname storage—IEBGENER 207
deblocking input records—IEBCOMPR 40
deblocking records in
 buffer—IEBCOPY 85
debugging aids—IEBDG 167
default values IEBDG field definition
 (FD) 156
default values—IEBDG 157
device mask table 4
directory scan—IEBCOPY 113
directory setup for an
 unload—IEBCOPY 126
do not copy bit, IEBCOPY 126
DSCB formatting—IEHLIST 444

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

E

edit control block—IEBGENER 219
editing of data—IEBPTPCH 317, 333
end-of-file situation—ATLAS 412
enqueuing the VTOC—ATLAS 403
equates for IEBDG 175
erasing tracks—IEBCOPY 103
error analysis—IEBCOPY 104
error record definition—IEHATLAS 392
error statistics by volume—IFHSTATR 484
errors during channel commands—IEHATLAS 392
exit routines—IEBUPDTE 384

F

FD table operations—IEBDG 163
FD table value changes—IEBDG 164
FD table—IEBDG 156, 157
field select—IEBDG 145, 153
file protected tape— 427
filling a pattern field—IEBDG 157
finding members to process—IEBPTPCH 343
flow of control—IEBEDIT 194
flow of control—IEHATLAS 393, 395
functions of program segments—IEBGENER 205, 207

G

generation of output records—IEBDG 163

H

H-set converter—IEBGENER 207
head switching—IEBCOPY 88
header information-IEBCOMPR 19
header information—IEBCOMPR 44
header information—IEBPTPCH 336

I

I/O error—IEBISAM 300
ICAPRTBL 12, 488
IEBCOMPR 15, 488
IEBCOPY 488
IEBDG 145
  cleanup function 153
  FD pattern construction 156
  FD table
    construction 156
    modification 163
    updating 164
  generalized module functions 145
  modifying the output record 163
  module residence 145
  output data set records 149
  processing control cards
    CREATE 158
    DSD 151
    DUMP 153
    END 153
    FD 154
    REPEAT 153
  reading control cards 151
  scanning control cards 149
  tables used by create module 158
IEBEDIT 194, 490
IEBGENER 204, 490
IEBIMAGE 241, 492
  data areas 286
    CESD records 293
    IEBIBLKS 286
    KEYTAB 291
    OPTAB 292
  method of operation diagram 244
  modules
    description 241
    flowcharts 249
  procedures—reference list 248
  program organization 248
IEBISAM 296, 492
  initialization of 296
  termination of 301
IEBPTPCH 312, 493
IEBUPDTE 493
IEHATLAS
  general discussion 392, 494
  restriction with 3330V 392
IEHINITT 425, 494
IEHLIST 440, 494
IEHMOVE 448, 495
IEHPROGM 467, 497
IFHSTATR 484, 497
increment-restore fields—IEBDG 165
INDD table—IEBCOPY 65
independent utility programs 8
initializing unload/load—IEBCOPY 104
input ddname table—IEBCOPY 65
input for modules—IEBDG 177
input parameter list-IEBCOMPR 18
input parameter storage—IEBPTPCH 314, 317, 319
input record format—IEBGENER 216
input to program
  IEBCOMPR 15
  IEBGENER 204
  IEBPTPCH 312
invoking
  IEBCOMPR 18
  IEBCOPY 54
  IEBGENER 209
  IEBISAM 296
  IEBPTPCH 317
  IEHATLAS 392

K

keyword processing from CREATE card—IEBDG 160

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

L

label buffer area 498
label processing routines—IEBGENER 211
label processing—IEBCOMPR 36
labeling tapes 426
listing alternate ddnames-IEBCOMPR 18
listing system control data 440
loading a BSAM data set 457
loading data—IEHMOVE 455
locating a gap between
members—IEBCOPY 108

M

main processor
initialization—IEBPTPCH 314
main storage allocation—IEBCOPY
directory entries 68
I/O Buffers 68, 71
MBBCCHHR conversion—IEBCOPY 80
member address within
directory—IEBCOPY 87
member data block format—IEBCOPY 75
member name sorting—IEBPTPCH 314, 324,
326
merging directory entries—IEBCOPY 117
merging partitioned data sets 54
message actions—IEBCOPY 123
message numbers—IEBCOMPR 45
message processing—IEBDG 165
message processing—IEBPTPCH 317, 336
missing address marker—ATLAS 410
modifying output records—IEBDG 163
modifying system control data 163
module functions-IEBCOMPR 15
module-microfiche cross reference
(Appendix D) 508
modules, list of (Appendix A) 488
moving and copying data 448
moving labels 448
multiple references to FD
table—IEBDG 164

N

new-name pointer table—IEBCOPY 60
note list TTRNs—IEBCOPY 95

O

obtaining input directory entry
address—IEBCOPY 110
opening data sets—IEHATLAS 398
opening input data sets for
QSAM-IEBCOMPR 15
OS control volume (CVOL)
IEHMOVE 448, 455
IEHPROGM 467
out-of-extent condition—ATLAS 392
output from modules—IEBDG 177
output from program
IEBCOMPR 15
IEBGENER 204
IEBPTPCH 313
overflow records—ATLAS 408
overflow records—IEBCOPY 128
overflow records—IEBISAM 300
overflow storage factor—IEBCOPY 71
overlay structure
IEBCOMPR 15
IEBPTPCH 313

P

packed to zoned converter—IEBGENER 207
page header parameter
list—IEBGENER 210
parameter conversion—IEBDG 157
parameter list contents—IEBPTPCH 319
parameter list, label processing 498
parameter list—ATLAS 401
parameter list—IEHINITT 427
parameter list—IEHPROGM 474
parameter storage—IEBPTPCH 314, 317
partitioned data set
listing the directory of members,
copying and merging 111
IEHMOVE 448
modifying the directory of 467
moving or copying a 448
updating a 383
password modification—IEHPROGM 467
pattern location movement—IEBDG 163
pattern storage
determination—IEBDG 156
PCI appendage—IEBCOPY 130
phases of IEHPROGM 474
physical sequential data set record
format IEBISU) 298
picture string specification—IEBDG 163
prescan of output directory—IEBCOPY 93
printing an OS CVOL 440
printing and punching records 313
printing messages—IEBDG 166
processing procedures for ATLAS 401

Q

QSAM used by system utilities 2, 6

R

random number generator—IEBDG 165
RDCDRT scan routine 6
re-executable channel
program—ATLAS 414
Read routine for unloaded data
sets—IEBCOPY 112
Read routine—IEBCOPY 87
read-back check—IEBCOPY 98
record descriptor table
(RDT)—IEBGENER 219
record format vs
processing—IEBPTPCH 329
record format-IEBCOMPR 15
record formats
how to find 459

This document contains restricted materials of IBM. © Copyright IBM Corp. 1972,1983
LY26-3926-1

SYSUT1 450
SYSUT2 454
SYSUT3 455
record identification—IEBPTPCH 332
record overflow—ATLAS 414
records
comparing 40
copying and modifying 205
renaming data set members 54
return codes
ATLAS 395
IEBCOMPR 47
routing tables—IEHPROGM 474

S

saving input labels—IEBCOMPR 36
scanning a control statement—IEBEDIT 194
scanning control statements 113
scratch VTOC request 472
select-exclude table—IEBCOPY 57
SETAB table sorting 116
SETAB/CTLTAB table—IEBCOPY 56, 64
SIO appendage—IEBCOPY 131
sorting a SETAB table 116
spilling directory entries—IEBCOPY 90, 98
stand-alone utility programs
See independent utility programs
statement errors-IEBCOMPR 20
status information address 498
storage allocation—IEBCOPY 66
subroutines of card scanner RDCDRT 6
support utility programs
See independent utility programs
SVC 86 instruction 395
SYNAD routine—IEBDG 152
syntax error messages 113
system control data
listing of 440
modification of 467
system utilities 2
system utility programs 2
SYSUT1 record processing—IEBPTPCH 314, 319
SYSUT4 data set use—IEBCOPY 118

T

table allocation—IEBCOPY 56
table storage and construction—IEBGENER 213
tables built by create module—IEBDG 158
tape label image—IEHINITT 425
termination procedures—IEHATLAS 399
termination processing—IEHMOVE 458
totaling image area 498
track balance—IEBCOPY 128
track chaining—ATLAS 414
track descriptor field—ATLAS 404
track overflow—ATLAS 408
track-overflow channel program—IEBCOPY 128
true name correspondence—IEBCOMPR 31
TTR conversion—IEBCOPY 80

U

unequal record comparison—IEBCOMPR 44
unequal true names—IEBCOMPR 31
unloaded records—IEBCOPY 73
unloaded version of a data set 448
unloading a BSAM data set 457
unloading a partitioned data set 72
user exit—IEBISAM 301
user label processing exits
IEBCOMPR 36
IEBUPDTE 384
IEHMOVE 452
return codes 500
user modifications to records—IEBDG 163
user parameter list—IEBGENER 210
user return codes—IEBISAM 301
user totaling exit routine
IEBGENER 213
IEBUPDTE 384
utility programs, types of 2

V

validity of record format—IEBCOPY 97
variable length records for IEBISAM 297
variable spanned records
IEBGENER 209, 217
IEHMOVE 454
volume lookup—IEHPROGM 473
volume mounter—IEHMOVE 452
volume moving or copying 482
scratching a data set from 467
volume statistics report 484
volume table of contents
listing of 440
writing of 300
VS record processing 454
VTOC format routines 445

W

work area formats—IEHMOVE 459
work area—IEBPTPCH 337
work area—IEHATLAS 398
work areas
IEBCOMPR 48
IEBGENER 222, 225
work data set record formats for IEHMOVE 459
Write routine for unloaded data sets—IEBCOPY 126
Write routine—IEBCOPY 98
write setup module—IEBCOPY 127
writing messages—IEBGENER 209, 222
writing volume label sets 425

Z

zoned to packed converter—IEBGENER 207

MVS/370 Utilities Logic
LY26-3926-1

Reader's Comment Form

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Please use pressure sensitive or other gummed tape to seal this form.

List TNLs here:

If you have applied any technical newsletters (TNLs) to this book, please list them here:

Last TNL _____

Previous TNL _____

Previous TNL _____

Fold on two lines, tape, and mail. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.) Thank you for your cooperation.

LY26-3926-1

Reader's Comment Form

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape | Please do not staple | Fold and tape

IBM®

LY26-3926-1 File No. S370-37 Printed in U.S.A. 3/79 Supplies Logic of the IBM 3705

Restricted Materials of IBM Corporation
LY26-3926-1

MVS/370 Utilities Logic (File No. S370-32) Printed in U.S.A. LY26-3926-1

