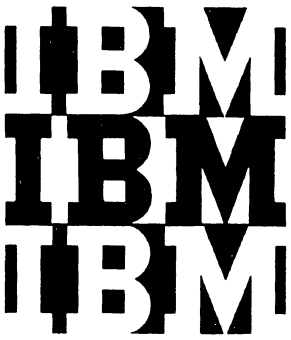


File No. S360-36
GY27-7236-1

Program Logic

OS Release 21

IBM System/360 Operating System

MFT Supervisor

Program Number 360S-CI-505

This publication describes the internal logic of the MFT Supervisor. The MFT Supervisor is one part of the control program of the IBM System/360 Operating System. The publication provides an introduction to control program logic and describes the supervisor components of the program. Specifically, the publication describes:

Interruption Supervision
Task Supervision
Task Termination
Contents Supervision
Main Storage Supervision
Communications Task
Timer Supervision
Overlay Supervision
Recording/Recovery Interfaces
Checkpoint/Restart
Device Independent Display Operator Console Support

Program Logic Manuals are intended for use by persons involved in program maintenance, and system programmers who are altering the program design. Program logic information is not necessary for program use and operation.

Information in this publication applies only to systems capable of multiprogramming with a fixed number of tasks (MFT).

Second Edition (March, 1972)

This is a major revision of and makes obsolete GY28-7236-0 and its associated TNL, GN27-1364. This edition corresponds to Release 21 of System/360 Operating System. It provides information concerning the Generalized Trace Facility and Status Display Support; enhancements and modifications to SYS1.LOGREC, RMS, DIDOCS, OLTEP, and ABEND; and other improvements and corrections to existing system features.

Release 20 added information concerning SVC DUMP, RMS for the Models 165 and 155, System/370 Time-of-Day Clock, and the Extended Precision Floating Point Decimal Simulator.

Significant changes or additions to the specifications contained in this publication are continually being made. When using this publication in connection with the operation of IBM equipment, check the latest SRL Newsletter for revisions or contact the local IBM branch office.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer using a special print train.

Copies of this and other IBM publications can be obtained through IBM branch offices.

A form for reader's comments appears at the back of this publication. Address any additional comments concerning the contents of this publication to IBM Corporation, Programming Publications, Department 636, Neighborhood Road, Kingston, New York 12401

This publication describes the MFT Supervisor, its relationship to the rest of the control program, and the interaction of its modules. The information in this publication is intended for personnel who are responsible for determining sources of errors within or making modifications to the MFT Supervisor.

The MFT Supervisor PLM is comprised of the following six sections: "Section 1. Introduction" describes the relationship of the supervisor to the other parts of the control program, its general characteristics, and its optional special features. "Section 2. Method of Operation" provides an overview of the services provided by the supervisor. "Section 3. Program Organization" describes in text and flowcharts the logical processing of each module in the supervisor and provides a synopsis of each module (module name, CSECT name, flowchart cross-reference, entry points, exits, other modules used, tables and work areas). "Section 4. Module Directory" cross-references entry points, module names, and CSECT names. "Section 5. Data Areas" contains the formats of the control blocks, request blocks, and work areas used by the supervisor and describes the fields within each area. "Section 6. Diagnostic Aids" contains information that may be useful in determining the source of an error within the supervisor.

The "Method of Operation" section and the module descriptions, module synopses, and flowcharts with the "Program Organization" section are organized by supervisor components. These components are, in order of presentation:

- Interruption Supervision
- Task Supervision
- Task Termination
- Contents Supervision
- Main Storage Supervision
- Communications Task
- Timer Supervision
- Overlay Supervision
- Recording/Recovery Services
- Checkpoint/Restart
- Device Independent Display Operator Console Support
- Special Features

Because you may wish to refer to the same diagram for several separate text descriptions, Diagrams 1 through 21 are provided as foldouts and are located immediately after Section 6.

PREREQUISITE PUBLICATIONS

IBM System/360 Principles of Operation, GA22-6821

IBM System/360 Operating System: Introduction to Control Program Logic, GY28-6605

PUBLICATIONS TO WHICH THE TEXT REFERS

IBM System/360 Operating System: Messages and Codes, GC28-6631

Operator's Guide, GC28-6540

Operator's Reference Manual, GC28-6691

System Generation, GC28-6554

Supervisor Services and Macro Instructions, GC28-6646

Input/Output Supervisor, GY28-6616

Job Control Reference, GY28-6704

Job Management with MFT, GY27-7128

Machine-Check Handler for IBM System/360 Model 65, GY27-7155

Machine-Check Handler for IBM System/360 Model 85, GY27-7184

Machine-Check Handler for the System/370 Models 135 and 145, GY27-7237

Machine-Check Handler for IBM System/370 Models 155 and 165, GY27-7198

Online Test Executive Program, GY28-6651

Programmer's Guide to Debugging, GY28-6670

Sequential Access Method, GY28-6604

Service Aids Logic, GY28-6721

Utilities, GY28-6614

Summary of Changes
Release 20 SVC DUMP RMS for Models 155 and 165 System/370 Time-of-Clock Extended Precision Floating Point Decimal Simulation
Release 21 Generalized Trace Facility Status Display Support SYS1.LOGREC RMS DIDOCS OLTEP ABEND

SECTION 1: INTRODUCTION 1
The Relationship of the MFT Supervisor to the Control Program 1
Purpose and Function of the MFT Supervisor 1
Tracing Facilities 4
Special Features 4
Operating Environment 4
Logical Characteristics 7
Summary 9

SECTION 2: METHOD OF OPERATION 11
Interruption Supervision 11
Passing Control to the Interruption Handling Portion of the Supervisor 11
Types of Interruptions 11
Supervisor Call (SVC) Routines 11
Interruption Supervision Routines 12
SVC Control Information 12
Resident Type 3 and 4 SVC Routine Option 13
Pseudo-Disabled Interruption Supervision 14
Interruption Supervision Control Flow 14
  Supervisor Call Interruptions 14
  Timer/External Interruptions 15
  Input/Output Interruptions 15
  Program Check Interruptions 15
  Dispatcher Processing 15
  Handling Machine-Check Interruptions 16

TASK SUPERVISION 17
Task Supervision Services 17
  Concurrent Execution of Programs within a Partition 17
  Changing the Dispatching Priority of a Task 17
  Extracting Information from a Task Control Block (MFT with Subtasking) 17
  Limited Extract (MFT without Subtasking) 17
  Communicating the Completion of an Event 17
  Gaining Exclusive Access to System Resources 17
  Serializing the Use of Shared Data Sets (Shared DASD only) 18
  Increasing System Availability 18
The Functions of Task Supervision 18
  Creating a Subtask 18
  Detaching a Subtask 20
  Changing the Dispatching Priority of a Task 20
  Causing a Program to Wait for One or More Events 20
  Posting the Completion of an Event 21
  Serializing the Use of a Resource 21
  Signaling That a Reserved Resource is no Longer Needed 25
  Intercepting Program-Check Interruptions 25
  Specifying a Task Asynchronous Exit 25
  Scheduling a User Exit Routine 27

TERMINATION SERVICES 30
Abnormal Termination Scheduling 30
Normal and Abnormal Termination 30
STAE Scheduling and Processing 32
Damage Assessment 32
Dumping 32

CONTENTS SUPERVISION 33
Contents Supervision Routines 33
Contents Supervision Functions 33
Creating and Maintaining Request Blocks 33
Servicing Requests For Supervisor-Assisted Linkage 35

The Transient Areas 36
Identifying Embedded Entry Points 37
Deleting a Loaded Module 37
Synchronizing Control Program and Problem Program Execution 37

MAIN STORAGE SUPERVISION 39
Main Storage 39
GETMAIN 39
FREEMAIN 40

COMMUNICATIONS TASK 41
Input Processing 41
Output Processing 41
Communication Task Processing 41
Multiple Console Communications 41
The System Log Function 42

TIMER SUPERVISION 45
System/360 Timer Supervision Services 45
System/370 Timer Supervision Services 45
Functions of Timer Supervision 46
  Measuring Elapsed Time in System/360 46
  Measuring Elapsed Time in System/370 47
  Maintaining the Timer Queue 48
  Measuring Time of Expiration in System/360 48
  Measuring Time of Expiration in System/370 49
  Determining Interval Values in System/360 49
  Determining Interval Values in System/370 49
  Timer Interruption Handling 50
  Determining What Actions Are To Be Performed in System/360 50
  Determining What Actions are to be Performed in System/370 51
  Altering Timer Requests 51
SMF Processing and Job Step Timing 52

OVERLAY SUPERVISION SERVICES 53
Overlay Supervision Routines 53
Functions of Overlay Supervision 53
Tables Used by Overlay Supervision 53
  Building and Initializing Control Tables 54
  Servicing Requests for Linkage to Overlay Program Segments 55
  Passing of Control 58

RECORDING/RECOVERY SERVICES 60
Machine-Check Routines 60
Channel-Check Handler 60
Alternate Path Retry Routine 60
Dynamic Device Reconfiguration Routine 61
Systems Without Recording/Recovery Routines 61
Entry to Recording/Recovery Routines 61

CHECKPOINT/RESTART 62

DEVICE INDEPENDENT DISPLAY OPERATOR CONSOLE SUPPORT (DIDOCS) 63

TRACING FACILITIES 65
Trace Table Facility 65
IEATRC 65
Generalized Trace Facility 65

SPECIAL FEATURES 66
Extended Precision Floating Point Simulator 66
System Management Facility 66
  Recording System Wait Time 66
  Handling Time/Output Limit Expiration 67
  Counting References to User Data Sets 67
  Controlling Output Limit for SYSOUT 67
  Recording Storage Blocks Assigned to User Programs 67

SECTION 3: PROGRAM ORGANIZATION 68
Interruption Supervision Routines 68
SVC First-Level Interruption Handler (IEAAIH00) Chart AA 68
Type-1 SVC Exit Routine (IEAAIH00) Chart AA 68
SVC Second-Level Interruption Handler (IEAATA00) Chart AB 68
Extended SVC Router (ESR) Chart AI 69
Exit (IEAATA00) Chart AC 69
Timer/External First-Level Interruption Handler (IEAAIH00) Chart AD . 70
Input/Output Interruption Handler (IEAAIH00) Chart AD 70
Program Check First-Level Interruption Handler (IEAAIH00) Chart AE . 71
Dispatcher (IEAAIH00) Chart AF 71

TASK SUPERVISION ROUTINES 78
ATTACH (with Subtasking) (IEAQAT00) Chart BA 78
CHAP (Only with Subtasking) (IEAQCH00) Chart BC 79
Detach (only with Subtasking) (IEAGED02) Chart BB 79
Extract (without Subtasking) (IEAAXR00) Chart BD 80
EXTRACT (with Subtasking) (IEABXR00) Chart BE 80
WAIT (IEAAWT00) Chart BF 81
POST (IEAAPT00) Chart BG 82
ENQ (IEAGENQ1) Chart BH 83
DEQ (IEAGENQ1) Chart BJ 85
SPIE (IEAAPX00) Chart BK 87
STAE (IEAAST00) Chart BL 88

NORMAL AND ABNORMAL TERMINATION ROUTINES 90
ABEND 0 -- Normal Termination and Abnormal Routing (IEANTM00) Chart
CA 90
ABEND G -- Abnormal Termination and ABEND Recursion (IEANTM0G)
Chart CB 91
ABEND 1 -- ABEND/STAE Graphics Linkage (IEANTM01) Chart CC 91
ABEND 2 -- PURGE I/O (IEANTM02) Chart CD 91
ABEND 3 -- Control Block Validity Check (IEANTM03) Chart CD 91
ABEND 4 -- Dump Test (IEANTM04) Chart CE 92
ABEND 5 -- Open Dump Data Set (IEAMTM05; without subtasking) 92
ABEND 5 -- Open Dump Data Set (IEANTM05; with Subtasking) Chart CF . 92
ABEND 6 -- Dump (IEANTM06) Chart CG 92
ABEND 7 -- Termination (IEANTM07) Chart CH 93
ABEND 8 -- Indicative Dump (IEANTM08) Chart CH 93
ABEND 9 -- Recursion Processing (IEANTM09) Chart CI 93
ABEND 10 -- Steal Main Storage (IEANTM0A) Chart CJ 94
ABEND 11 -- WTOR Purge (IEANTM0B; without Multiple Console Support)
Chart CK 94
ABEND 11 -- WTOR PURGE (IEACTM0B; with Multiple Console Support) . . 94
ABEND 12 -- Loading Program Purge (IEANTM0C; with Subtasking Only)
Chart CK 94
ABEND 13 -- Subtask ENQ Purge (IEANTM0D; with Subtasking Only)
Chart CK 94
ABEND 14 -- Data Set Close (IEANTM0E; With Subtasking Only) Chart CL 94
ABEND 15 -- IQE Purge and Subtask Cleanup (IEANTM0F; with
Subtasking Only) Chart CM 94
ABEND M -- WTP Processing for Type 1 SVC Routines (IEANTM0M)
Chart CN 95
ABEND H -- WTO Issuing for Valid Recursions (IEANTM0H) Chart CO . . . 95
ABEND J -- JPAQ and GQE Validity Check (IEANTM0J) Chart CP 95
ABTERM -- (IEAGAB00; without the Trace Table) Chart CY 95
ABTERM -- (IEAIAB00; with the Trace Table) 95
ABTERM -- (IEANAM00; with Subtasking) Chart CY 95
ASIR 1 -- ABEND/STAE Interface Routine 1 (IEASTM11) Chart CQ 96
ASIR 2 -- ABEND/STAE Interface Routine 2 (IEASTM12) Chart CR 96
ASIR 3 -- ABEND/STAE Interface Routine 3 (IEASTM13) Chart CS 96
ASIR 4 -- ABEND/STAE Interface Routine 4 (IEASTM14) Chart CT 96
DAR 1 -- Damage Assessment Routine 1 (IEADTM22) Chart CU 96
DAR 2 -- Damage Assessment Routine 2 (IEADTM23) Chart CV 97
DAR 3 -- Damage Assessment Routine 3 (IEADTM24) Chart CW 97
DAR 4 -- Damage Assessment Routine 4 (IEADTM24) Chart CX 97
SVC DUMP (IEAAAD0Y, IEAAAD0Z) Charts DA and DB 97

ABDUMP 1 (IEAMAD00) Chart DC 98
ABDUMP 2 (IEAAAD0D) Chart DD 99
ABDUMP 3 (IEAAAD0A) Chart DD 99
ABDUMP 4 (IEAAAD01) Chart DE 99
ABDUMP 5 (IEAAAD02) Chart DF 99
ABDUMP 6 (IEAAAD03) Chart DF 99
ABDUMP 7 (IEAAAD0B) Chart DG 100
ABDUMP 8 (IEAAAD04) Chart DG 100
ABDUMP 9 (IEAAAD05) Chart DH 100
ABDUMP 10 (IEAAAD0E) Chart DH 100
ABDUMP 11 (IEAAAD0C) Chart DH 100
ABDUMP 12 (IEAAAD0F) Chart DI 100
ABDUMP 13 (IEAAAD0K) Chart DJ 100
ABDUMP 14 (IEAAAD0M) Chart DK 100
TCAM ABDUMP (IEAAAD0G, IEAAAD0H, IEAAAD0I, IEAAAD0J) Charts DL and DM 100

CONTENTS SUPERVISION ROUTINES 102
Attach (IEAAAT00; without Subtasking) Chart EA 102
DELETE (IEAADL00) Chart EF 102
Load (IEAATC00) Chart EB 103
LINK (IEAATC00) Chart EB 104
XCTL (IEAATC00) Chart EB 104
FINCH (IEAATC00) Charts EC and ED 104
IDENTIFY (IEAAID00) Chart DE 105
SYNCH (IEAASY00) 105

MAIN STORAGE SUPERVISION ROUTINES 111
GETMAIN (IEAAMS) Chart FA 111
FREEMAIN (IEAAMS) Chart FA 112

COMMUNICATION TASK ROUTINES 113
Initialization Module (IEECVCTI) Chart GA 113
Graphic Console Initialization (IEECVGCI) Chart GY 113
Write to Operator Module (IEEMFWTO, IEENFWTO) Chart GC 113
Non-MCS Multiple-Line Write to Operator (IEECVML1, IEECVML2, IEECVML4) Chart GD 114
MCS Multiple-Line Write to Operator Routines (IEECVML3, IEECVML5, IEECVML6, IEECVML7) Chart GD 114
Write to Operator with Reply Module (IEEVWTOR) Chart GE 115
External Interruption Handler (IEECVCRX) Chart GB 115
Attention Interruption Handler (IEECVCRA) Chart GB 115
Wait Module (IEECVCTW) Chart GF 115
Router Module (IEECVCTR) Chart GG 115
Console Switch Module (IEECVCTX) Chart GH 115
Unit Control Module (IEECVUCM) 115

COMMUNICATIONS TASK ROUTINES FOR MULTIPLE CONSOLE SUPPORT 116
MCS Mini-Router Module (IEECMCTR) 116
MCS Router Module (IEECMAWR) Chart GI 116
MCS Console Switch Modules (IEECLCTX, IEECMCTX, IEECNCTX, and IEECOCTX) Chart GJ 116
MCS Device Interface Module (IEECMDSV) Chart GK 117
WTO Processor Module (IEECMWSV) Chart GL 118
DOM Service Module (IEECMDOM) Chart GM 118
NIP Message Buffer Writer Module (IEECMWTL) Chart GN 118

COMMUNICATIONS TASK DEVICE SUPPORT ROUTINES 119
1052 Device-Support Routines 119
Printer Device Support Routines 119
Card Reader Device Support Routines 119
2740 Device Support Routines (MCS only) Chart GT 120

TIMER SUPERVISION ROUTINES 121
Timer Second Level Interruption Handler (IEA0TI00)* Chart HB 121
Time (IEA0RT00) Chart HA 121
TTIMER (IEA0ST00) Chart HC 121
STIMER (IEA0ST00) Chart HD 121

SMF Time Limit Expiration Routine (IEATLEXT) Chart MB 123

OVERLAY SUPERVISION ROUTINE DESCRIPTIONS 124
Resident Overlay Supervisor (IEWSVOVR) Chart IA 124
Nonresident Overlay Supervisor (IEWSZOVR) Chart IA 124

CHECKPOINT/RESTART ROUTINES 125
CHKPT (SVC 63) Chart LA 125
Restart (SVC 52) Chart LB 127

RECORDING/RECOVERY ROUTINES 131
SER Routines 131
SER0 Chart AG 131
SER1 Chart AH 132
Environment Recording Area 133

DEVICE INDEPENDENT DISPLAY OPERATOR CONSOLE SUPPORT (DIDOCS)
ROUTINES 134
DIDOCS Processor 0, Load 1 (IGC6107B) CHART KA 134
DIDOCS Processor 0, Load 2 (IGC6Z07B) Chart KB 134
DIDOCS Processor 1, Load 1 (IGC5107B) Chart JA 134
DIDOCS Processor 1, Load 2 (IGC5Z07B) Chart JB 135
OPEN/CLOSE Routine (IGC5G07B) Chart KK 135
2250 I/O 1 Routine (IGC5P07B) Chart JC 135
2250 I/O 2 Routine (IGC5Q07B) Chart JD 136
2260 I/O 1 Routine (IGC5R07B) Chart JE 136
2260 I/O 2 Routine (IGC6R07B) Chart JF 136
Model 85 I/O Routine (IGC5H07B) Chart JG 136
Asynchronous Error Routine (IGC5C07B) Chart KC 137
Message 1 Routine (IGC5D07B) Chart JH 137
Message 2 Routine (IGC5E07B) Chart JI 137
Message Routine 3 (IGC6D07B) Chart JJ 137
Display 1 Routine (IGC5207B) Chart JK 137
Display 2 Routine (IGC5307B) Chart JL 138
Display 3 Routine (IGC6207B) Chart JM 138
Roll Mode Routine (IGC5J07B) Chart JN 138
Command Routine (IGC5407B) Chart JO 138
Options Routine (IGC5A07B) Chart JP 139
Delete 1 Routine (IGC5607B) Chart JQ 139
Delete 2 Routine (IGC5707B) Chart JR 139
Delete 3 Routine (IGC5807B) Chart JS 139
Delete 4 Routine (IGC5907B) Chart JT 140
Light Pen/Cursor Routine (IGC5F07B) Chart JU 140
PFK Routine 1 (IGC6A07B) Chart KL 140
PFK Routine 2 (IGC6B07B) Chart KM 140
Timer Interpreter (IGC5K07B) Chart JV 141
Status Display Interface 1 Routine (IGC6L07B) Chart KD 141
Status Display Interface 2 Routine (IGC6M07B) Chart KE 141
Status Display Interface 3 Routine (IGC6N07B) Chart KF 142
Status Display Interface 4 Routine (IGC6O07B) Chart KG 142
Status Display Interface 5 Routine (IGC6P07B) Chart KH 142
Status Display Interface 6 Routine (IGC6Q07B) Chart KI 142
Status Display Interface 7 (IGC6T07B) Chart KJ 142
Cleanup Routine (IEECVFTG) Chart KN 143

SPECIAL FEATURES 144
Extended Precision Floating Point Simulator Routines 144
SMF Routines 144

MODULE SYNOPSES 147
Interruption Supervision 147
Task Supervision 149
Task Termination 151
Contents Supervision 158
Main Storage Supervision 159
Communications Task 159
Communications Task with Multiple Console Support 162
Timer Supervision 165

DIDOCS 166

FLOWCHARTS 172

SECTION 4: MODULE DIRECTORY 365

SECTION 5: DATA AREAS 377
ABDUMP Parameter List 377
Boundary Box 378
Checkpoint Header Record (CHR) 379
Core Image Record (CIR) 379
Communication Vector Table 380
Data Set Descriptor Records 384
Resident Display Control Module (RDCM) 385
Transient Display Control Module (TDCM) 388
Entry Table (ENTAB) 394
Event Control Block 395
Free Queue Element (FQE) 396
Gotten Subtask Area Queue Element (GQE) 396
Interruption Queue Element (IQE) 396
Message Information List for Type-1 SVCs 397
Program Interruption Control Area (PICA) 397
Program Interruption Element (PIE) 398
Machine Check Record for SER0 and SER1 399
Channel Error Record for SER0 and SER1 400
Request Blocks 402
Reply Queue Element 406
Request Queue Element (RQE) 406
Segment Table (SEGTAB) 407
STAE Control Block (SCB) 408
STAE Parameter List 408
Storage Utilization Block 409
Supervisor Record (SUR) 410
SVC Table 411
Task Control Block 412
Timer Queue Element (TQE) 416
Time-Slice Control Element (TSCE) 417
Trace Table Record Format 418
Unit Control Module (IEECUCM) 419
UCM Prefix 419
MCS Prefix to UCM Base (MCS only) 419
Unit Control Module (UCM) Base 421
UCM Event Indication List (EIL) 422
UCM Entry Individual Device Map 423
UCM Message Text Area (Included with MCS and User Exit Only) 424
Write Queue Element (WQE) Single-Line 425
Write Queue Element (WQE) Major Non-MCS 427
Write Queue Element (WQE) Major MCS 428
Write Queue Element (WQE) Minor non-MCS 430
WRITE Queue Element (WQE) MINOR MCS 431
WTO/R Macro Expansion 432

SECTION 6: DIAGNOSTIC AIDS 433
Input to ABEND MODULES 433
Input to the ATTACH Module 434
Input to the CHAP Module 436
Input to the DELETE Module 436
Input to the DEQ Module 437
Input to the DOM Module 438
Input to the ENQ Module 439
Input to the EXTRACT Module 440
Input to FREEMAIN Module 441
Input to GETMAIN Module 442
Input to IDENTIFY Module 443
Input to LINK Module 444
Input to LOAD Module 445
Input to the POST Module 446
Input to the RETURN Module 446

Input to SAVE Module 446
Input to SNAP Module 447
Input to SPIE Module 448
Input to STAE Module 448
Input to STIMER Module 449
Input to TIME Module 450
Input to TTIMER Module 450
Input to WAIT Module 451
Input to WTO Module 452
Input to Multiple-Line WTO 454
Input to WTOR Module 456
Input to XCTL Module 458
ABEND Entry/Exit Table 459
ABDUMP Entry/Exit Table 461

INDEX 505

ILLUSTRATIONS

Figure 1. The Control Program Cycle 2
Figure 2. Handling a Request for Services 3
Figure 3. Main Storage during Execution 5
Figure 4. Main Storage with Hierarchy Support 7
Figure 5. System Task Control Blocks 8
Figure 6. Relationship of a TCB and Boundary Box to a Partition . . 9
Figure 7. The Dispatching Queue after System Generation 10
Figure 8. SVC Relocation Table 13
Figure 9. SVC Table Entries 13
Figure 10. Extended SVC Table Entries 13
Figure 11. Availability of Machine-Check Programs 16
Figure 12. Handling Shared and Exclusive Requests 22
Figure 13. Resource Queues 23
Figure 14. Scheduling Asynchronous Exit Routines 28
Figure 15. Control Block Queue after a Load and Link 36
Figure 16. Transient Area Request Queue and TCB/RCB Queue 38
Figure 17. System and Console Output Queues 43
Figure 18. Timer Clocks after IPL 46
Figure 19. Measuring Elapsed Time Intervals 47
Figure 20. The Nonexecutable Timer Module IEACVTPC 48
Figure 21. Position of Elements on the Timer Queue 48
Figure 22. Single Region Overlay Structure 53
Figure 23. Overlay Program Upward Branch 54
Figure 24. Branching to a Segment 56
Figure 25. Types of Overlay Supervision Processing 57
Figure 26. Processing of Segment Table Entries 57
Figure 27. Chain of ENTAB Entries Used to Branch to a Segment . . . 59
Figure 28. Problem Program Checkpoints 62
Figure 29. Dispatching Communications and Master Scheduler Tasks . . 73
Figure 30. Task Switching 74
Figure 31. Time Slice Control Element without Subtasking 76
Figure 32. Time Slice Control Element with Subtasking 76
Figure 33. Return Codes for the ENQ Routine 84
Figure 34. Error Conditions When use of a Resource is Signaled Complete 86
Figure 35. Determining if the Next Waiting Requester Should be Readied 87
Figure 36. BXLE Instruction 105
Figure 37. FINCH Processing 106
Figure 38. Queueing a Minor LPRB FOR Identify 110
Figure 39. Timer SLIH Modules 121
Figure 40. Timer SLIH Processing 122
Figure 41. Checkpoint Routine Control Flow 125
Figure 42. Restart Routine Control Flow 127
Figure 43. DIDOCS Processor Routine Entries 135
Figure 44. Example of TCT Pointers Used by EXCP Counting Routine . .146
Figure 45. Interruption Supervision Modules 365
Figure 46. Task Supervision Modules 365
Figure 47. Task Termination Modules 365
Figure 48. Contents Supervision Modules 366
Figure 49. Main Storage Supervision Modules 366
Figure 50. Communications Task Modules 366
Figure 51. Timer Supervision Modules 367
Figure 52. Overlay Supervision MOdules 367
Figure 53. Checkpoint/Restart Modules 367
Figure 54. DIDOCS Modules 368
Figure 55. Supervisor Module Cross Reference 369

Chart AA. SVC First-Level Interruption Handler and Type 1 SVC Exit .173
Chart AB. SVC Second-Level Interruption Handler 174
Chart AC. EXIT (Type 2, 3, and 4 SVC Routines) 176
Chart AD. Input/Output First-Level Interruption Handler and Timer/External First-Level Interruption Handler 178
Chart AE. Program-Check First-Level Interruption Handler and System Wait Time Collection Subroutine 179
Chart AF. Dispatcher 180
Chart AG. SER0 Routine 182
Chart AH. SER1 Routine 183
Chart AI. Extended SVC Router Routines 184
Chart BA. ATTACH with the Subtasking Option 185
Chart BB. DETACH 187
Chart BC. CHAP 188
Chart BD. EXTRACT without the Subtasking Option 189
Chart BE. EXTRACT with the Subtasking Option 190
Chart BF. WAIT 191
Chart BG. POST 192
Chart BH. ENQ 193
Chart BJ. DEQ 195
Chart BK. SPIE 197
Chart BL. STAE 198
Chart BM. Stage 1 Exit Effector 199
Chart BN. Stage 2 Exit Effector 200
Chart BO. Stage 3 Exit Effector 201
Chart CA. ABEND 0 202
Chart CB. ABEND G 203
Chart CC. ABEND 1 205
Chart CD. ABEND 2 and ABEND 3 206
Chart CE. ABEND 4 207
Chart CF. ABEND 5 208
Chart CG. ABEND 6 209
Chart CH. ABEND 7 and ABEND 8 210
Chart CI. ABEND 9 211
Chart CJ. ABEND 10 212
Chart CK. ABEND 11, ABEND 12, and ABEND 13 213
Chart CL. ABEND 14 214
Chart CM. ABEND 15 215
Chart CN. ABEND M 216
Chart CO. ABEND H 217
Chart CP. ABEND J 218
Chart CQ. ABEND/STAE Interface Routine 1 219
Chart CR. ABEND/STAE Interface Routine 2 220
Chart CS. ABEND/STAE Interface Routine 3 221
Chart CT. ABEND/STAE Interface Routine 4 222
Chart CU. Damage Assessment Routine 1 223
Chart CV. Damage Assessment Routine 2 224
Chart CW. Damage Assessment Routine 3 225
Chart CX. DAR 4 226
Chart CY. ABTERM 227
Chart DA. SVC DUMP 1 228
Chart DB. SVC DUMP 2 230
Chart DC. ABDUMP 1 231
Chart DD. ABDUMP 2 and ABDUMP 3 232
Chart DE. ABDUMP 4 233
Chart DF. ABDUMP 5 and ABDUMP 6 234
Chart DG. ABDUMP 7 and ABDUMP 8 235
Chart DH. ABDUMP 9, ABDUMP 10, and ABDUMP 11 236
Chart DI. ABDUMP 12 237
Chart DJ. ABDUMP 13 238
Chart DK. ABDUMP 14 239
Chart DL. TCAM: ABDUMP 1 and ABDUMP 2 240

Chart DM. TCAM: ABDUMP 3 and ABDUMP 4 241
Chart EA. ATTACH without Subtasking 242
Chart EB. LINK, LOAD, XCTL 243
Chart EC. FINCH: Processing for LINK, LOAD, ATTACH, and Nontransient Area XCTL 246
Chart ED. FINCH: Processing for SVC and I/O Transient Area Requests 247
Chart ED. Transient Area Loading Task within FINCH 249
Chart EE. IDENTIFY 250
Chart EF. DELETE 251
Chart FA. GETMAIN/FREEMAIN 252
Chart GA. Communications Task Initialization 255
Chart GB. Communications Task External Interruption and Attention Handlers 256
Chart GC. Communications Task Write-to-Operator without MCS 257
Chart GD. Multiple-Line Write-to-Operator (MLWTO) 258
Chart GE. Communications Task Write-to-Operator with Reply 262
Chart GF. Communications Task Wait (non-MCS) 263
Chart GG. Communications Task Router (non-MCS) 264
Chart GH. Communications Task Console Switch (non-MCS) 265
Chart GI. Communications Task Router (MCS) 266
Chart GJ. Communications Task Console Switch Load 1 (MCS) 267
Chart GK. Communications Task Device Interface (MCS) 271
Chart GL. Communications Task WTO/R Processor (MCS) 274
Chart GM. Communications Task Delete Operator Message (MCS) 277
Chart GN. Communications Task NIP Message Buffer Writer 278
Chart GO. Communications Task 1052 Processor (non-MCS) 279
Chart GP. Communications Task 1052 Processor Load 1 (MCS) 281
Chart GQ. Communications Task 1052 Open/Close 283
Chart GR. Communications Task Card Reader Processor (non-MCS) . . .284
Chart GS. Communications Task Card Reader Processor (MCS) 285
Chart GT. Communications Task 2740 Processor (MCS only) 286
Chart GU. Communications Task Log Writer 288
Chart GV. Communications Task LOG, Write-to-Log (Load 1) 289
Chart GW. Communications Task LOG, Write-to-Log (Load 2) 290
Chart GX. 1052 Processor Load 2 291
Chart GY. Graphic Console Initialization 293
Chart HA. TIME 294
Chart HB. Timer Second-Level Interruption Handler 295
Chart HC. TTIMER 297
Chart HD. STIMER 298
Chart HE. TIME with Time-of-Day Clock 299
Chart HF. TTIMER with Time-of-Day Clock 300
Chart HG. STIMER with Time-of-Day Clock 301
Chart HH. Timer Second-Level Interruption Handler with Time-of-Day Clock 302
Chart IA. Overlay Supervision 303
Chart JA. DIDOCS Processor 1, Load 1 304
Chart JB. DIDOCS Processor 1, Load 2 306
Chart JC. DIDOCS 2250 I/O 1 307
Chart JD. DIDOCS 2250 I/O 2 309
Chart JE. DIDOCS 2260 I/O 1 310
Chart JF. DIDOCS 2260 I/O 2 311
Chart JG. DIDOCS Model 85 I/O 312
Chart JH. DIDOCS Message 1 313
Chart JI. DIDOCS Message 2 314
Chart JJ. DIDOCS Message 3 316
Chart JK. DIDOCS Display 1 317
Chart JL. DIDOCS Display 2 319
Chart JM. DIDOCS Display 3 321
Chart JN. DIDOCS Roll Mode 323
Chart JO. DIDOCS Command 324
Chart JP. DIDOCS Options 325
Chart JQ. DIDOCS Delete 1 327
Chart JR. DIDOCS Delete 2 329
Chart JS. DIDOCS Delete 3 330
Chart JT. DIDOCS Delete 4 332
Chart JU. DIDOCS Light Pen/Cursor 333
Chart JV. DIDOCS Timer/Interpreter 334

Chart KA. DIDOCS Processor 0 Load 1 336
Chart KB. DIDOCS Processor 0 Load 2 338
Chart KC. DIDOCS Asynchronous Error 340
Chart KD. DIDOCS Status Display 1 342
Chart KE. DIDOCS Status Display 2 343
Chart KF. DIDOCS Status Display 3 345
Chart KG. DIDOCS Status Display 4 347
Chart KH. DIDOCS Status Display 5 348
Chart KI. DIDOCS Status Display 6 351
Chart KJ. DIDOCS Status Display 7 353
Chart KK. DIDOCS Open/Close 354
Chart KL. DIDOCS PFK 1 355
Chart KM. DIDOCS PFK 2 357
Chart KN. DIDOCS Cleanup 358
Chart LA. Checkpoint Modules 360
Chart LB. Restart Modules 361
Chart MA. System Management Facility EXCP Counter 362
Chart MB. System Management Facility Time Limit Expiration 364

Operation Diagram 01. The MFT Supervisor 463
Operation Diagram 02. Interruption Supervision 465
Operation Diagram 03. SVC FLIS and Type-1 Exit 467
Operation Diagram 04. SVC SLIH 469
Operation Diagram 05. EXIT 471
Operation Diagram 06. Timer/External FLIH 473
Operation Diagram 07. Input/output FLIH 475
Operation Diagram 08. Program Check FLIH 477
Operation Diagram 09. Dispatcher 479
Operation Diagram 10. Attaching a Subtask (Part 1) 481
Operation Diagram 11. Attaching a Subtask (Part 2) 483
Operation Diagram 12. Creating and Changing Dispatching Priorities .485
Operation Diagram 13. Processing for a Task Asynchronous Exit . . .487
Operation Diagram 14. Console Communications without MCS: Input . .489
Operation Diagram 15. Console Communications without MCS: Output .491
Operation Diagram 16. Communications Task with Multiple Console Support 493
Operation Diagram 17. Log Function 495
Operation Diagram 18. Timer SVC Interruption Handling 497
Operation Diagram 19. Timer Interruption Handling 499
Operation Diagram 20. Functional Flow of Overlay Supervision 501
Operation Diagram 21. 1052 and 2740 Console Support Routines with MCS 503

This section describes the purpose and function of the MFT Supervisor, its relationship to the control program, its operational environment, and its physical characteristics.

THE RELATIONSHIP OF THE MFT SUPERVISOR TO THE CONTROL PROGRAM

The MFT Supervisor is one part of the control program with MFT. The MFT Supervisor (referred to as "the supervisor") monitors each unit of work to be done in the operating system. Two additional parts of the operating system are job management, which accepts and schedules jobs in a continuous flow, and data management, which stores, maintains, and retrieves all data. Figure 1 illustrates how task management (the supervisor) works in conjunction with data and job management to process a program.

For additional information about the control program with MFT, refer to Job Management with MFT.

PURPOSE AND FUNCTION OF THE MFT SUPERVISOR

Job steps (tasks within the operating system) are executed under the control of the supervisor, which allocates needed resources on the basis of priorities. The supervisor assigns the resources to perform tasks, keeps track of all such assignments, and ensures that the resources are freed upon task completion. If one resource is needed by several tasks, queuing of requests may be required. The supervisor thus maintains control of resources that can be shared. This enables more efficient use of the central processing unit, main storage, system and user programs, and the interval timer.

The activity of the supervisor may be divided into two major areas: Interruption Supervision and Resource Handling.

Interruption Supervision

Supervisor activity generally begins with an interruption. In IBM System/360 and System/370, the interruption is a machine characteristic. It is the means by which the supervisor gains control of the central processing unit to provide resources and services for the performance of a task.

There are five types of interruptions:

- Supervisor call (SVC) interruption: a request for a particular supervisor service.
- Timer/external interruption: an attention signal from the interval timer, the console interruption key, or the direct control feature.
- Input/output interruption: a signal that an input/output event has occurred
- Program interruption: a signal that a program has attempted an invalid action
- Machine-check interruption: a signal that a machine error has occurred.

An interruption-handling program of the supervisor analyzes each interruption using control information available to it at the time of the interruption. Each of the five interruption types has associated with it two program status words (PSWs). The Old PSW contains the information needed by the supervisor to analyze the interruption. The New PSW contains the address of the appropriate interruption-handling routine.

When an interruption occurs, the CPU stores the contents of the current PSW in the Old PSW for that type of interruption and loads the New PSW. By loading the New PSW, the CPU places itself in supervisor state and passes control to the interruption-handling portion of the supervisor. The supervisor then passes control to those parts of the control program that handle the resources needed to service the interruption.

The exiting and dispatching routines of interruption supervision return control from an interruption-handling routine to the interrupted program. The exiting routines determine what type of program has completed execution and perform the clean-up operations (such as resetting control tables) before control is returned to the caller. The dispatching procedure is used to return control to the ready task with the highest priority.

Resource Handling

Figure 2 illustrates how the interruption-handling portion of the supervisor interacts with the resource-handling portion to service an interruption. The resource-handling portion of the supervisor

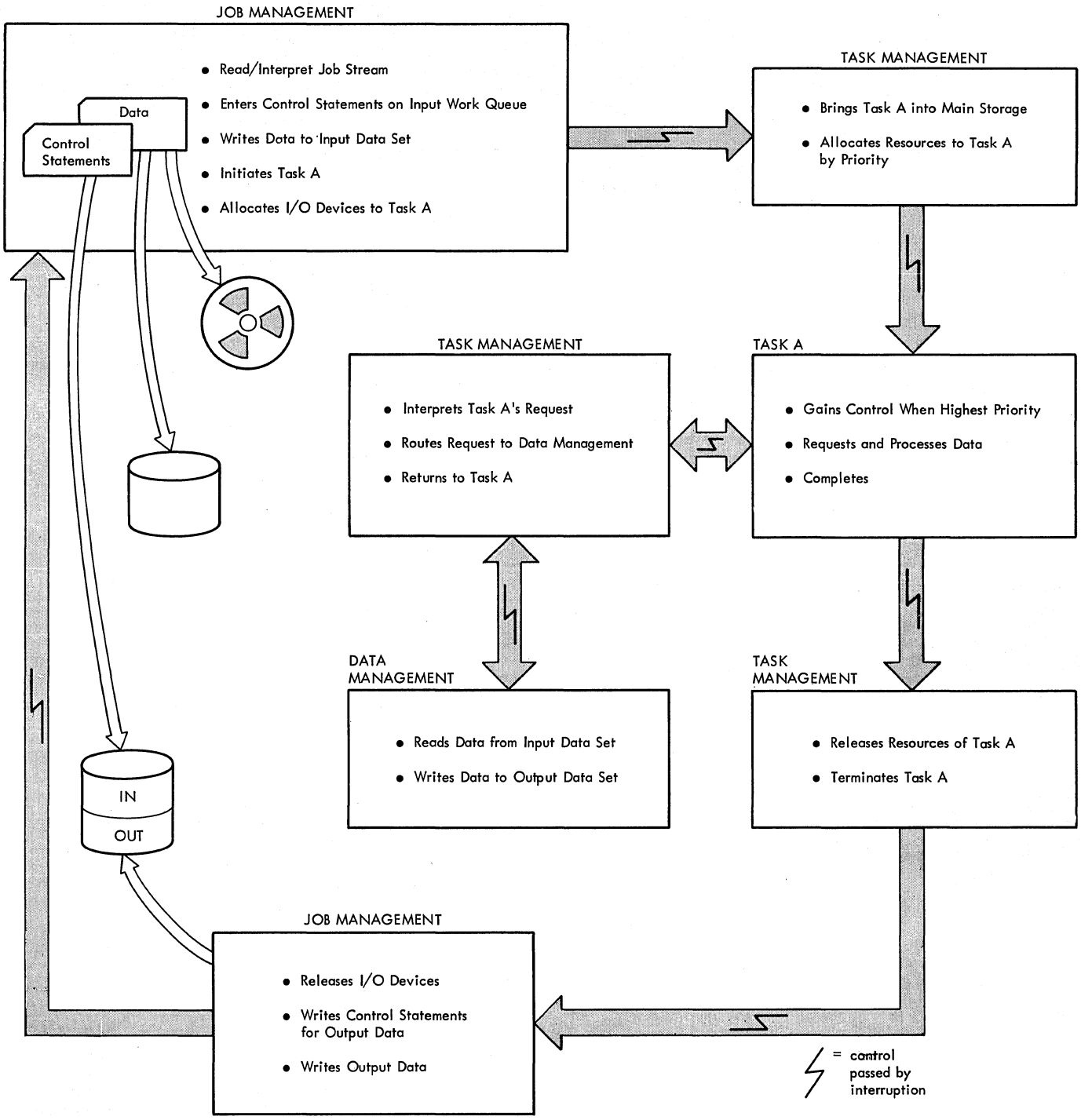


Figure 1. The Control Program Cycle

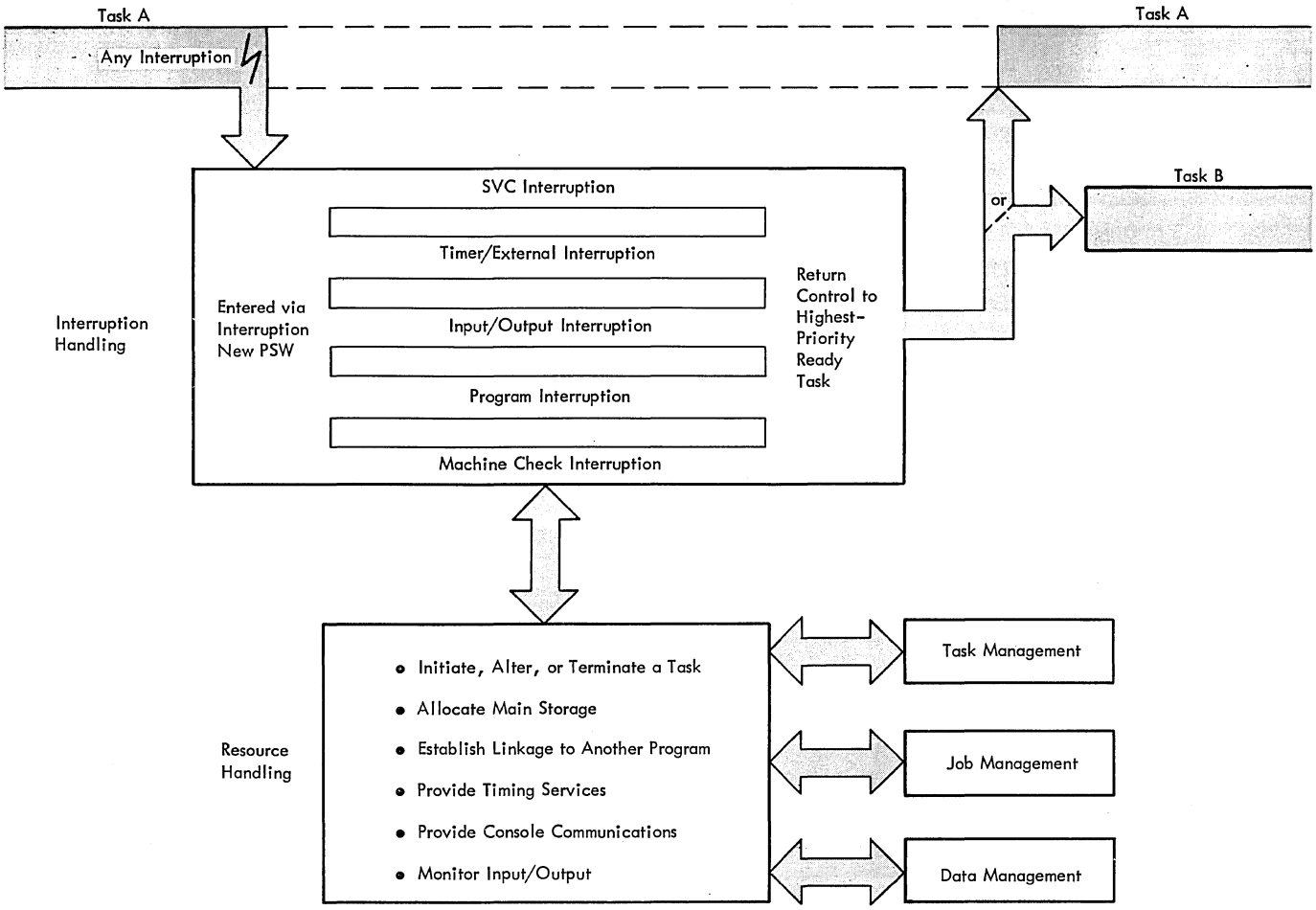


Figure 2. Handling a Request for Services

is comprised of system programs that may be grouped into the following functional catagories:

- Task Supervision. If the subtasking option was selected, the supervisor creates a task in response to a request to attach a subtask to an already existing task. The order in which tasks gain control of system resources (priority) is also affected by task supervision whether or not the system has subtasking.
- Task Termination. The supervisor processes the normal and abnormal termination of tasks.
- Contents Supervision. The supervisor keeps records of and provides linkage between all programs in main storage. Contents supervision also brings requested programs into main storage through linkage with Data Management.
- Main Storage Supervision. The supervisor assigns and releases main storage needed to perform tasks.
- Communications Task. The supervisor allows the operator to communicate directly with the system and a program to write a console message to the operator.
- Timer Supervision. The supervisor controls the use of the System/360 interval timer and the System/370 time-of-day clock.
- Recovery Management. The recovery management routines, which are optional, record information about a machine malfunction. The Machine-Check Handler records environmental data and attempts to analyze the malfunction and restore the system to normal operation.
- Checkpoint/Restart. On request, the supervisor writes records of a task's main storage along with control information so that the task can be restarted from that point at a later time.

After a control program service has been performed, the supervisor determines what

task is to be performed next, as seen in Figure 2.

TRACING FACILITIES

Two facilities, the Trace Table and the Generalized Trace Facility (GTF), are provided to assist in tracing program flow by monitoring and recording system events.

Trace Table

The Trace Table facility is an optional feature specified during system generation. The Trace Table routines place entries, each of which is associated with a certain type of event, in a trace table. The size of the table is also a system generation option; when the table is filled, the routine overlays old entries with new entries beginning at the top of the table. Trace table entries are formatted and printed out on SNAP dumps and stand-alone dumps.

Generalized Trace Facility

The Generalized Trace Facility (GTF) is initiated by the operator using the START command and executes as a problem program in its own partition. When GTF is started, the operator selects specific events to be traced and may select to record the trace data either in main storage or on an external device. When the internal storage option is selected, the recorded data is comparable to that provided by the optional Trace Table. When the external device storage option is selected, the recorded data is more comprehensive. When GTF is active, the optional Trace Table facility, if present, is disabled. If the trace records are maintained in main storage and, a SYSABEND DD statement is included, ABDUMP provides formatted trace data for abnormally terminated users. When trace records are stored on an external device, a trace EDIT function of IMDPRDMP will provide the output of selected data.

SPECIAL FEATURES

The following facilities are optional features of the supervisor that do not fall within the general functional areas of the supervisor.

Time Slicing

When the system is generated, the user selects a group of contiguous partitions for time-slicing. The dispatching and exiting portions of the supervisor recognize the dispatching priorities assigned to the time-slice group through a queue of time-sliced tasks. Time-slicing occurs only among the tasks in the group and only when the priority level of the group is dispatched for the specified time slice. The dispatching of tasks within the group continues until all the tasks are waiting, or a task with a higher priority than that of the group becomes ready. If the system has the subtasking option, tasks may be added to or removed from the time-slice group by changing the dispatching priorities of the tasks.

Main Storage Hierarchy Support

Main storage may be expanded by including IBM 2361 Core Storage in the system. Main Storage Hierarchy Support for IBM 2361, Models 1 and 2, is a control program option that permits selective access to either the processor storage (hierarchy 0) or 2361 Core Storage (hierarchy 1).

Extended Precision Floating-Point Decimal Simulator

This option allows all System/360 and System/370 CPUs that do not support or only partially support the extended precision floating-point instructions to simulate hardware processing when one of these eight instructions is encountered. The modules that support this function are accessed when one of these instructions causes a program check and is processed by an exit routine specified in a SPIE macro instruction.

System Management Facilities

System Management Facilities (SMF) is an optional feature of the control program which can be selected at system generation. It gathers and records information used to evaluate system usage. SMF functions are performed by both job management and supervisor routines.

Shared Direct Access Device

The Shared Direct Access Device (Shared DASD) feature enables independently operating data processing systems to share direct access storage devices. The two-channel switch and its control commands, device reserve and device release, control sharing of direct access storage between systems. The shared DASD feature provides the control program functions needed to control device reservation and release.

OPERATING ENVIRONMENT

In an operating system with MFT, main storage is divided in two major areas: the fixed area and the dynamic area. The fixed

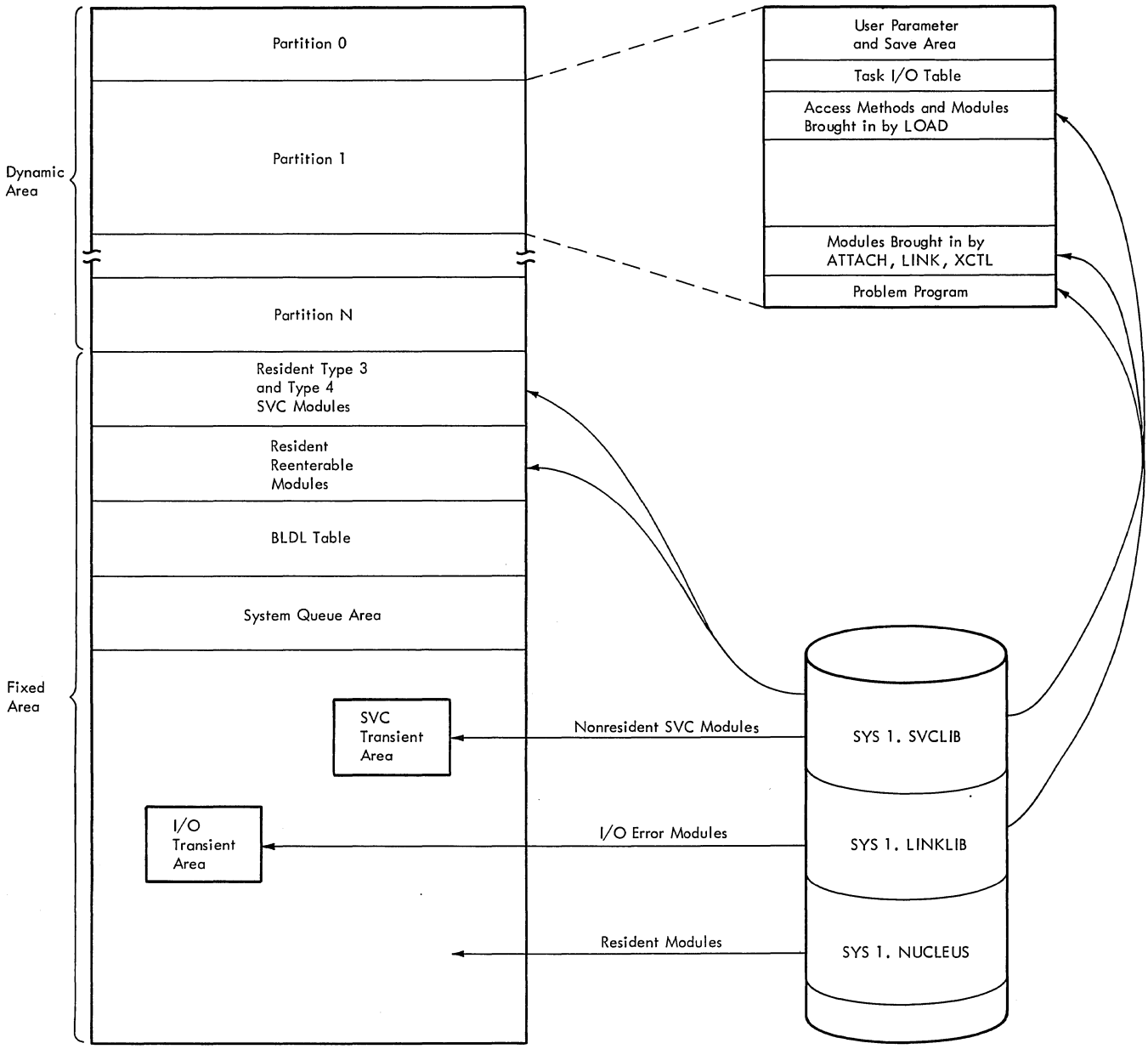


Figure 3. Main Storage during Execution

area and the dynamic area are subdivided as shown in Figure 3. The fixed area, located in the lower portion of main storage, contains the resident portion of the supervisor, and control blocks and tables used by the system. The size of the fixed area depends on the number of partitions established by the user and the supervisor options selected during system generation. The dynamic area, located in the upper portion of main storage, is divided into partitions. Each partition may be occupied by one or more processing programs, or by job, task, and data management routines when performing a service for a user task.

Fixed Area

In MFT, the fixed area is that part of main storage into which the nucleus is loaded at IPL. The storage protection key of the fixed area is 0 so that its contents can be modified only by the supervisor. The fixed area also contains the partitions for the communications task, the master scheduler task, and the two transient areas into which certain nonresident routines are loaded when needed: the SVC transient area (1024 bytes) and the I/O supervisor transient area (1024 bytes). Each transient area contains only one routine at a time.

When a nonresident SVC or error-handling routine is required, it is read into the appropriate transient area. The transient routines operate with a protection key of 0, as do other routines in the fixed area.

System Queue Area

The system queue area (SQA) is established by the Nucleus Initialization Program in the fixed area and provides the main storage required for WTOR emergency buffers, subtask TCBs, queue control blocks, and queue elements built by the control program. The SQA must be at least 1600 bytes for a minimum two-partition system. Its storage protection key is 0 so that it can be modified by control program routines only.

Dynamic Area

Figure 3 also shows how the contents of each partition in the dynamic area are organized and how they are related to the rest of main storage. Job management routines, processing programs, and routines brought into storage by a LINK, ATTACH, or XCTL macro instruction, are loaded at the lowest available address. The highest portion of the partition is occupied by the user's parameter area and user's save area. The next portion of the partition is occupied by the task input/output table (TIOT), which is built by job management. This table is used by data management routines and contains information about DD statements.

Each partition may be used for a problem program as well as for system tasks. When the supervisor requires main storage to build control blocks or work areas, it obtains this space from the partition of the processing program that requested the space. Access method routines and routines brought into storage by a LOAD macro instruction are placed in the highest available locations below the task input/output table.

Resident Portion of the Supervisor

The resident portion (nucleus) of the supervisor resides in SYS1.NUCLEUS. It is made up of those routines, control blocks, and tables that are brought into main storage at initial program loading (IPL) and are never overlaid by another part of the operating system. The nucleus is loaded into the fixed area of main storage.

The resident task management routines include all of the routines that perform:

- Interruption supervision
- Main storage supervision

They also include portions of the routines that perform:

- Task supervision
- Contents supervision
- Timer supervision
- Overlay supervision

The resident job management routines include several of the communications task routines. The MFT communications task is described in this publication.

The resident data management routines are the input/output supervisor and, optionally, the BLDL routines of the partitioned access method. These routines are described in the Input/Output Supervisor PLM, and the Sequential Access Method PLM.

The user may also select resident reenterable routines, which are access method routines from SYS1.SVCLIB, and other reenterable routines from SYS1.LINKLIB. At system generation, the user specifies that he wants such routines resident. At IPL, he identifies the specific routines desired in the RAM= and RSVC= entries. The selected routines are loaded during system initialization and reside adjacent to the higher end of the system queue area unless the BLDL table is also resident (see Figure 3).

Types 3 and 4 SVC routines can also be made resident through the RSVC option. NIP loads these routines adjacent to the higher end of the resident reenterable routines. If there is no resident BLDL table or resident reenterable routines, the routines are loaded adjacent to the higher end of the system queue area.

Nonresident Portion of the Supervisor

The nonresident portion of the control program comprises routines that are loaded into main storage as they are needed and can be overlaid after their completion. The nonresident routines operate in the partitions and in two sections of the nucleus called transient areas.

Expanding the Size of Main Storage

Main storage may be expanded by including IBM 2361 Core Storage (core storage) units in the system. Main Storage Hierarchy Support for IBM 2361 Models 1 and 2 permits access to either processor storage (hierarchy 0) or core storage (hierarchy 1). Each partition established during system generation is described by a boundary box (see Figure 4). The first half of the

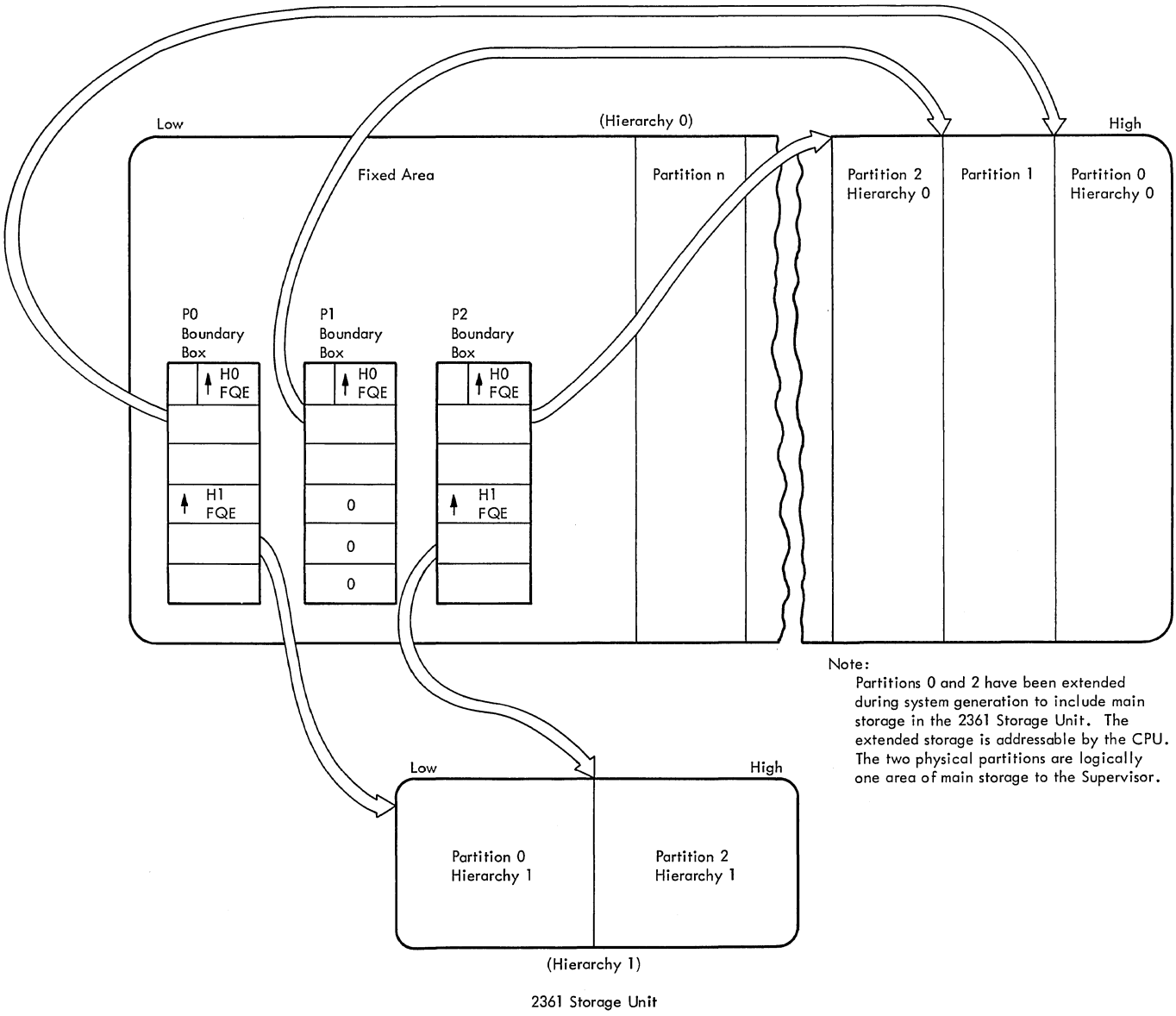


Figure 4. Main Storage with Hierarchy Support

boundary box describes the processor storage partition, and the second half describes the core storage partition. Pointers in the boundary box for an unassigned partition are set to 0. If a partition is established entirely within hierarchy 1, the processor storage pointers in the first half of the partition's boundary box are set to 0. If a partition segment is not generated in core storage, the core storage pointers in the second half of the partition's boundary box are set to 0. If core storage has been included in the system, but is offline, the second half of the boundary box will contain 0's. If core storage is excluded from the system, the second half of the boundary box is not generated.

LOGICAL CHARACTERISTICS

Each task in the system is represented by a task control block (TCB). The TCB contains control information related to the task and pointers to system resources assigned to perform the task.

When the operating system is generated, TCBs are built in the nucleus for each partition and for each system task (see Figure 5). The system tasks are: the master scheduler task of job management, the communications task, and the transient area loading task. If specified, TCBs are also created for the following optional system tasks: the system error task, the Multiple Console Support (MCS) write-to-log task,

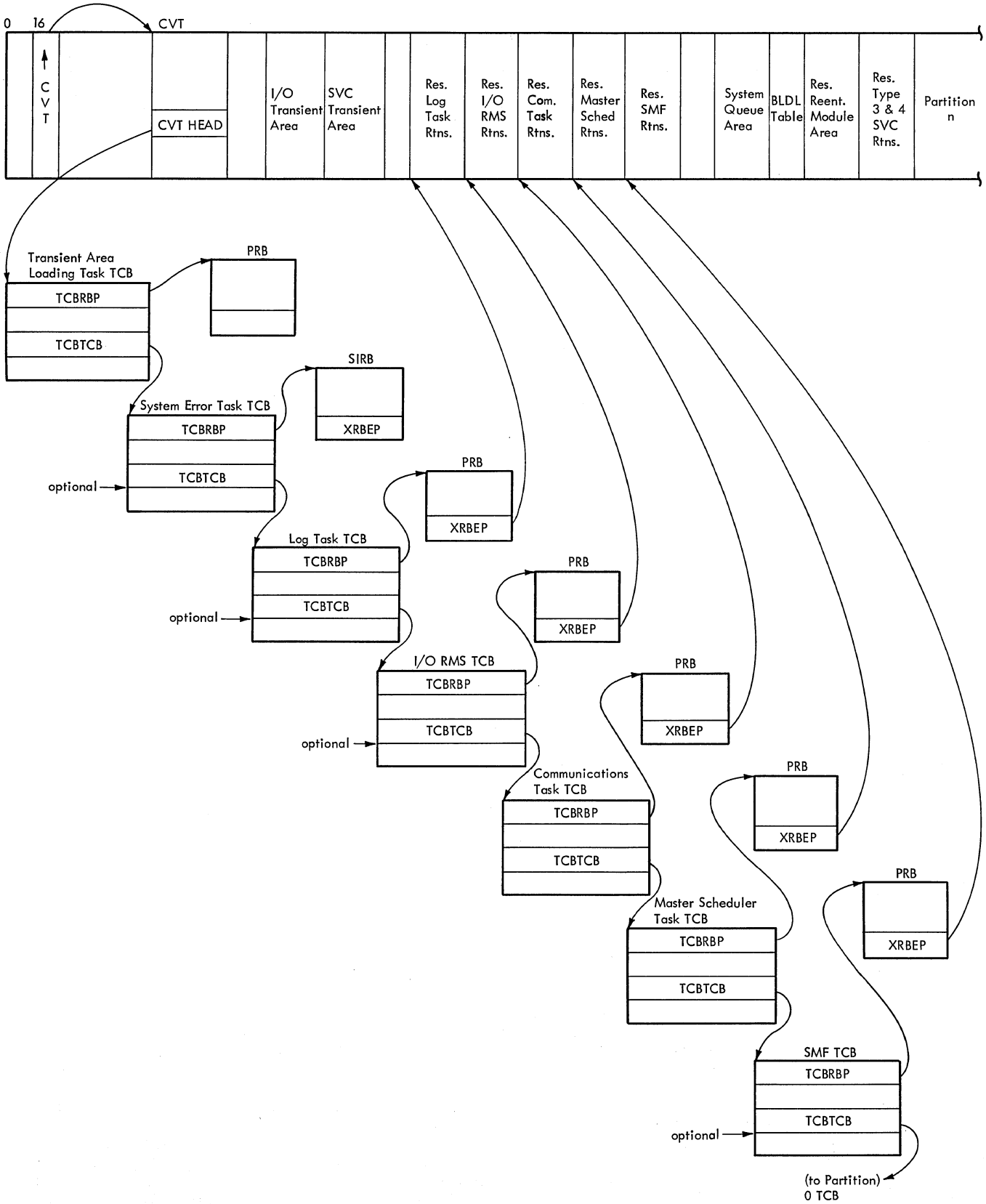


Figure 5. System Task Control Blocks

the I/O Recovery Management Support task (I/O RMS), and the System Management Facility task (SMF). If the subtasking option is selected during system generation, additional task control blocks may be constructed by the supervisor Attach routine in the system queue area, whenever a user program requests the creation of subtasks.

All the TCBs in the system are chained together according to dispatching priority to form the TCB queue. The relative dispatching priority of system tasks and problem program tasks is established when the system is generated. The system tasks are assigned higher dispatching priorities than the problem program tasks. The problem program tasks are assigned priorities directly corresponding to the location of the partition in main storage. Partition 0, located at the highest address of main storage is assigned the highest priority, while partition n, located at the lowest address, is assigned the lowest priority. Figure 6 shows the relationship of a TCB and boundary box to a partition, and Figure 7 illustrates the dispatching queue established during system generation.

SUMMARY

The MFT supervisor comprises those resident and nonresident modules that receive and service interruptions. These services include supervisor-assisted linkage between modules, management of main storage and transient areas, priority dispatching of ready tasks, and normal and abnormal termination of tasks. The supervisor modules are described in this publication by functional area as shown in Diagram 1.

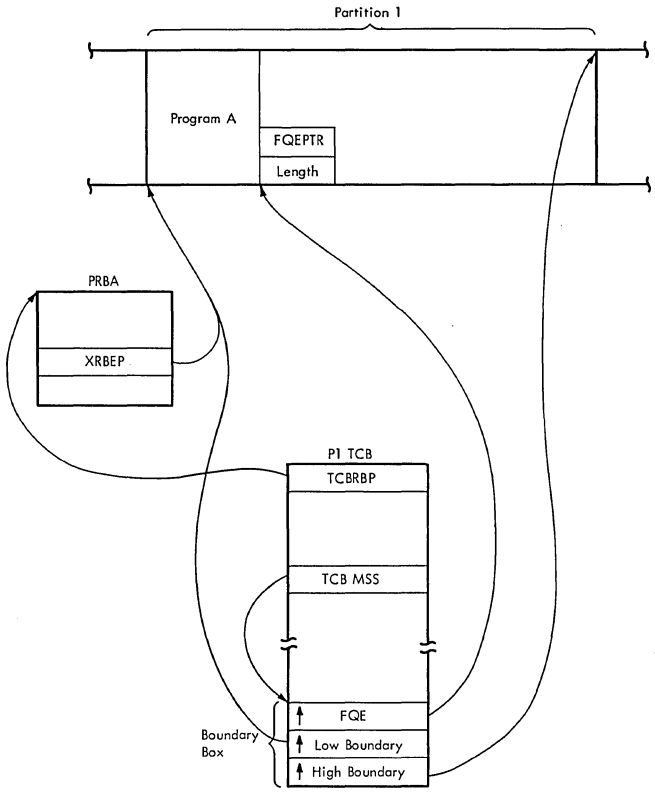


Figure 6. Relationship of a TCB and Boundary Box to a Partition

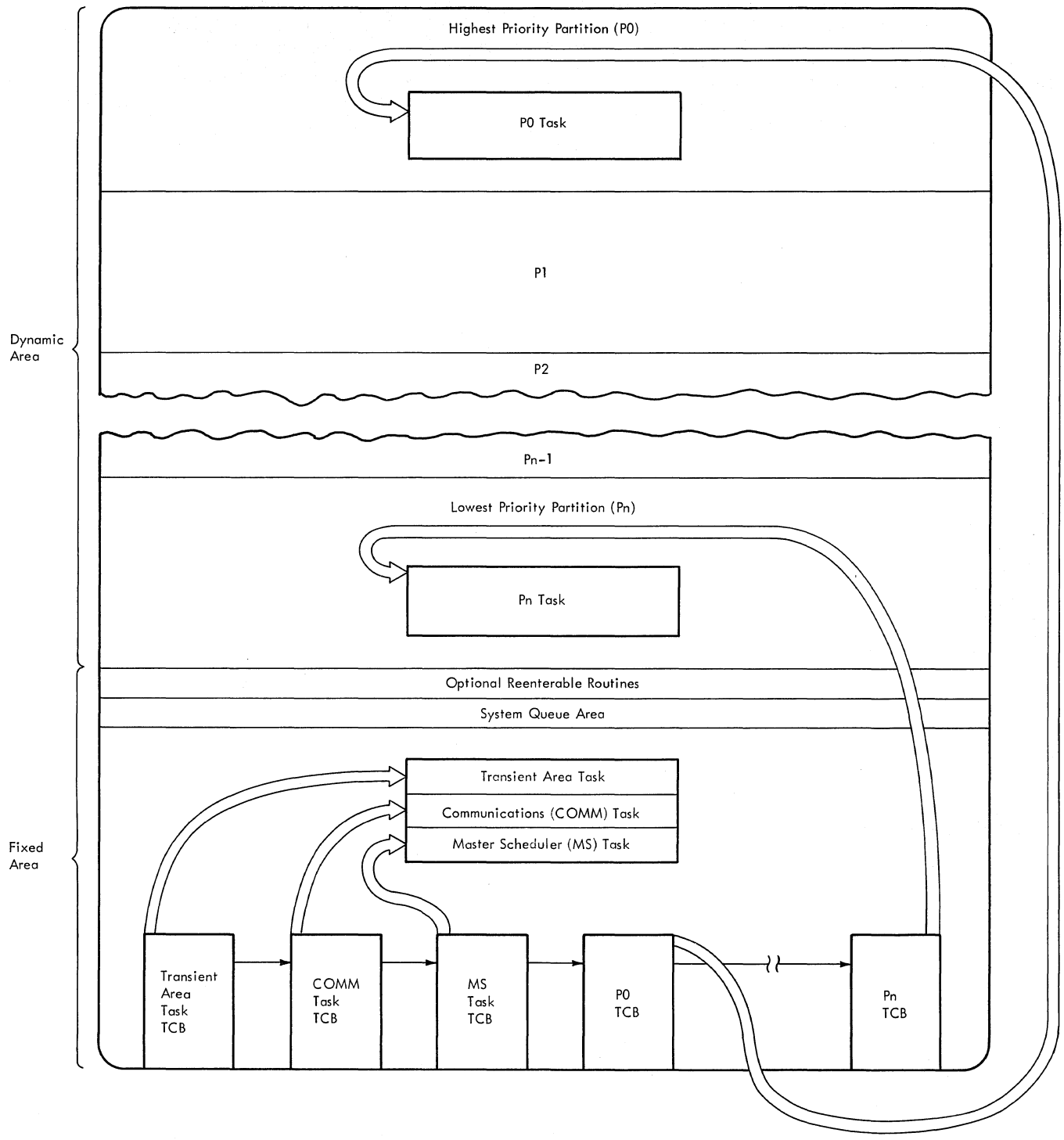


Figure 7. The Dispatching Queue after System Generation

This section describes the operation of the MFT Supervisor. Emphasis is placed on describing how the functional units and data areas interact in servicing requests. Cross-references to Section 3, "Program Organization," relate the logic descriptions in this section to the physical descriptions, flowcharts, and tables contained there. Operation diagrams in Section 3 illustrate the flow of control and transmission of data.

INTERRUPTION SUPERVISION

Interruption supervision provides initial processing of all interruptions; that is, control passes from the interrupted program to the interruption supervision routines and, in turn, to any additional routines needed to complete this processing (see Diagram 2). The interruption supervision routines:

- Save the interrupted environment so that it can be restored before returning control to the interrupted program.
- Isolate interruptions from each other.
- Pass control to routines that service the interruption.
- Return control to the interrupted program or to the highest priority ready task when servicing is complete.

PASSING CONTROL TO THE INTERRUPTION HANDLING PORTION OF THE SUPERVISOR

The program status word (PSW) contains information required for proper program execution. The current PSW, which is maintained by system hardware, always contains the address of the next instruction to be executed. (For a detailed description of the program status word, see Principles of Operation.) There are two permanent PSW areas in main storage for each of the five types of interruptions (see "Types of Interruptions" below). Each New PSW is set by the Nucleus Initialization Program to point to the first-level interruption handler for that type of interruption. Each Old PSW is used as a save area for the current PSW when that particular type of interruption occurs. This save area will contain the address of the next sequential instruction that would have been executed if that type of interruption had not occurred. Thus, when an interruption occurs:

1. The current PSW is saved in the Old PSW area associated with that type of interruption. The address in the PSW may be used to return control to the interrupted program.
2. The New PSW associated with that type of interruption is loaded from its main-storage location and becomes the current PSW. This New PSW contains the address of the interruption handling routine that services this type of interruption.

TYPES OF INTERRUPTIONS

There are five types of interruptions:

- Supervisor Call (SVC)
- Timer/External (T/E)
- Input/Output (I/O)
- Program Check
- Machine Check

There is one important difference between supervisor call (SVC) interruptions and the other four types of interruptions. An SVC interruption occurs when a problem program or system task requests supervisor services by using an SVC instruction or by using a macro instruction that expands to include an SVC instruction. Thus, an SVC interruption is a planned interruption, as opposed to the other four types which are unplanned (asynchronous).

SUPERVISOR CALL (SVC) ROUTINES

Interruption supervision provides all interface operations associated with the four types of SVC routines:

- Type 1 SVC routines: These routines are always resident and are executed disabled. They usually return control to the interrupted program without entering the Dispatcher. A type 1 SVC routine cannot use another SVC routine (except by a branch entry to a resident routine) because other type SVC routines operate either partially or fully enabled. Examples of a type 1 routine are GETMAIN, FREEMAIN, EXCP, WAIT and EXIT.

Note: A Type-1 SVC, Write-to-Programmer interface facility is provided to issue WTP messages for Type-1 SVC routines during abnormal termination processing (see Abnormal Termination in Section II).

- Type 2 SVC routines: These routines are also resident; but they are partially enabled, or call other than type 1 SVC routines. Examples are LINK, LOAD, and XCTL.

- Type 3 SVC routines: These routines are like type 2 routines except that they are not resident (these may be made resident; see "Resident Type 3 and 4 SVC Routine Option" below). They are each brought into the 1024-byte transient area by Program Fetch. Examples are IDENTIFY and WTO.

- Type 4 SVC routines: These are "multiphase" type 3 routines. They are too large to be brought into the transient area in one package, and therefore must be loaded in phases, each one overlaying a preceding one. Transfer of control between phases is through XCTL. Examples are OPEN, CLOSE, and ABEND.

INTERRUPTION SUPERVISION ROUTINES

Except for a machine-check interruption, each interruption causes control to be passed to a first-level interruption handler (FLIH) which prepares for additional processing by other supervisor or user-supplied routines. A machine check interruption causes control to be passed directly to a machine recording or recovery routine. The linkage is similar to that used to access the FLIH routines in that the machine check New PSW contains the address of the recording or recovery routine that is to process the interruption. Following is a summary of the routines that handle the other 4 types of interruptions. For a more detailed description of each routine, see Section 3: Program Organization.

- Supervisor Call (SVC) Interruption Routines
  SVC FLIH -- For a Type 1 SVC request, passes control to the appropriate service routine. For a Type 2, 3, or 4 request, it passes control to the SVC second-level interruption handler (SLIH) for further processing.
  SVC SLIH -- Monitors the SVC transient area and prepares for Type 2, 3, and 4 requests.
  Type 1 SVC Exit -- Provides return linkage from a Type 1 SVC routine to the interrupted program or to the Dispatcher.
  EXIT -- Provides return linkage from a Type 2, 3, or 4 SVC routine to the interrupted program or to the Dispatcher.

- Timer/External Interruption Routine
  T/E FLIH -- Prepares for and passes control to the Job Management External Interruption Handler or the Timer Second-Level Interruption Handler.

- Input/Output Interruption Routine
  I/O FLIH -- Prepares for and passes control to the I/O Supervisor to process the interruption.

- Program Check Interruption Routine
  Program Check FLIH -- Prepares for and passes control to a user-supplied program interruption routine, or to ABTERM to schedule abnormal termination of the interrupted routine.

- Common Routines
  Validity Check routine -- Provides address validity checking for other supervisor routines.
  Dispatcher -- Provides return linkage from supervisor routines to the highest priority task in the system that is ready to begin or resume execution

SVC CONTROL INFORMATION

The supervisor maintains SVC control information in the SVC Table and the Relocation Table. These tables are in module IEASVC00, which is assembled at system generation.

The Relocation Table (see Figure 8) is used to relate the SVC code number to its corresponding entry in the SVC table. The table consists of a number of 1-byte entries, each of which is addressed using the SVC code number as an index. If the entry is 0, the corresponding SVC code is invalid. If the entry is not 0, the number in the entry is an index to an entry in the SVC Table. The first section of the Relocation Table contains entries for IBM codes (codes assigned to IBM-provided SVC routines). The second section contains entries for user codes (codes assigned to user-provided SVC routines). The initial entry in the table is for SVC 0 and progresses through the highest IBM code, whether or not it is in use in the system. The first user code in the second section is 255 and progresses toward the lowest user code.

The SVC Table is also divided into two sections (see Figure 9). The first section contains a 3-byte entry for each type 1 or type 2 SVC routine. The second section contains a 1-byte entry for each type 3 or

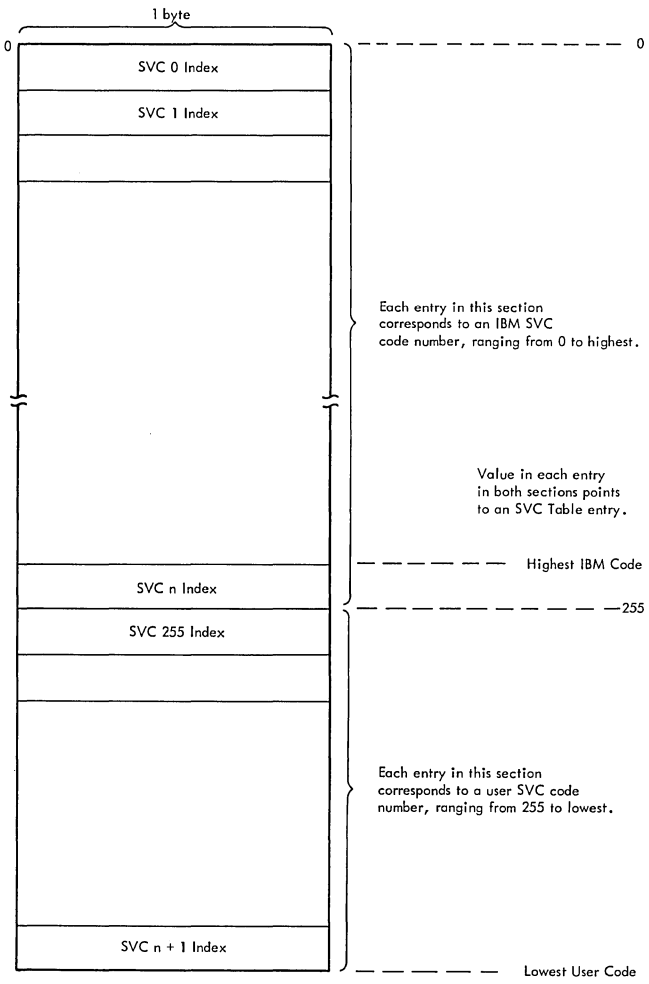


Figure 8. SVC Relocation Table

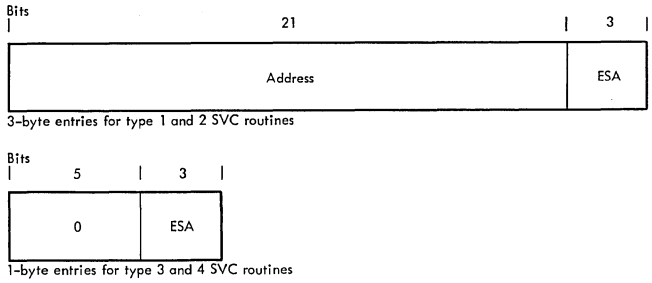


Figure 9. SVC Table Entries

type 4 SVC routine. Each 3-byte entry contains a 24-bit main-storage address of an SVC routine. The three low-order bits of this address indicate the number of doublewords required for the extended save area in the request block. These bits will be set to 0 for a type 1 SVC. Each 1-byte entry in the table contains the doubleword length of the extended save area in the last three bits.

The SVC Table may be extended during system generation so that each entry is 4 bytes long. The 3-byte entries have the same format except that the address is preceded by eight bits set to 0. Each 1-byte entry is replaced by track address (10 bits), the record number (8 bits), the length of the first text record (11 bits), and the extended save area information (3 bits). The format of an extended SVC Table is shown in Figure 10.

RESIDENT TYPE 3 AND 4 SVC ROUTINE OPTION

During system generation, the user may specify that certain type 3 and type 4 SVC routines be made resident so that frequently used SVC routines need not be brought into the SVC transient area each time they are used (the Extended SVC Table Option must also be selected). A resident type 3 or type 4 SVC routine has the characteristics of a type 2 SVC routine.

The following differences in operation result when the resident option and the Extended SVC Table option are selected during system generation:

1. When the Nucleus Initialization Program makes a type 3 or 4 SVC routine resident, the routine's entry in the SVC Table is changed. The track address, record number, and length fields are overlaid by X'FF' and the entry point address of the routine. The X'FF' (a number larger than any track address) will indicate to the SVC FLIH that the entry is for a resident type 3 or 4 SVC routine.

2. The SVC entry procedure for a resident type 3 or 4 routine is similar to that for a type 2 routine. The use of FINCH and Program Fetch is not required because the routine is not loaded into the transient area.

3. The SVC exiting procedure does not require the services of the transient area refresh subroutine (within the

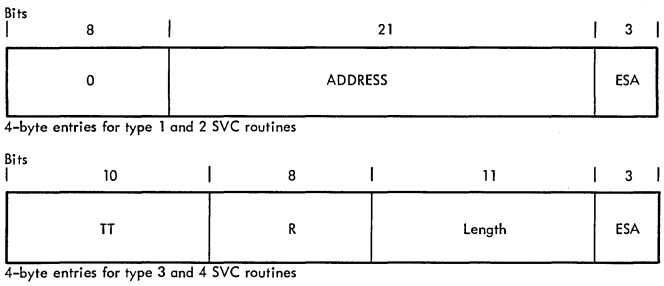


Figure 10. Extended SVC Table Entries

Dispatcher) if a resident type 3 or 4 routine receives control since a resident routine does not operate in the transient area and is not overlaid.

4. The XCTL service routine checks the resident SVC load list created by the Nucleus Initialization Program to determine whether the SVC routine is resident or requires loading.

PSEUDO-DISABLED INTERRUPTION SUPERVISION

The I/O First-Level Interruption Handler (FLIH) contains a switch that is set by the contents supervision subroutine FINCH when main storage is not available during processing of a transient area request. In this case, a fixed work area called the PANIC is used to hold a parameter list passed to the data management BLDL and Program Fetch routines. This switch prevents other processing of requests from overlaying the PANIC. If pseudo-disabled, BLDL and Program Fetch use internal loops rather than issuing a WAIT macro instruction for I/O events. When the switch is set, the I/O and Timer/External FLIHs and the Type 1 SVC Exit routine return control to the interrupted routine, rather than to the Dispatcher, to avoid a task switch. When FINCH processing is complete, the pseudo-disable switch is cleared so that processing for subsequent transient area requests can use the PANIC if necessary.

INTERRUPTION SUPERVISION CONTROL FLOW

The following paragraphs and Figures 13 through 19 describe the processing within and the flow of control between the interruption supervision routines that process supervisor call (SVC), timer/external (T/E), input/output (I/O), and program check interruptions. For a description of the processing of machine checks, see "Recording/Recovery Routines."

SUPERVISOR CALL INTERRUPTIONS

When an SVC interruption occurs, the current PSW is stored in the SVC Old PSW, and the SVC New PSW becomes the current PSW, thereby giving control to the SVC First-Level Interruption Handler (FLIH) (see Diagram 3). If tracing facilities have been specified, the SVC FLIH uses the Trace Table routine (a system generation option), or, if active, the Generalized Trace Facility (GTF) routines to record the interrupt. The SVC FLIH then determines the type of SVC routine requested from the SVC code and the SVC table. For a type 1 request, the SVC FLIH branches to the address indicated for the SVC routine in the SVC Table. Upon return from the type 1 SVC routine, the SVC FLIH uses the type 1 SVC Exit routine to provide linkage to the calling program (using the SVC Old PSW) or to the Dispatcher to determine which program is to receive control next. If the request is for a type 2, 3, or 4 SVC routine, the SVC FLIH branches to the SVC Second-Level Interruption Handler (SLIH) for further processing.

The SVC SLIH (see Diagram 4) obtains main storage and creates an SVRB for the requested routine. This SVRB is queued to the top of the RB queue on the TCB for the requesting program. If the request is for a nonresident type 3 or 4 SVC routine, the SVC SLIH uses FINCH and Program Fetch to bring the requested routine into the SVC transient area, and passes control to the requested routine. Upon return from the type 2, 3, or 4 SVC routine, the SVC SLIH branches to EXIT to provide return linkage to the appropriate routine.

EXIT may be entered from the SVC SLIH, from a user-supplied error processing routine, or from a terminating program that issued an SVC 3 (EXIT) instruction. The EXIT routine (see Diagram 5) returns control to the interrupted program if a user-supplied error handling (SPIE) routine is exiting. If a user-supplied asynchronous exit routine (STAE) is exiting and there are more requests for the routine, EXIT returns control to the user-supplied routine. If the request block for the exiting routine is not the last RB on the queue, EXIT dequeues the RB and uses FREEMAIN to release the RB and program area, and branches to the Dispatcher. If the RB is the last RB on the queue and the TCB is for a job step task, EXIT issues an SVC 13 (ABEND) to normally terminate the task.

In MFT with subtasking, if the RB for the exiting subtask is the last RB on the queue and the "task terminated" flag in the TCB is not set (this will be set by ABEND when only the SVRB for ABEND remains on the queue), EXIT issues an SVC 13 for normal termination. If the "task terminated" flag is set indicating that the only RB on the queue is the SVRB for ABEND, the SVRB is freed, the end-of-task exit routine (if requested) is scheduled, and the TCB is removed from the queue. EXIT indicates that a task switch is necessary, branches to POST (if an ECB was specified) to post the ECB, and branches to the Dispatcher.

The Validity Check Subroutine

Supervisor routines use the Validity Check subroutine to check main storage addresses passed as input parameters by user programs. The Validity Check subroutine tests the following attributes of each

input address: fullword boundary alignment (optional), whether the address lies within the boundaries of main storage, and whether the address specifies a storage area whose storage protection key matches the protection key in the TCB of the calling main-line program. If any of the addresses is invalid the subroutine informs the invoking supervisor routine by altering the condition code in the current PSW. Since the calling main-line program has made a serious error, the supervisor abnormally terminates the current task. Thus, the source of programming error is indicated at its point of occurrence, avoiding a program check during later processing. Such a program check might be difficult to diagnose. If it occurred during queue manipulation, the queues might be seriously disrupted, thus interfering with the performance of the other tasks.

TIMER/EXTERNAL INTERRUPTIONS

When a timer/external (T/E) interruption occurs, the current PSW is stored in the External Old PSW, and the External New PSW becomes the current PSW, thereby giving control to the T/E First-Level Interruption Handler (FLIH) (see Diagram 6). If the Generalized Trace Facility (GTF) is active, the T/E FLIH uses the GTF routines to record the interruption. For an external interruption, control is passed to the Job Management External Interruption Handler (IEECVCRX); for a timer interruption, control is passed to the Timer Second-Level Interruption Handler (IEA0TI02). The T/E FLIH uses the same exit as the Input/Output FLIH (below). Upon return from the routine that processed the interruption, the T/E FLIH returns control directly to the interrupted routine using the T/E Old PSW if the system is pseudo disabled, or passes control to the Dispatcher if the system is not pseudo disabled.

INPUT/OUTPUT INTERRUPTIONS

When an input/output (I/O) interruption occurs, the current PSW is stored in the I/O Old PSW and the I/O New PSW becomes the current PSW, thereby giving control to the I/O First-Level Interruption Handler (FLIH) (see Diagram 7). The I/O FLIH uses the Trace routine to record the interruptiin on the Trace Table (optional), and passes control to the I/O Supervisor to process the interruption. Upon return, the I/O FLIH uses the same exit as the Timer/External FLIH (above), loading the I/O Old PSW if the system is pseudo-disabled, or branching to the Dispatcher if the system is not pseudo disabled.

PROGRAM CHECK INTERRUPTIONS

When a program check occurs, the current PSW is stored in the Program Old PSW, and the Program New PSW becomes the current PSW, thereby giving control to the Program Check First-Level Interruption Handler (see Diagram 8). The interruption is first analyzed to determine if it is either a System/370 or a simulated System/360 Monitor Call instruction. If it is, the Generalized Trace Facility routines are used to record the event, and control is returned directly to the user's program, or to the Dispatcher if a task switch is required. Otherwise, the Generalized Trace Facility routines are used to record the program check interruption and return control to Program Check FLIH to continue processing. If no error-handling routine has been specified for this type of error, the Program Check FLIH passes control to ABTERM (IEA0PL00) to schedule abnormal termination of the program. If an error-handling routine has been provided by the user, the Program Check FLIH passes control to it. Upon return, the Program Check FLIH issues an SVC 3 (EXIT) instruction to pass control to EXIT (IEAATA00).

DISPATCHER PROCESSING

The Dispatcher provides return linkage between the supervisor routines and the program that is to receive control next using the two pointers NEW and OLD (IEATCBP and IEATCBP+4) within the Dispatcher (see Diagram 9). Upon entry to the Dispatcher, OLD contains the address of the TCB for the last program to have control; NEW contains the address of the TCB for the program to receive control next, or is set to 0 indicating that the Dispatcher is to determine which task is to receive control. If NEW is 0, the Dispatcher finds the TCB for the highest priority task that is ready, and sets OLD equal to NEW. If NEW points to a TCB different from the one pointed to by OLD, OLD is set equal to NEW. If tracing facilities have been specified, the Dispatcher uses the Trace Table routine (a system generation option), or if active, the Generalized Trace Facility (GTF) routines to record the task switch. For all cases, the Dispatcher uses the Timer ENQ routine (IEAQTI02) to process timer options included in the system and specified for the new task. If the request block pointed to by the TCB is for a transient SVC that is no longer in storage, the Dispatcher uses the SVC SLIH to create and queue an SVRB. This SVRB represents the subroutine FINCH within the Dispatcher, which passes control to Program Fetch. Program Fetch loads the transient SVC routine. The Dispatcher passes control to the program by loading the resume PSW from the request block at the top of the queue.

HANDLING MACHINE-CHECK INTERRUPTIONS

The function of Machine-Check Handling is to intercept machine-check interruptions and attempt to recover from and report on the particular machine malfunction.

The machine-check recovery programs available vary with the System/360 or System/370 model being used. The availability of each depends on the CPU and the other recovery management program(s) selected. There is one channel recovery program available to service both System/360 and System/370. Figure 11 indicates the availability of each of these programs.

The System Environment Recording, Editing, and Printing program (SEREP) is available for the Model 30 and each higher numbered model except for System/370 Model 155.

Functions of the Machine Check Handlers

When a machine-check interruption occurs, the processing varies according to the recovery option that the user has selected:

- If no recovery program has been selected, the machine loads the Machine-Check New PSW, and the CPU is placed in the wait state.
- If the SER0 routine has been selected, the machine loads the Machine-Check New PSW, and control is given to the resident portion of the SER0 routine to record the environmental data. The SER0 routine then places the CPU in the wait state.
- If the SER1 routine has been selected, the machine loads the Machine-Check New PSW, and control is given to the SER1 routine. This routine records environmental data and either abnormally terminates the job step affected by the machine check and causes resumption of processing, or places the CPU in the wait state.
- If a Machine-Check Handler program has been selected, the machine loads the Machine-Check New PSW, and control is given to the Machine-Check Handler. This program records environmental data and attempts to recover from the machine check. If recovery is not possible, the Machine-Check Handler places the CPU in the wait state.

  Note: When a System/360 or System/370 CPU is placed in the wait state, the operator may then load the System Environment Recording, Editing, and Printing program (SEREP) in order to format and print the CPU logout area.

CPU Model \ Option	MCH	CCH[1]	SER0/SER1
40	NA	NA	O
50	NA	NA	O
65	O	O	O[2]
75	NA	O	O
85	M	A	NA
135	M	A	NA
145	M	A	NA
155	M	A	NA
165	M	A	NA

[1]Available only if a 2860, 2870, 2880, 145, or 155 channel is selected.
[2]MCH cannot be used with SER0 or SER1.

Key: A = automatically included
M = must be included
NA = not available
O = optional

Figure 11. Availability of Machine-Check Programs

The task supervision routines control the execution of tasks primarily by creating and managing task control blocks. By using these routines, user and supervisor programs can change the sequence and timing of events occurring in the central processing unit. Following is an overview of the services and functions provided by task supervision. For additional information on supervisor services, see Supervisor Services and Macro Instructions.

TASK SUPERVISION SERVICES

The following paragraphs summarize the services provided by task supervision in MFT. Each service is initiated by a macro instruction. For information about these macro instructions, see Supervisor Services and Macro Instructions. The ATTACH routine described in "Task Supervision" is available only if the subtasking option is included in the system. This option is included at system generation using the SUPRVSOR macro instruction. (See the publication System Generation.) The ATTACH routine without the subtasking option is described in "Contents Supervision."

CONCURRENT EXECUTION OF PROGRAMS WITHIN A PARTITION

The task supervision ATTACH and DETACH routines allow a problem program to create a task within the bounds of the partition. When a program issues an ATTACH macro instruction, the ATTACH routine builds a task control block for the program specified by the user. This specified program, called a subtask, has the attributes of any other task in the system. Thus, the subtask is able to compete with all other tasks for the use of system resources.

The DETACH routine allows the user to release the main storage and other resources that were allocated to a subtask but were not freed when the subtask terminated.

CHANGING THE DISPATCHING PRIORITY OF A TASK

The priority of a task is determined by its position on the TCB queue. By issuing a CHAP macro instruction, the user can change a task's position on the TCB queue and thus alter its dispatching priority.

EXTRACTING INFORMATION FROM A TASK CONTROL BLOCK (MFT WITH SUBTASKING)

The EXTRACT routine allows the user to obtain information from a TCB. The TCB must be either the TCB of the issuing program or the TCB of one of its subtasks. The information, extracted from one or more of seven fields in the TCB, is returned in a main storage location supplied by the user.

LIMITED EXTRACT (MFT WITHOUT SUBTASKING)

The EXTRACT macro instruction allows a user to determine the address of the task input/output table for the job step.

COMMUNICATING THE COMPLETION OF AN EVENT

The WAIT and POST routines allow a program to wait for the completion of an event. When a program issues a WAIT macro instruction, the WAIT routine places the caller in a wait state. When a servicing program (for instance, the I/O supervisor performing an I/O operation) completes the required event, it issues a POST macro instruction. The POST routine indicates the completion of the event and frees the waiting program.

A program may wait on more than one event and may also wait on fewer events than specified. For instance, the user may place a program into a wait condition pending the completion of any one of three I/O operations. When one I/O operation completes, the user's program regains control.

GAINING EXCLUSIVE ACCESS TO SYSTEM RESOURCES

The ENQ and DEQ routines allow the user to obtain exclusive control of a resource and thus not allow another task to use that resource until it is freed. The resources that can be obtained in this manner include one or more data sets, programs, or work areas within main storage.

The DEQ routine releases the resources obtained by the user through the ENQ macro instruction.

SERIALIZING THE USE OF SHARED DATA SETS (SHARED DASD ONLY)

The ENQ and DEQ routines allow the user to serialize the use of shared data by indicating to the I/O supervisor when to reserve and when to release shared, direct access devices. If two CPUs share a direct access device, then each CPU must issue a RESERVE macro instruction (the RESERVE macro instruction passes control to the ENQ routine) before using the device, and a DEQ macro instruction when the device is no longer needed.

INCREASING SYSTEM AVAILABILITY

The STAE and SPIE routines offer a means of intercepting conditions that would otherwise result in abnormal termination. When an abnormal condition is intercepted, control is passed to user-provided error routines that analyze and, if possible, correct the error.

STAE (specify task asynchronous exit) allows the user to intercept an abnormal termination whenever the task enters the Abnormal End routine (ABEND). Through the parameters of the STAE macro instruction, the user instructs the system to pass control to a user's error routine whenever an abnormal condition is detected.

STAE can be used to prevent abnormal termination during the processing of a supervisor service or data management routines called by the user. Supervisor services such as GETMAIN, FREEMAIN, LINK, and XCTL and data management routines such as GET, PUT, OPEN, and CLOSE are provided in the operating system as a service to the user. An error occurring in these routines usually results in the user's task being abnormally terminated. Also, an error in a lower-level program invoked by CALL, LINK, LOAD, XCTL, or ATTACH macro instructions can result in abnormal termination. In both of the previously mentioned cases, the user can use STAE to prevent abnormal termination.

The SPIE (specify program interruption exit) routine allows the user to specify an error routine that processes selected kinds of program interruptions. The user selects the kind of program interruptions to be serviced by the error routine through the parameters available in the SPIE macro instruction.

THE FUNCTIONS OF TASK SUPERVISION

The functions of task supervision are creating and maintaining task control blocks, and scheduling and allocating system services and resources.

Functions related to creating and maintaining task control blocks are restricted to MFT systems with subtasking and include the following:

- Creating a subtask (ATTACH).
- Detaching a subtask when it has completed execution (DETACH).
- Changing the dispatching priority of a task control block (CHAP).
- Extracting information from a task control block (EXTRACT).

The allocating and scheduling functions include:

- Causing a program to wait for one or more events (WAIT).
- Posting the completion of an event (POST).
- Serializing the use of a resource (ENQ, RESERVE).
- Releasing a resource from exclusive control (DEQ).
- Specifying a program interruption exit (SPIE).
- Specifying a task asynchronous exit (STAE).
- Scheduling an end-of-task exit routine (ETXR operand).

CREATING A SUBTASK

When a user program issues an ATTACH macro instruction, the ATTACH routine gets control to build a task control block and queue the TCB on the TCB queue (see Diagram 10). ATTACH builds a dummy program request block (PRB) and queues it to the new TCB. ATTACH initializes the resume PSW field of the PRB with the address of an SVC 6 (LINK) instruction contained in the CVT. This procedure establishes linkage to the requested program so that the program will execute under control of the new TCB (see "Linking to the Subtask" below).

Prior to queueing the TCB, ATTACH prepares for subtask termination as follows:

- If the caller did not specify an event control block (ECB) or ETXR address through the operands of the ATTACH macro instruction, ATTACH schedules a system end-of-task exit routine.

- If the caller specified an ECB address, ATTACH checks the validity of the address.

- If the caller specified an ETXR address, ATTACH establishes linkage to the user end-of-task exit routine.

Upon completion of its processing, ATTACH issues an SVC 3 (EXIT) instruction. When the TCB created by ATTACH is the highest priority TCB that's ready, the LINK macro instruction is executed which brings the requested module into the partition and queues the request block for the module to the RB queue of the new TCB (see Diagram 11). LINK issues an SVC 3 (EXIT) instruction which causes its own SVRB to be removed from the RB queue, leaving only the newly created request block for the module. The requested module gains control when the Dispatcher determines that it is the highest priority ready task in the system.

TCB Address Table

Before processing the request to create a subtask, ATTACH searches the TCB address table to determine whether the maximum number of active subtasks is currently executing. The TCB table is created during system generation and initialized during nucleus initialization (NIP) to address the system task TCBs in order of priority. The TCB address table also contains empty fields following the partition TCB entries. These fields are used by ATTACH to address subtasks created by the system. The number of empty fields created during system generation corresponds with the maximum number of subtasks that can exist at any given time.

Queuing the Task Control Block

When the task receives control from the initiator/terminator, the TCB queue is arranged by priority, with the system tasks having the highest dispatching priorities at the top of the queue. The partition TCBs are queued below the system TCBs and range in priority from the highest, partition 0, to the lowest, partition n.

ATTACH places the subtask TCB on the dispatching queue, using its dispatching priority to determine its position. Once a new TCB is queued, the partition priority scheme is no longer relevant because subtasks TCBs may be assigned any priority lower than that of the creating task.

Since a subtask belongs to a particular job step, the subtask's TCB must also be queued to the calling task's TCB. If there are other subtask TCBs on this subtask queue, the TCB for the newly created task is placed on the top of the queue. The subtask queue is used to terminate all the subtasks of a job step task and to change the dispatching priority of a subtask.

When the time-slicing feature is included in the system, the ATTACH routine determines whether the new TCB represents a time-sliced task. A task is time-sliced if its dispatching priority falls within the limits of the dispatching priorities assigned to the time-sliced group. If the new task is time-sliced, ATTACH sets the time-slice bit (TCBFTS) in the TCB and updates the pointers in the Time Slice Control Element (TSCE).

Determining Whether a Task Switch is Needed

ATTACH determines whether the caller (parent task) or the subtask (daughter task) will receive control first from the Dispatcher. The dispatching priority of the parent task is compared to that of the daughter task and the address of the task with the higher priority is stored in the NEW TCB pointer (IEATCBP). NEW is later tested by the Dispatcher in determining which task gets control first.

Linking to the Subtask

Diagram 10 illustrates the TCB queue just before control passes to EXIT from ATTACH. The SVRB controlling the execution of the ATTACH routine is at the top of the RB queue between the caller's RB and the TCB. The subtask TCB is queued on the TCB queue in the position assigned to it according to its dispatching priority. The dummy PRB's resume PSW addresses an SVC (LINK) instruction. When the subtask TCB is dispatched, the one-instruction program (SVC 6) executes. The SVC handlers build an SVRB for the LINK routine and queue it at the top of the subtask's RB queue.

LINK obtains the name of the requested program from the caller's registers and the ATTACH parameter list (the parent task's registers were stored in the subtask TCB by ATTACH). LINK processes the request as follows:

- An attempt is made to find the requested program in main storage.

- If the search for the program is not successful, the program is loaded into main storage.

- An RB is built to control the execution of the program.

- The RB belonging to the caller (in this case the dummy PRB) is removed from the RB queue.

Diagram 11 illustrates the subtask TCB's RB queue at entry to LINK, at completion of LINK processing, and at completion of EXIT processing.

DETACHING A SUBTASK

The DETACH routine permits a program (the parent task) to detach its subtask (daughter task) if the daughter task has terminated. A DETACH macro instruction must be issued if the ECB or ETXR operands were originally specified in the ATTACH macro instruction that created the subtask. DETACH processes a request as follows:

- Checks the validity of the subtask's address.
- Dequeues the subtask's TCB from all queues to which it belongs (that is, the dispatching queue, the subtask queue, and, if applicable, the time-slice queue).
- If the subtask has not terminated, DETACH branches to ABTERM to terminate the subtask. If an ECB address was originally passed in the parameters of the ATTACH macro instruction, DETACH adjusts the ECB to inform the parent task that the daughter task has abnormally terminated.
- Returns control to the caller by issuing an SVC EXIT instruction.

CHANGING THE DISPATCHING PRIORITY OF A TASK

The CHAP macro instruction permits a problem program executing in a partition to change its own dispatching priority or the dispatching priority of any of its subtasks. A program issuing the CHAP macro instruction may change the dispatching priority to any value between 0 and the limit priority of the issuer's TCB. Diagram 12 illustrates the relative positions of TCBs on the queue before Program A issues an ATTACH macro instruction, after the ATTACH is issued, and after Program A changes its own dispatching priority. The distinction between dispatching and limit priorities is explained in the next paragraphs.

Both limit and dispatching priorities are specified as parameters of the ATTACH macro instruction. The dispatching priority determines the appropriate position of a TCB on the TCB queue, and thus determines which task or subtask gains control after an interruption occurs. The Dispatcher places in execution the ready program represented by the TCB of highest dispatching priority. In contrast, the limit priority of a task is used by the CHAP SVC routine to determine the maximum value to which it may increase the dispatching priority of the task. The limit priority of any subtasks will not be more than the limit priority of the originating task.

After the dispatching priority is changed, the CHAP routine rearranges the TCB queue by placing the affected TCB in a new logical location reflecting its new priority. If the changed task's new priority falls within the range of priorities assigned to the time-slice group, CHAP places the TCB on time-slice queue.

The CHAP routine can also be used to increase the limit priority of any of the active task's subtasks. The active task cannot change its own limit priority. When the dispatching priority of a subtask is raised above its own limit priority, the CHAP routine raises its limit priority to equal its new dispatching priority. However, a subtask's limit and dispatching priority cannot exceed the job-step's limit priority.

Reference information on using the CHAP macro instruction can be found in Supervisor Services and Macro Instructions.

CAUSING A PROGRAM TO WAIT FOR ONE OR MORE EVENTS

The WAIT routine allows a user or system program to wait until a specified number of events has occurred, such as the completion of one or more I/O operations. When the event is completed, the POST routine records the completion of the event in a communication area called the event control block (ECB) and takes the waiting program out of the wait condition if no other events are being waited on.

The functions performed by the Wait routine are:

- Placing the requesting program into a wait state.
- Initializing the wait flag in the event control block (ECB).
- Incrementing the wait count in the caller's RB.

Initializing the Event Control Block

The event control block is specified by a program when it issues a WAIT macro instruction. The ECB must be initialized by the caller so that bits 0 and 1 (the wait and completion bits respectively) are set to 0. The WAIT routine sets the wait bit to 1 upon entry. If the user specifies

more than one ECB, the wait bit in each ECB is set to 1 by the WAIT routine. The wait bit indicates to the POST routine that the event represented by an ECB is being waited on.

The WAIT routine also places the address of the waiting program's request block into the address portion of the completion-code field of the ECB, so that the POST routine can identify the waiting program.

Keeping Track of the Number of WAITs Issued by a Program

The wait count (the number of WAIT macro instructions issued by a routine) is maintained in the caller's RB by the WAIT routine. The wait count is incremented by the WAIT routine whenever it is entered. However, if the WAIT routine finds that the event to be waited on has already occurred, it decrements the wait count by one. Thus, the wait count retains its original value. If the wait count becomes 0, the caller is not placed in a wait state.

POSTING THE COMPLETION OF AN EVENT

The POST routine permits a program (the "posting" program or caller) to signal the completion of an event such as an I/O operation. The POST routine signals (posts) the event's completion by setting the completion bit in the ECB shared by both the waiting and posting program. The POST routine also places in the ECB a completion code, if the posting program supplies one. The completion code may later be inspected by the waiting program after it resumes execution to determine the type of completion that occurred. The POST routine returns control to the caller via the type 1 Exit routine.

SERIALIZING THE USE OF A RESOURCE

The ENQ routine permits programs issuing the ENQ macro instruction to gain one-at-a-time access to a resource or set of resources. The requested resource may include one or more data sets, records within a data set, programs, or work areas within main storage. The routine places in a resource queue all resource requests specified in the caller's macro instruction. If no other program is using a requested resource, the requester gains access to its resource. But if any requested resource is already in use by another program, the ENQ routine places the caller in a wait condition until the resource becomes available. When the resource is no longer needed by the other program, that program issues a DEQ macro instruction that causes the DEQ routine to remove one or more elements from the request queue and reduces the wait count for the waiting program. If the wait count becomes 0, the DEQ routine, via the Exit routine and the Dispatcher, returns control to the previously waiting (now ready) program. The latter then gains access to the resource.

Types of Resource Requests

There are two types of resource requests which may be specified using the ENQ macro instruction: an exclusive (E) request or a shared (S) request. The ENQ routine handles these two types of requests differently. Exclusive requests are handled on a first-in, first-out basis. That is, an exclusive request in the queue may not be serviced until all earlier requests of either type have been serviced. Also, later requests of either type may not be serviced until a previously entered exclusive request has been handled. However, shared requests placed consecutively in the queue may be serviced as a group, if one of the shared requests is at the top of the queue. That is, the group of shared requests are handled on a task-priority basis. Figure 12 indicates the handling of typical combinations of shared and exclusive requests.

Description of the Resource Queues

Each resource request in the ENQ macro instruction specifies a qname, which names a set of resources, and an rname which names a single resource within the set identified by the qname. The qname is represented on the resource queues by a major queue control block (QCB). Each major QCB contains the qname for a set of resources, for example, the name of a data set. A major QCB thus represents a set of resources.

A nonexecutable module has as its first word the address of the first major QCB. Each major QCB points to a minor QCB, which represents a particular resource within the set of resources, for example, a specific record within a data set. A minor QCB contains an rname that is the name of the particular resource that has been requested. Each minor QCB, if another resource within the set has been requested, points to another minor QCB. Thus, each minor QCB represents a particular resource that has been requested within a set of resources represented by a major QCB.

Each minor QCB contains the list origin for a queue of one or more queue elements (QELs). Each QEL represents a request for a single resource by a program belonging to a specific task. If a program requests more than one resource, the ENQ routine constructs additional QELs, each represent-

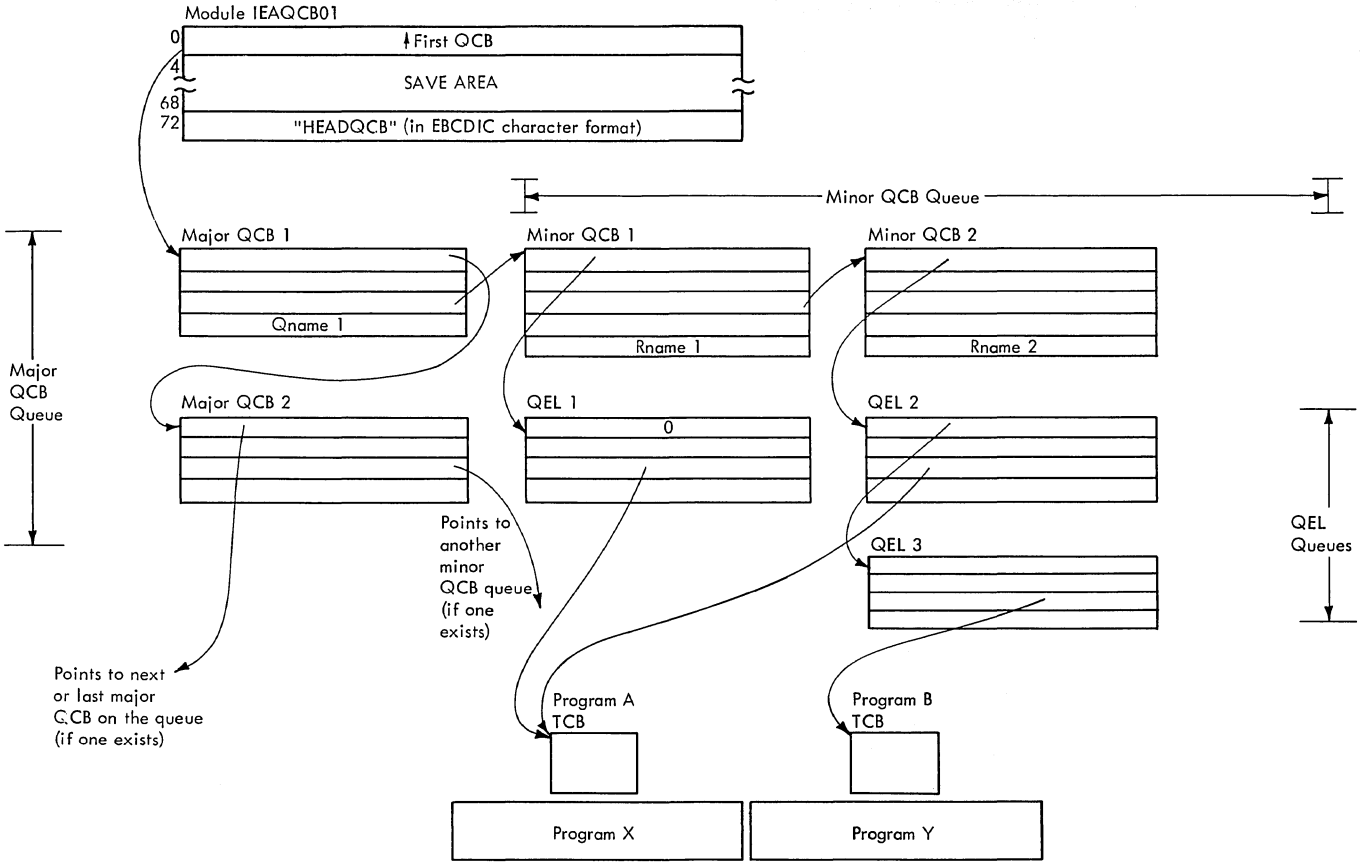
Condition 1: A group of shared requests is at the top of the resource queue.	shared request shared request exclusive request	The resources are used by the shared requesters on a task-priority basis. The exclusive requester waits until the shared requesters have completed their use of the resource and have removed their requests from the queue.
Condition 2: An exclusive request at the top of the queue is followed by a group of shared requests.	exclusive request shared request shared request	The exclusive requester has access to the resource. The shared requesters wait until the resource is free. They then share the resource on a task-priority basis.
Condition 3: An exclusive request at the top of the queue is followed by a second exclusive request.	exclusive request exclusive request	The first (top) exclusive requester uses the resource while the second exclusive requester waits. When the first requester has completed its use of the resource, the second requester can proceed.
Condition 4: A group of shared requests is at the top of the queue, followed by an exclusive request. The exclusive request is followed by a group of shared requests.	shared request shared request exclusive request shared request shared request	The resource is first shared on a task-priority basis by the shared requesters whose requests are at the top of the queue. The exclusive requester waits until the shared requesters have completed their use of the resource and have removed their requests from the queue. The exclusive requester then has exclusive access to the resource. The shared requesters lower on the queue wait until the resource is available. They then share the resource on a task-priority basis.

Figure 12. Handling Shared and Exclusive Requests

ing a request. If all the QELs that represent resource requests by a program are at the top of their respective QEL queues, the program may use the resources. That is, the program is not waiting and can gain access to the resources as soon as it is dispatched. But if all the QELs that represent requests by a program are not at the top of their respective QEL queues, the requesting program must wait. The program that is using the resources must finish and issue a DEQ macro instruction. The DEQ routine then moves the needed QELs to the top of the queues.

In Figure 13, program A is using, or is about to use, the resources represented by major QCB 1 and minor QCBs 1 and 2. Its requests are at the top of the queues, represented by QELs 1 and 2. Program B has requested one of the resources being used by program A, that represented by major QCB 1 and minor QCB 2. Since the resource desired by program B is already in use, the program must wait, its request remaining on the queue as QEL 3.

Note that each requested resource is represented by a combination of one major QCB and one of its associated minor QCBs. Each request is represented by a queue element (QEL), which points to the TCB associated with the requesting program. If there is not at least one QEL for a previously requested resource, the DEQ routine, when the DEQ macro instruction is issued, removes the associated minor QCB. (Under certain conditions, the DEQ routine also removes a major QCB.) Thus, if there are control blocks -- major QCB, minor QCB,



NOTES: 1. Arrows represent pointers.
2. Each combination of a major QCB, a minor QCB, and a QEL represents a resource requested for a particular task.
3. Program X is using resources Rname 1 and Rname 2.
4. Program Y awaits resource Rname 2.

Figure 13. Resource Queues

and QEL -- on the resource queues, there must be at least one request for a resource that has not yet been completed.

Functions of the ENQ Routine

MAJOR FUNCTIONS: When one or more resources are requested, via the ENQ macro instruction, the major functions are:

- If necessary, creating one or more queue control blocks (QCBs) to represent the requested resource, and placing these queue control blocks on the resource queues.
- Depending on the RET parameter, creating a queue element (QEL) to represent the request, and placing the QEL on a QEL queue.
- If the resource is available, returning control to the requester, with or without a return code that indicates the availability of the resource, depending on the RET parameter.
- If the requested resource is not available, taking one of two actions, depending on the RET parameter:
  - Placing the requester in a wait condition, pending the availability of the resource
  - Returning control to the requester with a code that indicates that the resource is unavailable.

MINOR FUNCTIONS: When one or more resources are requested, the minor functions are:

- Setting the caller's task to "must complete" status, if specified and the caller has a protect key of 0.
- Detecting abnormal conditions that can cause the generation of an error code

or the abnormal termination of the caller's task.

- Purging QELs from the resource queues for an abnormally terminated task.

- Increasing the "enqueue count" in the requester's TCB.

FUNCTIONS OF THE ENQ ROUTINE WHEN A RESERVE REQUEST IS ISSUED: When a caller issues a RESERVE macro instruction to reserve a resource, the ENQ routine:

- Creates, if necessary, one or more queue control blocks (QCBs) to represent the requested resource. These control blocks are then queued on the resource queues.

- If the resource is available, the requester gives control of the resource with a return code that indicates the availability of the resource, depending on the value coded in the RET parameter of the ENQ macro instruction.

- If the requested resource is not available, one of two actions is taken:

  - Places the requester in a wait condition, pending the availability of the resource.

  - Returns control to the requester with a code indicating that the resource is unavailable.

Functions of the DEQ Routine

When a program that previously issued an ENQ macro instruction (or, in systems that include the shared DASD feature, a program that issued a RESERVE macro instruction) is through with the requested resource, it releases the resource by issuing a DEQ macro instruction. The DEQ macro instruction, via an SVC 48 instruction, obtains supervisor-assisted linkage to the DEQ routine. This routine removes one or more QELs, a minor QCB, or a major QCB from the resource queues. It also reduces the RB wait count for the waiting program whose QELs are at the top of the QEL queue. If the RB wait count becomes 0, indicating that the waiting program is ready, the DEQ routine determines whether a task switch is needed, and issues an SVC 3 (EXIT) instruction to return control. The program that receives control is either the caller or a previously waiting program, depending on relative task priority.

An additional function of the DEQ routine, used only by a supervisor routine, is to turn off the "must complete" flags in the TCB set previously by an ENQ macro instruction issued by the caller.

MAJOR FUNCTIONS: When the use of one or more resources is signaled complete, via the DEQ macro instruction, the major functions are:

- Updating the resource queues by dequeuing and freeing the queue element (QEL) that represents the request for the resource. If there are no more requests for the resource, one or more queue control blocks (QCBs) that represent the resource are dequeued and their space is freed.

- For the next requester represented on the QEL queue, reducing the wait count in its RB, and determining whether the requester is ready to resume execution.

- Comparing dispatching priorities to determine whether a readied requester can replace the caller as the next-to-be executed routine.

- Returning control to the caller if no other readied requester's task has a higher priority than the caller's. If another readied requester's task has a higher priority, control is returned to the other requester instead of to the caller.

MINOR FUNCTIONS: When the use of one or more resources is signaled complete, via a DEQ macro instruction, the minor functions are:

- Clearing of the "must complete" status of the caller's task, if the "reset must complete" parameter is present.

- Checking the validity of input addresses.

- Determining whether a specified resource was originally requested for any task.

- Checking whether the caller has access to a specified resource.

- Reducing the "enqueue count" in the caller's TCB. This count is tested during normal task termination by the EOT routine to determine if all resource requests for the task have been dequeued. (See "Termination Procedures.")

- Issuing an EXCP instruction to release a reserved resource when the reserve count in the UCB is 0.

SIGNALING THAT A RESERVED RESOURCE IS NO LONGER NEEDED

When a program that previously issued a RESERVE macro instruction is finished using the reserved resource, it issues a DEQ macro instruction. The DEQ routine decrements the UCB reserve count pointed to by the specified minor QCB. DEQ then removes the minor QCB from its queue if the UCB reserve count (field UCBSQC in the UCB) has become 0. If there are no additional minor QCBs queued to the major QCB, DEQ frees the main storage of the major QCB. The DEQ routine also decrements the task enqueue count in the TCB (field TCBLMP) to indicate that a resource was freed. If the UCB reserve count is 0, DEQ issues an EXCP to free that device.

INTERCEPTING PROGRAM-CHECK INTERRUPTIONS

When a program interruption occurs, the program-interruption handler passes control to ABEND to abnormally terminate the task and (optionally) provide a dump.

The sequence of events taking place when a program interruption occurs can be altered. The SPIE macro instruction allows the user to specify a user-written routine that is to get control when a program interruption occurs.

Creating the Program Interruption Control Block

The SPIE macro instruction creates executable coding during assembly of the source program. When executed, this coding creates and initializes a program interruption control area (PICA) to be used by the SPIE routine when it gets control following a program interruption.

Functions Performed by the SPIE Routine

The SPIE routine completes the processing needed for a user to specify a program-interruption exit routine.

- SPIE obtains 32 bytes of main storage, builds a program-interruption element (PIE), and places the address of the PIE in the TCB.
- SPIE places the address of the PICA in the first four bytes of the PIE.
- SPIE stores a program mask (as determined from the types of interruptions to be intercepted) in the caller's resume PSW.

See Section 5 for the formats of the PIE and PICA.

SPECIFYING A TASK ASYNCHRONOUS EXIT

The STAE routine allows the user to specify an abnormal exit for a task. When a STAE macro instruction is issued, control is passed to the STAE routine (see Diagram 13). The STAE routine creates a STAE control block (SCB) to be used later when abnormal termination is scheduled.

Information Returned to User

When control is returned to the user after the STAE macro instruction has been issued, register 15 contains one of the following return codes:

Code	Meaning
00	An SCB is successfully created, overlaid, or canceled.
04	Storage for an SCB is not available.
08	The user is attempting to cancel or overlay a non-existent SCB, or is issuing a STAE in his STAE exit routine.
0C	The exit routine or parameter list address is invalid.
10	The user is attempting to cancel or overlay an SCB not associated with his level of control.

Processing When an Abnormal End is Indicated

When a program that issued a STAE macro instruction loses control to the ABEND routine as a result of an abnormal condition, the first load of ABEND calls the ABEND/STAE interface routines.

The ABEND/STAE interface routines:

- Purge active I/O.
- Establish a work area for the user's exit routine.
- Schedule the user's exit routine.
- Schedule the user's retry routine (optional).

Information Passed to User's Exit Routine

Before the user's exit routine is entered, the purge request, as specified in the STAE macro instruction, has been handled. The registers contain the following information:

- Register 0: One the following codes indicating the status of I/O operations for the task:

Code	Meaning
0	Active I/O at the time of the abnormal condition was halted and is restorable.
4	Active I/O at the time of the abnormal condition was halted and is not restorable.
8	No I/O was active at the time of the abnormal condition, or the STAE macro instruction specified PURGE=NONE.

- Register 1: Address of a 104-byte work area.
- Registers 2-12: Unpredictable.
- Register 13: Address of a register save area.
- Register 14: Return address.
- Register 15: Address of the STAE exit routine. Registers 13 and 14, if used by the STAE exit routine, must be saved and restored prior to returning to the calling program. Standard linkage conventions are used.

If storage was not available for the work area, the register contents upon entry to the user exit routine are as follows:

- Register 0: 12.
- Register 1: ABEND completion code as it appears in the TCBCMP field.
- Register 2: Address of STAE exit parameter list.
- Register 3-13: Unpredictable.
- Register 14: Return address.
- Register 15: Address of the STAE exit routine. Registers 13 and 14, if used by the STAE exit routine, must be saved and restored prior to returning to the calling program. Standard linkage conventions are used.

The User's Retry Routine

Upon completion of the user's exit routine, the user indicates whether ABEND processing is to continue for the task or a STAE retry routine should be scheduled. The return codes to be placed in register 15 are as follows:

Code	Meaning
0	ABEND processing to continue.
4	A retry routine has been provided and the request block chain should be purged.
8	A retry routine has been provided and the request block chain should not be purged.

The address of the retry routine will be returned from the user's exit routine in register 0.

The purpose of the user's retry routine is to restart the task. Upon entry to retry routine, the registers contain:

- Register 0: 0.
- Register 1: Address of the work area, as previously described, except that word 2 now contains either the address of the first I/O block on the restore chain, and word 26 now contains the address of the I/O block restore chain. Both words contain 0 if I/O was not purged.
- Registers 2-13: Unpredictable.
- Register 14: Return address.
- Register 15: Address of the user's retry routine.

Again, if storage was not available for a work area, the register contents upon entry to the user's retry routine are as follows:

- Register 0: 12.
- Register 1: ABEND completion code.
- Register 2: Address of the first I/O block on the restore chain, or 0 if I/O is not restorable.
- Registers 3-15: Same as above.

Relationship of STAE Modules

The functions of STAE are performed by five modules: the STAE service routine module, and the ABEND/STAE interface routine (ASIR), load modules 1 through 4. The function performed by each of these five modules is briefly stated below:

- STAE service routine (IGC00060) -- Receives control via an SVC 60 instruction when the STAE macro instruction is issued. It checks the validity of the STAE request, and creates, cancels, or overlays a STAE control block (SCB).

- ASIR1 (IGC0111C) -- Receives control from the ABEND routine. It purges I/O operations (unless PURGE=NONE) that are in progress for the abnormally terminating task, establishes a work area, and schedules and gives control to the user-written STAE exit routine. If the exit routine does not request that a retry routine be executed, the Message Information List is purged of entries for the current TCB, and ASIR1 returns control to ABEND processing when the exit routine has finished. If a retry routine is requested, ASIR1 passes control to ASIR2. If the program using STAE is in supervisor mode and requests execution of a retry routine without a purge of the RB chain, ASIR1 passes control to ASIR3.

- ASIR2 (IGC0211C) -- Receives control from ASIR1. It locates and closes data sets allocated to RBs between the RB requesting STAE and the RB of the failing program. It uses the WTOR Purge routine. If any of the data sets to be closed are being processed by BTAM, QTAM, BISAM, or QISAM, ASIR2 passes control to ASIR4. If none of the above access methods are indicated, ASIR2 passes control to ASIR3.

- ASIR3 (IGC0311C) -- Receives control from either ASIR1, ASIR2, or ASIR4. It sets the tasks related to the failing task dispatchable, frees the main storage occupied by the SCB, and schedules the user's retry routine so that it is the next program executed.

- ASIR4 (IGC411C) -- Receives control from ASIR2. It repeats the search for data sets that are being processed by BTAM, QTAM, BISAM, or QISAM, closes these data sets, and passes control to ASIR3.

SCHEDULING A USER EXIT ROUTINE

A user program may request the future execution of its exit routine to handle an unpredictable event, such as an end-of-task condition, expiration of a timer interval, or special I/O handling (for example, tape label checking or I/O error checking). The scheduling of user exit routines (sometimes called asynchronous exit routines) is handled by several supervisor routines: the Stage 1 Exit Effector, the Stage 2 Exit Effector, the Stage 3 Exit Effector, and the EXIT routine. These routines do not schedule the execution of user program-check routines. Abnormal termination that results from a program check can be intercepted by a SPIE or a STAE macro instruction.

As shown in Figure 14, the handling of a request for the future execution of a user exit routine is a multipart procedure, interwoven with the execution of programs that are part of other tasks. The procedure begins when the user program provides an exit routine via operands in such macro instructions as ATTACH (ETXR operand), STIMER, and DCB. A system routine (e.g., the ATTACH routine) then issues an SVC 43 instruction to pass control to the Stage 1 Exit Effector, which constructs an interruption request block (IRB) to handle future scheduling of the user exit routine. In addition, the system routine constructs an interruption queue element (IQE), which Stages 2 and 3 of the Exit Effector and the EXIT routine later manipulate to schedule the execution of the user exit routine. Data management routines, however, do not construct IQEs, since I/O queue elements already exist. These elements are called request queue elements (RQEs) and are made available by the I/O Supervisor.

After the Stage 1 Exit Effector constructs the IRB to represent the user routine, no scheduling occurs until an unpredictable event takes place that requires the exit routine. The Stage 2 Exit Effector, a supervisor subroutine, performs initial scheduling of the user exit routine by placing the previously constructed queue element, an IQE or RQE, on its appropriate exit queue. One queue contains IQEs that represent requests to use a routine such as a timer exit routine or an end-of-task routine. The other queue represents requests for data management exits and contains only RQEs. These RQEs are the same elements that the I/O Supervisor uses to schedule I/O requests. Both exit queues operate in first-in, first-out order, with no regard to the task priority.

When an element is placed on either exit queue, the IRB that represents an exit routine is not yet on a task's RB queue and cannot yet be executed. Placing a queue element on an exit queue by the Stage 2 Exit Effector is therefore only a "bookkeeping" manipulation.

The Stage 3 Exit Effector completes the scheduling of the user exit routine. Stage 3, a subroutine of the Dispatcher, removes queue elements from either of the two exit queues and places them on another queue whose list origin is the IRB representing the exit routine. The transferred queue elements are thus queued for a specific exit routine. Stage 3 completes the scheduling of the user exit routine by placing the IRB on the RB queue of the requesting program's task. The exit routine, so scheduled, competes for CPU time with programs being executed for other tasks. When the requesting program's TCB, which points

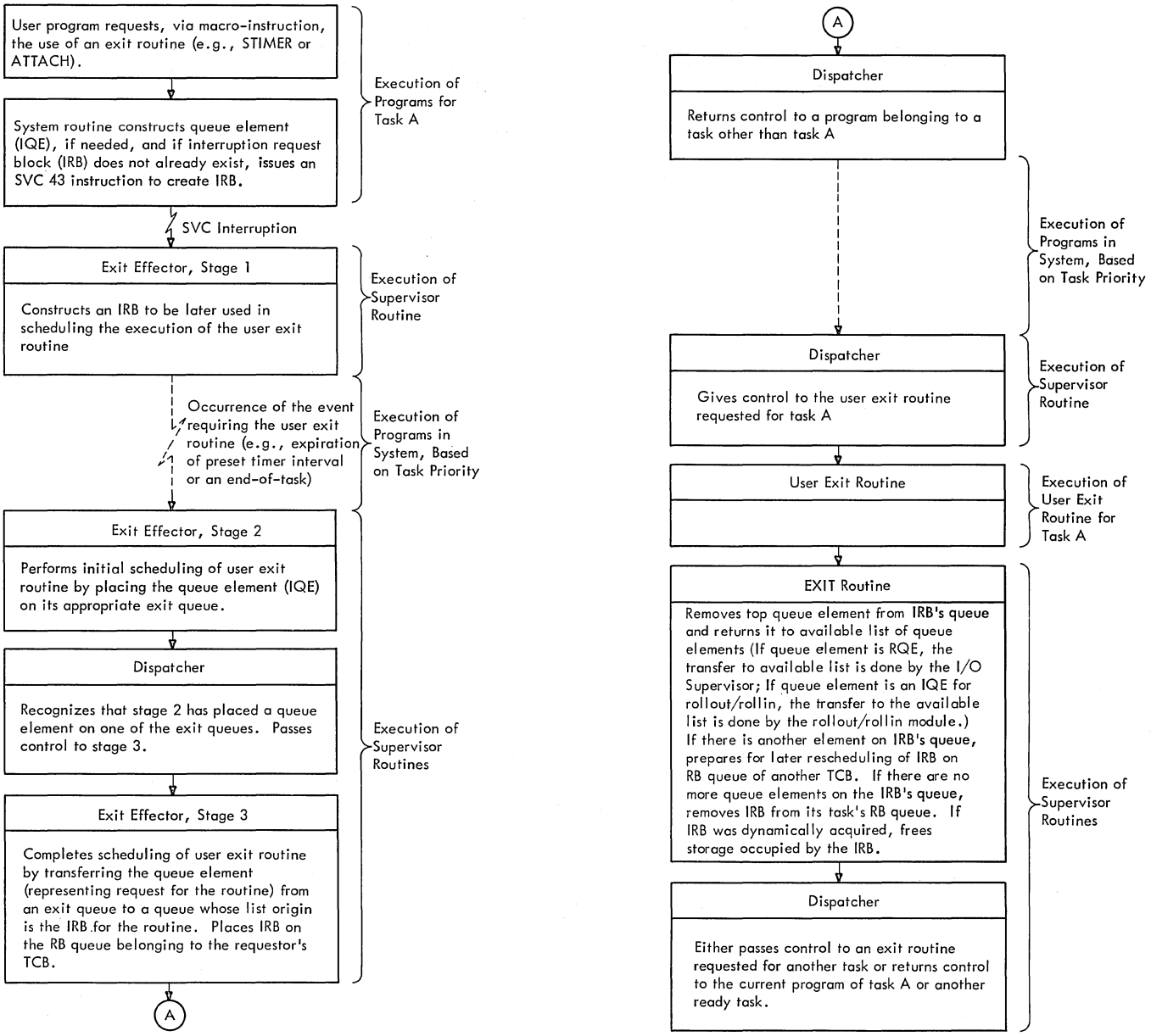


Figure 14. Scheduling Asynchronous Exit Routines

to the IRB, has the highest priority among the ready TCBs, the Dispatcher loads the IRB's old PSW to give control to the user exit routine.

When the user exit routine is finished, it invokes supervisor-assisted linkage to the supervisor EXIT routine. The supervisor EXIT routine removes the top queue element from the IRBs queue of request elements. The removed element represents the satisfied request for the user exit routine.

After removal from the IRB's queue, the queue element is returned to a free list. If there are more elements on the IRB's queue, representing other requests for the routine, the EXIT routine prepares for later rescheduling of the IRB on the RB queue of a different task. If there are no more queue elements on the IRB's exit queue, the EXIT routine dequeues the IRB from the TCB's RB queue, and, if the IRB is not a system RB, frees the storage occupied by the IRB. Thus, the scheduling process, which began with the construction of an IRB at macro-execution time, ends with the possible release of the IRB after the user exit routine is complete.

Functions of the Exit Effectors

The exit effectors schedule the execution of the user exit routine in the following way:

- The Stage 1 Exit Effector creates and initializes, depending on the input parameters passed by the caller, an interruption request block (IRB) which will control the execution of the user exit routine. The routine contains a work area in which the caller may construct interruption queue elements (IQEs), and optionally, a 72-byte register save area in which the user exit routine may later save the registers of the requesting program. The ATTACH routine uses the work area to construct the IQE for the ETXR or end-of-task exit routine. The Stage 1 Exit Effector obtains space for the IRB from the partition. The work area follows and is contiguous with the IRB.

- The Stage 2 Exit Effector is entered by a supervisor routine to schedule a user exit routine. Two typical callers are the supervisor End-of-Task routine (EOT), during end-of-task processing, and the Timer Second-Level Interruption Handler, when a preset timer interval has expired. Stage 2 places the queue element on the queue of RQEs used for scheduling data management exits. If, however, the input address is in complement form, Stage 2 interprets the input queue element as an IQE and places it at the end of the IQE list, whose elements are used to schedule non-data-management exits. (See Section 5 for the format and content of IQEs and RQEs.) Stage 2 then sets a Stage-3 switch (IEA0DS01), which the Dispatcher will test later to determine whether to call Stage 3 to complete the scheduling begun by Stage 2.

- The Stage 3 Exit Effector operates as a subroutine of the Dispatcher. Its purpose is to transfer IQEs and RQEs from their exit queues to queues belonging to particular IRBs and, if possible, to place these IRBs on the RB queues of the appropriate TCBs. As soon as an IRB is on an RB queue, the exit routine represented by the IRB may (if task priority permits) be placed in execution by the Dispatcher. An additional function of Stage 3 is to schedule a request (RQE) for an I/O error routine by placing its system interruption request block (SIRB) on the system error task TCB. (MFT with subtasking only. If subtasking is not included, the SIRB is queued to the TCB of the failing task.)

The EXIT Routine

This discussion of the EXIT routine includes only that part of its processing that affects the scheduling of user (asynchronous) exit routines. Other aspects of its processing are described under "Interruption Supervision."

The EXIT routine deletes the scheduling performed by the three Exit Effectors. The EXIT routine is given control after a user exit routine has issued a RETURN macro instruction. If there are no other requests (queue elements) for the user exit routine, the EXIT routine removes the IRB from its TCB (in MFT with subtasking, the SIRB is permanently queued to the system error task TCB) and frees the storage occupied by the IRB. The top IQE or RQE, representing a request for the user exit routine, is removed from the IRB's list of queue elements since the request has been satisfied. If there is a list of available unscheduled queue elements, the EXIT routine returns the removed element to the "available" list. If, however, another element remains on the RB's queue, representing an additional request for the asynchronous exit routine, the EXIT routine prepares for future reentry to the user exit routine and does not remove the RB from its TCB. In all cases except the last, the EXIT routine branches to the Transient Area Refresh routine. If the asynchronous exit routine must be reentered for another request, the EXIT routine branches to the Dispatcher.

Termination Services provide the final processing necessary for programs that have normally or abnormally completed execution. The ABTERM routines schedule abnormal termination; the ABEND routines actually terminate the task or subtask; the ABEND/STAE Interface routines schedule a user specified routine to handle program errors; the Damage Assessment routines attempt to recover from certain errors; and the ABDUMP routines provide the programmer with a picture of storage so that he may trace any problems within the program.

ABNORMAL TERMINATION SCHEDULING

ABTERM is used by a type 1 SVC routine to terminate the currently dispatched task and by any SVC routine to terminate a task that is not the currently dispatched task. ABTERM forces the specified task to terminate abnormally by replacing the address of the next sequential instruction, in the right half of the PSW, with the address of an SVC 13 (ABEND) instruction. It stores the completion code, sets the terminating task for immediate dispatching, and returns to the caller.

NORMAL AND ABNORMAL TERMINATION

Both normal and abnormal termination are performed by the ABEND modules. Essentially, the ABEND modules remove either a completed program or a program unable to continue processing because of an error. The modules then reinitialize the system to prepare it for the next task. All control blocks associated with the terminating program are purged, all outstanding I/O is halted, and the partition in which the program was located is made available for the next job.

ABEND is a type 4 SVC routine that performs both normal and abnormal task termination by freeing the resources and control blocks associated with the terminating task. The freed resources thus become available for use by other tasks in the system. The control blocks affected are:

- Task Control Blocks (TCBs)
- Data Extent Blocks (DEBs)
- Free Queue Elements (FQEs)
- Request Blocks (RBs)
- Interrupt Queue Elements (IQEs)
- Timer Queue Elements (TQEs)
- Gotten Subtask Area Queue Elements (GQEs)
- Program Interrupt Elements (PIEs)

ABEND can be entered directly from the problem program or system task via an ABEND macro instruction, which expands to include an SVC 13 instruction, or indirectly through the ABTERM service routine (see "Abnormal Termination Scheduling" above). The SVC SLIH causes the first load module of ABEND (IEANTM00) to be brought into the SVC transient area and passes control to it. Control is passed between ABEND modules using XCTL macro instructions (SVC 7).

ABEND processing varies depending on the type of termination (normal or abnormal), the type of task terminating (job step or subtask), options included in the system, conditions causing the termination, and conditions arising within ABEND during processing (macro failures, etc.). The following paragraphs describe the normal and abnormal task termination functions of ABEND. For a more detailed description of ABEND processing, refer to the module descriptions in Section 3.

Normal Termination

ABEND stores the normal completion code of the terminating task in the TCBCMP field in the TCB of the terminating task. If the system has an interval timer, the IQE pointer in the TQE is cleared. If the TQE is in use, a TTIMER macro instruction is used to cancel the timer. If a job-step task is terminating, ABEND closes all data sets whose DCBs are queued to the TCB, purges any unended MLWTO and WTOR requests, and dequeues all IQEs associated with the task. All main storage within the partition, except that for a dummy program, is freed.

ABEND returns control to the job management Step Deletion routines GO (IEFSD515) or SMALLGO (IEFSD599) by constructing a dummy program at the beginning of the partition. This dummy program contains an LPSW instruction followed by an XCTL macro instruction specifying the name of the job management routine that is to receive control. The dummy program's PRB is placed at

the beginning of the RB queue on the TCB, and ABEND branches to the dummy program.

If the terminating task is the Online Test Executive Program (OLTEP) and the software and/or hardware involved in testing has not been reset, ABEND issues an SVC 59 for performance of the cleanup functions necessary for system restoration. (See Online Test Executive Program.)

If the terminating task is a subtask, the IQEs for the task are purged, open data sets are closed, all gotten subtask area queue elements are dequeued and the area they describe is freed, and all request blocks, except the SVRB for ABEND, are removed from the active RB queue. ABEND sets the current task nondispatchable and issues an SVC 3 (EXIT) instruction.

Abnormal Termination

ABEND first determines whether SVC DUMP was in progress when the failure occurred. If so, ABEND clears the CVT lock byte, sets all tasks dispatchable, and reactivates the trace table or GTF.

ABEND next tests for a Damage Assessment Routine recursion and, if found, exits to the DAR service routines. If a subtask is terminating and an invalid recursion has occurred, ABTERM is used to terminate the job step and a WAIT macro instruction is issued to allow immediate termination of the job step.

If a CLOSE, OPEN, ABDUMP, or message recursion (WTO failure) has occurred, ABEND halts all I/O operations for this task and dequeues all request blocks created as a result of the recursion. Processing continues with the test for dump (below).

If this is a nonrecursive termination, ABEND determines if STAE processing should be initiated. If so, ABEND exits to the ABEND/STAE interface routines. If ABEND was invoked because the purge failed during STAE processing, this same exit is taken. If no STAE processing is indicated, ABEND purges IQEs, active TQEs, and transient area requests for the terminaing task and any of its subtasks. Unended MLWTO and WTOR requests are purged, I/O is halted, and control blocks are checked to ensure validity.

ABEND determines if a dump is requested and what type of dump is wanted. It ensures that sufficient main storage is available for the dump by using a conditional GETMAIN macro instruction. If necessary, the programs described by the load list and active RB chain are freed to obtain this main storage. ABEND searches the TIOT for a SYSABEND or a SYSUDUMP DD name. If neither are found and the terminating task is a job-step task, ABEND stores pertinent information in main storage for eventual printing by the job management routines. This is called an indicative dump. (For a detailed description of the indicative dump, see the Programmer's Guide to Debugging.) If a SYSABEND entry is found as the last entry in the TIOT, ABEND opens a DCB for the dump data set if necessary and calls ABDUMP to dump the nucleus, the system queue area, and that area within the task's partition that is not free. If a SYSUDUMP entry is found, ABEND opens a DCB for the dump data set if necessary and calls ABDUMP to dump only that area within the task's partition that is not free.

Upon completion of the dump, or if no dump was given, ABEND determines the type of task that is terminating. If the task is a job-step task with no subtasks, ABEND closes all data sets associated with the task, dequeues IQEs, and clears the pointers to the ABEND appendages and transient area queues. ABEND exits to the job management routines to print the indicative dump if necessary and to initiate the next task.

If the terminating task is itself a subtask, ABEND frees IQEs for the terminating task and any subtasks, chains all subtask data sets to the TCB of the current task, dequeues subtask TCBs from the ready queue, and frees the associated main storage.

If the terminating task is a job step task with subtasks having no outstanding ENQ requests, ABEND purges resources for its subtasks, and processing continues as for a job step without subtasks (above).

If the terminating task is a job-step task with subtasks having outstanding ENQ requests, ABEND dequeues them.

If the terminating task is the Online Test Executive Program (OLTEP) and the software and/or hardware involved in testing has not been reset, ABEND issues an SVC 59 for performance of the cleanup functions necessary for system restoration. (See Online Test Executive Program.)

In all cases, the Message Information List that is maintained in the nucleus and in which Type-1 SVCs can store data about the cause of an abnormal termination of a Type-1 SVC is searched for a matching TCB. If an entry is found, ABEND writes out the message.

Data sets queued to the TCB of the terminating task are closed, subtask resources are purged, and the request block queue from the TCB is terminated leaving only the

SVRB for ABEND. The subtask is set nondispatchable, and ABEND issues an SVC 3 (EXIT) instruction.

STAE SCHEDULING AND PROCESSING

The STAE routine is a type 3 SVC routine that prepares a user-specified exit routine to intercept abnormal termination processing. When the user's task becomes scheduled for abnormal termination, the ABEND/STAE interface routine (ASIR) is given control by ABEND. ASIR purges all active I/O for the task (unless PURGE=NONE) and gives control to the user-provided exit routine. Upon regaining control, ASIR returns control to ABEND for completion of termination, unless the user has specified that a STAE Retry routine is to be scheduled.

If retry is to be scheduled with a purge of the RB chain, all RBs between the RB associated with the current STAE and the RBassociated with ABEND are purged. All data sets related to the purged RBs are closed, the resume PSW of the RB using STAE is set to point to the retry routine, and control is passed to the Dispatcher to initiate the retry.

For a retry with no purge (system tasks only), an RB is created for the Retry routine and control is passed to the dispatcher to initiate the retry.

DAMAGE ASSESSMENT

The Damage Assessment Routine (DAR) is invoked by ABEND to attempt to recover from the following:

- Failures within system tasks.
- Failures within tasks in the "must complete" status.
- Invalid recursions within the termination process.

DAR 1 (IEADTM22) provides a dump of main storage (as it was at the time of the failure) to the SYS1.DUMP data set by issuing an SVC 51 instruction. DAR 2 (IEADTM23) attempts to reinstate failing system tasks that are not in "must complete" status. DAR 3 (IEADTM24) attempts to reinstate all other tasks. If the task is in the "must complete" status and resources are not critical, a task reinstatement message is issued and control is passed to ABEND 3 to continue termination processing. If resources are critical, or if the entry to DAR 3 is a recursion, DAR 4 sets the task nondispatchable and control is given to the Dispatcher for the next highest priority ready task. Messages are written to the console to inform the operator of the status of DAR at various points during its operation.

DUMPING

The ABDUMP routine (SVC 51) produces a dump of system and problem-program information. ABEND or a user may invoke ABDUMP by issuing a SNAP macro instruction to dump the major control blocks, save areas, and main storage. If Main Storage Hierarchy Support is in the system, ABDUMP dumps the main storage hierarchies (0 or 1 or both) associated with the calling task.

The services of contents supervision are loading a program into main storage, scheduling its execution, and deleting it from main storage. Contents supervision also keeps records of all programs that reside in main storage so that access may be gained to any program.

CONTENTS SUPERVISION ROUTINES

The contents supervision routines record the identity, main-storage location, size, and attributes of programs as they are brought into main storage. Programs that are no longer needed are released automatically by the EXIT routine (see "Interruption Supervision") or, if brought into main storage by a LOAD macro instruction, are released by a DELETE macro instruction. Contents supervision routines maintain queues of request blocks to identify the different types of routines and by which tasks they may be accessed. Following are the contents supervision routines:

ATTACH (MFT without subtasking): In systems without subtasking, the ATTACH routine provides compatibility with systems with the subtasking option. The contents supervision ATTACH routine provides linkage to a specified program (see LINK), but does not create a new TCB nor does it allow for asynchronous processing.

LINK: This routine provides linkage from the routine that issued the LINK macro instruction to another routine. When the called routine finishes processing, control is returned by the Dispatcher to the next sequential instruction following the LINK macro instruction in the calling routine.

LOAD: This routine ensures that a copy of a specified routine exists in main storage and returns the address of the routine to the requester.

XCTL: This routine passes control from the caller to the requested routine. The calling routine is released from main storage and will not regain control.

FINCH: This routine is used by the supervisor to schedule the loading of programs into main storage. It also contains the Transient Area Loading Task.

IDENTIFY: This routine allows the caller to dynamically specify an entry point name in a load module so that the caller may issue LINK and XCTL macro instruction to that identified entry point.

DELETE: This routine releases modules from main storage which were brought in by LOAD.

SYNCH: This routine passes control from the control program in supervisor state to a processing program in main storage, and provides return linkage to the control program in supervisor state.

CONTENTS SUPERVISION FUNCTIONS

The contents supervision routines keep records about program segments that are loaded, branched to, and released from main storage. Following are the functions of contents supervision:

- Creating and maintaining request blocks.
- Servicing requests for supervisor-assisted linkage.
- Bringing modules into main storage.
- Bringing modules into the SVC transient area.
- Identifying embedded entry points.
- Deleting a loaded module.
- Synchronizing control program and problem program execution.

This section discusses contents supervision by functional areas. For a more detailed description of the contents supervision modules, see Section 3, "Program Organization."

CREATING AND MAINTAINING REQUEST BLOCKS

Contents supervision creates a request block each time that a module is brought into main storage. Additional requests blocks are created for a module if a user requests a reenterable module that is being used by another program. Request blocks contain information required to schedule, execute, and terminate the program with which they are associated. (See Section 5 for the format of the request blocks.) There are seven types of request blocks in MFT:

- Program Request Block (PRB) -- represents a non-supervisory routine that must be executed in the performance of a task. PRBs are created by LINK, SYNCH, and XCTL.

- Supervisor Request Block (SVRB) -- represents a supervisor routine to be executed at the request of a system or user task. SVRBs are created by the Supervisor Call Second-Level Interruption Handler (SVC SLIH) for its own use and at the request of the Dispatcher.

- Interruption Request Block (IRB) -- represents a routine that must be executed in the event of an asynchronous interruption. IRBs are created before the interruption occurs, but are not placed on a queue until after the interruption occurs.

- System Interruption Request Block (SIRB) -- is used only for the system I/O error task. There is only one SIRB in the system, and is always fixed in the nucleus regardless of whether it is queued to a TCB.

- Loaded Program Request Block (LPRB) -- represents programs brought into main storage by LOAD. An LPRB is designated as a "dummy" when it is queued to the Loaded Program List or Job Pack Area Queue and is a copy of an LPRB that represents a module in the Resident Reenterable Module Area. An LPRB is designated as "minor" when it represents a control section of a program specified by an IDENTIFY macro instruction. LPRBs are created by LOAD, LINK, XCTL, and IDENTIFY.

- Loaded Request Block (LRB) -- represents modules that have the "load only" attribute. These modules can gain control only by a branch, not a LINK, XCTL, or ATTACH macro instruction. LRBs are created by LOAD.

- FINCH Request Block (FRB, present only in MFT with subtasking) -- represents programs that are in the process of being loaded by LOAD. An FRB is created upon entry to the LOAD routine, and is deleted upon completion of the LOAD processing.

The Request Block Queues

The supervisor maintains the attributes of and relationships between modules by placing their associated request blocks on appropriate queues. Maintaining these RBs facilitates loading, locating, passing control to, and deleting the modules. Thus, related modules will be executed in proper sequence, and control passed correctly between them. Normally, a request block is physically located immediately before its module in main storage. Logically, a request block is located on one or more of the following queues:

Active Request Block Queue: There is one active RB queue for each job step or subtask task control block (TCB) in the system. The first (top) RB on the queue is pointed to by the TCBRBP field in the TCB, and represents the module that is currently being executed for the task. This queue may have PRBs, IRBs, SVRBs, LPRBs, and the SIRB on it. (In MFT with subtasking, the SIRB is queued only to the System Error Task TCB. In MFT without subtasking, the SIRB will be queued to the TCB representing the task that had an I/O error.) At any point during execution of the task, the active RB queue will have on it an RB for each module that has not completed execution for the task. The relationship between RBs on this queue is maintained through the use of the XRBLNK field in each RB. Request blocks are added to the top of the queue so that each XRBLNK field points to the RB for the module that will receive control when this module has finished processing. Where LINK or ATTACH routines are used, processing occurs in a first-in, last-out manner (that is, the RB for the issuing program is pushed down on the queue). Where an XCTL macro instruction is used, the RB for the issuing program is removed from the queue so that the issuing program cannot regain control.

Loaded Program List: This queue also originates in the TCB for the job step or subtask and is pointed to by the TCBLLS field. The order of RBs on this queue is the order in which the modules were brought into main storage. The Loaded Program List contains LRBs, LPRBs, dummy LPRBs, and all minor LPRBs. All modules which are represented on this queue are assumed to be serially reusable and may be used only by the task in whose TCB the list originates. In MFT without subtasking, all modules that are brought into main storage by LOAD will have an associated LRB or LPRB on this queue. In MFT with subtasking, only those modules that are not reenterable are represented on this queue (by LPRBs). Because modules represented by LRBs are always reenterable, they will be represented on the Job Pack Area Queue, not the Loaded Program List.

Job Pack Area Queue (MFT with subtasking only): This queue is similar to the Loaded Program List in that only modules brought into main storage by LOAD are represented. However, the LRBs and LPRBs represent only those modules that are reenterable. Minor LPRBs are never placed on this queue. FRBs are found only on this queue, and they exist only while the module is being loaded

into main storage. The Job Pack Area Queue is pointed to by the Partition Information Block (PIB+X'2D') which is pointed to by each TCB in the partition. Modules represented on this queue may be used by any task in the partition.

Resident Reenterable Module Area Queue (optional): If this option is included in the system, the Nucleus Initialization Program identifies those programs that are permanently resident in main storage by creating a queue of LPRBs. This queue is pointed to by the CVTLPAQ field in the Communication Vector Table. Modules on this queue are always reenterable and may be used by any task in the system. The use of any of these modules results in the creation of a dummy LPRB which, for LINK, ATTACH, or XCTL requests, is queued to the Active Request Block Queue of the TCB for the requesting task or, for a LOAD request, to the Job Pack Area Queue.

Resident SVC Queue (optional): This queue contains LPRBs for SVC modules that have been made resident by the use of the RSVC option. The queue is pointed to by IEAARSV1 within module IEAATC00.

Resident Access Methods Queue (optional): This queue contains LPRBs for access method modules that have been made resident by use of the RAM option. The queue is pointed to by IEAARAM4 within module IEAATC00.

SERVICING REQUESTS FOR SUPERVISOR-ASSISTED LINKAGE

Contents supervision services requests for supervisor-assisted linkage by:

- Searching for the requested module.
- Determining whether the module is available.
- Bringing the module into main storage.
- Deferring a request for the module.
- Scheduling execution of the module.

Initialization and input processing are dependent upon the type of request. The requests are serviced by the ATTACH, LINK, LOAD, XCTL, and FINCH service routines.

Searching for a Requested Module

Upon receiving a request for supervisor-assisted linkage, contents supervision searches for the specified module in the Job Pack Area Queue (applicable only to MFT systems with subtasking), the Loaded Program List, and the Resident Reenterable Module Area (if included in the system). The search of these queues is terminated when a request block representing an available copy of the module is found.

Determining the Module's Availability

A module is available if it is in main storage and not in use. If an available copy is found, control is passed to the module for the XCTL, LINK, and ATTACH functions, or the address of the module is returned to the requester by the LOAD function. If the requested module is not found, or is found but is in use, another copy of the module is brought into main storage for that request. In MFT with subtasking, the request for a module may be deferred if another request has been made for the module, but the module has not been brought into main storage (see "Deferring a Request for a Module").

Bringing A Module Into Main Storage

If, in response to a LINK, LOAD, XCTL, or ATTACH request, the supervisor does not find a usable copy of the module in main storage, the subroutine FINCH is entered to prepare for the loading of the module by Program Fetch. The FINCH subroutine uses the directory entry supplied by the caller or obtained by the data management BLDL routine to determine the amount of main storage needed to contain the module and its request block. FINCH obtains that amount of main storage in the appropriate location and uses Program Fetch to bring the module into main storage or brings it into the transient area. FINCH initializes the request block and returns control to the routine that branched to FINCH.

Use of the "PANIC", a Fixed Work Area in FINCH

Program Fetch requires that FINCH provide a parameter list containing information necessary to bring the requested module into main storage. FINCH issues a conditional SVC 4 (GETMAIN) instruction to obtain main storage for the parameter list. If sufficient main storage is not available and the request is to load a module into the SVC or I/O error transient areas, FINCH uses a fixed-length work area called the PANIC. When FINCH processes a request to load a module from the SYS1.SVCLIB, the PANIC is always used.

While the PANIC is being used, the system is "pseudo-disabled," preventing entry to the Dispatcher (see "Pseudo-disabled Interruption Supervision" in Section 2). When Program Fetch returns control to FINCH, the system is pseudo-enabled, allowing use of the PANIC during the processing of subsequent requests.

Deferring a Request for a Module

In MFT with subtasking, a LOAD macro instruction requires special processing. When LOAD determines that the requested module is not in main storage, it creates a FINCH Request Block (FRB) and places it on the Job Pack Area Queue to indicate that the module is being loaded. If another request for the same module is received while the module is being loaded to satisfy the first request, a second copy of the same module is not loaded. Instead, LOAD finds the FRB for the module on the Job Pack Area Queue, creates a Wait List Element (WLE), and queues it on the FRB. The routine that issued the second request is placed in a wait state pending completion of the loading. When the module has been completely loaded into main storage, the FRB is removed from the Job Pack Area Queue, the request block for the module is placed on the appropriate queue, and all requests represented by WLEs are set dispatchable. LOAD is reentered for the second request and the Job Pack Area Queue is searched again. If the loaded module is reenterable, it is represented on the Job Pack Area Queue by an LPRB. LOAD returns the address of the routine to the requester. If the loaded module is not reenterable (not queued on the Job Pack Area Queue), an FRB is again created and queued to the Job Pack Area Queue, and a new copy of the module is brought into main storage.

Scheduling the Execution of a Module

The supervisor schedules the execution of a requested program for ATTACH, LINK, and XCTL macro instructions by placing the request block of the module on the top of the active request block queue. The Dispatcher gives control to the module when its RB is the top RB on the queue and its TCB represents the highest priority ready task. When the module exits, its RB is removed from the queue. If the module was invoked by an ATTACH or LINK macro instruction, control returns to the caller at the instruction following the LINK or ATTACH macro instruction. If invoked by an XCTL macro instruction, control does not return to the caller because its RB was removed from the active RB queue by XCTL. Figure 15 illustrates the active RB queue after Program A has issued LOAD and LINK macro instructions, each specifying the nonreenterable Program B.

Special ATTACH Processing

Although the ATTACH routine (MFT without subtasking) does not create a new task, it must perform certain additional functions to be compatible with the subtasking ATTACH routine. If the user specified an ECB parameter, ATTACH posts the ECB to indicate

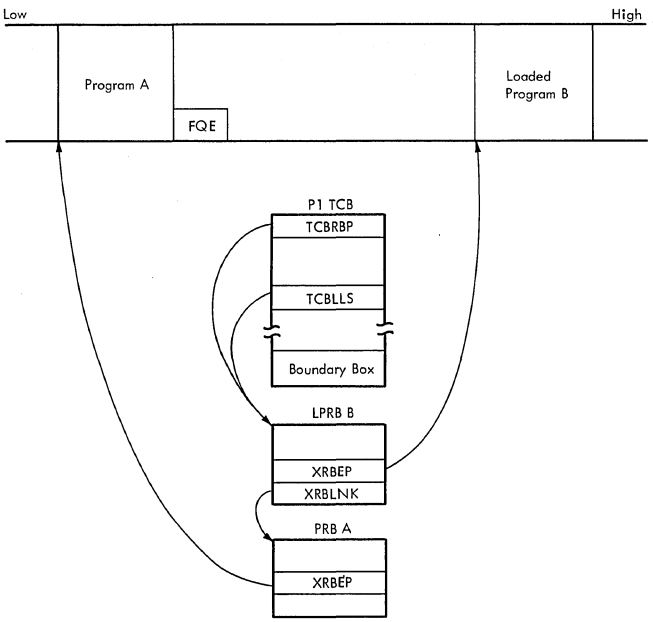


Figure 15. Control Block Queue after a Load and Link

the completion of the requested routine. If the user specified an end-of-task exit routine (ETXR), ATTACH builds and initializes an interruption request block (IRB) and queues it between the RB for the requesting program and the SVRB for ATTACH. When the attached routine exits, the end-of-task exit routine gains control.

THE TRANSIENT AREAS

The transient areas are two 1024-byte areas within module IEAATC00 (LINK, LOAD, XCTL, FINCH) into which I/O error handling and non-resident SVC routines are temporarily loaded for execution. These areas provide economy of main storage for routines that are less frequently executed than other control program routines.

Fetching a Module into the SVC Transient Areas

The SVC transient area is loaded by the FINCH routine and the transient area loading task. FINCH creates and initializes the parameter list necessary for the transient area loading task, places the requesting SVRB on the transient area request queue, and sets that SVRB to a wait condition. If the transient area is not currently being loaded, FINCH indicates to the transient area loading task the routine to be loaded, removes the SVRB for the transient area loading task from the wait condition, and branches to the Dispatcher. If the transient area is being loaded, FINCH

does not indicate that this routine is to be loaded, but branches to the Dispatcher.

The transient area loading task is the highest priority task in the system. Its task control block (TCB) is created at system generation with an SVRB in a wait condition as the only request block on its RB queue. When the SVRB is removed from the wait condition by FINCH, the transient area loading task gains control from the dispatcher. Although the loading of a module into the transient area may be interrupted, the module will be completely loaded before any other, higher priority request for the transient area will be performed. When the module is completely loaded, the transient area loading task removes all SVRBs from the transient area request queue, removes these SVRBs from the wait condition, and exits to the dispatcher.

The transient area request queue, which originates in FINCH, is a chain of SVRBs representing requests for nonresident SVC routines that are to be loaded into and executed in the transient area. The chain is created and maintained by FINCH and the transient area loading task using the two fullword pointers in the boundary boxes associated with the TCBs. (Figure 16 illustrates multiple requests for the transient area. See Section 5 for the format of the boundary box.) The fullword at boundary box-16 is a pointer to the boundary box (-16) for the previous outstanding request for the transient area. The fullword at boundary box-12 is a pointer to the SVRB for the last request for the transient area for that task. The transient area queue exists only while a module is being loaded; the transient area loading task purges the queue so that the SVRBs may contend for the transient area task by priority.

It is possible for a task to have a request on the queue waiting for the transient area to be loaded and yet issue another request for loading the transient area. An example of this is an Interruption Request Block (IRB) which is scheduled for execution and is queued immediately following the SVRB waiting on the transient area. If the program represented by the IRB issues a request for a type 3 or 4 nonresident SVC routine, the first request is dequeued and taken out of the wait state so that the most recent request will be queued and put into the wait. The previous requesting SVRB is no longer the active RB and will not regain control until the next request has finished using the transient area.

IDENTIFYING EMBEDDED ENTRY POINTS

The IDENTIFY function is an optional supervisory feature that may be selected during system generation. If this option is not selected, the IDENTIFY is treated as a NOP instruction.

The IDENTIFY routine IEAAID00 builds a minor LPRB describing the embedded entry point of a routine that already resides in main storage. IDENTIFY places this RB in the Loaded Program List so that the embedded routine can be found by the supervisor and can then be accessed using supervisor-assisted linkages (LINK and XCTL).

DELETING A LOADED MODULE

The DELETE routine removes modules that have been brought into main storage by a LOAD macro instruction. DELETE searches the Resident Access Method (RAM) List (optional), the Job Pack Area Queue (MFT with subtasking only), and the Loaded Program List for the request block representing the module to be deleted. If the request block is not found, DELETE indicates this by a return code and issues an SVC 3 (EXIT) instruction. If the RB is found on the RAM list, DELETE returns via a BR 14. If the request block is found elsewhere, its use count is decremented. If the use count is 0 and the module is inactive, DELETE frees the area of the module and its request block. The DELETE routine then issues an SVC 3 (EXIT) instruction.

SYNCHRONIZING CONTROL PROGRAM AND PROBLEM PROGRAM EXECUTION

When the control program executing in supervisor state is to give control to a problem program (executing in nonsupervisor state), and control is to return to the control program for further execution in the supervisor state, the control program issues a SYNCH macro instruction. This expands to include an SVC 12 instruction. The SYNCH routine builds a program request block in the lower area of the partition and places the PRB on the active RB queue immediately after the SVRB for SYNCH. SYNCH issues an SVC 3 (EXIT) instruction.

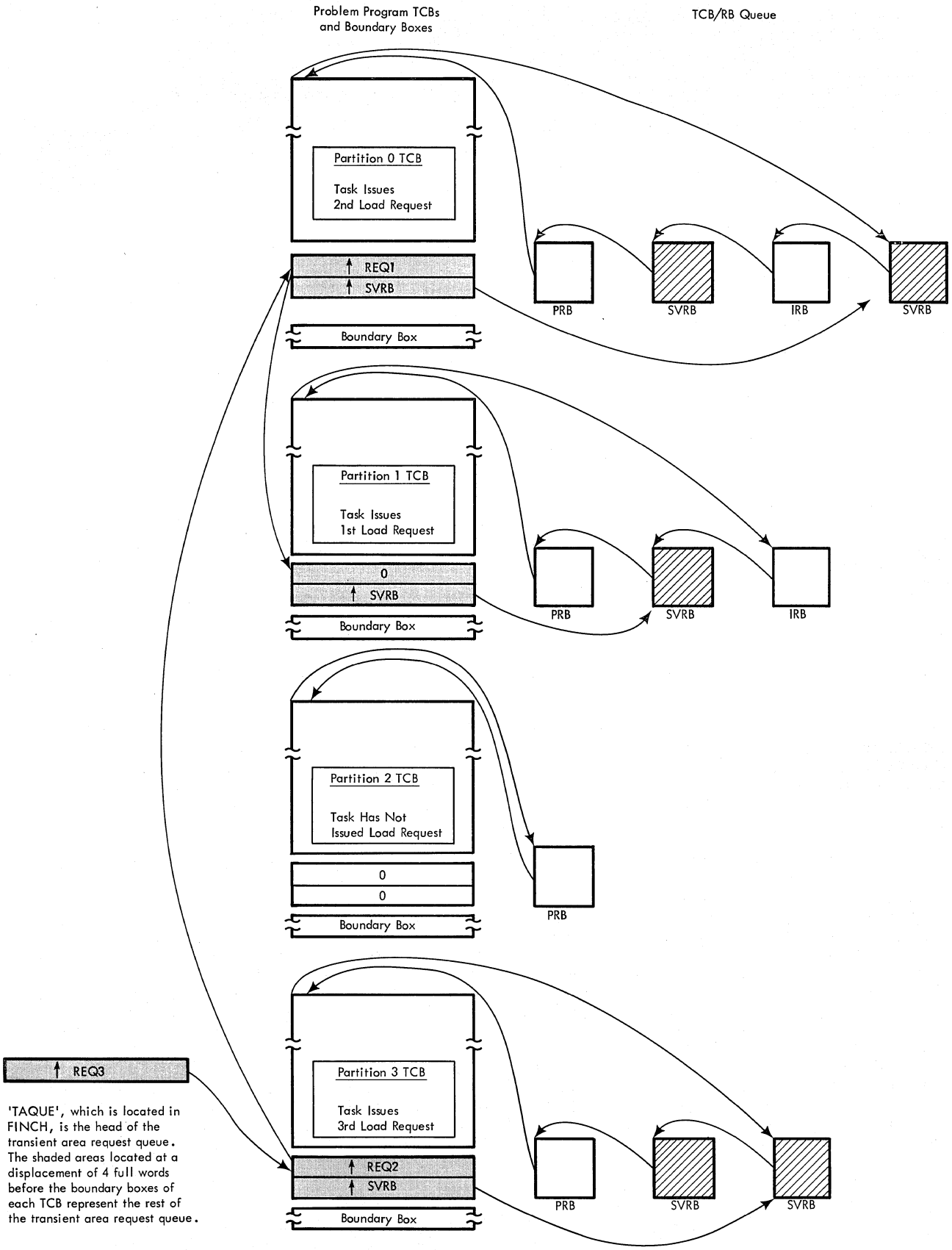


Figure 16. Transient Area Request Queue and TCB/RCB Queue

The Main Storage Supervision routine (GETMAIN/FREEMAIN) controls the allocation and release of all dynamic main storage and of the system queue area (SQA).

MAIN STORAGE

Each partition in the dynamic area has an associated TCB which contains a pointer to the main storage boundary box for that partition. This box defines the limits of each partition. (See Section 5 for the format of the boundary box.)

Dynamic main storage may be expanded by including IBM 2361 Core Storage in the system. If 2361 Core Storage and its programming support (Main Storage Hierarchy Support) are included in the system, the dynamic area is divided into two parts: processor storage (hierarchy 0) and 2361 Core Storage (hierarchy 1). The request for main storage allocation may specify either hierarchy. If space is not available in hierarchy 1, it will be allocated from hierarchy 0, regardless of the hierarchy specified in the GETMAIN macro instruction.

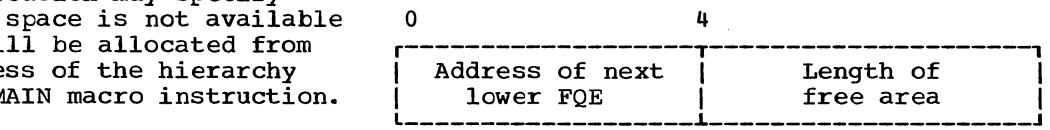
The routine IEAAMS, in response to a GETMAIN macro instruction, obtains storage from either the problem program partition or, if a system task issued a GETMAIN macro instruction specifying subpool 245 or 255, from the SQA. This area is used for system control blocks that might be destroyed by problem programs if they were placed in problem partitions. The boundary box for the SQA is in the master scheduler resident data area (offset X'64'). This area is addressed by the CVTMSER field in the Communications Vector Table.

The system tasks that request space from the SQA are:

- The CSCB (Command Scheduling Control Block) creation module of SVC 34 for CSCBs.
- The task supervision ATTACH routine for subtask TCBs, if the subtasking option is included in the system.
- The ENQ/DEQ processing routine of task supervision, for all control blocks associated with ENQ/DEQ.
- The communications task, for write-to-operator (WTO) buffers if no WTO buffer storage space specified during system generation is available.

GETMAIN

All requests for space within dynamic storage or the SQA are handled by GETMAIN. To find an area of free main storage large enough to fulfill the request, GETMAIN searches the Free Area Queue in the partition issuing the request. Each Free Area Queue Element (FQE) occupies the first two fullwords in the free area described and contains a pointer to the FQE describing the next lower free area within the partition or 0's if it describes the lowest free area. The second word contains the length in bytes of the free area it describes. The boundary box for the partition points to the FQE for the highest free area. Each FQE begins and ends on a doubleword boundary, and requests for main storage are always rounded-up to a doubleword boundary. If the address, after rounding-up, does not fit in three bytes, the requester will be abnormally terminated.



Free Area Queue Element (FQE)

Although subpools are not created in MFT, tasks may specify subpool numbers in the GETMAIN macro instruction. When a problem program issues a GETMAIN macro instruction specifying a subpool from 0 through 127, storage is allocated from the high-address portion of the partition in which the GETMAIN macro instruction was issued. When a problem program attempts to issue a GETMAIN macro instruction specifying any other subpool, the program is abnormally terminated.

When a system task issues a GETMAIN macro instruction specifying a subpool from 0 through 127, storage is allocated from the low-address protion of the partition; when the macro instruction specifies a subpool from 128 through 244 or 246 through 254, storage is allocated from the high-address portion of the partition. Subpools 245 and 255 are handled as a special case as described above.

The GETMAIN macro instruction can either be conditional or unconditional. If an unconditional request cannot be satisfied, the requesting task is abnormally terminated. However, before scheduling the task for termination, GETMAIN determines if

ABEND or the SVC SLIH was the requester. If so, abnormal termination is not scheduled. Instead an error code is returned to the caller. This action prevents a loop that could be caused by rescheduling an abnormal termination for a task.

If space cannot be found to satisfy a conditional request, a return code is placed in register 15, and control is returned to the caller.

If enough space is found, the address of the allocated space is returned to the caller. If the space allocated is less than that described by the FQE, the length indicator in the FQE is adjusted to reflect the new size of the free area. If the entire free space is allocated, the FQE is removed from the queue.

Processing with the Subtasking Option

If the subtasking option is included in the system, the GETMAIN routine creates a gotten subtask area queue to describe the main storage obtained for a subtask by a system-issued GETMAIN macro instruction. For example, if a subtask issues a LINK macro instruction for a module that is not in main storage, LINK processing in turn issues a GETMAIN macro instruction for space in the partition within which the requested module will execute. GETMAIN adds eight bytes to the requested size, creates a gotten subtask area queue element (GQE) within those eight bytes, queues the GQE, and returns the address of the available area (GQE+8) to the requesting routine. The first word of a GQE is the address of the previously created GQE or 0 if this is the first GQE, and the second word is the length of the allocated area. The queue of GQEs originates in the first word of the boundary box pointed to by the TCBMSS field in the subtask TCB. GQEs are added to the top of the queue as they are created. The order of GQEs indicates only the order in which main storage was obtained for the subtask, not the relative physical locations of their associated areas within the partition. The GQE on the bottom of the queue is always associated with the register save area obtained for the subtask. The next to last GQE will describe the area in which the subtask itself is executing, providing the subtask was not resident and inactive in main storage when the ATTACH macro instruction was issued.

Special ATTACH Processing

When the ATTACH routine issues a GETMAIN macro instruction, it specifies a length 8 bytes greater than the size of the module. In this case, ATTACH, not GETMAIN, creates the GQE and places it on the subtask's GQE queue.

FREEMAIN

All requests to release space in the dynamic areas or in the SQA are handled by FREEMAIN. FREEMAIN makes the space previously allocated by GETMAIN available for reallocation by modifying an existing FQE, or by creating a new FQE and adding it to the Free Area Queue.

When a subtask terminates, the normal or abnormal termination routines refer to the GQEs chained to the subtask's TCB in order to free the main storage belonging to the terminating subtask. The GQEs are removed from the queue whenever the subtask storage space is freed.

The communications task processes input and output between the central processing unit and one or more devices. Input results from an unplanned interruption from an external device or from the main console; output results from the WTO (Write to Operator) and WTOR (Write to Operator with Reply) macro instructions.

INPUT PROCESSING

The operator causes an I/O interruption by pressing the REQUEST key or START key on an I/O device. The I/O First-Level Interruption Handler passes control to the I/O Supervisor which determines that an I/O interruption from the operator has occurred. Control then passes to the resident I/O Attention Handler (IEECVCRA) which posts the attention ECB in the Unit Control Module.

When the operator presses the INTERRUPT key on an operator control panel, he causes an external interruption. In this case, control passes from the External First-Level Interruption Handler to the resident communication task External Interruption Handler (IEECVCRX), which posts the external ECB in the Unit Control module. Diagram 14 illustrates input processing.

OUTPUT PROCESSING

Console output is initiated when a user or system program issues a WTO or WTOR macro instruction. These macro instructions issue an SVC 35 instruction which causes the nonresident module IEENFWTO (or IEEMFWTO with MCS) to be brought into the SVC transient area. This module moves the message into a Write Queue Element (WQE), places the WQE on the system output queue, and posts the WTO ECB. If a WTOR macro instruction is being processed, IEENFWTO passes control to the nonresident module IEEVWTOR to create and queue to the Unit Control Module a Reply Queue Element before moving the message text to the WQE. If routing code 11 is specified, IEENFWTO passes control to the nonresident Write-to-Programmer module, IEEWTP00, before moving the message text. If this is a multiple-line WTO (MLWTO), control is passed to IEECVML1 (non-MCS) or IEECVML3 (MCS). Diagram 15 illustrates output processing.

Note: If routing code 11 is the only routing code specified for a WTO macro instruction, the message will not appear at a console, unless a console was specified at system generation to accept routing code 11.

COMMUNICATION TASK PROCESSING

The communication task executes under its own TCB and is composed of several nonresident modules and the following three resident modules: the Initialization module, IEECVCTI, which creates control blocks and initializes the Unit Control Module at NIP; the nonexecutable Unit Control Module, IEECVUCM, which contains storage areas, pointers, and the five event control blocks; and the communications task Wait module, IEECVCTW. The Wait module issues a WAIT macro instruction specifying a list of addresses (Event Indication List) of the external ECB, the attention ECB, the WTO ECB, RMS ECB, and I/O completion ECBs. When one of these ECBs is posted by initial input or output processing, the communication task becomes a ready task and receives control when it becomes the highest priority task in the system. The Wait module issues an SVC 72 instruction, which causes an SVRB to be built for the execution of nonresident modules, to pass control to the nonresident Router. Using the ECBs in the Unit Control Module and the I/O ECBs pointed to by the EIL, the Router determines the cause of entry to the communications task and passes control to a nonresident module to service the interruption.

External interruptions are serviced by the external interruption processor module, IEECVCTX, which performs the console switch. I/O, attention, and WTO interruptions are serviced by the device processor module which execute channel programs to read from or write to a device (or both), and the open/close device support routines.

MULTIPLE CONSOLE COMMUNICATIONS

The Multiple Console Support (MCS) routines handle I/O for up to 32 system operator consoles. Input follows from an attention (I/O interruption) from an active console. Output results from a WTO or WTOR macro instruction in a problem program or in a system task. MCS also handles external interruptions from the operator control panel, I/O complete conditions, Delete Operator Message (DOM) macro instructions, console switching, and system and console output queue management.

As in systems without MCS, control is routed by the External First-Level Interruption Handler for external interruptions, and by the I/O First-Level Interruption Handler for console device attentions and I/O complete interruptions (see Diagram 16). WTO, WTOR, and DOM macro instructions are handled by SVC 35 and SVC 87 respectively. The routing results in the posting of one of the 5 ECBs (RMS, external, attention, WTO, DOM) in the Unit Control Module (UCM), or the posting of one of the I/O ECBs pointed to by the Event Indication List (EIL) within the UCM. The communications task TCB, created within the nucleus at system generation, is then indicated as ready. The Dispatcher passes control to the communications task when its TCB is the highest priority TCB on the queue.

In addition to the ECBs and the pointer to the EIL, the UCM contains pointers to the system output queue, the reply queue elements, and the UCM entries (see Figure 17). At system generation, a UCM entry is constructed for each console device specified by the SCHEDULR and SECONSLE macro instructions (a composite console will have 2 UCM entries). Each UCM entry contains pointers to the processor module for that device, to the console output queue, and to the alternate console, and contains the routing codes and command authority codes assigned to the device. The console output queue is a series of pointers to selected WQEs on the system output queue. The first byte of each pointer indicates the status of the corresponding WQE in relation to the device.

The communication task with MCS uses the non-MCS Initialization module, IEECVCTI; the Unit Control module, IEECVUCM; the Attention Handling module, IEECVCRA; the External Interruption Handling module, IEECVCRX; and the following MCS modules:

- Mini-Router (IEECMCTR), which receives control as a result of an SVC 72 instruction issued by IEECMDSV, IEECMAWR, IEEC2740, or IEECMDOM. This SVC 72 causes an SVRB to be built for the execution of the device processors and other nonresident communications task modules.

- Router (IEECMAWR), which receives control when one of the ECBs in the UCM is posted. It passes control to other communication task modules to service the interruptions and for queue management.

- Console Switch (IEECLCTX, IEECMCTX, IEECNCTX, and IEECOCTX), which switches consoles as a result of an external interruption, an unrecoverable I/O error, or a VARY command.

- Device Service (IEECMDSV), which passes control through the Mini-Router to appropriate device processor modules when there is I/O to be performed, or consolidates system and console output queues.

- WTO Service (IEECMWSV), which queues WQEs to console output queues.

- DOM Service (IEECMDOM), which indicates that specified operator messages can be deleted.

- NIP Message Buffer Writer (IEECMWTL), which writes messages from the buffer created by the Nucleus Initialization Program.

THE SYSTEM LOG FUNCTION

In systems that include Multiple Console Support (MCS), a hard copy of all operator and system messages is required when there is an active graphic console or more than one active non-graphic console. Because of this requirement, a system log function is provided which may be specified as the hard copy log. In MFT, the System Log operates under its own TCB created at system generation.

The system log consists of two data sets cataloged at system generation on which critical system information is recorded. The existence of two data sets allows one (the primary data set) to receive data from the system, problem programs and/or operators, while the other (the alternate data set) is being written to an output device.

The log is initialized by the routines IEEVLIN (see Diagram 17), which locates the log data sets and establishes the Log Control Area and log buffers, and IEELOG02, which writes the log JFCBs to the job queue, creates log DCBs, attaches the Log Writer routine (IEELWAIT), and posts the log ECB. If the log data sets are not located by IEEVLIN, the operator is notified that the log option is not supported. Consequently, WTL macro instructions are reissued as WTO macro instructions to the primary/master console. LOG and WRITELOG operator commands are treated as NOPs by the control program and a message is sent to the issuing console informing the operator that the system log is not supported. IEEVLIN returns control to the Master Scheduler Initialization routine, IEFSD569.

Users communicate with the log through the WTL macro instruction and the operator commands LOG and WRITELOG. The WTL routines (SVC 36) schedule the entering of information into the system log. The LOG command is used to enter information into

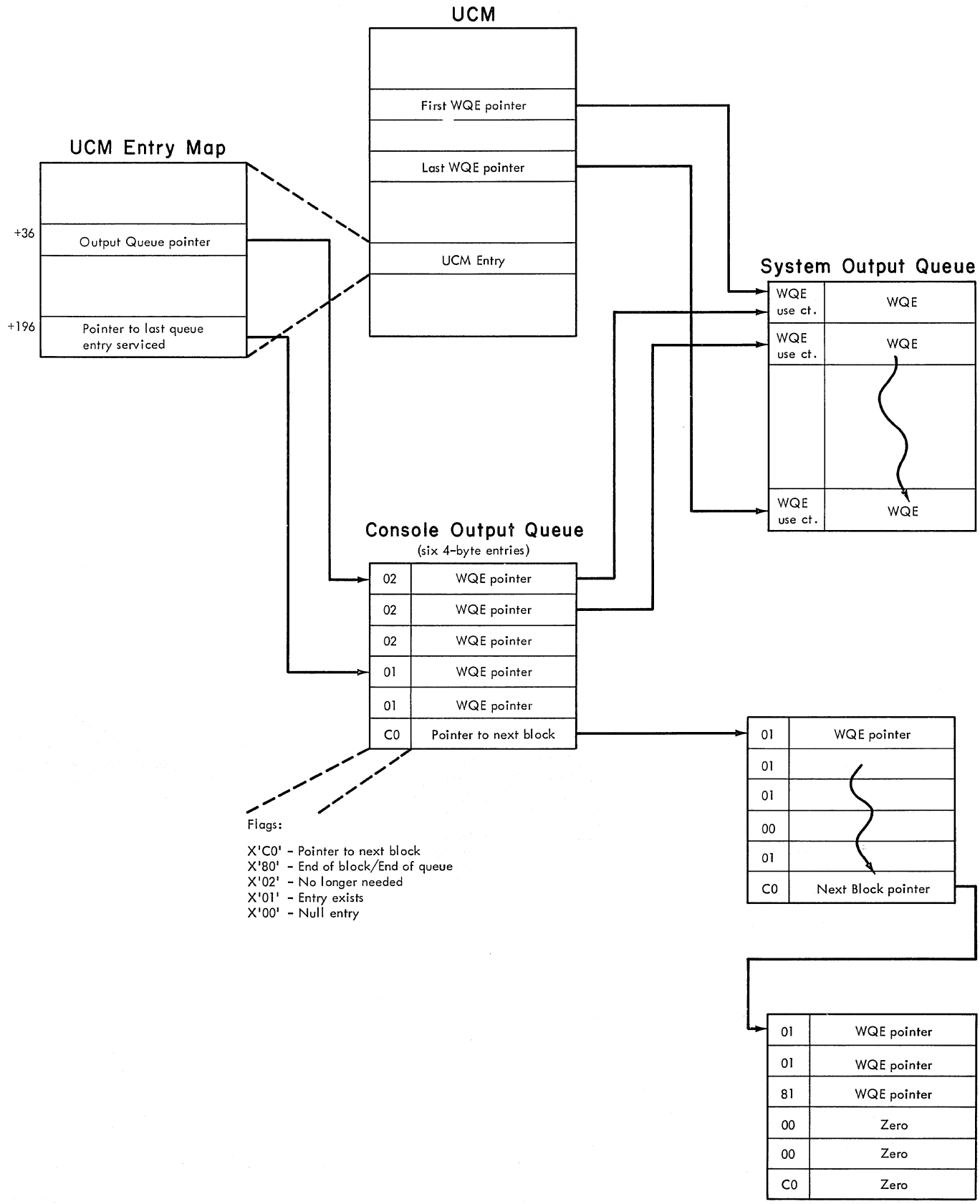


Figure 17. System and Console Output Queues

the system log from an operator's console. The WRITELOG command is used to request that the currently recording log data set be closed and queued to a SYSOUT writer of a particular class. If no class name is specified with the WRITELOG command, the data set will be scheduled for queueing to a SYSOUT writer of the class specified at system generation. When the system log is performing the functions of the hard copy log (a log function of the multiple console support option), WTO and WTOR messages, as well as operator and system commands and their responses, may be entered on the system log by converting them to a WTL macro instruction. WTO and WTOR conversion is done in the MCS communications task module IEECMDSV. Conversion of the LOG command is done in the Command Scheduler (SVC 34) routines.

The log support routines function under their own TCB in a manner similar to the Communications Task routines. The module IEELWAIT is attached as part of the initialization process and issues a WAIT macro instruction specifying the log ECB. Upon initial entry to IEELWAIT, SVC 36 is issued to open the log data sets.

Subsequent entries to IEELWAIT are caused when the log ECB is posted. The events which cause the log ECB to be posted are:

- A WRITELOG (CLOSE) command has been issued or HALT processing has occurred.
- The log buffer is full.

If IEELWAIT is entered because a WRITELOG command was issued, an SVC 36 is issued to cause data set switching. The recording data set (primary) is closed and the other data set (alternate) is opened for input.

If IEELWAIT determines that the log buffer is full, it ensures that the size of the log buffer to be written does not exceed the amount of space specified in the Data Extent Block (DEB). If the log buffer is equal to or less than the space remaining on the data set, the buffer ECB is posted, the log buffer is written to the data set, and a WAIT is again issued on the log ECB. Had the space remaining on the data set been insufficient, an SVC 36 would have been issued to simulate a WRITELOG command.

When a WRITELOG (CLOSE) command is entered, the process of writing the remaining text in the log buffer to the data set is the same as mentioned above for a full buffer condition. Then the currently recording data set is closed and the log function is deleted from the system.

An SVC 34 must be issued to process the LOG and WRITELOG commands. The SVC 34 module IEE1603D issues a WTL macro instruction. For the WRITELOG command, IEE1603D posts the log ECB and the WRITELOG indicator is set in the Log Control Area.

The WTL macro instruction results in an SVC 36. The module IEE0303F moves the message into the log buffer. If an additional message would overflow the buffer, the log ECB is posted, the buffer ECB is cleared, and control returns to the Master Scheduler. When IEELWAIT receives control, it switches buffer pointers, thereby indicating that the other (alternate) buffer is available for input. It then posts the buffer ECB, and proceeds to write the full buffer (primary) onto the data set. Since the buffer ECB has been posted, IEE0303F can now move a new message into the alternate buffer. When IEE0303F is entered from IEELWAIT because a WRITELOG was issued (SVC 36 is issued in IEELWAIT), control is passed directly to the second load of SVC 36 IEE0403F.

When IEE0403F is entered initially, the log data sets are opened and control blocks initialized. Upon subsequent entry, the currently recording data set (primary) is closed and scheduled for queueing to the SYSOUT writer. The other data set becomes the primary data set and pointers are adjusted accordingly. Messages are written to the operator indicating the switch of data sets, or the failure of a data set to be opened. When an alternate data set is not available (usually because the data set is still being written by the system output writer), the system log is temporarily made inactive. During the time that the log function is inactive, the buffer remains undisturbed and all incoming WTL macro instructions are reissued as WTO macro instructions to the primary/master console.

Anytime the operating system enters step termination and a log data set has been scheduled for queueing, the termination module IEFSD42Q passes control to the Log Dispatcher IEEVLDSP. IEEVLDSP enqueues the log data set to a system output queue of a particular class and issues a corresponding message to the operator.

After the log data set has been written, control is returned to the routine IEEVIOUT, which opens and immediately closes the data set to reinitialize the TTRLL field (a field that describes the space available on the data set). IEEVLOUT returns control to the SYSOUT Writer routine that called it.

The Timer Supervision routines extend the capabilities of the IBM System/360 interval timer feature and the System/370 Time-of-Day (TOD) Clock. Using the timer supervision routines, the programmer can obtain the date and time of day, measure periods of time between specified points in execution, or schedule activity for a specific time of day. The timer supervision routines also service the job-step timing and system management facility features.

This chapter contains a general description of timer supervision as follows:

- "Timer Supervision Services" describes the services available to the problem programmer.
- "The Functions of Timer Supervision" describes the operation of the supervisor in performing the services requested by the user.
- "Timer Supervision with the Time-of-Day (TOD) Clock."

Diagram 18 illustrates the control flow of timer supervision when invoked through the supervisor call interruption handlers.

SYSTEM/360 TIMER SUPERVISION SERVICES

The timing services available depend on the options selected when the operating system is generated. These options are the time option, which provides the ability to request the date and time of day, and the interval option, which includes the time option functions and also provides the ability to set, test, and cancel intervals of time. If neither option is selected, then the date is the only timing service provided.

The following routines provide the timer services:

- The TIME routine returns the current date and time of day. The operator initially gives a starting date and time of day with a SET command. Thereafter, Timer Supervision changes the date each midnight and maintains elapsed time. The TIME routine obtains the current date, adds elapsed time to the starting time given by the operator, and returns both values in general registers.
- The STIMER (Set Timer) routine processes requests for timed intervals by scheduling their placement into the interval timer to cause interruptions at requested times. For each STIMER macro instruction, this routine builds a Timer Queue Element (TQE) and places into it the information from the STIMER macro instruction that will be needed when the interval expires. It then positions the element in the timer queue using its time of expiration to determine its position. When a timer interruption occurs (the value in the timer has decremented to -1), timer interruption handling routines perform the requested actions and obtain a new interval to be placed into the timer from the next element in the timer queue.
- The TTIMER (Test Timer) routine returns the time remaining in a previously requested interval, or cancels previous STIMER requests. To determine remaining time, the TTIMER routine subtracts elapsed time from the time of expiration of the interval. To cancel previous requests, the TTIMER routine removes corresponding elements from the timer queue.

SYSTEM/370 TIMER SUPERVISION SERVICES

Control programs executing on System/370 CPUs use the interval timer feature and the Time-of-Day (TOD) Clock, a standard feature on these CPUs which provides timing resolution to one microsecond. In the following description of timer supervision, differences with the System/370 Time-of-Day Clock, if any, are discussed immediately after the descriptions of each timer routine or function.

The operator need reset the clock only when the current date and time of day are incorrect, not at each IPL. The TOD Clock runs continuously while power is on.

Because the TOD Clock maintains elapsed time automatically (using 0000 hours, January 1, 1960, as the base), the TIME routine uses the TOD Clock to update, if necessary, the current date and to determine the time of day. The current date and time of day are returned to the caller in the manner specified in the TIME macro instruction.

When the requested interval is greater than one hour, the STIMER routine uses the TOD Clock to determine the value expected to be in the TOD Clock at expiration. When the requested interval is less than or equal to one hour, the System/360 processing is used.

To determine the time remaining in an interval, the TTIMER routine subtracts the value in the TOD Clock from the value expected to be in the TOD Clock at expiration.

FUNCTIONS OF TIMER SUPERVISION

The functions of timer supervision are related to measuring elapsed time intervals and servicing a request upon expiration of a timer interval. Functions related to measuring elapsed time intervals include:

- Measuring elapsed time.
- Maintaining the timer queue.
- Measuring time of expiration.
- Determining interval values.
- Timer interruption handling.

Functions related to servicing requests upon expiration of a timer interval include:

- Determining what actions are to be performed.
- Altering timer requests.

MEASURING ELAPSED TIME IN SYSTEM/360

The TIME routine determines the current date and time of day and returns both values to the requesting routine. It obtains the date from the Communication Vector Table, into which it was placed by the Job Management SET Command routine. (Each day at midnight, the date is changed by the Timer Second-Level Interruption Handler.) To determine the time of day, however, the TIME routine must first determine how much time has elapsed since the operator gave the SET command.

The console operator issues the SET command to place the current time into a software clock called the local time pseudo-clock (LTPC). Unless the operator places the correct value into the LTPC, the timer routines will not calculate the correct time corresponding to the actual time of day. The LTPC should be initialized to the correct time during each initial program load (IPL) performed on the system.

After the operator gives a starting time, the interval timer must be kept continually operating so that elapsed time can be measured. The interval timer automatically decrements any value placed into it and causes an interruption when the value becomes negative. During initial program loading (IPL), a 6-hour value is placed into the interval timer, and, when this value expires, another 6-hour interval is placed into the timer by the Timer Second-level Interruption Handler.

To measure elapsed time, three pseudo clocks are used with the interval timer: the local time, the 6-hour, and the 24-hour pseudo-clock. Each time a 6-hour value is placed into the timer, one is also placed into the 6-hour pseudo clock. However, the value in the timer decrements, while that in the 6-hour pseudo clock does not. Thus, an elapsed time of up to 6 hours can be determined by subtracting the value in the timer from that in the 6-hour pseudo clock.

To measure intervals longer than 6 hours, a 6-hour value is added into the 24-hour pseudo clock each time one is placed into the 6-hour pseudo clock except for the first 6-hour interval. (Each time a 24-hour period elapses, the 24-hour pseudo clock is reset to 0.) The TIME routine determines elapsed time by subtracting the value in the timer from the sum of the values in the 6-hour pseudo clock (SHPC) and the 24-hour pseudo clock (T4PC):

Elapsed Time = (SHPC + T4PC) - Timer

Elapsed time is added to the starting time stored by the operator in the LTPC to arrive at the current time of day (TOD).

TOD = Elapsed Time + LTPC

For example, Figure 18 illustrates the timer, SHPC, T4PC, and LTPC immediately following initial program load. Figure 19 contains timer and pseudo-clock values at some later time. To calculate elapsed time, the formula is applied:

Elapsed Time = (6 + 6) - 3 = 9 hours

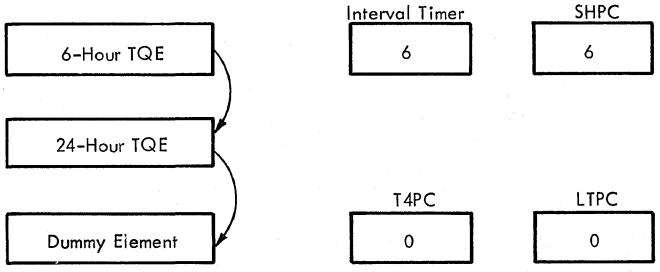


Figure 18. Timer Clocks after IPL

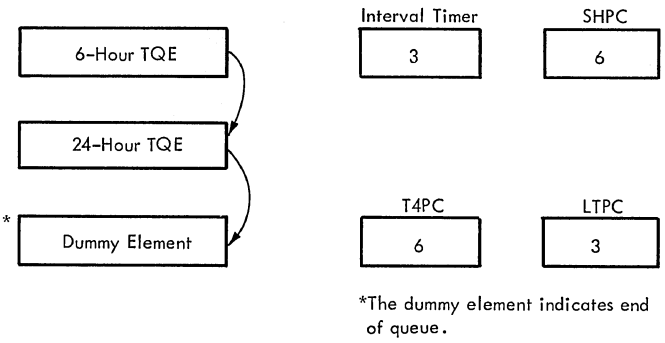


Figure 19. Measuring Elapsed Time Intervals

MEASURING ELAPSED TIME IN SYSTEM/370

With the TOD Clock, an additional parameter (MIC, address) is used to obtain the time of day in microseconds. The date is obtained from the CVT, as in System/360 processing, but all calculations to determine the time of day are performed in microseconds.

Each time that the TIME routine (IEA0RT01) is entered, the current date is verified using the value in the TOD Clock. The MNIGHT field in the nonexecutable timer module IEACVTPC contains the value that is expected to be in the TOD Clock at midnight of that day. From the MNIGHT value, the TIME routine subtracts the current value in the TOD Clock. If the result is positive, midnight of that day has not been reached and the current date is correct. If the result is 0 or negative, midnight of that day has been passed. The current date in the CVT is incremented by one day, the MNIGHT value is incremented by 24 hours (in microseconds), and the synchronization indicator in the midnight element is set on. Because the variance in the current date may have been more than one day, the procedure described above is repeated until the result of the subtraction is positive.

The TIME routine <u>does not</u> calculate the time of day in a manner similar to the TIME routine in System/360. Instead, the TIME routine uses the TOD Clock and the following algorithm to calculate the time of day in microseconds:

$$TOD = 86.4 \times 10^9 - (MNIGHT - CLOCK)$$

where:

86.4×10^9
is the number of microseconds in one day.

MNIGHT
is the value expected to be in the TOD Clock at midnight of the current day.

CLOCK
is the current value of the TOD Clock.

(MNIGHT - CLOCK)
yields the number of microseconds remaining until midnight of the current day.

For a time of day request specifying the MIC operand, the user-specified address is checked for validity. If the address is invalid, register 0 is set to 0, register 1 contains the date, the specified location is unchanged and control is returned to the caller with a code of 4 in register 15. For a valid address, register 15 is set to 0, the current date is returned in register 1, and the time of day is returned in the doubleword field specified by the caller; register 0 is set to 0.

If the TU operand was specified, the calculated time is converted to timer units by dividing the microsecond value by 26.041666. If the BIN or DEC operands were specified, the calculated time is divided by 10000 and the result is returned if BIN. If DEC was specified, the result is converted to HHMMSSth format.

<u>Note</u>: If the TIME routine is entered before nucleus initialization is complete, the date is returned as X'0000000F' and the time is returned as 0.

<u>Timer Units and Values</u>

The values used in the timer are timer units equalling approximately 13 microseconds; the values used in the pseudo clocks are timer units equalling 26.04166 microseconds. Timer values are converted to units of 26.04166 microseconds for calculations. The TIME routine converts the time of day to packed decimal form if the decimal (DEC) option was specified in the TIME macro instruction or to an unsigned binary value if the binary (BIN) option was specified. If the timer units (TU) option was given, no conversion is performed. The current time of day is returned to requesters in general register 0, and the date is returned in general register 1.

<u>Timer Queue Elements</u>

Timer Queue Elements (TQEs) are used by the system in supervising use of the hardware timer. Each task in a system with the interval timer option has a 96-byte block of main storage adjacent to the task control block for use as a timer queue element. A TQE is considered "not in use" until it is queued to a timer queue. A TQE that is not in use, however, can be used as an interruption request block (IRB) to execute a timer asynchronous exit routine.

MAINTAINING THE TIMER QUEUE

The timer queue is a chain of active TQEs that represent requests for timing services. The communications vector table (CVT) contains a pointer to module IEACVTPC. This module contains the pseudo clocks; the 6-hour, 24-hour, the optional SMF TQE, and the dummy TQEs; and the pointer to the timer queue (TQPTR). Figure 20 illustrates module IEACVTPC following nucleus initialization.

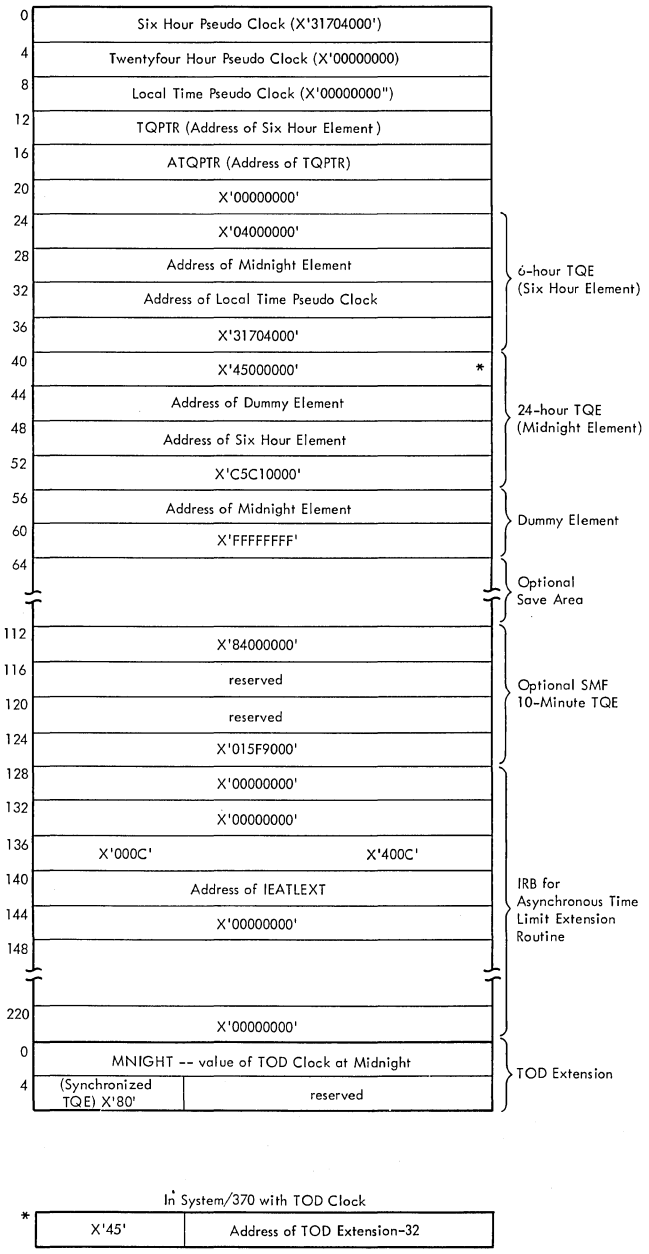


Figure 20. The Nonexecutable Timer Module IEACVTPC

All elements in the timer queue are arranged by time of expriation. After a timer interruption, the topmost element always represents the expired interval. This element is removed from the queue and used to determine what action is to be taken. Meanwhile, the interval represented by the next element (if it has not expired) is placed into the interval timer, and the procedure begins again.

MEASURING TIME OF EXPIRATION IN SYSTEM/360

The time of expiration, by which elements are ordered on the timer queue, is based on a 6-hour cycle. The STIMER routine places the interval requested in the element and uses the timer enqueue routine in the Timer SLIH to convert the interval to a time of expiration and to place the element on the timer queue. The timer enqueue routine subtracts the value in the interval timer from the value in the 6-hour pseudo clock (SHPC) and adds the interval requested:

TOX = (SHPC - Timer) + interval requested

For example, assume that no requests are pending and that 3 hours have elapsed since the operator issued a SET command. Figure 21 shows the timer queue and the values in

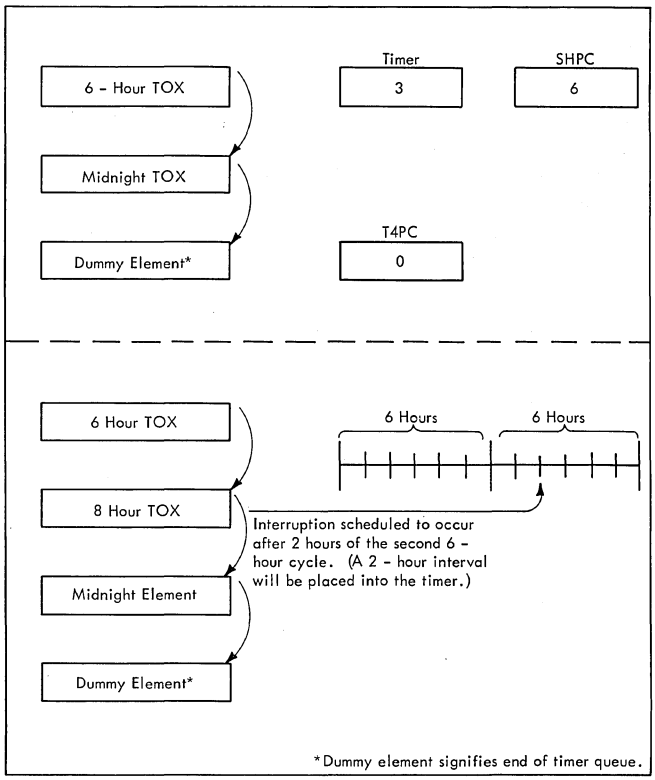


Figure 21. Position of Elements on the Timer Queue

both the timer and the 6-hour pseudo clock at this time. Assume now that a STIMER macro instruction requests a timer interruption in 5 hours. The time of expiration (TOX) is determined:

```
TOX = (SHPC - Timer) + interval requested
 8  = (  6  -   3  ) +   5
```

The element representing the 5-hour request is positioned on the timer queue between the 6-hour and midnight elements. Both the 6-hour and midnight elements always exist on the queue. When the 6-hour element expires, the Timer Second-Level Interruption Handler subtracts 6 hours from the times of expiration of all other elements on the timer queue and repositions the 6-hour element. Thus the element representing the request then becomes the topmost element, and its 2-hour time of expiration is placed into the interval timer. A timer interruption will occur on schedule -- 5 hours after receipt of the request. When the midnight element expires, the Timer Second-Level Interruption Handler changes the date given by the operator and repositions the midnight element.

MEASURING TIME OF EXPIRATION IN SYSTEM/370

The timer queue in System/370 is similar to that in System/360. However, the STIMER routine uses the TOD Clock to process REAL and WAIT type requests when the interval specified is greater than one hour. (REAL and WAIT type requests for intervals less than or equal to one hour and TASK type requests are processed as in System/360.) STIMER converts the requested interval from timer units to 1.048576-second units and adds the resulting value to the current value in the TOD Clock. (Bit 31 in the TOD Clock is incremented every 1.048576 seconds. Microsecond values are indicated by bit 51.) This yields the expected value of the TOD Clock at the time of expiration of the specified interval. This value is saved in the TQEWORK field in the element and will be used by the timer SLIH each time that the one-hour interval for this element expires. (See "Determining What Actions are to be Performed in System/370" below.) STIMER places an interval of one hour (in timer units) in the field TQEVAL in the element, sets on the synchronization indicator in the element, and uses the timer enqueue routine in the timer SLIH to convert the interval from timer units to a time of expiration and to place the element on the timer queue. Although the interval specified was greater than one hour, a timer interruption will occur in one hour and the element will be examined by the timer SLIH for further processing. (See "Timer Interruption Handling" below.)

DETERMINING INTERVAL VALUES IN SYSTEM/360

The STIMER macro instruction allows the user to specify the time interval in one of four ways: decimal form (DINTVL), binary form (BINTVL), timer units (TUINTVL), and time of expiration (TOD). When decimal or binary form is given, the STIMER routine converts the value to timer units (one timer unit = 26.041666 microseconds), before using the timer enqueue routine. When timer units are given, no conversion is necessary.

When time of expiration (TOD) is specified in the STIMER macro instruction, STIMER converts the time to an interval by subtracting the current time of day from the specified time of expiration. The interval is converted to timer units before using the timer enqueue routine. The current time of day is determined using the following algorithm:

```
Current Time of Day = LTPC + T4PC +
(SHPC - Timer)
```

where:

LTPC
: starting time given by the operator in the SET command.

T4PC
: value in the 24-hour pseudo clock.

SHPC
: value in the 6-hour pseudo clock.

Timer
: value in the timer.

Because no interval that exceeds 24 hours is valid, the STIMER routine replaces any interval that exceeds 24 hours with a 24-hour interval.

DETERMINING INTERVAL VALUES IN SYSTEM/370

The DINTVL, BINTVL, and TUINTVL operands are processed in a manner identical with System/360. When a time of expiration (TOD) is specified as the operand in a STIMER macro instruction, the STIMER routine uses the TOD Clock to determine the interval. STIMER first converts the time of expiration to a value in timer units that represents the interval between the beginning of that day (0000 hours) and the specified time of expiration. From this value, STIMER then subtracts the current time of day in timer units. The current time of day in timer units is calculated using the following algorithm:

TOD = (82397 - [MNIGHT - CLOCK]) x 40265

where:

MNIGHT
is the value expected to be in the TOD Clock at midnight of that day.

CLOCK
is the current value of the TOD Clock.

If the specified time of expiration has already been passed (the interval value is negative), STIMER adds the number of timer units in one day to the interval so that the time of expiration will occur at the specified time on the next day.

Initializing the Timer Queue Element

The STIMER routine initializes queue elements using information provided by the programmer in the STIMER macro instruction. It first checks to determine if the existing element can be used. This element could be an expired element, an element in the timer queue, or one that was changed to an interruption request block by the Timer Second-Level Interruption Handler. The STIMER routine reuses expired elements and removes and reuses elements that are on the timer queue. If the existing element has been changed to an interruption request block that is being used, the STIMER macro instruction is treated as a NOP, and control returns to the caller. The STIMER routine then uses the Timer ENQ subroutine to position the completed element on the timer queue.

TIMER INTERRUPTION HANDLING

When a time interval in the timer expires, an external interruption occurs, and control is automatically given to the Timer/External First-Level Interruption Handler (see Diagram 19). The Timer/External First Level Interruption Handler saves information about the interrupted program, distinguishes between timer and other types of external interruptions, and, for timer-caused interruptions, gives control to the Timer Second-Level Interruption Handler.

The Timer Second-Level Interruption Handler takes any actions the programmer specified (in the STIMER macro instruction) to be performed upon expiration and places another interval into the timer.

DETERMINING WHAT ACTIONS ARE TO BE PERFORMED IN SYSTEM/360

When a timer interruption occurs, the topmost element in the timer queue represents the expired interval. The Timer Second-Level Interruption Handler obtains the address of the topmost element from main storage location TQPTR, removes the element from the timer queue, and determines what action to take using bits 4 through 7 of the first word in the element.

If a TASK or REAL parameter was given in the STIMER macro instruction, and if an asynchronous exit routine was specified in the STIMER macro instruction, the Timer Second-Level Interruption Handler (TSLIH) must make further tests to determine what action should be taken. If no entry to an asynchronous exit routine is desired, the queue element is given an expired status. If an exit is specified and the timer queue element (TQE) is TASK type, the TSLIH changes the TQE to an interruption request block (IRB) containing an interruption queue element (IQE), and gives control to the Stage 2 Exit Effector.

If the TQE is REAL, if an exit is specified, and if an initiator issued the STIMER macro instruction, it indicates that the 30-minute wait limit (imposed by job step timing) has expired. When this case occurs, the problem program must be abnormally terminated while the timer queue element must be reinstated as TASK type with the actual CPU time remaining value. The Timer Second-Level Interruption Handler accomplishes this by branching to ABTERM with the address of the problem program job step TCB to schedule the step for ABEND. The TSLIH also passes ABTERM a unique ABEND code (522) which indicates that the 30-minute wait limit expired. Upon return from ABTERM, the TSLIH marks the timer queue element as TASK type and off the queue, and moves the CPU time remaining value from its save slot (TQESAV) to the time of expiration/time remaining slot (TQEVAL) within the TQE.

If a WAIT parameter was given in the STIMER macro instruction, the Timer Second-Level Interruption Handler gives control to the POST routine, directing it to post an appropriate event control block (contained within the timer queue element) and thus signal expiration of the interval.

After either of the above actions have been completed, the time of expiration (TOX) value of the topmost element is placed in the 6-hour pseudo clock. The TOX value minus the last value in the 6-hour pseudo clock is placed in the interval timer. (The element representing the recently expired interval has been removed from the queue.)

DETERMINING WHAT ACTIONS ARE TO BE PERFORMED IN SYSTEM/370

When the element at the top of the timer queue is removed by the timer SLIH, the synchronization indicator is checked to determine if this element is the 24-hour element or represents a REAL or WAIT type request that requires special processing by the timer SLIH. This indicator will have been set by the STIMER routine when the original interval requested was greater than one hour. Interim processing by the timer SLIH may have cleared this indicator.

If special processing is not required, System/360 processing continues. If the indicator is set, the timer SLIH subtracts the value in the TOD Clock from the value in the TQEWORK field. If the result is 0 or negative, the interval being timed has elapsed and processing continues as in System/360. If the result is positive but less than one hour, the synchronization indicator is cleared, and the remaining interval is converted from 1.048576-second units to timer units and placed in the TQEVAL field. The timer enqueue routine is used to convert the timer units to time of expiration and to place the element on the timer queue.

If the result of the calculation is positive and greater than one hour, the timer SLIH again sets the TQEVAL field to one hour, and uses the timer enqueue routine to convert the timer units to a time of expiration and to place the element on the timer queue.

Returning 6-Hour and Midnight Elements to the Queue in System/360

When intervals represented by either the 6-hour or midnight elements expire, the elements must be returned to the timer queue. Before it returns the 6-hour supervisor element, the Timer Second-Level Interruption Handler subtracts 6-hours from the times of expiration of all elements in the timer queue to reflect the passing of 6 hours since the elements were queued. It also adds 6 hours to the 24-hour pseudo clock unless its value is 18 hours, in which case it resets the 24-hour pseudo clock to 0. The Timer Second-Level Interruption Handler then uses the Timer ENQ subroutine to position and queue the 6-hour element on the timer queue.

Before the Timer Second-Level Interruption Handler returns the midnight element to the timer queue, it changes the date in the communications vector table.

Returning 6-Hour and Midnight Elements to the Queue in System/370

This processing is identical with System/360 except that the 24-hour pseudo clock is not used in System/370. This is replaced by the MNIGHT field in module IEACVTPC which is used with the TOD Clock.

ALTERING TIMER REQUESTS

The TTIMER routine returns the time remaining in a previously requested time interval or cancels a previously requested interval.

Determining Remaining Time in System/360

Before the TTIMER routine can determine remaining time, it must first locate the queue element that represents the affected interval. It obtains the address of the element from the TCB of the task being performed when the TTIMER macro instruction was given. If no element exists, or if the interval represented by the element has expired, this routine returns a 0. If an unexpired interval exists, the TTIMER routine determines remaining time by using the following formula:

Remaining Time = TOX - (SHPC - Timer)

where:

TOX
: time of expiration of the element.

SHPC
: value in the 6-hour pseudo clock.

Timer
: value in the interval timer.

The interval may have expired while the TTIMER routine was being executed, in which case the above calculation would yield a negative remaining time value. If so, a 0 value is returned in general register 0. If a positive remaining time value is obtained, it is placed unaltered (in timer units) into general register 0.

Determining Remaining Time in System/370

When a TTIMER request specifies an element that has the synchronization indicator set, the TTIMER routine determines the remaining time by subtracting the value in the TOD Clock from the value in the TQEWORK field in the element. A 0 or negative value indicates that the interval has elapsed and TTIMER returns a 0 in register 0. A positive value is converted to timer units and returned to the requester in register 0.

Canceling an Interval

If the CANCEL option was used in the TTIMER macro instruction, the TTIMER routine uses the Timer Dequeue subroutine to remove the corresponding element from the timer queue. The TTIMER routine also clears the TQE pointer (TCBTME) in the current TCB. The current task thus no longer has a timer queue element.

SMF PROCESSING AND JOB STEP TIMING

When the System Management Facility (SMF) has been included in the system, the Timer SLIH may perform additional processing.

If the timer interruption is recognized as following the expiration of a supervisor 10-minute TQE, the Timer SLIH obtains the accumulated system wait time for the 10 minutes from the second word of the save area SYSWSAVE. This value is added to the SMCAWAIT + 4 field in the system management control area. (Refer to "Special Features" for a description of the SMF option.) Each time that step termination is entered, this field is checked. If it is not zero, a system 10-minute wait record is generated.

The Timer SLIH then sets the accumulated wait time field to zero, places a value of 10 minutes in the 10-minute TQE, and returns the TQE to the timer queue.

If the timer interruption is recognized as a job, step, or wait time expiration, the Timer SLIH checks the timing control table (TCT) for the address of a user time limit expiration routine (IEFUTL). If such a routine is present, and if the expired TQE belongs to the initiator, the Timer SLIH initializes an IRB/IQE representing the SMF Time Limit Expiration routine (IEATLEXT). The Stage 2 Exit Effector is then entered to schedule the execution of the SMF Time Limit Expiration routine.

Overlay supervision controls the loading of overlay program segments and assists the flow of control between the segments of an overlay program. While performing these functions, overlay supervision places data into, and uses data from, the segment table (SEGTAB) and the entry tables (ENTABs).

Because the segment and entry tables are part of each overlay program, the overlay supervisor is reenterable; its services can be used concurrently by many overlay programs.

During execution, an overlay program issues requests for segments. The requests can be explicit via a SEGLD or SEGWT macro instruction, or implicit via a branch that is routed through an ENTAB. In either case, the overlay supervisor receives control from the SVC handler and checks the SEGTAB to determine whether the requested segment is in main storage. If not, the overlay supervisor requests Program Fetch to load the segment. When this segment is part of an overlay program that is being tested, the overlay supervisor also passes control to the TESTRAN interpreter. Program Fetch and the TESTRAN interpreter each return control to the overlay supervisor after their functions have been performed.

OVERLAY SUPERVISION ROUTINES

Overlay supervision is composed of a resident module (IEWSVOVR) called overlay supervisor 1, and either of two nonresident modules selected during system generation called overlay supervisor 2. The module name of overlay supervisor 2 is IEWSYOVR for the basic sychronous module, or IEWSXOVR for the basic synchronous module with optional SEGWT checking. To pass control to either version of overlay supervisor 2, overlay supervisor 1 issues a LINK macro instruction that specifies IEWSZOVR, which is the member name of the selected module in the LINKLIB.

FUNCTIONS OF OVERLAY SUPERVISION

Overlay supervision in this manual includes the functions of and operations performed by the overlay supervisor, the linkage editor, contents supervision, and Program Fetch. The functions of overlay supervision include the following:

- Building and Initializing Control Areas.
- Servicing Requests for Linkage to Overlay Program Segments.
- Passing Control to the Requested Segment.

TABLES USED BY OVERLAY SUPERVISION

The segment table (SEGTAB) and the entry tables (ENTABs) that contain the data used by the overlay supervisor are created by the linkage editor from information in the relocation dictionary (RLD) and the user's control statements. Figure 22 shows the SEGTAB and ENTABs in a typical single region overlay structure; the ENTAB and SEGTAB formats are given in Section 5.

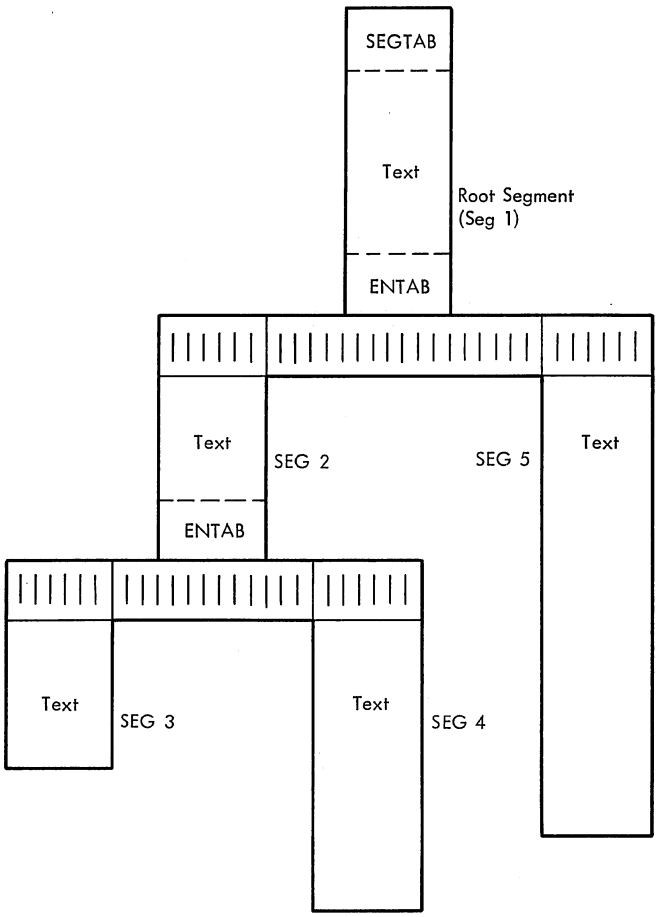


Figure 22. Single Region Overlay Structure

Segment Table

The segment table (SEGTAB) contains data that describes the structure and status of an overlay module, and is a directory for the segments of that module. It contains both fixed and variable information. The fixed information includes:

- TEST indicator. This indicator is set by Program Fetch if the partitioned organization directory record indicates that the program is being tested under TESTRAN.
- Last segment number of each region. This value defines the segment that ends a region and is used to determine the region that contains a particular segment.
- Previous segment number of each segment in the module. The overlay supervisor uses this field to determine the additional segments that must be loaded with the requested segment. (These additional segments are those in the path of the requested segment.)

The variable information includes:

- Pointers. These pointers are addresses of the NOTE list and DCB.
- Highest number segment of each region in main storage. This value is initialized to 1 for the first region by the linkage editor.
- Status indicator for each segment. The overlay supervisor sets a status indicator for each segment to indicate either that the segment is not in main storage, that the segment is being loaded into main storage, or that the segment is present in main storage.

Entry Tables

The entry tables (ENTABs) assist in passing control between the overlay supervisor and an overlay program. They handle downward branches in an overlay program, that is, the branches to segments lower in the path.

When the overlay program executes an upward branch, the overlay supervisor is not entered, and the ENTABs and SEGTAB are not used. An upward branch is direct because segments in the path are always in main storage (Figure 23).

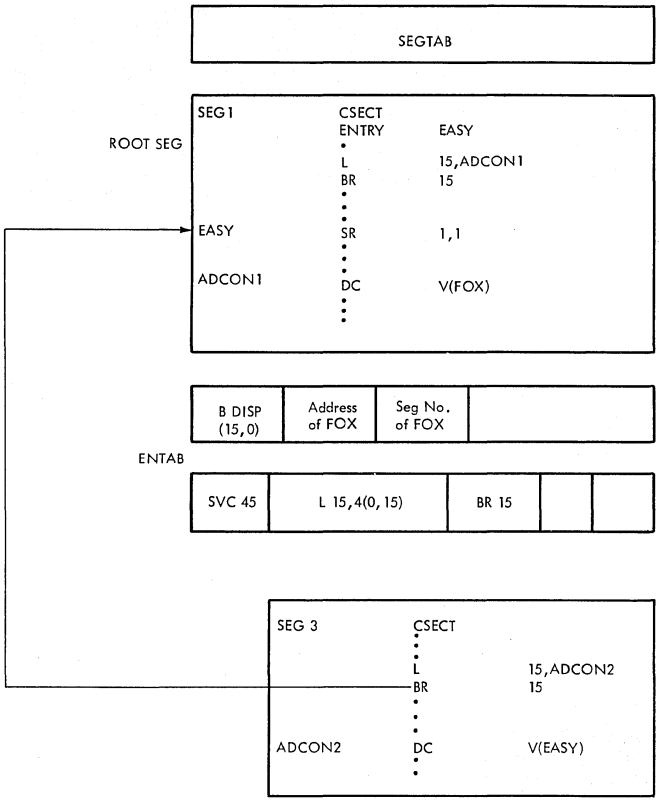


Figure 23. Overlay Program Upward Branch

BUILDING AND INITIALIZING CONTROL TABLES

Before execution of an overlay object module, the Linkage Editor builds, from information in the relocation list dictionary (RLD) and the user's Linkage Editor control statements, a segment table and one or more entry tables. These tables are made a part of the overlay load module and are used by the overlay supervisor during module execution.

The input to the Linkage Editor is an object module including Linkage Editor control statements. The OVERLAY control statement indicates the beginning of an overlay segment. It may also indicate the beginning of an overlay region.

An overlay module can be designed in single or multiple regions (as many as four). A region is a contiguous area of main storage within which segments can be loaded independently of paths in other regions.

When the Linkage Editor processes an overlay object module, it builds a segment table (SEGTAB) as part of the root segment and an entry table (ENTAB) for each segment except the lowest segment in a path. The segment table contains one entry for each segment of the overlay module. The entries in the segment table are ordered to correspond to the segment numbers of the overlay structure. Each entry contains the number

of the preceding segment in the path and a field of status indicators. The segment table entries form a tabular representation of the overlay structure.

The SEGTAB records the relationship of the segments in the overlay module and specifies which segments are loaded into main storage at a given time. A segment's ENTAB contains an entry for each V-type address constant that refers to a symbol in a different segment of the overlay module.

SERVICING REQUESTS FOR LINKAGE TO OVERLAY PROGRAM SEGMENTS

The overlay supervisor receives control either when an overlay segment issues a SEGLD or SEGWT request to load another segment, or when a segment issues a CALL or branch instruction to an external address in another segment not in main storage. When the overlay supervisor receives control because of a SEGLD request, it returns immediately to the interrupted program. When the overlay supervisor receives control for any other reason, it examines the segment table to determine whether the requested segment is already in main storage, and whether all segments in its path have been loaded. If it is not already in main storage, it causes the requested segment (and any needed segments in its path) to be loaded by Program Fetch.

When loading is complete and the caller has issued a CALL or branch instruction, the overlay supervisor alters the entry tables of the loaded segments. The modified entry tables permit future branches to the same points in the loaded segments without help from the overlay supervisor.

Finally, depending on the type of invoking macro instruction, control is given to the caller, after loading is complete (SEGWT), or to the branch address in the requested segment, after the segment is loaded (CALL or branch instruction).

Linkage to the Overlay Supervisor

Linkage to the overlay supervisor is initiated directly for a SEGLD or a SEGWT macro instruction. It is initiated indirectly for a CALL or branch instruction.

Direct Supervisor Linkage: When the SEGLD macro instruction expands, a 0 is placed in register 0 and an SVC 37 instruction is issued. When the SEGWT macro instruction expands, a 1 is placed in register 0 and an SVC 37 instruction is issued. Execution of the SVC instruction causes an SVC interruption, and control passes to the SVC FLIH, SVC SLIH, and resident module IEWSVOVR of the Overlay Supervisor (at entry point IGC037). If the interruption was caused by a SEGLD request (0 in register 0), the Overlay Supervisor returns immediately to the routine that issued SEGLD. SEGLD is treated as a NOP (no operation) in MFT. If the interruption was caused by a SEGWT request (1 in register 0), then the overlay supervisor determines if a direct branch entry to the requested segment, via the caller's ENTAB, has been prepared through a previous branch or CALL. If it has, control is returned to the caller (see Diagram 20) because further processing of the current request is not needed. If a direct branch entry has not been prepared, module IEWSVOVR, after performing initialization, issues a LINK macro instruction to obtain supervisor-assisted linkage to nonresident module IEWSZOVR. IEWSZOVR is the name on the LINKLIB given to either module IEWSYOVR or module IEWSXOVR. One of these two modules is selected during System Generation. IEWSYOVR is the basic synchronous transient Overlay Supervisor Module. IEWSXOVR is the synchronous Transient Overlay Supervisor Module with optional SEGWT checking. Module IEWSZOVR processes the request, as described in "Types of Processing."

Indirect Supervisor Linkage: When a branch instruction or CALL macro instruction in an overlay segment is executed, specifying a V-type address constant, a branch is made to the associated ENTAB entry, which branches to an SVC 45 instruction in the last ENTAB entry. When the SVC instruction is executed, an SVC interruption occurs, and control passes to the SVC FLIH, SVC SLIH, and resident module IEWSVOVR of the Overlay Supervisor (at entry point IGC045). (See Figure 24, steps A, B, and C.) After performing initialization, module IEWSVOVR issues a LINK macro instruction to obtain supervisor-assisted linkage to nonresident module IEWSZOVR. Module IEWSZOVR processes the branch request, as described in "Types of Processing."

Types of Processing

The processing performed by the overlay supervisor depends on:

1. Whether the requested segment and all segments in its path are in main storage (because of a previous request).

2. How the overlay supervisor received control: a SEGLD request caused an SVC 37 interruption; a SEGWT request caused an SVC 37 interruption; a CALL or branch to an ENTAB caused an SVC 45 interruption.

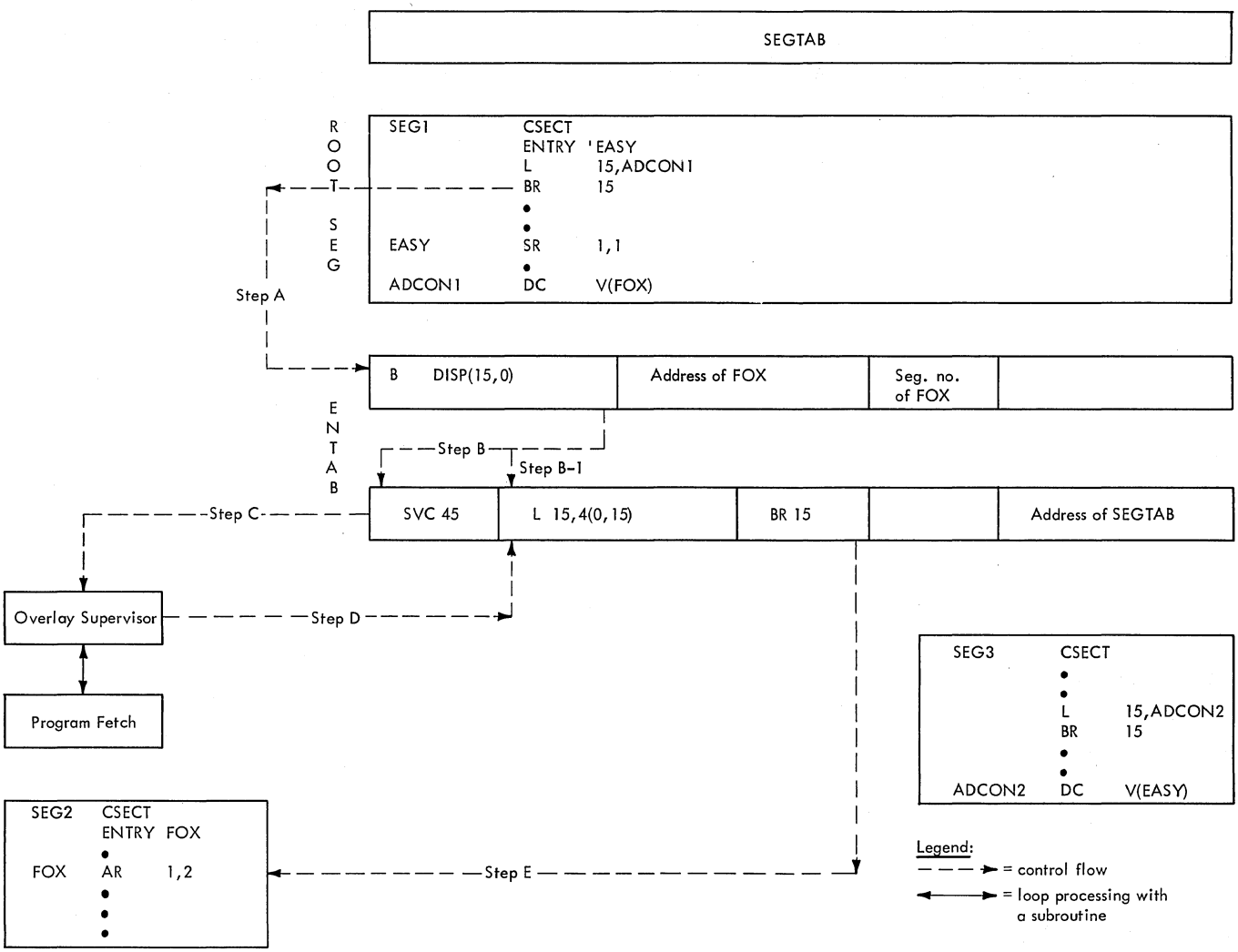


Figure 24. Branching to a Segment

A CALL or branch to an ENTAB does not cause an SVC (45) interruption if the requested segment is in main storage and if the caller's ENTAB has already been altered to provide a direct branch (by a previously serviced SVC (45) interruption).

Overlay supervision processing for each of the above sets of conditions is shown in Figure 25.

Determining the Status of a Requested Segment (and All Segments in Its Path)

The nonresident module of the overlay supervisor (IEWSZOVR) determines which segments in the path of the requested segment should be loaded by scanning the status indicators in the segment table of the overlay module. For each segment that must be loaded, IEWSZOVR sets indicators to control a subsequent fetch process. During the scan of the segment table, the entry for the requested segment is located, and its status indicators are examined. The resultant processing is tabulated in Figure 26.

Loading the Required Segments into Main Storage

After the SEGTAB has been scanned and the entries for the segments to be loaded have been marked, these segments are loaded into main storage using Program Fetch. If the segments are loaded in response to a SEGWT request, the overlay supervisor returns to the calling routine. If the segments are loaded in response to a CALL or branch request, the overlay supervisor adjusts the ENTAB entries to permit an unassisted branch to the newly loaded segment(s).

Type of Instruction	Status of Requested Segment(s)	Major Overlay Supervision Processing
1. SEGLD (SVC 37)	Irrelevant	Control is returned to the caller.
2. SEGWT (SVC 37)	Not in main storage	Needed segments are loaded. The caller's entry table is not altered to prepare for a branch to the requested segment control is returned to the caller after the requested segment and any needed segments in its path have been loaded.
3. SEGWT (SVC 37)	In main storage	Control is returned to the caller.
4. CALL or branch (SVC 45)	In main storage (Segment was requested via SEGWT)	The caller's ENTAB is altered to prepare for a future branch to the same external address without entry to the Overlay Supervisor. Control is given to the requested segment at the specified address.
5. CALL or branch (SVC 45)	Not in main storage	Needed segments are loaded. When loading is complete, the reamining processing is the same as in 4.

Figure 25. Types of Overlay Supervision Processing

Conditions	Resultant Processing by IEWSZOVR
1. Requested segment in main storage (indicator 10).	If entry is for a SEGWT request, control is returned to caller. If entry is for CALL or branch, ENTAB entries are altered to provide future entry to segment.
2. Requested segment is not in main storage (indicator 11).	Sets indicator to show "loading scheduled" (01) and continues the scan. Determines if the preceding entry is for a segment in the path of the requested segment.
3. The preceding entry is for a segment in the path of the requested segment.	Checks status indicator of preceding entry to determine if its segment is in main storage. (Next step is 5 or 6.)
4. The preceding entry is for a segment not in the path of the requested segment.	Sets status indicator of preceding entry to "not in main storage" (11) in preparation for overlaying the segment. Continues scan.
5. Preceding entry is for a segment in the path, and indicates its segment is in main storage.	Scan is stopped. The assumption is that all segments in the path of the requested segment are in main storage (except the requested segment itself).
6. Preceding entry is for a segment in the path, and indicates its segment is not in main storage.	Sets the status indicator of the entry whose segment is in the path to "loading scheduled" (01) and continues the scan.

Figure 26. Processing of Segment Table Entries

Adjusting an ENTAB to Permit Unassisted Branches to Loaded Segments

When module IEWSZOVR is entered after an SVC 45 interruption caused by a CALL or branch instruction, it alters the caller's ENTAB when it has determined that the requested segment is in main storage, or when it has loaded the segment. It adds 2 to the displacement (DISP) field of the ENTAB entry through which the branch to the SVC 45 instruction was routed (see Figure 24, Step B). When the caller executes another branch to this ENTAB entry, the SVC 45 instruction will be bypassed, and control will be given to the second field of the last ENTAB entry (see Figure 24, Step B-1). Execution of the instruction in this field will cause general register 15 to be loaded with the value assigned to the address constant (in the example, the address of FOX). A branch to that location in the requested segment will then be executed.

All entry tables in the same overlay region that have been altered to bypass the SVC 45 instruction are chained together in a "caller chain." A pointer to the last-altered entry table is placed in the segment table. When a segment is to be overlaid, module IEWSZOVR uses the appropriate caller chain to reset all modified entry tables that refer to the segment to be overlaid. Thus, an unassisted branch cannot occur to a segment that is no longer in main storage. The resetting of ENTAB entries in a caller chain accompanies the processing shown for condition 4 of Figure 26.

PASSING OF CONTROL

The last function of the Overlay Supervisor is to pass control. Control is given to the requested segment or returned to the calling segment, depending on the type of invoking instruction (SEGLD, SEGWT, CALL, or branch). See Figures 24 and 25.

Branching to a Segment in Main Storage

When a segment is loaded into main storage, because of an implicit call (a branch through an ENTAB), the displacement (DISP) field in the ENTAB entry through which the branch was routed is increased by 2 (Figure 24, Step B-1). When the overlay program executes another branch to this ENTAB entry, the SVC instruction is bypassed, and control is given to the second field of the last ENTAB entry. Execution of the instruction in this field causes general register 15 to be loaded with the main storage address assigned to the indicated symbol. A branch to that location is then executed.

A caller is an ENTAB entry that assisted in routing a branch from a segment to an entry point in a segment lower in the path. ENTAB entries that have been modified to bypass the SVC instruction are chained together in a caller chain (Figure 27). These entries are chained only if the called and calling segments are located in the same region. Chaining is accomplished by placing a pointer to (address of) the modified ENTAB entry into the caller field of the SEGTAB when the segment is brought into main storage. If this segment is requested again, the contents of the SEGTAB caller field (a pointer to a previous caller) is placed into the previous caller field of the referred to ENTAB entry, and a pointer to this ENTAB entry is placed in the caller field of the SEGTAB. In this way, a chain is created that begins at the SEGTAB entry and points to all the ENTAB entries (in the same region) that were modified (+2) to bypass the SVC 45 instruction. When the segment is to be overlayed, the caller chain is used to reset all of the modified ENTAB entries in the chain.

Branching to a Segment Not in Main Storage

When an overlay program branches to a segment not in main storage, control is passed to the applicable ENTAB (step A of Figure 24). The branch instruction in the ENTAB passes control to an SVC instruction contained in the first field of the last ENTAB entry (step B). The SVC instruction causes an SVC interruption, and passes control to the SVC handler and then to the overlay supervisor (step C). The overlay supervisor uses a pointer in general register 15 to obtain the information required to:

- Determine the number of the requested segment from the ENTAB.
- Determine the status of the requested segment from the SEGTAB.
- Pass control to the requested segment at the entry point specified by the address of the entry point field in the ENTAB.

After the segment is loaded, control is returned to the second field of the last ENTAB entry, the instruction following the SVC (step D). When the load and branch instructions have been executed, control is passed to the correct entry point.

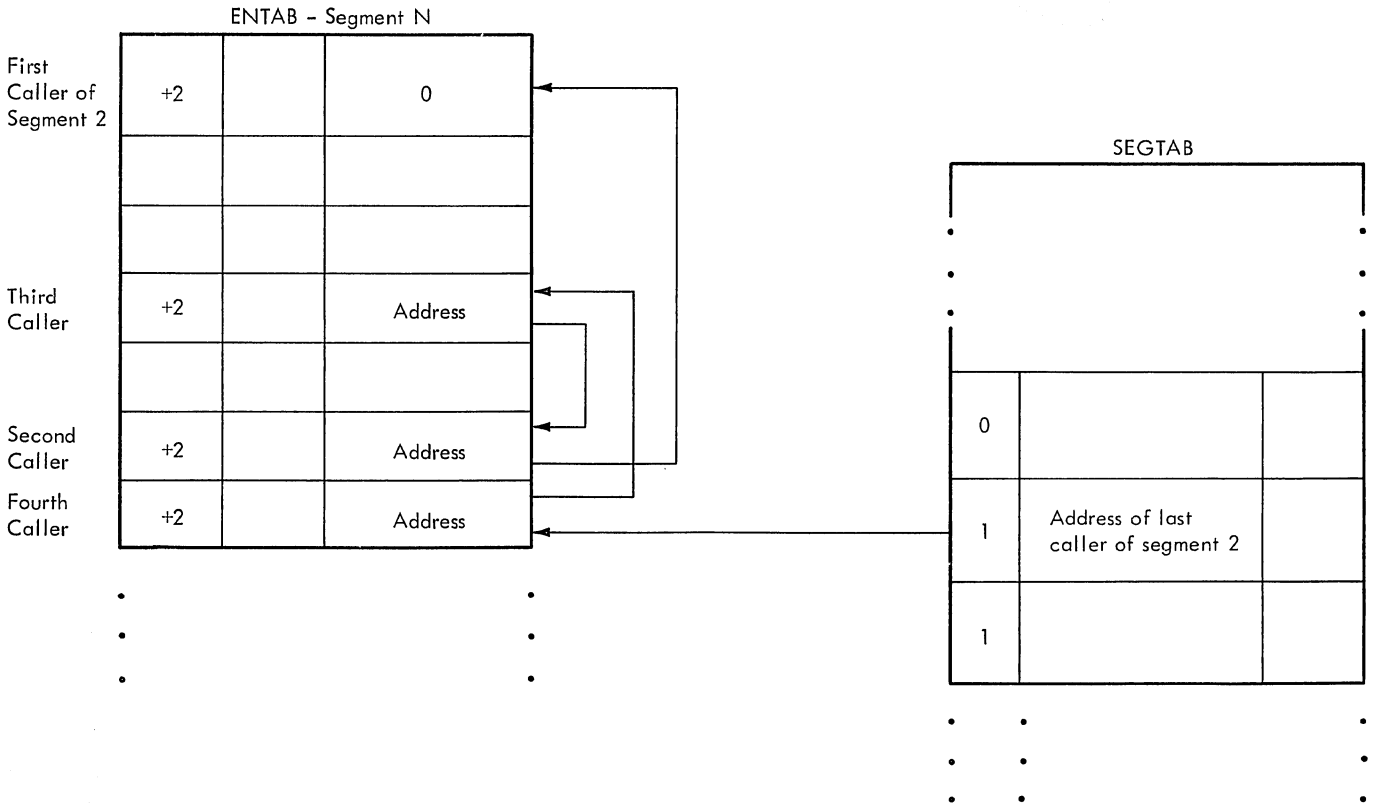


Figure 27. Chain of ENTAB Entries Used to Branch to a Segment

Operating System Recording/Recovery routines are optional control program routines which may be selected during system generation. They handle malfunctions of the central processing unit (CPU), in main storage, in a channel, and of devices.

Recording/Recovery routines are divided into two groups: System Environment Recording and Recovery Management. Recovery Management includes:

- Machine-Check Handler (MCH), described in the appropriate model-dependent Machine-Check Handler Program Logic Manuals.
- Channel-Check Handler (CCH), described in the Input/Output Supervisor, Program Logic Manual.
- Alternate Path Retry (APR), described in the Input/Output Supervisor, Program Logic Manual.
- Dynamic Device Reconfiguration (DDR), described in the Input/Output Supervisor, Program Logic Manual.

MACHINE-CHECK ROUTINES

The following routines service machine-check interruptions:

SER0
: which records information about the error and then places the system in a wait state.

SER1
: which records information about the error and attempts to associate the error with a task. If it can do this, it abnormally terminates the task and allows the system to continue operation.

MCH
: which records information about the error and attempts to recover the system from the adverse effects of the error.

For the Model 65, any one of these three routines may be selected during system generation. For the Model 40, 50, and 75 either SER0 or SER1 is used by default. For the System/360 Model 65 Multiprocessor and Model 85 and the System/370 Models 135, 145, 155, and 165, MCH is standard. The version used by default depends on the model (or models) specified, and on the size of the system (see System Generation).

CHANNEL-CHECK HANDLER

The channel-check handler (CCH) may be selected during system generation for System/360 Models 65 and 75. CCH is automatically included for System/360 Model 85 and System/370 Models 135, 145, 155, and 165. CCH supports the stand-alone 2860, 2870, and 2880 channels and the integrated 145 and 155 channels.

CCH aids recovery from channel errors (channel control checks and interface control checks) by providing channel error information to IBM-supplied, device-dependent error-recovery procedures. CCH also builds a record entry which is later written on SYS1.LOGREC by the outboard recorder of the I/O supervisor.

The outboard recorder (see the Input/Output Supervisor, Program Logic Manual) issues an SVC 76 instruction (IFBSTAT) to write the record on SYS1.LOGREC. IFBSTAT issues a GETMAIN macro instruction, an ENQ macro instruction specifying SYS1.LOGREC, and an EXCP instruction to write the record on that data set. In MFT with subtasking, the main storage for all request blocks is allocated from the 544-byte system error task partition in the nucleus. Handling of main storage is done by IEAMSERB. In MFT without subtasking, this main storage is obtained from the user's partition as opposed to System Queue Area in MVT. If the error task is invoked while the user has control, this error task will need up to 544 bytes in the partition to format and write a record on SYS1.LOGREC.

ALTERNATE PATH RETRY ROUTINE

Alternate Path Retry (APR) allows an I/O operation that has developed an error on one channel to be retried on another channel (if another channel is assigned to the device performing the I/O operation). APR accomplishes this by causing channel-detected errors to be retried in a selective manner on the available paths to a device. As paths are found to be inoperative, they are marked offline, thus preventing unnecessary retry from being initiated to the failing paths.

APR also provides the capability to VARY a path to a device online or offline, using

the VARY PATH command. The VARY PATH command processor is part of the Master Scheduler (SVC 34). The last path to a device will not be varied offline.

APR is not model-dependent. Four paths to each device are supported; teleprocessing paths are not supported.

DYNAMIC DEVICE RECONFIGURATION ROUTINE

Dynamic Device Reconfiguration (DDR) allows a demountable volume to be moved from one device to another, and repositioned if necessary, without abnormally terminating the affected job or reperforming IPL. A request to move a volume may be initiated by the operator with the SWAP command. The SWAP command processor is part of the Master Scheduler (SVC 34).

The system may request a SWAP after a permanent I/O error for non-SYSRES devices or after an error in a system fetch operation for SYSRES devices. DDR is not model-dependent.

SYSTEMS WITHOUT RECORDING/RECOVERY ROUTINES

A machine check or I/O interruption caused by an equipment malfunction places in a wait state those IBM System/360 models that do not have Recording/Recovery routines. A message is issued on the console telling the operator to load the System Environment Recording, Editing, and Printing (SEREP) program. SEREP is a model-dependent, stand alone diagnostic program. Its use is described in the Operator's Guide.

ENTRY TO RECORDING/RECOVERY ROUTINES

When a machine-check interruption occurs, the machine-check new PSW is loaded. This causes control to pass directly to the Recording/Recovery routine which was selected during system generation.

When an I/O interruption occurs because of a channel error, the I/O new PSW is loaded. This causes control to pass to the I/O FLIH and then to the I/O Supervisor.

If the Channel-Check Handler option was not selected during system generation, the I/O Supervisor enters the SER Interface subroutine (SERR04) within the I/O Supervisor. This routine loads the machine-check new PSW.

If the Channel-Check Handler was selected during system generation, the I/O Supervisor enters the Channel-Check Handler Interface within the I/O Supervisor.

When a permanent I/O error is indicated (after retry by the device-dependent error-recovery procedures of IOS), the OBR/SDR routine is entered to record the error.

If DDR was selected during system generation, the OBR/SDR routine enters DDR after recording the permanent I/O error. When the permanent I/O error occurs on a system fetch operation, the DASD ERP or FINCH enters DDR SYSRES before OBR/SDR receives control (if DDR SYSRES was selected during system generation).

The CHECKPOINT and RESTART service routines minimize the amount of time wasted when a program abnormally terminates. For example, CHKPT macro instructions may be used to divide the program into sections. When the program abnormally terminates, it can be restarted immediately (automatic restart) or it can be restarted later by the programmer (deferred restart).

If abnormal termination occurs in the first section of a program, restart begins at the beginning of the step (Step Restart). If abnormal termination occurs in any other section, restart may begin at the beginning of that section (Checkpoint Restart). Checkpoint restart eliminates the need to rerun sections of a program that have already run successfully.

If a program is coded using three CHKPT macro instructions, it is divided into four sections (see Figure 28). If abnormal termination occurs in section 3, an automatic checkpoint restart begins at CHKPT B (if the programmer has requested automatic restarts).

The CHECKPOINT routine is called directly when a problem program issues a CHKPT macro instruction. The RESTART routine is called by a job management program when a restart is scheduled.

Section 4 contains a list of CHECKPOINT/RESTART SVC Modules and Register Usage Table. Section 5 contains the format of records used by CHECKPOINT/RESTART.

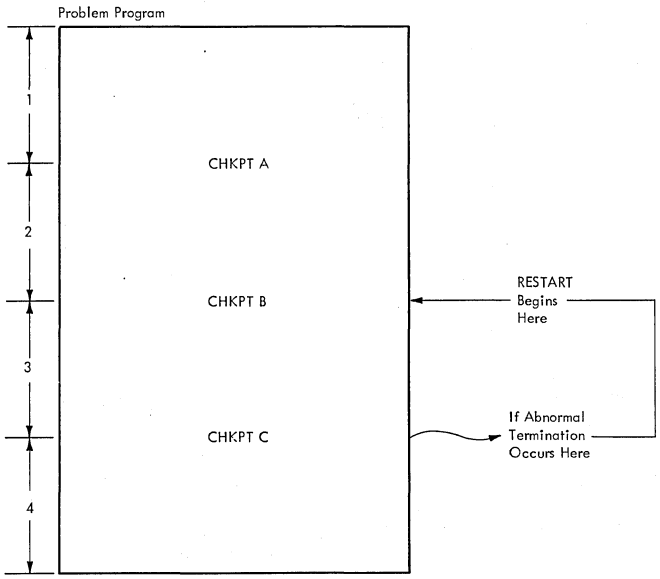


Figure 28. Problem Program Checkpoints

Device Independent Display Operator Console Support (DIDOCS) is an option of the MFT control program. DIDOCS provides uniform operator console services for the following display console devices:

- 2250 Display Unit, Models 1 or 3
- 2260 Display Station, Model 1 with 2848 Display Control, Model 3
- Model 85 CRT Display
- Model 165 CRT Display

Display Console Support takes advantage of device-dependent features of each display device (such as the use of the light pen on the 2250 for message deletion). The DIDOCS routines receive control from MCS when one of the following events occurs for a display console:

- An attention caused by the operator
- Console switching
- I/O completion
- A WTO, WTOR, or command
- A Delete Operator Messages (DOM) request
- A Timer Interruption
- Status Display on the Queue

DIDOCS support comprises the following routines:

- DIDOCS Processor routines (IGC5107B, IGC5Z07B, IGC6107B, and IGC6Z07B) - provide an interface between DIDOCS and MCS. The Processor routines receive control from MCS when an operator or system request is entered that requires processing by the DIDOCS routines. If the console involved in the request has a transient DCM, Processor 0, Load 1 (IGC6107B) receives control. Processor 0, Loads 1 and 2 provide transient DCM swapping and permanent PFK update support, and then pass control to Processor 1 (IGC5107B). Processor 1, Load 1 receives control either from Processor 0 or directly from MCS (when the request involves a console that has a resident DCM). Processor 1, Loads 1 and 2 route control to other display console routines as required to process the request.
- Open/Close routine (IGC5G07B) - Opens and closes display console device.
- 2250 I/O 1 and 2 routines (IGC5P07B, IGC5Q07B) - Handles input/output operations for the 2250 Display Unit.
- 2260 I/O 1 and 2 routines (IGC5R07B) - Handles input/output operations for the 2260 Display Station.
- Model 85 I/O routine (IGC5H07B) - Handles input/output operations for the Model 85 and Model 165 CRT Display used as an operator console.
- Asynchronous Error routine (IGC5C07B) - Initializes DCM on an OPEN; blanks the screen and displays an appropriate error message for the operator. If a permanent error occurs, it passes control to MCS for console switching.
- Message 1, 2, and 3 routines (IGC5D07B, IGC5E07B, IGC6D07B) - Contain all messages used for Display Console Support.
- Display 1, 2, and 3 routines (IGC5207B, IGC6207B, IGC5307B) - Write all messages from the Operating System to the operator except those to be deleted and status display messages and mark messages according to their descriptor codes.
- Roll Mode routine (IGC5J07B) - Removes messages from the screen at interval specified by the operator.
- Command routine (IGC5407B) - Analyzes type of command in entry area and takes appropriate action, or passes control to another Display Console Support routine for action.
- Options routine (IGC5A07B) - Analyzes CONTROL command entries specifying the S operand to determine their legitimacy. If legitimate, this routine changes the screen options as requested by the operator and routes control to another Display Console Support routine as required.
- Delete 1, 2, 3, and 4 routines (IGC5607B, IGC5707B, IGC5807B, IGC5907B) - Erase answered WTOR messages from the screen as well as messages specified for deletion by the operator (CONTROL command, light pen, and cursor), or by the system (DOM macro instruction).

- Light Pen/Cursor Service routine (IGC5F07B) - Analyzes the type of function indicated by light pen or cursor operations and routes control to the appropriate Display Console Support routine.

- PFK 1 and 2 routines (IGC6A07B and IGC6B07B) - Analyze and enter commands requested by the depression of a PFK key or by a light pen selection of a displayed PFK number. These routines also process requests to define or redefine the commands associated with PFK keys.

- Timer Interpreter routine (IGC5K07B) - Analyzes timer intervals and passes indicators to the Processor routine to notify it as to which, if any, of the display consoles are ready to be rolled, assuming roll mode has been specified for one or more consoles.

- Status Display Interface routines 1 through 7 (IGC6L07B; IGC6M07B; IGC6N07B; IGC6O07B; IGC6P07B; IGC6Q07B; IGC6T07B) - Process multiple-line WTO requests and status displays to be displayed on an operator console.

- Cleanup routine (IEECVFTG) - Removes status displays from the message queues and reinitializes the screen area control blocks (SACBs), when a console undergoes a status change.

The following control blocks are used by Display Console Support routines:

- Communication Task Extended Save Area (CXSA) - Control block with the address of the UCM entry. MCS places the address of this control block into register 1 when it passes control to the Display Console Support routines.

- Unit Control Module Entry - Control area in Unit Control Module (UCM) with the addresses of a block of WQE pointers, UCB, DCM, and I/O Blocks. The UCM entry provides access to all information required by a display.

- Write Queue Element (WQE) - Messages on the output queue to be written.

- Unit Control Block (UCB) - Attention information.

- Display Control Module (DCM) - Console screen and support interface information.

- Storage Utilization Block (SUB) - Control information used by DIDOCS, RMS channel programs, and (if transient DCMs or PFK definitions are included in the system) direct access I/O information. There is only one SUB in the system.

- I/O Blocks - Status of device.

The DCM is unique for Display Console Support. The support requires one base DCM of 124 bytes, plus a unit DCM for each display operator console in the system. The size of each unit DCM varies according to the type of device being used.

The base DCM contains executable code to handle timer functions when in roll mode.

Each unit DCM is divided into two sections: a resident section and a section that may or may not be resident, at the user's option. The resident portion of the DCM (RDCM) contains screen control information and the address of the main storage area assigned to the optionally transient portion of the DCM (if the optionally transient portion of the DCM is actually transient, the main storage area addressed by the RDCM may be used by the transient DCMs of several consoles).

The RDCM may also contain one or more screen area control blocks (SACBs), which contain information about each status display area defined for the console's screen.

The optionally transient portion of the DCM (TDCM) contains two sections: the device independent section and the device dependent section. The device independent section contains:

- DOM addresses

- CCW area

- Input area

- Delete request buffer

- Processor name

- Option values

- Communication bits

- Field sizes for dependent section

The dependent section includes the following fields:

	Number of Bytes		
Field	2250	2260	M85
Screen Control Table	94	16	60
Screen Image Buffer	3848	960	2800
DOM ID Numbers	188	32	120
Buffer Address Table	10	6	8
CCW Chain for RMS	48	32	42
Secondary Screen Control Table	47	8	30

TRACE TABLE FACILITY

The Trace Table facility is an optional facility that may be specified during system generation. The facility provides a record of system conditions at the time of the following system events:

- SVC interrupts
- I/O interrupts
- Start I/O operations
- Dispatcher task switches

Each of the events is recorded in the Trace Table that is built in the nucleus by Trace routine IEATRC. Most entries contain the old PSW, the contents of registers 0, 1, and 15, and the current TCB address. When the Trace Table is filled, the Trace routine overlays old entries with new entries beginning with the oldest entries. For the format of the Trace Table, refer to Section 5, "Data Areas."

IEATRC

The Trace routine is resident in the nucleus and is loaded during nucleus initialization. The Trace routine is invoked by the SVC First-Level Interrupt Handler (SVC FLIH), the I/O FLIH, and the Dispatcher. When entered, the routine records the event in the Trace Table.

GENERALIZED TRACE FACILITY

The Generalized Trace Facility (GTF) is invoked as a system task when the operator issues the START command. When GTF is active, the supervisor Trace Table facility (a system generation option) is inhibited until GTF is stopped. When starting the GTF task, the operator may select to record the trace data either in main storage or in the SYS1.TRACE data set on an external device specified on the IEFRDER DD statement in the GTF cataloged procedure.

When the internal storage option has been selected, the data recorded for each event class is comparable to that recorded by the Trace Table facility. The event classes recorded are:

Event Class	
2	I/O Interrupts
3	SVC Interrupts
4	Dispatcher task switches
5	Start I/O operations
D	External and Program Check Interrupts

When the external storage option has been selected, the data recorded is more comprehensive and may include user-supplied trace data. Also, the operator may specify events within an event class, such as: I/O interrupts from specific devices, only certain SVCs, etc.

The GTF routines are entered from the interrupt handlers and the dispatcher when a HOOK macro instruction is issued, or from a user's program when a GTRACE macro instruction is issued. The macro instruction specifies the events to be recorded. The termination routines use the HOOK macro instruction to temporarily suspend GTF tracing, or to terminate GTF processing. For a more comprehensive discussion of the Generalized Trace Facility, refer to the Service Aids Logic, Program Logic Manual.

EXTENDED PRECISION FLOATING POINT SIMULATOR

The extended precision floating point simulator provides each System/360 and System/370 CPU with the capability of processing all of the eight extended precision floating point arithmetic instructions. The System/360 Models 85 and 195 and the System/370 models (with the extended precision feature) provide hardware support for all of the extended precision floating point instructions except the divide. The other System/360 models provide no hardware support of these instructions. Depending on the CPU, the supervisor simulates execution of the divide instruction or all of the extended precision floating point instructions:

Instruction	Assembler Mnemonic	Operation Code
Add normalized (extended)	AXR	36
Divide (extended)	*	
Load Rounded (extended to long)	LRDR	25
Load Rounded (long to short)	LRER	35
Multiply (long/extended)	MXD	67
Multiply (long/extended)	MXDR	27
Multiply (extended)	MXR	26
Subtract Normalized (extended)	SXR	37

*The divide instruction is simulated by the DXR macro instruction. See the publication Supervisor Services and Macro Instructions.

Relationship to the Operating System

The attempted execution of any of the eight extended precision floating point instructions that are not supported by CPU hardware causes a program interruption. The failing instruction is simulated by the supervisor if the user has provided the necessary linkage to the extended precision floating point simulator via the SPIE macro instruction (see Supervisor Services, GC28-6646). The three modules which comprise the simulator are:

- IEAXPSIM, which must be entered first to determine which of the other two modules is appropriate for the CPU.
- IEAXPALL, which simulates the 8 extended precision floating point instructions on System/360 CPUs that do not have hardware support for the instructions.
- IEAXPDXR, which simulates only the DXR instruction on the System/360 Models 85 and 195 and System/370 models.

SYSTEM MANAGEMENT FACILITY

The supervisor performs the following functions if the System Management Facility (SMF) feature has been selected at system generation:

- Maintains a record of system wait time.
- Assists in handling time limit expirations.
- Counts and records the number of references to user data sets.
- Controls the output limit for SYSOUT data sets.
- Records the number of 2048-byte blocks of storage assigned to a user program.

RECORDING SYSTEM WAIT TIME

Whenever the Dispatcher puts the system in the wait state, it places the contents of the interval timer in the first word of a special save area, SYSWSAVE. When an external or input/output interruption ends the wait state, the interruption handlers call the SMF Wait Time Collection routine. This routine reads the interval timer again and compares its value with the value stored by the Dispatcher to determine the elapsed system wait time. It then adds this elapsed time to the value in the second word of SYSWSAVE, giving the accumulated wait time for the system.

When a supervisor 10-minute timer interval expires, the Timer SLIH routine reads out the value in the second word of SYSWSAVE, which represents the total system wait time for the 10-minute interval. This

value is added to a field in the system management control area, SMCAWAIT + 4.

Each time the Step Termination routine of Job Management is entered, the total wait time recorded in the system management control area is checked. If it is nonzero, a system 10-minute wait record is generated.

HANDLING TIME/OUTPUT LIMIT EXPIRATION

Whenever a job, step, or wait time limit expires, the SMF Time Limit Expiration subroutine of the Timer SLIH routine passes control to a user time limit expiration routine. The user routine determines whether or not to grant a time limit extension. If an extension is granted, the SMF Time/Output Limit Expiration routine resets the TQE. If no extension is granted, the routine prepares for step termination in case of job/step time expiration, or prepares for abnormal termination in case of wait time expiration.

COUNTING REFERENCES TO USER DATA SETS

Whenever a reference is made to a user data set, the EXCP Counting routine records the reference in an EXCP counter. There is a counter for each data set/device combination. The counters are part of the TCT I/O Table segment of the timing control table.

CONTROLLING OUTPUT LIMIT FOR SYSOUT

Whenever a reference is made to a SYSOUT data set, the EXCP Counting routine checks for an output limit in the TCT I/O Table segment of the timing control table. If an output limit is specified, it is compared to the updated EXCP counters.

If the output limit is exceeded, the SMF Output Limit Expiration routine gives control to a user output limit routine. The user routine determines whether or not to increase the limit. If the increase is granted, the SMF Output Limit Expiration routine increases the output limit specified in the TCT I/O Table. Otherwise, the routine prepares for abnormal termination.

RECORDING STORAGE BLOCKS ASSIGNED TO USER PROGRAMS

Whenever the main storage supervision routines allocate or release 2048-byte blocks of storage within a region assigned to a user program, the following information is recorded in the timing control table:

- The highest address currently allocated from the bottom of the region. This address is called the "low water mark," or LWM.
- The lowest address currently allocated from the top of the region. This address is called the "high water mark," or HWM.
- The smallest amount of space within the region that has been available to this program at any one time (the minimum difference between the LWM and the HWM).
- The size of the region.
- The number of borrowed 2048-byte blocks currently assigned to the program (if the rollout feature is present).
- The highest number of 2048-byte blocks that have been assigned to the program at any one time (if the rollout feature is present).

If IBM 2361 Core Storage is included in the system, storage information is maintained both for processor storage and for 2361 Core Storage.

The following text and the flowcharts at the end of the section describe the routines that perform the management and service functions of the supervisor. The organization of this section corresponds to the organization of Section 2. Modules that perform similar functions (interruption supervision, task supervision, etc.) are grouped. Module names facilitate use of program listings (see "Section 4: Module Directory") and descriptive names provide reference to the "Method of Operation" section (Section 2).

INTERRUPTION SUPERVISION ROUTINES

The interruption supervision routines provide the interface between the problem programs in main storage and the other supervision routines. All services requested of the supervisor by the user or by other routines in the control program are scheduled by interruption supervision. When such a request is made, control is automatically given to one of the four interruption handling routines: supervisor call, timer/external, input/output, or program check. These routines determine the service requested and pass control to other supervisor routines or to user-supplied routines to perform that service. A machine check causes control to be passed directly to recording or recovery routines if these are included in the system.

SVC FIRST-LEVEL INTERRUPTION HANDLER (IEAAIH00) CHART AA

The SVC FLIH receives control as a result of a hardware-generated branch from the routine that issued the SVC instruction. After saving registers, a HOOK macro instruction is issued to inform the Generalized Trace Facility (GTF) to record the SVC interruption, provided the GTF is active. If GTF is not active, the SVC interruption is recorded in the optional Trace Table (a system generation option). The SVC code is then tested to determine if it is a valid request. If the SVC code is not IBM-supplied, or no entry for the SVC code exists in the SVC Table, the SVC FLIH issues an SVC 13 (ABEND) instruction to abnormally terminate the task that issued the SVC request. If the code is valid, the SVC FLIH loads register 3 with the address of the Communication Vector Table, register 4 with the address of the TCB for the current task (pointed to by OLD at IEATCBP+4 in the Dispatcher), and register 5 with the address of the current request block (pointed to by TCBRBP in the current TCB). If the request is for a type 1 SVC routine (the ESA for the entry in the SVC Table is b'000'), the type 1 indicator is set, and control is passed to the appropriate SVC routine. The return address for the type 1 SVC routine is that of the Type 1 SVC Exit routine, which follows the SVC FLIH in the module IEAAPS.

If the requested routine is other than a type 1 SVC, the SVC FLIH tests for a type 2 or 3 SVC request and branches to the SVC Second-Level Interruption Handler (SLIH). The first instruction in the SVC SLIH is a conditional branch dependent upon the condition code set by this test.

TYPE-1 SVC EXIT ROUTINE (IEAAIH00) CHART AA

This routine is entered by a branch from a type 1 SVC routine, including the SVC FLIH, and provides return linkage to the routine that issued the SVC, or linkage to the Dispatcher if a task switch is necessary. The type 1 SVC Exit routine resets the type 1 indicator to 0 and restores the registers from the SVC save area. If the pointers in the 2-word field IEATCBP in the Dispatcher contain the address of the same TCB (indicating that no task switch is necessary), if the pseudo-disable switch is set, or if the calling task is disabled for interruptions, the Exit routine loads the SVC Old PSW to return control to the calling routine. Otherwise (for example, if the TCB addresses are not the same and the task is enabled for all interruptions), a task switch is necessary. Registers 10 through 1 are saved in the current TCB, the SVC Old PSW is saved in the current request block, and control passes to the Dispatcher.

SVC SECOND-LEVEL INTERRUPTION HANDLER (IEAATA00) CHART AB

The SVC SLIH receives control from the SVC FLIH to further process requests for types 2, 3, and 4 SVC routines. The condition code set by the SVC FLIH prior to passing control to the SVC SLIH is used to separate type 2 requests from type 3 and 4 requests. The SVC SLIH saves the SVC Old PSW in the requester's RB. The SVC SLIH uses GETMAIN (IEAAMS) to obtain main storage for a supervisor request block (SVRB), which the SVC SLIH initializes and places on the RB queue of the TCB for the

requesting routine. If the request is for a type 2 SVC routine, the SVC SLIH branches directly to the requested routine. If the routine is a nonresident type 3 or 4 SVC, the SVC SLIH uses FINCH to load the routine into the SVC transient area if necessary before branching to the routine. Upon normal completion of the requested SVC routine, control is returned to the SVC SLIH which issued an SVC 3 instruction to pass control to the EXIT routine for type 2, 3, and 4 SVC routines.

Special processing occurs when main storage is not available for the SVRB. If the request originated in ABEND, and the current task is a subtask, the job step is terminated. In this case, the entire TCB queue is searched, and the highest priority ready task is located. The address of its TCB is placed in NEW (IEATCBP in the Dispatcher), and the SVC SLIH branches directly to the Dispatcher.

For other cases where there is no space within the partition for an SVRB, a pool of 3 SVRBs is provided within the SVC SLIH for initial and recursive ABEND processing. When in use, the fullword (USERTCB) following the third SVRB points to the TCB of the abnormally terminating task. When not in use, the fullword is set to zero. The use of each SVRB within the pool is indicated by the RBSIZE field; 0 indicates not in use.

The SVRB pool cannot be used concurrently by more than one task. If the pool is not available and main storage has not already been stolen, the SVRB is created in the beginning of the partition and processing continues as if main storage had been available (above). If main storage has already been stolen, the wait count in the request block is incremented, the Damage Assessment routine (DAR) dispatchability bit is set. The Subsystem Purge routine determines if the request is from GTF. If it is, GTF is disabled to prevent a system task from causing a system failure. Upon return, the resume PSW is enabled, the pointer NEW (IEATCBP) is set to 0 indicating that a task switch is necessary and the SVC SLIH branches to the Dispatcher.

EXTENDED SVC ROUTER (ESR) CHART AI

The primary function of the Extended SVC Router (ESR) is to provide linkage to Supervisor Service routines by logically extending the routing capability of the SVC Interruption Handlers. ESR does this via a secondary routing algorithm based on a parameter established prior to issuance of one of the ESR SVCs (109, 116, or 117). The interface provided to the Supervisor Service routine invoked by the ESR is identical to that provided by the SVC Interruption Handlers for Type 1, 2, 3 and 4 SVC routines except for an additional parameter used by the ESR for its routing.

ESR Processing for Type 1 and 2 Supervisor Service Routines

The Type 1 and Type 2 ESR Supervisor Routers are invoked via issuance of the Type 1 SVC 116 and Type 2 SVC 117, respectively. Upon entry to SVC 116 (IGC116) or SVC 117 (IGC117), the ESR code in register 15 is checked to ensure that a corresponding function exists. If the ESR code does not represent a supported function, the calling task is abnormally terminated. If the ESR code is valid, linkage to the appropriate resident supervisor service is made by converting the ESR code received in register 15 to an index value into an internal branch table of V-type address constants. The appropriate V-con is loaded into register 15 and control is passed to the routine via a branch instruction. Except for the use of register 15 for input of an ESR code, these SVCs (116 and 117) are identical to any other SVC routine. The same interface is presented to the Supervisor Service routines to which ESR is routing control as would have been presented to them by the SVC Interruption Handlers had they been invoked directly by an SVC number.

ESR Processing for Type 3 and 4 Supervisor Service Routines

The Type 3 and Type 4 ESR Supervisor Service Router is invoked via issuance of SVC 109 (IGC109). The ESR code in register 15 is checked to ensure that it is not greater than the highest assigned number. If the ESR code is invalid, the caller is abnormally terminated. Otherwise, the linkage to the appropriate supervisor service routine is made via XCTL. The XCTL parameters are developed by converting the ESR code passed in register 15 to a 3-character EBCDIC number which is appended to a prefix (IGC00) to form an ESR name (that is, if the binary value in register 15 is 89, the resultant ESR name would be IGC00089). This ESR name represents the first load of an SVC and is used as input to the XCTL service routine. Control is then transferred to the module represented by the ESR name via XCTL. Any subsequent loads of the Type 4 routine are named by incrementing the third and fourth characters of the original name (that is, several loads would be named IGC00089, IGC01089, IGC02089, etc.).

EXIT (IEAATA00) CHART AC

EXIT is used by types 2, 3, and 4 SVC routines, as well as asynchronous exit rou-

tines (STAE, SPIE, etc.) and routines entered by supervisor-assisted linkage, to return control to the calling routine or to pass control to the Dispatcher when a task switch is necessary. Because EXIT is a type 1 SVC routine and may be entered through the SVC FLIH as a result of an SVC 3 instruction, as well as by a branch, the type 1 SVC indicator set by the SVC FLIH must be reset to 0. EXIT is itself a linkage routine and does not use the type 1 SVC Exit routine.

If the exiting program is a user-provided program interruption routine (the SVC Old PSW is equal to the Program Interrupt Old PSW), the contents of registers 13 (which contains the resume address) through 2 are restored from the Program Interruption Element (PIE), the resume address saved in the PIE is moved to the SVC Old PSW, and the PIE is indicated as reenterable. EXIT loads registers 3 through 13 from the TCB of the exiting program and returns control to the interrupted program by loading the SVC Old PSW.

If the exiting routine is not a program interruption routine, EXIT frees all STAE control blocks created by the exiting routine that do not have the XCTL option specified. STAE control blocks are queued from the TCBNSTAE field in the TCB.

If the request block for the exiting routine is an IRB or an SIRB, EXIT dequeues the top IQE from the IRB. If the IQE has a 2-byte link field, the I/O Supervisor returns it to the queue of available IQEs. If the top IQE has a 4-byte link field, it is discarded unless there is a NEXAVL field in the IRB, in which case it is returned to this queue. If there is another IQE queued to the IRB, EXIT initializes the SVC Old PSW, places the address of an SVC 3 (EXIT) instruction in register 14, and loads the SVC Old PSW to return control to the asynchronous routine.

If there are no other IQEs on the IRB, or if the RB for the exiting routine is not an IRB or SIRB, EXIT determines whether the RB for the exiting program is the last RB on the queue. If not, the RB is dequeued and, if it is not a minor LPRB or if its use count is 0, its area and that of the routine is freed. If the next RB on the queue is in a wait state, NEW (IEATCBP) is set to 0. EXIT branches to the Dispatcher, which determines the routine that is to receive control next.

If the RB for the exiting routine is the last RB on the queue and its TCB is for a job-step task, EXIT issues an SVC 13 (ABEND) to normally terminate the task. However, if this task has any active subtasks, or has an ENQ count that is not zero, EXIT issues an SVC 13 instruction with the appropriate error code in register 1.

When the normal termination processing of ABEND is complete, an SVC 3 instruction is issued and processing resumes in EXIT as follows. EXIT frees the RB and the area for the routine, and provides processing for the end-of-task (EOT) exit routine. (If the user had not specified an EOT exit routine, the system EOT exit routine was specified by ATTACH. This routine is part of IEAATA00 and issues an SVC 62 (DETACH) to free the task and its TCB. The IRB is queued to the TCB of the mother task.)

EXIT removes the TCB from the TCB queue and sets NEW equal to OLD. Processing is optionally provided for time slicing, ECB posting, and job step timing. If POST (for ECB processing) did not change the contents of NEW, EXIT sets NEW to 0 and branches to the Dispatcher.

TIMER/EXTERNAL FIRST-LEVEL INTERRUPTION HANDLER (IEAAIH00) CHART AD

The T/E FLIH saves registers 2 through 9 within the T/E FLIH. A HOOK macro instruction is then issued to inform the Generalized Trace Facility (GTF) routines to record the external interrupt, provided that GTF is active and that class D monitoring has been specified. If the system is pseudo disabled, registers 10 through 1 and the T/E Old PSW are saved within the T/E FLIH. If the system is not pseudo disabled, registers 10 through 1 are saved in the TCB for the current task (pointed to by IEATCB+4 in the Dispatcher), IEAQWAIT is used to update the system wait-time values, and the T/E Old PSW is stored in current RB with the wait bit set off. The T/E FLIH determines from the T/E Old PSW the type of interruption that occurred, and passes control to the appropriate second-level interruption handler (external or timer).

Upon return from the second-level interruption handler, the T/E restores registers 2 through 9 and branches to the Dispatcher if the system is not pseudo disabled. If the system is pseudo disabled, all registers are restored and control is returned to the interrupted routine.

INPUT/OUTPUT INTERRUPTION HANDLER (IEAAIH00) CHART AD

The first instruction of the I/O FLIH is a NOP-branch switch, set to a branch by the first input/output interruption, allowing input/output interruptions, to be processed in groups. The first interruption of a group causes the I/O FLIH to execute some initialization instructions that prevent

any further execution of this "first-time logic" for successive interruptions in a group. Registers 2 through 9 are saved, and the subroutine IEAQWAIT is used to update the system wait-time values stored in the Dispatcher.

If the system is not pseudo disabled, the I/O Old PSW is saved in the current RB. The wait bit in the I/O Old PSW is set to zero (non-wait state), and registers 10 through 1 are saved in the TCB's general register save area. If the system is pseudo disabled, registers 10 through 1 are saved in the interruption supervision pseudo-disable save area, and the I/O Old PSW is saved.

The I/O FLIH branches directly to the part of the input/output supervisor that handles interruptions. When it regains control from the I/O supervisor, the I/O FLIH sets the NOP/branch switch to no-operation and restores registers 2 through 9.

The pseudo disable switch is tested. If it is off, the I/O FLIH enters the Dispatcher. If it is on, the I/O FLIH restores registers 10 through 1 from the pseudo disable save area, and returns control to the interrupted routine by loading the I/O Old PSW.

PROGRAM CHECK FIRST-LEVEL INTERRUPTION HANDLER (IEAAIH00) CHART AE

The Program Check FLIH saves all registers in the program check save area. An immediate branch is made to IHLMCIH to determine if the interruption was from either a System/370 or a simulated System/360 Monitor Call instruction. If it was, the Generalized Trace Facility (GTF) routines record the event and return control directly to the user's program, or to the Dispatcher if a task switch is required. Otherwise, the interruption is a valid program check and GTF is used to record the interruption and returns control to the Program Check FLIH to continue processing. Subsequent processing depends on whether a program interruption element (PIE) exists. A PIE exists if the problem program has issued a SPIE macro instruction. If no PIE exists, or if the program check occurred in a supervisor routine, control passes to ABTERM to schedule abnormal termination of the task. The PIE contains the address of an associated program interruption control area (PICA) which contains a mask indicating the type of program interruptions that the user-provided routine will handle. If this program check is not handled by the user's routine, control passes to ABTERM to schedule abnormal termination of the task. Otherwise, the Program Old PSW (at time of interruption) and registers 12 through 2 are moved to the PIE, register 14 is loaded with the return address of an SVC 3 instruction, the address of the user-provided routine is stored in the Program Old PSW, and control is passed to the routine by loading the Program Old PSW.

DISPATCHER (IEAAIH00) CHART AF

The second instruction in the Dispatcher is a NOP-branch to the Stage 3 Exit Effector. If this instruction has been made active by the Stage 2 Exit Effector, the Dispatcher branches to the Stage 3 Exit Effector to schedule requests for system asynchronous exit routines.

Processing by the Dispatcher depends on the contents of NEW and OLD. These are one word pointers within the Dispatcher at IEATCBP and IEATCBP+4 respectively which contain the addresses of the TCB for the task to be dispatched next (NEW), and the address of the TCB for the task which last had control (OLD). At system generation, both fields point to the first TCB on the queue. Other supervisor routines indicate to the Dispatcher that a task switch is necessary by altering the contents of NEW. Upon entry, NEW may be equal to OLD, indicating that no task switch is necessary; NEW may contain an address other than that contained in OLD, indicating that the task pointed to by NEW is the next one to be dispatched; or NEW may be zero, indicating that the Dispatcher is to determine which task is to be dispatched next. If there is a task in the system that is dispatchable, the Dispatcher will set both NEW and OLD to point to the TCB for that task before passing control to it.

If NEW equals OLD, no task switch is necessary. The Dispatcher queues any timer queue elements necessary and returns control to the current task by loading the resume PSW which is contained in the top RB on the TCB pointed to by OLD.

If NEW does not equal OLD, a task switch is indicated. The contents of registers 2 through 9 and the floating-point registers are stored in OLD's TCB. If job-step CPU timing is included in the system, and OLD's TCB has an address in its TCBPIB field, (indicating that OLD is not a system task), the Dispatcher queues the job-step timer queue element. If necessary, the Dispatcher dequeues timer elements associated with the task currently in control. Then it determines if NEW equals 0.

If NEW does not equal 0, it contains the TCB address of the task to be given control. The Dispatcher sets OLD equal to NEW, goes to the trace routine (if pre-

sent), or if active, the Generalized Trace Facility (GTF) by issuing a HOOK macro instruction to record the pending task switch. The Dispatcher then restores the floating-point registers (if present). If job-step CPU timing is included in the system, the dispatcher determines whether the TCB to receive control has a PIB pointer in its TCBPIB field. If there is a pointer, the task is not a system task, and the Dispatcher queues the job-step timer queue element. The Dispatcher then branches to a transient area refresh subroutine. Upon return, the Dispatcher branches to the task pointed to by NEW.

If NEW equals 0, the Dispatcher must examine the TCB queue to determine which task should be given control. This examination begins with the TCB addressed by OLD. (For a task of higher priority than OLD to receive control, the address of its TCB must be inserted in NEW by a supervisory routine).

When examining a TCB to determine if its associated task should be given control, the Dispatcher first determines if the request block (RB) of the program executing under the TCB is waiting. This is done by examining field XRBWT in the RB addressed by TCB field TCBRBP. If the RB is not waiting, the Dispatcher examines TCB field TCBFLGS to determine if the task is dispatchable. If so, the Dispatcher sets NEW and OLD to the address of the TCB and queues timer elements. Control then passes to the new task.

The Dispatcher does not pass control directly to the new task when the TCBRBP field for the task to be given control addresses an SVRB for a loaded transient SVC routine which is no longer present in the SVC transient area. In this case, the SVC routine was overlaid (before its execution completed) by another SVC routine. The Dispatcher determines whether the SVC routine currently in the transient area is the one required for the new task by comparing the contents of the doubleword XSNTCC with the XRBNM field of the SVRB. (XSNTCC is located in the SVC Second-Level Interruption Handler-IEAATA00, and contains the name of the SVC routine currently in the transient area). If the fields are indentical, the SVC routine currently in the transient area is the one required for the new task, and the dispatcher passes control to the new task.

If XSNTCC and XRBNM do not match, the Dispatcher prepares to reload the SVC transient area with the SVC routine indicated in the XRENM field. It issues a branch and link to the SVC SLIH for creation and initialization of a new SVRB. This SVRB represents the refresh request for the SVC routine that was overlaid before its execution completed. The PSW in the SVRB contains the address of the FINCH interface routine within the Dispatcher. The Dispatcher then examines OLD and NEW again. The same conditions exist as before except that the top RB on the TCB's queue is the SVRB for the FINCH interface subroutine. This module branches and links to FINCH which loads the requested routine into the SVC transient area. Upon return, the Dispatcher branches to the EXIT routine (IEAATA00) which will dequeue the SVRB for the FINCH interface routine, indicate that a task switch is necessary, and return control to the Dispatcher to dispatch the newly loaded SVC routine.

If the RB for the task is waiting or the task is nondispatchable, the task is not ready to receive control. The dispatcher then examines each TCB lower on the queue to determine whether it is ready to receive control. This process continues until a ready task is found or until the end of the queue is reached.

If no task is able to receive control, the Dispatcher sets the resume PSW wait bit of the TCB addressed by OLD. This PSW is then loaded, placing the CPU in a wait condition. The resume PSW is located in field XRBPSW of the RB addressed by TCB field TCBRBP.

If the System Management Facility is included in the system, the Dispatcher records the beginning of a system wait before loading the RB old PSW. It reads the interval timer and stores its value in the first word of a special save area, SYSWSAVE. This value is later used by the System Management Facility wait time collection routine to calculate elapsed system wait time.

Figures 29 and 30 illustrate how control is switched, assuming a three partition system in which P1 is inactive. All tasks are dispatchable except task P1. Initially, only the communications task and master scheduler task are waiting. Because task P0 is the highest priority task that is dispatchable and not waiting, it is given control. Task P0 has already enqueued and received exclusive control of a resource which task P2 will later enqueue (see Figure 30).

Dispatching the Communications Task and Master Scheduler Task

Figure 29 illustrates how control passes to the communications task and master scheduler task through the Dispatcher. In the example, the communications task receives control in order to read a DEFINE command from the operator console.

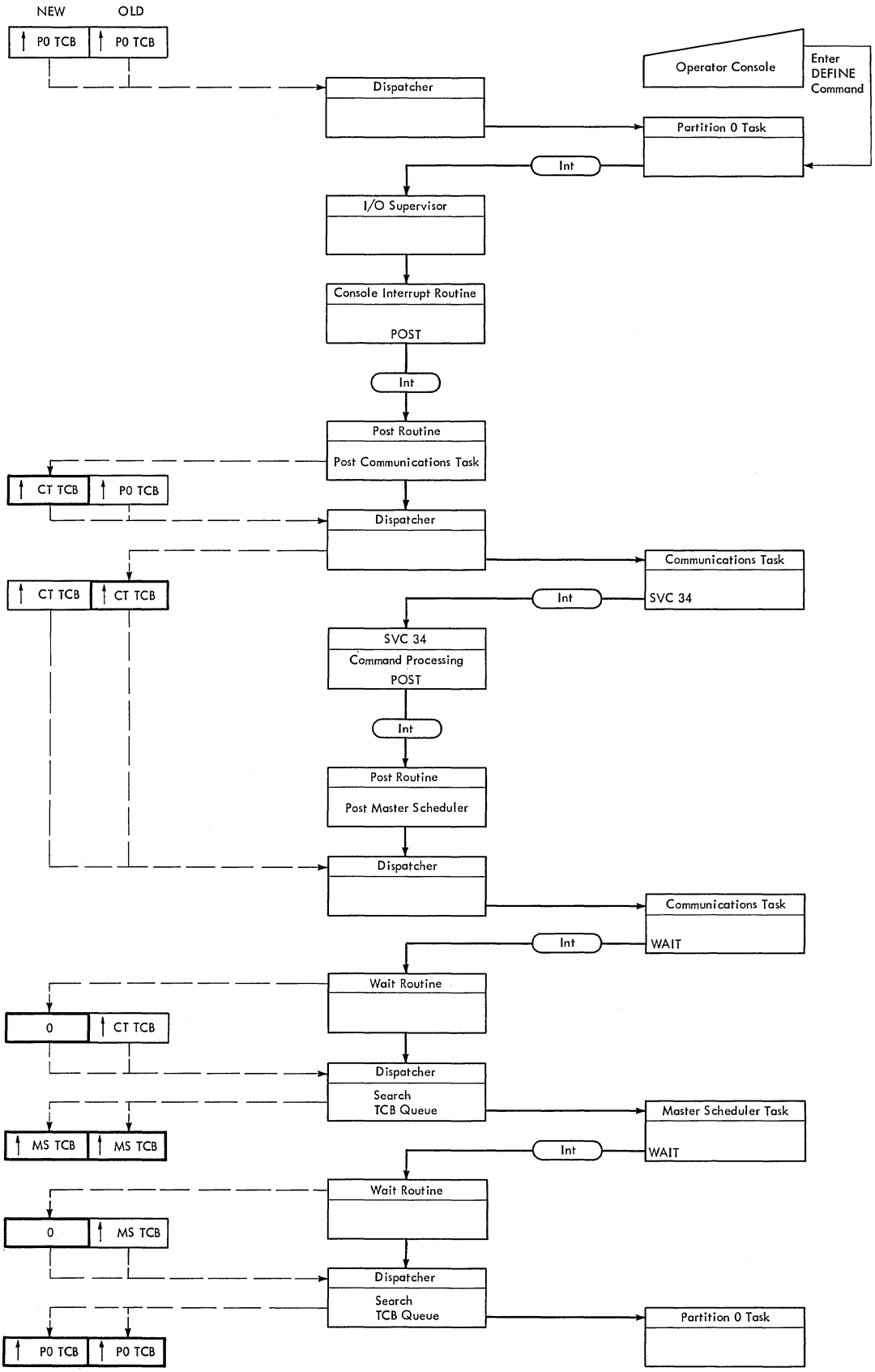


Figure 29. Dispatching Communications and Master Scheduler Tasks

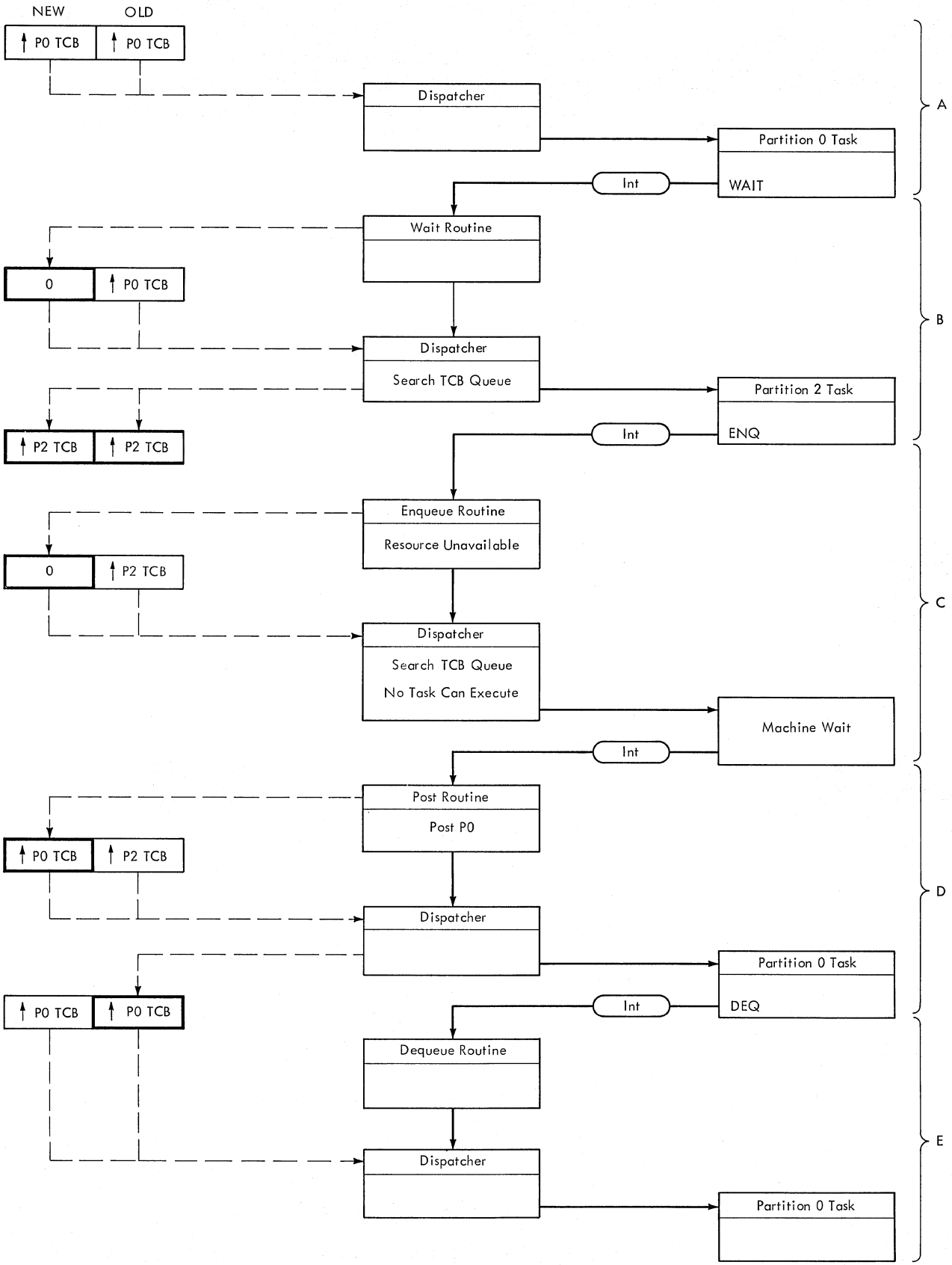


Figure 30. Task Switching

Initially, the task in P0 has received control from the Dispatcher and is executing. The operator presses the REQUEST key to indicate that he wishes to enter a command from the console. An I/O interruption is generated and control passes to the I/O supervisor which identifies the interruption as an attention signal. The I/O supervisor then passes control to the console interruption routine which issues a POST macro instruction. The POST routine posts the attention ECB and sets the communications task RB to a non-wait condition. Because the communications task is of higher priority than the task in partition 0, the POST routine places the address of the communications task TCB in location NEW. Control then passes to the Dispatcher.

The Dispatcher gives control to the communications task which passes control to resident device-support routines or issues SVC 72 for transient device-support routines. The device-support routines read the console's command and then issue SVC 34 to process the command. SVC 34 processes some commands completely but must pass control to the master scheduler resident command processor routine to complete processing the DEFINE command. SVC 34 issues a POST macro instruction to post the master scheduler task. The POST routine sets the master scheduler RB to a non-wait condition and gives control to the Dispatcher. Because the master scheduler task is of lower priority than the communications task, locations NEW and OLD remain unchanged and the dispatcher returns control to the communications task.

The communications task issues a WAIT macro instruction and waits on an ECB. The WAIT routine sets the communications task RB in a wait state and sets location NEW to 0. The Dispatcher then receives control and searches the TCB queue. Since the master scheduler task is the next ready task on the TCB queue, the address of the master scheduler TCB is placed in locations NEW and OLD, and the Dispatcher passes control to the master scheduler.

The master scheduler completes processing the DEFINE command and then issues WAIT. The WAIT routine sets location NEW to 0 and passes control to the Dispatcher which searches the TCB queue until it finds a task ready to receive control. In Figure 29, control returns to the task that was executing before the operator entered the DEFINE command.

Dispatching Tasks by Partition Priority

Figure 30 illustrates task switching among tasks executing in partitions.

A. The task in partition P0 (task P0) is the highest-priority ready task and is given control by the Dispatcher. When task P0 issues a WAIT on an ECB, an interruption occurs and control passes to the WAIT routine.

B. The WAIT routine places the RB for partition 0 in a wait condition and sets location NEW to 0. It then passes control to the Dispatcher which searches the TCB queue beginning with the TCB for partition 0. Since task P0 is waiting and task P1 is not dispatchable, the Dispatcher passes control to task P2, the highest priority task ready to execute. When task P2 attempts to request a resource through use of the ENQ macro instruction, an interruption occurs and control passes to the ENQ routine.

C. The resource is unavailable because task P0 has already requested it. Therefore, task P2 connot continue executing. The ENQ routine places zero in location NEW and then passes control to the Dispatcher which searches the TCB queue. The Dispatcher sets the wait bit in the resume PSW of task P2. The Dispatcher passes control to task P2, placing the CPU in a machine wait condition.

D. While the CPU is waiting, an interruption occurs signifying the completion of the event for which task P0 was waiting. The POST routine receives control and posts the ECB for task P0 which is now able to resume control. The POST routine places the TCB address for task P0 in location NEW and gives control to the Dispatcher. The Dispatcher sets OLD equal to NEW and gives control to task P0. When P0 is finished using the resource that it has requested, it issues a DEQ macro instruction.

E. An interruption occurs and the DEQ routine receives control. The queue element for task P0 is removed from the resource queue. The next element on the resource queue is for task P2. The resource is assigned to task P2 and its RB is placed in a non-wait condition. The DEQ routine then compares the priority of the task which has been in control with the priority of the task which is now ready. Because task P0 has a higher priority than task P2, location NEW remains unchanged. The DEQ routine passes control to the Dispatcher which returns control to task P0.

Dispatching a Task with Time Slicing

If the time-slicing option is selected at system generation in an MFT system without subtasking, one or more contiguous partitions will contain tasks that share CPU time. The number of time-slicing partitions, and the maximum amount of time each task will have, are determined by the time-slice parameters set at system generation. Both may be modified by the DEFINE command.

When a TCB of a partition within the time-slice partitions is the highest ready TCB, the Dispatcher will share the CPU time among all the ready TCBs within the time-slice partitions. These TCBs will share the CPU until none of the TCBs are dispatchable or a higher priority task is ready, at which time the Dispatcher will no longer be limited to the time-slicing tasks.

The Dispatcher determines whether the tasks are to share the CPU by testing the time-slice control element (see Figure 31). The TCB of the highest priority partition within the time-slice group is addressed by the field FIRST; the lowest priority partition within the group is addressed by the field LAST. The time-slice dispatching queue is that portion of the dispatching queue starting with the FIRST TCB and ending with LAST TCB. The Dispatcher stores the address of the TCB to next receive control in the field NEXT and allows each task to run for the interval given in the field LENGTH.

If time-slicing is selected in a system with subtasking, two additional fields are created within the TSCE, as shown in Figure 32. In addition to the addresses of the first and last TCBs, the priorities given to the time-slicing group are included. The value for FIRST is the initial dispatching or limit priority of the job step TCB in the highest priority partition in

FIRST - Address of the fist time-slice TCB on the TCB queue	4
LAST - Address of the last time-slice TCB on the TCB queue	4
NEXT - Address of the next time-slice TCB to be dispatched	4
LENGTH - Time-slice length (in milliseconds)	4

Figure 31. Time Slice Control Element without Subtasking

Highest Dispatching Priority	FIRST - Address of the first time-slice TCB
Lowest Dispatching Priority	LAST - Address of the last time-slice TCB
NEXT - Address of the next time-slice TCB	
LENGTH - Time-slice length (in milliseconds)	

Figure 32. Time Slice Control Element with Subtasking

the time-slice group; the value for LAST is one more than the dispatching priority of the partition below the LAST partition. Any TCB whose dispatching priority falls within the range of values from FIRST to LAST will be dispatched as a time-slice task.

When time-slicing is selected, the Dispatcher performs functions in addition to those explained in the preceding paragraphs. The following text describes the additional functions.

NEW Equals OLD and OLD is a Time-Slice Task: If the interval has expired (the time-slice queue element is not on the queue), the next ready time-slice task must be dispatched. The Dispatcher searches the time-slice group beginning with the TCB pointed to by NEXT in the TSCE through the TCB pointed to by LAST. If no task is found, the search continues resuming with the TCB pointed to by FIRST. When a ready task is found, NEXT is updated, the time-slice TQE is queued, and the ready task is dispatched. If no ready time-slice task is found, the Dispatcher searches the TCB queue for the highest priority ready task. If the interval has not expired, control is returned to the interrupted task.

NEW Equals OLD and OLD is not a Time-Slice Task: Control is returned to the interrupted task.

NEW Equals 0 and OLD is a Time-Slice Task: The Dispatcher removes the time-slice TQE from the queue and searches for the next ready time-slice TCB as described above in "NEW Equals OLD."

NEW is a Time-Slice Task and OLD is a Time-Slice Task: The task represented by OLD, if ready, is dispatched. If OLD is not ready, the time-slice TQE is removed from the queue, and the Dispatcher searches for the next ready time-slice task as described above in "NEW Equals OLD."

NEW is not a Time-Slice Task and OLD is a Time-Slice Task: The Dispatcher removes the time-slice TQE from the queue and dispatches the task represented by NEW.

NEW and OLD are not Time-Slice Tasks: The Dispatcher searches the TCB queue for the highest priority ready task. If this is a time-slice task, the Dispatcher searches the time-slice group as described above in "NEW Equals OLD."

Dispatching the 7094 Emulator Program for the Model 85

If the 7094 Emulator option is specified at system generation for a Model 85, the Dispatcher uses the Diagnose instruction to enter or leave the emulator mode. Each time the Dispatcher receives control, it checks bit 3 of the TCBTRN field of the TCBs addressed by NEW or OLD or both to determine if either TCB is that of the 7094 Emulator Program. If it is, the Dispatcher issues a Diagnose instruction to enter or leave the emulator mode.

Handling Job Step Timing When a Task Switch is to Occur

Job step timing is requested by a job step's Initiator via an STIMER macro instruction that specifies the TASK operand. The Dispatcher handles job step timing if a task switch is needed and the job step timing option was specified at system generation.

If these conditions are met, the Dispatcher suspends timing of the job step whose task is giving up control by branching to the timer DEQ Routine (IEAQTD01) to dequeue the old task's TQE. It then restarts timing of the job step whose task is next to be dispatched by branching to the timer ENQ routine (IEAQTE00) to add the new task's TQE to the timer queue.

Removing from the Timer Queue the TQE for the Job Step Associated with the Task that is Giving up Control: The Dispatcher does the following to determine if job step timing is being performed for the job step that is giving up control:

- It tests the TCBPIB field of the TCB for the address of a PIB. If this field contains 0's, a system task is giving up control and therefore has no job timing associated with it.
- If the TCBPIB field contains a PIB address, the Dispatcher tests the high-order bit in the job step timing status bits field of the PIB. If this bit is off, an initiator, a reader, or a writer is executing in the partition and also has no job step timing associated with it.
- If the bit is on, the Initiator has requested job step timing for a problem program and a job step TQE is on the timer queue. The Dispatcher therefore examines the TQEFLGS field to determine the TQE type. If the TQE is a REAL type, it will not be removed from the timer queue because it represents a wait time limit interval. It was established as a wait time limit TQE and queued by the WAIT routine (IEAAWT), and it will be dequeued by the POST routine (IEAAPT).

If the TQE is a TASK type, the Dispatcher branches to the timer second level interruption handler, to suspend job step timing for the old task.

Placing on the Timer Queue the Job Step TQE for the Job Step Associated with the Task to be Dispatched: The Dispatcher does the following to determine if job step timing is being performed for the job step that is about to be dispatched:

- It tests the TCBPIB field of the TCB for the address of a PIB. If this field contains 0's, a system task is receiving control and therefore has no job step timing associated with it.
- If the TCBPIB field contains a PIB address, the Dispatcher tests the high-order bit in the job step timing status bits field of the PIB. If this bit is off, a reader, a writer, or an initiator is receiving control and also has no job step timing associated with it.
- If the bit is on, job step timing has been requested for the problem program in the partition via a STIMER macro instruction issued by the Initiator. The Dispatcher therefore branches to the timer ENQ routine (IEAQTE00) in the Timer Second-Level Interruption Handler to add the TQE to the timer queue and restart job step timing for this task that is about to be dispatched.

ATTACH (WITH SUBTASKING) (IEAQAT00) CHART BA

ATTACH first determines if the request was made within a STAE user-exit routine. If so, ATTACH loads a return code of 4 into register 15 and issues an SVC 3 (EXIT) instruction to return control to the caller. Otherwise, ATTACH examines the TCB address table to determine if there is an empty field. If no empty field is found, the caller is abnormally terminated because the limit on the number of subtasks established at system generation has been surpassed.

If an empty slot in the TCB address table is found, ATTACH issues a GETMAIN macro instruction requesting the allocation of space from the system queue area for the subtask TCB. The address of the new TCB is placed in the TCB address table, and the location of the TCB table entry is indicated in the TCBID field of the new TCB. ATTACH initializes the TCBMSS field with the address of the boundary box, and, if jobstep timing is in the system, ATTACH initializes the timer queue element.

ATTACH then obtains main storage from the partition for a dummy program request block (PRB), creates a gotten subtask area queue element for the PRB, chains the GQE to the boundary box of the new TCB, and places the PRB on the RB queue of the new TCB. ATTACH requests additional main storage for use as a save area by the subtask. The address of the save area is stored in the TCBFSA field of the new task control block.

If the ETXR operand has been specified in the ATTACH macro instruction and an interruption request block (IRB) already exists, indicating that the requested exit routine is already scheduled for execution, ATTACH acquires main storage for an interruption queue element (IQE) to indicate that an additional request for the exit routine is pending. If an IRB does not exist for the desired exit routine, ATTACH obtains main storage for an IRB and IQE and branches to the CIRB (construct IRB) routine to initialize and schedule the IRB.

ATTACH determines in the following manner whether an IRB already exists: the ATTACH routine searches the subtask TCB queue belonging to the caller's TCB, looking for a subtask TCB that indirectly points to the same end-of-task exit address as that specified in the caller's ATTACH macro instruction. A subtask's exit-routine address is determined indirectly via the TCBIQE field of the subtask's TCB. The TCBIQE field points to an interruption queue element (IQE), if one has been created for the subtask. If the TCBIQE field is 0, this subtask has no IQE, and thus no end-of-task routine has been requested for this subtask. If a subtask's TCB IQE field is not 0, it points to an IQE, which points to an IRB, which points to an end-of-task exit routine for the subtask. If the exit routine address in the subtask's IRB is equal to the end-of-task exit routine address specified in the caller's ATTACH macro instruction, an IRB for the desired exit routine already exists.

If ATTACH successfully obtains storage for the IQE, it initializes the fields of the IQE and increases the "use" count (RBUSE) of the IRB by one. This use count indicates the number of subtasks that use the same end-of-task exit routine.

If neither an ECB address nor an ETXR address was specified in the ATTACH macro instruction, ATTACH schedules the system end-of-task (EOT) routine by constructing an IRB and IQE specifying the EOT routine. The system end-of-task routine, a subroutine of the EXIT routine, issues an SVC 62 (DETACH) instruction to free the subtask and its TCB when processing is complete.

The ATTACH routine initializes the newly created subtask TCB. If the ETXR operand was specified in the caller's ATTACH macro instruction or if the system EOT routine is scheduled, the address of the IQE is placed in the TCB (TCBIQE field), and a flag is set to indicate that an end-of-task exit routine has been requested.

If the ECB address has been specified, the ATTACH routine ensures the validity of the ECB address and places the ECB address in the TCBECB field of the new TCB. If the ECB address is invalid (does not specify a fullword boundary or violates storage protection), the ATTACH routine issues an ABEND macro instruction to abnormally terminate the caller.

The ATTACH routine next determines the limit and dispatching priorities from input parameters, and stores the priorities in the new TCB. The limit priority of the subtask is set according to input parameters but not higher than the limit priority of the caller's task. The dispatching priority of the subtask is similarly set

according to the input parameter but not higher than its own limit priority. If the priority parameters have not been specified by the caller, the ATTACH routine sets the subtask priorities to that of the calling task.

If the time-slicing feature is included in the system, the ATTACH routine tests whether the new TCB represents a time-sliced task (is within the range of dispatching priorities recorded in the TSCE). If the dispatching priority is not within range, the new task is not a member of the time-sliced group and further time-slice processing is bypassed. If the new task is a member of the time-sliced group, ATTACH sets the time-slice bit (TCBFTS) in the TCB. If the new TCB represents the only task in the group, its address is placed in the FIRST, LAST, and NEXT fields of the TSCE. ATTACH then queues the TCB on the dispatching and subtask TCB queues. ATTACH queues the new TCB to the subtask task queue by adjusting the TCBOTC, TCBNTC, and TCBLTC fields of the new TCB.

ATTACH determines whether the current program or the subtask will receive control next from the dispatcher. The ATTACH routine examines the dispatching priorities of the two tasks and stores the address of the TCB for higher priority task in a TCB pointer (IEATCBP) to be later tested by the Dispatcher when it receives control during the exiting procedure.

ATTACH places the address of an SVC 6 (LINK) instruction in the resume PSW in the dummy PRB. This will cause the loading and execution of the requested program when the SVC 6 is executed.

The ATTACH routine prepares for exiting by storing the caller's registers in the subtask's TCB. ATTACH then issues an SVC 3 (EXIT) instruction.

CHAP (ONLY WITH SUBTASKING) (IEAQCH00) CHART BC

The CHAP routine receives control as a type 2 SVC routine from the SVC SLIH or serves as a subroutine via a branch entry from a supervisor routine. If the input parameter is zero, then the caller wants to change the dispatching priority of his own task. Otherwise, CHAP tests the address of the TCB passed by the caller for validity. (Validity testing is bypassed if the caller is a system task.)

If time-slicing was selected during system generation, and the task is a member of the time-slice group, CHAP dequeues the TCB from the Time-slice queue.

CHAP calculates the new dispatching priority for the TCB and queues the TCB in a new location on the dispatching and Time-slice queues.

Placing the Changed TCB on the Dispatching and Time-Slice Queues

CHAP performs several tests to determine the extent of the priority change that can be permitted. The first test checks whether the result of the change in dispatching priority is 0 or negative. The dispatching priority of the changed TCB is set to 0 whenever the calculated priority is less than or equal to 0.

If the calculated priority is greater than 0 and less than or equal to the limit priority of the caller, CHAP stores the calculated dispatching priority in the TCBDSP field of the changed TCB. If the limit priority of the changed TCB is less than its new dispatching priority, CHAP sets the limit priority equal to the new dispatching priority.

If the calculated dispatching priority is greater than 255 or greater than the limit priority of the caller, CHAP propagates the limit priority of the caller to the limit (TCBFTLMP) and dispatching (TCBDSP) fields of the TCB to be changed.

If the time-slicing feature is included in the system, CHAP tests whether the new dispatching priority falls within the priority range represented in the TSCE. If the task is now a time-slice task, CHAP sets the time-slice bit (TCBFTS) in the TCB. The CHAP routine tests the NEXT field in the TSCE. If the field contains 0, the specified task is the only member of the time-slice group, and the CHAP routine places its TCB address in the FIRST, NEXT, and LAST fields in the TSCE. If the NEXT field in the TSCE is not 0, the FIRST or LAST field is adjusted if necessary (NEXT is not changed).

DETACH (ONLY WITH SUBTASKING) (IEAGED02) CHART BB

DETACH first tests the validity of the subtask TCB address. If any validity check fails, DETACH loads an error code and issues an ABEND macro instruction to abnormally terminate the caller's task.

If the input address is valid, DETACH determines if the caller belongs to the parent task of the specified subtask. If the subtask TCB address is not found on the caller's subtask queue, DETACH loads the same error code as that for an invalid TCB address, and issues an ABEND macro instruction to abnormally terminate the caller's

task. If the specified subtask TCB address is found in the subtask queue, processing continues.

The DETACH routine next determines if the subtask is complete (that is, whether the task has been terminated by either the EOT or ABEND routine) by testing the "completion" indicator TCBNDSP3 in the TCBFLGS field of the subtask TCB. This indicator bit is set by the EOT routine or by the ABEND routine. If the subtask has been terminated, the DETACH routine:

- Removes the subtask TCB from the subtask queue of the caller's TCB. This is necessary since the ABEND or EOT routines will try, when a later request to terminate the caller's task is issued, to release resources belonging to its subtasks.
- Frees the main-storage areas belonging to the subtask that were not freed during the termination processing. These areas include the main storage occupied by the subtask TCB and the optional 72-byte problem-program register save area. (The address of the problem-program save area is contained in the TCBFSA field of its TCB.)
- Returns control to the caller.

If DETACH discovers that the subtask was not normally or abnormally terminated, it initiates abnormal termination of the subtask, unless it is already being terminated. (The TCBNDSP3 flag, if set, indicates that the subtask has been normally or abnormally terminated.) If the subtask is not already being terminated, DETACH loads an error code and branches to ABTERM to schedule the abnormal termination. The DETACH routine cannot invoke the ABEND routine directly, since the caller's task is not to be terminated. The ABEND routine, when invoked directly, can terminate only the current or calling task.

DETACH next indicates that the subtask has been abnormally terminated and saves (in its SVRB) the address of the subtask's event control block (ECB), if one was specified when the subtask was attached. (The ECB address is contained in the subtask's TCBECB field.) DETACH then obtains four bytes of space for a new ECB in which the ABEND routine can post the subtask's termination. (The actual posting is performed for the EOT routine, which is invoked by the ABEND routine when the termination is complete.) DETACH initializes the new ECB to 0 and places its address in the subtask TCB. DETACH also clears the IQE pointer (TCBIQE) in the subtask TCB, so that an end-of-task exit routine (if one exists) will not be scheduled by the EOT routine when the subtask is terminated. (The TCBIQE field contains an indirect pointer to an end-of-task exit routine if the caller specified the ETXR operand when it attached the subtask.)

DETACH issues a WAIT macro instruction to wait until the ABEND routine completes the abnormal termination of the subtask. The abnormal termination of the subtask is signaled by the posting of the new ECB by the EOT routine. DETACH then frees the storage occupied by the special ECB it had created and tests whether the subtask has its own ECB. (DETACH saved the subtask's ECB address -- if it had an ECB -- in the current SVRB.)

If the subtask does not have an ECB, DETACH cannot inform the caller of the subtask termination when control is returned to the caller. If, however, the subtask has an ECB, DETACH uses the EOT validity check routine to ensure that the ECB contains valid information and has not been altered by a user program. This check is needed since the POST routine will not make the check when it posts the ECB. If the user program has altered the ECB contents, and this alteration remains undetected, processing of the ECB by the POST routine causes a program check.

If the ECB does not contain a valid RB address, DETACH loads an error code and invokes the ABEND routine to abnormally terminate the caller's task. If the ECB is valid, DETACH posts the ECB to indicate to the caller that the subtask has been abnormally terminated. Then, via the EXIT routine and the Dispatcher, it returns control to the caller.

EXTRACT (WITHOUT SUBTASKING) (IEAAXR00) CHART BD

The EXTRACT routine is entered from interruption supervision when the EXTRACT macro instruction is issued. Upon entry, EXTRACT sets all fields in the list area specified by the user to zero, except for the task input/output table (TIOT) address field. If the macro instruction parameters specified TIOT or ALL, the address in the TCB of the TIOT is inserted into its respective field in the user's list. EXTRACT issues an SVC 3 (EXIT) instruction on completion.

EXTRACT (WITH SUBTASKING) (IEABXR00) CHART BE

The EXTRACT SVC routine permits the macro-issuing or calling program to obtain from a specified TCB or a subsidiary control block the information contained in

seven of its fields. The specified TCB must be either the TCB of the issuing program or of one of its subtasks -- tasks attached by the issuing program. The information may be extracted from any combination of the seven fields or from all of the seven fields. When extracted, the information is placed in a user-specified list. The fields from which information may be extracted and the information contained in each field are described in the publication Supervisor Services and Macro Instructions.

Besides extracting information from a specified TCB or a subsidiary control block, EXTRACT performs several checks to determine if the input parameters passed by a problem program are valid. This validity checking prevents the extraction of meaningless data, or the later occurrence of a program interruption whose cause may be difficult to interpret. Invalid parameters cause the routine to generate an error code and cause an abnormal termination of the task.

EXTRACT first tests type of entry and sets indicators accordingly. It also determines whether information is to be extracted from the current TCB (caller's TCB) or from the TCB of one of its subtasks. If information is to be extracted from the current TCB, its address is set up, and the following test for a subtask is bypassed.

If the specified TCB represents a subtask of the caller's TCB, EXTRACT prevents a possible program interruption by forcing the TCB pointer (second word of the input parameter list) to a fullword boundary. The routine then scans the subtask queue of the caller's TCB to determine if the specified TCB address represents a subtask of the caller's TCB. If no match of TCB addresses can be obtained, the caller must have incorrectly specified the TCB address. Since useful information cannot be obtained from the specified TCB, EXTRACT issues an SVC 13 (ABEND) instruction specifying error code 328.

EXTRACT next passes control to the supervisor's Validity Check routine to perform the needed checking. The Validity Check routine performs three tests to determine the correctness of the extract list address. The extract list is the user-specified table in which the EXTRACT routine places the requested TCB information. One test determines if the list address lies on a fullword boundary, as required. Another test checks whether the list address lies within the boundaries of main storage. The remaining test determines if the list address specifies a storage area whose storage protection key matches the protection key in the caller's TCB. If any of these tests indicates that the calling program has incorrectly specified the extract list address, EXTRACT issues an SVC 13 (ABEND) instruction.

If parameters have been specified correctly, as indicated by the several validity checks, or if validity checks have been bypassed, normal processing of the requested TCB information continues. EXTRACT tests each bit of an extraction byte, a part of the parameter list, which represents the FIELDS parameter of the EXTRACT macro instruction. For each bit that is set, EXTRACT places the appropriate information from the specified TCB into the user list. If a bit is not set, the routine makes no entry in the list for the field represented by that bit. The resulting list is of variable length and packed in a standard order. (See Supervisor Services and Macro Instructions.)

Note that some of the fields that may be requested are not contained in the specified TCB. These fields are those requested by the following parameters: GRS (general register save area), FRS (floating-point register save area), the AETXR (end-of-task exit routine), and COMM (CSCB communications list). For the first two parameters the returned value is the address of the appropriate save area. The value is calculated from the address of the specified TCB. The returned value points to the save areas which are in the TCB. For the third parameter, AETXR, the returned value (address of the exit routine) is obtained indirectly from the TCB. The TCBIQE field in the TCB points indirectly to the end-of-task exit routine, via pointers in an interruption queue element (IQE) and an interruption request block (IRB). The address of the end-of-task exit routine is obtained from the IRB. For the fourth parameter, COMM, the returned value (address of the communications list) is also obtained indirectly. The TCB points to the JFCB, which points to the CSCB in which the communications list resides. When EXTRACT has placed all the requested information in the user-specified list, it issues an SVC 3 (EXIT) instruction.

WAIT (IEAAWT00) CHART BF

WAIT tests the wait count specified as an operand of the WAIT macro instruction. If the wait count is 0 or negative, the WAIT routine ignores the request and branches to the type 1 Exit routine to return control to the caller.

If a wait count greater than 0 has been specified, WAIT sets the system mask in the SVC old PSW to 1's. Thus, when the SVC old

PSW is loaded to pass control back to the caller, the caller will be enabled for I/O and external interruptions. (This action prevents supervisor routines that operate disabled and use the WAIT macro from placing the task into a disabled wait state.)

WAIT then tests register 1. If register 1 has the address of a single ECB, WAIT stores the address of the ECB in location WTSIMLST and flags the ECB address to appear as the last address entry in a parameter list. WAIT then loads the address of location WTSIMLST into register 1, thereby simulating a list of ECBs containing only one entry.

If the validity check option is in the system and the caller is a user program, WAIT branches to the validity check subroutine to test each ECB address. If an error is indicated, WAIT determines if the communications ECB was specified. If so, normal WAIT processing continues. If the ECB specified is not the communications ECB, the task is abnormally terminated. If the caller is a system routine, indicated by a 0 protection key, WAIT does not verify the ECB addresses passed by the caller.

After checking the validity (optional) of the ECBs, the wait count is saved and a register is initialized with the address of the ECB from the macro instruction parameter list. An ECB counter is incremented as each ECB is addressed.

WAIT tests the wait bit in the ECB. If the wait bit is on, an error condition exists, and WAIT then branches to the ABTERM routine to schedule abnormal termination. If the wait bit is off, the completion bit is tested. If the completion bit is off, indicating that POST has not posted the event as being completed, the wait bit is turned on, and the address of the current RB is placed in the ECB.

If the completion bit is on, the wait count is decremented by 1 and the remainder is tested for 0.

1. If the wait count is 0, and if the number of events being waited on is less than the number of specified ECBs (as indicated by the "search" flag in the caller's RB), WAIT clears the wait bit in each ECB that has not been posted. The purpose of clearing the wait bit in each ECB that has not been posted is to prevent POST from later mistaking an uncleared wait in a ECB for a new wait request. Since the wait count is 0, all events being waited on have occurred, and therefore the caller must not wait. WAIT stores the wait count in the caller's RB and returns control by branching to the type 1 Exit routine.

2. If the wait count is not 0, then another ECB is addressed and the tests described in the preceding paragraphs are repeated. If the ECB tested was the last or only ECB, WAIT continues.

If all the ECBs have been tested and the wait count has not become 0, the total number of ECBs specified is compared to the original wait count. If the number of ECBs specified is less than the count, the count cannot be satisfied and the task is abnormally terminated.

If the wait count is less than the number of ECBs, a bit (the "search" flag) is turned on in the RB to indicate to POST that a multiple-event wait has been issued. (In a multiple-event wait, the number of ECBs is greater than the wait count.) This means that the caller wishes to wait on fewer events than specified. (For example, the caller may wait on the completion of any one of three possible I/O operations.) When an event being waited on takes place (the single I/O operation), the wait bit remaining set in each of the two ECBs not yet posted is now misleading. These bits may cause later incorrect processing by POST. If the search bit is set, POST clears the wait and search bits in each of the ECBs not yet posted.

When the WAIT routine has processed all ECBs in the parameter list, it inserts the final wait count in the wait count field (XRBWT) of the caller's RB. Since the wait count is greater than 0, the caller is now in the wait condition, or "waiting", and cannot be dispatched. In this case, the WAIT routine sets NEW to 0 to force a task switch and returns by branching to the Type-1 Exit routine.

POST (IEAAPT00) CHART BG

There are four branch entry points to POST:

- IEAAPT01 is used by supervisor routines.
- IEA0PT01 is used exclusively by the I/O supervisor.
- IEA0PT01+4 and IEAAPT01+4 are used by the I/O supervisor and task management supervisor, respectively, when these routines do not need to check the validity of user-specified event control blocks.

An additional entry point, IGC002, is entered when the caller issues an SVC 2 or a POST macro instruction.

If validity checking has not been bypassed, POST checks the validity of the ECB address passed by the caller. In addition, if the wait bit is set in the specified ECB, POST checks the validity of the RB address contained in the ECB.

The next step is checking the completion bit in the ECB. If the completion bit is set, indicating that the event to be posted has already been posted, there is no need for further processing. The wait flag is cleared and control is returned to the caller via the Type 1 Exit routine. If the completion bit is not set and the wait flag is not set, POST stores a completion code in the ECB, overlaying the RB address field. (The completion code was specified by the caller of POST as an operand of the POST macro instruction.) POST then sets the completion bits, clears the wait bit in the ECB, and returns control via the Type 1 SVC Exit routine. These bits indicate to WAIT as well as to POST and the caller that the specified event has occurred.

If the wait flag is set, POST next determines whether to decrease the wait count stored in the waiting program's RB. The wait count indicates the number of specified events that must occur before the waiting program can resume execution. The wait count is tested by POST:

- If the wait count is 0, POST stores the completion code in the ECB, adjusts the completion and wait flags, and returns control to the caller via the Type 1 SVC Exit routine. (This can occur in the special case in which the waiting program is being abnormally terminated due to an event asynchronous to the waiting program. The ABTERM routine resets to 0 the wait count in the top RB on the RB queue.)

- If the wait count is not 0, POST subtracts 1 from the wait count (XRBWT) in the waiting program's RB and determines whether the wait count is now 0.

  1. If the reduced wait count is not 0, all events awaited by the program have not yet occurred. The completion code is stored in the ECB, the completion and wait flags are adjusted, and control is returned via the Type 1 SVC Exit routine.

  2. If the reduced wait count is 0, processing continues.

POST next determines if the specified ECB points to the top RB on the queue. If not, processing continues with the test for the multiple-event wait (below). If this top RB has a REAL job-step TQE (this is for a 30-minute wait) on the timer queue, the timer dequeue subroutine in the timer SLIH is used to remove it from the queue, the TQEVAL is restored, and the TQE is again indicated as TASK type. This allows the Dispatcher to again calculate the CPU time used by this job step.

Because the program is no longer awaiting the completion of events, POST must next determine from the "subject" TCB (the TCB pointed to by the top RB) if the task is dispatchable and, if so, which task is to be dispatched next. If the task is not dispatchable, processing continues with the test for the multiple-event wait (below). If the task is dispatchable and NEW (the pointer in the Dispatcher to the task next to be dispatched) is 0, or if NEW is not 0 but the "subject" TCB has a higher dispatching priority than the task pointed to by NEW, the address of the "subject" TCB is placed in NEW to indicate to the Dispatcher that a task switch is necessary. If the two tasks have the same dispatching priority, POST puts the address of the "subject" TCB in NEW only if it is placed higher on the TCB queue. If it is placed lower on the queue, NEW remains unchanged.

The multiple-event wait allows the user to await the completion of more than one event. If only one ECB was specified in the WAIT macro instruction, POST stores the completion code in the ECB, sets the completion flag, clears the wait flag and passes control to the Type 1 SVC Exit routine. If more than one ECB was specified, POST ensures that the wait flag in all the ECBs specified by the WAIT macro instruction are set to zero. POST then stores the completion code, adjusts the completion and wait flags, and branches to the Type 1 SVC Exit routine.

ENQ (IEAGENQ1) CHART BH

The ENQ macro instruction is issued as a conditional or unconditional request. The differences in ENQ processing when the 'RET=' parameter is specified as 'TEST', 'USE', or 'HAVE' are discussed separately under "Conditional Requests."

Unconditional Request: If the "set must complete" option was specified by a user program, the task is abnormally terminated (an SVC 13 is issued) and a return code is placed in register 1. ENQ then verifies parameters passed by the caller if the caller is a user routine. This protects the queues by preventing a possible program interruption during queue manipulation. If the caller is a system routine (the RB old PSW has a 0 protection key), the parameters are assumed to be correct. If the caller is a user routine and an addressing error is found, ENQ sets an error code and issues

an SVC 13 (ABEND) to abnormally terminate the program. If the rname length is 0, the code is set to 238 before the task is abnormally terminated. If the caller's protection key is not 0 and the SMC= operand was specified, the completion code is 338. If the parameter list is not on a fullword boundary, the completion code is 438. (See Figure 33 for a description of ENQ return codes.)

ENQ then creates, initializes, and queues the necessary control blocks for the resource requests of the task that issued the ENQ macro instruction. (See Section 5 for a description of the control blocks.) Each resource request will be represented by three control blocks. The major queue control blocks (QCBs) will be chained to the last major QCB on the queue or, if no others exist, to IEAQCB01. The minor QCBs will be chained to their associated major QCB and the queue elements (QELs) to their associated minor QCB. Main storage required for the control blocks is acquired in the system queue area (subpool 255) via an SVC 10 (GETMAIN). ENQ searches the queues, starting with the major QCB queue, and builds control blocks and performs additional processing according to the existing control blocks that represent the requested resource:

- If there is no existing major QCB for the qname specified, or if there is a major but no minor QCB for this resource, control of the resource has not been previously requested. ENQ processes the necessary control blocks and increments the enqueue count (TCBLMP) in the requester's TCB by one. If the rname length specified in the parameter list element is incorrect, an error code is set in register 1 and the task is abnormally terminated.

- If a major and minor QCB already exist for this resource, ENQ then tests for a duplicate request by the same task.

Return Code	Meaning
8	The caller's task is already enqueued. (This is an error condition.)
4	The resource is in use, and the caller's request has not been enqueued.
0	The resource is available, or the caller's request has been enqueued.

Figure 33. Return Codes for the ENQ Routine

This occurs when the requesting program issued a second ENQ macro instruction for the same resource without an intervening DEQ macro instruction, and is indicated by the existence of a QEL containing the requester's TCB address. If a QEL is found, ENQ places an error code in register 1 and issues SVC 13 (ABEND) to abnormally terminate the program. If a duplicate QEL is not found, ENQ determines the availability of the resource by testing the type of request made and the types of previous QELs on the queue. If the request is for shared control, and the previous requests were for shared control, the requester can get immediate control of the resource. If a QEL representing an exclusive request is already queued for the resource, the resource is under the control of another task, and the RB wait count in the SVRB is incremented by one. In either case, ENQ creates a QEL and increments the enqueue count in the TCB by 1. This count, which is decremented by the DEQ service routine when the previous exclusive requesters have completed use of the resource and issued a DEQ, causes the requesting task to wait in ENQ but does not effect the dispatchability of the task as a whole. Asynchronous routines, represented by IRBs on the TCB's request block queue, can still operate under the task's control.

After all the resources requested in the ENQ macro instruction have been processed, ENQ determines which routine is to gain control and branches to it. The operations performed by ENQ depend on the availability of the resources and on whether "set must complete" was specified.

- If the resources are available (the RB wait count in the SVRB equals 0), ENQ determines if "set must complete" was specified. If it was not specified, ENQ issues an SVC 3 which will return control to the caller via EXIT and the Dispatcher. If it was specified, ENQ places the caller's task in "STEP or SYSTEM must complete" status. For a system must complete request, nondispatchability flags are set in all the TCBs in the system except the communications task, the master scheduler, the system error task, the transient area loading task, and the caller. This ensures that the caller will gain immediate control and retain control until it issues a DEQ macro instruction for these resources, or is abnormally terminated. For step "must complete," all tasks in the job step except for the caller are set nondispatchable. ENQ then returns control to the caller via

EXIT and the Dispatcher by issuing an SVC 3.

- If the resources are not available (the RB wait count in the SVRB is not 0), ENQ prepares for the caller's future restart. An address in the ENQ routine is placed in the SVRB old PSW so that when the resources become available, ENQ will test for and, if specified, perform the "set must complete" function, and issue the SVC 3 which will return control to the caller. The caller remains in a wait state until the resources requested have been made available. ENQ then sets the TCB pointer (IEATCBP in the CVT) to 0 and branches to the Dispatcher for a task switch. The next highest priority task will receive control.

Conditional Request: If 'TEST', 'USE', or 'HAVE' was specified in the 'RET=' parameter in the ENQ macro instruction, request for the resource is conditional. Validity checking function is performed, and the task is abnormally terminated if an error condition is found. Whether the resources are enqueued to that task depends on the conditions below:

- If 'RET=TEST' was specified, ENQ tests the status of the resources and returns control to the caller through the EXIT routine and the Dispatcher via an SVC 3. No control blocks are created or enqueued. Register 15 contains a return code of 0 if the return codes of all requested resources are 0, or will contain the address of a list of return codes in main storage. (See Figure 33 for a description of return codes and their meanings.)

- If 'RET=USE' was specified, ENQ determines whether the resources are available and assigns control of the resources only if all are immediately available. If they are available, ENQ performs the same operations as for an unconditional request and places a zero in register 15. If they are not all available, processing is as 'RET=TEST' above.

- If 'RET=HAVE' was specified, ENQ determines if this is a duplicate request. If a previous request has not been made by this task for the same resources, ENQ performs as for an unconditional request and passes a return code or the address of a list of return codes in register 15. If a previous request has been made, ENQ loads a return code of 8 and returns control to the caller via an SVC 3.

DEQ (IEAGENQ1) CHART BJ

The DEQ macro instruction is normally used for an unconditional request for release of resources. The differences in DEQ processing when 'RET=HAVE' is specified are discussed separately under "Conditional Requests."

Unconditional Request: The first function of the DEQ routine is to check the validity of input parameters. If the issuing program is a supervisor routine (the protection key in the RB old PSW equals 0), the parameters are assumed to be correct. If it is a user program and an addressing error is found, or the rname length specified is zero, an SVC 13 (ABEND) is issued to abnormally terminate the task. (See Figure 34 for other DEQ error conditions.)

If this is a valid request for release of resources, the queues are searched to find the current QEL associated with the specified major and minor QCBs. To be a valid request, a major and minor QCB and a QEL must exist for the resource specified in the DEQ macro instruction, and the task must have been given control of the resource:

- If any of the control blocks (major QCB, minor QCB, and QEL) associated with the DEQ request are not found, the request has not been enqueued and the caller is abnormally terminated (an SVC 13 is issued) with an error code in register 1.

- If the QEL is not on the top of the queue (the TCB address in the current QEL is not the same as the current TCB address), and it represents an exclusive request or there are previous exclusive requests on the queue, the DEQ was issued by a task that did not have control of the resource. An error code is set in register 1, and an SVC 13 is issued.

If a valid DEQ has been issued (the QEL is at the head of the queue or it is a shared request with no previous exclusive requests), DEQ then decrements the enqueue count in the TCB (TCBCT), dequeues the QEL, and releases the main storage occupied by the QEL via an SVC 10 (FREEMAIN). DEQ then tests for outstanding requests for the same resource.

- If there are more QELs on the queue, DEQ determines if the next waiting requester, represented by the new top QEL, should be readied. The decision is based on the type of new top QEL, shared or exclusive, and on the type of dequeued QEL. The conditions and results for the three possible situa-

Condition	RET Operand of DEQ Macro Instruction Is:	Resultant Processing
Resource names are not found in the QCB queues, or a QEL containing the caller's TCB address is not found.	HAVE	(1) Sets up return code of 8, indicating that the resource is not enqueued, and after processing other parameter-list elements, returns control to the caller, via the Exit routine and the Dispatcher
	omitted or NONE	(2) Sets up an error code of 130 and obtains supervisor linkage to the ABEND routine to abnormally terminate the caller's task
QEL containing caller's TCB address is found, but is not at the logical top of the QEL queue, nor is it one of a "shared" group of QELs at the logical top of the queue.	HAVE	Same as (1) but return code is 4, indicating that the caller's task does not have access to the resource.
	omitted or NONE	Same as (2) except that the error code is 230.
RMC= operand is specified but user's protection key is not 0.		Same as (2) except that the error code is 330.

Figure 34. Error Conditions when Use of a Resource is Signaled Complete

tions are described in Figure 35. DEQ then determines if a readied requester should be dispatched. If DEQ has just reduced the wait count to 0, the resources are now available. If there are no higher priority tasks ready to be dispatched, DEQ indicates a task switch by placing the requesting task's TCB address in the NEW TCB pointer (IEATCBP) in the Dispatcher. If a higher priority task is to be dispatched, its address in NEW is left undisturbed. (It is also left undisturbed if both the dequeued QEL and new top QEL represent shared requests.)

- If there are no more QELs on the queue, there are no further requests for the resource. The minor QCB is therefore dequeued and the main storage space it occupies is released via an SVC 10 (FREEMAIN). In the same way, if there are no more minor QCBs on the queue, the major QCB is dequeued and its space released.

When all the parameter list elements representing resource requests to be dequeued have been processed, DEQ determines if it must perform the "reset must complete" functions. If "reset must complete" has been specified for system, the nondispatchability flags previously set in each TCB in the system (except the communications task, the transient area loading task, the system error task, and the master scheduler) are cleared so that these tasks may be restarted. If "reset must complete" is specified for a step, the nondispatchability flags previously set in other TCBs in the partition are cleared. DEQ also puts the TCB address of the highest priority ready task now dispatchable in the NEW TCB pointer. If "reset must complete" is requested by a user routine, the task is abnormally terminated with an error code of 330 in register 1.

DEQ returns control to the caller or a readied requester via the EXIT routine and the Dispatcher (an SVC 3 is issued). The EXIT routine dequeues the SVRB from its RB queue and frees the space that the SVRB occupies. The Dispatcher determines whether the caller or a readied requester is to get control, according to the TCB address in the NEW TCB pointer (IEATCBP).

Conditional Request: If the issuing task specified 'RET=HAVE', a conditional request for release of resources has been made. Normal DEQ functions are performed only if there is a major QCB, a minor QCB, and a QEL for the resource, and if the task has control of the resource. The task has control of a resource if the associated QEL is at the top of the queue or represents a shared request with no previous exclusive requests on the queue. If the conditions are not met, no dequeuing is performed. In either case, if no error conditions are found, a return code is set in register 15, and an SVC 3 is issued to return control to the caller or to a readied requester via

Conditions	Status of QEL Queue	Resultant Processing
Condition A dequeued QEL new top QEL	"shared" QEL --- "shared" QEL --- --- ---	Routine does not reduce wait count in requester's RB. (Requester already has access to the resource.)
Condition B dequeued QEL new top QEL	QEL of either type --- "exclusive" QEL --- --- ---	Routine reduces wait count in requester's RB and if new wait count is zero, it invokes the Task Switching routine to test whether the requester may be dispatched instead of the caller.
Condition C dequeued QEL new top QEL	"exclusive" QEL --- "shared" QEL --- --- ---	Routine reduces wait count in requester's RB and if new wait count is zero, it invokes the Task Switching routine. Since new top QEL is the first QEL of a "shared" group, the routine repeats this procedure for the other QELs of the group.

Figure 35. Determining if the Next Waiting Requester Should be Readied

the EXIT routine and the Dispatcher. (See Figure 34 for a further description of processing when 'RET=HAVE'.) If an error condition is found, an error code is placed in register 1 and an SVC 13 (ABEND) is issued to abnormally terminate the task.

SPIE (IEAAPX00) CHART BK

SPIE first determines if a PIE must be created for this task. If a PIE already exists, this task has previously issued a SPIE macro instruction which it wishes to override. Only one PIE is created for a given task, and it is addressed by TCB field TCBPIE. (There may be several program interruption control areas (PICA), one for each issuance of the SPIE macro instruction, but only the last specified PICA is active.)

If a PIE does not exist for this task (the pointer to the PIE in TCBPIE equals 0), the SPIE routine builds a PIE at the beginning of the extended save area of the SVRB, which was constructed when the SVC 14 was issued. The address of the PIE and the program mask (from the RB) are placed in the TCB field TCBPIE. The PIE is initialized to 0.

After the PIE has been located or created, SPIE saves the address of the previous PICA, which it obtains from the PIE. The field is 0 if this is the first issuance of the SPIE macro instruction for this task. The address of the new PICA, received as input in register 1, is then placed in the PIE. This will be 0 if no user program interruption handling routine is specified.

SPIE then determines the program mask to be placed in the caller's resume PSW by testing register 1 for 0. The program mask will be used by the program FLIH (first level interruption handler) to determine whether to give control to a user error

handling routine or to abnormally terminate the program.

- If register 1 contains a new PICA address, SPIE moves the program mask from the new PICA to the resume PSW field of the RB, where the SVC old PSW is stored. This ensures that the user exit routine will be entered if one of the specified types of program interruptions occurs.

- If register 1 contains 0, the user does not wish to handle program interruptions and is canceling a previously issued SPIE macro instruction. The initial program mask (before the first SPIE was issued) is restored to the resume PSW field of the RB from the TCB field TCBPIE. Thus when a program interruption occurs and the program mask is tested, the program FLIH recognizes that a user program interruption handling routine has not been requested and the task is abnormally terminated.

The SPIE routine then places the address of the previous PICA into register 1, and sets a return code of 0 in register 15. Interruptions are enabled, and an SVC 3 is issued to pass control to EXIT.

STAE (IEAAST00) CHART BL

The STAE service routine first obtains the address of the STAE user's RB from the RBLINK field of the STAE SVRB. If the STAE macro instruction has been issued in the user's STAE exit routine, an error code of 8 is placed in register 15, and control is returned to the user. The STAE request is not serviced, since the user's STAE exit routine processing is already attempting to deal with an error. STAE next tests the contents of register 0 to determine the kind of request -- to create, to cancel, or to overlay a STAE control block (SCB).

If register 0 contains a 0 (the create option), the address of the user's exit routine and parameter list specified in the STAE macro instruction are checked. If either address is invalid, an error code of 12 is placed in register 15, and control is returned to the user. If both addresses are valid, STAE issues a conditional GETMAIN macro instruction requesting 16 bytes of main storage for an SCB.

1. If main storage is not available, control is returned to the user with the return code 4 in register 15.

2. If main storage is available, the SCB is initialized by placing the address of the previous SCB, or 0, in the first word, the address of the user's STAE exit routine in the second word, the address of the user's exit routine parameter list in the third word, and the address of the user's RB in the fourth word. The address of the newly created SCB is placed in the last 3 bytes of the TCBNSTAE field of the TCB (displacement decimal 160).

   If XCTL=YES was coded in the STAE macro instruction (the high order bit of register 1 is on), STAE turns on the XCTL flag (bit 2) in the TCBNSTAE flag field to prevent cancellation of this SCB when an XCTL macro instruction is issued.

If register 0 does not contain 0, or if register 0 contains a 0 but the address of the user's STAE exit routine is 0, either the cancel or the overlay option is being specified. STAE tests the TCBNSTAE field to determine whether an SCB is already in effect. If it is 0, indicating that one is not, an error code of 8 is placed in register 15, and control is returned to the user, since an SCB that does not exist cannot be canceled or overlayed. If an SCB is in effect, STAE continues processing.

STAE next compares the fourth word of the SCB in effect with the address of the RB for the task that is requesting that the SCB be canceled or overlayed. If the addresses are not the same, a return code of 16 is placed in register 15, and control is returned to the user. This test prevents the destruction of an SCB belonging to another program.

The STAE routine now determines whether the STAE request is to cancel or to overlay an SCB.

- If register 0 contains a 4 (the cancel option), the addresses of the previous SCB, which is contained in the first word of the SCB in effect, is moved into the TCBNSTAE field. A FREEMAIN macro instruction is then issued to free the main storage occupied by the canceled SCB.

- If register 0 contains an 8 (the overlay option), the address of the user's exit routine and STAE parameter list are obtained and checked. If either address is invalid, an error code of 12 is placed in register 15, and control is returned to the user. If both addresses are valid, they are moved into the second and third words, respectively, of the SCB in effect. If the XCTL option is specified, the XCTL flag (bit 2) in the TCBNSTAE flag field is turned on.

When control is returned to the user from STAE, register 15 contains one of the following return codes:

Code	Indication
00	A STAE request is successfully created, overlayed, or canceled.
04	Storage for the STAE request is not available.
08	The user is attempting to cancel or overlay a nonexistent STAE request or is issuing a STAE in his exit routine.
0C	The address of the exit routine or the parameter list is invalid.
10	The user is attempting to cancel or overlay a STAE request not associated with his request block (RB).

In the following descriptions of the Normal and Abnormal Termination routines, condition/action tables are used where feasible to describe module processing. Where the condition in the left column is met, the action in the right column is taken. If the condition is not met, the processing for the condition/action in the following line occurs. If no condition is given, the action in the right column is always taken, unless the condition/action has been bypassed as a result of a previous action.

ABEND 0 -- NORMAL TERMINATION AND ABNORMAL ROUTING (IEANTM00) CHART CA

ABEND 0 is the first module entered to process normal and abnormal terminations. Processing of normal terminations is described below.

If entered because of an abnormal termination, ABEND 0 passes control to ABEND G. However, before exiting to ABEND G, ABEND 0 determines if there is sufficient main storage available for the SVRB created for the XCTL. If not, and if the task being terminated is a job step and the SVRB pool within the SVC SLIH is unavailable, ABEND 0 will steal storage from within the partition for the creation of the XCTL SVRB.

If the task to be terminated is a subtask, ABTERM is invoked to abnormally terminate the job step, and a WAIT macro instruction is issued.

Condition	Action
1. Entry from IEANTM0B	Go to step 9.
2.	Store completion code.
3. Interval timer exist	Set IQE pointer to TQE to zero.
4. TQE in use	Issue TTIMER CANCEL.
5.	Close DCBs, dequeue DEBs for task.
1. MCS in system	Pass control to IEACTM0B (ABEND 11).
7. WTOR requests to purge	Pass control to IEANTM0B (ABEND 11)
8. Subtask terminating	Pass control to IEANTM0C (ABEND 12)
9.	Dequeue IQEs for terminating task.
10. Limit priority does not equal dispatching priority	Issue CHAP macro instruction
11.	Initialize boundary box and FQE for hierarchy 0 (and hierarchy 1, if applicable), clear pointers to TIOT and problem program save area.
12. Small partition	Pass control to IEFSD599 (SMALLGO).
13.	Pass control to IEFSD515 (GO).

ABEND G -- ABNORMAL TERMINATION AND ABEND RECURSION (IEANTM0G) CHART CB

ABEND G provides initial processing for abnormal terminations and provides the interface to the OLTEP STAE Exit routine if OLTEP is the failing task.

Condition	Action
1. Return from IEANTM0M	Go to step 16.
2. Failure in SVC DUMP	Set all tasks dispatchable, turn on trace table, restore NEW in IEATCBP.
3. Not ABEND recursion	Go to step 9.
4. Not DAR recursion	Go to step 16.
5. Primary DAR recursion	Pass control to IEADTM22 (DAR 1).
6. Not valid CLOSE recursion after DAR	Pass control to IEADTM23 (DAR 2).
7. Job step ABEND	Pass control to IEANTM02 (ABEND 2).
8. Subtask ABEND (invalid)	Go to step 18.
9. ABTERM not entered	Store completion code.
10.	Issue HOOK macro instruction to disable GTF.
11. Purge fail in STAE	Store address of SVC 3 in caller's RB, pass control to IEAATA00 (EXIT).
12. Valid STAE issued; ABEND not caused by operator's cancel, timer expiration, invalid DETACH, or STAE recursion.	Free PIE, indicate ABTERM entered, indicate first ABEND entry from STAE, pass control to IEASTM11 (ASIR 1).
13. Step ABEND request	Go to step 18.
14. System task, must complete status, or Scheduler or Writer ABEND.	Pass control to IEANTM01 (ABEND 1).
15. Entries in message information list.	Pass control to IEANTM0M (ABEND M).
16.	Clear entries in list, free ABEND M WTO buffer.
17. Job step ABEND	Pass control to IEANTM01 (ABEND 1).
18.	Use ABTERM to abnormally terminate job step, issue WAIT macro instruction (never posted).

ABEND 1 -- ABEND/STAE GRAPHICS LINKAGE (IEANTM01) CHART CC

ABEND 1 processes nonrecursive terminations for all tasks or for failures when GJP is included in the system. The following table gives the possible conditions within the module and the resulting action taken.

Condition	Action
1. Graphics recursion	Pass control to graphics hook routine.
2. GTF is terminating	Permanently disable GTF.
3.	Purge task of IQEs, active TQEs, SPIE requests, transient area requests, and STAE requests.
4. Terminating task has subtasks	Perform step 3 for each subtask of terminating task.
5.	Pass control to IEACTM0B (MCS) or IEANTM0B (non-MCS) ABEND 11.

ABEND 2 -- PURGE I/O (IEANTM02) CHART CD

Condition	Action Taken
Nonrecursive ABEND	Purge I/O for terminating task and any subtasks
Recursive ABEND	Purge I/O for terminating task only
System task terminating, terminating task is in "must complete" status, or Writer or Scheduler failure	Exit to DAR 1 (IEADTM22)
Nonrecursive ABEND	Exit to ABEND 3
Recursive ABEND	Exit to ABEND 9

ABEND 3 -- CONTROL BLOCK VALIDITY CHECK (IEANTM03) CHART CD

ABEND 3 tests the validity of the control blocks associated with a nonrecursive terminating task. This ensures system integrity by preventing an abnormal termination within ABEND. Except for certain FQE conditions, invalid control blocks cause the termination of the queue on which they reside after the last valid control block. The table below gives the validity checks for each control block. Where an FQE condition is indicated by a bullet (•), the FQE is dequeued, but the queue is not terminated.

This process is duplicated for all subtasks of the terminating job step. GQE and JPAQ control blocks may exist only if subtasking is included in the system.

Validity Requirements	FQE	Load List	DEB	Active RB
Alignment on a fullword boundary			X	
Alignment on a doubleword boundary	X	X		X
Block not in free core		X	X	X
Maximum of 300 blocks on queue		X	X	X
Valid forward and backward pointers		X		
Block size is a multiple of 8	•			
Block does not extend beyond partition boundary	•			
Block does not describe an area described by an FQE	•			
Block resides in the partition	X	X	X	X
Blocks are queued in descending order	•			

ABEND 3 exits to ABEND J for MFT with subtasking or to ABEND 4 without subtasking.

ABEND 4 -- DUMP TEST (IEANTM04) CHART CE

ABEND 4 determines if a dump is required and ensures that sufficient main storage is available for any type of dump, for an ENQ purge, or for a CLOSE. A dump is not given if there is insufficient main storage available for the dump. If a SYSABEND or SYSUDUMP DD name is not found in the TIOT, an indicative dump is given. The following table gives the conditions possible within the module and the exits taken.

Condition	Exit Taken
*Full dump requested	IEANTM05
Indicative dump requested	IEANTM08
Insufficient main storage available for providing dump	IEANTM0A
*ABEND is for subtask	IEANTM0C
*Subtasks need ENQ purge	IEANTM0D
*Subtasks do not need ENQ purge	IEANTM0E
Job step termination	IEANTM07
*MFT with subtasking only	

ABEND 5 -- OPEN DUMP DATA SET (IEAMTM05; WITHOUT SUBTASKING)

If a SYSABEND or SYSUDUMP DD name is not found in the TIOT, or if the data set cannot be opened, ABEND 5 exits to ABEND 9. ABEND 5 opens the dump data set for the terminating task and exits to ABEND 6.

ABEND 5 -- OPEN DUMP DATA SET (IEANTM05; WITH SUBTASKING) CHART CF

ABEND 5 opens the dump data set if necessary for the terminating task. This data set remains open until termination of the job-step task, and need not be reopened for subsequent ABEND processing. Any subtasks of the terminating task that are enqueued on the dump data set are dequeued by a branch entry to DEQ since these subtasks are nondispatchable. An ENQ macro instruction is issued on the dump data set to prevent other tasks from using it until the dump is finished. If the data set cannot be opened, or if a SYSABEND or SYSUDUMP entry is not found in the TIOT, ABEND 5 exits to ABEND 9. If any task in the job step is in the process of opening the dump data set, the current task is set nondispatchable, its resume PSW is set to reenter this module, and control is passed to the Dispatcher. If the data set is opened, all tasks in the job step that were set nondispatchable while waiting for the dump data set to open are reset dispatchable, and control is passed to ABEND 6.

ABEND 6 -- DUMP (IEANTM06) CHART CG

ABEND 6 prints appropriate messages pertinent to the dump and issues a SNAP macro instruction (SVC 51) to call ABDUMP for the terminating task.

If GTF is active and formatting has been requested, ABEND 6 sets up the SNAP parameter list to indicate that ABDUMP is to format the GTF trace data. If formatting has been requested, GTF tracing will be suspended during the dump.

In MFT without subtasking, if GTF is inactive and the trace table was included during system generation, the SNAP parameters are set to dump the trace table.

For MFT with subtasking ABDUMP is invoked for all subtasks.

The following table gives the conditions within the module and the exits taken.

Condition	Exit Taken
*Job step task has no subtasks	IEANTM07
SNAP macro instruction failed	IEANTM09
*Subtask is terminating	IEANTM0C
*Subtask needs ENQ purge	IEANTM0D
*Subtasks do not need ENQ purge	IEANTM0E
Job step termination	IEANTM07
*MFT with subtasking only	

ABEND 7 -- TERMINATION (IEANTM07) CHART CH

IEANTM07 first determines if GTF tracing is suspended. If it is, the HOOK routine is used to restart the tracing. IEANTM07 tests the OLTEP bit in the TCBRTRN field of the TCB to determine if system restoration is required for the OLTEP task. If so, ABEND 7 issues an SVC 59 for performance of the cleanup functions necessary for system restoration.

Then, ABEND 7 closes all data sets chained to the TCB of the terminating task, dequeues IQEs, and clears the pointers to the ABEND appendages and transient area queues. If an indicative dump has been created, it is moved to the upper part of the partition (end-632). If TCAM is abnormally terminating, ABEND 7 passes control to the TCAM termination module IEAQTM0Q which will return control to ABEND 7 when it completes processing. IEANTM07 exits through the dummy program to GO (IEFSD515) if its partition size is equal to or larger than the scheduler, or, if smaller, to SMALLGO (IEFSD599).

If subtasking is included in the system, this module is entered only if a job step task is terminating and its subtask's TCB has been freed.

ABEND 8 -- INDICATIVE DUMP (IEANTM08) CHART CH

ABEND 8 creates an indicative dump area (for job step termination only) containing the following information:

- Register contents at entry to ABEND
- Floating-point register contents, if any
- Completion code
- Program-check instruction if a program interruption occurred.
- TCB flags
- Program name
- Entry point
- Resume PSW
- Active RBs
- Loaded RBs

ABEND 8 exits to ABEND 7 for job step termination.

If subtasking is included in the system, ABEND 8 exits to ABEND 13 to purge resources allocated to subtasks by ENQ, or to ABEND 15 if subtasks do not have allocated resources. If no subtasks exist, ABEND 8 exits to ABEND 7.

ABEND 9 -- RECURSION PROCESSING (IEANTM09) CHART CI

ABEND 9 dequeues all request blocks created as a result of the recursion and processes all OPEN, CLOSE, ABDUMP, and message recursions. The following table gives the conditions possible within the module and the actions taken.

Condition	Action
Message recursion	Pass control to IEANTM04 (ABEND 4).
*OPEN recursion	Reestablish JPAQ chain, reset dispatchable any tasks waiting for open of dump data set.
*Subtask ABEND	Reestablish GQE chain.
OPEN recursion	Set flag for indicative dump, reset dispatchable any tasks waiting for open of dump data set.
CLOSE recursion	Indicate no dump.
*ABDUMP recursion	Free the ABDUMP work area, set flag for indicative dump, issue DEQ macro instruction for dump data set.
ABDUMP recursion	Free the ABDUMP work area, set flag for indicative dump.
	Pass control to IEANTM0H (ABEND H).
*MFT with subtasting only	

ABEND 10 -- STEAL MAIN STORAGE (IEANTM0A) CHART CJ

ABEND 10 steals main storage from the problem program partition whenever ABEND 4 determines that sufficient free main storage is not available for ABEND to perform its essential functions.

Condition	Action
Job step ABEND	Free storage occupied by programs on JPAQ.
	Free main storage occupied by programs on Load List.
Terminating task has subtasks	Free main storage occupied by programs on Load Lisss of subtasks.
ABEND SVRB is in PRB area	Move SVRB to top of partition.
FQE is in PRB area	Dequeue FQE
	Free main storage occupied by programs on active RB queue.
Terminating task has subtakks	Free main storage occupied by programs on subtasks' active RB queue.
Dump requested, but not enough storage could be freed	Turn off flag indicating dump to be given, indicate message needed.
Task owns SVRB pool	Obtain main storage and copy SVRB from pool into obtained storage, and indicate copied SVRBs free.
All used SVRBs in pool copied	Indicate task no longer owns SVRB pool.
	Pass control to IEANTM04 (ABEND 4).

ABEND 11 -- WTOR PURGE (IEANTM0B; WITHOUT MULTIPLE CONSOLE SUPPORT) CHART CK

ABEND 11 ensures an end line in all multiple-line WTO messages and purges outstanding WTOR requests for the terminating task and exits to ABEND 0 for normal job step termination, or to ABEND 2 for abnormal termination.

Exits are to ABEND 12 for normal subtask termination, or to ASIR 3 if STAE was the requester.

ABEND 11 -- WTOR PURGE (IEACTM0B; WITH MULTIPLE CONSOLE SUPPORT)

ABEND 11 (IEACTM0B) has the same functions and exits as ABEND 11 (IEANTM0B) (above).

ABEND 12 -- LOADING PROGRAM PURGE (IEANTM0C; WITH SUBTASKING ONLY) CHART CK

ABEND 12 is entered only when the terminating task is a subtask. Partially loaded programs and FRBs for the terminating task and its subtasks are freed, all SNAP nondispatchability flags are cleared, and control is passed to ABEND 13.

ABEND 13 -- SUBTASK ENQ PURGE (IEANTM0D; WITH SUBTASKING ONLY) CHART CK

ABEND 13 releases allocated resources for all subtasks of the terminating task and for the terminating task itself if it is a subtask. Outstanding ENQs for the job step task will be purged by the job management routines IEFSD597 or IEFSD598 at step termination. ABEND 13 exits to ABEND 15.

ABEND 14 -- DATA SET CLOSE (IEANTM0E; WITH SUBTASKING ONLY) CHART CL

ABEND 14 is entered from ABEND 15 to complete subtask termination. For abnormal termination, all open data sets for the terminating task are closed. If there is not sufficient main storage available for close processing, ABTERM is entered to schedule abnormal termination of the job step. If TCAM is abnormally terminating, ABEND 14 passes control to the TCAM termination module IEAQTM0Q which returns control to ABEND 14 upon completion of its processing. For both normal and abnormal termination, ABEND 14 removes from the RB queue all RBs except the SVRB for ABEND. All GQEs are freed, the task is set nondispatchable, and is indicated as terminated. ABEND 14 then passes control to the SVC EXIT routine IEAATA00.

ABEND 15 -- IQE PURGE AND SUBTASK CLEANUP (IEANTM0F; WITH SUBTASKING ONLY) CHART CM

ABEND 15 purges IQEs for the terminating task and any subtasks. If the top task is a subtask terminating normally, minor LPRBs are freed, any entries in the TSCE for the task are removed, and ABEND 15 passes control to ABEND 14. For abnormal termination, all DEBs and GQEs of any daughter tasks are chained to the TCB of the terminating task, and minor LPRBs are dequeued and freed. Any entries in the TSCE for the daughter tasks are removed.

For GTF, a check is made for subtasks that are abnormally terminating or issuing SNAP dumps to ensure that the GTF tracing is resumed for the terminating task. Then, main storage associated with any SCBs is freed. Finally, the subtask TCBs are removed from the TCB queue and their main

storage is freed. If the job step is abnormally terminating, ABEND 15 passes control to ABEND 7. Otherwise, ABEND 15 passes control to ABEND 14.

ABEND M -- WTP PROCESSING FOR TYPE 1 SVC ROUTINES (IEANTM0M) CHART CN

ABEND M receives control from ABEND G if there is a WTP message to be issued for a Type 1 SVC routine. The Message Information List entries are scanned for a matching TCB. ABEND M then builds a parameter list in the extended save area of the current RB and issues a conditional GETMAIN macro instruction for a message buffer. If the GETMAIN is successful, ABEND M issues the WTP message and returns to ABEND G. If the GETMAIN is not successful, ABEND M sets the length in the GETMAIN parameter list to zero, and returns to ABEND G.

ABEND H -- WTO ISSUING FOR VALID RECURSIONS (IEANTM0H) CHART CO

ABEND H receives control from ABEND 9 after an OPEN, CLOSE, or ABDUMP recursion to issue a WTO message identifying the recursion. ABEND H tests for availability of storage for WTO processing before issuing the message and, if sufficient storage is not available, the message is not issued. ABEND H returns control to ABEND 4.

ABEND J -- JPAQ AND GQE VALIDITY CHECK (IEANTM0J) CHART CP

ABEND J receives control from ABEND 3 to validity check RBs on the job pack area queue and gotten subtask area queue elements (GQEs). Both must be within the partition, aligned on a doubleword boundary, and within allocated storage. The RBs on the JPAQ must also have valid forward and backward pointers. GQEs must not extend beyond the partition boundary, must not describe an area described by an FQE, and must have a size that is a multiple of 8. ABEND J returns control to ABEND 4.

ABTERM -- (IEAGAB00; WITHOUT THE TRACE TABLE) CHART CY

The ABTERM routine is used to force a specified routine to terminate.

Condition	Action
1. Caller is the XSNT Area Task*	1. Exit to the Reinstate XSNT Routine
2. TCB address not on a fullword boundary and Type 1 SVC switch on	2. Set SVC Old PSW to an SVC 13 (ABEND) and exit to the FLIH at entry point DISMISS
3. TCB address not on a fullword boundary and Type 1 switch not on	3. Exit to the Dispatcher
4. ABEND is the caller (ABEND bit on in the TCB)	4. Go to action 8
5. STAE process progressing	5. Go to action 7
6. ABTERM has had control previously	6. Exit to the caller because the PSW has already been set to an SVC 13
7.	7. Store the completion code
8.	8. Turn on the ABTERM bit in the TCB, prevent asynchronous exits
9. ABEND is the caller	9. Store the completion code in the ABEND appendage
10. The current task is not terminating and the terminating task has a higher priority than the current task	10. Set the terminating task for immediate dispatching
11.	11. Turn off the pseudo-disable switch
12. Type 1 SVC switch is on	12. Set the SVC Old PSW to an SVC 13, turn off the SVC nesting switch, and exit to the caller
13. Type 1 SVC switch is not on	13. Set the RB wait count to 0, set the resume PSW to an SVC 13, and exit to the caller
*The XSNT Area Task coordinates all loading of the transient area. It is external to ABTERM.	

ABTERM -- (IEAIAB00; WITH THE TRACE TABLE)

This module is exactly the same as IEAGAB00 except that it contains the address of the Trace Stop routine so that it can be referred to by ABEND and SVC DUMP1.

ABTERM -- (IEANAM00; WITH SUBTASKING) CHART CY

This module is exactly the same as IEAGAB00 except that before "Resulting Action 13" (see table above), the terminating task would be set dispatchable.

ASIR 1 -- ABEND/STAE INTERFACE ROUTINE 1 (IEASTM11) CHART CQ

ASIR 1 receives control when ABEND determines that the terminating program has issued a STAE macro instruction. It schedules the user's exit routine and routes control for a retry if one was requested. If a retry is scheduled, ASIR 1 purges the Message Information List of all entries for the current TCB.

Condition	Action
1.	1. Set STAE recursion flag in TCB to prevent second entry into STAE
2. Current task in must complete status and user not in supervisor state	2. Issue an ABEND
3. SIRB on the RB queue	3. Issue an ABEND to prevent interlock
4. Current SCB invalid (RB address in SCB does not point to an RB on the RB queue but more SCBs exist	4. Dequeue and free storage for the invalid SCB thereby making the previous SCB current, repeat Condition 4
5. Current SCB invalid with no more SCBs	5. Issue an ABEND macro instruction
6. Current SCB valid	6. Purge I/O with quiesce option
7. Purge not successful	7. Purge I/O with halt option
8.	8. Conditional GETMAIN for work area and register save area
9. Sufficient storage available	9. Initialize work area and set parameter registers
10. Sufficient storage not available	10. Set parameter registers
11.	11. SYNCH - schedule user exit routine
12.	12. Free register save area if one was acquired
13. Retry not requested	13. Issue an ABEND
14. Retry requested without purge of RB queue and user in supervisor mode	14. Pass control to ASIR 3
15. Retry requested without purge of RV queue and user not in supervisor state	15. Issue an ABEND macro instruction
16. Retry requested with purge of RB queue and user in supervisor state	16. XCTL to ASIR 2
17. Retry requested with purge of RB queue and user not in supervisor state	17. Verify retry routine and parameter list addresses
18. Addresses not valid	18. Issues an ABEND
19.	19. Pass control to ASIR 2

ASIR 2 -- ABEND/STAE INTERFACE ROUTINE 2 (IEASTM12) CHART CR

ASIR 2 receives control when ASIR 1 determines that the RB queue must be purged before the retry routine is scheduled. Since all RBs between the abnormally terminating RB for the task and the RB for the program that issued the STAE will be purged, all their restorable IOBs must be dequeued and all their open data sets must be closed.

ASIR 2 dequeues IOBs and closes DCBs, unless the DCB organization is in the ISAM or TAM groups, for all RBs (SVRBs excluded) that will be purged. If any ISAM or TAM DCBs were encountered, ASIR 2 passes control to ASIR 4 to have them closed. Otherwise, control goes to ABEND 11 to purge the WTOR reply queue.

ASIR 3 -- ABEND/STAE INTERFACE ROUTINE 3 (IEASTM13) CHART CS

ASIR 3 receives control whenever a retry has been requested. If the user specified a purge of the RB queue, the resume PSW address in all RBs between the RB for the terminating task and the RB for the program that issued the STAE are set to an SVC 3 (EXIT) instruction. The SCB is then freed, the work area (if one was acquired by ASIR 1) is reinitialized, and the parameter registers are set. The resume PSW of the STAE user is set to point to the Retry routine, and control is passed to the dispatcher.

If no purge was requested, a new RB is created, initialized for the Retry routine, and queued on the RB chain. The SCB is freed, the work area is reinitialized, the parameter registers are set, and control is passed to the Dispatcher.

ASIR 4 -- ABEND/STAE INTERFACE ROUTINE 4 (IEASTM14) CHART CT

ASIR 4 receives control from ASIR 2 to close DCBs organized in the ISAM and TAM group and dequeue the IOBs related to them, for the RBs that are going to be purged. Upon completion, ASIR 4 passes control to ABEND 11 to purge the WTOR reply queue.

DAR 1 -- DAMAGE ASSESSMENT ROUTINE 1 (IEADTM22) CHART CU

DAR 1 is called to dump main storage when ABEND detects a failing task in must complete status, a failing system task, or an invalid recursion.

If any of the following conditions are found, DAR will not attempt to write the dump:

1. Main storage was stolen by ABEND or DAR.

2. Storage is not available in the partition to schedule SVC DUMP.

3. This entry is a primary DAR recursion (a previous failure to write the dump).

4. The SYS1.DUMP data set has not been allocated.

If any of these conditions exists, DAR 1 purges the message information list and writes an appropriate message to the console. It then traces the RB queue to determine whether the SIRB is enqueued. If it is, it is dequeued. If the failing task is a system task and is not in "must complete" status, control is passed to DAR 2 (IEADTM23). For all other failing tasks, control is passed to DAR 3 (IEADTM24).

If the failing task is in "must complete" status and there is not enough main storage for WTO processing, DAR 1 "steals" main storage for the messages.

Otherwise, DAR 1 calls the SVC DUMP (SVC 51) routine to write the dump of main storage to the SYS1.DUMP data set. When control is returned, DAR 1 exits as before.

For MFT with subtasking, all tasks related to the failing task are marked for termination. Also, there is no special handling of the SIRB.

DAR 2 -- DAMAGE ASSESSMENT ROUTINE 2 (IEADTM23) CHART CV

If DAR was invoked by a failing system task that is not in "must complete" status, DAR 2 attempts to reinstate the task to avoid a Wait state in the CPU. DAR 2 writes a task reinstatement message to the operator (unless main storage has been stolen), sets the top RB to the reinitialize routine, sets the wait count to 0, and passes control to the Dispatcher.

For MFT with subtasking, a check is made to determine if the failing task is a system error task operating on behalf of another task. If it is, ABTERM is used to schedule the abnormal termination of the requesting task and then DAR 2 passes control to the Dispatcher.

DAR 3 -- DAMAGE ASSESSMENT ROUTINE 3 (IEADTM24) CHART CW

If DAR 3 is entered because of a secondary DAR recursion for a subtask, ABTERM is entered to schedule abnormal termination of the job step. If a job step is terminating, a HOOK macro instruction is issued to resume GTF tracing, a task reinstatement failed message is written, and control is passed to DAR 4.

If this is not a secondary DAR recursion, and resources are not critical,a task reinstatement message is written, appropriate nondispatchable flags are set, and control is passed to ABEND 3.

If resources are critical, a task reinstatement failed message is written and control is passed to DAR 4.

If the task is in "must complete" status, processing is the same as for resources not critical (above).

DAR 4 -- DAMAGE ASSESSMENT ROUTINE 4 (IEADTM24) CHART CX

DAR 4 enters IEAASPRG for subsystem cleanup and sets the failing task permanently nondispatchable. If GTF tracing is to be resumed, a HOOK macro instruction is issued to effect this. DAR 4 sets NEW (IEATCBP) to the address of the highest priority ready TCB and branches to the Dispatcher.

SVC DUMP (IEAAAD0Y, IEAAAD0Z) CHARTS DA AND DB

SVC DUMP provides a dump of selected areas of main storage to either tape or a direct access device.

SVC DUMP 1 (IEAAAD0Y) is entered when the caller issues an SVC 51. Initially, SVC DUMP 1 determines if this is a SNAP or SDUMP request. The ninth byte (offset 8) in the parameter list is set to X'80' if this is an SDUMP request. For a SNAP request, SVC DUMP 1 passes control to ABDUMP 1.

SVC DUMP 1 checks the highest level RB of the invoking task to ensure it is operating with a protection key of 0, that is, to ensure that it is in supervisor state. If it is not, the dump exits with a return code.

The lock byte is then tested to determine whether a dump is already in progress. If it is, the dump exits with a return code. If it is not, the lock byte is set in order to prevent simultaneous access to dump data sets. Next, all tasks except the

Communications Task, Transient Area Loading Task, System Error Task, and current task are set nondispatchable and the dump-invoked bit in the TCB is turned on.

If the DCB operand is 0, the SYS1.DUMP data set is assigned by default. If the SYS1.DUMP data set is not available or if the DCB for the data set is not open, SVC DUMP 1 performs exiting procedures (below).

SVC DUMP 1 then issues the TIME DEC macro instruction to obtain the date and time. These are stored in the header record.

The UCB representing the device upon which the dump data set resides is checked to ensure that it is in the "ready" state. If it is not, a return code is set and the routine exits after resetting dispatchability bits and the lock byte. The device type, either tape or direct access, is then determined and initialization of the control blocks is completed as required.

The trace table, if present, or GTF, if active, is made inoperative for the duration of the dump so that entries leading up to the failure are preserved. Prior to exiting, the trace facilities will be reactivated.

If the device is direct access, a test is required to determine if the data set is available to receive a core image dump. A channel program is initialized to read the first record on the data set. If end-of-file is not detected, the data set already contains a core image dump, in which case a return code is set and the routine exits. Otherwise, the data set is empty and available to receive a core image dump. The user supplied header record is the first data block written. After this the channel programs are reinitialized to perform the actual writing of the core image dump.

Abnormal end, channel end, and program controlled interrupt (PCI) appendages are provided. The writing of the dump is performed via EXCP and WAIT. The PCI appendage updates the channel programs required to write the core image. The channel end and abnormal end appendages restart the channel program if end-of-cylinder or if I/O errors are encountered. Standard ERPs are used while writing the dump except if SVC DUMP was invoked due to a failure of the System Error Task or if, in MFT without subtasking, the SIRB is active.

An unrecoverable error condition causes the writing of the dump to be terminated and a return code set prior to exiting. Upon completion of the dump, a normal completion return code is set. For SYS1.DUMP an end-of-file record is written after the dump is completed (normally or abnormally). For user DASD data sets, the DCBFDAD field in the user's DCB is filled in with the address of the last block written. The DCBTRBAL field is set to 0 if the last record written filled a track. Otherwise, it is set to 1024. These steps are required to present a standard DCB interface to CLOSE. All tasks are set dispatchable prior to exiting, the lock byte is reset, and the "dump invoked" bit is turned off. Control is then returned to the caller.

If the device is tape, control is passed to SVC DUMP 2 (IEAAAD0Z). When writing to tape, processing is basically the same as direct access with the exception that the PCI and channel end appendages are not used. A tapemark is written only for a NIP-allocated tape unit. If the tape is user-supplied, no tapemark is written. The user must close the DCB. The "LEAVE" option may be specified thus allowing multiple dumps on tape.

SVC DUMP 2 checks for unit exception after writes to tape. If this occurs, the dump is terminated for lack of space. In this case, a tapemark is written and the tape unloaded. A special return code is provided. If a channel data check occurs, the dump continues at the next storage address to be displayed.

If dump storage boundaries are indicated in the parameter list, the dump routine dumps main storage from the addresses specified. Beginning addresses are rounded down to a 2K boundary; ending addresses are rounded up to a 2K boundary. If the parameter list indicates that nucleus and SQA are to be dumped, this is done first. All parameters are validity checked. If invalid, the aread defined by the invalid parameters is not dumped. Instead, the parameter pair is marked as invalid, and the next set of parameters is examined.

Upon completion of the last WRITE, SVC DUMP 2 returns to the caller after setting all tasks dispatchable, turning off the lock byte and dump invoked bit, and turning on the trace facilities.

ABDUMP 1 (IEAMAD00) CHART DC

ABDUMP 1 ensures that all ABDUMP requirements have been met. If the DCB has not been opened or does not have the proper attributes (DSORG=PS, MACRF=W, RECMF=VBA, LRECL=125, MFT 2 BLKSIZE=882) a return code is placed in register 15, and control is returned to the caller. The TCB address is also verified and any error results in an error code and a return to the caller. Otherwise, ABDUMP 1 acquires and initial-

izes a 672-byte work area and loads into it the output routine that is used by all loads of ABDUMP. If enough main storage is available, a buffer is acquired to facilitiate the operation of the Output Routine. A test is made to determine if SNAP is the caller and if trace data is requested in the dump. If both cases are true, and if GTF is inactive and the Trace Table is present, tracing in the Trace Table is temporarily suspended to preserve the contents of the Trace Table. If GTF is active, and if the GTF trace records are maintained in main storage, the GTF trace is temporarily suspended to preserve the GTF trace data while the data is being dumped. Control is then passed to ABDUMP 2.

To avoid any modification of the supervisor queues, all subtasks of the job step (except for the current subtask) are set nondispatchable before control is passed.

ABDUMP 2 (IEAAAD0D) CHART DD

ABDUMP 2 formats and writes the following information common to all types of dumps:

- The heading line of the dump
- The completion code of the terminating task
- A message indicating the type of interruption for 0C1-0C7 interrupts
- The actual address of the program interruption for the types mentioned above
- The resume address at the time of the interruption
- The PSW at entry to ABEND or SNAP

Control is passed to ABDUMP 3 if supervisor data is required. Control is passed to ABDUMP 5 if supervisor data is not wanted but the save areas are. Control is passed to ABDUMP 6 if neither supervisor data nor the save areas are requested.

ABDUMP 3 (IEAAAD0A) CHART DD

If ABDUMP 2 detects that supervisor data is wanted by the caller, ABDUMP 3 receives control to format and print the following portion of supervisor data:

- The TCB
- The registers from the TCB save area when the current task or subtask is not being dumped
- The RBs on the active RB queue
- The RBs on the load list

Control is then passed to ABDUMP 4.

ABDUMP 4 (IEAAAD01) CHART DE

ABDUMP 4 writes the remainder of the supervisor data as follows:

- The RBs on the job pack area queue
- The problem program boundaries
- The free queue elements
- The GQEs for all subtasks

Control is then passed to ABDUMP 5 if a save area trace is requested, or to ABDUMP 6 if it is not.

ABDUMP 5 (IEAAAD02) CHART DF

ABDUMP 5 is entered if the user requested a save area trace. Save areas are initially displayed in a forward order, that is, the order in which the associated routines were invoked, starting with the supervisor-provided save area. The forward trace continues until all program-provided save areas have been displayed or until incorrect forward or backward chaining of save areas has been encountered. Then a partial backward trace is performed, providing the save area associated with the lowest PRB on the chain (the PRB of the program being terminated or the program that issued the SNAP) and the save area indicated by its backpointer. Control is then passed to ABDUMP 6.

ABDUMP 6 (IEAAAD03) CHART DF

ABDUMP 6 prints, at the user's request, the data set information (the DD name, the UCB, and the DEB and DCB addresses for open data sets) if ABEND did not "steal" any storage. The reason for this is that ABEND's "steal" might have destroyed some DEBs or DCBs and thus invalidated the partial list that ABDUMP could provide. Control is then passed to the TCAM ABDUMP modules if TCAM is in the system and the TCAM control program is being dumped. If it is not, and if tracing has not been requested, control is passed to ABDUMP 8. If the optional Trace Table is in the system and GTF is inactive, control is passed to ABDUMP 7 to format the Trace Table. However, if GTF is active, the optional Trace Table output is bypassed and control is passed to ABDUMP 13 to format the GTF trace data if formatting has been

requested. Otherwise, control is passed to ABDUMP 8.

ABDUMP 7 (IEAAAD0B) CHART DG

ABDUMP 7 formats and prints the trace table (oldest entry first) when the user has requested it. ABDUMP 7 then passes control to ABDUMP 8.

ABDUMP 8 (IEAAAD04) CHART DG

ABDUMP 8 assembles the EBCDIC translate routine and moves it into the work area created by ABDUMP 1. If requested, it formats and writes the register contents that were saved at entry to either ABEND or SNAP. Control is then passed to ABDUMP 9.

ABDUMP 9 (IEAAAD05) CHART DH

If the caller requested it, ABDUMP 9 dumps the nucleus with its EBCDIC translation. Control is then passed to ABDUMP 10 if a dump of problem-program storage was requested or to ABDUMP 11 if it was not.

ABDUMP 10 (IEAAAD0E) CHART DH

ABDUMP 10 dumps all requested problem-program storage with its associated EBCDIC translation. Any freed areas within the problem program partition are skipped. Control is then passed to ABDUMP 11.

ABDUMP 11 (IEAAAD0C) CHART DH

ABDUMP 11 formats and prints a list of specific storage locations which may or may not duplicate some other parts of the dumped material and can only be requested for a dynamic dump (a user issued SNAP macro instruction). These areas are formatted and dumped in the order requested by the user. Control is then passed to ABDUMP 12.

ABDUMP 12 (IEAAAD0F) CHART DI

ABDUMP 12 formats and prints the END-OF-DUMP or END-OF-SNAP message. If a buffer was acquired by ABDUMP 1 it is freed along with ABDUMP's work area. Control is then returned to the caller (either the user or ABEND).

For MFT with subtasking, all subtasks of the job step are set dispatchable before exiting.

ABDUMP 13 (IEAAAD0K) CHART DJ

ABDUMP 13 formats and prints the GTF trace data when the user has requested that the data be maintained in main storage. ABDUMP 13 gains control from ABDUMP 6 or TCAM ABDUMP 4 in GTF is active and if GTF is to format the trace data.

If the TIME option was chosen when GTF was started, an additional record is generated containing the time in the form 'seconds.microseconds.' This record follows the formatted trace record. When control records or error records are encountered, control is passed to ABDUMP 14 for processing.

After all records have been formatted, GTF tracing is resumed and control is passed to ABDUMP 8.

ABDUMP 14 (IEAAAD0M) CHART DK

ABDUMP 14 formats GTF control records (time records and lost-event-count record) and prints out the control records, error records and error messages. When the record and/or error message has been formatted and written, control is returned to ABDUMP 13.

TCAM ABDUMP (IEAAAD0G, IEAAAD0H, IEAAAD0I, IEAAAD0J) CHARTS DL AND DM

The TCAM ABDUMP modules receive control from ABDUMP 6 when the program being dumped is the Telecommunications Access Method Message Control Program.

TCAM ABDUMP 1 (IEAAAD0G) formats and prints the header line, the Address Vector Table (AVT), and the basic section of the Terminal Name Table (TNT). The AVT is dumped in three parts -- the basic section, the core section, and the disc section. Then the TNT section is dumped and control is passed to the second TCAM ABDUMP load. (Control between all TCAM ABDUMP modules is passed via the XCTL macro instruction.)

TCAM ABDUMP 2 (IEAAAD0H) dumps the entries in the TNT and their associated Terminal Table (TRM) entries. First, the TNT entry is formatted, and immediately after, its associated TRM entry is also formatted. This is repeated for every TNT entry. Control is then passed to the third TCAM ABDUMP load.

TCAM ABDUMP 3 (IEAAAD0I) dumps the TCAM Destination Queue Control Blocks associated with the TRM entries. They are located by searching the TNT entries, and are formatted in the order of ascending storage locations. After all the queue control

blocks have been dumped, control is passed to the fourth TCAM ABDUMP load.

TCAM ABDUMP 4 (IEAAAD0J) dumps the three types of TCAM DCBs (line group, message queue, and checkpoint) and the Line Control Blocks associated with each line group DCB. Only open TCAM DCBs are dumped and the order in which they are formatted depends on the order in which their corresponding DEBs are chained. The Line Control Blocks for each line group DCB are formatted immediately after their associated DCBs.

When all DCBs have been printed, control is transferred to ABDUMP if the user has not requested trace data. If the user requested tracing, and if the optional Trace Table is in the system and GTF is inactive, control is passed to ABDUMP 7 to format and print out the Trace Table. However, if GTF is active, the optional Trace Table output is bypassed, and control is passed to ABDUMP 13 to format and print out the GTF trace data if formatting has been requested.

ATTACH (IEAAAT00; WITHOUT SUBTASKING) CHART EA

Attach searches for the RB of the requested routine in the loaded program list and the resident reenterable module area (optional). If the RB is not found, Attach uses the FINCH subroutine to bring the requested routine into the dynamic area. If the requested routine is "only loadable," ATTACH issues an ABEND macro instruction to abnormally terminate the task. Otherwise, ATTACH alters its resume PSW, so that after branching to the Dispatcher control will return to that portion of the ATTACH routine which passes control to the attached routine (entry point IEAOATO1). ATTACH then branches to the Dispatcher to dispatch any asynchronous routines that may have been scheduled while FINCH was executing.

If the RB of the requested routine is found when searching the loaded program list, ATTACH routine determines if the requested routine is "only loadable." If so, ATTACH issues an ABEND macro instruction to abnormally terminate the task. If it is not "only loadable", ATTACH determines if the RB is active. If the RB is not active, ATTACH can use the routine that is already in the dynamic area. The ATTACH routine branches to that portion of ATTACH which passes control to the routine to be attached (below). If the RB of the requested routine is an active RB on the loaded program list, the ATTACH routine determines if the RB is a minor LPRB (see Section 2 for a definition of minor RBs). If it is not a minor LPRB, ATTACH searches the resident reenterable module area. If it is a minor LPRB, the ATTACH routine issues a GETMAIN macro instruction to get 32 bytes of main storage for a PRB. ATTACH then initializes this RB and branches to the portion of ATTACH which passes control to the routine to be attached.

If a usable copy of the module has been found, or was loaded by FINCH, the ATTACH routine places the RB for the attached routine on the active RB queue, zeros the wait count, and sets the active bit in the RB for the attached routine. ATTACH issues a GETMAIN macro instruction to get 72 bytes of main storage to be used as a save area by the attached routine. ATTACH alters its SVRB resume PSW so that after branching to the Dispatcher to pass control to the attached routine, it will regain control (from the dispatcher when the attached routine completes executing) at the point where it determines if an event control block (ECB) is to be posted (entry point IEA0AT02).

ATTACH builds, at location 8, the PSW which will be loaded to pass control to the attached routine, and loads register 14 with the address of an SVC 3 instruction to be executed when the attached routine completes execution. ATTACH then loads the PSW built at location 8, passing control to the attached routine.

Upon completion of the attached routine, ATTACH determines if the attaching routine specified an ECB to be posted. If an ECB is to be posted, ATTACH determines if the ECB "complete" bit is already set. If it is not set, ATTACH posts the ECB by setting the "wait" bit off and the "complete" bit on, and then frees the 72 byte save area which was used by the attached routine. If the "complete" bit is already set, or if an ECB is not to be posted, ATTACH only frees the 72-byte save area.

If the attaching routine specified an exit routine to receive control after completion of the attached routine (in the ETXR parameter of the ATTACH macro instruction), ATTACH issues a GETMAIN macro instruction to get main storage for an IRB and a 72-byte save area. ATTACH then initializes the IRB and places it on the active RB queue between the ATTACH SVRB and the RB of the attaching routine. ATTACH then issues an SVC 3 instruction to pass control to EXIT. If an exit routine was not specified by the attaching routine, ATTACH issues an SVC 3 instruction to pass control immediately to EXIT and eventually to the attaching routine.

DELETE (IEAADL00) CHART EF

If the specified module name is IGC019xx and DELETE finds the module on the list of Resident Access Method modules (RAM list), DELETE returns control to the caller.

DELETE next searches the load list and the job pack area queue (JPAQ) for a loaded request block (LRB/LPRB) containing the specified entry-point name. If such an LRB/LPRB is not found, the routine sets up a return code (4) and returns control to the caller. DELETE obtains the load-list origin from the TCBLLS field of the current TCB and the JPAQ origin in the PIB.

If the specified entry-point name is found, DELETE subtracts one from the "use" count (XRBUSE) in the request block for the specified module. This count is a record of the number of outstanding load requests for the module. Each execution of DELETE will similarly decrease the use count until the count reaches 0. A use count of 0 indicates that there are no outstanding load requests; that is, there have been as many delete requests for the module as there have been load requests. If the use count in the request block is 0, the routine removes the element from the loaded program list, and issues a FREEMAIN macro instruction to free the main storage occupied by the module and its RBs (both major and minor RBs, if both types exist). After freeing the module's main storage, DELETE returns control to the caller with a return code of 0.

If the count is not 0, at least one requesting program within the current task has not completed its use of the module. That is, the program has not yet issued a RETURN macro instruction, nor has a DELETE macro instruction been issued for it. Since the module's main-storage areas cannot be freed, the routine returns control to the caller.

LOAD (IEAATC00) CHART EB

LOAD searches the queues of the job pack area (MFT with subtasking only), the resident access method area (optional), the loaded program list, and the resident reenterable module area (optional) for the request block of the requested module. The following table summarizes the possible conditions and the resulting actions taken for each.

Condition	Action
1. *RB found on JPAQ and Minor LPRB on load list	Issue SVC 13 to abnormally terminate the task
2. *LRB or LPRB found on JPAQ	Go to action 13
3. *	Create FRB and place on top of JPAQ
4. RB found on RAM list	Go to action 12 (skip action 13)
5. RB found on load list	Dequeue FRB, remove all WLE SVRBs from wait state
6. Minor LPRB found on load list	Issue SVC 13 to abnormally terminate the task
7. LRB or LPRB found on load list	Go to action 12
8. *RB found on resident reenterable module area	Create dummy LPRB and place it on JPAQ or load list. Go to action 12
9.	Use FINCH to bring the module into main storage
10. *Module loaded by FINCH is reenterable	Queue RB on JPAQ. Go to action 12
11.	Queue RB on Load List
12.	Dequeue FRB; remove all WLE SVRBs from wait state
13.	Increment use count in RB
14.	Load entry point address into register 0
15.	Issue SVC 3
*MFT with subtasking only	

LINK (IEAATC00) CHART EB

LINK searches the job pack area queue (MFT with subtasking only), the loaded program list, and the resident reenterable module area for the request block of the requested routine. If the module is represented by a FINCH request block (FRB) on the job pack area queue, indicating that the module is currently being loaded as a result of a previous request, LINK queues a wait list element (WLE) for this request to the FRB, indicates that a task switch is necessary by setting NEW to 0, sets the resume PSW in the SVRB for LINK to reenter LINK, and branches to the Dispatcher.

If the module is represented on the job pack area queue or the loaded program list by a dummy LPRB, indicating that the module was found in the resident reenterable module area by a previous load request, or if the module is found in the resident reenterable module area, LINK creates a dummy LPRB for this request.

If the module is not found, LINK uses FINCH to bring a copy of the module into the partition. When an accessible copy of the module exists in main storage, LINK determines if entry was caused by a dummy PRB created by the subtasking ATTACH routine (IEAQAT00). If so, the dummy PRB is dequeued from the subtask TCB and its area is freed.

If the accessible module is represented by an LRB, or by a minor LPRB and this is not processing as a result of an SVC 6 instruction being executed for the subtasking ATTACH routine, LINK issues an SVC 13 to abnormally terminate the task.

If a copy is available for use, LINK initializes the request block for the module and places it on the TCB RB queue following the SVRB for LINK. If the XCTL option has been specified in a STAE macro instruction issued by the program that issued the LINK macro instruction, the RB address fields in the STAE control blocks are updated to reflect the address of the new request block. LINK terminates processing by issuing an SVC 3 instruction (EXIT).

XCTL (IEAATC00) CHART EB

If the calling routine is a type 3 or 4 SVC routine, XCTL either finds the module in the resident SVC area (optional) or uses FINCH to bring the module into the SVC transient area. XCTL then branches to that part of LINK that initializes the request block, places it on the RB queue and exits.

If the XCTL was issued by a type 2 SVC routine, or by a routine represented by an IRB or an SIRB, XCTL issues an SVC 13 to abnormally terminate the task.

If the request was not from a type 3 or 4 SVC routine, XCTL removes the caller's request block from the RB queue. If the calling routine was not brought into main storage by LOAD, the main storage for the routine and its request block is freed.

XCTL then searches the queues of the job pack area, the loaded program area, and the resident reenterable module area. The search of these queues by XCTL, the loading by FINCH when the module is not found, and the final processing of the request block is the same as that done by LINK, except that XCTL does no processing for ATTACH.

FINCH (IEAATC00) CHARTS EC AND ED

The FINCH subroutine is entered by a branch and link instruction from the following entry sources to cause the loading of a specified module:

- ATTACH
- LINK
- LOAD
- XCTL from a user program.
- XCTL from a routine in the SVC transient area.
- XCTL from a routine in the I/O error transient area.
- SVC SLIH loading the SVC transient area.
- Dispatcher refreshing the SVC transient area.
- LOAD from the SVC library.

FINCH consists of initialization instructions and several sections of code. These sections are either executed or bypassed, depending upon the routine which branched to FINCH. Thus, there are eight different combinations of instructions executed (requests from LINK and XCTL are processed identically) depending upon the source of the request. Each section of code is immediately preceded by a BXLE instruction and a comment card that indicates which requests will be processed by that section of code.

Each branch and link instruction to FINCH is followed in the requesting routine by a one-byte identifier used to access one

of the nine halfword constants within FINCH. The bit pattern of each halfword is used by BXLE instructions within FINCH to determine which sections of code will be executed for each request.

With the appropriate halfword constant in the high-order position of register JBR, the format of the BXLE instruction is:

```
BXLE  JBR,JBR,addr
```

where addr is the label of another instruction, or an address relative to the address of the BXLE instruction (i.e., *+4, *+8, etc.); this is the address used when the branch is taken.

The result of the BXLE instruction is effectively a shift left single (which sets a condition code) and a branch on condition 4. If the high-order bit in the register after the shift left single is 1, the branch will be taken. If the high-order bit is 0, the branch will not be taken. Figure 36 illustrates the BXLE instruction as used in a program that has three entry sources.

Generally, FINCH uses the data management BLDL routine to locate a named routine on an external storage device. Using the information provided by BLDL, FINCH initializes the Program Fetch parameters and uses the Program Fetch routine to bring the specified routine into main storage. FINCH processes the necessary RBs when issuing GETMAIN, and initializes them with the RB type and size of the storage space they and their routines occupy.

```
ENTER   EQU *                    INITIALIZE BITS FOR BXLE
* ALL   (PROGA, PROGB, PROGC)
        LH   JBR,0(JRETRN)           GET IDENTIFIER FROM CALLER
        LH   JBR,BITTAB(JBR)         LOAD HALFWORD BIT PATTERN
        SLL  JBR,16                  MOVE BIT PATTERN TO HIGH ORDER
        BXLE JBR,JBR,SECT2           SECT2 FOR PROGC
             *,PROGA,PROGB
SECT1   .
        .
        .
        BXLE JBR,JBR,SECT3           SECT3 FOR PROGB
             *,PROGA,PROGC
SECT2   .
        .
        .
        BXLE JBR,JBR,OUT4            OUT4 FOR PROGA
             *,PROGB,PROGC
SECT3   .
        .
        .
        BXLE JBR,JBR,2(JRETRN)       RETURN FOR PROGC
             *,PROGA,PROGB
OUT4    .
        .
        BC   15,2(JRETRN)            RETURN FOR PROGA AND PROGB
BITTAB  EQU  *                       INDEXED BY CALLER INDENTIFIER
             *,PROGA USES BIT PATTERN 0001 0000 0000 0000
             *,WHICH IS D'4096' OR X'1000'
        DC   H'4096'                 PROG A
             *,PROGB USES BIT PATTERN 0010 0000 0000 0000
             *,WHICH IS D'8192' OR X'2000'
        DC   H'8192'                 PROG B
             *,PROGC USES BIT PATTERN 0101 0000 0000 0000
             *,WHICH IS D'20480' or X'5000'
        DC   H'20480'
JBR     EQU  9
JRETRN  EQU  14
```

Figure 36. BXLE Instruction

Figure 37 gives a more detailed description of the processing involved. The left column is the label for the first instruction executed for that section of code. The center column indicates the source of the FINCH request for which this code will be executed. The right column describes the processing done within that section of code.

IDENTIFY (IEAAID00) CHART DE

IDENTIFY determines if the identified entry point is within the boundaries of the routine that issued the IDENTIFY macro instruction, or within the boundaries of a routine represented on the job pack area queue (with subtasking only) or on the loaded program list. If the identified entry point address is not within the correct boundaries, IDENTIFY places a return code of 12 (decimal) in register 15 and issues an SVC 3 (EXIT) instruction.

If the entry point address is within the correct boundaries, IDENTIFY checks the RB (field XSTAB) of the routine that contains the identified entry point to determine if that RB is a PRB or an LPRB. If it is not, IDENTIFY places a return code of 16 (decimal) in register 15 and issues an SVC 3 (EXIT) instruction.

If the RB of the routine that contains the entry point address is a PRB or an LPRB, IDENTIFY determines if the name of the entry point is a duplicate of any RB program name (RB field XRBNM) already on the loaded program list, JPAQ, or TCB RB Queue. If it is, IDENTIFY places a return code of 4 (duplicate address in minor LPRB) 20 (duplicate name in minor LPRB), or 8 (duplicate name found in other than minor LPRB) in register 15 and issues an SVC 3 (EXIT) instruction. If it is not, IDENTIFY obtains main storage for a minor LPRB and initializes the minor LPRB on a chain of minor LPRBs attached to the major RB (see Figure 38). IDENTIFY then places a return code of 0 in register 15 and issues an SVC 3 (EXIT) instruction.

SYNCH (IEAASY00)

SYNCH first issues a GETMAIN for a 32-byte request block in lower storage. SYNCH then queues this RB on the RB queue below the SVRB for SYNCH. SYNCH initializes the PSW location in the RB (RBPSW) to address the location specified in register 15 upon entry to SYNCH. (The PSW is set in problem state with the task protection key recorded in the TCB.) SYNCH exits via an SVC 3 instruction.

Label or Instruction	Source of Request	Operation
IEA0FN00	Entry point for the ATTACH, SVC SLIH, and Dispatcher Entry for XCTL from I/O error transient area	If an error routine is requested and is found in the I/O error transient area or is represented on the error recovery procedure load list (IEAAERP), the entry point address in the resume PSW in the SIRB is reset to point to the error routine. FINCH issues an SVC 3 (EXIT) instruction.
XFINCH	All Entry for XCTL, XCTL from SVC transient area, LINK, LOAD, LOAD from SVC library	FINCH identifies the caller via entry code. BXLE register is loaded with bit pattern.
MMLABEL	LOAD	FINCH indicates that main storage in the low area of the partition is to be obtained.
IEAARAM0	All except LOAD from SVC library	A GETMAIN macro instruction is issued for Program Fetch work area.
XF1	Load from SVC library, XCTL from SVC or I/O error transient area, SVC SLIH loading SVC transient area, Dispatcher refreshing the SVC transient area (if GETMAIN failed)	The PANIC is used as a work area for Program Fetch. This happens only if GETMAIN failed to obtain work area. Load from SVC library always uses the PANIC.
XFINCH1	All	If LCS is in the system, the scatter table list is cleared. FINCH initializes the BLDL address in the FINCH parameter list.
(no label)	ATTACH, LINK, LOAD, XCTL from SVC transient area, and LOAD from SVC library	Registers 0 and 1 are initialized with pointers to name of desired program or pointers to BLDL. The name of program in register 0 and the address of the DCB to be initialized goes in register 1. Move BLDL list, if provided to work area.
XF2	XCTL from I/O error transient area, SVC SLIH, Dispatcher	FINCH initializes BLDL parameters.
XF12	All except Dispatcher refilling transient area (BLDL list is not provided)	FINCH finishes BLDL initialization and links to data management BLDL routine to locate the requested module on auxiliary storage. (This information is already available to EXIT.)
XFINCH2	ATTACH, LINK, LOAD, XCTL, and LOAD from SVC library	Information obtained from BLDL operation is tested. Determination is made of status of program.

Figure 37. FINCH Processing (Part 1 of 4)

Label or Instruction	Source of Request	Operation
LRBCMPD	ATTACH, LINK, LOAD, XCTL, LOAD from SVC library	FINCH determines the size of the requested program and its RB prior to obtaining storage space for the routine. This information is obtained from BLDL list. If the requested module is an overlay module, space for a note-list is needed.
	LOAD, LOAD from SVC library	Indicates storage to be obtained from high area of partition.
XF5 (LCS Only) XFLCS1 XFLCS2	ATTACH, LINK, LOAD, XCTL, LOAD from SVC library	If the system has the LCS Option, FINCH clears the scatter translation table. FINCH then checks whether hierarchy 0 or 1 is requested, and prepares GETMAIN parameter. FINCH checks if program is to be scatter loaded and hierarchy was not indicated. If program is not to be scatter loaded, FINCH sets GETMAIN parameter to indicate H0.
XFLCS3	ATTACH, LINK, LOAD, XCTL, LOAD from SVC library	If a job pack area queue does not exist for this partition, a REGMAIN is issued to get storage for its RB.
PIBON, SAVEDATA YESRENT	LOAD, LOAD from SVC library (job pack area queue exists)	Since the FRB built by the last load request (the request now in FINCH) is queued first, FINCH obtains the address of the current FRB from the PIB. FINCH, upon determining that the module to be fetched is reenterable, prepares to obtain RB storage from subpool 240. FINCH adjusts the FRB to indicate reenterability.
ENDFNI LEAVEJX	ATTACH, LINK, XCTL, LOAD, LOAD from SVC library (job pack area queue exists)	If module is not reenterable, FINCH adjusts FRB to reflect this status. FINCH issues a register-form GETMAIN macro instruction for the program and its RB.
ENDFN2 XFLCS4	ATTACH, LINK, XCTL, LOAD, LOAD from SVC library	FINCH then initializes RB and checks for LRB and validity checking.
XFSCHED	ATTACH, LINK, LOAD, XCTL, LOAD from SVC library	FINCH inserts either job library or link library DCB address in Program Fetch parameter list if request entered with a 0 in DCB address field.
XF10	Dispatcher	Initializes Program Fetch parameter list.
XF4	Transient area routines (XCTL from SVC transient area, XCTL from error transient area, SVC SLIH, Dispatcher)	FINCH initialized the Fetch parameter list (inserts TTR, STAB, SVCDCB address, etc.)

Figure 37. FINCH Processing (Part 2 of 4)

Label or Instruction	Source of Request	Operation
XSNTQUES	XCTL from SVC transient area, SVC SLIH, Dispatcher	If this request is from time slice task and the previous request was also from a time slice task which is still dispatchable, FINCH does not allow this request to preempt the previous request for the transient area. If the requested module is already in the transient area, FINCH branches to XSNTEXIT.
QAVBLE	XCTL from SVC transient area, SVC SLIH, Dispatcher	If the transient area is not currently being loaded, the address of the TCB for the transient area loading task is placed in NEW, and the SVRB for the task is set dispatchable.
QUEREG	XCTL from SVC transient area, SVC SLIH, Dispatcher	The boundary box for the requesting task is placed on the transient area request queue, the address of the SVRB for the request is placed in the boundary box, and the SVRB is put in a wait state.
TMSLCEXT	XCTL from SVC transient area, SVC SLIH, Dispatcher	The resume PSW in the SVRB is set to the address of XSNTQUE to allow reprocessing if this request is not handled at this time.
DISPEXIT	XCTL from SVC transient area, SVC SLIH, Dispatcher	If transient area is currently being loaded, sets NEW to 0. For either case, FINCH branches to the dispatcher to give control to the transient area loading task.
IEAFNCH	XCTL from SVC transient area, SVC SLIH, Dispatcher	The resume PSW in the transient area loading task's SVRB points to the label IEAFNCH. This code (through DEQEND) is executed under the TCB and SVRB for the transient area loading task. The address of the TCB is loaded into register 4 for Program Fetch, and the system is disabled for interruptions.
XF6	All	FINCH branches and links to Program Fetch (IEWMSEPT) which brings the requested module into main storage.
(no label)	XCTL from SVC transient area, SVC SLIH, Dispatcher	If an error occurred in Program Fetch FINCH retries to load the module by branching to XF6 (maximum retries = 4).
DDRINTF	XCTL from SVC transient area, SVC SLIH, Dispatcher	If the error persisted in Program Fetch, FINCH branches to the Dynamic Device Reconfiguration support so that the system residence device may be changed. FINCH then branches to XF6 to retry the loading.

Figure 37. FINCH Processing (Part 3 of 4)

Label or Instruction	Source of Request	Operation
RECOVER1	XCTL from SVC transient area, SVC SLIH, Dispatcher	If the error occurs on the new device, FINCH branches and links to ABTERM to schedule abnormal termination of the task. Return is to ERROR.
NOERROR	XCTL from SVC transient area, SVC SLIH, Dispatcher	If the requesting task is time-sliced, the pointers to the TCB and the SVRB are saved so that it will not be preempted.
ERROR NOTTMSLC	XCTL from SVC transient area, SVC SLIH, Dispatcher	The switch indicating that the transient area is currently being loaded is set off. The transient area queue is freed, and any SVRBs on it are removed from the wait state.
DEQEND	XCTL from SVC transient area, SVC SLIH, Dispatcher	The transient area loading task is reinitialized. FINCH branches to DISPEXIT (above).
XSNTEXIT (Branched to from XSNTQUES when requested module is in SVC transient area)	XCTL from SVC transient area, SVC SLIH, Dispatcher	The SVRB for the requested module is initialized and the work area is freed. FINCH returns to the caller.
XF8	ATTACH, LINK, LOAD, XCTL from I/O error transient area, LOAD from SVC library	If an error occurred during Program Fetch, the task is abnormally terminated. If the Automatic System Recovery option is in the system, and the loaded module is refreshable, the refreshable bit is initialized in the loaded module's RB. FINCH enables callers that were pseudo-disabled. If the program is being tested, FINCH passes control to the TESTRAN interpreter. If the requested routine is an overlay program, FINCH issues a LOAD macro instruction to bring overlay supervisor 2 into the system (see "Overlay Supervisor").
XF14	ATTACH, LINK, LOAD, XCTL, LOAD from SVC library	FINCH returns to the calling routine.
XF13	XCTL from I/O error transient area	FINCH loads the address of the error transient area into register 15 and disables the PSW in the SIRB. FINCH branches and links to the calling routine. Upon return, FINCH issues an SVC 3 (EXIT) instruction.

Figure 37. FINCH Processing (Part 4 of 4)

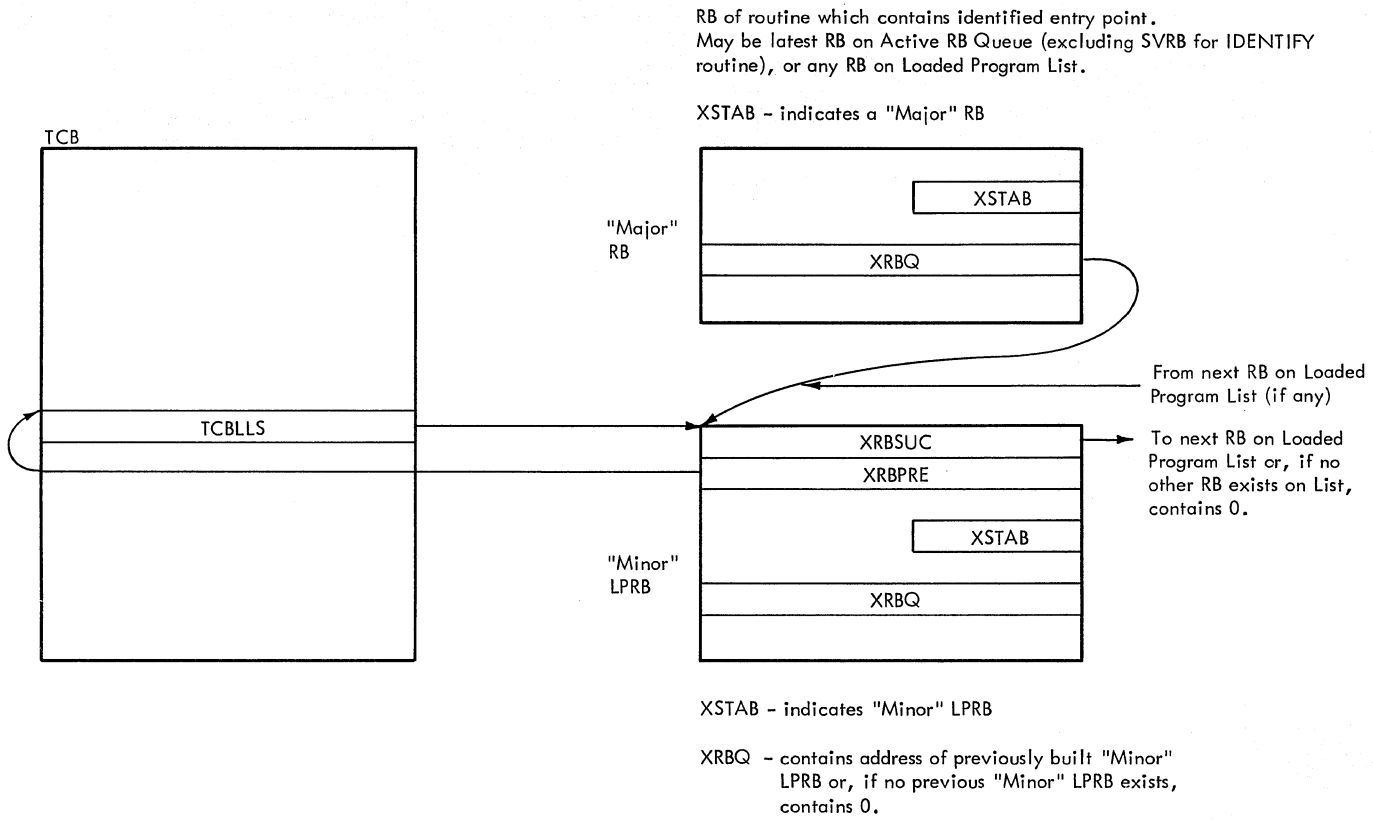


Figure 38. Queueing a Minor LPRB FOR Identify

GETMAIN (IEAAMS) CHART FA

Both the GETMAIN and FREEMAIN macro instructions may be written in two forms, S (storage type) and R (register type). S-type macro instruction supplies parameters in a parameter list; an R-type supplies parameters in the general registers.

For a S-type request, the GETMAIN routine determines whether it is a list request (a request for more than one area of main storage). In the event of a list request, GETMAIN passes control to ABTERM to schedule abnormal termination of the task, since list requests are not valid in MFT systems.

GETMAIN then tests the subpool numbers. All valid main storage requests from problem programs are allocated from the highest available main storage in the partition, regardless of the subpool number specified in the GETMAIN macro instruction. For problem programs, only subpool numbers 0 through 127 are considered valid. All others cause abnormal termination of the requester.

When a system task issues a GETMAIN macro instruction specifying subpools 0 through 127, storage is allocated from the low-address portion of the partition; when the macro instruction specifies a subpool from 128 through 244 or 246 through 254, storage is allocated from the high-address portion of the partition. Subpools 245 and 255 get storage from the system queue area (SQA).

GETMAIN indicates whether the request is to be satisfied from the high- or low-end of the partition. The free area queue is then searched, beginning with the free area queue element (FQE) that has the highest address in the partition. GETMAIN compares the size of the request to the size of the free area indicated by that FQE. If the request is larger than the free area, the next FQE is checked. As GETMAIN checks each FQE, it keeps a record of the largest free area found. When a free area equal to or larger than the requested size is found, GETMAIN allocates the requested main storage from the high end of this free area.

If the request is for space in the low-end of the partition, GETMAIN saves the address of the first free area of sufficient size and searches the free area queue. As each FQE is examined, the address of the lowest free area of sufficient size to fulfill the request is always saved. When all FQEs have been checked, the requested main storage is allocated from the low end of the lowest free area that is large enough.

If an area large enough to satisfy an R-type or an unconditional S-type request is not found, or if a request was made for zero bytes of storage, GETMAIN identifies the caller. If either ABEND or the SVC SLIH issued the request, an error code is placed in register 15, and control is returned to the caller. This eliminates the possibility of entering a loop caused by rescheduling an abnormal termination. If any other task issued the GETMAIN macro instruction, control is passed to ABTERM to schedule abnormal termination of the caller.

If an area large enough to satisfy a conditional S-type request (both variable and nonvariable) is not found, GETMAIN places a return code of 4 in register 15 and returns control to the caller.

When GETMAIN allocates main storage, the address of the allocated space is returned to the caller in register 1 for R-type requests, or in the area indicated by the pointer in the parameter list for S-type requests. If the request is for a variable amount of space, the size of the allocated space is also returned in the area indicated by the parameter list.

GETMAIN places a return code of 0 in register 15 (indicating that the request was satisfied), removes the newly allocated space from the free area queue or modifies the FQE if the whole space was not allocated, and branches to the Type-1 SVC Exit routine to return control to the caller.

If SMF is in the system, the GETMAIN routine also passes control to its SMF storage information subroutine (GMSMFCRE). It tests the TCBTCT field of the TCB for the address of a Timing Control Table (TCT). If this field contains 0's, there is no TCT and storage usage information cannot be recorded. If the TCBTCT field contains a TCT address, the subroutine determines whether the newly allocated storage alters either the low-water mark (LWM) or the high-water mark (HWM). (The LWM is the value of the highest storage address allocated from the bottom of the partition. The HWM is the value of the lowest storage address allocated from the

top of the partition.) If either is altered, the SMF subroutine stores the new value in the appropriate TCT field (TCTLWM or TCTHWM).

It also calculates the all-time minimum difference, in terms of 2048-byte blocks, between the LWM and the HWM. If the new allocation creates a new minimum difference, the SMF storage subroutine records the new difference in the TCTMINC field of the TCT. It then returns control to GETMAIN.

In MFT systems with subtasking, main storage supervision processing differs from systems without subtasking as follows:

- The supervisor builds a queue of gotten subtask area queue elements (GQEs) to describe the main storage obtained by system-issued GETMAIN macro instructions for a subtask unless the request specified subpool 240. The first GQE associated with a subtask originates in the TCBMSS field of the subtask's TCB. Succeeding GQEs are built in the first eight bytes of main storage acquired for a subtask.
- When a subtask terminates, the normal or abnormal termination routines refer to the GQEs chained to the subtask's TCB to free the main storage belonging to the terminating subtask. The GQEs are removed from the queue whenever the subtask's main storage is freed.

FREEMAIN (IEAAMS) CHART FA

When main storage is to be released, FREEMAIN is entered either at entry point IGC005 for S-type requests or at entry point IGC010 for R-type requests.

If any of the following errors is discovered, FREEMAIN passes control to ABTERM to schedule abnormal termination of the task:

- The request is an S-type list request (valid only in MVT).
- The area to be freed does not begin on a doubleword boundary.
- The area to be freed overlaps an existing free area.
- The area to be freed is not totally contained in the task's partition.

If none of these errors exists, FREEMAIN makes the space being freed available for reallocation. This is done either by updating the FQE of an adjacent free area or, if there is no adjacent free area, by creating an FQE and adding it to the free area queue. If a GQE is associated with and describes the entire area to be freed, the GQE is freed also.

If SMF is in the system, FREEMAIN passes control to its SMF storage information subroutine (FMSMFCRE). The subroutine first tests the TCBTCT field of the TCB for the address of a TCT. If there is no TCT, storage-usage information cannot be recorded. If the TCBTCT field contains a TCT address, the subroutine determines whether the newly released storage causes a change in the low-water mark (LWM) or the high-water mark (HWM) for the partition. (The LWM is the value of the highest storage address allocated from the bottom of the partition, and the HWM is the value of the lowest storage address allocated from the top of the partition.) If either is changed, the SMF storage subroutine stores the new value in the appropriate TCT field (TCTLWM or TCTHWM) and returns control to FREEMAIN.

FREEMAIN then branches to the Type-1 SVC Exit routine to return control to the caller.

In the description below, the modules that support the communication task with Multiple Console Support (MCS) follow the non-MCS communication task modules. The Initialization (IEECVCTI), External Interruption Handler (IEECVCRX), Attention Handler (IEECVCRA), and the Unit Control (IEECVUCM) modules are common to both MCS and non-MCS communication tasks. See Diagrams 14 and 15.

INITIALIZATION MODULE (IEECVCTI) CHART GA

The Initialization module is entered from the Master Scheduler. If MCS is included in the system and the hard copy log is a console, IEECVCTI posts the NIP message buffer ECB in the UCM so that NIP messages will be written by the module IEECMWTL. The Initialization module then searches the UCM, changing the UCB name of each device to an address. If an address cannot be determined, a message is issued to the master console and the search continues. For each UCB name that is correctly matched with an address, a message is written to the console advising the operator of its unit address, its alternate's address, its console identification number, its display area configuration, its command code and routing code authorization, its status (active or inactive), and whether it is the master console, a secondary console, or the hard copy log. The Initialization module then builds an event indication list (EIL), which contains the addresses of the attention, external, WTO, and I/O ECBs for the consoles initialized. For MCS, the EIL will also contain the addresses of the DOM ECBs.

If the system includes display consoles, control is passed to the Graphic Console Initialization module (IEECVGCI). Upon return from IEECVGCI, control returns to the Master Scheduler IPL routine.

GRAPHIC CONSOLE INITIALIZATION (IEECVGCI) CHART GY

The Graphic Console Initialization module is entered from the Console Initialization Module when that routine determines that console initialization is required for one or more display consoles. IEECVGCI first searches for a UCM representing a display console. If none are found, control is returned to the Console Initialization routine. If one is found, the routine issues a LOCATE macro instruction to search for SYS1.DCMLIB, and, if found, issues an OPEN macro instruction to open SYS1.DCMLIB. If either the LOCATE or the OPEN fail, an error message is written to the operator's console and one console in each transient group is made resident (the console whose TDCM was initially placed in the transient area). If SYS1.DCMLIB is successfully opened, IEECVGCI attempts to read a copy of the PFK definitions for each console into main storage. If the read is unsuccessful, a message is issued to the operator. When all display consoles in the system have been initialized, control is returned to the Console Initialization module.

WRITE TO OPERATOR MODULE (IEEMFWTO, IEENFWTO) CHART GC

This module is entered as a result of an SVC 35 instruction issued from a WTO or WTOR macro instruction. IEEMFWTO handles processing for MCS systems; IEENFWTO handles processing for non-MCS systems. If entered for a WTOR or WTP request, control is passed to the Write to Programmer module IEEWTP00 or the WTOR module IEEVWTOR respectively.

If entry is for a multiple-line WTO, control is passed to either the Non-MCS MLWTO routine (IEECVML1), or to the MCS MLWTO routine (IEECVML3) as appropriate. Upon return from either of these two modules or if this is a WTO request, IEEMFWTO or IEENFWTO edits the message, moves the text into an available Write Queue Element (WQE), and puts the WQE on the system output queue that originates in the UCM.

If a WQE is not available and the communication task is issuing the WTO(R), entry is from DAR or System Log, or entry is from the system error task, IEEMFWTO or IEENFWTO issues a GETMAIN macro instruction to obtain main storage for a WQE. For other cases when a WQE is not available, IEEMFWTO or IEENFWTO issues an ENQ macro instruction to gain exclusive access to a buffer. Upon return from ENQ, if a WQE is still not available, a WAIT macro instruction is issued specifying the WQE message buffer request ECB in the UCM. When a WQE becomes available, processing resumes with message editing (above).

When the WQE has been placed on the system output queue, IEEMFWTO or IEENFWTO posts the WTO ECB in the UCM and branches to the EXIT routine. If MCS is in the sys-

tem, the WQE ID number is placed in the WQE, and MCS routing codes, descriptor codes, and flags are moved into the WQE before the WTO ECB is posted.

NON-MCS MULTIPLE-LINE WRITE TO OPERATOR (IEECVML1, IEECVML2, IEECVML4) CHART GD

IEECVML1 is entered from IEENFWTO to process requests for multiple-line WTOs in a system without MCS. This routine builds the major WQE for the multiple-line WTO. Upon initial entry, the routine checks to see if entry has been made to add additional lines to an existing WTO. If so, the major WQE has already been built, so control is passed to IEECVML2.

If IEECVML1 is entered to build a major WQE, the first line passed is checked to see that it is within the user's core. Also, the number of lines passed is determined. If the number is zero or less, the line number field in the WQE is set to 1. If the number of lines passed is greater than 10 for a problem program, the number will be set to 10; if the number is greater than 255 for a supervisor mode or protect key 0 program, the number will be set to 1.

If no error conditions are found, the routine attempts to obtain buffer space for the major WQE. If space is not available, and the routine issuing the request is the communications task, DAR, or an SIRB, buffer space in main storage is immediately obtained by means of the GETMAIN macro. Otherwise, the routine issues an ENQ macro and a WAIT macro (specifying the WQE message buffer request ECB in the UCM). When a buffer becomes available, it is allocated from fixed buffer space.

When the buffer has been obtained, the text fields are filled in, and exit is then made to IEECVML4.

IEECVML4 completes building of the major WQE. Upon entry, this routine stores the MLWTO identification number and partition identifier in the appropriate areas of the major WQE. If there are minor WQEs to be built, exit is made to IEECVML2. Otherwise, the WTO ECB in the UCM is posted and exit is made to the calling routine.

If additional lines are to be added to an existing WTO (requiring minor WQEs), control is passed to IEECVML2. Upon entry, this routine checks for available buffer space for the minor WQE.

If none is available, and the routine is the communications task, DAR or an SIRB, a GETMAIN macro is issued immediately for the required main storage. Otherwise, the routine issues an ENQ and a WAIT macro (specifying the WQE buffer request ECB in the UCM). When a buffer is obtained, the text fields are filled in. If more lines remain to be output, processing continues. When the end line is reached (all lines having been placed on the system output queue), the routine posts the WTO ECB and exits to the calling routine.

MCS MULTIPLE-LINE WRITE TO OPERATOR ROUTINES (IEECVML3, IEECVML5, IEECVML6, IEECVML7) CHART GD

IEECVML3 is entered when the Write-to-Operator routine (IEEMFWTO) determines that a multiple-line WTO request has been entered in a system with multiple console support (MCS). This routine builds a major WQE for a multiple-line WTO request.

Upon entry, the routine checks to see if a major WQE already exists for the request. If so, the request is to build a minor WQE and control is passed to IEECVML5 to connect minor WQEs to the existing major WQE. If the request is to build a major WQE, the routine attempts to obtain WQE buffer space in main storage for the major WQE. If space is not available, and the request is from the communications task, DAR or an SIRB, an ENQ macro and a WAIT macro are issued for the WQE ECB. When buffer space has been obtained, the text fields are filled in. Control is then passed to IEECVML5.

IEECVML5 completes processing of the major WQE and begins processing of the minor WQE. If entry is to complete a major WQE, the MCS fields of the major are filled in. If no minors are to be chained to the major, control is passed to IEECVML6 to chain the major to the system output queue. If entry is made to connect minors to a major (or if, upon completion of a major, minors are ready to be connected to it), buffer space for the minor WQEs is obtained in the same way that it was for the major WQE. (On subsequent entries to build minor WQEs, a check is made to see if any minor WQEs previously connected to the major have become available for reuse by having been written to all the consoles required by their routing indicators. If so, their buffer space is reused for another minor WQE.) As each minor WQE is completed, control is passed to IEECVML6 to chain the request to the system output queue.

If IEECVML6 is entered to chain a major WQE to the system output queue, the partition identifier (under MFT) and the MLWTO identification number are placed in the appropriate areas of the major WQE. If entry is to chain minor WQEs to the major, they are chained and appropriate hardcopy information and use count information is

moved into them from the major. If there are additional minor WQEs to be processed, control is returned to IEECVML5. Otherwise, control is passed to IEECVML7 which DEQs from the WTO resource if necessary, sets the appropriate return code in register 15, and returns control to the routine that issued the request with an appropriate return code in register 15.

WRITE TO OPERATOR WITH REPLY MODULE (IEEVWTOR) CHART GE

This module is entered from IEEMFWTO or IEENFWTO to process WTOR requests. If the parameter list is invalid or the message length is not greater than 0, IEEVWTOR issues an ABEND macro instruction to abnormally terminate the task. If reply queue element is available, it is reinitialized and placed on the reply queue element chain which originates in the UCM.

If a reply queue element (RPQE) is not available and the WTOR is issued by the communication task, System Log, DAR, or the system error task, IEEVWTOR issues a GETMAIN macro instruction to obtain main storage for a reply queue element. Otherwise, if a reply queue element is not available, an ENQ macro instruction is issued to gain exclusive access to an RPQE. Upon return from ENQ, if a reply queue element is still not available, IEEVWTOR issues a WAIT macro instruction specifying the reply queue element buffer request ECB in the UCM. When a buffer becomes available, processing resumes with initializing the reply queue element. IEEVWTOR issues an XCTL macro instruction to return control to IEECVWTO.

EXTERNAL INTERRUPTION HANDLER (IEECVCRX) CHART GB

This module receives control as a result of an external interruption and branches topost the external event control block (ECB) in the unit control module.

ATTENTION INTERRUPTION HANDLER (IEECVCRA) CHART GB

This module receives control as a result of an operator caused attention at an input or output device. IEECVCRA branches to post the attention event control block in the unit control module.

WAIT MODULE (IEECVCTW) CHART GF

This module issues a WAIT macro instruction specifying a list of ECB addresses (EIL). When one of the ECBs is posted, IEECVCTW issues an SVC 72 instruction which builds an SVRB for nonresident modules and gives control to the Router module, IEECVCTR.

ROUTER MODULE (IEECVCTR) CHART GG

This module determines from the ECBs pointed to by the event indication list (EIL) the type of interruption that has occurred. IEECVCTR issues an XCTL macro instruction to pass control to the appropriate, nonresident, device-processor routine.

CONSOLE SWITCH MODULE (IEECVCTX) CHART GH

This module receives control from the Router module to switch from the primary to the alternate console as a result of an external interruption.

UNIT CONTROL MODULE (IEECVUCM)

This nonexecutable resident module contains pointers, ECBs, and storage areas used by the communications task.

The Initialization (IEECVCTI), External Interruption Handler (IEECVCRX), Attention Handler (IEECVCRA), and Unit Control (IEECVUCM) modules, which are common to both MCS and non-MCS communications tasks, are described under "Communications Task Routines."

MCS MINI-ROUTER MODULE (IEECMCTR)

This routine is entered as a result of an SVC 72 instruction issued by the modules IEECMDSV, IEECMAWR, IEEC2740, and IEECMDOM. An SVRB is created as a result of the SVC instruction for the execution of this and the nonresident, device-processor modules. IEECMCTR issues an XCTL macro instruction to pass control to the appropriate device-processor module. See Diagram 16.

MCS ROUTER MODULE (IEECMAWR) CHART GI

The Router contains a WAIT macro instruction specifying the EIL and is entered when one of the five ECBs in the UCM or an I/O ECB is posted, or when previous communication task processing requires additional output processing or queue management. As the highest priority ready task in the system, the Router examines the ECBs and queue management and output processing indicators in the following order:

1. Recovery management
2. External interruption
3. Attention interruption
4. I/O completion
5. Output processing
6. WTO, WTOR, WTP processing
7. Queue management
8. DOM processing
9. NIP message-buffer processing

The Router determines the processing necessary and passes control to the appropriate communications task service module. When control returns, the Router re-tests all ECBs and status indicators that may have changed while the first interruption was being processed. To allow processing or higher-level interruptions, output processing returns control to the Router after each console output queue is processed (see IEECMDSV below). When no further processing can be done, the Router reissues the WAIT macro instruction, and control returns to the Dispatcher.

MCS CONSOLE SWITCH MODULES (IEECLCTX, IEECMCTX, IEECNCTX, AND IEECOCTX) CHART GJ

The Console Switch modules receive control from:

- The Router to switch master consoles as a result of an external interruption.
- The device support routines to switch to the alternate console when there is an unrecoverable I/O error on a console.
- SVC 34 to switch to the alternate of the master console as a result of a VARY MSTCONS command.
- IEECMDSV to switch the hard copy from the SYSLOG to the master console when SYSLOG is inoperative.

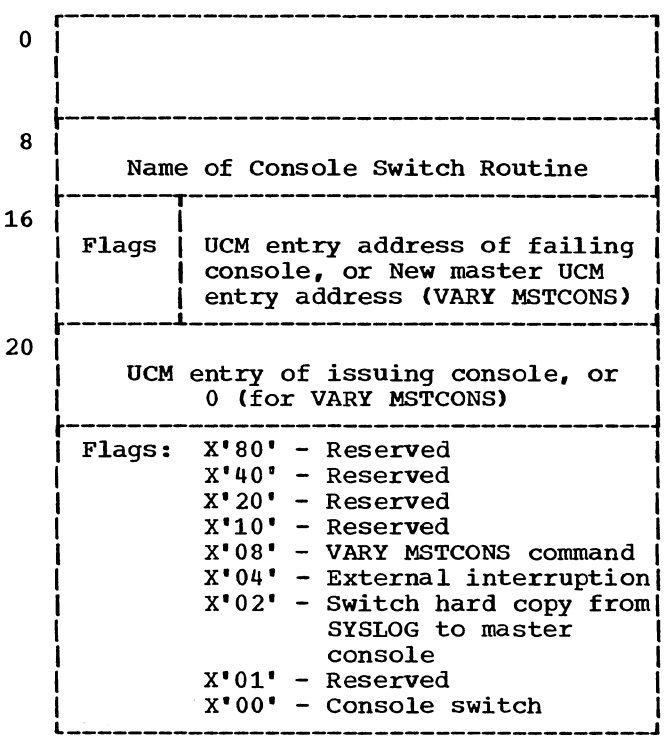
IEECLCTX uses the parameter list passed by the calling routine to determine the type of function required. If entered because of a VARY MSTCONS command, IEECLCTX adds the routing and command codes of the master console to those of its alternate. IEECLCTX passes control to IEECOCTX if the master console was the hard copy log and its alternate is a graphics console. Otherwise, control is passed to IEECMCTX to issue indicative messages.

If IEECLCTX was entered because of hard copy failure on SYSLOG, control is passed directly to IEECOCTX.

If IEECLCTX was entered because of an external interruption or for a failing console, the routing and command codes of the master or failing console are added to those of its alternate. If the new console is a graphics console and the old was the hard copy log, control is passed to IEECOCTX. Otherwise, for a successful console switch, control is passed to IEECMCTX.

If a secondary console does not have an active alternate, the routing and command codes are added to those of the master console. If the master console does not have an active alternate, a message is issued to all active consoles requesting a VARY

Console Switch Parameter List



MSTCONS command from any console. If there are no active consoles and a console device is included that has an audible alarm, an OPEN macro instruction is issued for the device, and the alarm is sounded three times. For all cases where a console switch cannot be successfully made, IEECLCTX returns control to the calling routine.

IEECMCTX constructs two messages to indicate to the operator that a console switch has occurred and the attributes of the new console. IEECMWSV is used to queue the WQEs to the console output queue of the new console. IEECMCTX passes control to IEECNCTX to adjust the output queues of the new console.

IEECNCTX places the WQEs that were on the output queue of the old console on the console output queue of the new console, deleting duplicate messages.

If the WQE represents an unended multiple-line WTO, it is checked for Descriptor codes 8 and 9 (indicating that it is a status display). If Descriptor codes 8 and 9 are found, the WQE is deleted. Multiple-line WTO messages, other than status displays, are passed to the new console if the end line of the message is on the old console's message queue at the time of the console switch. The close-pending indicator is set and the device busy flag is set off in the UCM entry of the old console, and IEECNCTX returns control to the calling routine.

IEECOCTX switches the hard copy log to the alternate of a new console when the new console is a graphic device and the old console was receiving the hard copy. If the alternate console is also a graphics device, IEECOCTX searches all active consoles starting with the master console for an active, nongraphics device. If none is found, the master console is specified as the hard copy device. When the hard copy log has been assigned to a console, a message indicating this is placed on the console output queue of that console, the WQEs for the hard copy log are placed on the console output queue, and IEECOCTX passes control to IEECMCTX.

IEECOCTX also switches the hard copy log from SYSLOG to the master console when there is a SYSLOG failure. If the master console is a graphics console, IEECOCTX searches for an active, nongraphics device as previously described, but does not requeue WQEs, and returns control to the calling routine instead of IEECMCTX.

MCS DEVICE INTERFACE MODULE (IEECMDSV) CHART GK

This module consists of four subroutines that control the interface with the device-support processor routines and manage the output queues for the devices. It receives control from the Router when an attention or I/O ECB has been posted, when it has been processing output and there is more output to process (see DEVSERVA), or when an output queue needs consolidating. It also receives control from the WTO processing routine IEECMWSV when there is output to be processed.

DEVSERVB receives control when an attention or I/O ECB has been posted. After testing console conditions, control is passed to DEVSERV to interface with the appropriate device-support routine.

DEVSERVA receives control when there is more output to be processed. When it is first entered, it searches from the first UCM entry for a UCM entry of an active console whose output queue can be processed. When it is subsequently entered from the Router, the search starts with the next entry after the last UCM entry processed. The search ends when an output queue is found that can be processed, or when the last UCM entry has been inspected. If an output queue is found, control is passed to DEVSERVA to interface with the appropriate device support routine.

Note: Because one WQE may be placed on several device output queues, and because DEVSERVA handles only one UCM entry each time it is entered, DEVSERVA may be re-entered from the Router to finish processing output queues. DEVSERVA must return to the Router after processing each output queue to allow for the servicing of higher priority external, attention, and I/O ECBs that may have been posted while DEVSERV was processing.

DEVSERV is entered from DEVSERVA or DEVSERVB to branch to the appropriate device support routine. Upon return, if entries on the console output queue have been processed and marked as no longer needed, control is passed to the DEQ subroutine.

The subroutine DEQ receives control when DEVSERV or CLEANUP need console output queues serviced. DEQ inspects the console output queue for WQE pointers that are no longer needed. Each WQE so flagged is marked as a null entry, and the use count of the WQE is decremented by one. If this decrementing results in the use count reaching 0 (all specified consoles have received the message), and DEQ determines that the message is to go to hard copy, control is passed to IEECMQCN (in IEECMWSV) to put the WQE pointer on the hard copy device's output queue, or a WTL is issued to SYSLOG. If the message is not to go to hard copy and there is no Reply Queue Element associated with the WQE, the WQE is freed or marked available.

If a major WQE (representing a multiple-line WTO) has a related minor WQE with a use count of 0 (indicating that it has been passed to all consoles), the WQE chain is searched. The storage space for all minor WQEs (except the last one) with a use count of zero is returned to the system. If a major WQE is flagged as no longer needed, the entire major/minor WQE chain is released via FREEMAIN. Return is made when the last WQE pointer on the console output queue has been examined.

CLEANUP receives control when system output queues need consolidation. It examines each WQE on the system output queue and passes control to DEQ for each WQE that has been serviced and is to be sent to hard copy. If the WQE isn't to be sent to hard copy, DEQ frees the WQE. Control returns to CLEANUP to process all WQEs on the system output queue.

WTO PROCESSOR MODULE (IEECMWSV) CHART GL

This module gains control from the Router when the WTO ECB is posted. WQEs that have been queued to the system output queue and have not been serviced by this routine are processed at this time. If a user exit routine is provided, a parameter list containing the text, routing codes, and descriptor codes is passed to it. If the routing codes have not been modified or suppressed, if the WQE is for a WTOR, or if there is no user exit routine, control is passed to the subroutine IEECMENQ, which compares routine codes, IDs, and hard copy requirements, and places the WQE on the appropriate console output queues.

This routine sets an output-pending flag in the UCM to which the request has been queued. When all WQEs have been examined, control is passed to IEECMDSV (DEVSERVA) to initiate output queue processing. Upon return, control is returned to the Router.

DOM SERVICE MODULE (IEECMDOM) CHART GM

This module gains control from the Router when the DOM ECB is posted (by the DOM macro instruction being issued in a problem program or system task). It may also be entered directly from the system purge routines at the end of a job step. When entered from the Router, message IDs in the DOM element list are compared with the WQEs on the system output queue. The matching WQEs are marked for deletion unless the WQE is to go only to hard copy. If a display console is active, control is passed to the processor routine for that device for deleting messages from the console screen. Upon return, the DOM element list is freed by a FREEMAIN macro instruction, and the DOM ECB is cleared.

When entered from the system purge routines, IEECMDOM compares WQEs with a protect key. Those that match are marked for deletion unless they are to go only to hard copy. If any graphics consoles exist, the messages in each device's Display Control Module are similarly compared and marked for deletion.

NIP MESSAGE BUFFER WRITER MODULE (IEECMWTL) CHART GN

This module issues an SVC 35 to write the NIP messages to the hardcopy Log or hardcopy console.

The communications task with MFT supports the following console devices in both MCS and non-MCS systems:

- IBM 1052 Printer-Keyboard Models 5 and 6
- IBM 1403, 1443, and 3211 Printers
- IBM 2540 Card Read Punch

The following console devices are also supported in MFT, but only if the Multiple Console Support option is included in the system:

- IBM 1052 Printer-Keyboard Model 7
- IBM 2250 Display Unit
- IBM 2260 Display Station
- IBM 2740 Communication Terminal Model 1
- IBM 5450 Operator Console with CRT Display

The 2250, 2260, and 5450 are discussed under "Device Independent Display Operator Console Support." Following are descriptions of modules that support the 1052, 1403/1443, 2540, and 2740. Where modules are used both with and without MCS, the differences are noted following the description of the non-MCS module.

1052 DEVICE-SUPPORT ROUTINES

The device-support routines for the IBM 1052 Printer-Keyboard perform Read, Write, Open, and Close functions as well as buffer management (see Diagram 21).

Processor Module 1 (IEECVPMX) Chart GO

This module provides I/O buffer management and constructs channel programs to perform read and write operations.

Processor Module 2 (IEECMPM1 in MCS) Chart GX (also IEECVPM1 in non-MCS)

This module provides the processing necessary to present multiple-line WTOs (including system status displays) on the 1052 Printer Keyboard operator console.

Processor Module 1 with MCS (IEECMPMX) Chart GP

Modification: This module does not perform buffer management.

Open/Close Module (IEECVOCX) Chart GQ

This module provides open and close functions for the console device.

PRINTER DEVICE SUPPORT ROUTINES

The device support routines for printers provide write, open and close functions, and buffer management.

Processor Modules (IEECVPMP, IEEVPM1)

These modules provide I/O buffer management and issues a WRITE macro instruction to cause output to the printer.

Processor Module with MCS (IEECMPMP)

Modification: This module does not perform buffer management.

Open/Close Module (IEECVOCP)

This module provides open and close functions for the console device.

Open/Close Module with MCS (IEECMOCP)

This module provides open and close functions for the console device.

CARD READER DEVICE SUPPORT ROUTINES

The device support routines for card readers provide read, open, and close functions and buffer management.

Processor Module (IEECVPMC) Chart GR

This module provides I/O buffer management and issues a READ macro instruction to bring input from the card reader into a WQE.

Processor Module with MCS (IEECMPMC) Chart GS

Modification: This module does not perform buffer management.

Open/Close Module (IEECVOCC)

This module provides open and close functions for the console device.

2740 DEVICE SUPPORT ROUTINES (MCS ONLY) CHART GT

The 2740 Communications Terminal Model 1 is available only in systems that include MCS. The 2740 Processor routine, IEEC2740, is created at System Generation by the macro instruction SGIHB000. It performs OPEN and CLOSE; READ and WRITE use the following BTAM modules:

- IGG019MA, BTAM read/write routine
- IGG019MB, BTAM channel end and abnormal end appendages
- IGG019M0, BTAM device I/O module (table used by IGG019MA)

Diagram 21 illustrates the logic flow of the 2740 routines.

OPEN is performed when the open pending flag in the UCM entry is on and the UCM entry is not already open. A LOAD is issued to obtain the addresses of the BTAM modules. The address of the 2740 ECB is placed in the UCM entry and Event Indication List. The DEB is initialized from the Communications Task TCB and placed at the head of the TCB DEB queue. The Appendage Vector Table is initialized and the line to the 2740 is initialized via the LOPEN macro instruction. The open flag of the DCB is turned on, the active flag of the UCM entry is turned on, and the open pending flag is turned off.

CLOSE is performed when the close pending flag in the UCM entry is on, the output queue is empty, and the console is not busy. The address of the ECB in the EIL is replaced with the address of the UCM entry, the address of the ECB in the UCM entry is set to 0, the DEB is removed from the TCB DEB queue, and the UCB is set to indicate that it can be allocated. A DELETE macro instruction is issued on the BTAM modules and the active flags in the UCM entry and the control blocks are set to indicate that the device is closed.

READ is performed when the 2740 is not busy and a message has been sent to the terminal or when a READ I/O complete has been successfully processed and the output queue is empty. The BTAM module IGG019MA is given control to perform the function and, if the read is successful, the busy flag is set in the UCM entry.

WRITE is performed when the output pending flag in the UCM entry is set, the 2740 is not busy, and the output queue is not empty. WQE pointers on the console output queue are examined. When a WQE is found that can be written, control is passed to IGG019MA. If the writing operation was successful, the busy flag is set in the UCM entry.

For I/O complete conditions, IEE2740 turns off the busy flag in the UCM entry. If the complete is for a successful READ, the message text is translated from 2740 code to EBCDIC and searched for backspace and cancel codes. If the message is to be ignored, control is passed to IGG019MA for another READ operation. If the message is to be accepted, an SVC 34 instruction causes the message to be passed to the Master Scheduler for the reply processing. Upon return from the Master Scheduler, the processor will attempt to write if the 2740 is not busy and there is output on the console output queue. If the 2740 is busy, control is returned to the Communications Task module IEECMDSV. If the I/O complete condition is for a WRITE, the WQE pointer in the console output queue is marked as no longer needed, and control is returned to IEECMDSV.

If a condition code other than 0 or 4 is returned by a BTAM module, the 2740 processor will retry the I/O operation, or control will be passed to the Console Switch routine IEECMCSW in the Communications Task.

The timer supervision routines provide the processing necessary for the TIME, TTIME, and STIMER macro instructions, which use the System/360 interval timer feature and the System/370 Time-of-Day Clock. The timer second-level interruption handler processes interruptions that occur when the interval timer expires.

TIMER SECOND LEVEL INTERRUPTION HANDLER (IEA0TI00)* CHART HB

*The module used depends upon the timer options specified at system generation and the availability of the Time-of-Day Clock (System/370 models only). Figure 39 lists the timer SLIH module names.

Upon entry, the timer second-level interruption handler (SLIH) tests the timer to ensure that it is negative. If it is positive, control is returned to the timer/external first-level interruption handler (FLIH). The timer SLIH obtains the topmost timer queue element (TQE) from the timer queue and tests the first word of the TQE. Figure 40 lists the processing done by the timer SLIH in response to the flag-field values.

Timer SLIH with System/370 Time-of-Day Clock, Chart HH

In System/370, the timer SLIH determines if the top element in the TQE chain is a synchronized element (greater than one hour interval). For a synchronized TQE, the remaining interval is checked and if still greater than one hour, the new interval is set to one hour and the TQE remains on the TQE queue. If the remaining interval is less than or equal to one hour (and is positive), the TQE synchronize indicator is turned off, the remaining interval is placed in the TQEVAL field, and the TQE is returned to the queue in time-of-expiration order. If the remaining interval is 0 or negative, the interval has expired and normal System/360 processing continues.

Module Name	SMF	TOD	TIMER=
IEA0TI00	No	No	INTVL
IEA0TI02	Yes	No	INTVL
IEA0TI03	No	Yes	INTVL
IEA0TI04	Yes	Yes	INTVL
IEA0TI01	No	No	TIME
IEA0TI01	No	Yes	TIME

Figure 39. Timer SLIH Modules

TIME (IEA0RT00) CHART HA

TIME calculates the time of day in timer units and converts the time to binary or decimal units if requested. If the timer has expired and a twenty-four hour TQE was the topmost element, TIME calculates a new date. The time is returned to the caller in register 0 and the date in register 1.

TIME in System/370 with Time-of-Day Clock Processing, Chart HE

In System/370, the TIME routine verifies the date in the communications vector table each time the routine is entered. The value in the TOD Clock is compared to the expected midnight value. When the current value exceeds the expected value, the CVDATE field is updated and a new expected midnight value is stored in the TQE.

TIME then obtains the time of day value from the TOD Clock. For a request specifying the time in microseconds, the date is returned in register 1, and the time of day is placed in the storage area specified by the requester. If the request was for time of day to be returned in either binary, decimal, or in timer units, the microsecond value is converted to the proper value and the time of day is returned in register 0, the date in register 1.

TTIMER (IEA0ST00) CHART HC

If the TQE of the caller's TCB is not queued to the timer queue, then it has already expired, and a remaining time of 0 is returned to the caller. If the TQE has not expired, TTIMER calculates the remaining time. If the cancel option was specified, TTIMER branches to the timer SLIH to dequeue the TQE from the timer queue. The remaining time is then passed to the caller in register 0, and control is returned through the type-1 EXIT routine.

TTIMER in System/370 with Time-of-Day Clock Processing, Chart HF

In System/370, the Time-of-Day Clock is used by the TTIMER routine to calculate the time remaining in the interval if the synchronous TQE indicator is set.

STIMER (IEA0ST00) CHART HD

Control is immediately returned to the caller if its TCB does not address a timer queue element (TQE). The STIMER routine

Bits 4-7 of TQE	Indicates That:	Elapsed Time Represents:	Action Taken by Timer Second-Level Interruption Handler
0000	TASK parameter was used in STIMER macro instruction.	Time used to perform task for which the STIMER macro instruction was given.	Checks bit 3, which contains a 1 if an asynchronous exit routine is to be entered. If so, passes control to Stage 2 Exit Effector.
[1]010X	Interval that expired was a 6-hour supervisor interval.	Total elapsed time, measured from time that interval was placed into timer.	Checks bit 7, which will contain a 1 if a 24-hour period has passed. If so, increments date by 1.
[1]0100	Interval that expired was a 10-minute supervisor interval.	Total elapsed time measured from time that interval was placed into timer.	Computes system wait time and stores in system management control area. Sets accumulated wait time field to 0.
0110	WAIT parameter was used in STIMER macro instruction.	Total elapsed time, measured from time that interval was placed into timer.	Issues POST macro instruction. (Performance of task for which macro instruction was issued cannot be resumed until POST is given.)
1000	REAL parameter was used in STIMER macro instruction.	Total elapsed time, measured from time that interval was placed into timer.	Checks bit 3, which will contain a 1 if an asynchronous exit routine is to be entered. If so, gives control to Stage 2 Exit Effector.[2]

[1]The address of the 10-minute supervisor element is compared with the address of the top TQE to determine if the interval that expired was a 10-minute or 6-hour supervisor element.
[2]If an Initiator issued the STIMER macro instruction, the TQE will be converted to TASK type, and control will be passed to ABTERM.

Figure 40. Timer SLIH Processing

converts the specified interval to timer units if the decimal or binary options were specified. If the time of day was specified, STIMER calculates an interval that will cause the timer to expire at the specified time of day.

If the TQE addressed by the caller's TCB is active, STIMER uses the timer SLIH to cancel the interval. The TQE is reinitialized to represent the new interval. If an exit address was specified by the caller, a save area is acquired for the exit routine if one was not provided by the caller. The TQE is then placed on the timer queue and timing begins.

If the TQE addressed by the caller's TCB is not active (not currently on the timer queue), a test is made to determine whether the TQE has been flagged as an active IRB. (This is done because an active IRB would indicate that the caller is a timer user-exit routine.) The STIMER macro instruction is effectively a NOP if issued by a timer user-exit routine, and control is returned to the caller. If the TQE has not been flagged as an active IRB, it is initialized to represent the requested interval. If an exit address was specified by the caller, a save area is acquired for the timer user-exit routine if one was not provided by the caller. The TQE is then placed on the timer queue and timing begins.

When timing has begun, STIMER determines whether the caller has requested to wait on the time of expiration. If the caller specified the wait option, STIMER obtains the ECB address and issues a WAIT macro instruction to place the caller in the wait condition pending expiration of the interval.

STIMER in System/370 with Time-of-Day Clock Processing, Chart HG

STIMER in System/370 uses the TOD Clock to calculate the interval and the expected time of expiration of the interval. For REAL or WAIT type requested intervals in excess of one hour, STIMER flags the TQE as

synchronized, stores the expected value of the TOD Clock at expiration in the TQEWORK field of the TQE, and sets the interval to one hour. The TQE is thus checked every hour until the entire requested interval has expired. When the interval becomes less than one hour, the timer second-level interruption handler turns off the synchronizing bit and places the TQE back on the queue as a non-synchronized TQE.

SMF TIME LIMIT EXPIRATION ROUTINE (IEATLEXT) CHART MB

This routine, which resides in the nucleus, provides an interface with a user time limit expiration routine (IEFUTL). It receives control after a job step or wait time limit has expired (see above). It passes control to the user time limit routine, indicating the type of expiration in Register 15. It also passes the address of a 72-byte register save area and the contents of the user data field (TCTUDATA) in the TCT.

The user routine determines whether or not a time extension is allowed. It returns control to the SMF Time Limit Expiration routine with a return code of 0 if the time limit is extended, 4 if it is not extended. If the return code is 4, Register 1 contains the number of timer units allowed for the extension. The SMF Time Limit Expiration routine places the value of the extension into the expired TQE and places the TQE back on the timer queue.

If no extension is granted, the action taken depends on the type of time limit that has expired:

- If wait time expired, the problem program will be abnormally terminated with an error code of 522.
- If job-step time expired, the step will be terminated with a completion code of 322. The SMF Wait Time Expiration routine converts the TQE into an IRB/IQE for standard linkage to the Stage 2 Exit Effector.

RESIDENT OVERLAY SUPERVISOR (IEWSVOVR) CHART IA

The overlay supervisor first determines the cause of the interruption. If it was a SEGLD macro instruction, the overlay supervisor returns control to the caller since SEGLD is a NOP instruction in MFT. If the interruption was caused by a SEGWT macro instruction, the supervisor determines if a direct branch entry to the requested segment, via the caller's entry table (ENTAB), has been prepared because of a previous branch or CALL macro instruction. If the branch or CALL was prepared, the supervisor returns control to the requester, because processing of the request is not needed.

If a branch entry was not prepared, the overlay supervisor performs initialization and issues a LINK macro to request contents supervision assisted linkage to the nonresident module IEWSZOVR. (IEWSZOVR is the name on the linkage library given to either IEWSYOVR or IEWSXOVR. Either of these two modules may be selected during system generation.)

NONRESIDENT OVERLAY SUPERVISOR (IEWSZOVR) CHART IA

IEWSZOVR is the library name for either IEWSYOVR (Basic Synchronous Transient Overlay Supervisor module) or IEWSXOVR (Synchronous Transient Overlay Supervisor module with optional SEGWT checking).

IEWSZOVR processes the request depending on the following conditions:

- If the requester issued a SEGWT (SVC 37) and the requested segment and/or segments in its path are not in main storage, then IEWSZOVR branches to FINCH to bring the needed module into storage. The caller's entry table is not altered to prepare for a branch to the requested segment. Control is returned to the caller only after the requested segment and any segments in its path have been loaded. IEWSZOVR then branches back to the requester. (The requested segment is not entered.)
- If the requested segment or segments in its path are in main storage, IEWSZOVR returns to the caller.
- If the requester issues a branch or CALL and the segment was previously requested by a SEGWT, the overlay supervisor alters the caller's entry table to prepare for a future branch to the same external address without reentering the overlay supervisor.
- If the requester issues a branch or CALL and the segment was not previously requested by a SEGWT, the overlay supervisor branches to FINCH to bring the needed segments into main storage. The caller's entry table is altered to prepare for a future branch to the same external address without entering the overlay supervisor. Control is then given to the requested segment at the specified address.
- If the caller previously issued a CALL or branch instruction specifying the same external address, the overlay supervisor is not entered. The caller's entry table, altered by the previous call or branch, provides a direct branch to the requested segment.

The Checkpoint/Restart routines provide the processing necessary to save the problem program and its environment at a specified point (Checkpoint), and to restore this program and environment when an abnormal termination occurs and return control to the restored program (Restart).

CHKPT (SVC 63) CHART LA

CHKPT:

1. Suspends user I/O requests.
2. Builds a checkpoint entry and writes it in the checkpoint data set.
3. Restores the user I/O requests.
4. Returns to the caller.

If the caller has suppressed checkpoints, through use of the RD parameter in job control statements, no CHKPT entry is written.

The routine consists of 10 load modules which are executed in the SVC transient area after an SVC 63 instruction (CHKPT) macro instruction is issued. When the SVC 63 instruction is executed, an SVC interruption occurs and control passes to the SVC FLIH, the SVC SLIH, and to the first load module of CHKPT (see Figure 41). The remaining load modules receive control via XCTL macro instructions.

Initialization Modules (IGC0006C, IGC0106C, IGC0206C)

The first load module of CHKPT (IGC0006C) determines if checkpoints have been suppressed. If they have or if a subtask is active in the partition, an SVC 3 instruction is issued to pass control to the SVC EXIT routine and return to the caller. If they have not, module IHJACP00 determines if the CANCEL operand was specified in the CHKPT macro instruction being serviced. If CANCEL was specified, processing continues as described below in "CANCEL Processing." If CANCEL was not specified, module IGC0006C issues an OPEN macro instruction for the checkpoint data set (if the caller has not already opened the data set) and then issues a GETMAIN macro instruction for a work area in the dynamic area of main storage. The second load module (IGC0106C) tests the validity of the request. If an error is detected, control passes to checkpoint exit module (IG0Q06C). If no errors are found, the third load module (IGC0206C) reads the job control table (JCT) into the work area, builds the checkpoint header recorder (CHR), and passes control to the check I/O module.

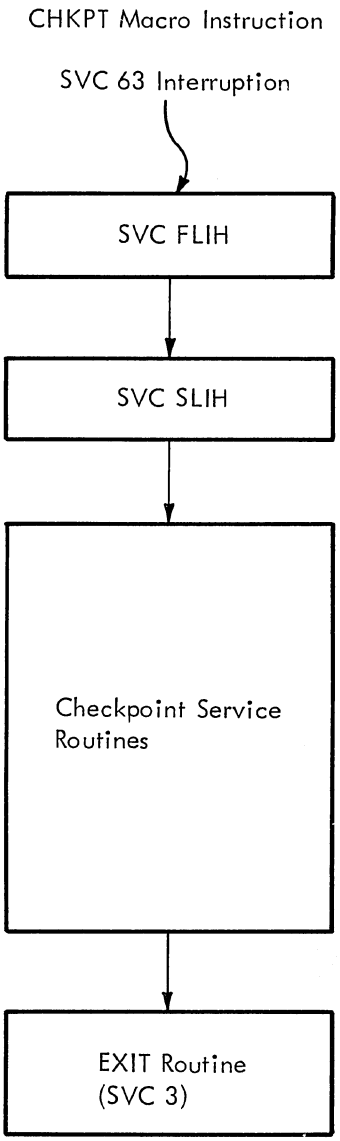


Figure 41. Checkpoint Routine Control Flow

CANCEL Processing

The CANCEL operand of the CHKPT macro instruction indicates that the caller does not want to create a new checkpoint entry, but wants to suppress automatic restarts from any previously created checkpoints.

When CANCEL is specified, module IGC0006C issues a GETMAIN macro instruction to obtain a small work area and then passes control to module IGC0206C. Module IGC0206C reads the job control table (JCT) into the work area and passes control to the exit module, (IGC0Q06C).

The exit module sets a checkpoint indicator to show that no checkpoint entry has been written, and alters the JCT so that it doesn't show the previous checkpoint entries which have been written. The exit module then returns the JCT to the input queue and returns control to the caller via an SVC 3 instruction. (No messages are written to the operator.)

If an abnormal termination occurs after CANCEL processing has been completed, checkpoint/restart is not performed. However, the checkpoint entries that have been written are retained, and the programmer can restart the step from one of these entries at a later time by submitting the proper restart job control language.

Check I/O Module (IGC0506C)

The check I/O module issues a PURGE macro instruction specifying the QUIESCE option for each data extent block (DEB) associated with the caller's task control block. This causes all of the caller's pending I/O requests to be removed from the logical channel queues or, if they are being serviced, to be completed. If a permanent error occurs in a QSAM or QISAM I/O request, an error code is returned to the caller, and no checkpoint is written (unless the QSAM ACC option was specified for the data set). When all of the caller's I/O activity has completed, control passes to the preserve 1 module (IGC0A06C).

Preserve Modules (IGC0A06C, IGC0D06C)

The preserve modules write the checkpoint header record (CHR) created by the third module, then build and write a data set descriptor record (DSDR) for each job file control block, job file control block extension, and generation data group bias count table. If end-of-volume occurs for the checkpoint data set on tape, IGC0206C is called to attempt to rewrite with a new tape. If end-of-volume occurs for the second time on tape or the checkpoint data set is on a direct access device and end-of-volume is detected or an I/O error occurs in either module, control is transferred via XCTL to the resume I/O module. If none of the above errors occurs control then passes to the checkmain module.

Checkmain Module (IGC0F06C)

The checkmain module writes the contents of problem-program main storage onto core image records (CIRs). Then a supervisor record (SUR) is constructed with task control information and written as the last record in the checkpoint entry. Control then passes to the resume I/O module via an XCTL. If an I/O error occurs or end-of-volume is detected on either tape for the second time or on a direct access device, control is passed to the resume I/O module with an error code. If the end-of-volume occurs for the first time on tape, control is passed to IGC0206C to reprocess the tape.

Resume I/O Module (IGC0N06C)

The resume I/O module issues the RESTORE macro instruction for each data extent block (DEB) associated with a previously suspended I/O request. The requests are restored to the logical channel queues, and if possible, started. Control is then passed to exit module, (IGC0Q06C).

Exit Module (IGC0Q06C)

For a normal exit, IHJACP50 issues a STOW macro instruction if the checkpoint data set has a partitioned organization. It then issues a CLOSE macro instruction for the checkpoint data set (unless the caller issued the OPEN), updates the checkpoint flags and count fields in the job control table (JCT), restores the JCT to the job queue, and frees the work area. For an exit after an error has occurred, the preceding functions are performed if necessary. Control is then passed to the message module.

Message Module (IGC0S06C)

The message module writes a message indicating successful or unsuccessful completion. One of the following return codes is placed in Register 15 before control is returned to the caller via an SVC 3 instruction:

X'00' Valid checkpoint entry written.

X'08' No checkpoint written; calling error.

X'0C' Permanent I/O error.

X'10' A valid checkpoint entry was written, but there were outstanding ENQs. It is the responsibility of the user to restore these ENQs during restart.

RESTART (SVC 52) CHART LB

RESTART uses information in a checkpoint entry to recreate the conditions that existed when the checkpoint entry was written. RESTART:

1. Restores the problem program to its original location in main storage.

2. Opens and positions any problem program data sets that were open when the checkpoint entry was written.

3. Restores task control information.

4. Passes control to the problem program instruction immediately following the CHKPT macro instruction from which RESTART is occurring.

The routine consists of 14 load modules which are executed in the SVC transient area after an SVC 52 instruction is issued. Before the SVC 52 instruction is issued, a job management routine (IEFDSDRP) adjusts the job queue and ensures that device allocations are compatible with those which were in effect when the CHKPT macro instruction was issued. Just before exiting, IEFDSDRP changes the name of the restarting step to IEFRSTRT. This program consists of only an SVC 52 instruction.

When the SVC 52 instruction is executed, an SVC interruption occurs and control is passed to the SVC FLIH, the SVC SLIH, and to first load module of the RESTART (see Figure 42).

Initialization Modules (IGC0105B, IGC0205B)

The first load module of RESTART (IGC0105B) receives the address of a parameter list built by job management routines from information in the checkpoint header record (CHR). From this parameter list, RESTART determines what the problem program's boundaries were when the checkpoint entry was written. It then issues a GETMAIN macro instruction for the same area (this includes a RESTART work area). A data control block (DCB) for the checkpoint data set is constructed in the work area, and the RESTART SVRB and the current task input/output table (TIOT) are moved into the area. An OPEN macro instruction is issued for the checkpoint data set, and the next module is called.

The second load module (IGC0205B) moves additional checkpoint data set control blocks into the work area. It positions the checkpoint data set at the first core image record (CIR) and calls the REPMAIN.

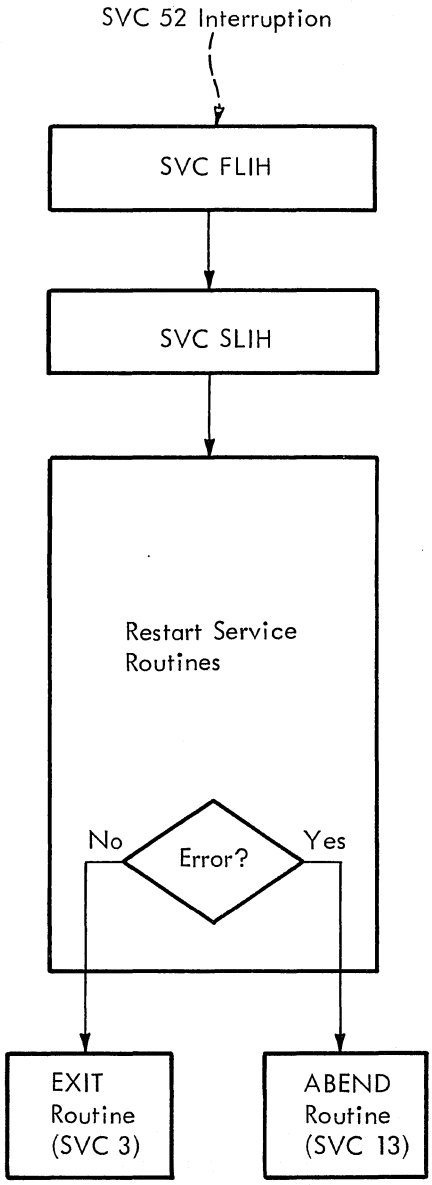


Figure 42. Restart Routine Control Flow

REPMAIN Modules (IGC0505B, IGC0605B)

The REPMAIN module IGC0505B reads the core image records (CIR) into problem program storage and the supervisor record (SUR) into the work area from the checkpoint data set. If the size of the partition has been increased, the free area queue element is adjusted accordingly, and the address of the FQE is moved into the boundary box. If the system includes the protection feature, each 2K block of main storage within the partition is set to the specified protection key.

The REPMAIN module IGC0605B restores the TCB fields and any floating point registers

that were saved by the CHECKMAIN module in the supervisor record (SUR). If subtasking is included in the system, the pointer to the job pack area queue is restored in the program information block (PIB). The protection key in the TCB is stored in all request blocks (except supervisor request blocks) that are queued to the TCB. The protection key and the TCB address are also stored in all data extent blocks (DEB) queued to the TCB. Control is then passed to the first job file control block processing module (IGC0G05B).

Job File Control Block Processing Modules (IGC0G05B, IGC0G95B, IGC0I05B)

The first two job file control block (JFCB) processing modules (IGC0G05B and IGC095B) create a table in the work area for each JFCB associated with a data set that was open when the CHKPT macro instruction was issued. A DEB, a DCB, an IOB, and an ECB are constructed within the table for later repositioning I/O operations. Control then passes to the second JFCB processing module.

The third JFCB processing module (IGC-0I05B) reads in a JFCB extension for sequential access method (SAM) data sets that reside on more than five volumes. Several extensions may be read until the one containing the volume serial number that was in use when the CHKPT macro instruction was issued is found. Unless all problem program data sets reside on direct access storage devices, control passes to the first mount/verify module. If all data sets are on direct access storage, control passes to the direct access mount/verify module.

The Dummy Data Set Processor (IGC0H05B) receives control from IGC0I05B if any of the data set description tables correspond to a dummy data set. (For a discussion of dummy data sets, refer to Job Control Reference.)

For each dummy data set encountered, a check is made to determine if the data set was a dummy data set when the checkpoint was made. If it was, no further processing is done.

For each dummy data set that was not a dummy data set when the checkpoint was taken, IGC0H05B deletes the subroutine loaded by the OPEN macro instruction, and unchains and frees the DEB associated with the data set. If the data set was being processed by QSAM or BSAM and the checkpoint was not made during an end-of-volume exit for the data set, IGC0H05B frees the IOBs created by the OPEN macro instruction, creates a dummy DEB and adds it to the DEB chain, loads the dummy access method (IGC0-19AV), and sets pointers to it in the DCB associated with the data set.

An error condition exists if a data set that has been made a dummy is not being processed by either QSAM or BSAM at the time the checkpoint was made, or if the checkpoint was made during an end-of-volume exit. When an error occurs, an error code of 79 is placed in the restart work area.

If no errors are detected, control is passed to the mount verification portion of Restart once all of the table entries have been processed.

If an error has been detected, control is passed to the Restart Error routine (IGC0V05B).

TCAM Data Set Processor (IGC0J05B)

If there are any TCAM data sets to be restarted, the TCAM data set processor receives control from IGC0G95B. It initializes the TCAM region and sets up TCAM control blocks and queues for each data set. Control is given to the Restart Error routine (IGC0V05B) if the data set's QNAME parameter is not known to TCAM, if the process entry defined by QNAME is already being used, or if a GETMAIN by TCAM was unsuccessful. If IGC0J05B executes normally, control is passed to the last JFCB Processor (IGC0I05B).

Mount/Verify Modules (IGC0K05B, IGC0M05B)

The first mount/verify module (IGC0K05B) processes all data sets except those residing on direct access devices. For SYSIN, SYSOUT, unit record, and graphics data sets, processing consists only of adjusting the data extent block (DEB).

For magnetic tape data sets, the volume serial number in the primary unit control block (UCB) in the data set's task input/output table (TIOT) entry is compared to the volume serial number in the work-area table built from the data set descriptor record (DSDR). If they are the same, the necessary adjustments are made to the UCB and the DEB. If the volume serial numbers do not match, the secondary UCBs (if any) are searched. If the correct volume is specified in one of them, it becomes the primary UCB. If the volume is not mounted, a suitable UCB is selected from the TIOT entry, and a MOUNT message is written to the operator. For any tapes with nonstandard labels, a user-supplied verification subroutine is called. After all tape data sets are processed, control is passed to the direct access mount/verify module.

The direct access mount/verify module (IGC0M05B) performs exactly the same func-

tions as the first mount/verify module, except that no label checking is done, and all volumes of a concatenated data set with partitioned or direct access organization are mounted. An error, such as no suitable UCB for a volume, causes RESTART to terminate with an error message. If no error occurs, control is passed to the first direct access position I/O module (IGC0N05B).

SYSIN/SYSOUT Nondirect Access Data Set Processor (IGC0L05B)

This module receives control from IGC0K05B or IGC0M05B for certain SYSIN or SYSOUT data sets. It primes the buffers for a SYSIN data set from the card reader or writes header labels on a SYSOUT tape data set with deferred restart. If an ASCII label is used on a SYSOUT tape or an error occurs in writing the SYSOUT header labels, control is given to the Restart Exit routine (IGC0V05B). Otherwise control goes to IGC0N05B (for SYSIN/SYSOUT data sets on direct access) or to IGC0P05B.

Position I/O Modules (IGC0N05B, IGC0Q05B, IGC0P05B, IGC0R05B, IGC0S05B)

The first direct access position I/O module (IGC0N05B) adjusts the DCB, DEB, and channel programs for direct access data sets that have been deleted. When processing is completed, control is passed to the second direct access position I/O module (IGC0Q05B). This module calculates the number of tracks in each extent in each SYSIN or SYSOUT DEB and places the number in the DEB. It passes control to nondirect access position I/O module IGC0P05B or to direct access position I/O module IGC0R05B.

The nondirect access position I/O module moves magnetic tape data sets to where they were located when the CHKPT macro instruction was issued. IGC0P05B assumes the following:

- System input data sets have been positioned by the scheduler to the first data record of the user's input stream.
- Nonstandard label data sets have been positioned by the user label routine to the first data reocrd on the current volume.
- If this is a deferred restart, system output data sets have been positioned by module IGC0L05B. (If this is an automatic restart, system output data sets will be rewound and positioned now.)
- All other tape data sets are positioned at load point.

IGC0P05B positions tapes with standard labels or no labels to the first data record. Then, using the BLKCT field of the DCB, IGC0P05B advances each tape data set to where it was located when the CHKPT macro instruction was issued. If the BLKCT field is negative or 0, the data set is positioned to the beginning or end, depending on whether forward or backward processing was in progress when the CHKPT macro instruction was issued. Control then passes to the final processing module or to IGC0S05B.

The Data Set Processor 1A (IGC0S05B) module works from the data set description table, processing all magnetic tape data sets except those tapes created under DOS which contain either embedded checkpoint records or possibly a leading tapemark. On entry, all but two types of tape volumes are positioned at the load point. The exceptions are SYSIN data sets, which are positioned to read the first user input record, and nonstandard labeled tapes, which are positioned at the first data record by the user label routine.

Data Set Processor 1A first advances the tape past the label, if necessary, to the correct data set, using the file sequence number. Then the DCB block count field (DCBBLKCT), saved at CHKPT, is used to advance the data set to the correct record. If the DCBBLKCT field is zero or negative, the data set is positioned at the first record of end-of-file, depending on whether the forward or backward processing was taking place at CHKPT time. An I/O error in repositioning causes Restart termination.

IGC0S05B passes control to IGC0U05B if any data sets are open at CHKPT time for which either a leading tapemark or embedded DOS checkpoint record is indicated. Positioning of these tapes occurs similarly as in IGC0S05B with two exceptions:

1. For the indication of a leading tapemark, a READ is issued and the resulting status is tested for indication of a unit exception. If a tapemark is not sensed, the tape is repositioned to the load point prior to performing record positioning.
2. To perform record positioning of data sets containing DOS embedded checkpoint records requires the reading of the first 20 bytes of every block. During record positioning, the embedded checkpoint records encountered are spaced over and are not reflected in the block count.

The Data Set Processor 2 module is called from IGC0S05B except when the above described DOS tape volumes require posi-

tioning. If either of these tape volumes requires positioning, the Data Set Processor 2 module is called from IGC0U05B. If there are no direct access data sets, the Access Method Disposition module (IGC0T05B) is given control.

The Data Set Processor 2 module (IGC0R05B) checks each data set residing on a direct access storage device to determine if the space allocation limits of the data set (described in the data set control block (DSCB) on the volume) have changed since the CHKPT macro instruction was issued. The limits which existed at that time are described in the data extent block (DEB) saved by CHKPT. If the space allocation limits of an input data set have changed (indicating that the data set has been modified), RESTART is terminated.

If the space allocation limits of an output data set have changed, the smaller of the two space allocations is placed in both the DSCB and the DEB. If the DSCB allocation is reduced, the partial release module of the CLOSE routine is called to return the released space to the free area. When all direct access data sets have been checked, control passes to the final processing module.

ISAM and BDAM Data Set Procesor (IGC0W05B)

If there are any ISAM or BDAM data sets to be restarted, IGC0W05B receives control from IGC0R05B. For ISAM data sets, it reads the Format 2 DSCB and then gives control to ISAM OPEN (IGC01920 or IGC01950) to validate certain pointers in the DSCB. If an I/O error occurs in the ISAM OPEN module, IGC0W05B sets a return code and gives control to the Restart Exit routine (IGC0V05V). For BDAM data sets, IGC0W05B reinitializes addresses within reentrant BDAM access method modules. IGC0W05B then passes control to the I/O Restart routine (IGC0T05B).

Final Processing Module (IGC0T05B)

The final processing module reads the directories of any user output data sets with partitioned organization to detect members added after the CHKPT macro instruction was issued. If any are found, they are deleted with the STOW macro instruction.

Finally, the RESTORE macro instruction is used to reschedule any user I/O requests suspended by PURGE during execution of CHKPT. Control is then passed to the exit module.

Exit Module (IGC0V05B)

The exit module first tests an error code field in the work area to determine if entry is for an error termination. If an error code is found, message IHJ007I is issued. The exit module then issues an ABEND macro instruction to abnormally terminate the task.

If no error has occurred, the exit module compares the sizes of the old task input/output table (TIOT) and current TIOT (which was saved in the RESTART work area). If the current TIOT is smaller or equal to the old TIOT, it overlays the old TIOT, and the RESTART work area is freed. If the current TIOT is larger, it is moved to the end of the work area, and both the remainder of the work area and the old TIOT are freed. The exit module writes message IHJ008I to inform the operator that the job is being restarted. It then loads a completion code of X'04' into Register 15 to inform the problem program that it is being restarted, and issues an SVC 3 instruction to pass control to the problem program. If the program again abnormally terminates (and RESTART has not been deferred), RESTART will again be attempted.

SER ROUTINES

SER0 is the less complex version of system environment recording. It determines the type of malfunction and, if possible, writes a record on the SYS1.LOGREC data set describing the error. SYS1.LOGREC is located on the primary system residence volume. If SER0 cannot write the record, the computer is placed in a wait state and a message is printed requesting that the operator use system environment recording, edit, and print (SEREP). If SER0 can write a partial or complete record, the computer is placed in a wait state, and a message is printed requesting that the operator reload the operating system.

SER1 is the more complex version of system environment recording. It also collects and writes machine environment data, but in addition it attempts to associate the malfunction with the task being executed. If the malfunction can be associated with the task and if the control program has not been damaged, the task is abnormally terminated. If not, the computer is placed in a wait state.

A message is issued to the operator when the SYS1.LOGREC data set is 90% full. The operator should then run the environment recording edit and print (IFCEREP0) service aid. IFCEREP0 formats the SYS1.LOGREC records, and then writes the records onto printer, tape, or disk according to user specifications. IFCEREP0 is described in the Service Aids Logic, Program Logic Manual. If the operator delays in "emptying" the SYS1.LOGREC data set, it may become full; when it does, a message will be issued to the operator. He must then run IFCEREP0 immediately or system performance may be degraded.

SER0 CHART AG

SER0 collects, formats, and writes error information after a machine-check or a channel error has occurred. It is divided into two modules: IFBSR000, resident in the nucleus, and IFBSR0xx (where xx is the model number: 40, 50, 65, or 75), located on the link library. This module is model dependent. The required modules are selected during system generation.

Resident Module -- IFBSR000

Module IFBSR000 is nonreusable and does not require operating system facilities. It halts all I/O activity and then reads the first text record of module IFBSR0xx into main storage (beginning 32 bytes past the end of the nucleus).

Module IFBSR000 saves information (in a 22-byte field in lower storage) to be used later by IFBSR0xx. After it has halted I/O activity on all devices, IFBSR000 attempts to read the first 1024 bytes of module IFBSR0xx into main storage. If after ten retries, these 1024 bytes have not been read into main storage, IFBSR000 builds IOS wait state code 000F0A and then branches to the bell ring/wait state module which sounds the console alarm and places the computer in a wait state. Wait state code 000F0A is displayed in the instruction counter.

Link Library Module -- IFBSR0xx

Like IFBSR000, module IFBSR0xx does not require operating system facilities. IFBSR0xx first loads the remainder of itself into main storage. It then checks location 50 to determine which type of malfunction has occurred, a machine-check error or a channel error. Location 50 is preassembled to X'FF'. If the error is a machine-check error, location 50 will have been overlayed by the machine-check old PSW. If the error is a channel error, location 50 remains unchanged.

If the error is a machine-check error, IFBSR0xx builds a machine-check record entry in which to place information about the error. If the error is a channel error, IFBSR0xx builds a channel error record entry.

Module IFBSR0xx then enables machine-check interruptions. General registers are checked for valid parity on all models except Model 40. Parity indicators are available for all registers except 13, 14, and 15 on Models 50 and 75. Floating point registers are also checked for valid parity if the model is equipped with floating point.

Module IFBSR0xx checks the "busy bit" in each unit control block (UCB) to determine which I/O units were busy when the error occurred. The addresses of as many as ten busy I/O devices are collected. IFBSR0xx then builds a record containing the program identification, job name, step name, day, and time. After examining the seek address obtained from the header record of the SYS1.LOGREC data set, IFBSR0xx writes (on

that data set) the record it has just created and an end-of-file record.

Module IFBSR0xx then prints the following message to the operator:

IFBF05W MACHINE ERROR. RELOAD OS/360

This message indicates that a complete error record has been written on SYS1.LOGREC. If a message cannot be printed, IFBSR0xx builds IOS display code 000F05 and branches to the bell ring/wait state module.

If another machine-check interruption occurs while IFBSR0xx is collecting data for an error record, IFBSR0xx stops collecting data and attempts to write a partial error record on SYS1.LOGREC containing the data it has already collected. If it is able to do this, it prints the following message:

IFBF06W MACHINE ERROR. RELOAD OS/360

If a message cannot be printed, IFBSR0xx builds IOS display code 000F06 and branches to the bell ring/wait state module.

If another machine-check interruption occurs while IFBSR0xx is attempting to write a partial error record, IFBSR0xx cannot continue processing. It prints the following message:

IFBF07S MACHINE ERROR. EXECUTE SEREP

Other errors besides a machine-check interruption may prevent IFBSR0xx from writing an error reocrd on SYS1.LOGREC. These errors (with the messages IFBSR0xx prints to the operator) are as follows:

1. An I/O error,

   IFBF08S MACHINE ERROR. EXECUTE SEREP

2. SYS1.LOGREC data set is full,

   IFBF09S MACHINE ERROR. EXECUTE SEREP

3. Module IFBSR0xx could not be loaded into main storage,

   IFBF0AS MACHINE ERROR. EXECUTE SEREP

SER1 CHART AH

Like SER0, SER1 collects, formats, and writes error information after a machine-check or a channel error has occurred. Unlike SER0, it is a single, serially reusable module that resides in the nucleus.

In addition to writing error records, SER1 attempts to associate the error with the task that was executing. If it can do this, and if the control program is not damaged, SER1 abnormally terminates the task. The system continues to operate.

If SER1 cannot write a complete error record or cannot associate the error with the task, or if the error damaged the control program, the computer is placed in a wait state. The system must then be reloaded.

SER1 checks location 50 to determine which type of malfunction occurred, a machine-check error or a channel error. Location 50 is preassembled to X'FF'. If the error is a machine-check error, location 50 will have been overlayed by the machine-check old PSW. If the error is a channel error, location 50 remains unchanged.

SER1 gathers error data into either a machine-check record or a channel error record and writes the record on SYS1.LOGREC. SER1 uses I/O routines provided by the operating system (it uses the EXCP macro instruction to communicate with the SYS1.LOGREC data set) unless the control program was damaged by the error. If the control program was damaged, SER1 uses its own I/O routines. The DEB and DCB required when EXCP is used reside in the nucleus and are opened by the nucleus initialization program (NIP).

If SER1 can associate the error with the task and if the control program is not damaged, SER1 terminates the task by branching to the ABTERM routine. When SER1 regains control from ABTERM, it reinitializes itself and branches to the dispatcher so that the system can continue to operate.

In order for the system to continue operating:

- Another error cannot occur while SER1 is collecting data for a previous error. If one does, SER1 stops collecting data and attempts to write a partial record of the original error on SYS1.LOGREC. The partial record contains the data collected before the second error occurred.
- SER1 must be able to associate the error with the task that was executing.
- The control program cannot be damaged by the error.

If the system cannot continue operating, SER1 prints a message on the primary output device instructing the operator to reload the operating system. SER1 then places the system in a wait state.

ENVIRONMENT RECORDING AREA

SYS1.LOGREC is a data set on the system residence device used exclusively by SER0, SER1, and all preservation recording systems. It is formatted during system generation by utility program IFCDIP00. The data placed in SYS1.LOGREC is edited and printed by the IFCEREP0 (EREP) service aid. These programs are described in the Service Aids Logic, Program Logic Manual.

SYS1.LOGREC contains three types of records:

1. Header Record - This is the first record in the data set. It defines the extent of the data set, and addresses the last record written. It also contains a safety byte used to detect overrun. The record is 38 bytes long. The format of the Header Record is described in the Input/Output Supervisor, Program Logic Manual.

2. Record Entry Area - This area begins immediately after the Header Record. It contains machine check records written by SER0, SER1, or the Machine Check Handler (MCH); channel check records written by SER0, SER1, or the Channel Check Handler (CCH); and unit check records written by the Outboard Recording routine (OBR) or the Miscellaneous Data Recording routine. The formats of the unit check (outboard and miscellaneous data) records and the channel check (inboard) records are described in the I/O Supervisor, Program Logic Manual. The machine check record format is contained in Section 5 of this publication.

This section contains a description of each of the Display Console Support routines. These descriptions provide a general explanation of each routine and references the applicable flowcharts.

DIDOCS PROCESSOR ROUTINES

The DIDOCS Processor routines (IGC5107B, IGC5Z07B, IGC6107B, and IGC6Z07B) receive control from MCS when an operator or system request is entered that requires processing by the DIDOCS routines. The following requests are processed by DIDOCS:

- Open/Close request
- Attention
- I/O complete
- DOM request
- Hardcopy inoperative
- Hardcopy operative again
- Messages to be displayed
- Timer interruption
- Roll needed

DIDOCS PROCESSOR 0, LOAD 1 (IGC6107B) CHART KA

Procesor 0, Load 1 receives control from MCS to begin processing a request involving a display console with a transient DCM. Load 1 queues the request and then determines if the transient portion of the DCM is required to process the request. If required, and if it is not already in main storage, Processor 0 brings it into main storage. Control is then given to the Processor 1 routine (IGC5107B).

Processor 0, Load 1 also checks for a request for a permanent PFK update (permanent copies of PFK definitions are maintained on SYS1.DCMLIB; the operator may make permanent changes to the definitions). If a request for a permanent change is encountered, Processor 0, Load 1 builds the appropriate channel program and issues an EXCP macro instruction to initiate the change. When I/O is complete for a PFK update, Processor 0, Load 1 gives control to Processor 0, Load 2 to check for successful I/O.

When Processor 0 is entered because an I/O operation is complete, control is given to Processor 0, Load 2 to determine if an I/O error occurred or if additional I/O processing is required.

When Processor 0, Load 1 is entered from Load 2 (indicating that I/O is satisfacotrily completed), control is passed to Processor 1.

DIDOCS PROCESSOR 0, LOAD 2 (IGC6Z07B) CHART KB

Processor 0, Load 2 receives control from Processor 0, Load 1 when an I/O operation is signaled complete. Load 2 determines if any additional I/O processing is required and if any I/O errors occurred. Action is taken as follows:

- I/O Error -- Writes an error message to the operator's console and passes control back to MCS with an indication that a console switch is required.
- Additional I/O necessary -- Updates and executes the channel program.
- I/O Complete -- Passes control to Processor 0, Load 1.

DIDOCS PROCESSOR 1, LOAD 1 (IGC5107B) CHART JA

DIDOCS Processor 1, Load 1 receives control be begin processing an operator or system request. Control is received either directly from MCS (when the console involved in the request does not have a transient DCM) or from the Processor 0 routine (when the console has a transient DCM).

Processor 1, Load 1 determines the reason for entry and passes control to the appropriate display console support routine. Figure 43 summarizes the major reasons for entry to the DIDOCS Processor routine, the resulting exit, and the reason for the exit. In addition, the module functions as a controller for DIDOCS, prioritizing functions and returning control for functions which are not yet complete.

When DIDOCS has nothing to do for a particular console, or the device is busy, Processor 1, Load 1, returns to MCS (BR 14), possibly through Processor 0 if the DCM is transient.

Reason for Entry	Exit	Reason for Exit
Open/Close request	Open/Close Routine	Open/Close I/O blocks
Reopen request	I/O Routine	Initialize DCM
I/O error	Asynchronous Error Routine	Handle I/O error
DOM issued	Delete 2 Routine	Mark messages for deletion
Hardcopy down/ recovered	Message 1 Routine	Provide/remove no hard copy message
Messages to be displayed	Display 1 Routine	Display messages
Timer interruption	Timer Interpreter Routine	Analyze timer interval
Roll needed	Roll Mode Routine	Perform roll
Read complete	Command Routine	Handle command read-in
Attention	I/O Routine	Read Manual Input to determine type
Cancel	Command Routine	Restore entry area
PFK attention	PFK 1 Routine	Process commands
Enter	I/O Routine/Command Routine	Read command if necessary
Light pen, or cursor not in entry area	Light Pen/Cursor Routine	Interpret desired function

Figure 43. DIDOCS Processor Routine Entries

DIDOCS PROCESSOR 1, LOAD 2 (IGC5Z07B) CHART JB

Processor 1, Load 2 receives control from Processor 1, Load 1 for processing close requests and for continued processing of request parameter lists.

If entry is for processing a close request, Processor 1, Load 2 determines whether the screen has been erased. If it has not, flags are set in the DCM and control is passed to the device-dependent I/O routine to erase the screen. If the screen has been erased, control is passed to the Open/Close routine to complete processing of the close request.

If entry is for processing of a request parameter list, the type of request is determined and control is passed to the appropriate display console support routine (this process is a continuation of the request handling process begun in Processor 1, Load 1). If the request in the parameter list has already been processed, control is returned to Processor 1, Load 1.

OPEN/CLOSE ROUTINE (IGC5G07B) CHART KK

The Open/Close routine determines whether to open or close a device by checking a parameter passed to it in the communications task extended save area (CXSA).

After opening a DCB, control is returned either to the DIDOCS Processor, if a console exists, or to the MCS Console Switch routine, if there is no console. After closing a DCB, control is returned to the Cleanup routine.

2250 I/O 1 ROUTINE (IGC5P07B) CHART JC

The 2250 I/O 1 routine performs requested I/O operations. It checks the communication bytes in the DCM (I/O communication bytes 1, 2, and 3 and message communication byte 1) to determine the type and format of I/O requested.

Each I/O request is checked and the appropriate CCWs are built until all the I/O requests are set up in the channel program. This routine builds CCWs for the following I/O requests:

- Read manual input (RMI)
- Write 'message waiting' message
- Write message area
- Write instruction line
- Sound alarm
- Write entry area
- Write warning line
- Insert cursor

If a light pen interruption occurs, control is passed to the Light Pen/Cursor routine (IGC5F07B). If one of the following I/O operations is requested, control is passed to the 2250 I/O 2 routine:

- Write asynchronous error message
- Write message; message waiting
- Write status display
- Erase screen
- Sound alarm

For all other cases, control passes to the DIDOCS Processor (IGC5107B).

2250 I/O 2 ROUTINE (IGC5Q07B) CHART JD

The 2250 I/O 2 routine is called by the 2250 I/O 1 routine to process one or more of the following I/O requests by building the required CCWs:

- Write asynchronous error message
- Write status display
- Erase screen
- Read entry area
- Write PFK lines
- Light PFK keys

This routine also initializes the instruction line, and blanks the entry area, and warning line as requested prior to performing I/O in these areas.

This routine also picks up the channel program address from DCMDSAV. The exit from this routine is to the DIDOCS Processor (IGC5107B).

2260 I/O 1 ROUTINE (IGC5R07B) CHART JE

The module first tests for a status switch request. If this is indicated, control is passed to the 2260 I/O 2 routine (IGC6R07B). The 2260 I/O routine then checks bits set in the DCM to determine which I/O operation is to be performed. One of three possible I/O operations may be requested: (1) read manual input (RMI) to find cursor, (2) write screen, or (3) read entry area. This routine normally exits to the DIDOCS Processor routine (IGC5107B) unless the cursor is located adjacent to the start of message (SOM) symbol and the screen is in "hold" mode. In the latter case, the CANCEL function is to be performed and the 2260 I/O routine exits to the Command routine (IGC5407B). If the cursor is not located in the entry area, the line and character positioning is computed and exit is to the light pen/cursor routine (IGC5F07B).

2260 I/O 2 ROUTINE (IGC6R07B) CHART JF

This routine is entered from 2260 I/O routine 1 when a change in a 2260 console's operating mode is requested by the operator or required by the system. IGC6R07B is called twice during each console switch. On the first pass, the routine determines the new console operating mode (full capability or output only) and passes control to the DIDOCS Cleanup routine (IEECVFTG) to continue processing of the console mode switch.

When IGC6R07B receives control for the second time during a console switch, the routine determines if "message stream" mode has been requested. If so, the routine sets a "roll delete" mode indicator in the DCM (this places the console in "roll-deletable" mode for message deletion) and exits to the Timer/Interpreter routine (IEECVETK) to continue processing of the console mode switch. If message stream mode was not requested, control is returned to 2260 I/O 1 (IEECVETR) to complete processing of the console mode switch request.

MODEL 85 I/O ROUTINE (IGC5H07B) CHART JG

The module first tests for a status switch request. If this is indicated, control is passed to the Cleanup routine (IGC6G07B). The Model 85 I/O routine then checks the communication bytes in the DCM (I/O communication bytes 1, 2, and 3 and message communication byte 1) against preestablished bit settings to determine the type and format of I/O requested.

Each I/O request is checked and the appropriate CCWs are built until all the

I/O requests are set up in the channel program. This routine builds CCWs for the following I/O requests:

- Read manual input (RMI)
- Read entry area
- Write message area
- Write asynchronous error message
- Write message waiting message
- Write status display
- Write instruction line
- Write entry area
- Write warning line
- Insert cursor
- Blank warning line
- Blank instruction line
- Blank entry area
- Erase screen
- Sound alarm

Control is returned to the DIDOCS Processor (IGC5107B).

ASYNCHRONOUS ERROR ROUTINE (IGC5C07B) CHART KC

The Asynchronous Error routine handles asynchronous errors and reopen conditions. For permanent synchronous or asynchronous errors, this routine performs console switching by exiting to the MCS Console Switch routine (IGCXL07B).

When an asynchronous error occurs, the Asynchronous Error routine sets indicators in the DCM to erase the screen and display an error message. These indicators are set for the Message 2 routine (IGC5E07B), which moves the appropriate error message into the DCM, and for the appropriate device I/O routine which erases the screen and writes the error message. Control is passed to the Message 2 routine (IGC5E07B).

When a reopen condition is met, this routine initializes the DCM and passes control to the appropriate device I/O routine to write the screen.

MESSAGE 1 ROUTINE (IGC5D07B) CHART JH

The Message 1 routine contains messages used by Display Console Support. It tests bit settings in the DCM to determine which message to move into the screen-image buffer while distinguishing between error and warning messages. This routine then sets indicators in the DCM to write the screen. Message 1 sets the sound alarm bit unless an UNVIEWABLE MESSAGE or DELETION REQUESTED warning message is written. If the bit settings in the DCM indicate that a message is to be written, the Message routine moves the message into the DCM and exits to the appropriate device I/O. Otherwise, control passes to the DIDOCS Processor (IGC5107B).

MESSAGE 2 ROUTINE (IGC5E07B) CHART JI

The Message 2 routine contains the asynchronous error and deletion request error messages used by Display Console Support. It tests bit settings in the DCM to determine which messages to move into the screen image buffer. This routine then sets indicators in the DCM to write the screen, sets the sound alarm bit, and exits to the appropriate device I/O routine.

MESSAGE ROUTINE 3 (IGC6D07B) CHART JJ

The Message 3 routine contains the PFK support error messages used by Display Console Support. Message 3 tests bit settings in the DCM to determine which messages to move into the screen image buffer (in the DCM). It then sets indicators in the DCM to write the screen, sets the sound alarm bit, and exits to the appropriate device I/O routine.

DISPLAY 1 ROUTINE (IGC5207B) CHART JK

Display 1 displays all operating system messages except those to be deleted (unless DEL=N) and status display messages. It searches the output queue for write queue elements (WQEs) to be displayed. If this search is successful, control is passed to Display 3. Upon return from Display 3, Display 1 exits to the appropriate device I/O routine.

If a status display is overlaying messages on the screen, Display 1 exits to either Delete 2 (IGC5707B), in case an intervention required action message is on the screen, or Delete 4 (IGC5907B), if there are no action messages and automatic deletion is specified and not yet tried. If automatic deletion is not specified or has been tried, this routine exits to either Message 1 (IGC5D07B) to display an UNVIEWABLE MESSAGE or the appropriate

device I/O routine to display a MESSAGE WAITING. If roll mode is specified, Display 1 exits to Display 2 (IGC5307B) to set the timer as required. If a message is too long or an accepted reply is on the screen, Display 1 exits to Display 2 (IGC5307B). When no messages are added to the DCM, Display 1 returns control to MCS via a branch on register 14.

DISPLAY 2 ROUTINE (IGC5307B) CHART JL

The Display 2 routine splits messages longer than 72 characters (70 for the 2250), or marks all accepted replies and associated WTORs as automatically deletable in the screen-image buffer. In the latter case, Display 2 sets the timer as required. After line splitting, Display 2 exits to Display 1 (IGC5207B). It exits to one of the following routines if marking accepted replies is performed:

- Delete 2 routine (IGC5707B) to delete intervention required action messages.
- Delete 4 routine (IGC5907B) to perform automatic deletion.
- Device-dependent I/O routine to perform input/output.
- Message 1 routine (IGC5D07B) to write UNVIEWABLE MESSAGE warning.
- DIDOCS processor (IGC5107B) to continue processing.

DISPLAY 3 ROUTINE (IGC6207B) CHART JM

Display 3 uses the WQEs found by Display 1 to place the message(s) into the next available line in the screen image buffer of the DCM. It marks each message according to its descriptor code as provided by MCS and returns control to DISPLAY 1.

ROLL MODE ROUTINE (IGC5J07B) CHART JN

The Roll Mode routine performs the roll function when messages are on the WQE and the time specified for roll expires. When messages on a display console are ready to be rolled, the routine rolls the messages in the screen image buffer. If the CONTROL command operand DEL is set to roll deletable (RD) mode, the specified number of deletable messages are removed. If DEL is set to roll (R) mode, the specified number of messages, including action messages, are removed. It also stores the information for inserting the number of message lines remaining on the output queue into the first two character positions of the first new message to be displayed following a roll. Control is passed to the Timer Interpreter (IGC5K07B) unless DEL=RD and no deletable messages are on the visible portion of the screen; in that case, control is passed to the appropriate device I/O routine to write the message-waiting message.

The RNUM value specified in the CONTROL S command determines the number of lines the Roll Mode routine removes, unless one of the following exceptions exists:

- Fewer lines are on the output queue than the RNUM value (less lines rolled).
- The number of lines displaced by a status display is smaller than RNUM value (less lines rolled).
- Roll Deletable (RD) was mode specified and the number of deletable lines on screen is smaller than the RNUM value (less lines rolled).
- The top line on screen following the roll is a continuation line (one more line rolled).

COMMAND ROUTINE (IGC5407B) CHART JO

The Command routine analyzes the commands in the entry area, processes CANCEL attentions, and processes requests to remove message numbers. According to the reason for entry, the routine takes the following action:

- CANCEL attention: the Command routine sets flags, alters the DCM as appropriate, and then passes control to the appropriate I/O routine.
- CONTROL command (with no other operands): the Command routine passes control to the Delete 3 routine to erase a segment of messages from the screen.
- Delete verification: the Command routine passes control to the Delete 4 routine to process the deletion verification.
- CONTROL commands: the Command routine issues SVC 34 to pass the command to the system's command processing routines. Upon return from SVC 34, the Command routine determines if a parameter list was provided by the SVC 34 routines. If not, the Command routine blanks the entry area and passes control to the appropriate I/O routine. If a parameter list was provided, and the command in the entry area was CONTROL E,N, the Command routine erases

the message numbers from the screen and passes control to the appropriate I/O routine. For all other commands, the Command routine passes control to the routine indicated in the SVC 34 parameter list.

- All other commands: the Command routine issues SVC 35 to write the command to the message area of the console screen, and then issues SVC 34 to pass the command to the system's command processing routines. Upon return from SVC 34, the Command routine determines if a parameter list was provided. If not, the Command routine blanks the entry area nnd passes control to the appropriate I/O routine. If a parameter list was provided, the Command routine passes control to the routine indicated in the parameter list.

OPTIONS ROUTINE (IGC5A07B) CHART JP

The Options routine receives control from the Command routine to analyze the minor operand values (DEL, CON, SEG, RNUM, RTME) of the CONTROL command specification (S) operand. If these minor operands and their respective values are valid, the Options routine changes the current values accordingly and indicates the screen options in effect. If these minor operands and/or their values are not valid, this routine exits to the appropriate device I/O routine with instructions to write the appropriate message from the Message routine. The Options routine also displays all the current screen option values when requested by the CONTROL S[,REF] command.

- CON=N and no hard copy
- DEL=R or RD and no hard copy
- DEL=R or RD and no timer

If a warning message is required, this routine sets the appropriate indicators and continues processing. If an error message is required, the Options routine exits to the Message 1 routine (IGC5D07B).

Upon preparing to exit, the Options routine indicates the proper location of the cursor. If a warning message is to be displayed, this routine transfers control to Message 1 (IGC5D07B). If the value of DEL is changed to or from R or RD, or if the RTME operand has been changed, the Options routine sets an indicator bit in the DCM and exits to the Timer Interpreter routine (IGC5K07B). If a warning or error message is required, and a return to the Timer Interpreter routine is also indicated, the Options routine exits to the Timer Interpreter with instructions to pass control to the Message 1 routine when finished.

DELETE 1 ROUTINE (IGC5607B) CHART JQ

Delete 1 handles the erase commands CONTROL E,F and CONTROL E,nn[,nn]. If conversational mode is in effect, Delete 1 updates the screen-image buffer by marking and numbering those messages selected for deletion, and passes control to Message 1 (IGC5D07B) to write the DELETION REQUESTED message. If nonconversational mode is in effect, Delete 1 updates the screen image buffer by marking and numbering those messages to be deleted, and passes control to Delete 4 (IGC5907B) for immediate deletion. Delete 1 exits to Message 2 (IGC5E07B) if the erase command is inconsistent, or if it has an invalid operand.

DELETE 2 ROUTINE (IGC5707B) CHART JR

The Delete 2 routine handles the deletion of intervention-required action messages and messages indicated for deletion by the DOM macro instruction. Delete 2 marks as automatically deletable those messages successfully acted upon or indicated for deletion by the DOM macro instruction, if the display device is not busy or a delete request is pending. If the device is busy or a delete request is pending, Delete 2 passes control to the MCS Router routine (IGCXL07B). Delete 2 marks all intervention required action messages as automatically deletable and passes control to the DIDOCS Processor (IGC5107B) if a delete request is pending. Otherwise, Delete 2 passes control to either Delete 4 (IGC5907B) if automatic deletion is desired, or to the appropriate device I/O routine to write the message area when automatic message deletion is not specified.

DELETE 3 ROUTINE (IGC5807B) CHART JS

Delete 3 handles the erase command CONTROL [E,SEG] and deletion by cursor or light pen. It updates the screen image buffer by marking and numbering the message area segment for deletion. If conversational mode is in effect, Delete 3 passes control to Message 1 (IGC5D07B) to write the appropriate message on the instruction line requesting operator verification of the messages marked for deletion. If nonconversational mode is in effect, Delete 3 passes control to Delete 4 (IGC5907B) for immediate deletion. If the value of SEG is 0, or if there are no deletable messages within the message area segment, Delete 3 exits to Message 2 (IGC5E07B) to display the appropriate error message.

DELETE 4 ROUTINE (IGC5907B) CHART JT

Delete 4 examines the message indicators in the screen control table in the DCM and blanks those lines in the screen image buffer marked for automatic or regular deletion, while it moves nondeletable lines up to the first available message line. This routine also handles the command CONTROL D,N[,HOLD] by numbering all visible message lines.

If the deletion of messages is successful when a status display is not on the screen and a full screen condition exists, Delete 4 returns control to Display 1 (IGC5207B). Otherwise, Delete 4 exits to either Message 1 (IGC5D07B) to display a message informing the operator that unviewable messages are on the output queue if the screen is not yet full, or to the appropriate device I/O routine to display the MESSAGE WAITING warning message and write the entire screen.

LIGHT PEN/CURSOR ROUTINE (IGC5F07B) CHART JU

The Light Pen/Cursor Service routine handles light pen and cursor interruptions. If a light pen or cursor delete request occurs on the message line of a non-action message or on the asterisk of an action message, the routine transfers control to Delete 4 (IGC5907B) or Delete 3 (IGC5807B), depending on whether delete verification is indicated. If a light pen detect or cursor placement is on *ENTER* in the instruction line, this routine passes control to the appropriate device I/O routine to perform a READ operation. When the light pen or cursor is positioned on *CANCEL* in the instruction line (to cancel a request) or on *E* in the status display title line (to erase a status display), the Light Pen/Cursor Service routine passes control to the Command routine (IGC5407B). When the light pen or cursor is positioned on *D C,K* in the instruction line or *F* in the status display title line, control passes to the Command routine. If the light pen is positioned in the PFK line, exit is to PFK 1 (IGC6A07B). If the light pen or cursor is positioned on any location other than those mentioned above, an error exists. Control is passed to Message 1 (IGC5D07B) or Message 2 (IGC5E07B) to display the appropriate error message.

PFK ROUTINE 1 (IGC6A07B) CHART KL

PFK 1 receives control from Processor 1, Load 1 when a PFK is pressed or when a PFK key has been verified and there are more commands or keys to be processed. It also receives control from the Light Pen/Cursor routine when a light pen interruption occurs for a number displayed in the PFK display line.

If entry is to cancel a PFK request, the commands associated with the canceled PFK key are removed from the entry area of the DCM. Also, control information is removed from the PFK area of the DCM so that later use of the same PFK key will cause new copies of the definitions to be used.

If entry is to enter a command, the number of the key is placed in the DCMPFKNM field of the DCM, and the routine checks to see if the key is valid. If it is not, exit is made to Message Module 1 so that an error message can be issued. Otherwise, a check is made to see if the key is already in process (a key may have several commands associated with it, each of which must be processed separately). If the key is not in process, it is flagged as in-process-now, and a check is made to see if the key is defined as a list of key numbers. If it is, control is returned to the entry point of the routine (PFK 1) so that each key number in the list can be processed separately.

When the routine determines that a key number in the list is associated with a command to be processed, the command is moved to the entry area buffer in the DCM. If the key is in conversational mode, the command is written to the screen where the operator may cancel or accept it. If the key is not in conversational mode, the command is executed by simulating an operator ENTER action.

After all commands have been processed, the PFK work area is cleaned up and control is returned to the Processor 1 routine.

PFK ROUTINE 2 (IGC6B07B) CHART KM

PFK 2 receives control from the Processor 1 routine when entry is made to redefine a PFK or to write or erase the PFK display line (the line of the display console screen containing PFK key numbers which may be entered by light pen selection).

The routine first determines which function is requested. If the request is to erase or display the PFK display line, exit is made to the I/O for the device for which the display is requested. If the request is to redefine a PFK, the routine scans the new command and, if no errors are encountered, replaces the old definition with the new. Control is then returned to the Processor 1 routine.

TIMER INTERPRETER (IGC5K07B) CHART JV

The Timer Interpreter analyzes the timer intervals at which each of the display console devices specifying roll mode is scheduled to be rolled, and sets indicators to notify the Roll Mode routine (IGC5J07B) whenever the timer interval has elapsed for one or more of the display console devices.

The routine initially stores the timer interval (RTME if roll mode is specified, 0 if not) for each display console device in its respective DCM. The greatest common divisor (GCD) of all the non-zero timer intervals is constantly updated as new devices are added as display consoles. The Timer Interpreter routine sets the timer equal to the GCD.

Whenever the timer elapses, this routine adds the GCD to the time counter of each of the display console devices whose timer interval is not equal to 0. If the new total equals or exceeds the timer interval for one or more display console devices, the Timer Interpreter notifies the DIDOCS Processor routine (IGC5107B) that the messages on each of these devices are ready to be rolled.

The Timer Interpreter gets control for one of three reasons. First, if control is passed from the DIDOCS Processor, the timer has elapsed, in which case it returns to the DIDOCS Processor. Second, if the routine gets control from the Options routine, the Open-Close routine, or the Asynchronous Error routine, the GCD needs to be updated for a new set of intervals. Control is passed from the Options routine when a display device is put into or taken out of roll mode, or the value of RTME is changed, in which case an exit is taken to the appropriate device I/O routine or one of the message routines. Entry is from the Open/Close routine when a display device in roll mode is removed as an operator's console; control returns to the Open/Close routine (IGC5G07B). Entry is from the Asynchronous Error routine in order to set a single timer interval of 30 seconds for the removal of the error message; exit is to the Message 2 routine (IGC5E07B). Third, the routine gets control from the Roll Mode routine (IGC5J07B) to insert in the first two character positions of the first new message the number of message lines remaining on the output queue. The Timer Interpreter then exits to the appropriate device I/O routine if the warning message bit is on. Otherwise, the Timer Interpreter passes control to the Display 1 routine.

STATUS DISPLAY INTERFACE 1 ROUTINE (IGC6L07B) CHART KD

This routine receives control from the Processor routine (IGC5107B) or Display 3 (IGC6207B) to process multiple-line WTO messages that do not have descriptor codes 8 and 9 (descriptor codes 8 and 9 indicate that a display is a response to an operator's request and is to be presented out-of-line, in a display area of a display console).

Upon initial entry, the routine sets a flag in the UCM indicating that a status display is in progress. This flag insures that all of the lines of the status display will be displayed contiguously to prevent interleaving of other messages with the lines of the status display. The routine then moves lines of the WTO into the DCM until a sufficient number of lines have been passed to fill the screen or until the last line is encountered. If the screen becomes full before the last line is encountered, exit is made to the device dependent I/O routine to issue a MESSAGE WAITING message. When the last line is passed to the DCM, the WQE turns off the multiple-line WTO flag in the UCM and checks the WQE buffer for additional messages to be processed. If any are found, control is passed to the Display 3 routine (IEECVFT2) for further processing. Otherwise, control is passed to the I/O routine to display the messages that were moved to the DCM.

STATUS DISPLAY INTERFACE 2 ROUTINE (IGC6M07B) CHART KE

This routine receives control from the processor routine (IGC5107B) to process requests for multiple-line WTO messages that have descriptor codes 8 and 9 (these codes indicate that the message is a status display requested by the operator, and that the display is to be presented out-of-line in a display area on a display console screen). This routine functions as a controller for Status Display Interface routines 4 and 6.

Upon initial entry, the routine searches the console output queue for a WQE representing a multiple-line WTO. When one is found, the routine locates in the DCM the screen area control block (SACB) for the display area in which the display will appear. If the WQE represents a new display, the SACB is initialized, and any status display in the area is dequeued. If the display area is now empty (that is, it contains no other operator messages), control passes to the Status Display Interface 6 routine (IGC6007B), which puts as many lines as possible of the message into the

screen image buffer (in the DCM). If the display will overlay other operator messages, control passes to Status Display Interface 4, which sets up a write-area. This separate write-area will be used by the I/O routines to write the display to the screen (this avoids using the DCM screen image buffer as a write-area, which would destroy the overlaid operator messages). If message area lines below the display area in use require blanking, control is passed to the Status Display Interface 5 routine (IGC6P07B).

STATUS DISPLAY INTERFACE 3 ROUTINE (IGC6N07B) CHART KF

This routine receives control from the Processor routine (IGC5107B) to process CONTROL commands affecting out-of-line status displays (K D,H; K D,U; K D,F).

If entry has been made for processing CONTROL commands, the routine takes action according to the operands of the command:

- K D,H -- an internal PM A is issued to stop the time interval updating of the display. The display remains in the screen but the routine rewrites the control line to contain frame and update options.
- K D,U -- an internal MN A is issued to continue updating the display. The control line is rewritten to contain stop and hold options.
- K D,F -- a control line containing the new frame number is built in the write area and control is passed to the I/O routine to write the new frame.

STATUS DISPLAY INTERFACE 4 ROUTINE (IGC6O07B) CHART KG

This routine receives control from the Status Display Interface 2 routine (IGC6M07B) to move lines of a status display from the WQE into a temporary write area in the DCM. This routine is entered only for status displays that overlay other operator messages on a display console screen.

This routine uses the instruction line and the two lines of the entry area as the temporary write area. It moves three lines of the message into this temporary write area, and then exits to the I/O routine to write the lines to the screen. When the last line of the display is put into the temporary work area, the routine blanks any remaining display area lines 3 lines at a time and puts the FRAME LAST indicator in the control line.

STATUS DISPLAY INTERFACE 5 ROUTINE (IGC6P07B) CHART KH

This routine is entered from the Processor routine (IGC5107B) to process CONTROL commands affecting out-of-line status displays (K E,D; PM A). Action is taken as follows:

- K E,D -- the appropriate major and minor WQEs are removed from the queue, and control is passed by means of the XCTL macro instruction to the I/O routine, which erases the display from the screen.
- PM A -- the dynamic display indicator in the DCM is turned off, the appropriate WQEs are removed from the queue, and control is passed to the I/O routine, which erases the display from the screen.

STATUS DISPLAY INTERFACE 6 ROUTINE (IGC6Q07B) CHART KI

This routine is entered from the Status Display Interface 2 routine (IGC6M07B) to move lines of a status display from the WQE into the appropriate message area lines of the screen image buffer (in the DCM). This routine is entered only if the status display does not overlay any other operator messages.

On entry, this routine moves lines from the WQE into the screen image buffer until the display area is filled. When all lines of the display have been moved, any lines remaining in the display area are blanked, and the control line is rewritten to indicate FRAME LAST.

STATUS DISPLAY INTERFACE 7 (IGC6T07B) CHART KJ

This routine receives control from Status Display Interface 5 (IGC6P07B) to blank message area lines below a status display that is being displayed in a display area.

Upon entry, this routine determines the number of lines that require blanking and locates the first line to be blanked. It then fills the instruction line and the entry area (in the DCM) with blanks. After these lines are blanked, control is passed to the device-dependent I/O routine to blank the first three lines that require blanking. The I/O routine uses the three lines as a write area to write blanks to the screen. (This procedure prevents the message area from being broken up into blocks of general message traffic and blocks of status displays. It also allows any messages that were overlayed with

blanks to remain untouched in the DCM screen image buffer. When the status display is erased, the screen is again written from the screen image buffer, and the messages reappear.)

CLEANUP ROUTINE (IEECVFTG) CHART KN

This routine receives control from the OPEN/CLOSE routine when a request for closing a device has been entered; it also receives control from the Asynchronous Error routine when a request to vary console status has been entered or when a device is recovering from an asynchronous error. This routine removes status displays from the message queues, and reinitializes the screen area control blocks (SACBs).

Upon entry, this routine determines the reason for entry and then determines if the console involved in the entry has a MONITOR Active display in "update" mode. If so, the display is stopped (STOPMN). The routine then searches the console's SACBs for partially output status displays. If any are found, they are freed. The routine then takes the following action according to the reason for entry:

- CLOSE -- the SACB configuration is reinitialized to the value specified during system generation, all messages are removed from the queue, and control is passed to Processor 1.
- Change in status to SD -- inline messages are freed, the SACBs are reinitialized to the values specified during system generation, and control is passed to the Asynchronous Error routine (IEECVETC).
- Change in status to MS -- the SACB configuration is reinitialized to the values specified during system generation, all SACBs are marked as unused, and control is passed to the Asynchronous Error routine (IEECVETC).
- Change in status to FC -- the SACB configuration is reinitialized to the values specified during system generation, and control is passed to the Asynchronous Error routine (IEECVETC).
- Asynchronous Error recovery -- out-of-line displays in progress and on the output queue are freed, the SACB configuration is retained, and control is passed to the Asynchronous Error routine (IEECVETC).

EXTENDED PRECISION FLOATING POINT SIMULATOR ROUTINES

IEAXPSIM

The SPIE user routine passes in register 1 the address of a pointer to a doubleword area. IEAXPSIM determines from the field CVTOPTA (offset X'182.7') in the CVT whether the extended precision floating-point feature is supported by CPU hardware. If so, IEXPSIM moves the name of the module IEAXPDXR into the doubleword area; if not, the module name IEAXPALL is moved into the area. The user routine uses this name to bring the appropriate processing module into main storage.

IEAXPALL

The user SPIE routine passes in register 1 the address of a parameter list containing the address of the PIE, the address of the register save area (containing the contents of the registers at the time of the interruption), a pointer to a 400-byte work area, and a pointer to a byte of main storage. If this byte of main storage is not 0, the validity of the low order bit of the result of a DXR instruction has not been ensured. IEAXPALL examines the operation code and register specifications of the failing instruction pointed to by the program check old PSW. If any of these are invalid, an appropriate code and error indicator are returned to the caller. For the MXD instruction, IEAXPALL also issues a SPIE macro instruction to intercept any interruptions caused by invalid addressing.

IEAXPALL then performs the necessary computation and indicates any exceptional conditions encountered during computation. For AXR and SXR instructions, the condition code is set in the Old PSW in the PIE. A code in register 15 and an indicator in bits 28-31 of the PSW in the PIE are returned to the caller.

Code	Indicator	Meaning
00	unchanged	Operation successful; no exceptional conditions.
FF	1	Operation code did not specify an extended precision operation to be processed by this module; no simulation attempted.
FF	4	Protection exception encountered during simulation of an MXD instruction; operation suppressed.
FF	5	Addressing exception encountered during simulation of an MXD instruction; operation suppressed.
FF	6	Specification exception encountered during simulation of an MXD instruction; operation suppressed.
FF	C	Exponent overflow encountered; operation completed.
FF	D	Exponent underflow encountered; operation completed.
FF	E	Significance exception encountered; operation completed.
FF	F	Floating-point divide exception encountered; operation is completed.

IEAXPDXR

The user SPIE routine passes a parameter list identical to that passed to IEAXPALL except that the work area is 240 bytes. IEAXPDXR processing is identical to that in IEAXPALL except that only the divide instruction is handled. Any other operation code causes control to return to the caller with a code of X'FF' in register 15 and an interruption code of 1 in the PSW.

If IEAXPDXR is used on a CPU without the floating-point feature, an 0C1 ABEND code will result when an attempt is made to simulate an extended precision divide instruction.

SMF ROUTINES

A discussion of the two major SMF routines, which perform the functions described above, follows.

SMF Wait Time Collection Routine (IEAQWAIT) Chart AE

This routine, which resides in the nucleus, records system wait time. It is entered from the first-level interruption handlers whenever an external or input/output interruption occurs.

If the current TCB represents the system wait pseudo task, the system has been in wait condition. (Otherwise, control is simply returned to the calling interruption handler.) The SMF Wait Time Collection routine reads the interval timer. The timer value is compared with the value in the first word of a special save area, SYSWSAVE. The value in SYSWSAVE was placed there by the Dispatcher when the system entered the wait condition. The comparison yields the elapsed system wait time. It is added to the value in the second word of SYSWSAVE, which gives the accumulated system wait time.

After recording the accumulated system wait time, the routine returns control to the interruption handler that called it.

SMF EXCP Counting Routine (IEASMFEX) Chart KA

This routine, which resides in the nucleus, counts and records the number of EXCPs associated with user data sets. The count is maintained for all data sets recorded in the TIOT of a problem program. It includes both direct EXCPs (SVC 0) and indirect EXCPs (those resulting from channel end/abnormal end conditions or programmer-controlled interruptions). The routine counts references to the data sets by the Open, Close, and EOV routines.

Upon entry from IOS the EXCP Counting routine performs the following tests; if any test fails, control is returned to the caller.

- The TCBTCT field of the TCB is checked for the existence of a timing control table (TCT).
- The TCTIOTBL field of the TCT is checked for the existence of an I/O extension of the TCT.
- The DCBOFLGS field of the DCB is checked to assure that the DCB is either open or in the process of being opened or closed.

If all tests are successful, the routine searches the TCT I/O Table segment of the TCT to find the correct EXCP counter (TCTDCTR). There is a counter for each combination of DCB and UCB. Counts are accumulated on a data set/device basis. (See Figure 44.)

When the correct EXCP counter has been found, the routine adds 1 to the counter. Then, if the data set is not SYSOUT, the routine returns to the caller.

If the reference was made to a SYSOUT data set, the routine checks the TCTOUTLM field of the TCT I/O Table to determine if an output limit is specified. If an output limit is found, it is compared to the total of the EXCP count fields for each device associated with the data set. If the output limit is not exceeded, or if none was specified, the routine returns to the caller.

Whenever an output limit is exceeded, one of the following actions is taken:

- If exits are not allowed, the program is abnormally terminated with an error code of 722.
- If exits are allowed, the routine creates an IRB/IQE representing the SMF Output Limit Expiration routine (IEATLEXT). The Stage 2 Exit Effector is then entered to schedule the execution of IEATLEXT.

SMF Output Limit Expiration Routine (IEATLEXT) Chart KB

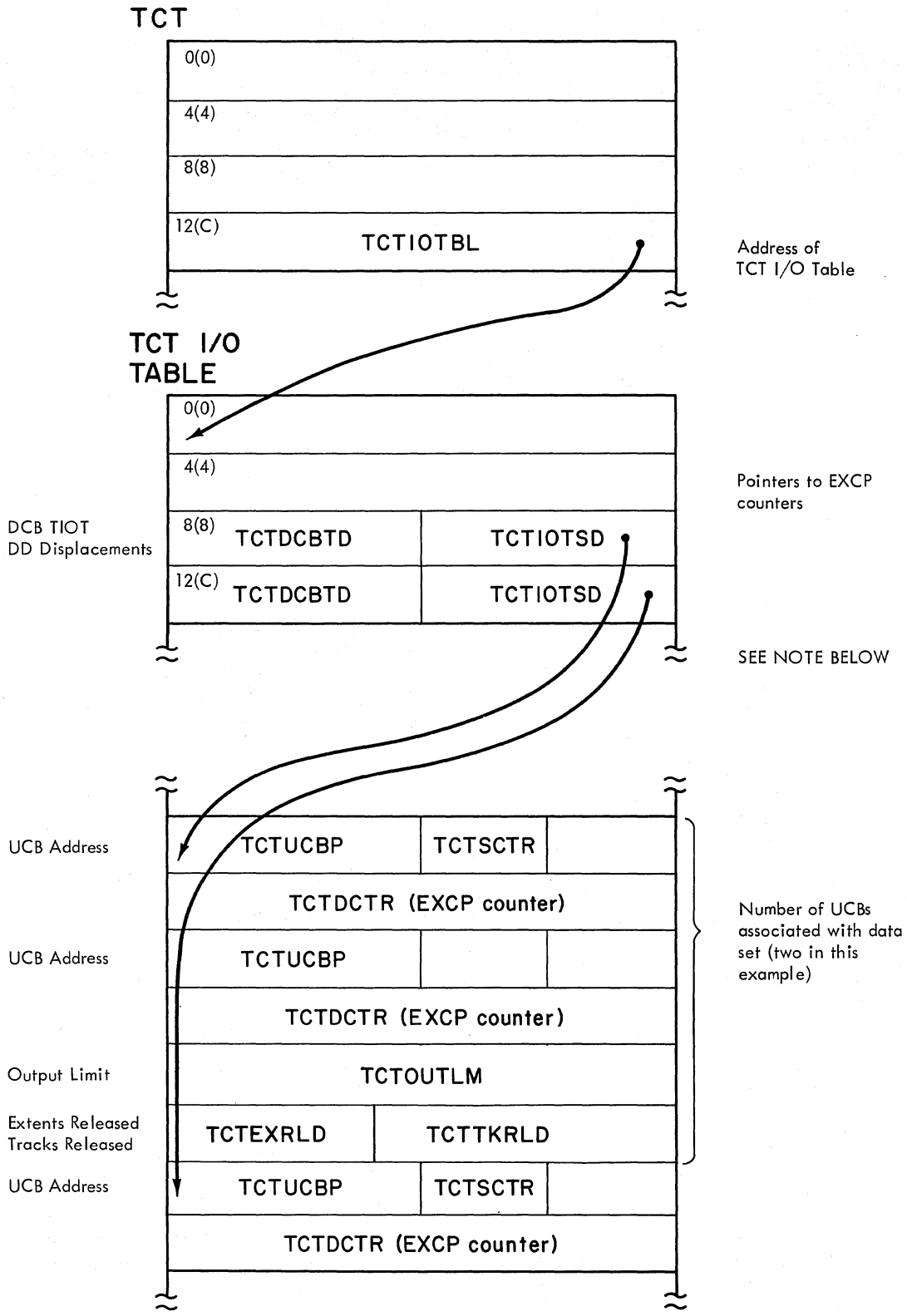
This routine, which is resident in the nucleus, provides an interface with a user output limit routine (IEFUSO).

The routine (IEATLEXT) receives control from the EXCP Counting routine when the output limit has been exceeded for a SYSOUT data set. The routine passes control to the user output limit routine. It also passes a two-word parameter list containing the address of the JMR (Job Management Region) and the address of the DCB.

The user routine determines whether or not to grant an increase to the output limit. It returns control to the SMF Output Limit Expiration routine with a return code of 0 for no increase, or 4 for an increase. If the return code is 4, register 1 contains the amount of increase to be granted.

If an increase is granted, the SMF Output Limit Expiration routine adds the value of the increase to the output limit specified in the TCT I/O Table. If an increase is not granted, the routine schedules an abnormal termination of the program with an error code of 722.

Note: This routine (IEATLEXT) also handles time limit expirations. See "Timer Supervision Routines."



NOTE: The end of the first section of the TCT I/O Table segment is marked by an entry of all zeros. The second section follows immediately.

Figure 44. Example of TCT Pointers Used by EXCP Counting Routine

The following pages provide nonprocessing information about the supervisor modules in the following format:

Module Name -- Common Name: The library name of the module. Common name is the descriptive name used for the module within this publication.

Chart: The identification of the flowchart found following this section.

CSECT: The CSECT name. This is understood to be an entry point and is not included under "Entry."

Entry: The additional entry point(s) specified by entry statements.

Exit: The module that receives control from this module upon completion of processing.

Modules Used: The modules that perform processing for this module. Entry to these modules is usually by an SVC or BALR instruction.

Table/Work Areas: The control blocks, and work areas used by this module.

Function: A brief description of what this module does.

Modules are grouped by type of supervision corresponding to Section 2, "Method of Organization." Where one module performs several distinct functions, one synopsis is given to each of the functions.

INTERRUPTION SUPERVISION

IEAAIH00 -- SVC First-Level Interruption Handler

Chart: AA

CSECT: IEAAIH00

Entry: IEAASC00

Exit: IEAATA00 (SVC SLIH)
IEA0XE00 (SVC Type 1 Exit)
IEANTM00 (ABEND)
IEA0DS (Dispatcher)

Modules Used: IEATRC (Trace)

Tables/Work Areas: SVCSAV (within IEAAIH00), SVCTABL, CVT, IEATCBP (NEW/OLD in Dispatcher), SVC Old PSW.

Function: Determines the type of SVC request, passes control to the appropriate Type 1 SVC routine, or passes control to the SVC SLIH for Types 2, 3, and 4 requests for further processing.

IEAAIH00 -- SVC Type 1 Exit

Chart: AA

CSECT: IEAAIH00

Entry: IEA0XE00 from SVC FLIH

Exit: IEA0DS (Dispatcher in IEAAIH00)

Tables/Work Areas: SVCSAV (within IEAAIH00), IEATCBP (NEW/OLD in Dispatcher), SVC Old PSW.

Function: Provides return linkage from a Type 1 SVC routine.

IEAAIH00 -- I/O First-Level Interruption Handler

Chart: AD

CSECT: IEAAIH00

Entry: IEAAIO02

Exit: DISMISS (in Timer/External FLIH)

Modules Used: IECINT (I/O Supervisor)
PSWNDX2 (Trace Routine)
IEAQWAIT (Wait Time Collection)

Tables/Work Areas: IORGSAV (in IEAAIH00), I/O Old PSW.

Function: Provides initial processing for I/O interruptions.

IEAAIH00 -- Timer/External First-Level Interruption Handler

Chart: AD

CSECT: IEAAIH00

Entry: IEAQEX00
DISMISS from I/O FLIH

Exit: IEA0DS (Dispatcher)

Modules Used: IEECVCRX (External SLIH for external interruption)
IEA0TI00 (Timer SLIH for timer interruption)
IEAQWAIT (Wait Time Collection)

Tables/Work Areas: IORGSAV (in IEAAIH00), PDSAV (in IEAAIH00), External Old PSW, IEATCBP (NEW/OLD in Dispatcher).

Function: Provides initial processing for timer and external interruptions.

IEAAIH00 -- Program Check First-Level Interruption Handler

Chart: AE

CSECT: IEAAIH00

Entry: IEAAPK00

Exit: IEA0PL00 (ABTERM)

Modules Used: IHLMCIH (Monitor Call Interruption Handler)

Tables/Work Areas: Program Old PSW, IEATCBP (NEW/OLD in Dispatcher), PIE, PICA.

Function: Processes program check interruptions.

IEAAIH00 -- Dispatcher

Chart: AF

CSECT: IEAAIH00

Entry: IEA0DS
XTG from Stage 3 Exit Effector

Exit: IEAEXTEF (Stage 3 Exit Effector)
IEAATA00 (DXA in EXIT)
Highest-priority ready task via LPSW instruction

Modules Used: IEA0TI02 (Timer Enqueue/Dequeue)
IEATRC (Trace)
IEAATA00 (SVRB create in SVC SLIH)
IEA0FN00 (FINCH)

Tables/Work Areas: CVT, PIB, XSNTCC (in IEAATC00).

Function: Pass control to the highest priority ready task in the system.

IEAATA00 -- SVC Second-Level Interruption Handler

Chart: AB

CSECT: IEAATA00

Entry: SVE from SVC FLIH
ABENDSUC return from GETMAIN

Exit: IEAAXT (EXIT in IEAATA00)
IEA0DS (Dispatcher)
IEELOGWR (Log routine)

Modules Used: IEAAMS00 (GETMAIN)
IEA0AB00 (ABTERM)
IEA0FN00 (FINCH in IEAATC00)
IEAASPRG (Subsystem purge)

Tables/Work Areas: SVC Old PSW, CVT, XSNTCC.

Function: Determines the type of SVC request and passes control to the appropriate type 2, 3, or 4 routine.

IEAATA00 -- SVC EXIT

Chart: AC

CSECT: IEAATA00

Entry: IEAAXT
IGC003 (SVC 3)
DXA from FINCH

Exit: IEA0DS (Dispatcher)
IEANTM00 (ABEND)

Modules Used: AIOS (I/O Supervisor)
IEAADL00 (DELETE)
IEAAMS00 (FREEMAIN)
IEA0EF00 (Stage 2 Exit Effector)
IEAAPT00 (POST)
IEA0TI02 (Timer Dequeue)

Tables/Work Areas: SVCSAV, SVC Old PSW, PIE, IEATCBP (NEW/OLD in Dispatcher), CVT.

Function: Provides return processing from Types 2, 3, and 4 SVC routines.

IEAATA00 -- Validity Check

CSECT: IEAATA00

Entry: IEA0VL00

Exit: Return to caller

Tables/Work Areas: IEATCBP (NEW/OLD in Dispatcher) and protection key

Function: Provides address validity checking.

IEAATA00 -- Task Switch Determination

CSECT: IEAATA00

Entry: TASKSWIT

Exit: Return to caller

Tables/Work Areas: IEAHEAD, IEATCBP (NEW/OLD in Dispatcher).

Function: Searches the TCB queue for the highest priority ready task and sets IEATCBP (NEW in Dispatcher) to that address.

TASK SUPERVISION

IEAQAT00 -- ATTACH with Subtasking

Chart: BA

CSECT: IGC042

Exit: IEA0DS (Dispatcher)
IEAATA00 (EXIT)
IEANTM00 (ABEND)

Modules Used: IEA0PL00 (ABTERM)
IEAAMS00 (GETMAIN)

Tables/Work Areas: IEATCBP (NEW/OLD in Dispatcher)

Function: Creates a TCB and a dummy PRB for the specified subtask. The resume PSW in the dummy PRB points to an SVC 6 (LINK) instruction which will create the request block for the requested module and place it on the RB queue of the subtask TCB.

IEAGED02 -- DETACH

Chart: BB

CSECT: IGC062

Exit: IEAATA00 (EXIT)
IEANTM00 (ABEND)

Modules Used: IEA0VL00 (Validity Check in IEAATA00)
IEA0AB00 (ABTERM)
IEAAMS00 (GETMAIN/FREEMAIN)
IJEAAWT00 (WAIT)
IEAAPT00 (POST)

Tables/Work Areas: CVT

Function: Removes the TCB and RB for an attached subtask.

IEAQCH00 -- CHAP

Chart: BC

CSECT: IGC044

Exit: IEANTM00 (ABEND)
return to caller

Modules Used: IEAATA00 (TASKSWIT)

Tables/Work Areas: CVT.

Function: Changes the dispatching priority of the caller or a subtask of the caller, and reorders the queue of TCBs accordingly.

IEAAXR00 -- EXTRACT without Subtasking

Chart: BD

CSECT: IGC040

Exit: IEAATA00 (EXIT)

Tables/Work Areas: TIOT, CVT.

Function: Returns the address of the Task I/O Table to the caller.

IEABXR00 -- EXTRACT with Subtasking

Chart: BE

CSECT: IGC040

Exit: IEAATA00 (EXIT)
IEANTM00 (ABEND)

Tables/Work Areas: IEATCBP (NEW in Dispatcher)

Function: Returns requested data from fields in the TCB of the caller, or from the TCB of a caller's subtask.

IEAAWT00 -- WAIT

Chart: BF

CSECT: IGC001

Exit: IEA0XE00 (SVC Type 1 Exit)

Modules Used: IEA0VL00 (Validity Check)
IEA0AB00 (ABTERM)

Tables/Work Areas: IEATCBP (NEW/OLD in Dispatcher)

Function: Enables the caller for I/O and external interruptions and sets the wait bit on in the specified ECB.

IEAAPT00 -- POST

Chart: BG

CSECT: IGC002

Exit: IEA0XE00 (SVC Type 1 Exit)

Modules Used: IEA0AB00 (ABTERM)

Tables/Work Areas: CVT, IEATCBP (NEW/OLD in Dispatcher)

Function: Sets on the completion bit in the specified ECB, decrements the wait count, and, if the wait count is 0, sets off the wait bit.

▶IEAGENQ1 -- ENQ

Chart: BH

CSECT: IGC048

Entry: IGC056

Exit: IEA0DS (Dispatcher)
IEAATA00 (EXIT)
IEANTM00 (ABEND)

Modules Used: IEA0VL00 (Validity Check)
IEAAMS00 (GETMAIN)

Function: Provides the caller with exclusive or shared access to a specified device.

▶IEAGENQ2 -- ENQ with Shared DASD

See IEAGENQ1 (ENQ).

Function: In addition to the ENQ functions, this module processes RESERVE macro instruction requests.

▶IEAGENQ1 -- DEQ

Chart: BJ

CSECT: IGC048

Exit: IEAATA00 (EXIT)
IEANTM00 (ABEND)

Modules Used: IEAAMS00 (FREEMAIN)
IEAATA00 (TASKWIT)

Function: Releases resources from accessibility to a program.

▶IEAGENQ2 -- DEQ with Shared DASD

See IEAGENQ1 (DEQ).

Function: In addition to the DEQ functions, this module initiates I/O activity when the reserve count is zero so that IOS will release the device.

▶IEAAPX00 -- SPIE

Chart: BK

CSECT: IGC014

Exit: IEAATA00 (EXIT)

Tables/Work Areas: PIE, PICA, CVT.

Function: Creates pointers from the TCB through the PIE and PICA to the specified user error handling routine.

▶IEAAST00 -- STAE

Chart: BL

CSECT: IGC00060

Exit: Return to caller.

Modules Used: IEAAMS00 (GETMAIN/FREEMAIN)

Tables/Work Areas: SCB

Function: Creates, cancels, or overlays a STAE control block

▶IEAQEF00 -- Stage 1 Exit Effector

Chart: BM

CSECT: IGC043

Exit: IEAATA00 (EXIT)

Modules Used: IEAAMS00 (GETMAIN)

Function: In response to an ETXR operand in a macro instruction, creates and queues an IRB for a user exit routine.

▶IEAAIH00 -- Stage 2 Exit Effector

Chart: BN

CSECT: IEAAIH00

Entry: IEA0EF00

Exit: Return to caller

Tables/Work Areas: LEQA (queue of RQEs in IEAAIH00), LEQC (queue of IQEs in IEAAIH00)

Function: Places IQEs and RQEs on the appropriate queues and sets a switch in the Dispatcher to enter the Stage 3 Exit Effector.

▶IEAAIH00 -- Stage 3 Exit Effector

Chart: BO

CSECT: IEAEXTEF

Exit: IEA0DS (Dispatcher)

Modules Used: SXA (Error Transient Area Scheduler in IEAAIH00)
TASKSWIT (Task Switch Determination in IEAATA00)

Function: Moves IQEs and RQEs from their respective queues to queues originating in the IRBs for the associated user exit routine.

TASK TERMINATION

IEANTM00 -- ABEND 0

Chart: CA

CSECT: IGC0001C

Exit: IEANTM0B (ABEND 11)
IEANTM0C (ABEND 12)
IEANTM0G (ABEND G)
IEFSD515 (Master Scheduler GO)
IEFSD599 (Master Scheduler SMALLGO)
IEAAWT00 (WAIT)

Modules Used: IEAQCH00 (CHAP)
IEAAMS00 (GETMAIN/FREEMAIN)
IEA0AB00 (ABTERM)
IEA0ST00 (TTIMER)
IGC0002x (Close)

Tables/Work Areas: CVT

Function: Determines the type of termination; processes normal termination or routes abnormal termination.

IEANTM01 -- ABEND 1

Chart: CC

CSECT: IGC0101C

Exit: IEANTM0B (ABEND 11)

Modules Used: IEA0ST00 (TTIMER)
IEAQ0CR (Graphics Purge)
IEAAMS00 (FREEMAIN)

Tables/Work Areas: CVT, TCB

Function: Schedule graphics hook; purge request queues.

IEANTM02 -- ABEND 2

Chart: CD

CSECT: IGC0201C

Exit: IEANTM03 (ABEND 3)
IEANTM09 (ABEND 9)
IEADTM22 (DAR 1)

Tables/Work Areas: CVT, IEATCBP (NEW/OLD in Dispatcher)

Function: Performs I/O purge function.

IEANTM03 -- ABEND 3

Chart: CD

CSECT: IGC0301C

Exit: IEANTM04 (ABEND 4)

Tables/Work Areas: CVT, IEATCBP (NEW/OLD in Dispatcher)

Function: Checks validity of request blocks.

IEANTM04 -- ABEND 4

Chart: CE

CSECT: IGC0401C

Exit: IEANTM05 (ABEND 5)
IEANTM07 (ABEND 7)
IEANTM08 (ABEND 8)
IEANTM0A (ABEND 10)
IEANTM0C (ABEND 12)
IEANTM0D (ABEND 13)
IEANTM0F (ABEND 15)

Modules Used: IEAAMS00 (GETMAIN/FREEMAIN)

Tables/Work Areas: CVT

Function: Determines if a dump is required and ensures that sufficient main storage is available for subsequent functions.

IEAMTM05 -- ABEND 5 (without Subtasking)

IEANTM05 -- ABEND 5 (with Subtasking)

Chart: CF

CSECT: IGC0501C

Exit: IEANTM09 (ABEND 9)
IEANTM06 (ABEND 6)
IEA0DS (Dispatcher)

Modules Used: IEAAMS00 (GETMAIN)
IGC0001I (OPEN)
IEA0AB00 (ABTERM)
IEAGENQ1 (ENQ)

Tables/Work Areas: IEATCBP (NEW/OLD in Dispatcher), CVT, TIOT, DEB, DCB, UCB.

Function: Opens the dump data set if possible. For MFT with subtasking, subtasks enqueued on the dump data set are dequeued and the terminating task is enqueued.

IEANTM06 -- ABEND 6

Chart: CC

CSECT: IGC0601C

Exit: IEANTM07 (ABEND 7)
IEANTM09 (ABEND 9)
IEANTM0C (ABEND 12)

IEANTM0D (ABEND 13)
IEANTM0F (ABEND 15)

Modules Used: IEAAMS00 (GETMAIN/FREEMAIN)
IEAAAD00 (SNAP)

Tables/Work Areas: CVT, TCB

Function: Prints dump messages and invokes ABDUMP.

IEANTM07 -- ABEND 7

Chart: CH

CSECT: IGC0701C

Exit: IEFSD515 (Master Scheduler GO)
IEFSD599 (Master Scheduler SMALLGO)

Modules Used: IEA0AB00 (ABTERM)
IEAQCH00 (CHAP)
IGC0002x (CLOSE)

Tables/Work Areas: CVT, IEEMSER (Master Scheduler Resident Data Area), IEATCBP (NEW/OLD in Dispatcher), TCB.

Function: Closes all data sets, dequeues IQEs.

IEANTM08 -- ABEND 8

Chart: CH

CSECT: IGC0801C

Exit: IEANTM07 (ABEND 7)
IEANTM0D (ABEND 13)
IEANTM0F (ABEND 15)

Function: Formats indicative dump.

IEANTM09 -- ABEND 9

Chart: CI

CSECT: IGC0901C

Exit: IEANTM04 (ABEND 4)

Modules Used: IEAAMS00 (GETMAIN/FREEMAIN)
IEAGENQ1 (DEQ)
IEECVWTO (WTO)

Tables/Work Areas: IEATCBP (NEW/OLD in Dispatcher), CVT, GQE.

Function: Processes recursions.

IEANTM0A -- ABEND 10

Chart: CJ

CSECT: IGC0A01C

Exit: IEANTM04 (ABEND 4)

Modules Used: IEAAMS00 (GETMAIN/FREEMAIN)

Tables/Work Areas: CVT

Function: Steals main storage for essential ABEND functions when there is insufficient main storage.

IEACTM0B -- ABEND 11 (with MCS)

See IEANTM0B.

IEANTM0B -- ABEND 11 (without MCS)

Chart: CK

CSECT: IGC0B01C

Exit: IEANTM00 (ABEND 0)
IEANTM01 (ABEND 1)
IEANTM0C (ABEND 12)
IEASTM13 (ASIR 3)

Modules Used: IEAAMS00 (GETMAIN/FREEMAIN)

Tables/Work Areas: IEECUCB (pointer to UCM), IEATCBP (NEW/OLD in Dispatcher)

Function: Purge WTOR queue and end any unended Multiple-line WTORs.

IEANTM0C -- ABEND 12

Chart: CK

CSECT: IGC0C01C

Exit: IEANTM0D (ABEND 13)

Modules Used: IEAAMS00 (GETMAIN/FREEMAIN)

Tables/Work Areas: IEATCBP (NEW/OLD in Dispatcher)

Function: Frees partially loaded programs and FRBs for the terminating task.

IEANTM0D -- ABEND 13 (Subtasking Only)

Chart: CK

CSECT: IGC0D01C

Exit: IEANTM0F (ABEND 15)

Modules Used: IEAGENQ1 (DEQ)

Tables/Work Areas: IEATCBP (NEW/OLD in Dispatcher).

Function: Performs an ENQ purge.

▶IEANTM0E -- ABEND 14 (Subtasking Only)

Chart: CL

CSECT: IGC0E01C

Exit: IEANTM07 (ABEND 7)
IEAATA00 (EXIT)

Modules Used: IEAAWT00 (WAIT)
IGC0002x (CLOSE)
IEAAMS00 (GETMAIN/FREEMAIN)

Tables/Work Areas: IEATCBP (NEW/OLD in Dispatcher), LEQC (IQE queue in IEAAIH00), IECIXAVL (pointer to TCB table of addresses), CVT.

Function: Closes data sets and purges RB queue.

▶IEANTM0F -- ABEND 15 (Subtasking only)

Chart: CM

CSECT: IGC0F01C

Exit: IEANTM0E (ABEND 14)

Modules Used: IEAAMS (GETMAIN/FREEMAIN)

Tables/Work Areas: TCB

Function: Purges IQEs and queues associated with subtasks of the terminating task.

▶IEANTM0G -- ABEND G

Chart: CB

CSECT: IGC0G01C

Exit: IEANTM01 (ABEND 1)
IEANTM02 (ABEND 2)
IEANTM0M (ABEND M)
IEAAAT00 (EXIT)
IEADTM22 (DAR 1)
IEADTM23 (DAR 2)
IEASTM11 (ASIR 1)

Modules Used: IEAAMS00 (GETMAIN/FREEMAIN)
IEA0AB00 (ABTERM)
IEAAWT00 (WAIT)

Tables/Work Areas: TCB, CVT, SCB, PIE, PICA

Function: Process abnormal termination and ABEND recursions.

▶IEANTM0H -- ABEND H

Chart: CO

CSECT: IGC0H01C

Exit: IEANTM04 (ABEND 4)

Modules Used: IEAAMS00 (GETMAIN/FREEMAIN)
IEEMFWTO (WTO)

Tables/Work Areas: TCB, CVT, DEB, TIOT

Function: Issues WTO messages identifying valid recursions.

▶IEANTM0J -- ABEND J (with Subtasking)

Chart: CP

CSECT: IGC0J01C

Exit: IEANTM04 (ABEND 4)

Tables/Work Areas: CVT, IEATCBP, TCB

Function: Checks validity of subtasking control blocks.

▶IEANTM0M -- ABEND M

Chart: CN

CSECT: IGC0M01C

Exit: IEANTM0G (ABEND G)

Modules Used: IEAAMS00 (GETMAIN/FREEMAIN)
IEECVWTO (WTO)

Tables/Work Areas: CVT, Message Information List

Function: When invoked by ABEND G, issues WTP messages for Type-1 SVC routines.

▶IEASTM11 -- ASIR 1

Chart: CQ

CSECT: IGC0111C

Exit: IEASTM12 (ASIR 2)
IEASTM13 (ASIR 3)
IEANTM01 (ABEND 1)
IEAATA00 (EXIT)

Modules Used: IEAAMS00 (GETMAIN/FREEMAIN)
IEAASY00 (SYNCH)
IGC0001F (Purge)

Tables/Work Areas: CVT, SCB

Function: Purges I/O, schedules user exit routine, and routes control for retry if requested.

▶IEASTM12 -- ASIR 2

Chart: CR

CSECT: IGC0211C

Exit: IEASTM14 (ASIR 4)
IEANTM0B (ABEND 11)
IEAATA00 (EXIT)

Modules Used: IGC0002x (Close)

Tables/Work Areas: CVT, IOB, SCB.

Function: Dequeues IOBs and closes DCBs with organizations other than ISAM or TAM for request blocks that will be purged.

▶IEASTM13 -- ASIR 3

Chart: CS

CSECT: IGC0311C

Exit: IEAATA00 (EXIT)
User's retry routine

Modules Used: IEAAMS00 (GETMAIN/FREEMAIN)

Tables/Work Areas: CVT, SCB.

Function: Purges request block queue and schedules retry routine.

▶IEASTM14 -- ASIR 4

Chart: CT

CSECT: IGC0411C

Exit: IEASTM13 (ASIR 3)
IEANTM0B (ABEND 11)
IEAATA00 (EXIT)

Modules Used: IGC0002x (Close)

Tables/Work Areas: CVT, SCB, IOB.

Function: Dequeue IOBs and close DCBs with ISAM or TAM organizations associated with request blocks that are going to be purged.

▶IEAIAB00 -- ABTERM (with Trace Table)

▶IEAGAB00 -- ABTERM (without Trace Table)

Chart: CY

CSECT: IEA0AB00

Exit: Return to caller
IEAAIH00 (DISMISS in I/O FLIH)

Modules Used: IEAATC00 (Transient Area Loading Task refresh routine)

Tables/Work Areas: CVT, TCB

Function: Schedules the abnormal termination of a specified task.

▶IEANAM00 -- ABTERM (with Subtasking)

Chart: CY

CSECT: IEA0AB00

Exit: Return to caller

Modules Used: IEAATA00 (IEAXERR -- Transient Area Loading Task refresh routine.

Function: Schedules the abnormal termination of a specified task.

▶IEADTM22 -- DAR 1

Chart: CU

CSECT: IGC0221C

Exit: IEADTM23 (DAR 2)

Tables/Work Areas: CVT

Function: Sets all tasks except the current task nondispatchable, and issues an SVC 51 (SVC Dump).

▶IEADTM23 -- DAR 2

Chart: CV

CSECT: IGCG0321C

Exit: IEA0DS (Dispatcher)
IEAGAB00 (ABTERM)

Modules Used: IEECVWTO (WTO)

Tables/Work Areas: SVT

Function: Attempts to reinstate system takks not in "must complete" status.

▶IEADTM24 -- DAR 3

Chart: CW

CSECT: IGC0421C

Exit: IEANTM03 (ABEND 3)
IEADTM25 (DAR 4)

Modules Used: IEEMFWTO (WTO)
IEAASPRG (Subsystem purge)
IEEVWTOR (WTOR)

Tables/Work Areas: CVT, QCBs, QEL, TCB.

Function: Attempts to reinstate system tasks in "must complete" status and problem program tasks. Routes control for continuation of abnormal termination.

IEADTM25 -- DAR 4

Chart: CX

CSECT: IGC0521C

Exit: IEA0DS (Dispatcher)

Modules Used: IEAASPRG (Subsystem Purge)

Tables/Work Areas: CVT, TCB, RB

Function: Sets nonreinstatable tasks permanently nondispatchable.

IEAMAD00 -- ABDUMP 1

Chart: DC

CSECT: IGC0L05A

Exit: IEAAAD0D (ABDUMP 2)

Modules Used: IEAAMS00 (GETMAIN/FREEMAIN)
IEAGENQ1 (ENQ)

Tables/Work Areas: CVT, DCB, TCB.

Function: Validity checks DCB and TCB addresses and obtains work area.

IEAAAD0D -- ABDUMP 2

Chart: DD

CSECT: IGC0D05A

Exit: IEAAAD0A (ABDUMP 3)
IEAAAD02 (ABDUMP 5)
IEAAAD03 (ABDUMP 6)

Tables/Work Areas: TIOT

Function: Dumps heading, completion code, interruption code, and PSW.

IEAAAD0A -- ABDUMP 3

Chart: DD

CSECT: IGC0A05A

Exit: IEAAAD01 (ABDUMP 4)

Function: Dumps TCB, active RBs, and LRBs.

IEAAAD01 -- ABDUMP 4

Chart: DE

CSECT: IGC0105A

Exit: IEAAAD02 (ABDUMP 5)
IEAAAD03 (ABDUMP 6)

Function: Dumps Job Pack Area Queue, GQEs, FQEs, and problem program boundaries.

IEAAAD02 -- ABDUMP 5

Chart: DF

CSECT: IGC0205A

Exit: IEAAAD03 (ABDUMP 6)

Function: Dumps save areas.

IEAAAD03 -- ABDUMP 6

Chart: DF

CSECT: IGC0305A

Exit: IEAAAD0B (ABDUMP 7)
IEAAAD04 (ABDUMP 8)
IEAAAD0K (ABDUMP 13)

Tables/Work Areas: TIOT, DEB, DCB, UCB, TCB.

Function: Dumps data set information pertinent to the task.

IEAAAD0B -- ABDUMP 7

Chart: DG

CSECT: IGC0B05A

Exit: IEAAAD04 (ABDUMP 8)

Function: Dumps the trace table.

IEAAAD04 -- ABDUMP 8

Chart: DG

CSECT: IGC0405A

Exit: IEAAAD05 (ABDUMP 9)

Function: Dumps registers.

IEAAAD05 -- ABDUMP 9

Chart: DH

CSECT: IGC0505A

Exit: IEAAAD0E (ABDUMP 10)
IEAAAD0C (ABDUMP 11)

Tables/Work Areas: CVT

Function: Dumps the nucleus of main storage.

IEAAAD0E -- ABDUMP 10

Chart: DH

CSECT: IGC0E05A

Exit: IEAAAD0C (ABDUMP 11)

Tables/Work Areas: CVT

Function: Dumps problem program storage.

IEAAAD0C -- ABDUMP 11

Chart: DH

CSECT: IGC0C05A

Exit: IEAAAD0F (ABDUMP 12)

Function: Dumps snapshot and resident reenterable modules.

IEAAAD0F -- ABDUMP 12

Chart: DI

CSECT: IGC0F05A

Exit: Return to caller

Tables/Work Areas: CVT

Function: Prints end of dump message and frees work area.

IEAAAD0G -- TCAM ABDUMP 1

Chart: DL

CSECT: IGC0G05A

Exit: IEAAAD0H (TCAM ABDUMP 2)

Tables/Work Areas: TCAM AVT, TCAM TNT

Function: Formats and prints the AVT in the basic section of the TNT.

IEAAAD0H -- TCAM ABDUMP 2

Chart: DL

CSECT: IGC0H05A

Exit: IEAAAD0I (TCAM ABDUMP 3)

Tables/Work Areas: TCAM AVT, TCAM TNT, TCAM TRM

Function: Formats and prints the terminal names as they appear on the TNT and their associated TRM entries.

IEAAAD0I -- TCAM ABDUMP 3

Chart: DM

CSECT: IGC0I05A

Exit: IEAAAD0J (TCAM ABDUMP 4)

Tables/Work Areas: TCAM AVT, TCAM TNT, TCAM TRM, TCAM QCB

Function: Formats and prints the TCAM Destination QCBs associated with the TRM entries.

IEAAAD0J -- TCAM ABDUMP 4

Chart: DM

CSECT: IGC0J05A

Exit: IEAAAD0B (ABDUMP 7)
IEAAAD04 (ABDUMP 8)
IEAAAD0K (ABDUMP 13)

Tables/Work Areas: TCAM DCB, TCAM LCB, CVT

Function: Formats and prints open TCAM DCBs and the LCBs associated with Line Group DCBs.

IEAAAD0K -- ABDUMP 13

Chart: DJ

CSECT: IEAAAD0K

Entry: IGC0K05A

Exit: IEAAAD04 (ABDUMP 8)
IEAAAD0M (ABDUMP 14)

Tables/Work Areas: CVT, TCB, CMT, HCT

Function: Formats and prints the GTF trace records.

IEAAAD0M -- ABDUMP 14

Chart: DK

CSECT: IEAAAD0M

Entry: IGC0M05A

Exit: IEAAAD0K (ABDUMP 13)

Tables/Work Areas: CVT, HCT

Function: Prints GTF control and error records.

IEAAAD0Y -- SVC DUMP 1

Chart: DA

CSECT: IGC0005A

Exit: IEAATA00 (EXIT)
IEAAAD0Z (SVC DUMP 2)
IEAMAD00 (ABDUMP 1)

Modules Used: IEAA0AB00 (ABTERM)
EXCP
IEAAWT00 (WAIT)
IEAAMS (GETMAIN)
IEA0RT00 (TIME)

Tables/Work Areas: CVT, SVRB, IOB, DCB, DEB, ECB

Function: Sets up necessary control blocks and writes a dump of main storage to a specified data set on a direct access device.

IEAAAD0Z -- SVC DUMP 2

Chart: DB

CSECT: IGC0Z05A

Exit: IEAATA00 (EXIT)

Modules Used: EXCP
IEAAWT00 (WAIT)
IEAAMS (FREEMAIN)
IEA0RT00 (TIME)

Tables/Work Areas: CVT, SVRB, IOB, DCB, DEB, ECB

Function: Writes a dump of main storage to a specified data set on tape.

IEAASPRG -- Subsystem Purge

CSECT: IEAASPRG

Entry: IEAASPRG

Exit: Return to Caller

Function: Allows subsystems to execute recovery procedures before being set permanently nondispatchable.

IEAAAT00 -- ATTACH without Subtasking

Chart: EA

CSECT: IGC042

Entry: IEA0AT01 reentry from Dispatcher
IEA0AT02 return from called routine

Exit: IEAATA00 (EXIT)
IEA0DS (Dispatcher in IEAAIN00)

Modules Used: IEAAMS00 (GETMAIN/FREEMAIN)
IEA0FN00 (FINCH)

Tables/Work Areas: CVT

Function: Creates a requested module, places it on the RB queue, and ensures that the requested module is in main storage.

IEAATC00 -- LINK, LOAD, XCTL

Chart: EB

CSECT: IGC006 (LINK)

Entry: IGC007 (XCTL)
IGC008 (LOAD)

Exit: IEAATA00 (EXIT)
IEANTM00 (ABEND)
IEA0DS (Dispatcher)

Modules Used: IEAAMS00 (GETMAIN/FREEMAIN)
TASKSWIT (Task switch determination in IEAATA00)
XFINCH (in IEAATC00)
IEAADL00 (DELETE)

Tables/Work Areas: CVT, Loaded Program List, Job Pack Area Queue, Resident Reenterable Module Area, Resident Access Method List.

Function: LOAD ensures that a copy of a module is available in main storage and returns its entry point to the caller. LINK ensures that a copy of the module is available in main storage, places a request block for it on the top of the RB queue, and provides for return linkage to the caller. XCTL ensures that a copy of the module is available in main storage, places a request block for it on the RB queue, and removes the caller and its RB from main storage.

IEAATC00 -- FINCH

Chart: EC, ED

CSECT: IGC006

Entry: IEA0FN00
XFINCH

Exit: Return to caller
IEANTM00 (ABEND)

Modules Used: IEAAMS00 (GETMAIN/FREEMAIN)
BLDL, Program Fetch,
IEAATC (LOAD)

Tables/Work Areas: CVT, IEATCBP (NEW/OLD in Dispatcher), XSNTCC (in IEAATA00).

Function: Provides the preparation necessary for Program Fetch to bring a requested module into main storage.

IEAATC00 -- Transient Area Loading Task

Chart: ED

CSECT: IGC006

Entry: IEAFNCH

Exit: Return to caller

Modules Used: Program Fetch
IEA0AB00 (ABTERM)
IGFDDRSR (Dynamic Device Reconfiguration)

Function: This section of IEAATC00 executes under its own TCB to bring the requested module in the SVC transient area.

IEAAID00 -- IDENTIFY

Chart: EE

CSECT: IGC041

Exit: IEAATA00 (EXIT)

Modules Used: IEAAMS00 (GETMAIN)

Function: Creates a minor request block for the specified module and places it on the Loaded Program List and on the minor RB queue of the identified routine.

IEAADL00 -- DELETE

Chart: EF

CSECT: IGC009

Exit: Return to caller

Modules Used: IEAAMS00 (FREEMAIN)

Tables/Work Areas: Resident Access Method List, Loaded Program List, Job Pack Area Queue.

Function: Finds the RB for the nonresident module and decrements the use count. If the use count becomes zero, the RB and the module are removed from main storage.

MAIN STORAGE SUPERVISION

▶IEAAMS -- GETMAIN/FREEMAIN

Chart: FA

CSECT: IGC010

Entry: IGC004 (GETMAIN)
IGC005 (FREEMAIN)

Exit: IEAAIH00 (IEA0XE00 -- Type 1 SVC Exit)

Tables/Work Areas: Boundary Box, FQE, GQE.

Function: Allocates and frees main storage.

COMMUNICATIONS TASK

▶IEECVCTI -- Communications Task Initialization

Chart: GA

CSECT: IEECVCTI

Exit: IEFSD569 (Master Scheduler Initialization)

Modules Used: IEAATC00 (LOAD)
IEAAMS00 (GETMAIN/FREEMAIN)
IEEVRFRX (Refer)
IEECVWTO (Write-to-Operator)
IEAAPT00 (POST)

Tables/Work Areas: UCM, Event Indication List (EIL).

Function: Initializes the Unit Control Module (UCM).

▶IEECVCRA -- Communications Task I/O Attention Handler

Chart: GB

CSECT: IEEBA1

Exit: Return to IOS

Modules Used: IEAAPT00 (POST)

Tables/Work Areas: CVT, UCM.

Function: Posts the attention ECB for an I/O attention.

▶IEECVCRX -- Communications Task External Interruption Handler

Chart: GB

CSECT: IEEBC1PE

Exit: Return to I/O FLIH

Modules Used: IEAAPT00 (POST)

Tables/Work Areas: CVT, UCM.

Function: Posts the external ECB for an external interruption.

▶IEEVWTOR -- Communications Task WTOR Service

Chart: GE

CSECT: IGC0103E

Exit: IEAATA00 (EXIT)
IEANTM00 (ABEND 0)
IEECVWTO (WTOR Service)

Modules Used: IEAAMS00 (GETMAIN)
IEAGENQ1 (ENQ/DEQ)
IEAAWT00 (WAIT)

Function: Provides buffer management and message identification.

▶IEECVCTW -- Communications Task Wait (non-MCS)

Chart: GF

CSECT: IEECIR45

Exit: IEECVCTR (Router)

Tables/Work Areas: UCM, EIL, CVT, IEEBASEB.

Function: Issues an SVC 72 instruction when one of the ECBs waited on by IEECVCTW is posted.

▶IEECVCTR -- Communications Task Router (non-MCS)

Chart: GG

CSECT: IGC0007B

Exit: IEECVCTX (Console Switch)
IEECVPMx (Device processor indicated by "x")
Return to IEECVCTW

Function: Determines from the ECB posted the type of processing necessary and passes control to the appropriate routine.

▶IEECVCTX -- Communications Task Console Switch (non-MCS)

Chart: GH

CSECT: IGCXL07B

Exit: Return to IEECVCTW
IEECVPMx (Device processor indicated by "x")

Function: Assigns functions of the primary console to the alternate console.

▶IEECVGCI -- Graphic Console Initailization

CSECT: IEECVGCI

Exit: IEECVCT1 (Console Initialization)

Tables/Work Areas: UCM, DCM

Modules Used: IEECVWTO (WTO)
EXCP
OPEN
LOCATE
BLDL

Function: Initializes display operator consoles.

▶IEECVPMX -- 1052 Processor Load 1 (non-MCS)

Chart: GO

CSECT: IGC0107B

Entry: IEECVPM

Exit: IEECVCTX (Console Switch)
IEECVCTR (Router)
IEECVOCX (1052 Open/Close)
IEECVPM1 (Non-MCS Load 2)

Modules Used: IEAAMS00 (GETMAIN/FREEMAIN)
IEAAPT00 (POST)
IEAAWT00 (WAIT)
IEE0303D (Command Processor)

Function: Performs read and write operations for a 1052 console.

▶IEECVOCX -- 1052 Open/Close

Chart: GQ

CSECT: IGC0I07B

Entry: IEECVOC

Exit: IEECVPMX (1052 Processor non-MCS)
IEECVCTX (Console Switch non-MCS)
IEECMPMX (1052 Processor MCS)
IEECLCTX (Console Switch MCS)

Modules Used: IEAAMS00 (GETMAIN/FREEMAIN)

Function: Performs Open/Close function for the 1052 console.

▶IEECVPMC -- Card Reader Processor (non-MCS)

Chart: GR

CSECT: IGC1107B

Entry: IEECVPM

Exit: IEECVCTX (Console Switch)
IEECVOCC (Card Reader Open/Close)
IEECVCTW (Wait routine)

Modules Used: IEE0303D (Command Processor)
IEAAWT00 (WAIT)
IEAAMS00 (FREEMAIN)

Function: Performs read operations for a card reader.

▶IEECVOCC -- Card Reader Open/Close

CSECT: IGC1I07B

Entry: IEECVOC

Exit: IEECVPMC (Card Reader Processor non-MCS)
IEECVCTX (Console Switch non-MCS)
IEECMPMC (Card Reader Processor MCS)
IEECLCTX (Console Switch MCS)

Modules Used: IEAAMS00 (GETMAIN)

Function: Performs Open/Close functions for a card reader.

▶IEECVPMP -- Printer Processor (non-MCS)

CSECT: IGC2107B

Entry: IEECVPM

Exit: IEECVCTW (Wait routine)
IEECVOCP (Printer Open/Close)
IEECVCTX (Console Switch)
IEECVPM1 (Non-MCS Load 2)

Modules Used: IEAAMS00 (FREEMAIN)

Function: Performs write operations to a printer.

▶IEECVPM1 -- 1052 Processor Load 2 (non-MCS)

CSECT: IGC0207B

Exit: IEECVPMX (1052 Processor non-MCS)
IEECVPMP (1052 Open/Close)

Modules Used: EXCP, POST, FREEMAIN, XCTL, WAIT

Tables/Work Areas: WQE, UCM, XSA, DCBD

Function: Processes multiple-line WTO messages for 1052 consoles and printer consoles. It also rings the audible alarm on the 1052 console when a permanent I/O error occurs.

IEECVOCP -- Printer Open/Close

CSECT: IGC2I07B

Entry: IEECVOC

Exit: IEECVPMP (Printer Processor non-MCS)
IEECVCTX (Console Switch non-MCS)
IEECMPMP (Printer Processor MCS)
IEECLCTX (Console Switch MCS)

IEEMFWTO (MCS) and IEENFWTO (non-MCS) WTO Service Routines

Chart: GC

CSECT: IGC0003E

Exit: IEANTM00 (ABEND)
IEEVWTOR
IEECVML1
IEECVML3

Modules Used: GETMAIN
WAIT
ENQ/DEQ
POST

Tables/Work Areas: CVT, USM, WQE, RQE, EIL

Function: This routine provides buffer management for the WTO macro instruction.

COMMUNICATIONS TASK WITH MULTIPLE CONSOLE SUPPORT

IEECMAWR -- Communications Task Router (MCS)

Chart: GI

CSECT: IEECMAWR

Exit: IEECMCTR (Mini-Router)
IEECMDSV (Device Interface)
IEECMWSV (WTO Processor)
IEECMWTL (NIP Message Buffer Writer)
IEECMDOM (Delete Operator Message)

Modules Used: IEAAWT00 (WAIT)

Tables/Work Areas: UCM

Function: Determines from the list of ECBs the function to be performed and passes control to the appropriate routine.

IEECMCTR -- Communications Task Mini-Router (MCS)

CSECT: IGC0007B

Exit: IEECLCTX (Console Switch)
IEECMPMx (Device processor indicated by "x")

Function: Passes control to the appropriate nonresident routine as indicated by the Router. The nonresident routines must receive control from a routine operating under control of an SVRB.

IEECLCTX -- Communications Task Console Switch Load 1 (MCS)

Chart: GJ

CSECT: IGCXL07B

Exit: IEECMAWR (Router)
IEECMCTX (Console Switch Load 2)
IEECOCTX (Console Switch Load 4)

Tables/Work Areas: CVT, UCM, CXSA.

Function: Switches consoles from master or primary to alternate.

IEECMCTX -- Communications Task Console Switch Load 2 (MCS)

Chart: GJ

CSECT: IGCXM07B

Exit: IEECNCTX (Console Switch Load 3)

Modules Used: IEECMWSV (WTO Processor)

Tables/Work Areas: CVT, UCM.

Function: Constructs message indicating console switch and new attributes of console.

IEECNCTX -- Communications Task Console Switch Load 3 (MCS)

Chart: GJ

CSECT: IGCXN07B

Exit: IEECMAWR

Tables/Work Areas: CVT, UCM.

Function: Move RQEs and WQEs from queue of old console to that of new.

IEECOCTX -- Communications Task Console Switch Load 4 (MCS)

Chart: GJ

CSECT: IGCXO07B

Exit: IEECMAWR (Router)
IEECMCTX (Console Switch Load 2)

Tables/Work Areas: CVT, UCM.

Function: Switches the hard copy log to an active, nongraphic console.

IEECMDSV -- Communications Task Device Interface (MCS)

Chart: GK

CSECT: IEECMDSV

Exit: IEECMAWR (Router)
IEECMCTR (Mini-Router)

Modules Used: IEAAMS00 (FREEMAIN)
IEAAPT00 (POST)
IEEVL03F (Write-to-Log)
IEECMWSV (WTO Processor)
IEECLCTX (Console Switch Load 1)

Function: Passes control to the device processor routines through the Mini-Router; provides queue management for console and system output queues.

IEECMWSV -- Communications Task WTO/R Processor (MCS)

Chart: GL

CSECT: IEECMWSV

Exit: IEECMAWR (Router)

Modules Used: IEECMDSV (Device Interface)
IEAASY00 (SYNCH)
IEAAMS00 (GETMAIN)

Tables/Work Areas: CVT, UCM.

Function: Places WQEs on appropriate console output queues.

IEECMDOM -- Communications Task Delete Operator Message (MCS)

Chart: GM

CSECT: IEECMDOM

Exit: IEECMAWR (Router)

Modules Used: IEAAMS00 (FREEMAIN)
IEECMCTR (Mini-Router)

Function: Indicates specified messages that are to be deleted.

IEECMWTL -- Communications Task NIP Message Buffer Writer (MCS)

Chart: GN

CSECT: IGC0907B

Exit: IEECMAWR

Modules Used: IEECVWTO (Write-to-Operator)
IEE0303F (Write-to-Log)

Function: Writes NIP messages to the system log or the operator.

IEECMPMX -- 1052 Processor Load 1 (MCS)

Chart: GP

CSECT: IGC0107B

Entry: IEECVPM

Exit: IEECLCTX (Console Switch Load 1)
IEECMDSV (Device Interface)
IEECVOCX (1052 Open/Close)
IEECMPM1 (1052 Processor Load 2)

Modules Used: IEAAMS00 (GETMAIN/FREEMAIN)
IEE0303D (Command Processor)

Function: Performs read and write operations to a 1052 console.

IEECMPM1 -- 1052 Processor Load 2 (MCS)

CSECT: IGC0207B

Exits: IEECMPMX (1052 Processor Load 1)

Modules Used: EXCP

Tables/Work Areas: WQE, UCM, UCME, XSA

Function: This routine writes the lines of a multiple-line WTO on a 1052 console.

IEECMPMC -- Card Reader Processor (MCS)

Chart: GS

CSECT: IGC1107B

Entry: IEECVPM

Exit: IEECMDSV (Device Interface)
IEECVOCC (Card Reader Open/Close)

Modules Used: IEAAMS00 (GETMAIN/FREEMAIN)
IEE1403D (Log)

Function: Performs read operations for a card reader.

IEECMPMP -- Printer Processor (MCS)

CSECT: IGC2107B

Entry: IEECVPM

Exit: IEECVOCP (Printer Open/Close)
IEECMDSV (Device Interface)
IEECLCTX (Console Switch Load 1)

Modules Used: IEAAMS00 (GETMAIN)

Function: Performs write operations to a printer.

IEECVML1 -- Non-MCS Multiple-line WTO Processor (Load 1)

CSECT: IEECVML1

Exits: IEECVML2, IEECVML4

Modules Used: XCTL, SVC 7

Tables/Work Areas: XSA, UCM, WQE, JSCB, TCB, RB

Function: This routine builds the major WQE for multiple-line WTO requests (for non-MCS systems).

IEECVML2 -- Non-MCS Multiple-line WTO (Load 2)

CSECT: IEECVML2

Exits: IEECVML4

Tables/Work Areas: WQE, UCM, XSA

Function: This routine builds the minor WQEs for the multiple-line WTO requests (non-MCS systems).

▶IEECVML3 -- MCS Multiple-line WTO Processor (Load 1)

CSECT: IEECVML3

Exits: IEECVML5, IEECVML6

Modules Used: XCTL, SVC 7

Tables/Work Areas: XSA, UCM, WQE, JSCB, TCB, RB

Function: This routine builds the major WQE for a multiple-line WTO request.

▶IEECVML4 -- Non-MCS Multiple-line WTO Processor (Load 3)

CSECT: IEECVML4

Exit: IEECVML1
IEECVML2
return to caller

Tables/Work Areas: XSA, UCM, WQE

Modules Used: XCTL, POST, ENQ/DEQ, GETMAIN

Function: This routine obtains WQE buffer space and queues the major and minor WQEs for multiple-line WTO requests (non-MCS systems).

▶IEECVML5 -- MCS Multiple-line WTO Processor (Load 2)

CSECT: IEECVML5

Exits: IEECVML6

Tables/Work Areas: WQE, UCM, XSA

Function: This routine builds minor WQEs for multiple-line WTOs.

▶IEECVML6 -- MCS Multiple-line WTO Processor (Load 3)

CSECT: IEECVML6

Exit: IEECVML7
IEECVML5
IEECVML3
return to caller

Modules Used: XCTL, POST, ENQ/DEQ, GETMAIN

Function: This routine obtains WQE buffer space and queues the major and minor WQEs for multiple-line WTO requests.

▶IEECVML7 -- MLWTO Load 7 (MCS)

Chart: GD

CSECT: IEECVML7

Exit: Return to SVC 3 issuer

Modules Used: IEAGENQ1 (DEQ)

Function: Removes request from the WQE resource request queue and sets return code.

▶IEEC2740 -- 2740 Processor (MCS only)

Chart: GT

CSECT: IEEC2740

Entry: IEECVPM

Exit: IEECMDSV (Device Interface)
IGG019MA (BTAM read/write)
IEECLCTX (Console Switch)

Modules Used: IEAATC00 (LOAD)
IEAAMS00 (GETMAIN)

Function: Performs open and close functions for the 2740 console.

TIMER SUPERVISION

IEA0RT00 -- TIME Service

Chart: HA

CSECT: IGC011

Exit: IEA0XE00 (Type 1 SVC Exit)

Function: Returns the date and time of day.

IEA0ST00 -- TTIMER Service

Chart: HC

CSECT: IGC047

Entry: IGC046

Exit: IEAATA00 (EXIT)

Modules Used: IEAQTD00 (Timer DEQ in IEA0TI02)

Function: Returns the amount of time remaining in a time interval or cancels a specified time interval.

IEA0ST00 -- STIMER Service

Chart: HD

CSECT: IGC047

Exit: IEAATA00 (EXIT)

Modules Used: IEAAWT00 (WAIT)
IEAAMS (GETMAIN)
IEAQTD00 (Timer DEQ in IEA0TI00)
IEAQTE00 (Timer ENQ in IEA0TI00)

Function: Creates a TQE for a specified time interval.

IEA0TI00 -- Timer SLIH

Chart: HB

CSECT: IEA0TI00

Exit: DISMISS (in IEAAIH00)

Modules Used: IEAAPT00 (POST)
IEA0EF00 (Stage 2 Exit Effector)
IEAQTE00 (Timer ENQ in IEA0TI00)
IEA0AB00 (ABTERM)

Function: Processes timer interruptions.

IEA0TI00 -- Timer ENQ/DEQ

Chart: HB

CSECT: IEA0TI00

Entry: IEAQTD00 (DEQ)
IEAQTE00 (ENQ)

Exit: Return to caller.

Function: Provides queue management for TQEs and the timer queue.

IEA0RT01 -- TIME Service with System/370 Time-of-Day Clock

Chart: HE

CSECT: IGC011

Exit: IEA0XE00 (Type 1 SVC Exit)

Modules Used: IEA0VL00 (Validity Check)

Function: Returns date and time of day.

IEA0ST01 -- TTIMER Service with System/370 Time-of-Day Clock

Chart: HF

CSECT: IGC047

Entry: IGC046

Exit: IEA0XE00 (Type 1 SVC Exit)

Modules Used: IEAAMS (FREEMAIN)
IEAQTD00 (Timer DEQ)

IEA0ST01 -- STIMER with System/370 Time-of-Day Clock

Chart: HG

CSECT: IGC047

Exit: IGC003 (EXIT)

Modules Used: IEAAMS (GETMAIN)
IEAQTE00 (Timer ENQ)
IEAAAWT00 (WAIT)

IEA0TI03 -- Timer SLIH with System/370 Time-of-Day Clock

Chart: HH

CSECT: IEA0TI00

Exit: DISMISS (in IEAAIH00)

Modules Used: IEAAPT00 (POST)
IEA0EF00 (Stage 2 Exit Effector)
IEAQTE00 (Timer ENQ)
IEA0AB00 (ABTERM)

IGC5A07B -- DIDOCS Options

Chart: JP

CSECT: IEECVETA

Exit: IGC5D07B (Message 1)
IGC5H07B (M85 I/O)
IGC5J07B (Roll Mode)
IGC5K07B (Timer Interpreter)
IGC5P07B (2250 I/O)
IGC5R07B (2260 I/O)

Tables/Work Areas: UCM

Function: Ensures validity of selected options and records the selected options in the Display Control Module.

IGC5C07B -- DIDOCS Asynchronous Error

Chart: KC

CSECT: IEECVETC

Exit: IGCXL07B (Console Switch)
IGC5D07B (Message 1)
IGC5E07B (Message 2)
IGC5H07B (M85 I/O)
IGC5K07B (Timer Interpreter)
IGC5P07B (2250 I/O)
IGC5R07B (2260 I/O)
IGC6G07B (Cleanup)

Tables/Work Areas: DCM

Function: This routine processes requests for reopening a device and asynchronous I/O errors for display consoles.

IGC5D07B -- DIDOCS Message 1

Chart: JH

CSECT: IEECVETD

Exit: IGC5H07B (M85 I/O)
IGC5P07B (2250 I/O)
IGC5R07B (2260 I/O)
IGC5107B (Processor)
IGC5307B (Display 2)

Tables/Work Areas: DCM

Function: Places error or informational messages into appropriate line.

IGC5E07B -- DIDOCS Message 2

Chart: JI

CSECT: IEECVETE

Exit: IGC5H07B (M85 I/O)
IGC5P07B (2250 I/O)
IGC5R07B (2260 I/O)

Tables/Work Areas: DCM

Function: Places error messages into the appropriate line.

IGC5F07B -- Light Pen/Cursor

Chart: JU

CSECT: IEECVETF

Exit: IGC5D07B (Message 1)
IGC5E07B (Message 2)
IGC5H07B (M85 I/O)
IGC5L07B (Status Display 1)
IGC5P07B (2250 I/O)
IGC5R07B (2260 I/O)
IGC5407B (Command)
IGC5807B (Delete 3)
IGC5907B (Delete 4)
IGC6A07B (PFK 1)

Tables/Work Areas: DCM

Function: Determines validity of light pen or cursor interruption.

IGC5G07B -- DIDOCS Open/Close

Chart: KK

CSECT: IEECVETG

Exit: IEECMDSV (Device Interface)
IGCXL07B (Console Switch)
IGC5K07B (Timer Interpreter)
IGC5107B (Processor)

Tables/Work Areas: DCM

Function: Performs open and close functions for specified consoles.

IGC5H07B -- DIDOCS Model 85 I/O

Chart: JG

CSECT: IEECVETH

Exit: IGC5F07B (Light Pen/Cursor)
IGC5107B (Processor)

Tables/Work Areas: DCM

Function: Initiates requested I/O to the Model 85 Operator's Console.

IGC5J07B -- DIDOCS Roll Mode

Chart: JN

CSECT: IEECVETJ

Exit: IGC5K07B (Timer Interpreter)
IGC5107B (Processor)
IGC5207B (Display 1)

Tables/Work Areas: DCM

Function: Rolls the screen buffer.

IGC5K07B -- DIDOCS Timer Interpreter

Chart: JV

CSECT: IEECVETK

Exit: IGC5D07B (Message 1)
IGC5E07B (Message 2)
IGC5G07B (Open/Close)
IGC5H07B (M85 I/O)
IGC5P07B (2250 I/O)
IGC5R07B (2260 I/O)
IGC5107B (Processor)
IGC5207B (Display 1)

Tables/Work Areas: DCM

Function: Determines the time intervals for rolling each console.

IGC5P07B -- DIDOCS 2250 I/O 1

Chart: JC

CSECT: IEECVETP

Exit: IGC5F07B (Light Pen/Cursor)
IGC5Q07B (2250 I/O 2)
IGC5107B (Processor)

Tables/Work Areas: DCM

Function : Initiates I/O to a 2250 console.

IGC5Q07B -- DIDOCS 2250 I/O 2

Chart: JD

CSECT: IEECVETQ

Exit: IGC5107B (Processor)

Tables/Work Areas: DCM

Function: Initiates I/O to a 2250 console.

IGC5R07B -- DIDOCS 2260 I/O 1

Chart: JE

CSECT: IEECVETR

Exit: IGC5F07B (Light Pen/Cursor)
IGC5107B (Processor)
IGC5407B (Command)

Tables/Work Areas: DCM

Function: Initiates I/O to a 2260 console.

IGC5Z07B -- DIDOCS Processor 1, Load 2

Chart: JB

CSECT: IEECVETZ

Exit: IGC5107B (Device-dependent I/O routine)
IGC5G07B (Open/Close routine)
IGC5H07B (Model 85 I/O routine)
IGC5P07B (2250 I/O routine)
IGC5R07B (2260 I/O routine)

Tables/Work Areas: DCM

Function: Continues processing begun by Processor 1, Load 1.

IGC5107B -- DIDOCS Processor 1, Load 1

Chart: JA

CSECT: IEECVET1

Exit: IEECMAWR (Communications Task Router)
IGC5C07B (Asynchronous Error)
IGC5D07B (Message 1)
IGC5F07B (Light Pen/Cursor)
IGC5G07B (Open/Close)
IGC5H07B (M85 I/O)
IGC5J07B (Roll Mode)
IGC5K07B (Timer Interpreter)
IGC5P07B (2250 I/O)
IGC5R07B (2260 I/O)
IGC5207B (Display 1)
IGC5407B (Command)
IGC5707B (Delete 2)
IGC5907B (Delete 4)
IGC6M07B (Status Display Interface 2)
IGC6107B (Processor 0)
IGC6A07B (PFK 1 Routine)

Tables/Work Areas: DCM

Function: Determines processing required and passes control to the appropriate module.

IGC5207B -- DIDOCS Display 1

Chart: JK

CSECT: IEECVET2

Exit: IGC5D07B (Message 1)
IGC5M07B (Display Status 2)
IGC5107B (Processor)
IGC5307B (Display 2)
IGC5707B (Delete 2)
IGC5907B (Delete 4)
IGC6207B (Display 3)

Tables/Work Areas: DCM

Function: Displays messages and marks those that require action or may be deleted.

IGC5307B -- DIDOCS Display 2

Chart: JL

CSECT: IEECVET3

Exit: IGC5D07B (Message 1)
IGC5107B (Processor)
IGC5207B (Display 1)
IGC5707B (Delete 2)
IGC5907B (Delete 4)
IGC6207B (Display 3)

Tables/Work Areas: DCM

Function: Indicates accepted reply message by vertical bar.

IGC5407B -- DIDOCS Command

Chart: JO

CSECT: IEECVTE4

Exit: IGC5A07B (Options)
IGC5C07B (Asynchronous Error)
IGC5D07B (Message 1)
IGC5E07B (Message 2)
IGC5H07B (M85 I/O)
IGC5L07B (Status Display 1)
IGC5P07B (2250 I/O)
IGC5R07B (2260 I/O)
IGC5607B (Delete 1)
IGC5807B (Delete 3)
IGC5907B (Delete 4)

Tables/Work Areas: DCM

Function: Determines command type; processes CANCEL commands.

IGC5607B -- DIDOCS Delete 1

Chart: JQ

CSECT: IEECVET6

Exit: IGC5D07B (Message 1)
IGC5E07B (Message 2)
IGC5H07B (M85 I/O)
IGC5P07B (2250 I/O)
IGC5R07B (2260 I/O)
IGC5907B (Delete 4)

Tables/Work Areas: DCM

Function: Analyzes requests for deletion of messages.

IGC5707B -- DIDOCS Delete 2

Chart: JR

CSECT: IEECVET7

Exit: IGC5H07B (M85 I/O)
IGC5P07B (2250 I/O)
IGC5R07B (2260 I/O)
IGC5107B (Processor)
IGC5907B (Delete 4)

Tables/Work Areas: DCM

Function: Processes intervention-required messages.

IGC5807B -- DIDOCS Delete 3

Chart: JS

CSECT: IEECVET8

Exit: IGC5D07B (Message 1)
IGC5E07B (Message 2)
IGC5H07B (M85 I/O)
IGC5P07B (2250 I/O)
IGC5R07B (2260 I/O)
IGC5907B (Delete 4)

Tables/Work Areas: DCM

Function: Analyzes deletion of messages requested by light pen, cursor, or K E,SEG command

IGC5907B -- DIDOCS Delete 4

Chart: JT

CSECT: IEECVET9

Exit: IGC5D07B (Message 1)
IGC5H07B (M85 I/O)
IGC5P07B (2250 I/O)
IGC5R07B (2260 I/O)
IGC5207B (Display 1)

Tables/Work Areas: DCM

Function: Removes messages for regular or automatic delete requests.

IGC6A07B -- DIDOCS PFK 1

Chart: KL

CSECT: IEECVFTA

Exit: IGC6D07B (Message 3)
IEECVET1 (Processor 1, Load 1)

Tables/Work Areas: DCM

Function: Analyzes and enters commnnds requested by a PFK key or a selector pen detection of a displayed PFK number.

IGC6B07B -- DIDOCS PFK 2

Chart: KM

CSECT: IEECVFTB

Exit: IGC6D07B (Message 3)
IEECVFTD (PFK 2)
IEECVET1 (Processor 1, Load 1)

Tables/Work Areas: DCM

Function: Processes operator requests to define or redefine the commands associated with a PFK key; to write or erase the PFK display line on the screen.

IGC6D07B -- DIDOCS Message 3

Chart: JJ

CSECT: IEECVFTD

Exit: IEECVETH (Model 85 I/O Routine)
IEECVETP (2250 I/O Routine)
IEECVETR (2260 I/O Routine)

Tables/Work Areas: DCM

Function: Writes error messages in the instruction line of display consoles.

IGC6G07B -- DIDOCS Cleanup Routine

Chart: KN

CSECT: IEECVFTG

Exit: IGC5C07B (Asynchronous Error)
Return to Caller

Modules Used: FREEMAIN

Tables/Work Areas: UCM, Resident portion of the DCM (including the SACBs), Transient portion of the DCM

Function: This routine dequeues status displays and reinitializes the SACBs when a device status change is requested or when an asynchronous error occurs.

IGC6L07B -- DIDOCS Status Display Interface 1

Chart: KD

CSECT: IEECVFTL

Exit: IGC5D07B (Message 1)
IGC5H07B (M85 I/O Routine)
IGC5P07B (2250 I/O Routine)
IGC5R07B (2260 I/O Routine)
IGC5107B (Processor 1, Load 1)
IGC5207B (Display 1)
IGC6207B (Display 3)

Modules Used: XCTL

Tables/Work Areas: CXSA, UCM, WQE, DCM

Function: This routine displays multiple-line WTOs in the message area of a display console screen.

IGC6M07B -- DIDOCS Status Display Interface 2

Chart: KE

CSECT: IEECVFTM

Exit: IGC6O07B (Status Display Interface 4)
IGC6Q07B (Status Display Interface 6)
IGC6P07B (Status Display Interface 5)
IGC5107B (Processor 1, Load 1)

Tables/Work Areas: UCM, Resident portion of the DCM (including the SACBs), Transient portion of the DCM

Function: This routine is the control module for out-of-line status displays (status displays to be presented in a display area on a display console screen). It initializes the screen area control blocks (SACBs) in the DCM and routes control to the appropriate module to process the status display.

IGC6N07B -- DIDOCS Status Display Interface 3

Chart: KF

CSECT: IEECVFTN

Exit: IGC5H07B (M85 I/O Routine)
IGC5P07B (2250 I/O Routine)
IGC5R07B (2260 I/O Routine)

Modules Used: SVC 34, XCTL

Function: This routine processes CONTROL commands affecting out-of-line status displays (CONTROL D,H; CONTROL D,U; CONTROL D,F)

IGC6O07B -- DIDOCS Status Display Interface 4

Chart: KG

CSECT: IEECVFTO

Exit: IGC5H07B (M85 I/O Routine)
IGC5P07B (2250 I/O Routine)
IGC5R07B (2260 I/O Routine)

Modules Used: SVC 34, XCTL

Function: This routine moves the display into the DCM and passes control to the I/O routine to write the display to the screen (for displays that overlay other messages on the screen).

IGC6P07B -- DIDOCS Status Display Interface 5

Chart: KH

CSECT: IEECVFTP

Exit: IGC5H07B (M85 I/O Routine)
IGC5P07B (2250 I/O Routine)
IGC5R07B (2260 I/O Routine)
IGC6M07B (Status Display Interface 2)

Modules Used: SVC 34, XCTL

Tables/Work Areas: UCM, DCM, WQE

Function: This routine handles CONTROL commands affecting out-of-line status displays (CONTROL E,D) and the STOPMN A command.

IGC6Q07B -- DIDOCS Status Display Interface 6

Chart: KI

CSECT: IEECVFTQ

Exit: IGC5H07B (M85 I/O Routine)
IGC5P07B (2250 I/O Routine)
IGC5R07B (2260 I/O Routine)
IGC6M07B (Status Display Interface 2)

Tables/Work Areas: UCM, Resident portion of the DCM (including the SACBs), Transient portion of the DCM

Function: This routine moves the display into the DCM message area buffer and passes control to the I/O routine to write the display to the screen. This routine is used for displays that do not overlay other messages on the screen.

IGC6R07B -- 2260 I/O 2

Chart: JF

CSECT: IEECVFTR

Exit: IGC6G07B (Cleanup)
IGC5K07B (Timer/Interpreter)
IGC5R07B (2260 I/O 1)
IEECVET1 (Processor 1, Load 1)

Tables/Work Areas: DCM

Function: Switches console status from full-capability mode to output-only mode, from one output-only mode to the other, and from output-only mode to full-capability mode for 2260 operator's consoles.

IGC6T07B -- DIDOCS Status Display Interface 7

Chart: KJ

CSECT: IEECVFTT

Exit: IGC5H07B (M85 I/O Routine)
IGC5P07B (2250 I/O Routine)
IGC5R07B (2260 I/O Routine)

Modules Used: XCTL

Tables/Work Areas: UCM, DCM

Function: This routine blanks message area lines below status displays.

IGC6Z07B -- Processor 0, Load 2

Chart: KB

CSECT: IEECVFTZ

Exit: IGC5107B (Processor 1, Load 1)

Tables/Work Areas: DCM, UCM Entry

Function: Checks I/O complete for transient DCM switching.

IGC6107B -- Processor 0, Load 1

Chart: KA

CSECT: IEECVFT1

Exit: IGC5107B (Processor 1, Load 1)
IGC6Z07B (Processor 0, Load 2)

Tables/Work Areas: DCM

Function: Brings transient portion of DCM into main storage.

IGC6207B -- DIDOCS Display 3

Chart: JM

CSECT: IEECVFT2

Exit: IGC5307B (Display 2)
IGC5207B (Display 1)
IGC6L07B (Status Display Interface 1)

Tables/Work Areas: CXSA, WQE, DCM, UCM

Function: This routine continues processing from Display 2. It displays warning messages and marks action and deletable messages.

The flowcharts are arranged in the order in which the modules are described in this publication. Each logical flowchart is assigned an alphabetic ID by which it is referenced elsewhere in the flowcharts, text, and tables. Each logical flowchart may be comprised of several pages of flowcharts which together describe the sequence of processing within the module. References to the pages that make up the logical flowchart appear in the following format:

AA/N/BB

where:

AA
is the logical chart ID.

N
is the sequential page number within the logical flowchart AA.

BB
is the block reference on page N of logical chart AA.

The first letter of each flowchart indicates the type of supervisor module being described:

A -- Interruption Supervision
B -- Task Supervision
C,D -- Task Termination
E -- Contents Supervision
F -- Main Storage Supervision
G -- Communications Task
H -- Timer Supervision
I -- Overlay Supervision
J,K -- Device Independent Display Operator Console Support
L -- Checkpoint/Restart
M -- Special Features

Chart AA. SVC First-Level Interruption Handler and Type 1 SVC Exit

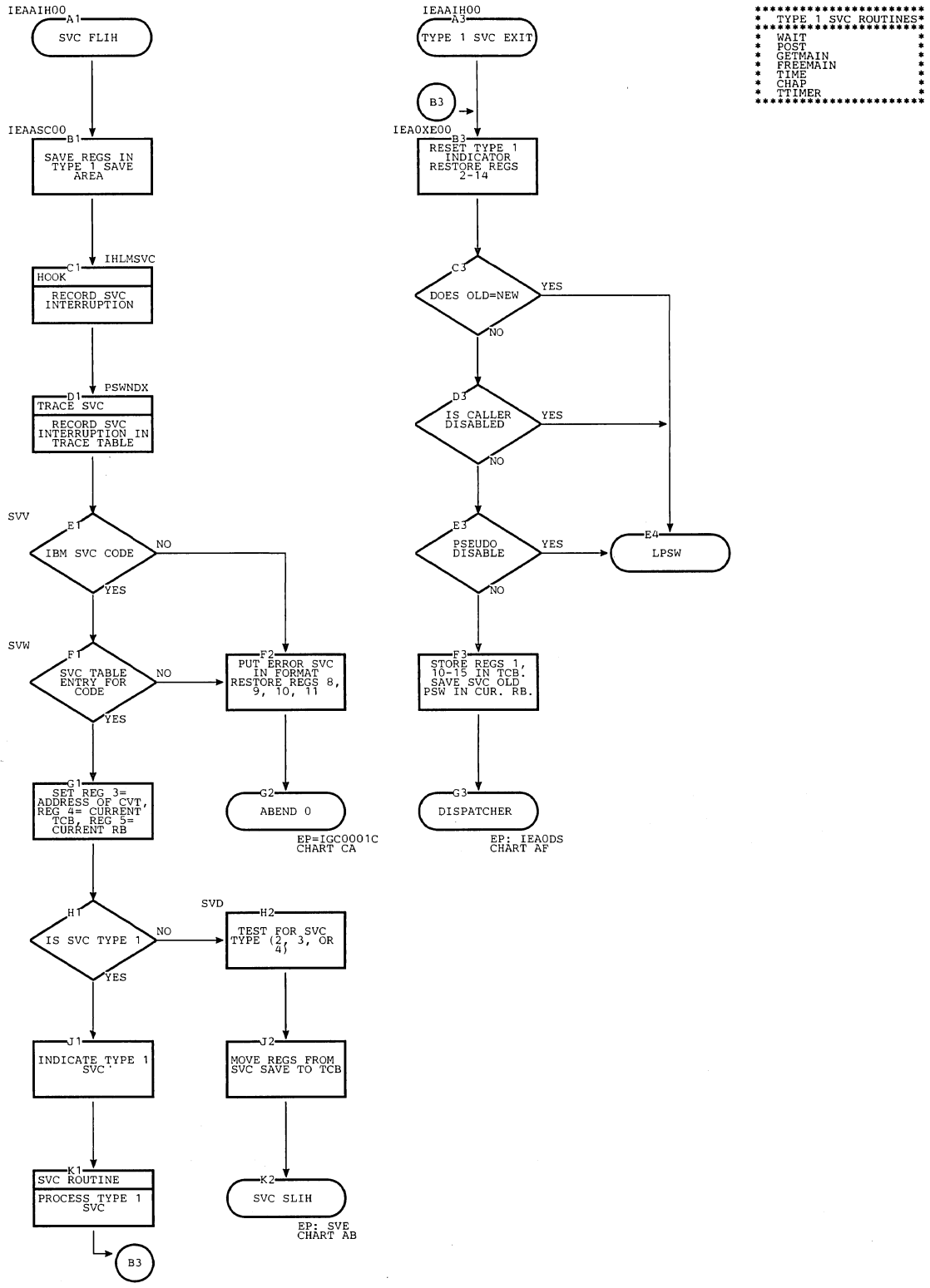


Chart AB. SVC Second-Level Interruption Handler (Part 1 of 2)

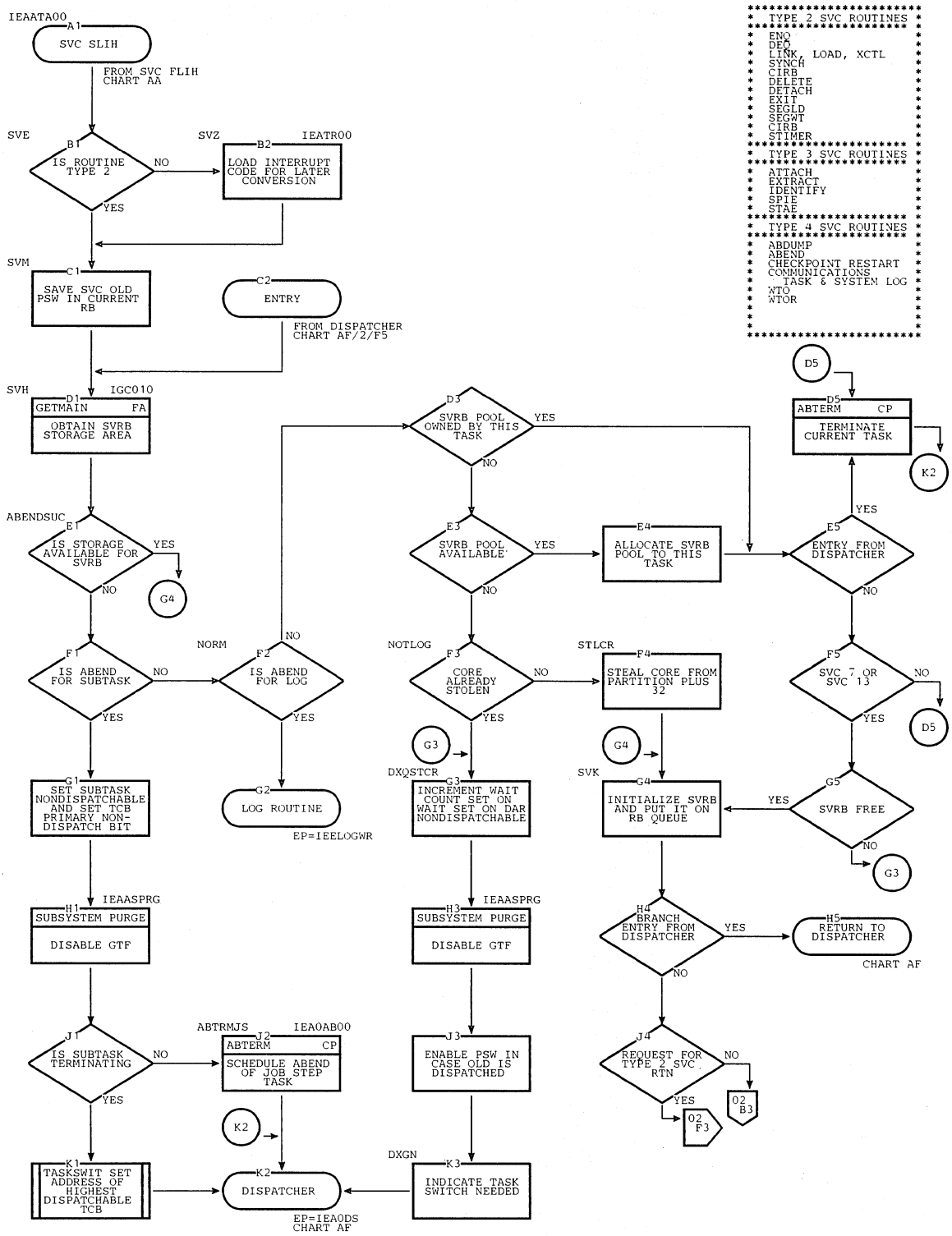


Chart AB. SVC Second-Level Interruption Handler (Part 2 of 2)

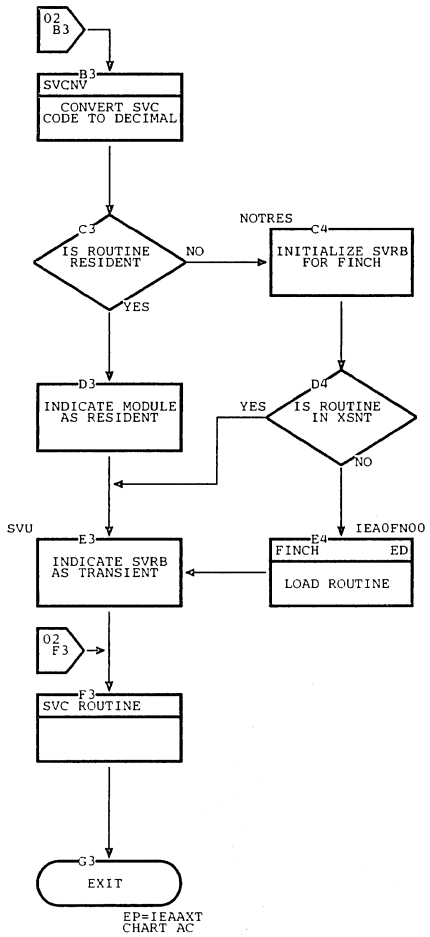


Chart AC. EXIT (Type 2, 3, and 4 SVC Routines) (Part 1 of 2)

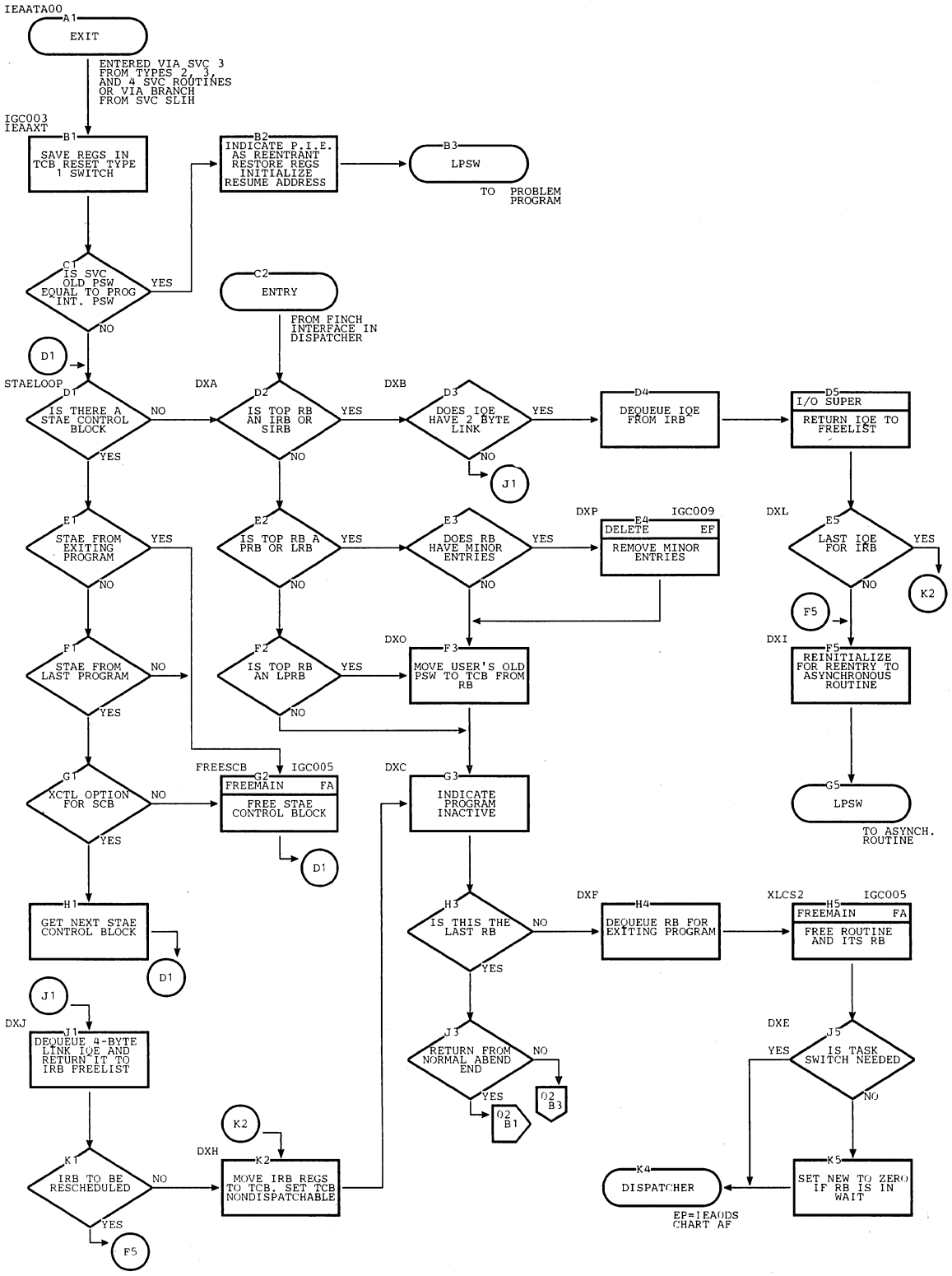


Chart AC. EXIT (Type 2, 3, and 4 SVC Routines) (Part 2 of 2)

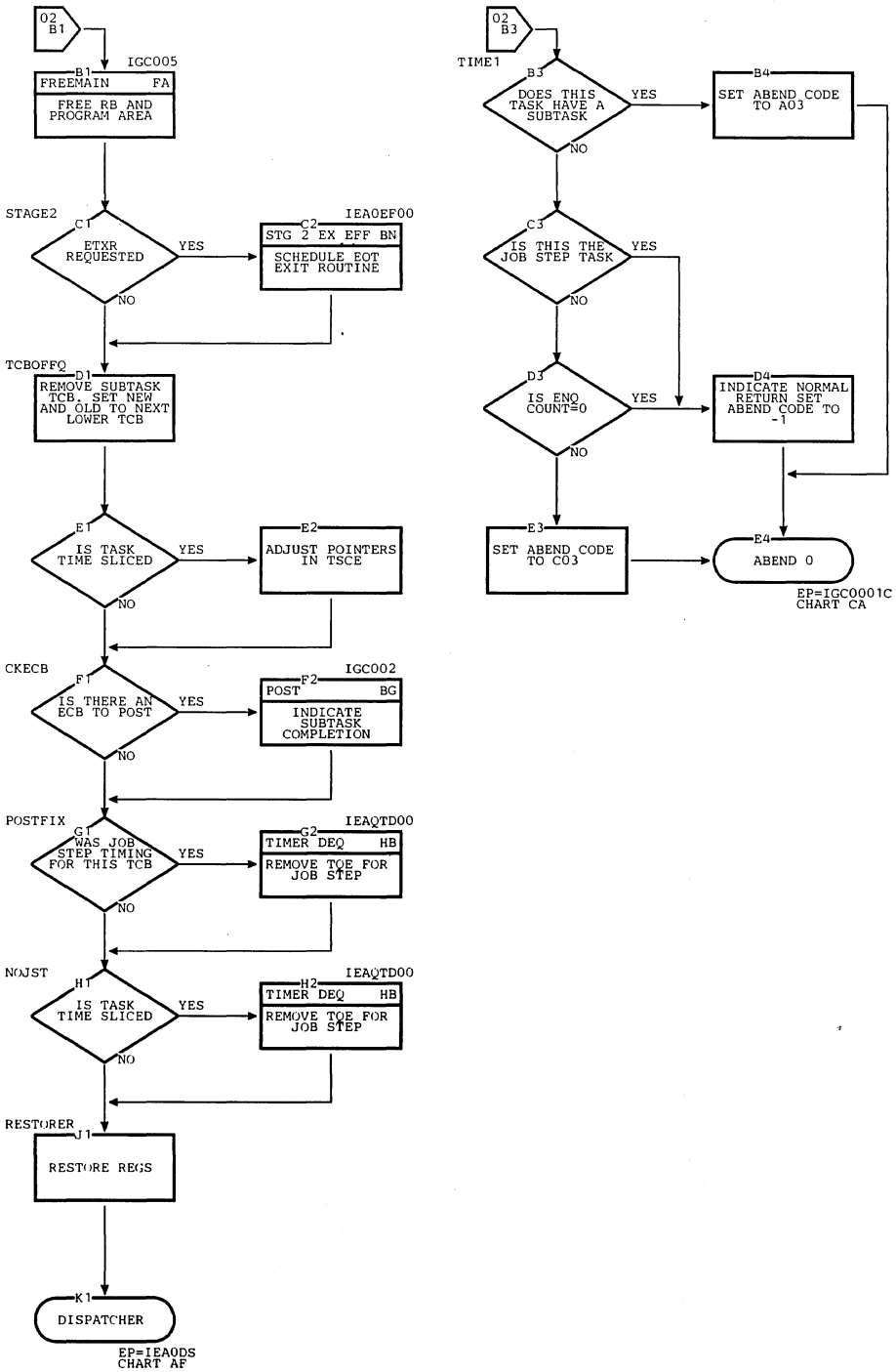


Chart AD. Input/Output First-Level Interruption Handler and Timer/External First-Level Interruption Handler

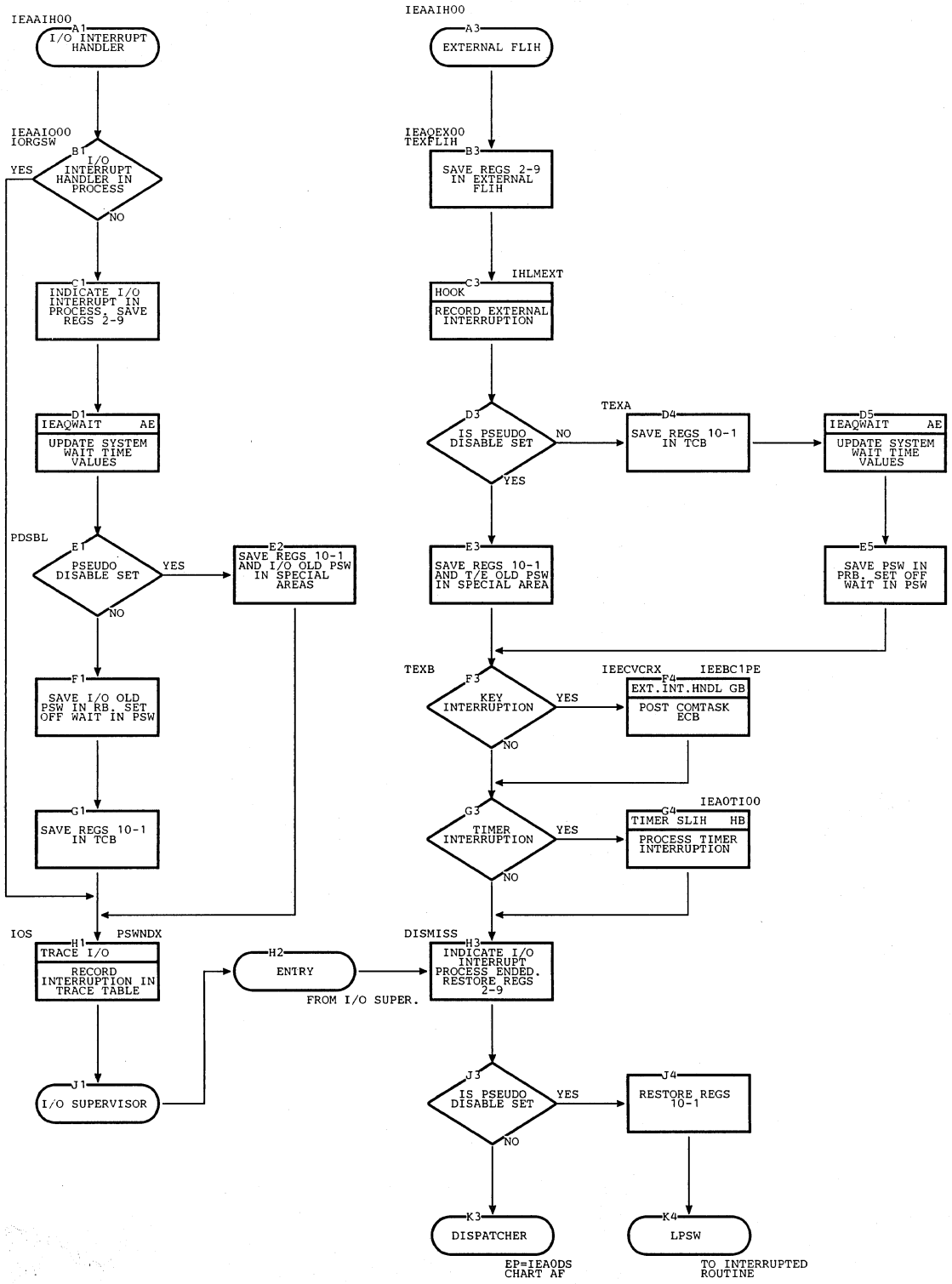


Chart AE. Program-Check First-Level Interruption Handler and System Wait Time Collection Subroutine

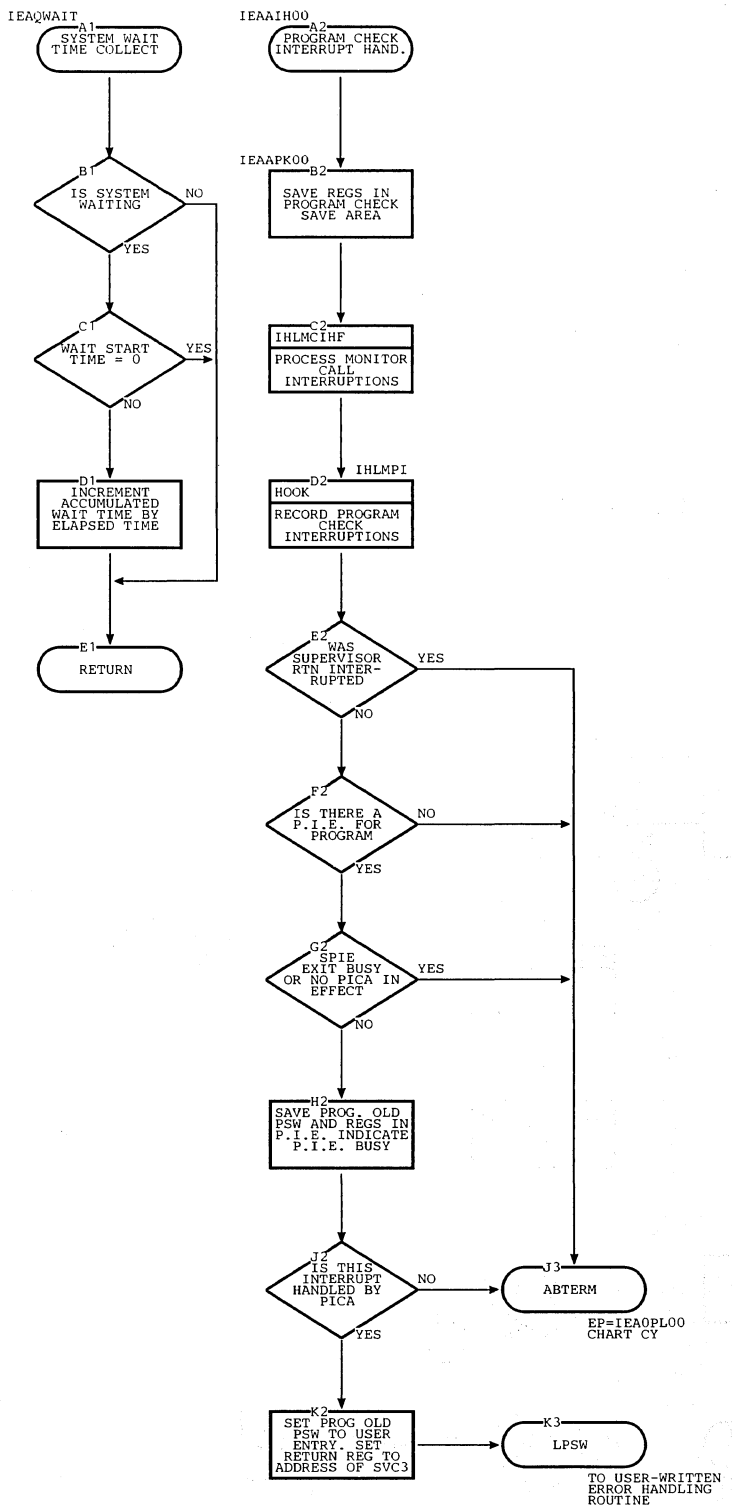


Chart AF. Dispatcher (Part 1 of 2)

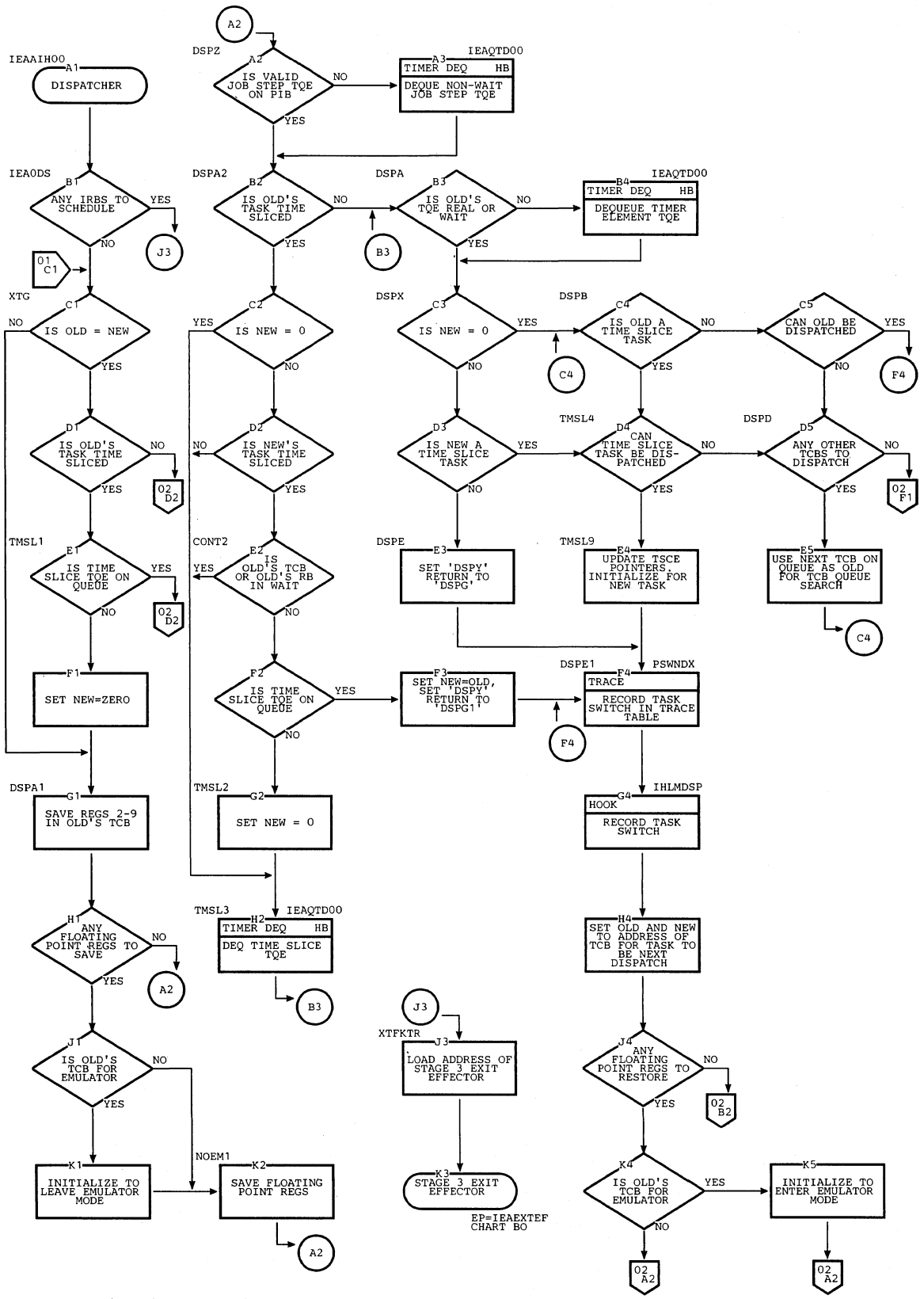


Chart AF. Dispatcher (Part 2 of 2)

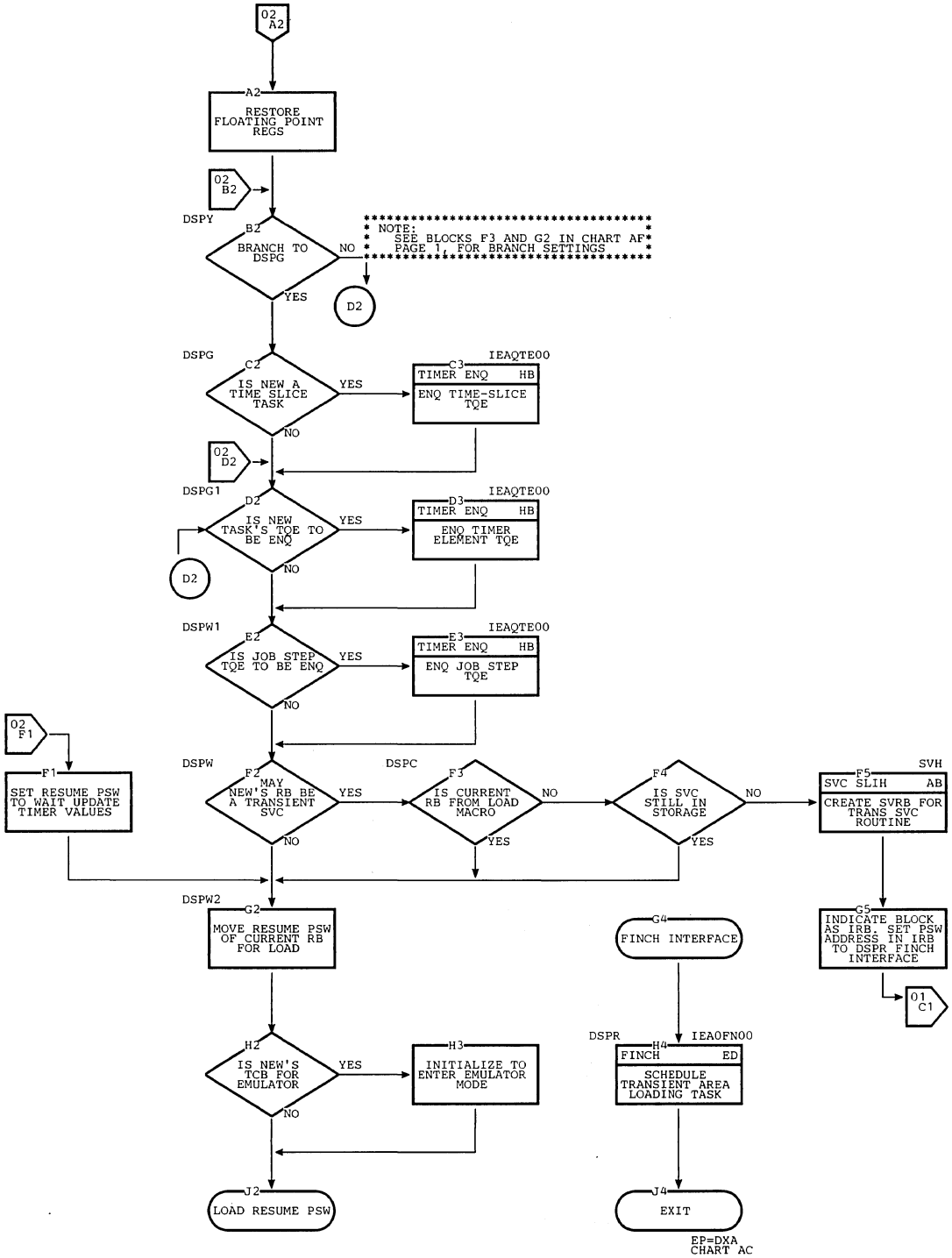


Chart AG. SER0 Routine

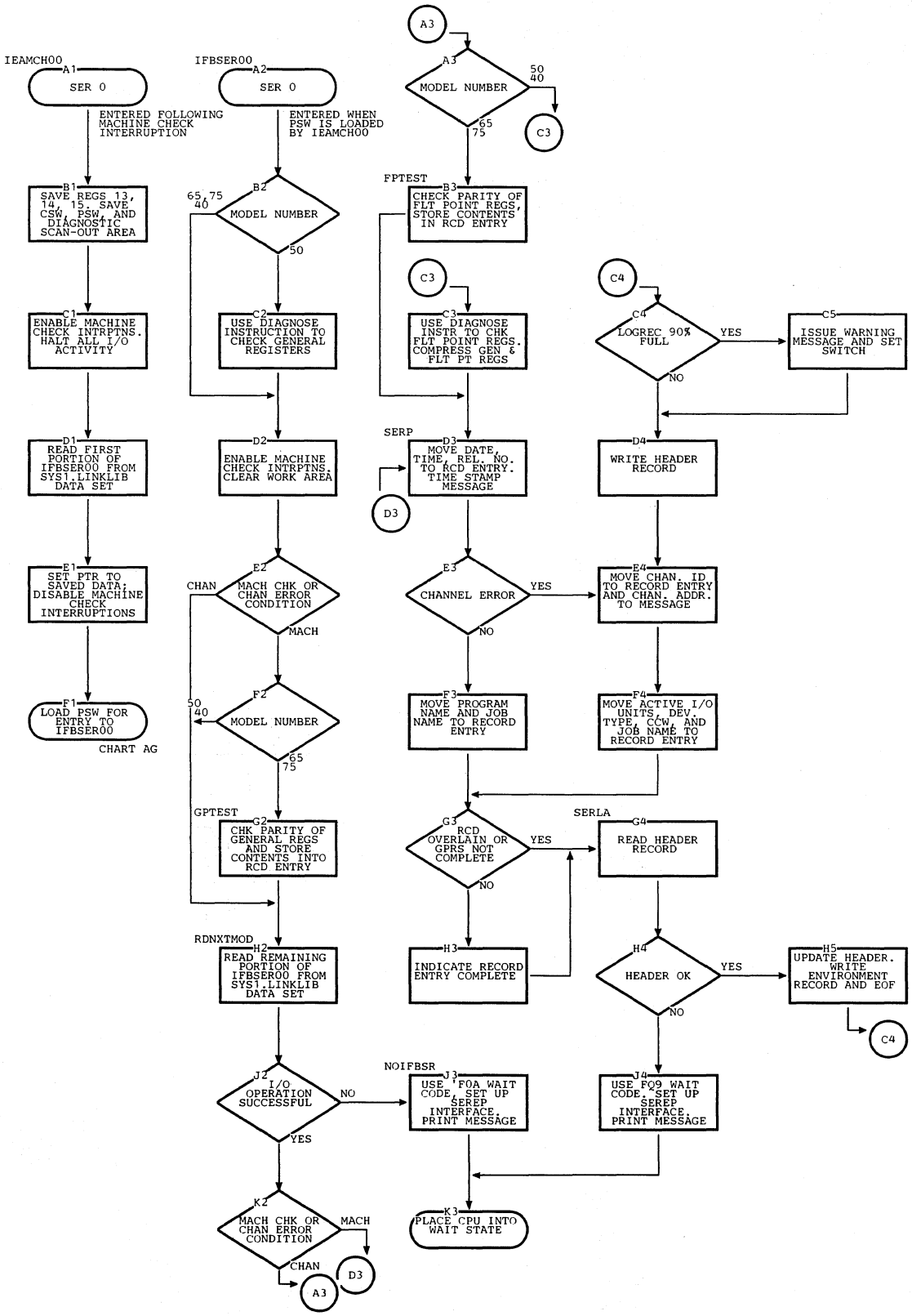


Chart AH. SER1 Routine

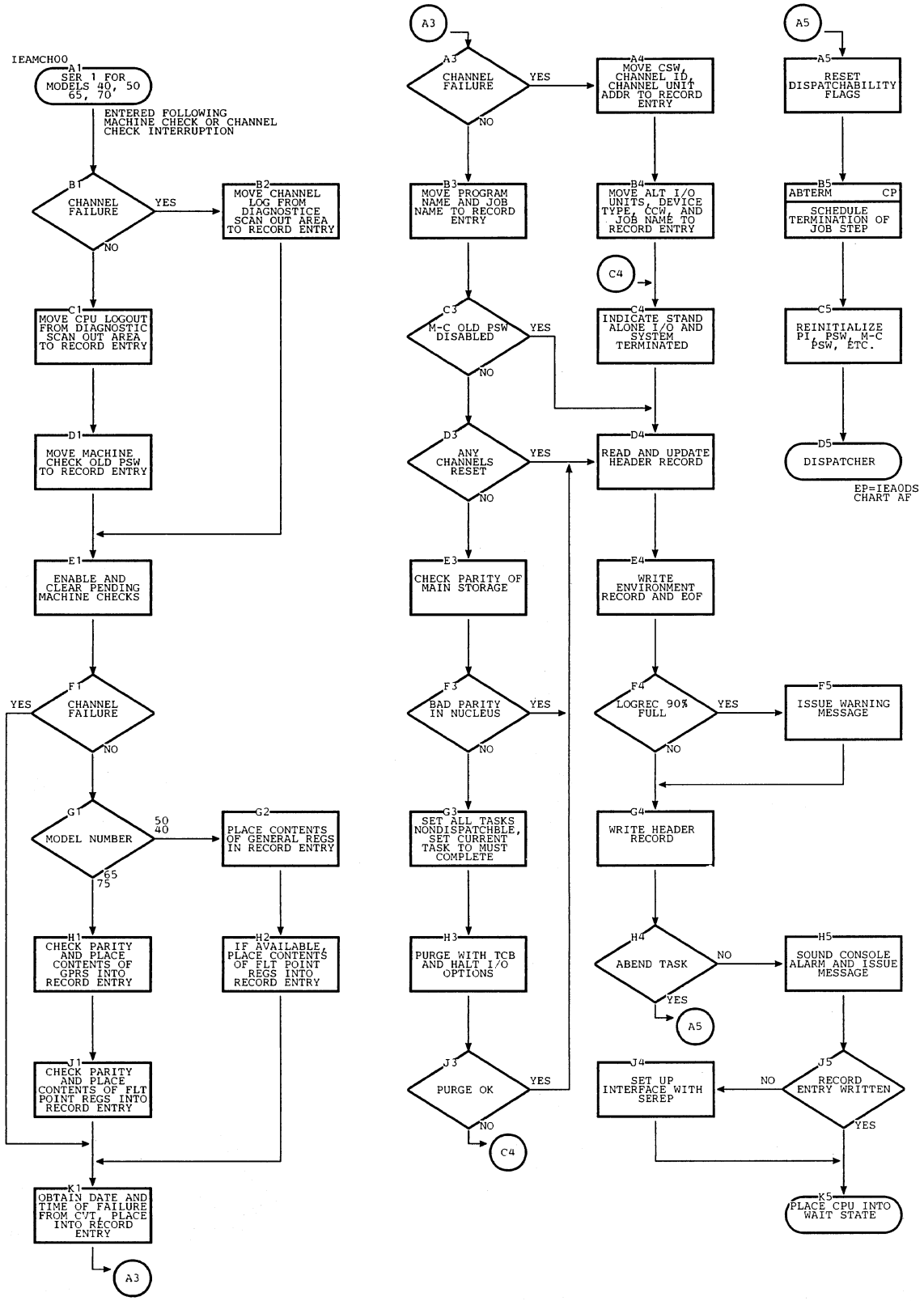


Chart AI. Extended SVC Router Routines

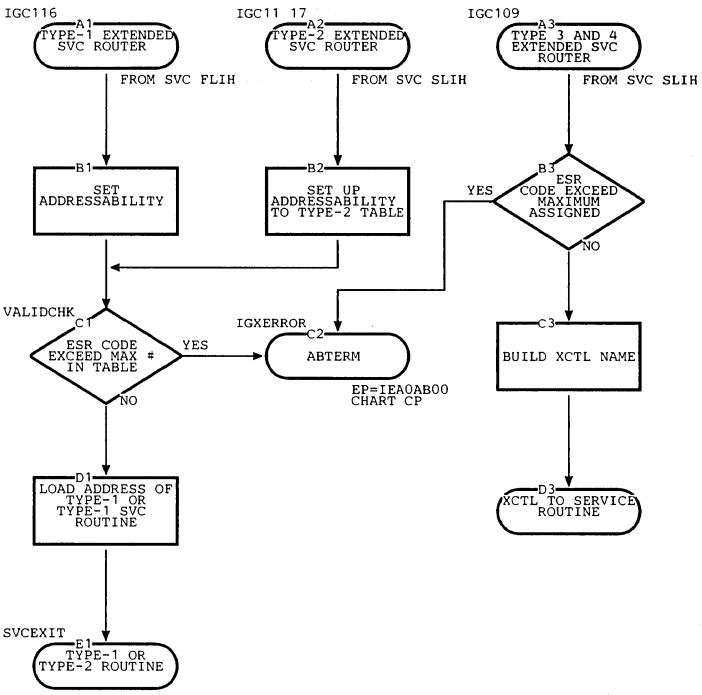


Chart BA. ATTACH with the Subtasking Option (Part 1 of 2)

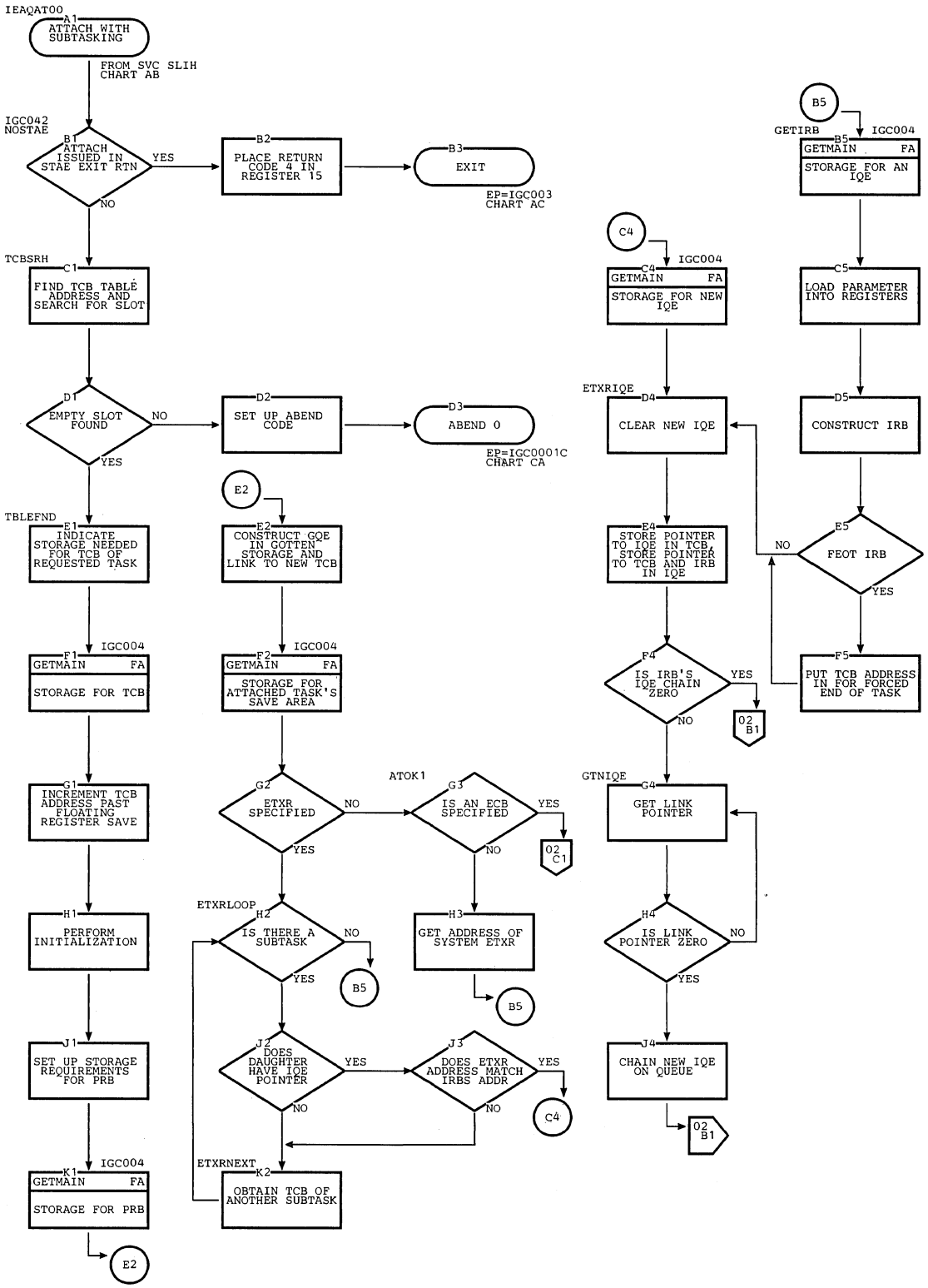


Chart BA. ATTACH with the Subtasking Option (Part 2 of 2)

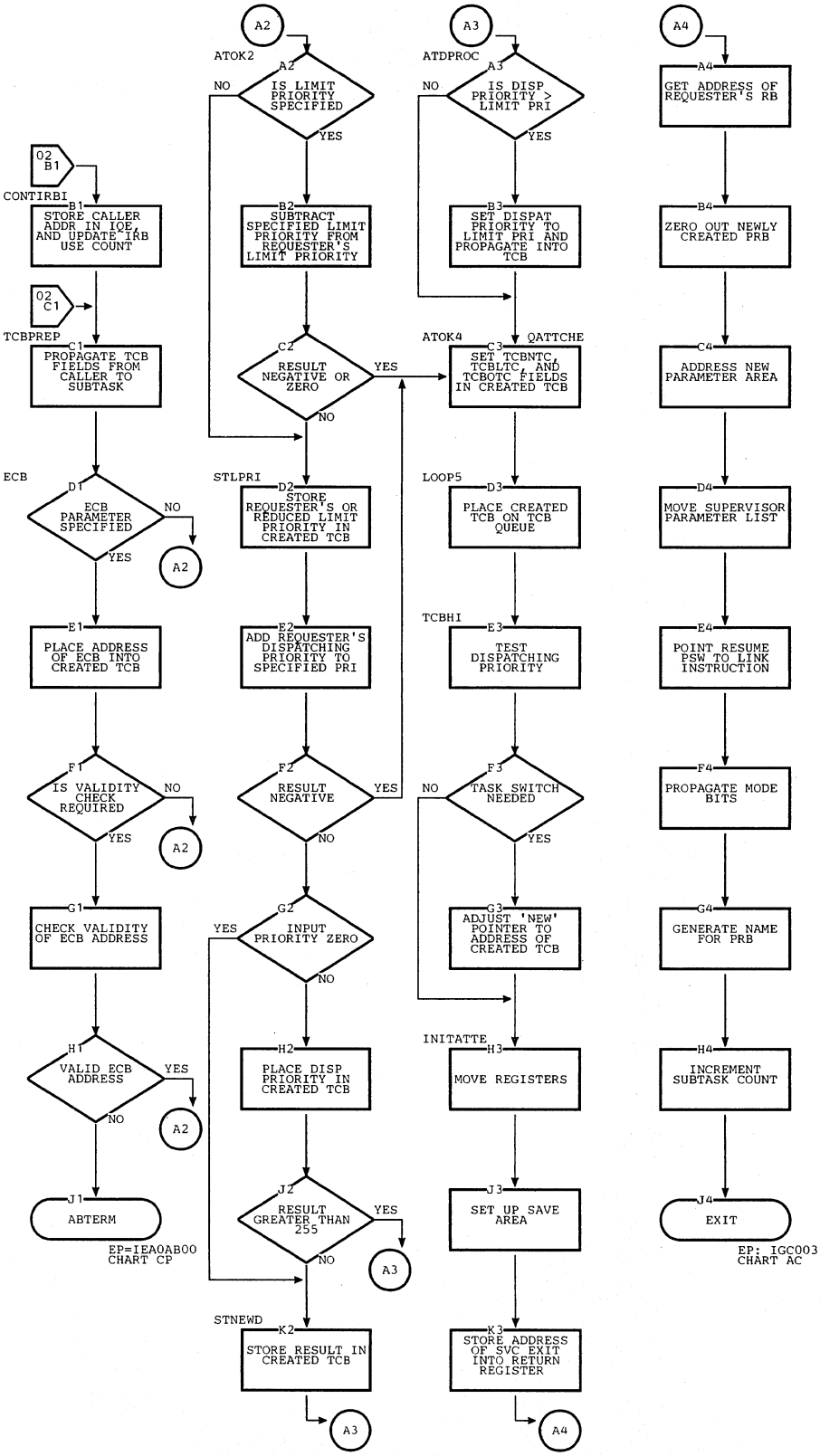


Chart BB. DETACH

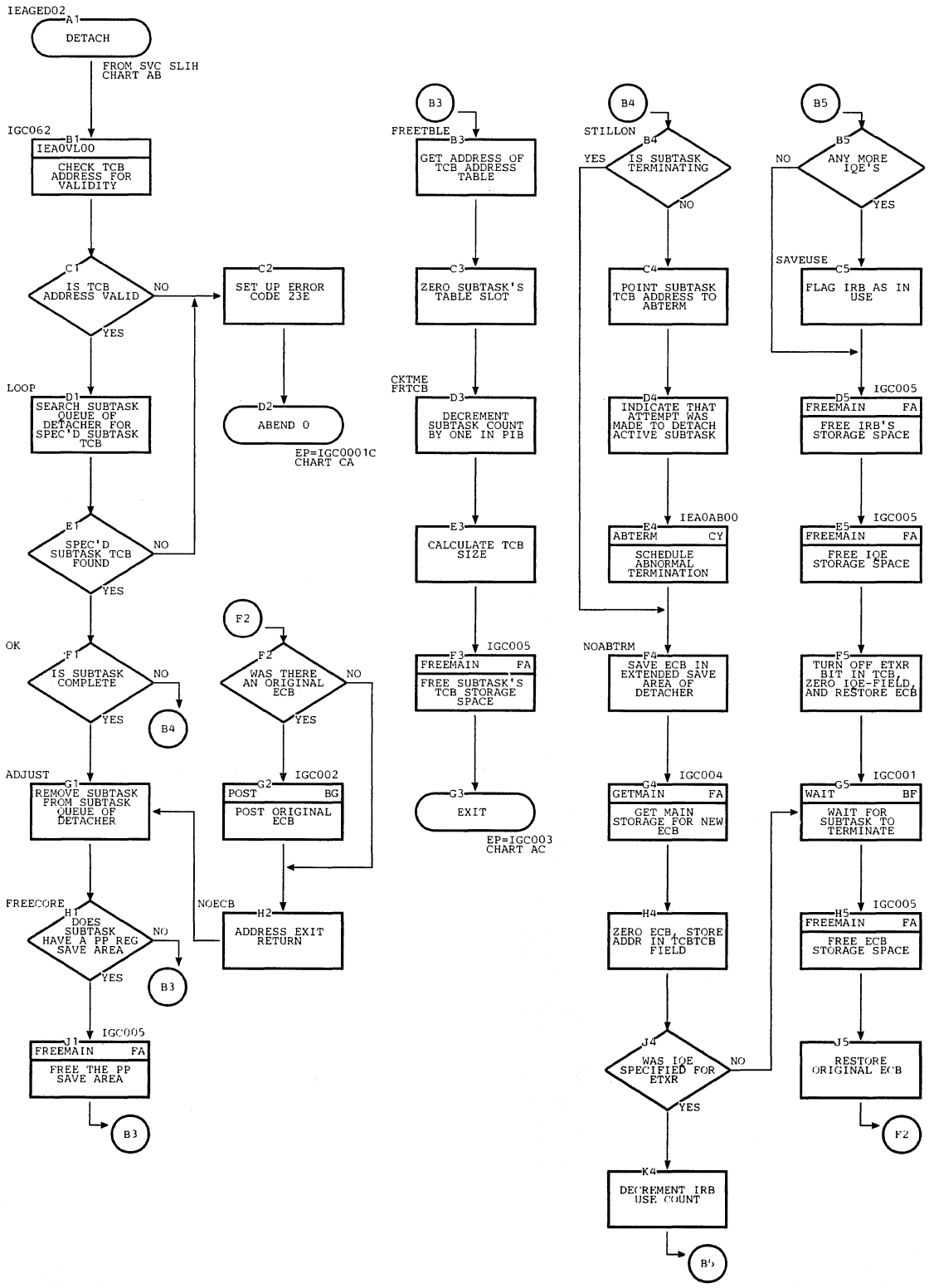


Chart BC. CHAP

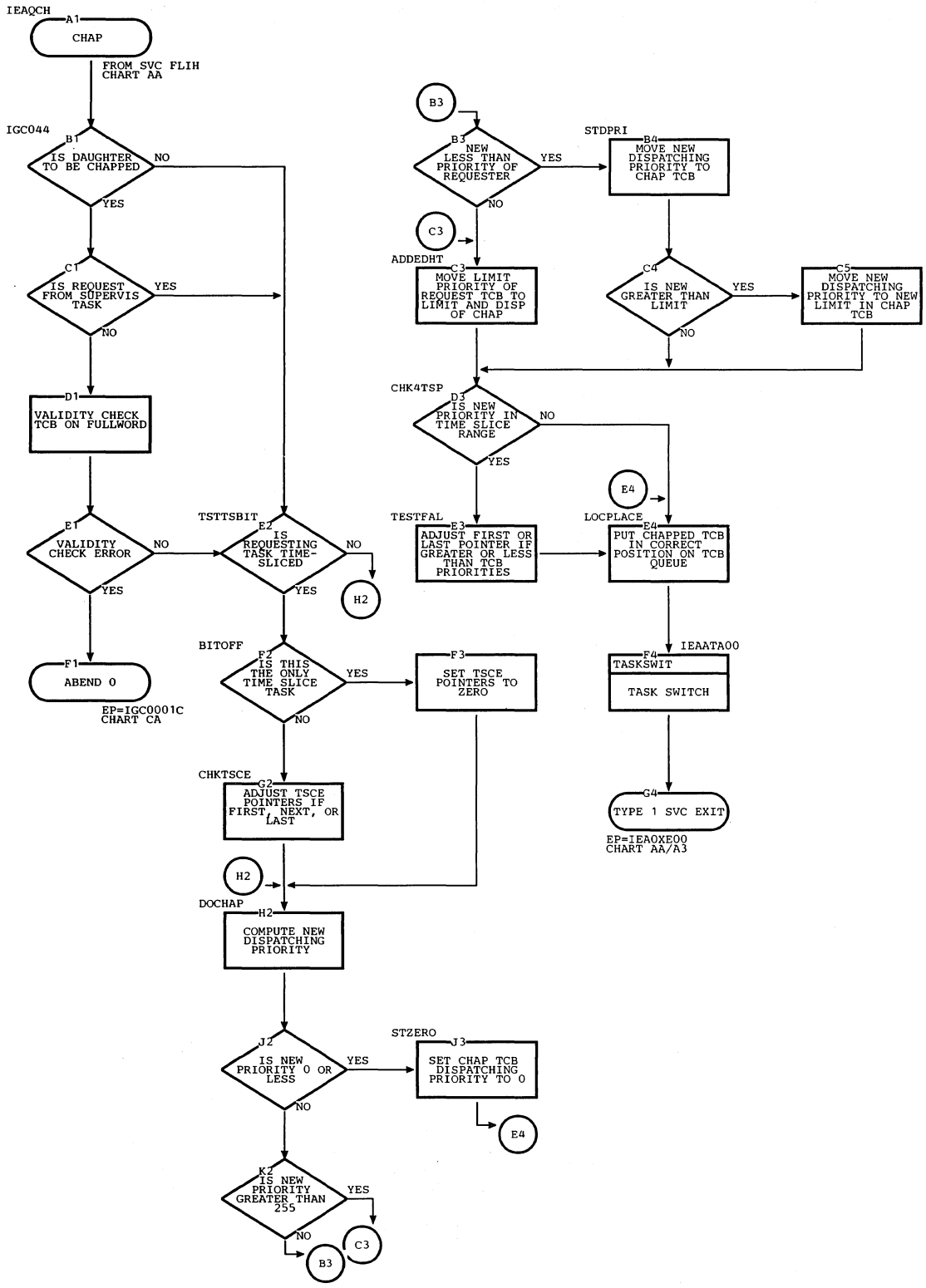


Chart BD. EXTRACT without the Subtasking Option

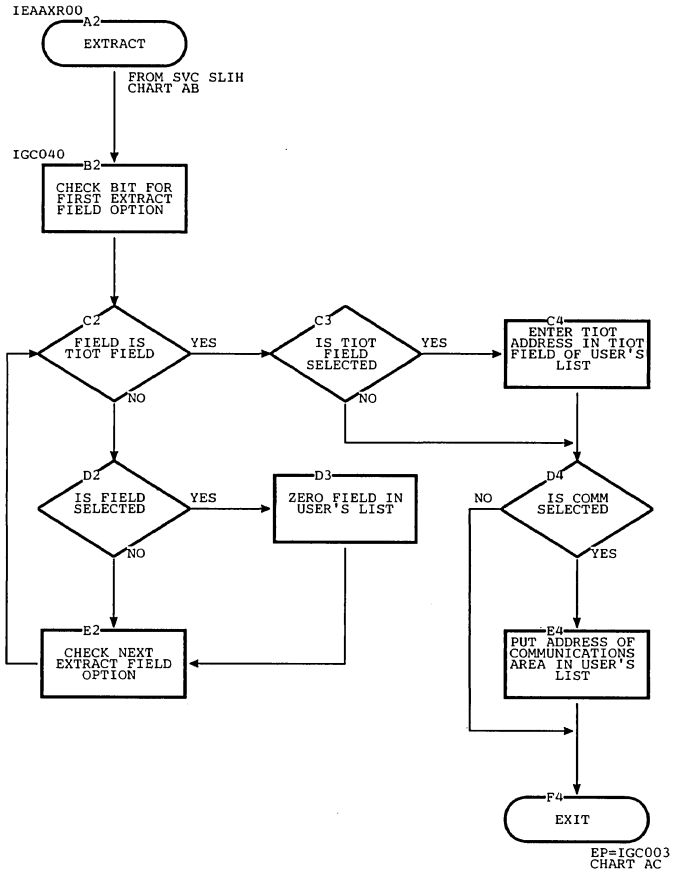


Chart BE. EXTRACT with the Subtasking Option

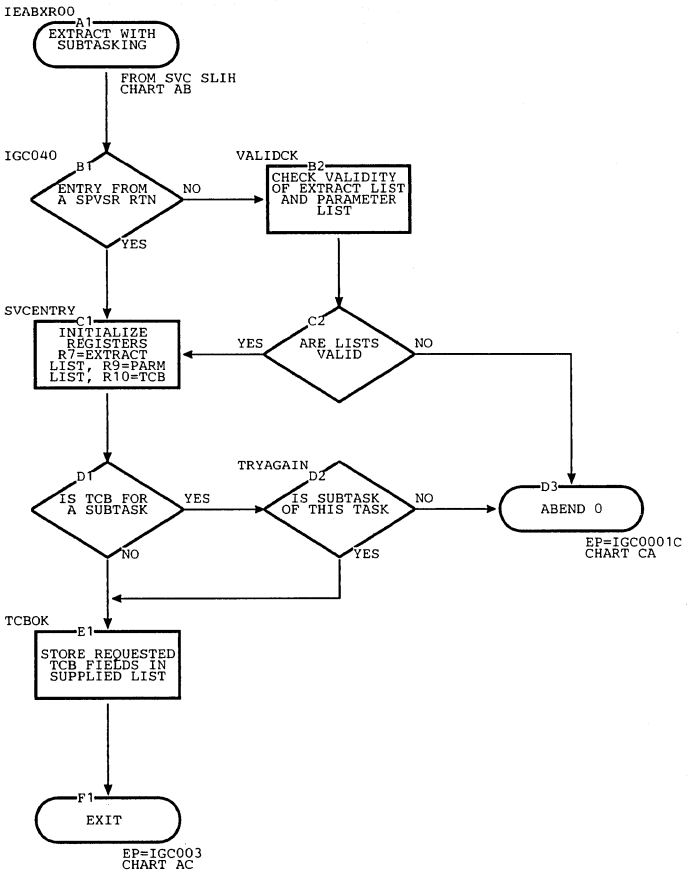


Chart BF. WAIT

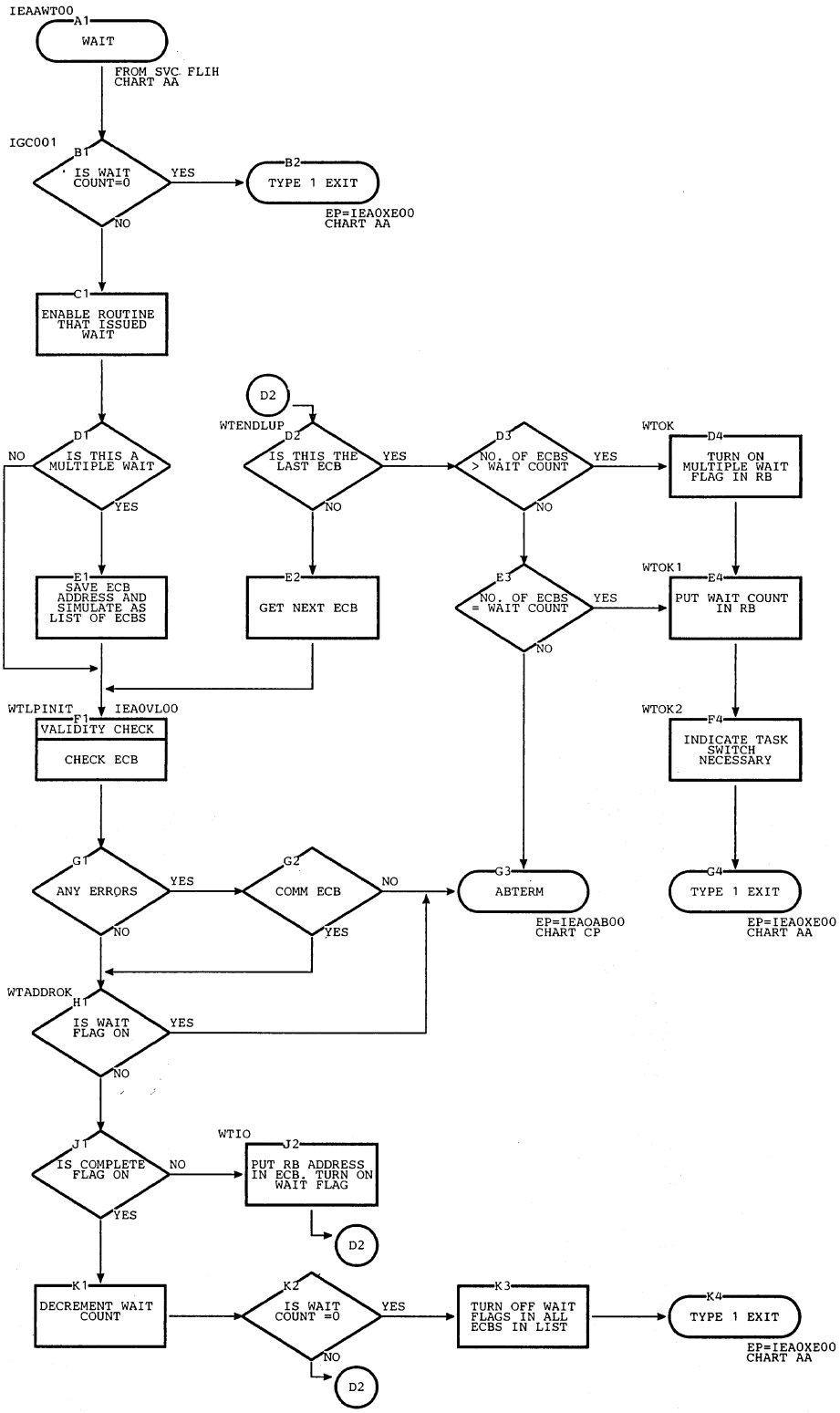


Chart BG. POST

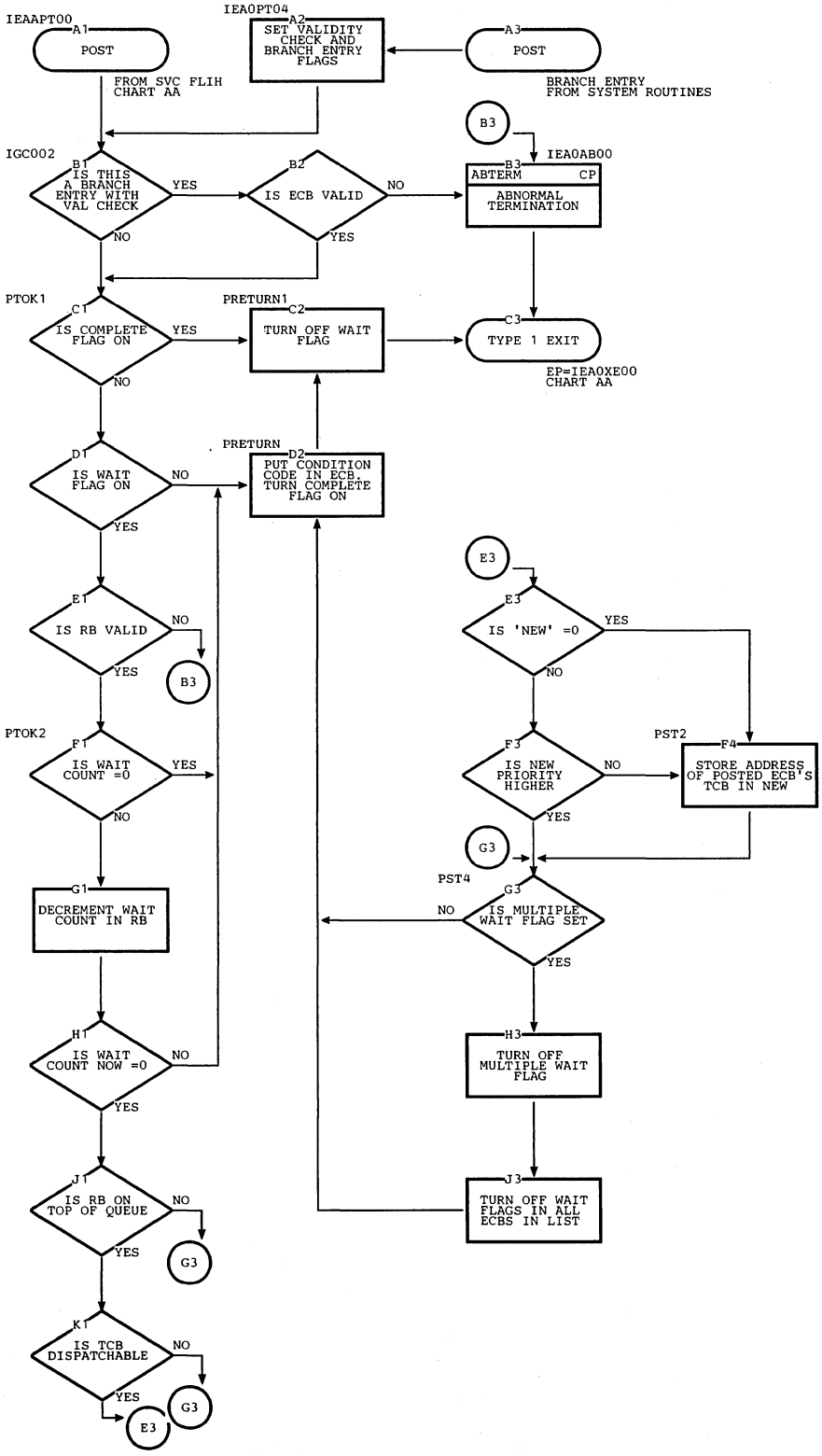


Chart BH. ENQ (Part 1 of 2)

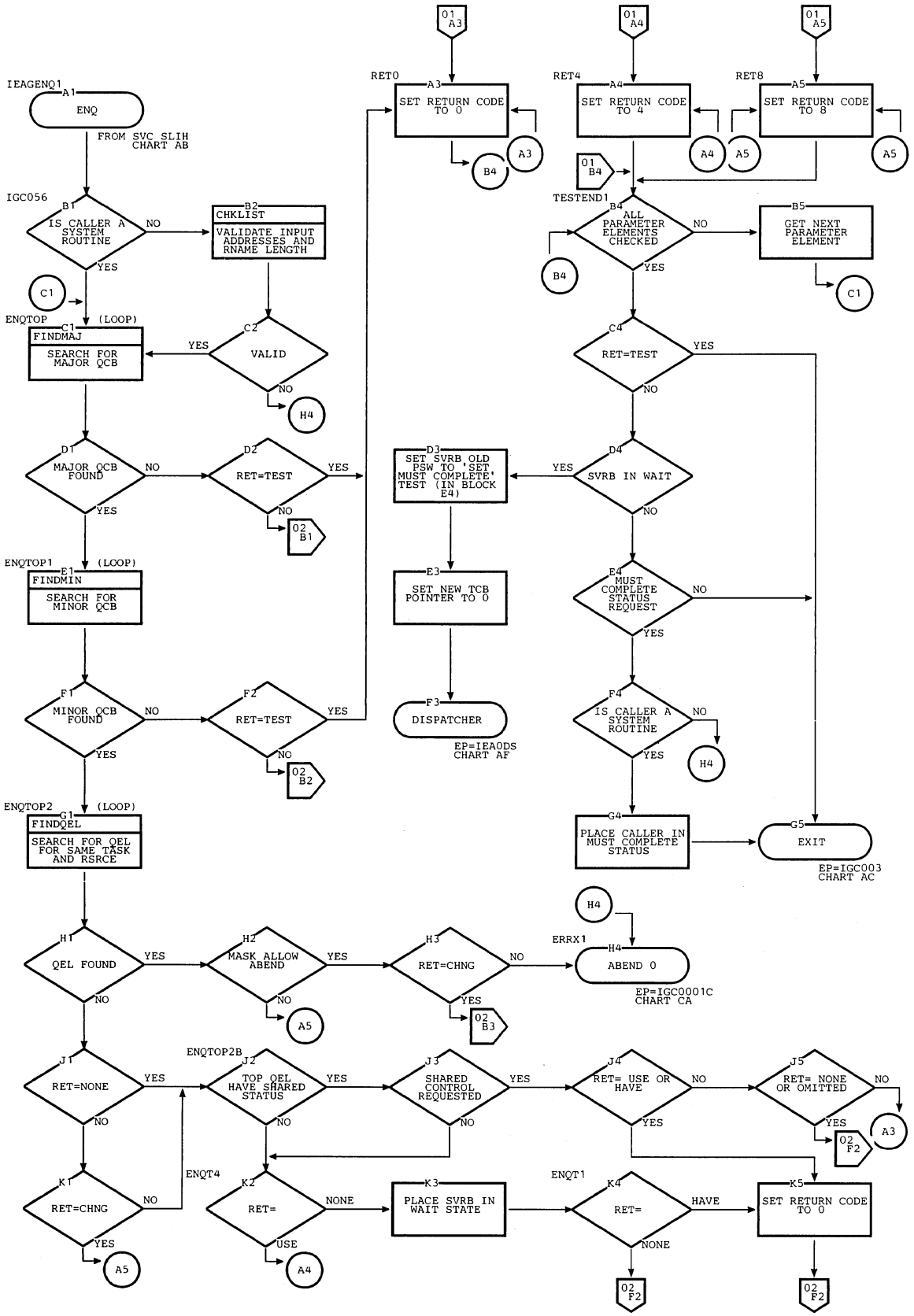


Chart BH. ENQ (Part 2 of 2)

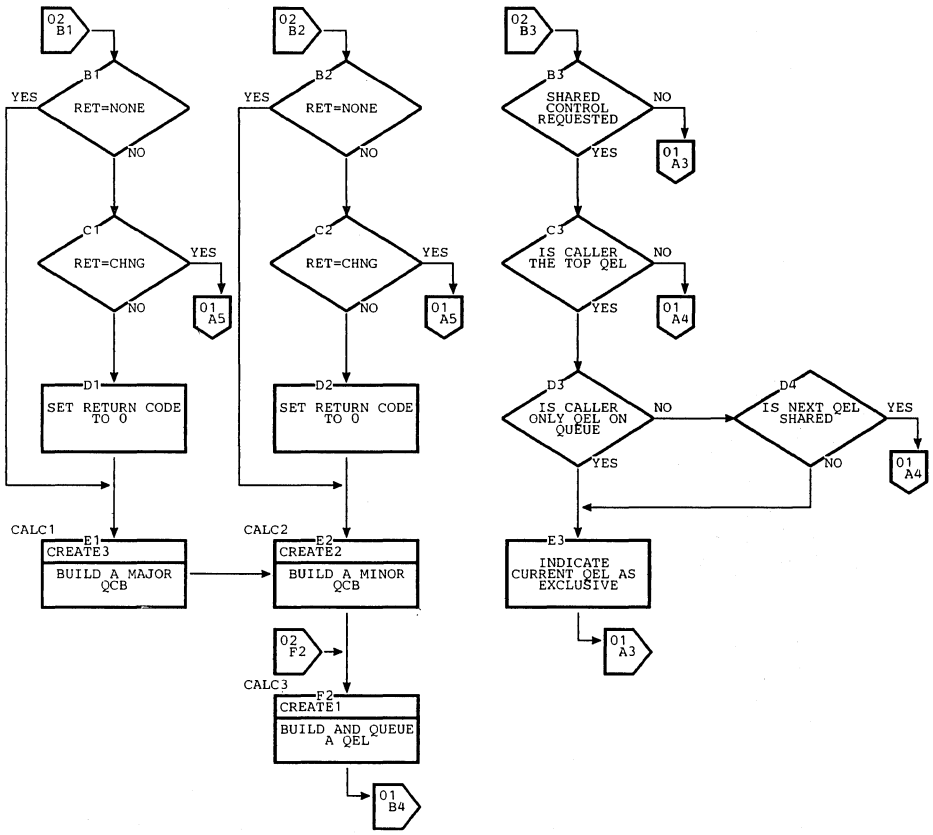


Chart BJ. DEQ (Part 1 of 2)

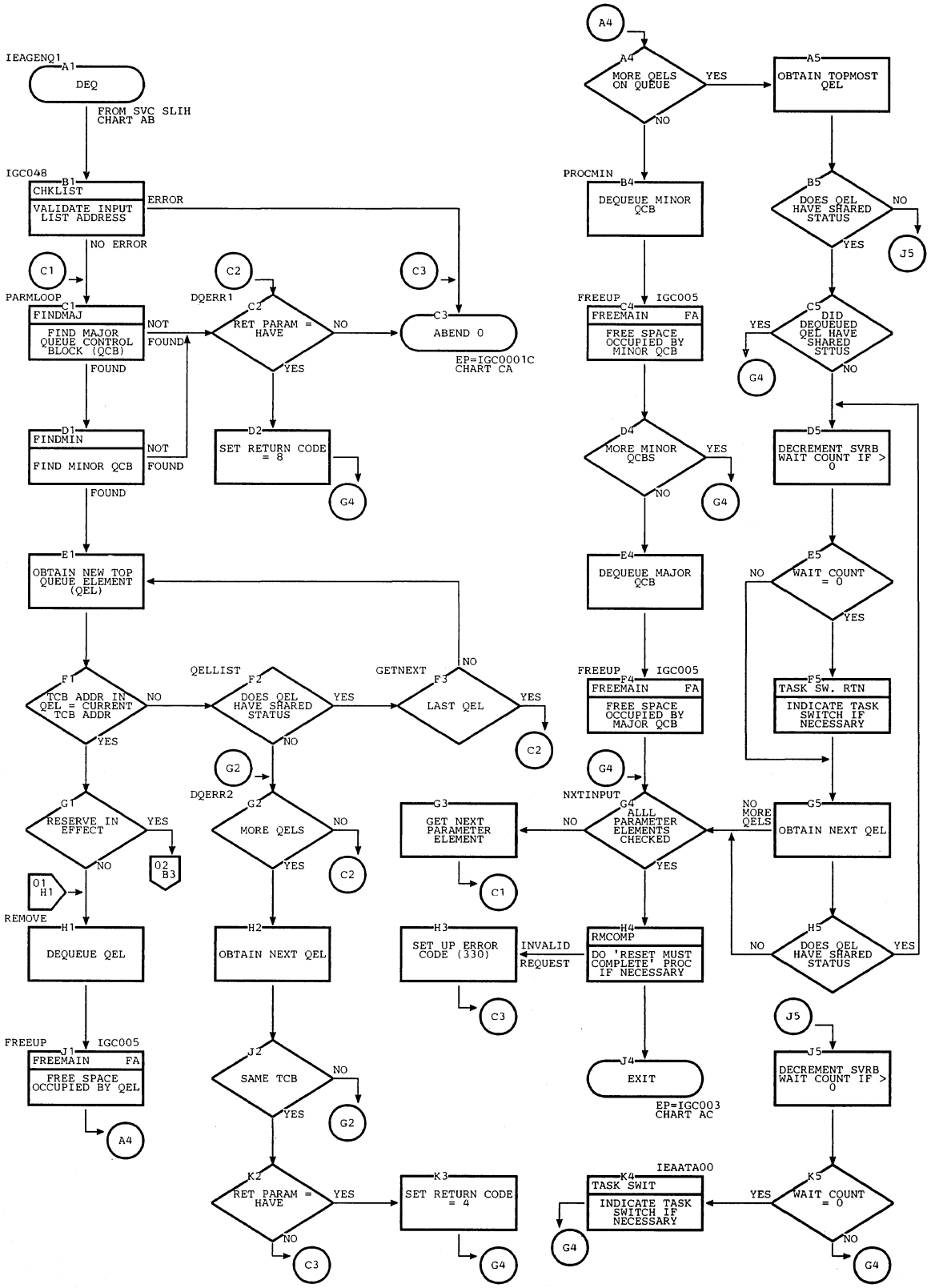


Chart BJ. DEQ (Part 2 of 2)

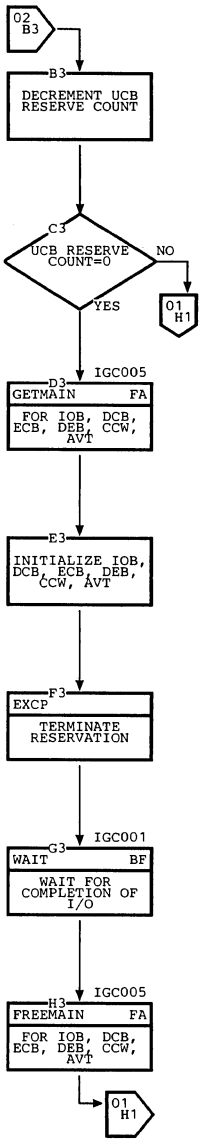


Chart BK. SPIE

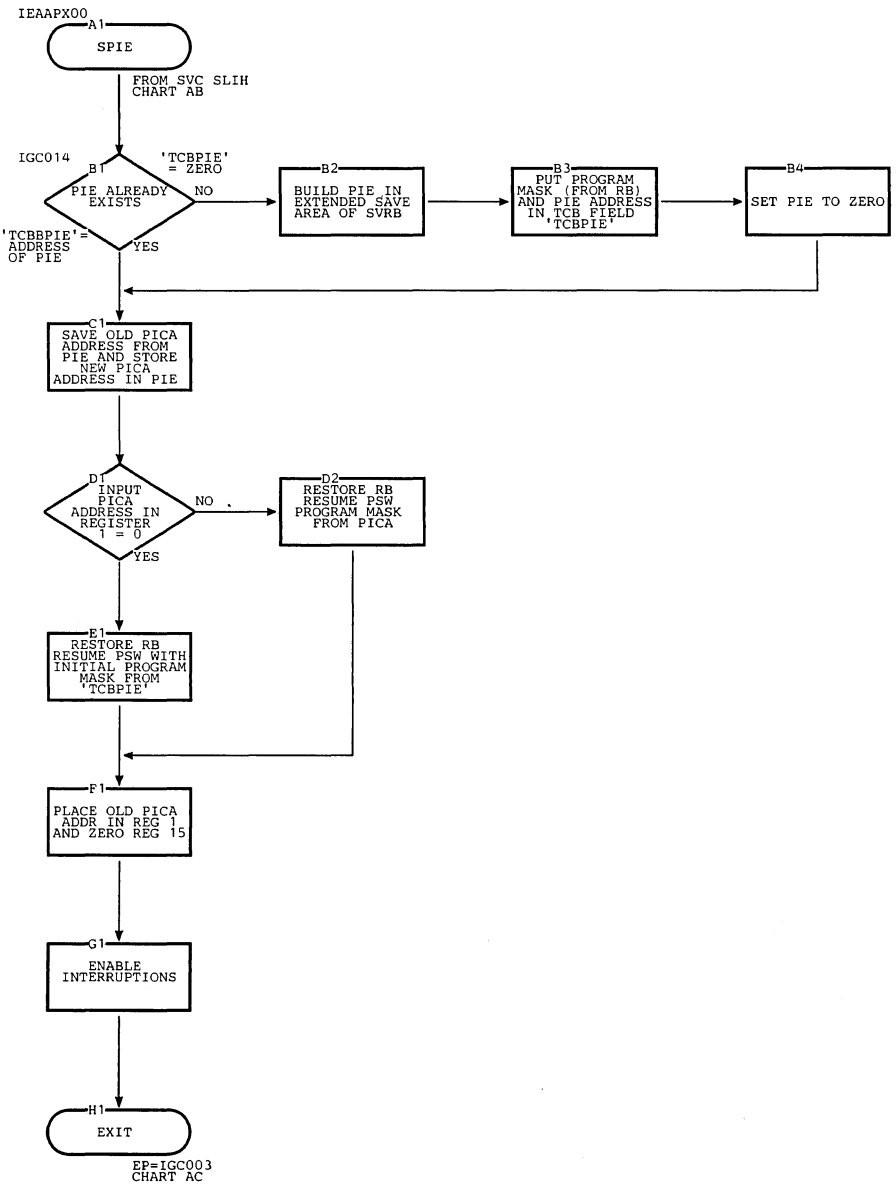


Chart BL. STAE

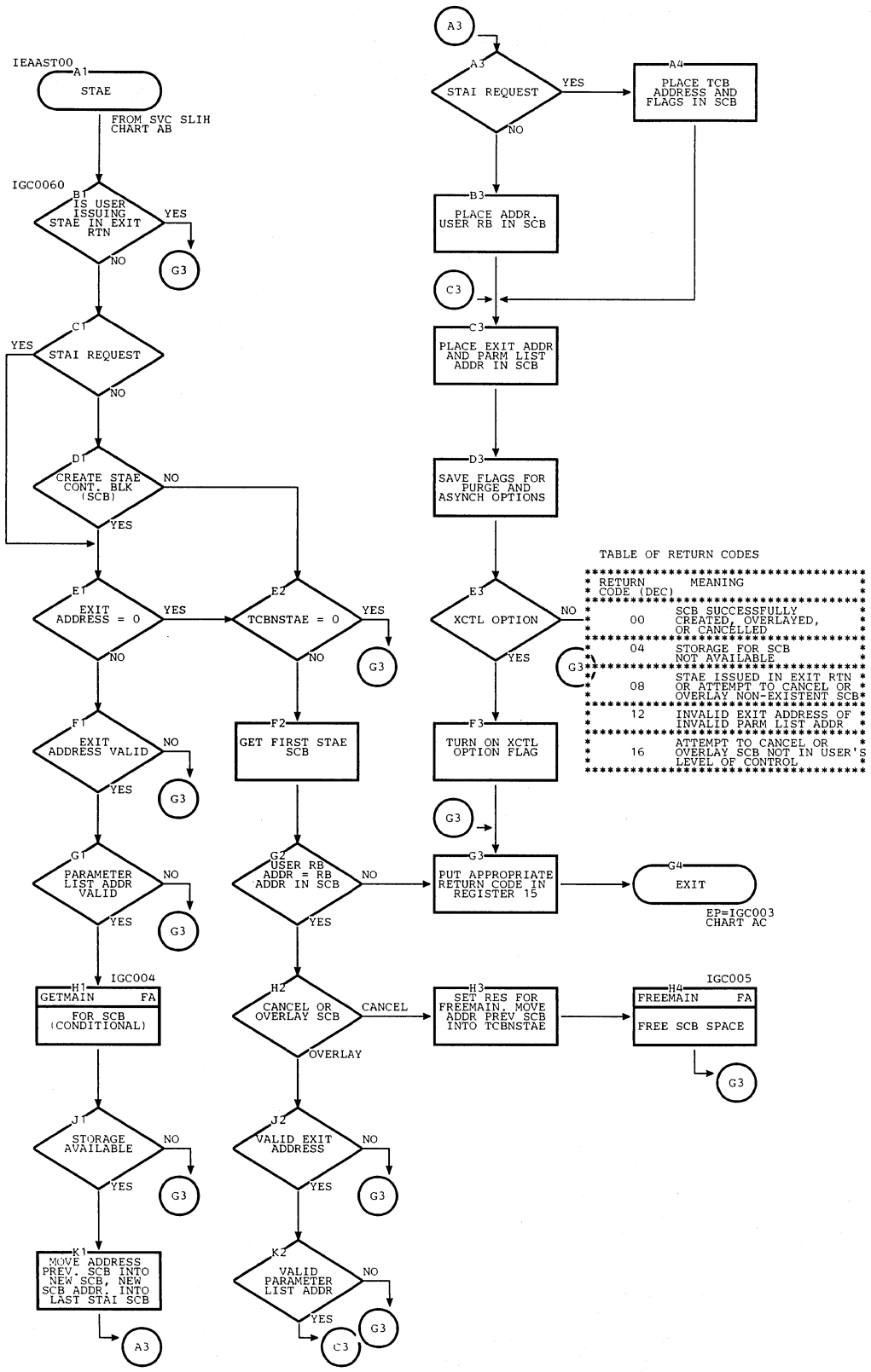


Chart BM. Stage 1 Exit Effector

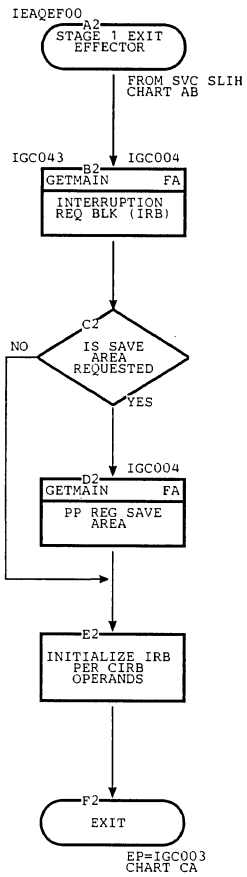


Chart BN. Stage 2 Exit Effector

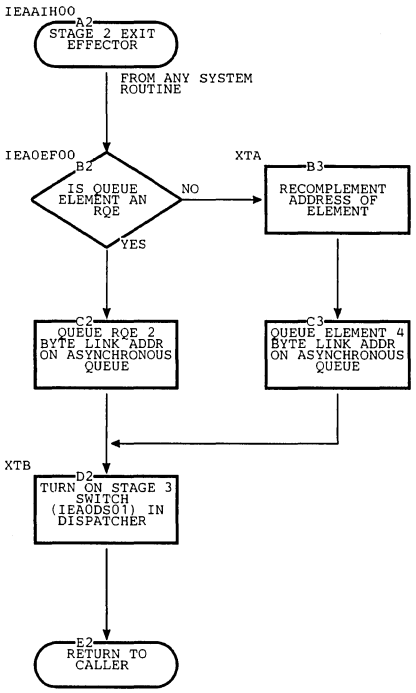


Chart BO. Stage 3 Exit Effector

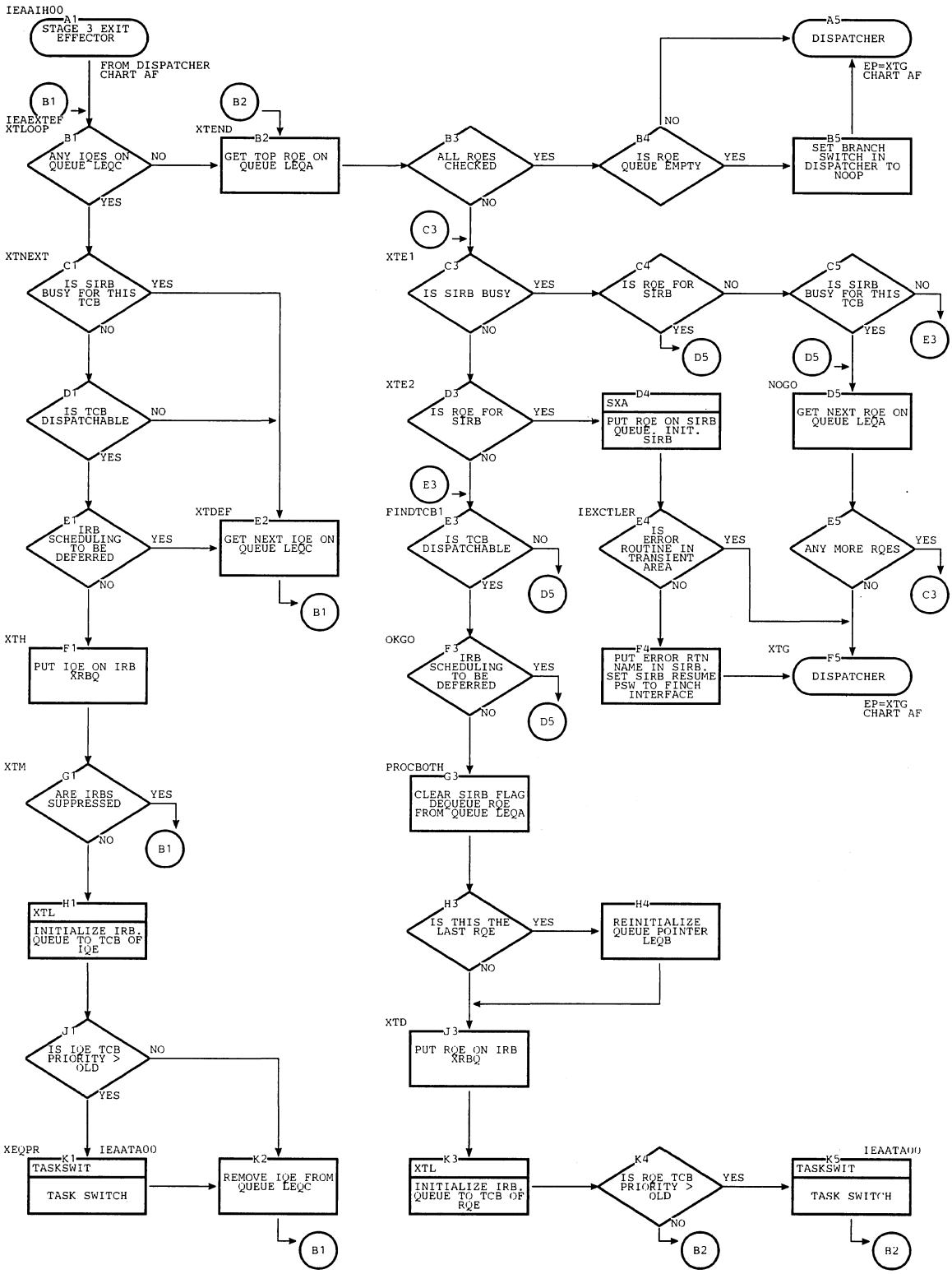


Chart CA. ABEND 0

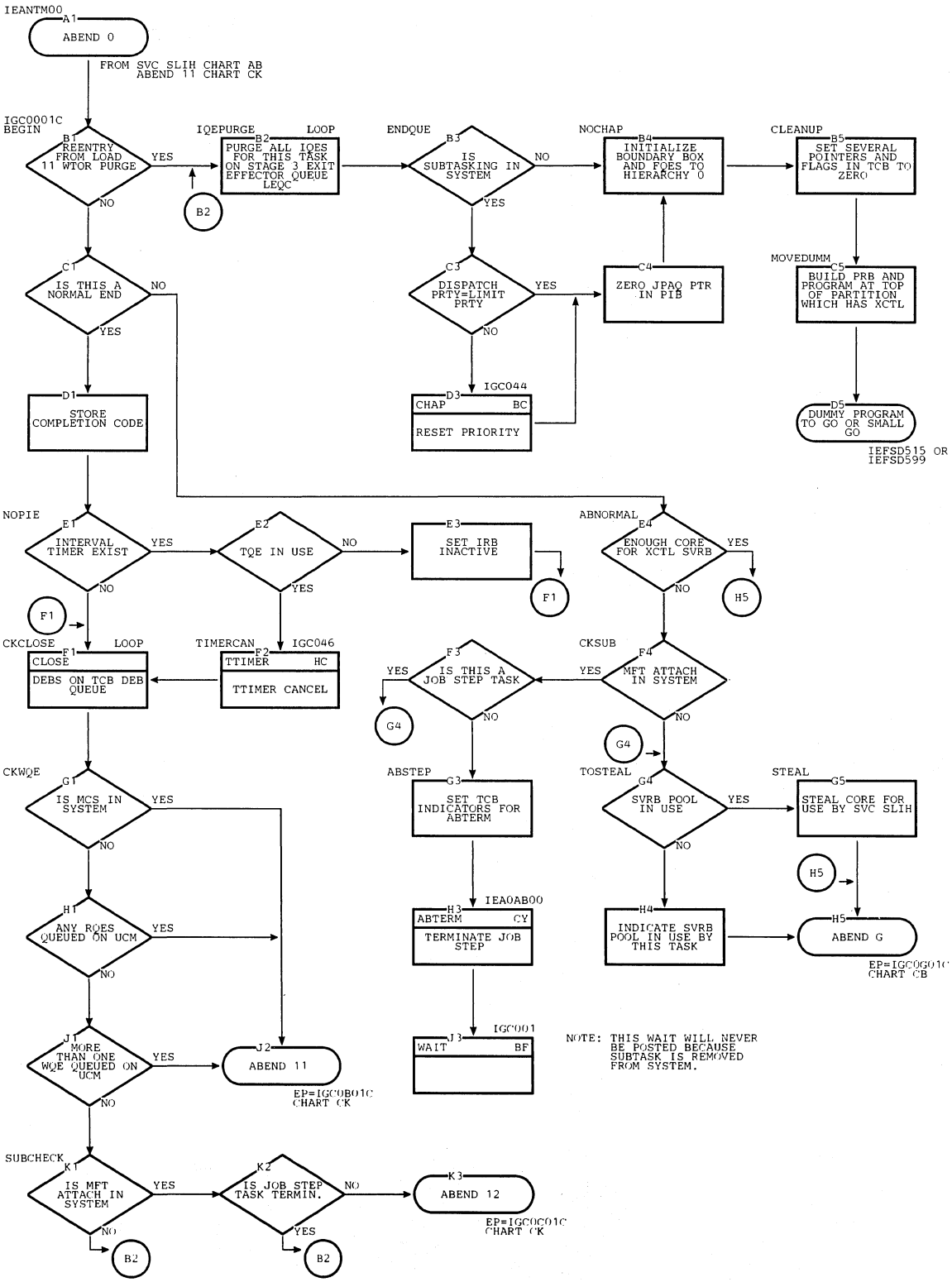


Chart CB. ABEND G (Part 1 of 2)

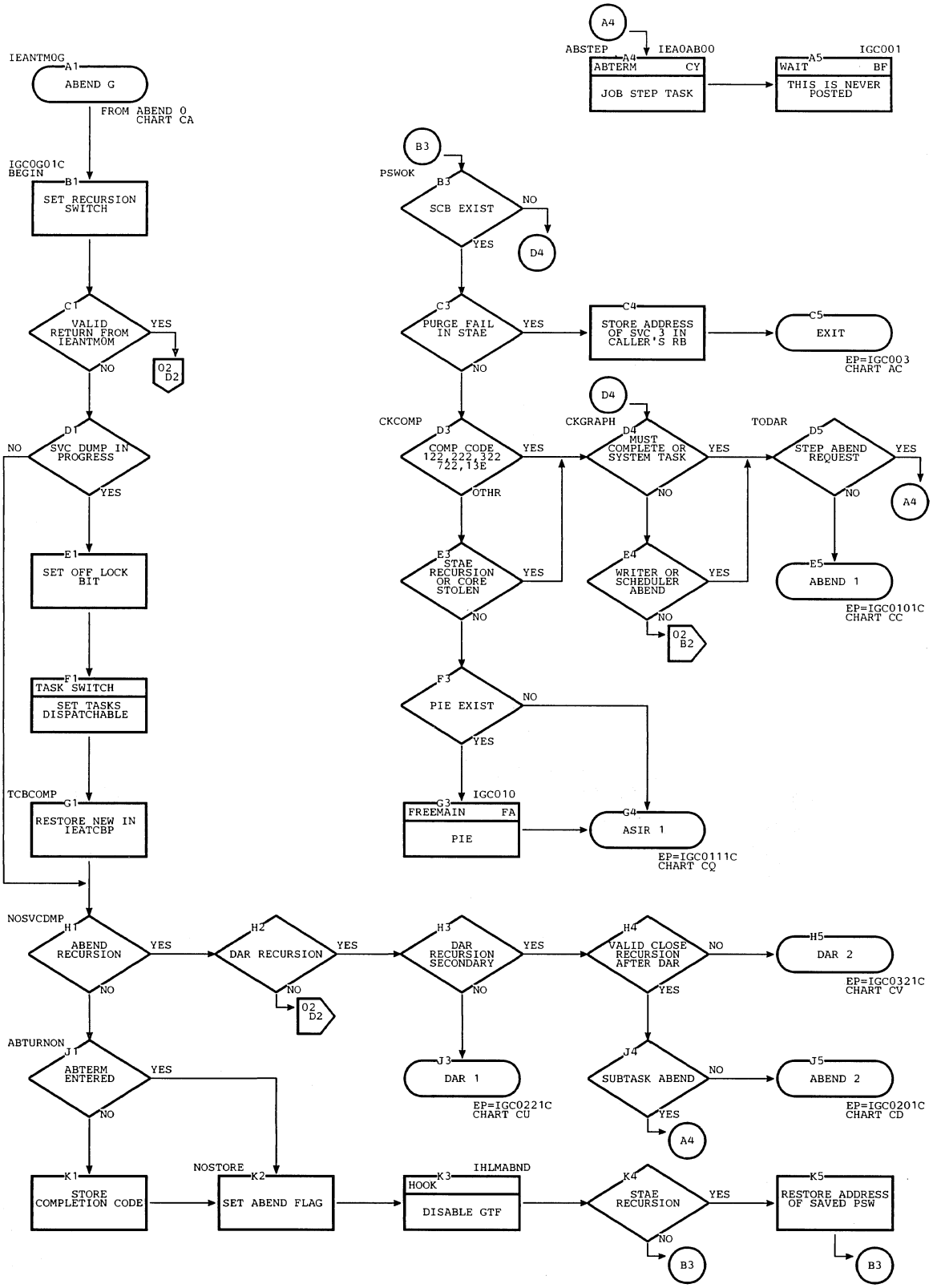


Chart CB. ABEND G (Part 2 of 2)

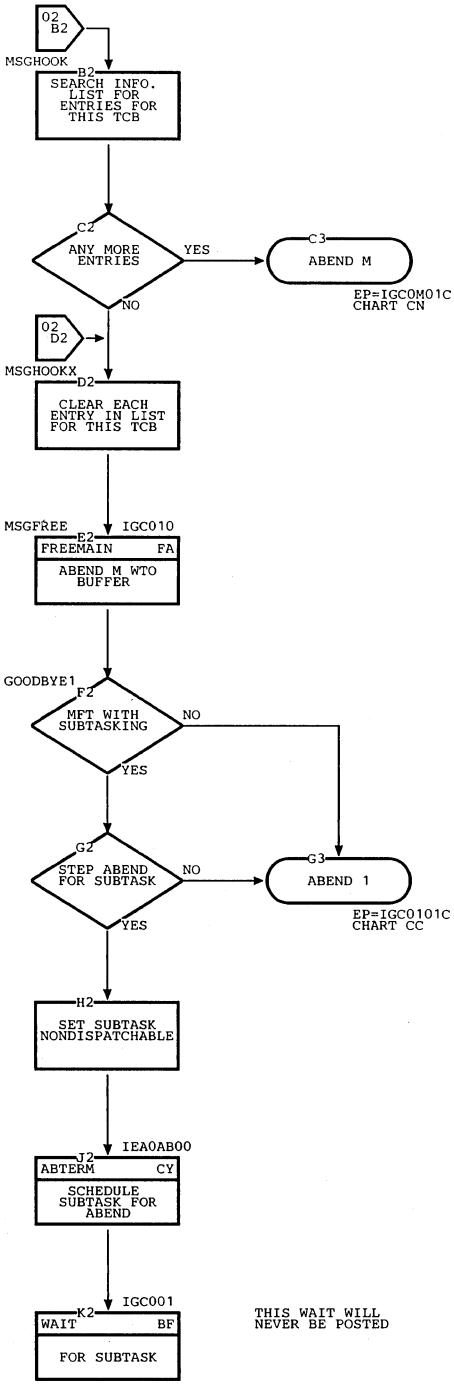


Chart CC. ABEND 1

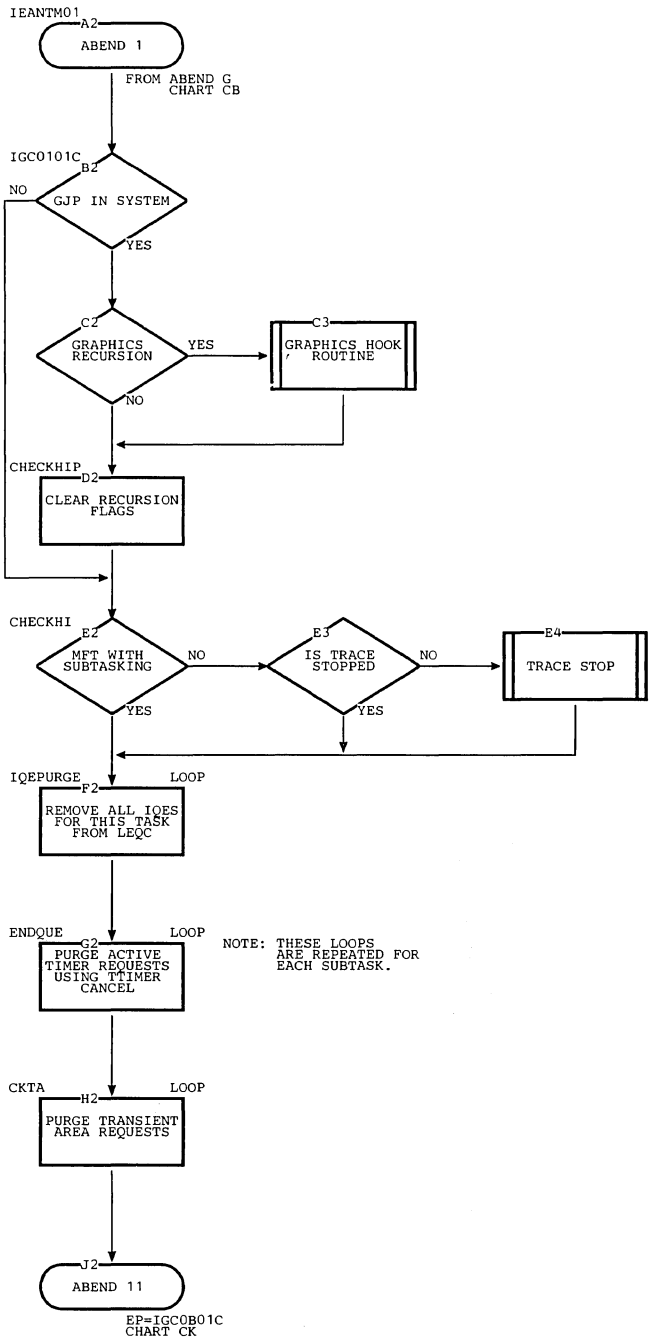


Chart CD. ABEND 2 and ABEND 3

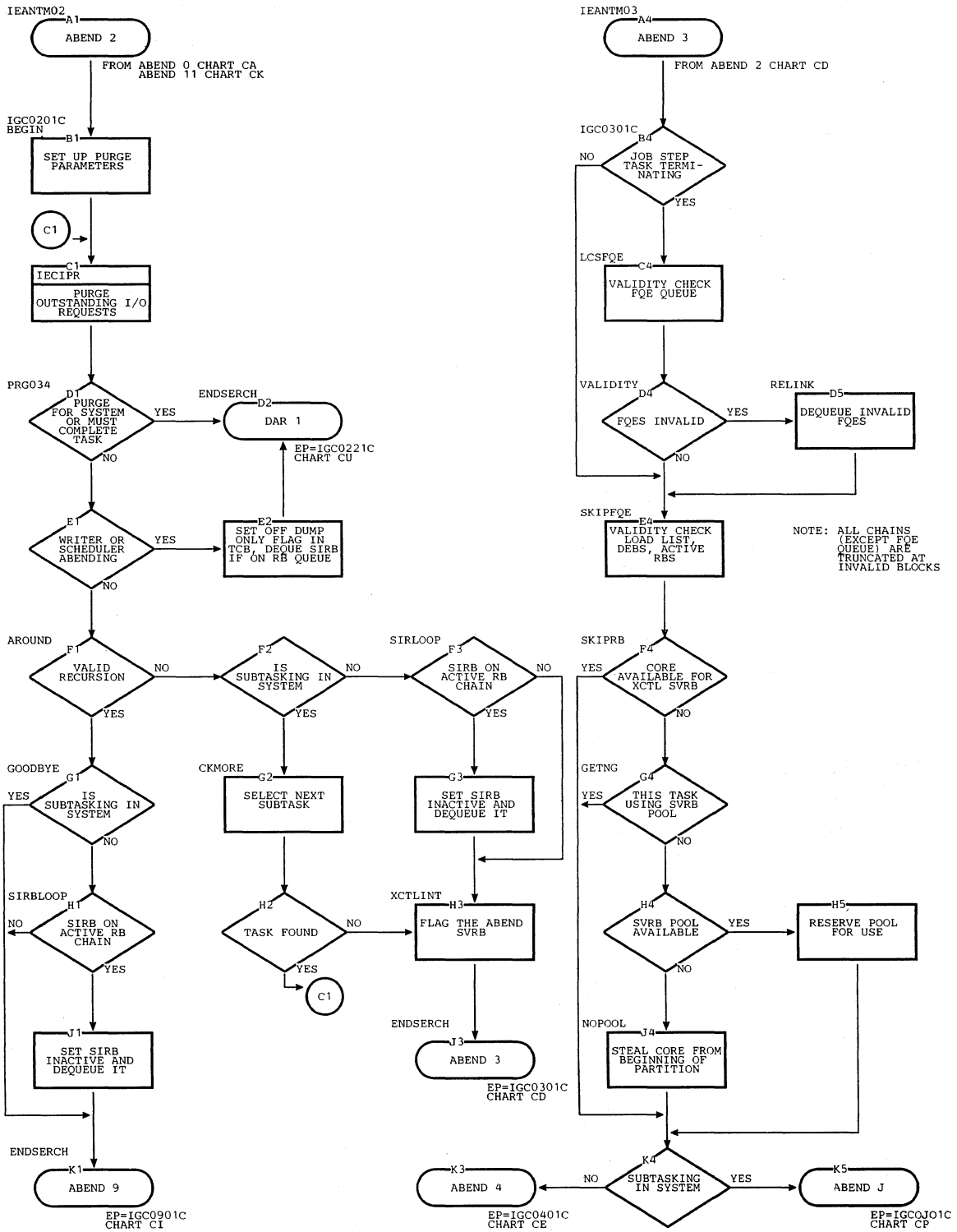


Chart CE. ABEND 4

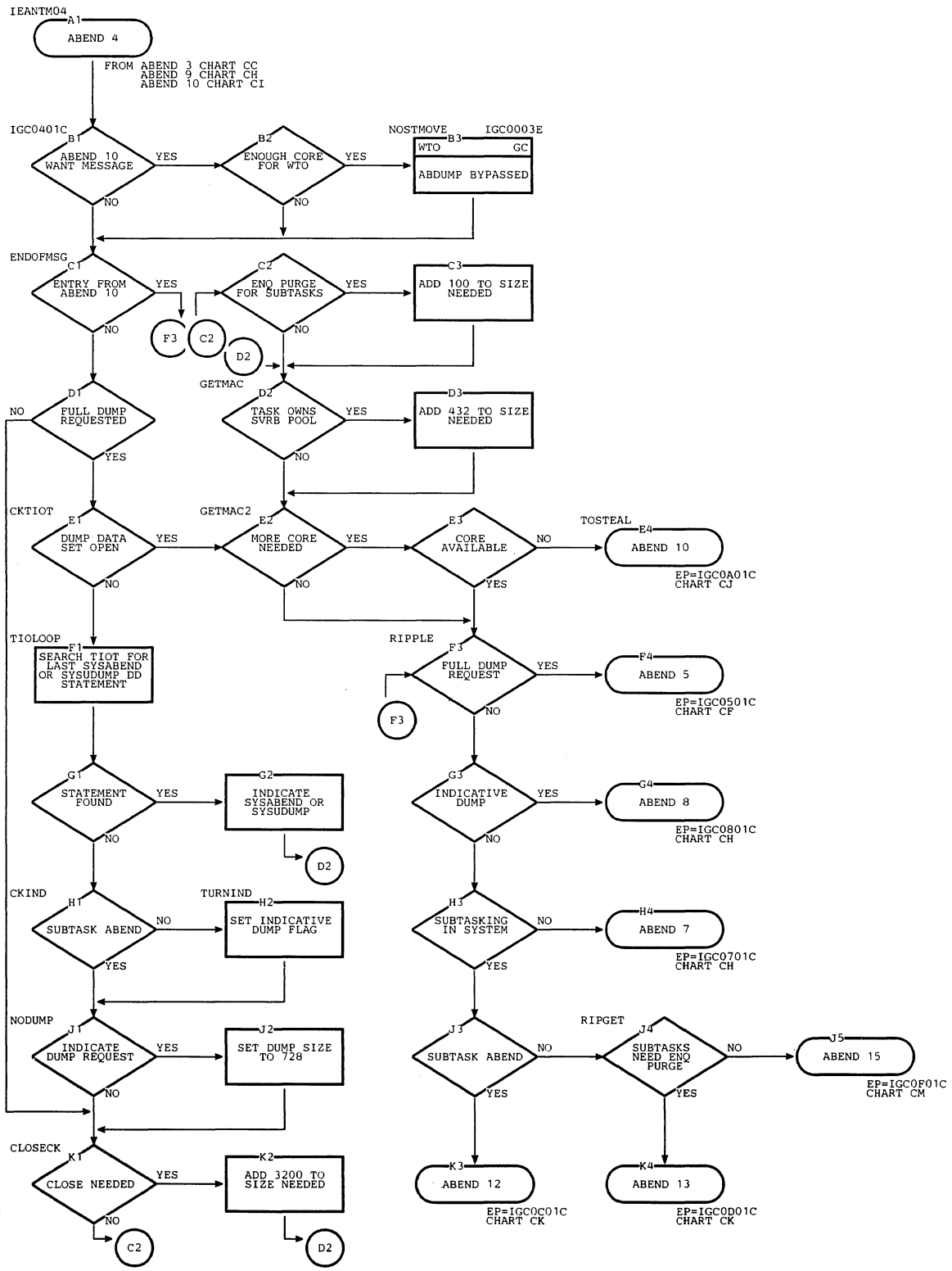


Chart CF. ABEND 5

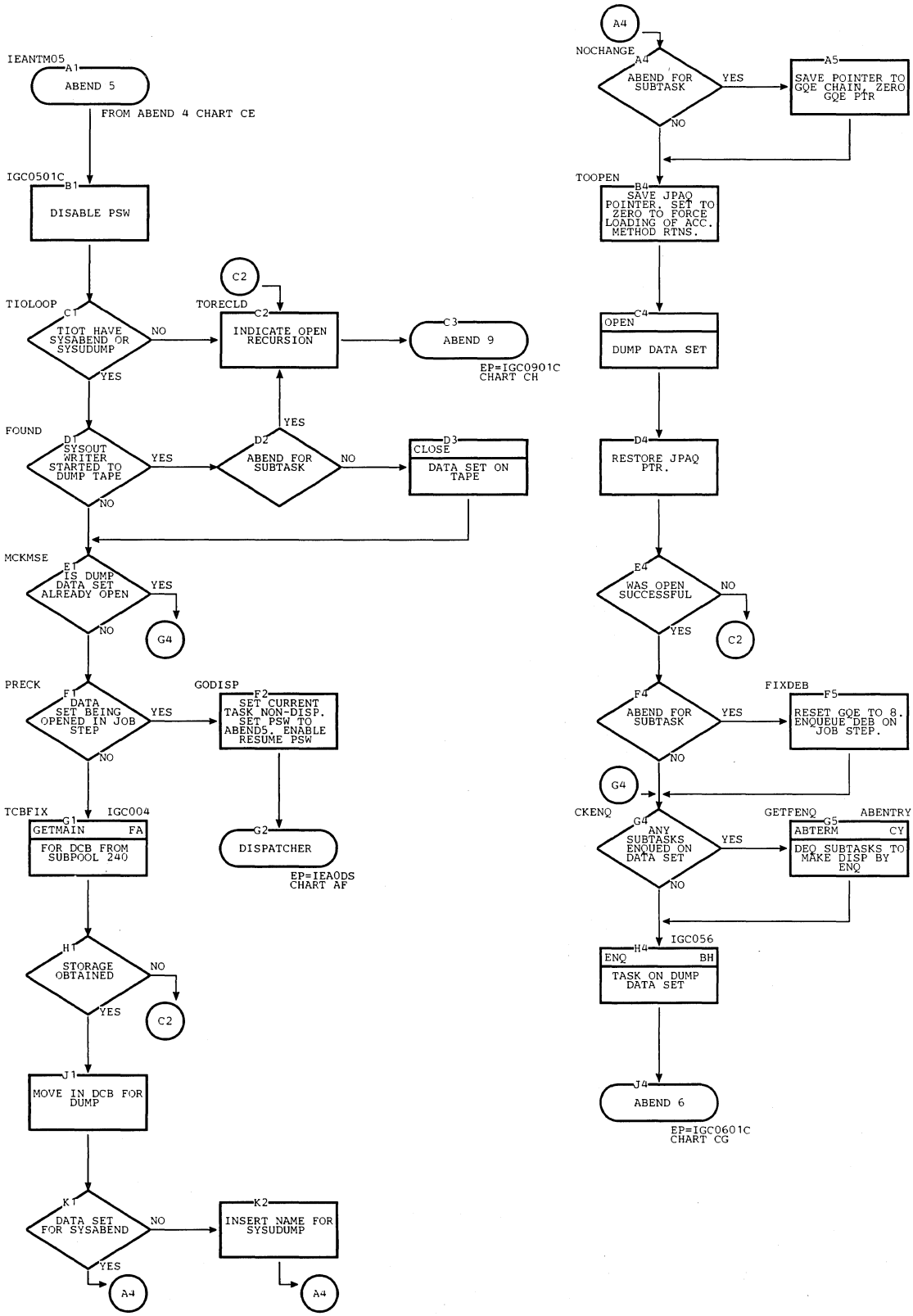


Chart CG. ABEND 6

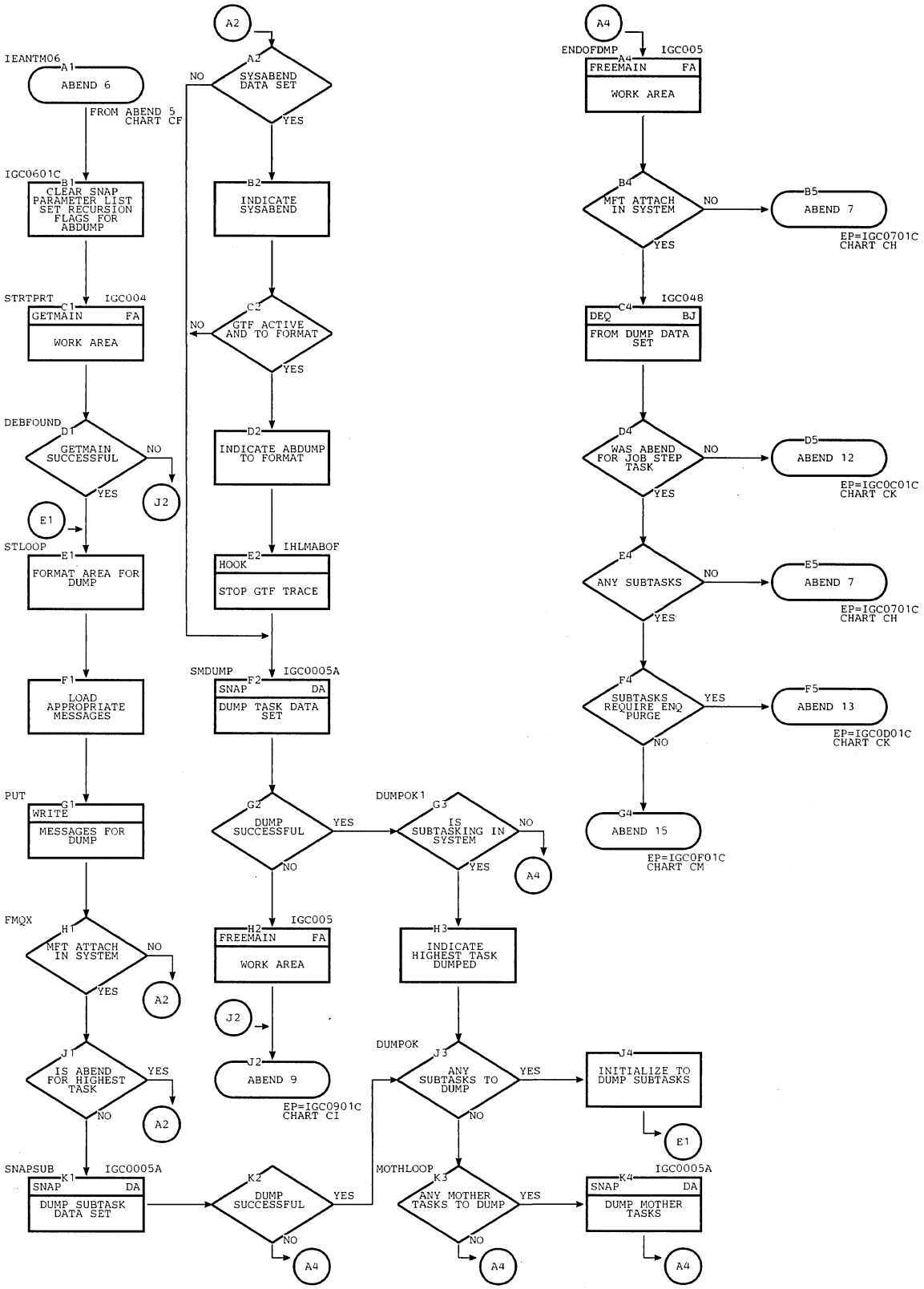


Chart CH. ABEND 7 and ABEND 8

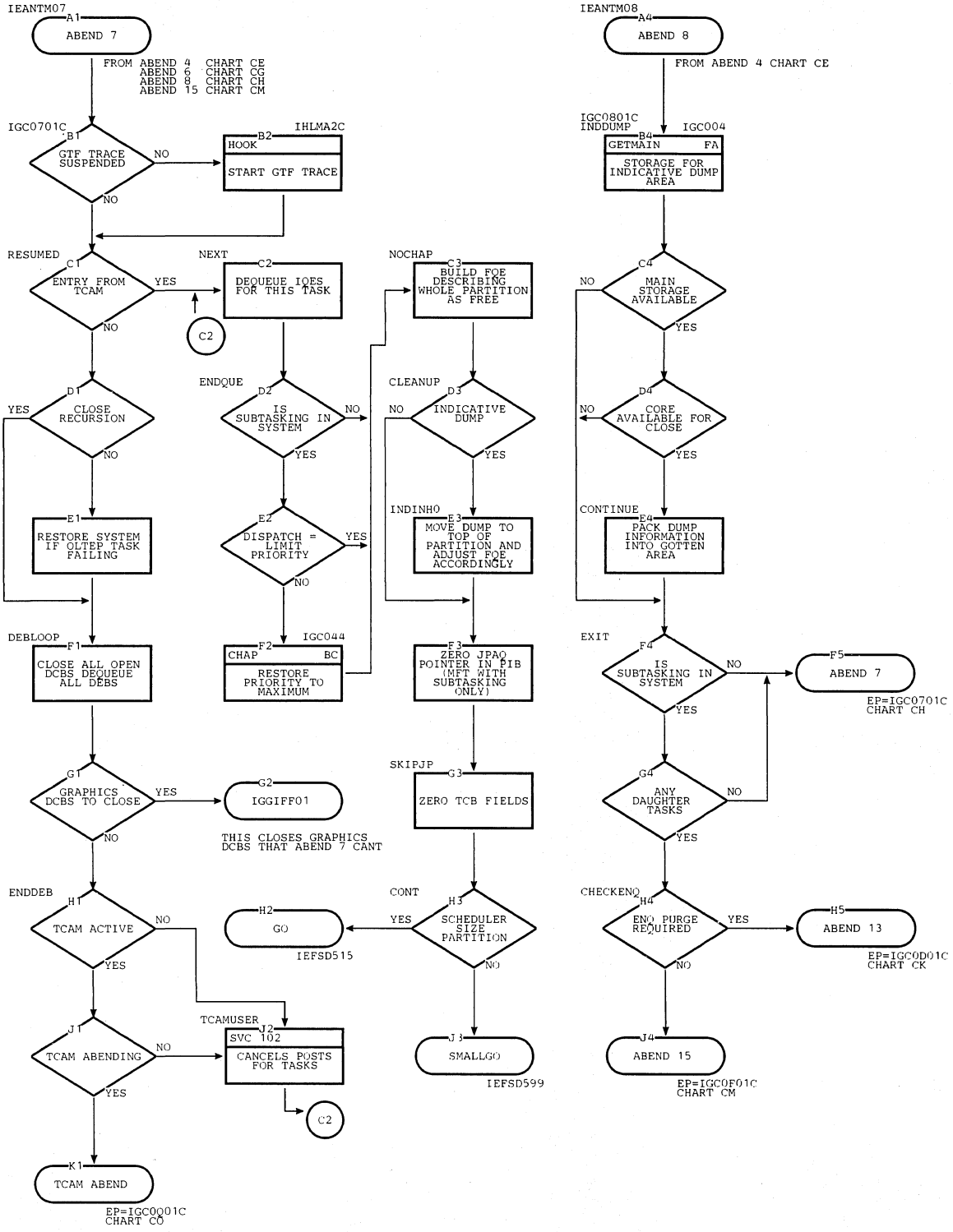


Chart CI. ABEND 9

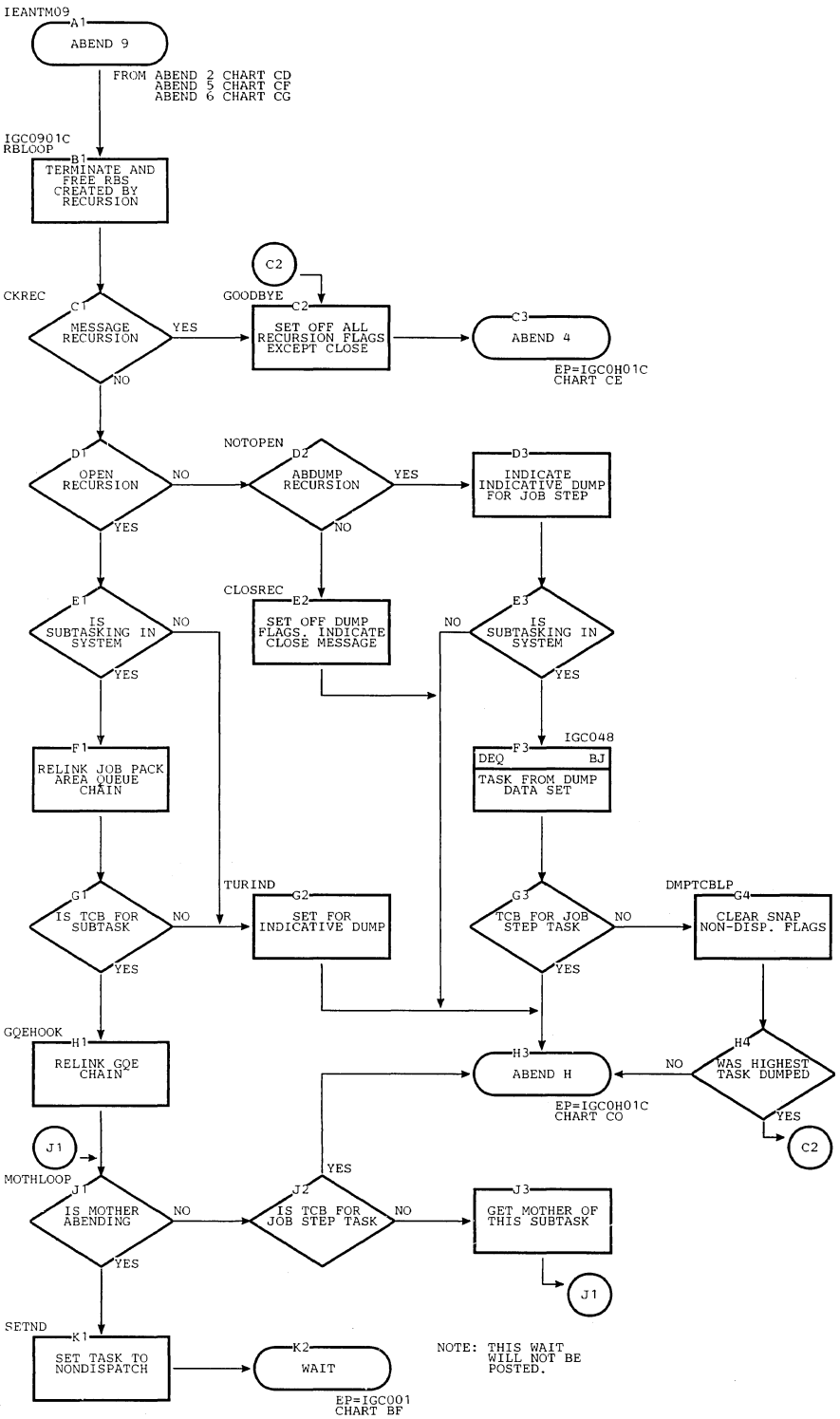


Chart CJ. ABEND 10

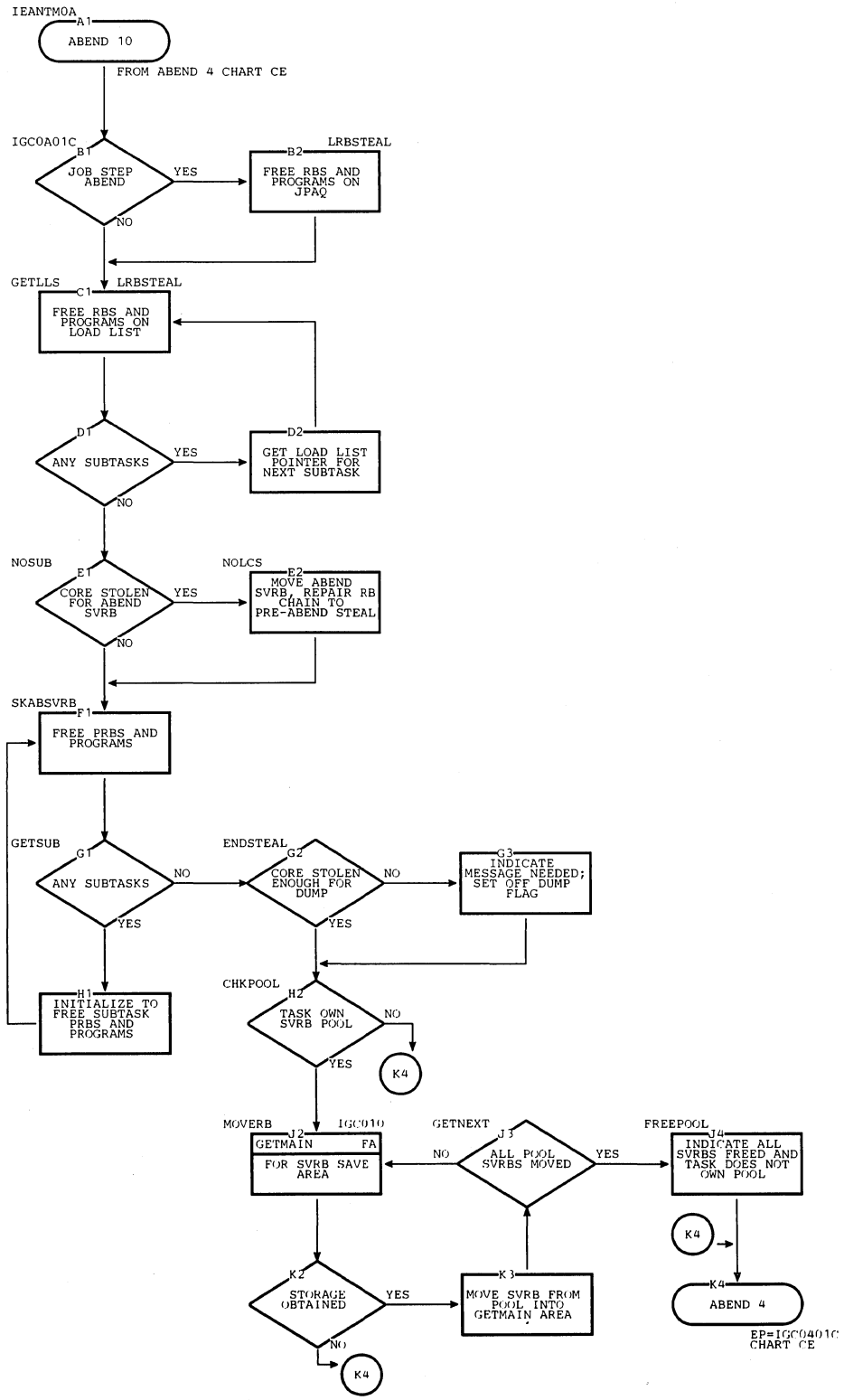


Chart CK. ABEND 11, ABEND 12, and ABEND 13

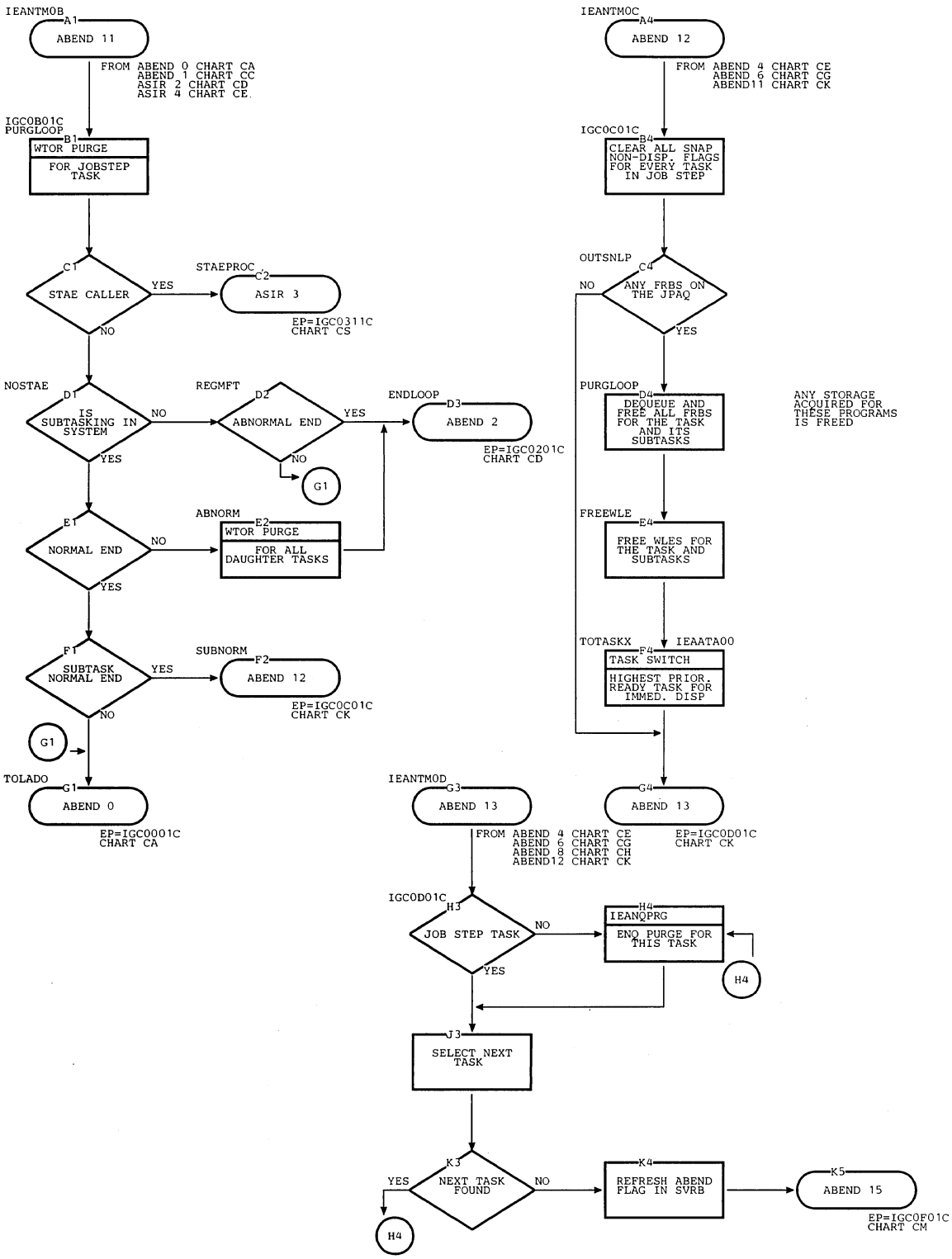


Chart CL. ABEND 14

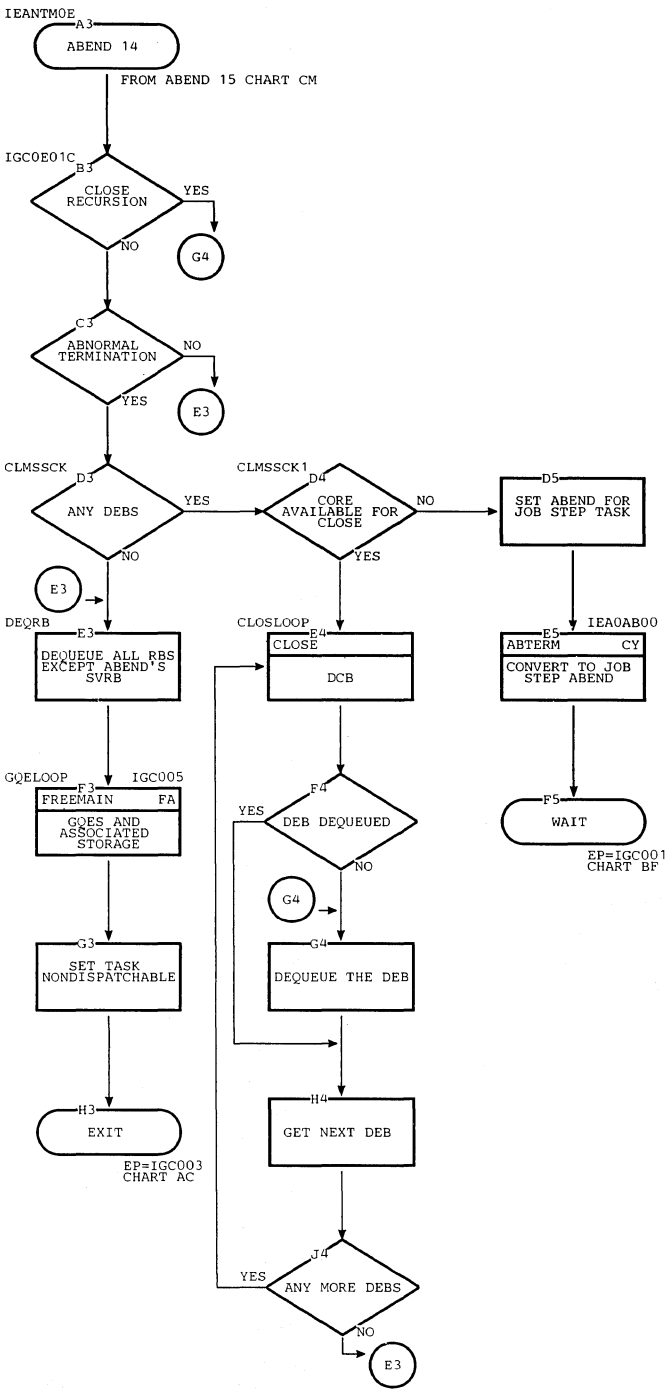


Chart CM. ABEND 15

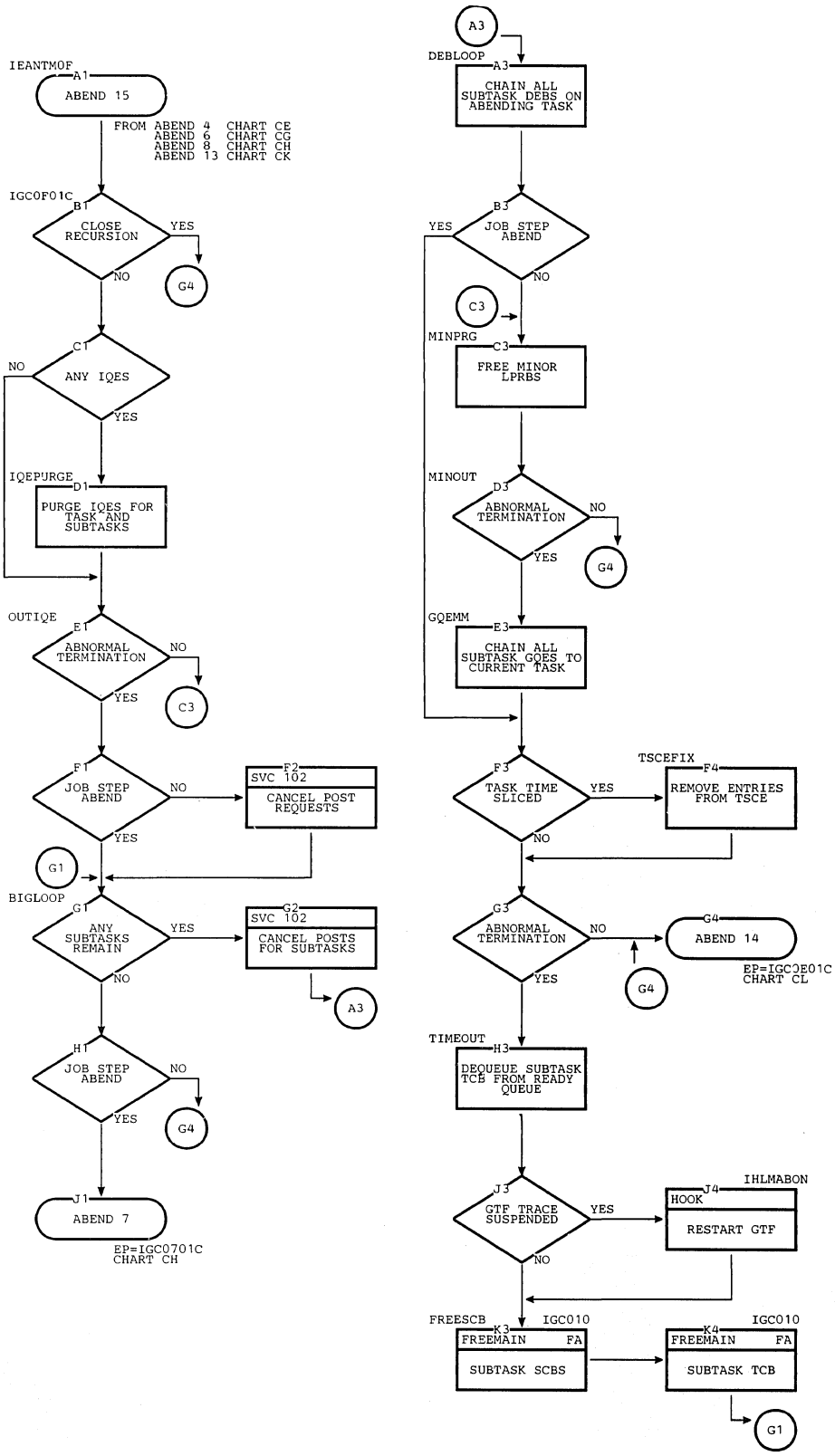


Chart CN. ABEND M

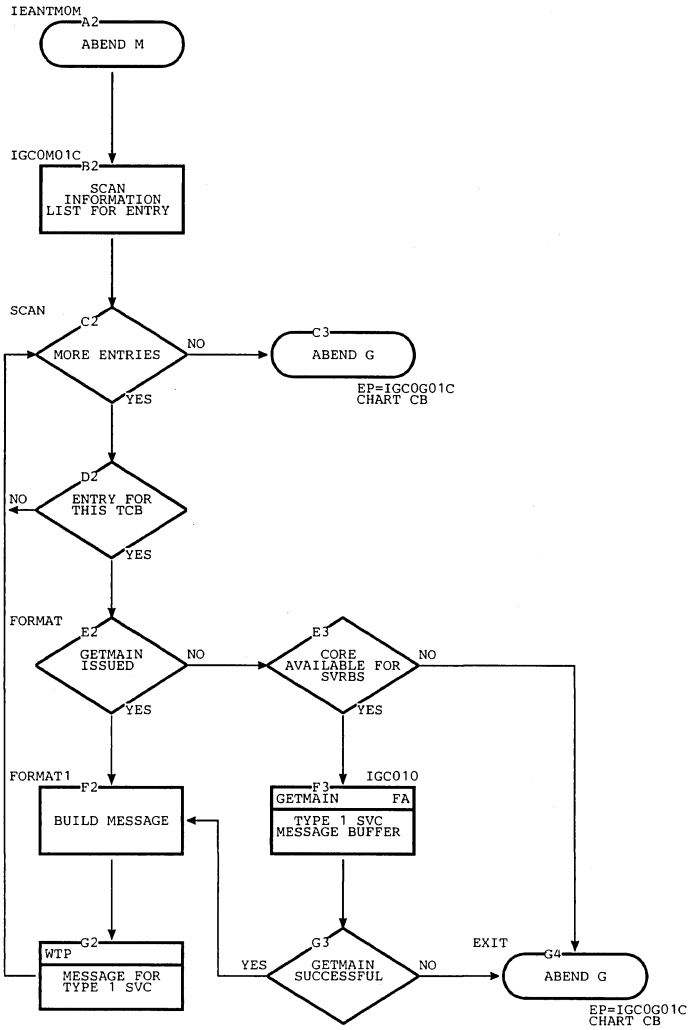


Chart CO. ABEND H

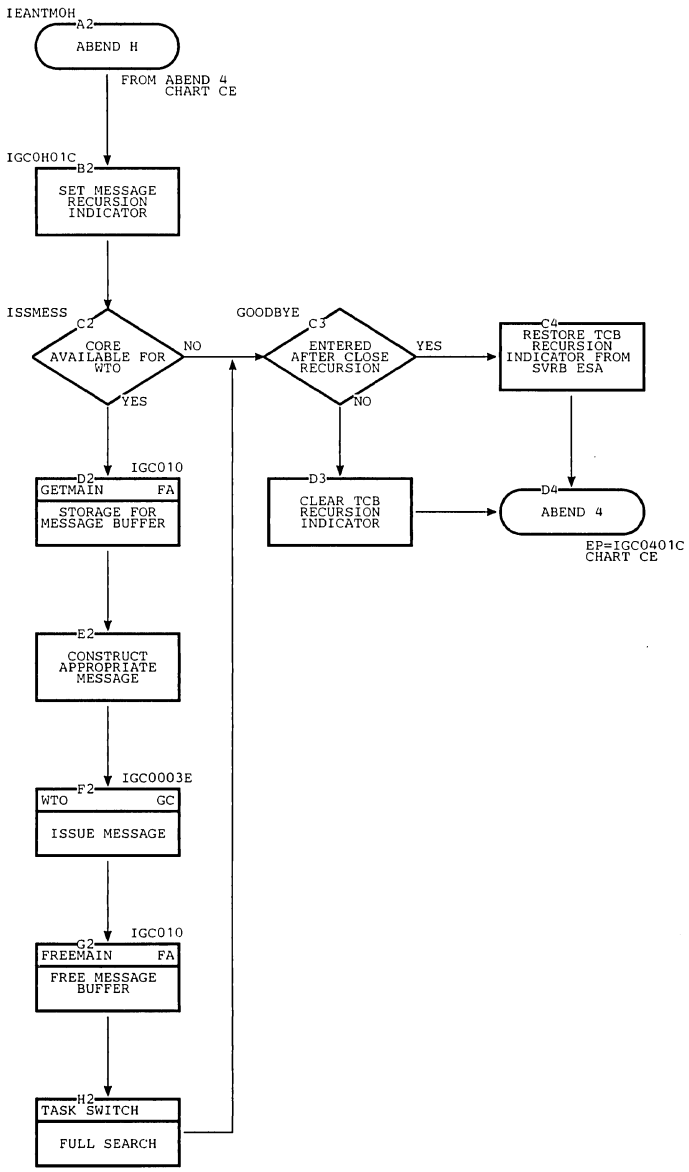


Chart CP. ABEND J

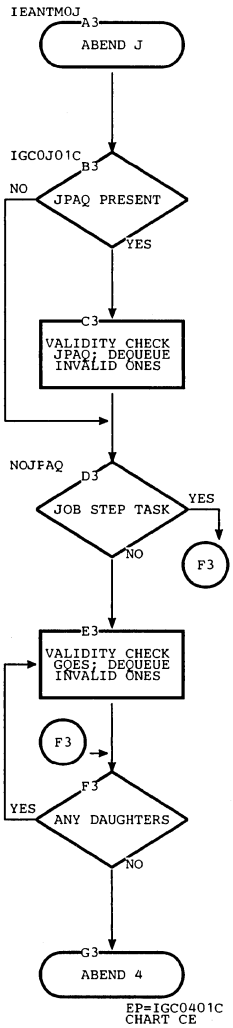


Chart CQ. ABEND/STAE Interface Routine 1

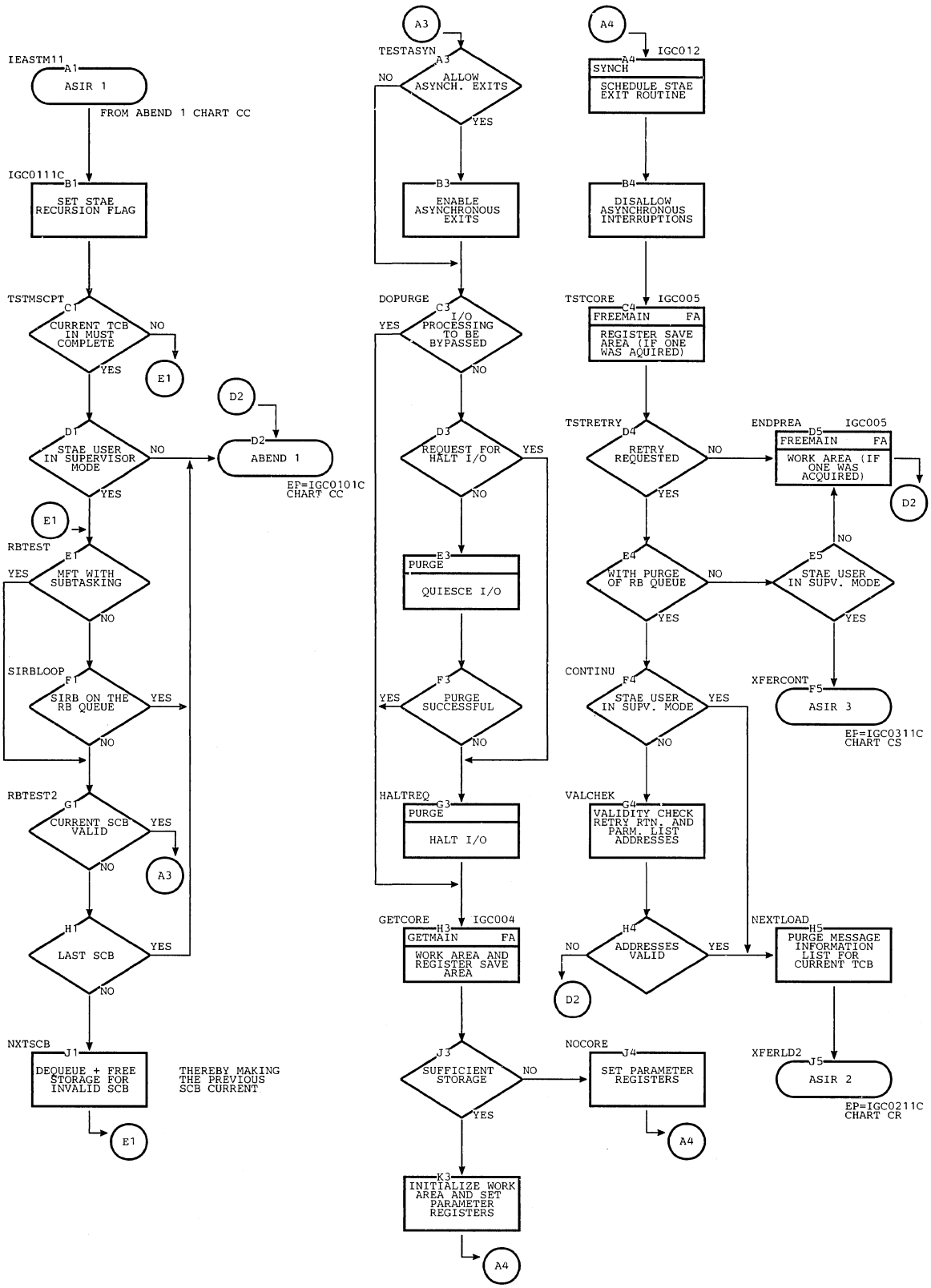


Chart CR. ABEND/STAE Interface Routine 2

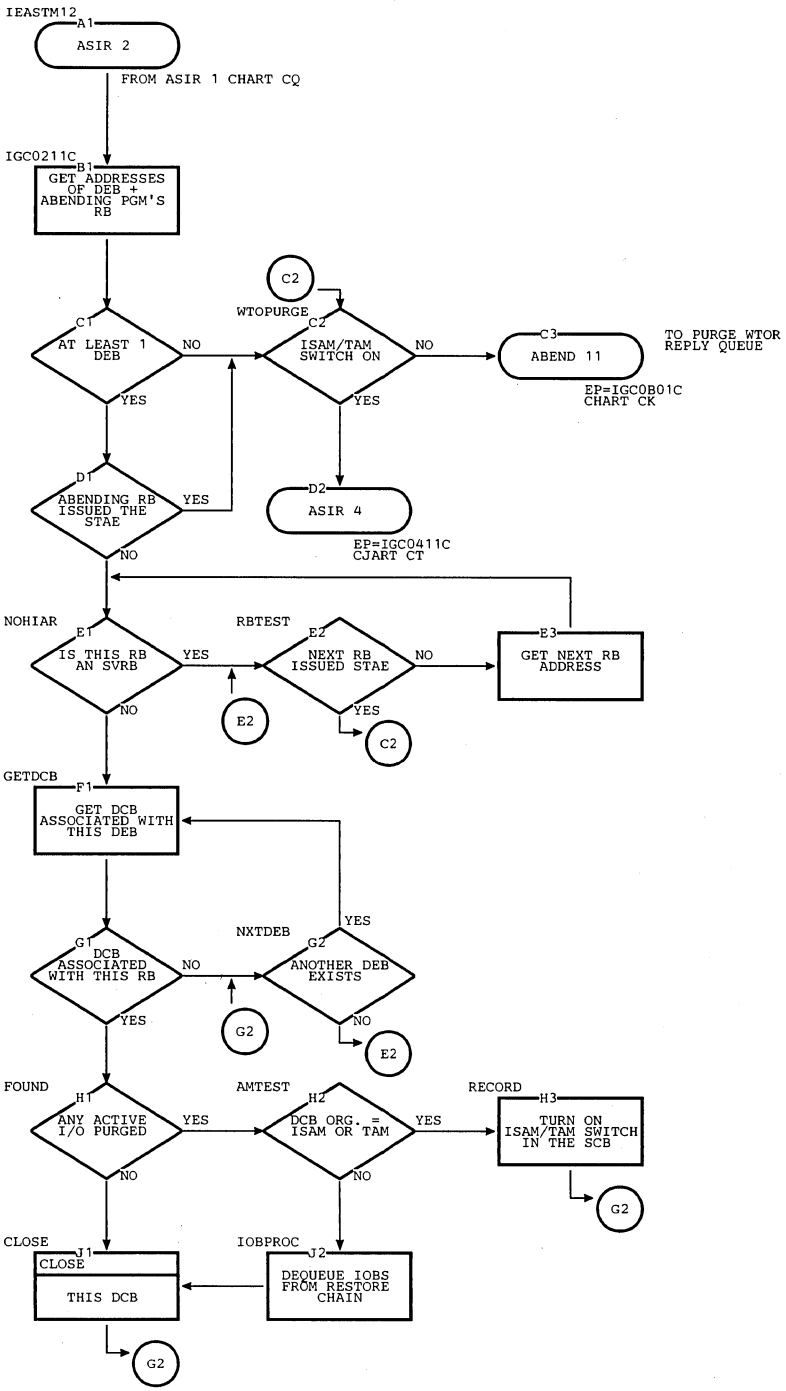
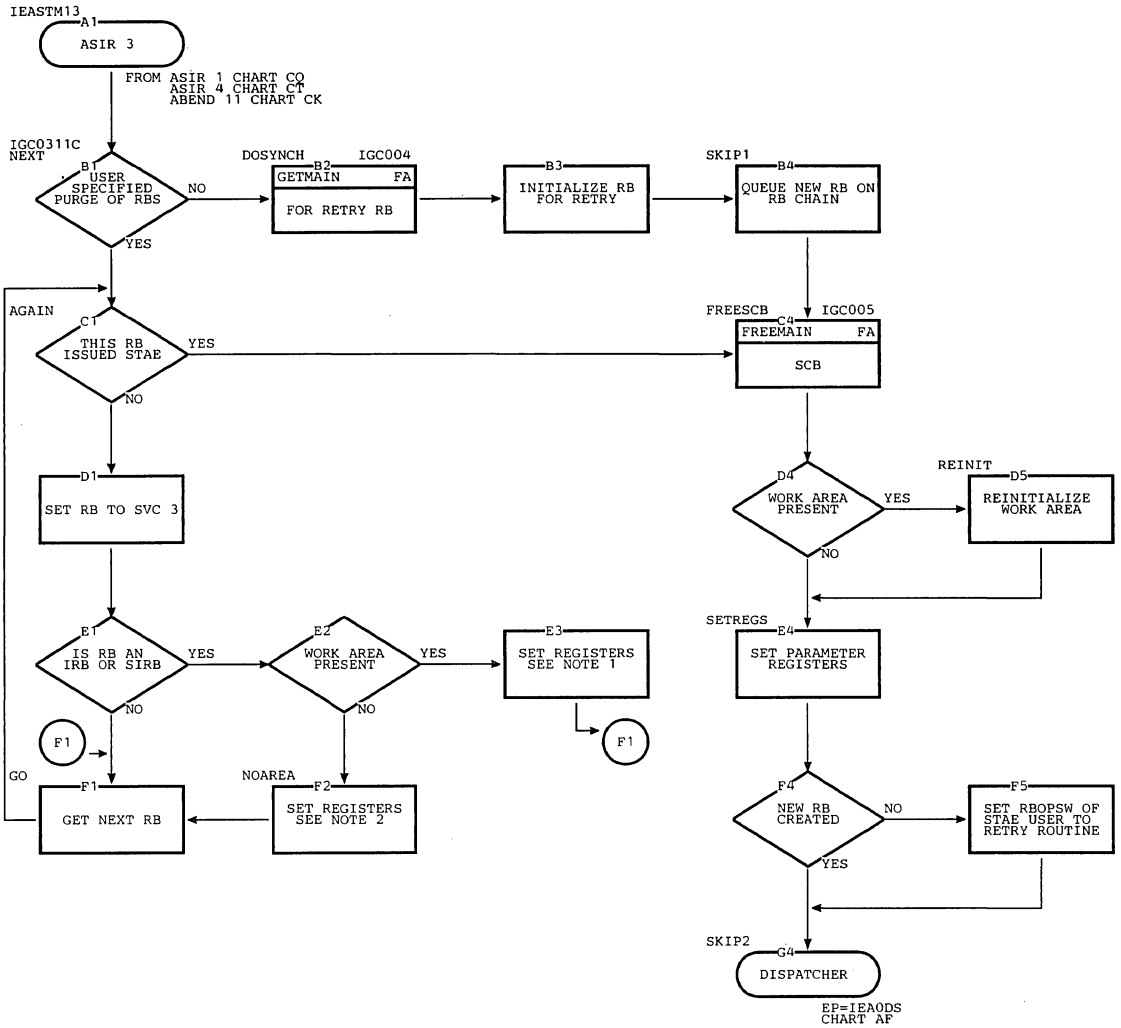


Chart CS. ABEND/STAE Interface Routine 3



NOTE 1: ZERO IN REGISTER 0
WORK AREA ADDRESS IN REGISTER 1
RETRY ADDRESS IN REGISTER 15

NOTE 2: CODE IN REGISTER 0
COMPLETION CODE IN REGISTER 1
I/O BLOCK ADDRESS IN REGISTER 2
RETRY ADDRESS IN REGISTER 15

Chart CT. ABEND/STAE Interface Routine 4

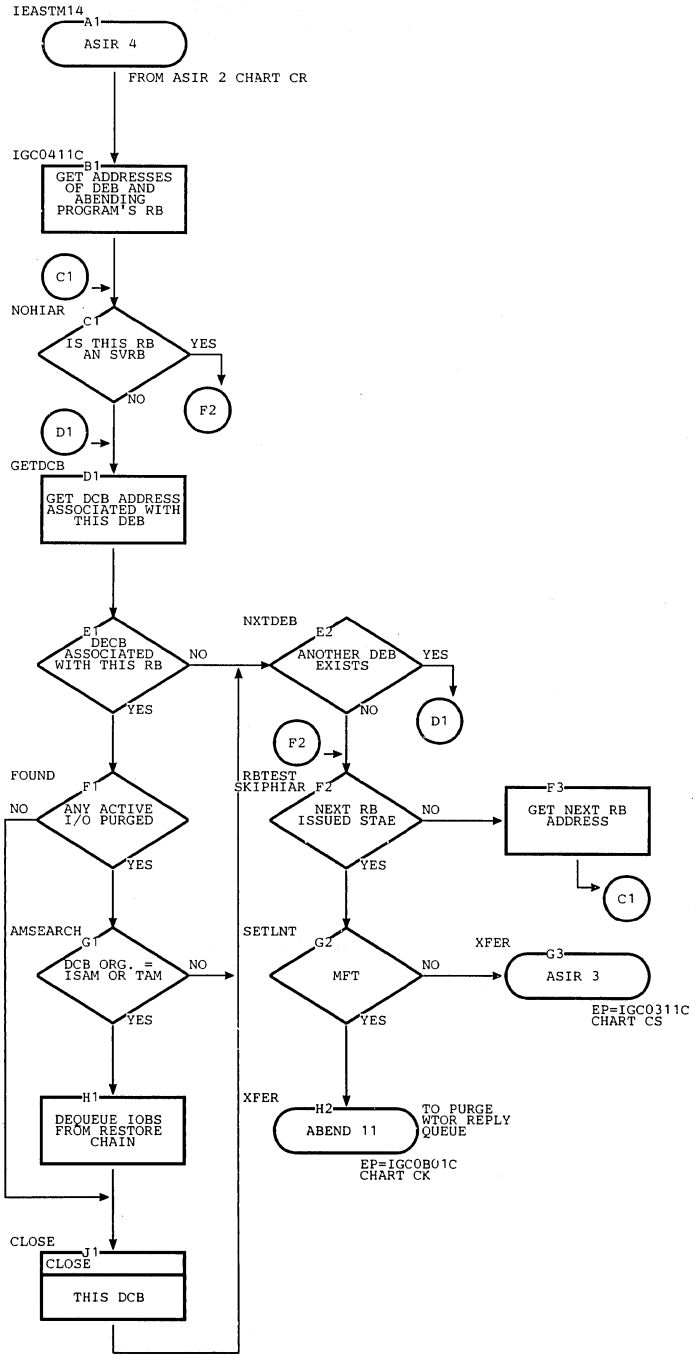


Chart CU. Damage Assessment Routine 1

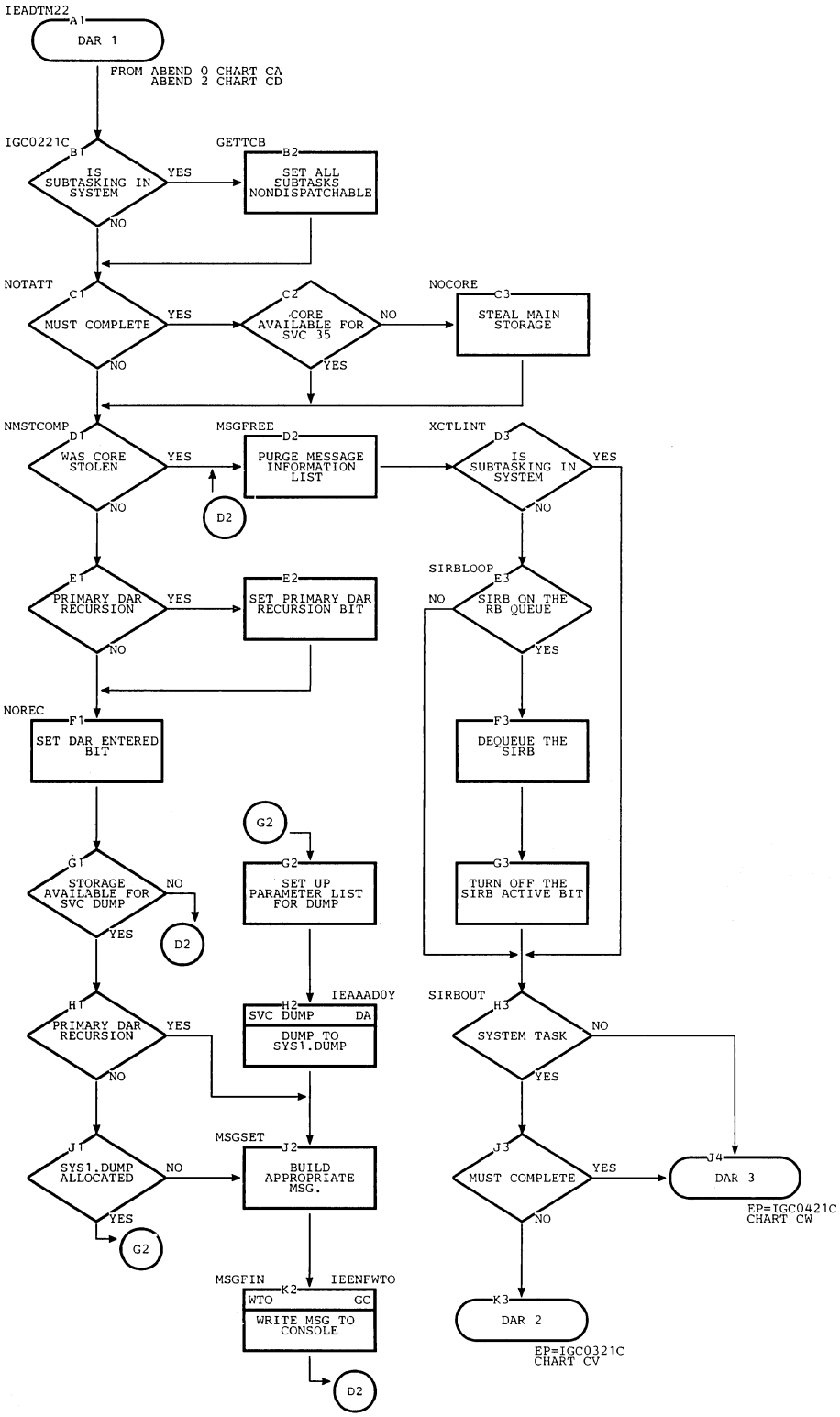


Chart CV. Damage Assessment Routine 2

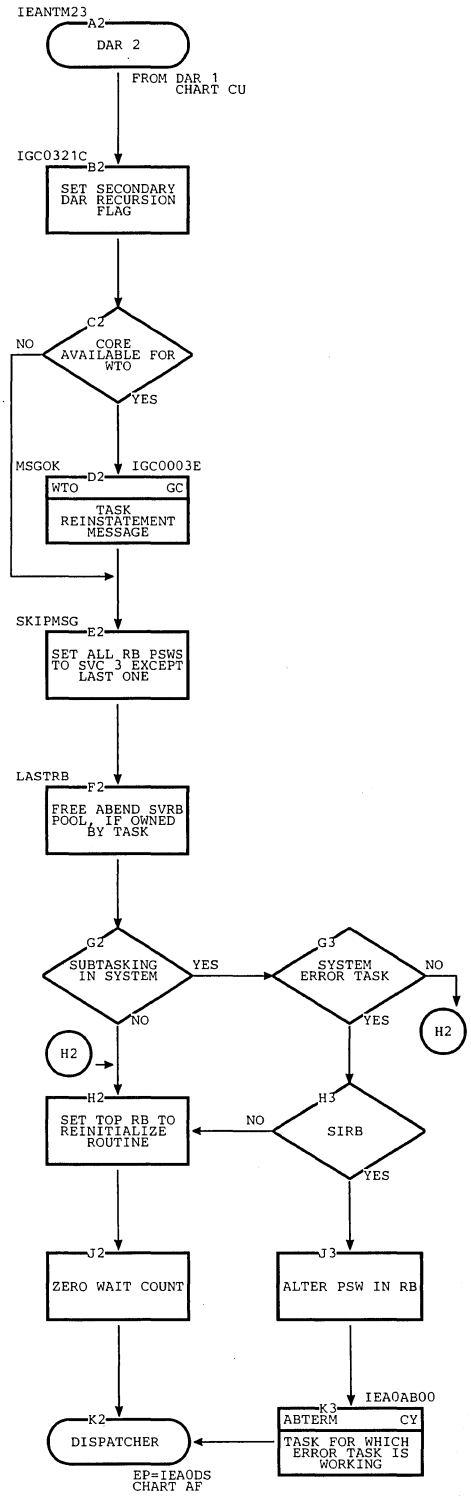


Chart CW. Damage Assessment Routine 3

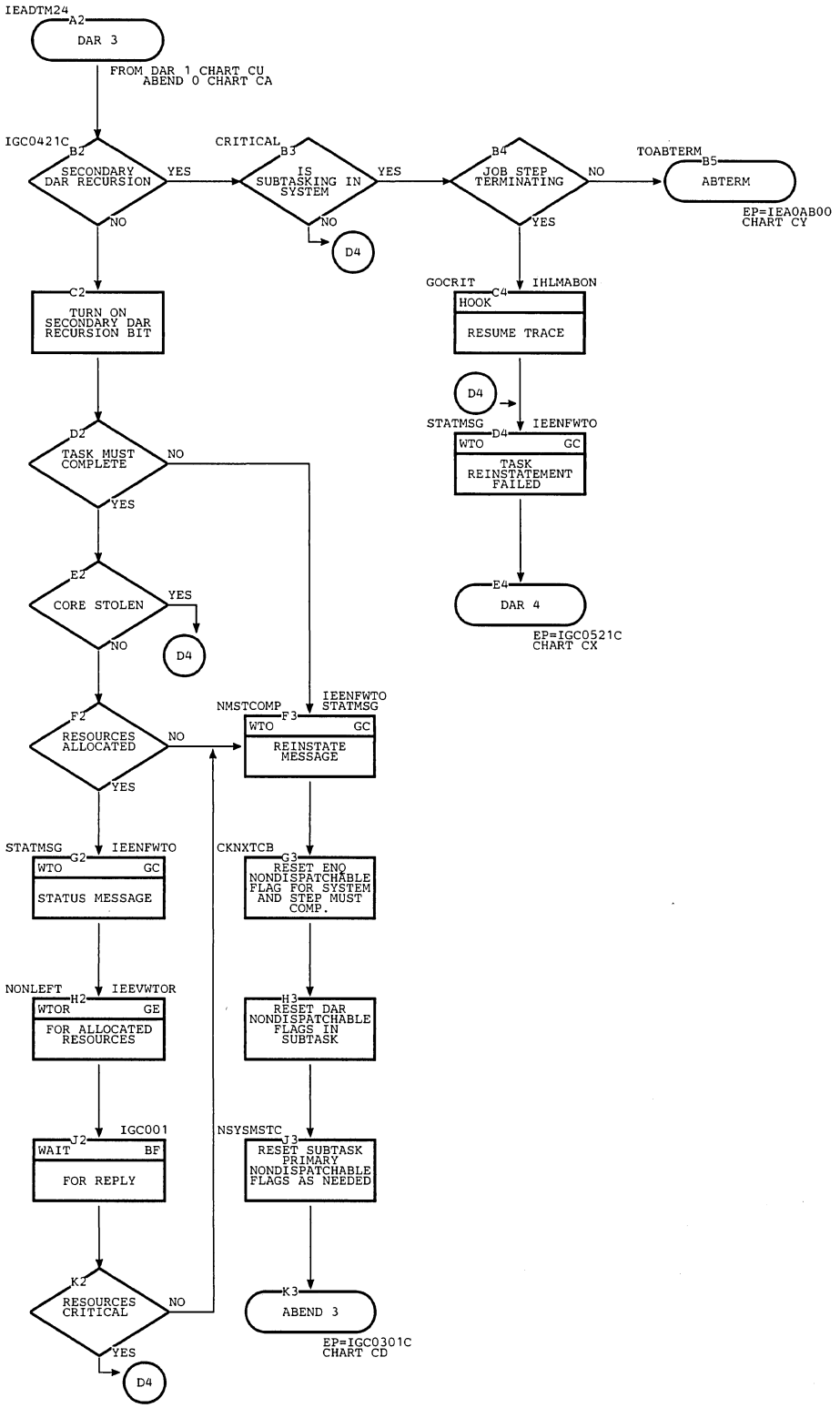


Chart CX. DAR 4

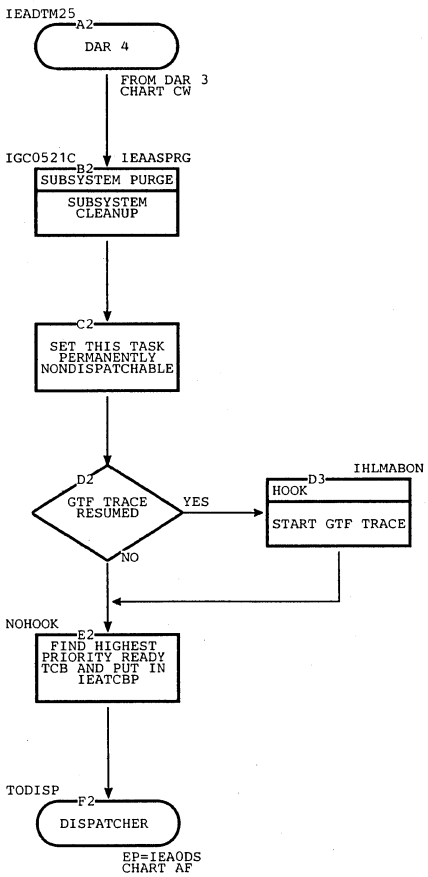


Chart CY. ABTERM

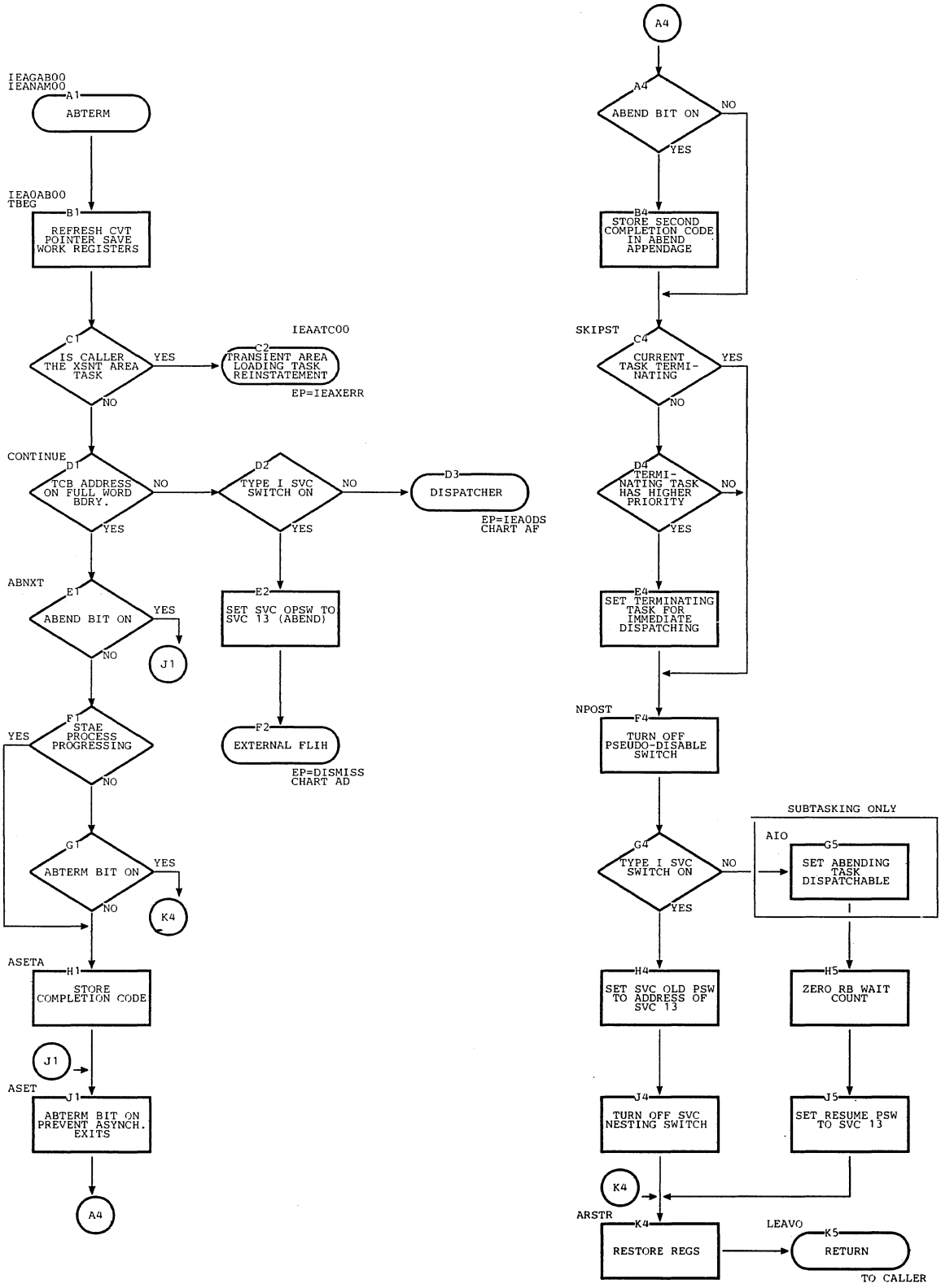


Chart DA. SVC DUMP 1 (Part 1 of 2)

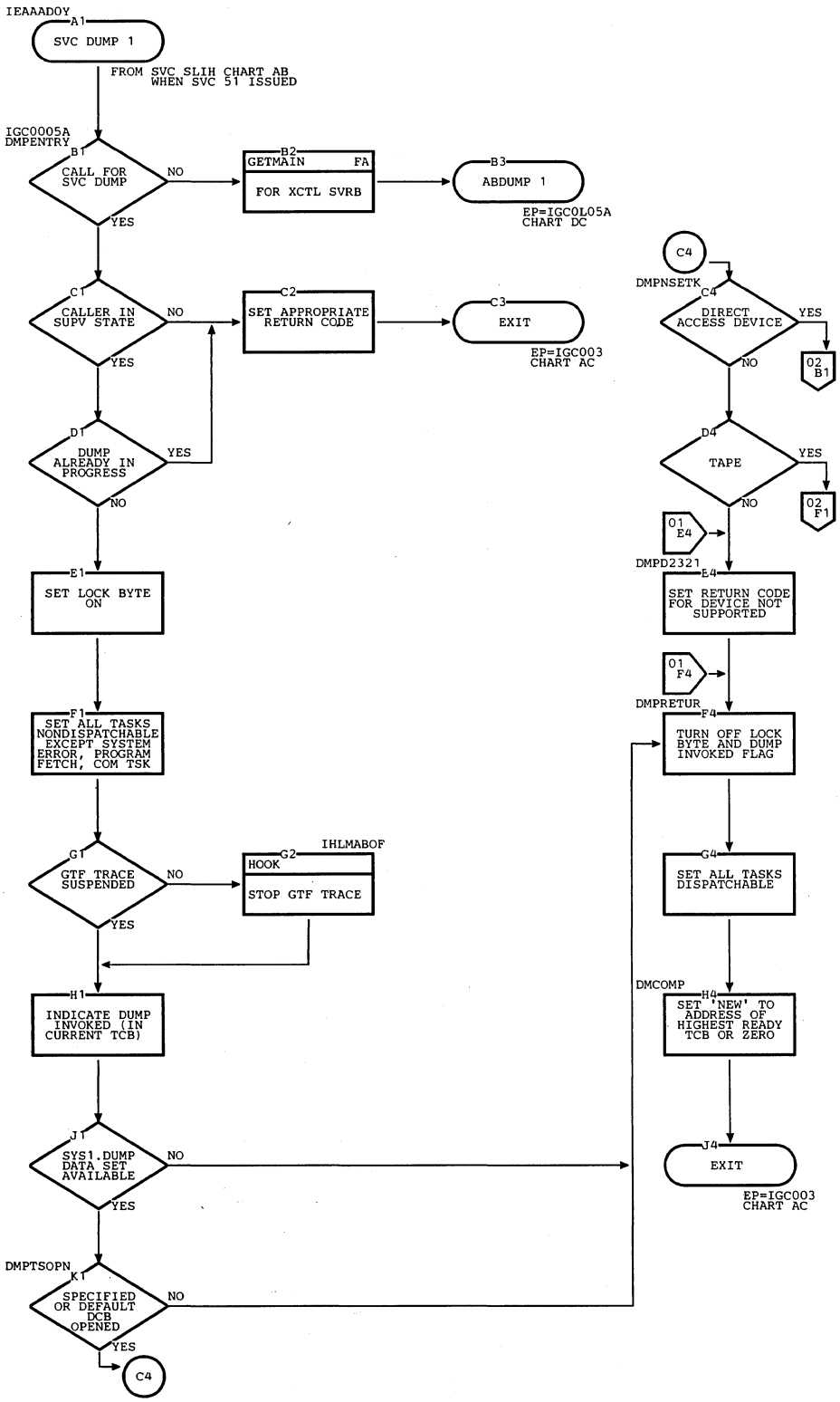


Chart DA. SVC DUMP 1 (Part 2 of 2)

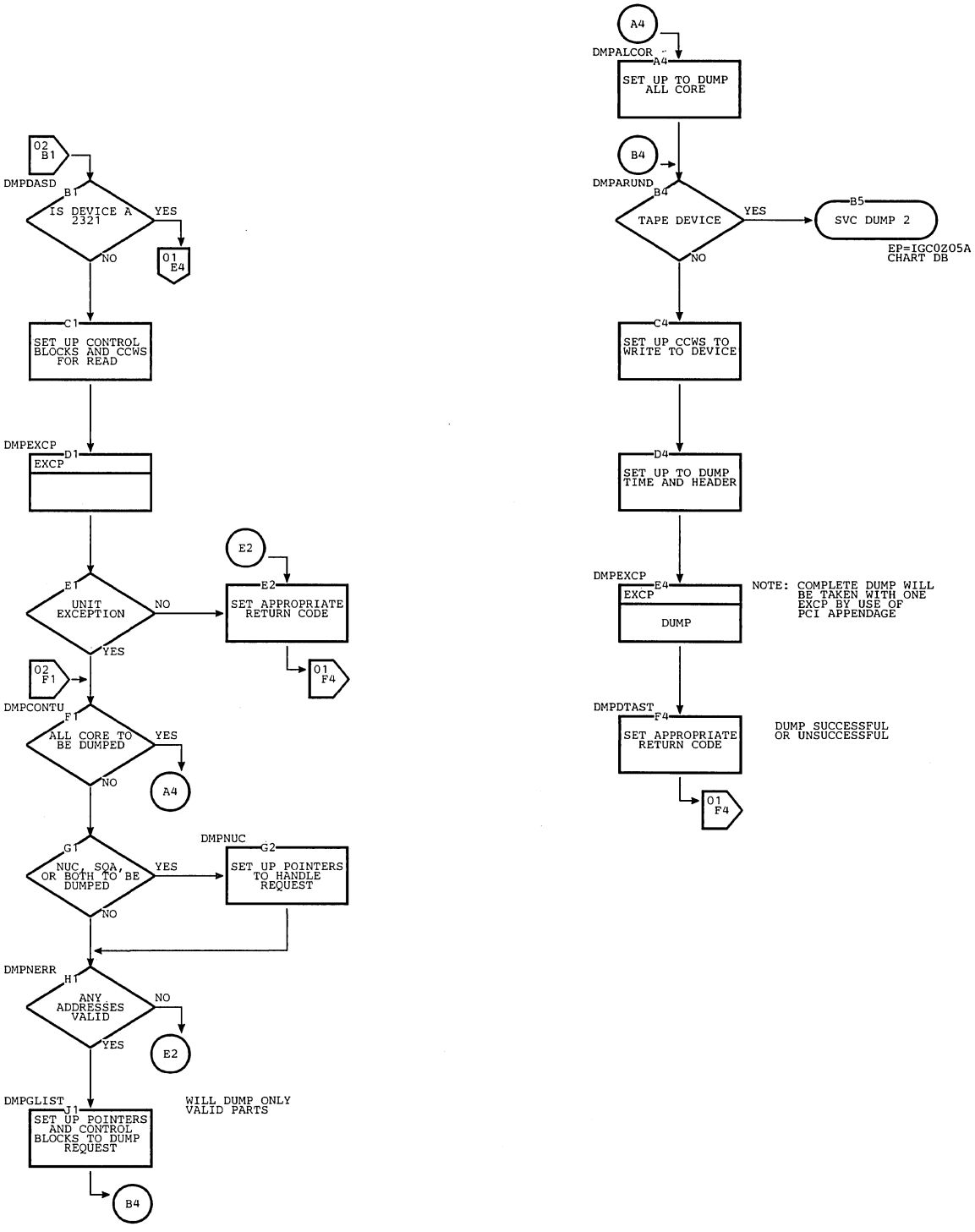


Chart DB. SVC DUMP 2

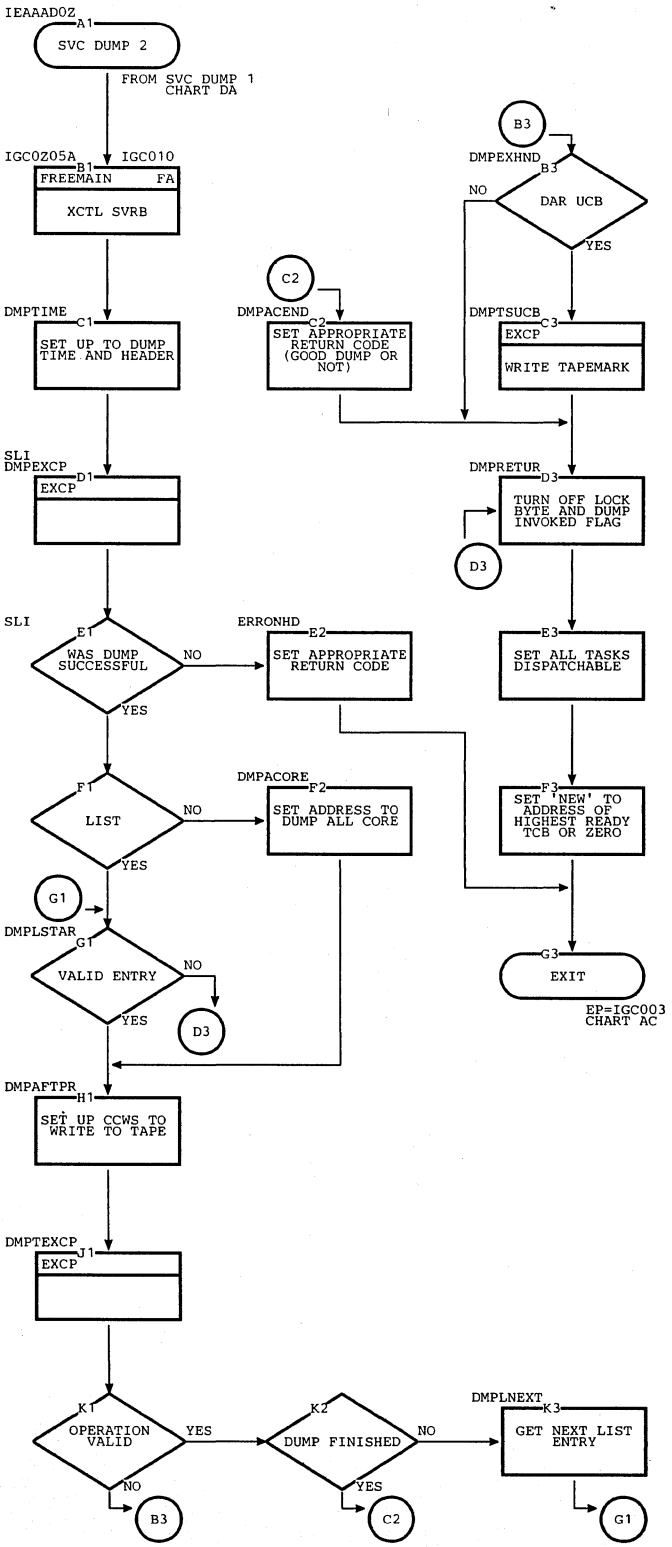


Chart DC. ABDUMP 1

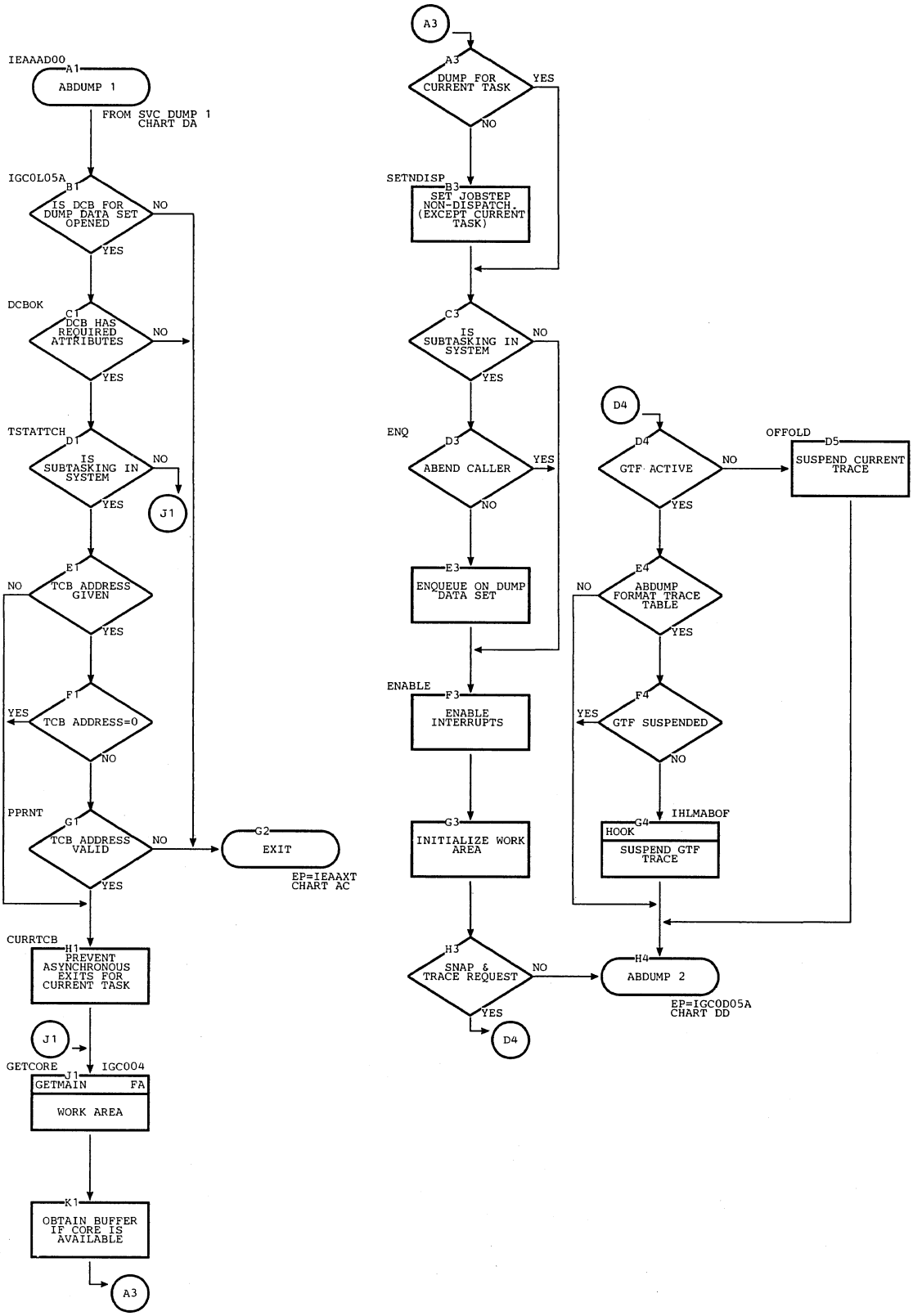


Chart DD. ABDUMP 2 and ABDUMP 3

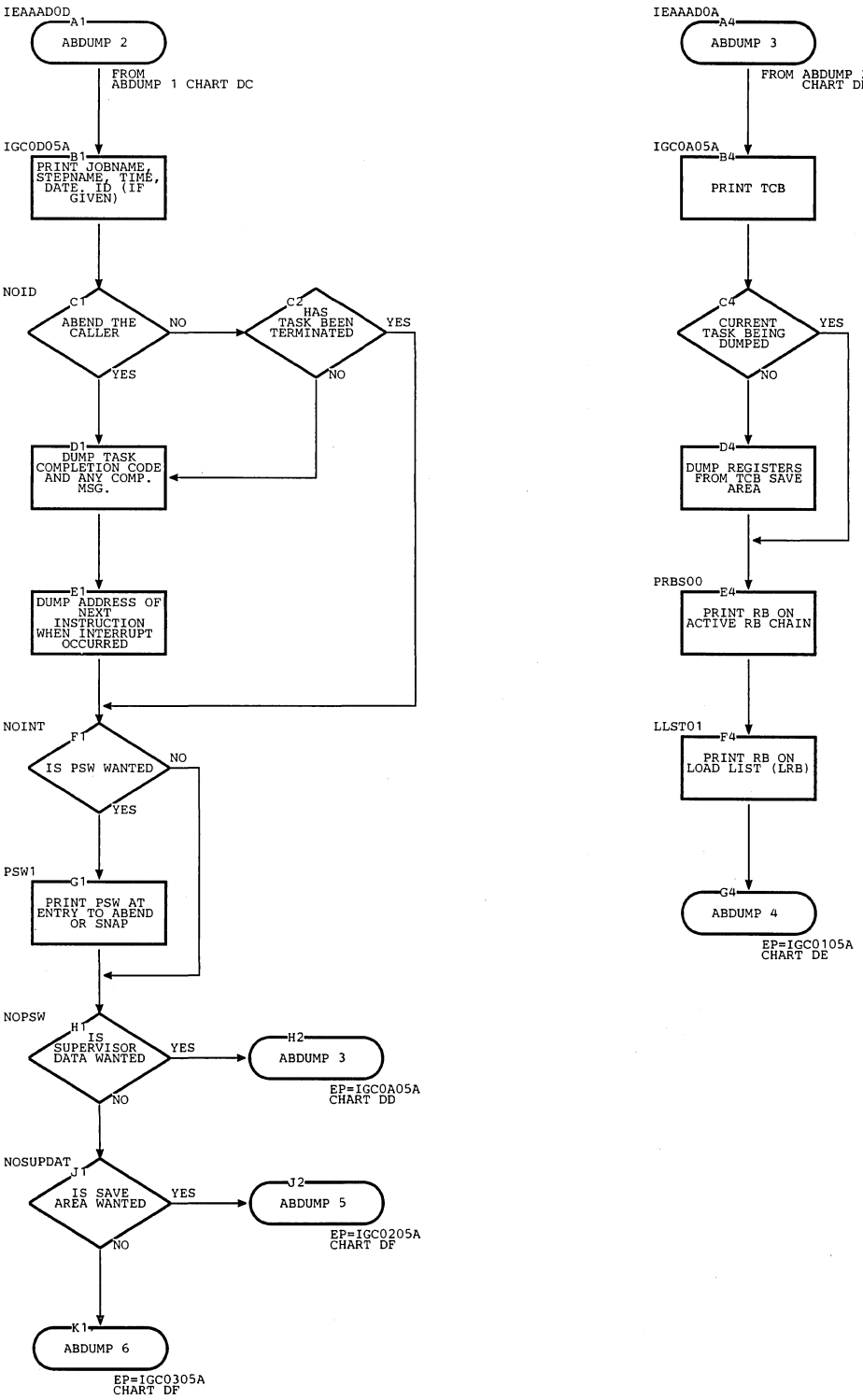


Chart DE. ABDUMP 4

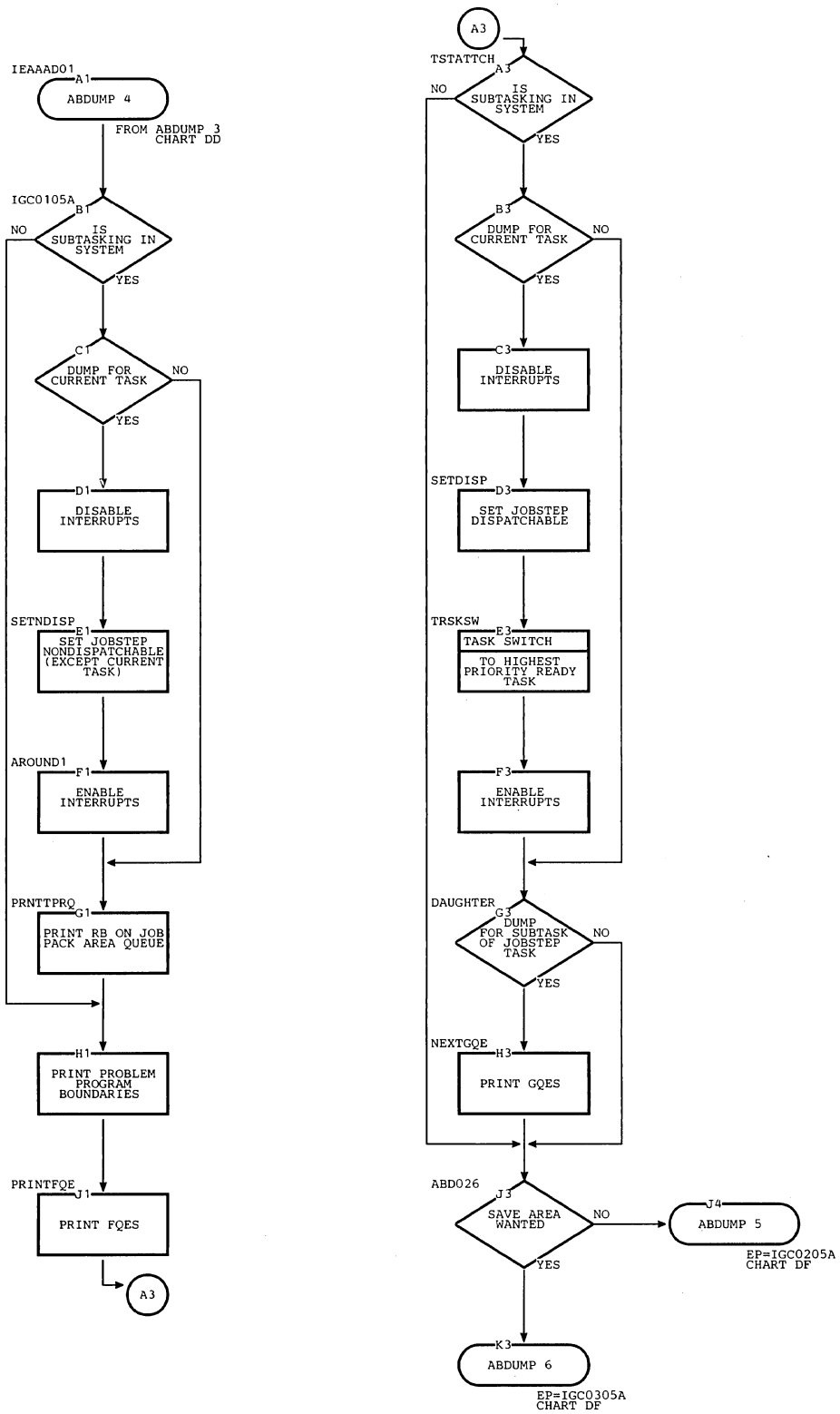


Chart DF. ABDUMP 5 and ABDUMP 6

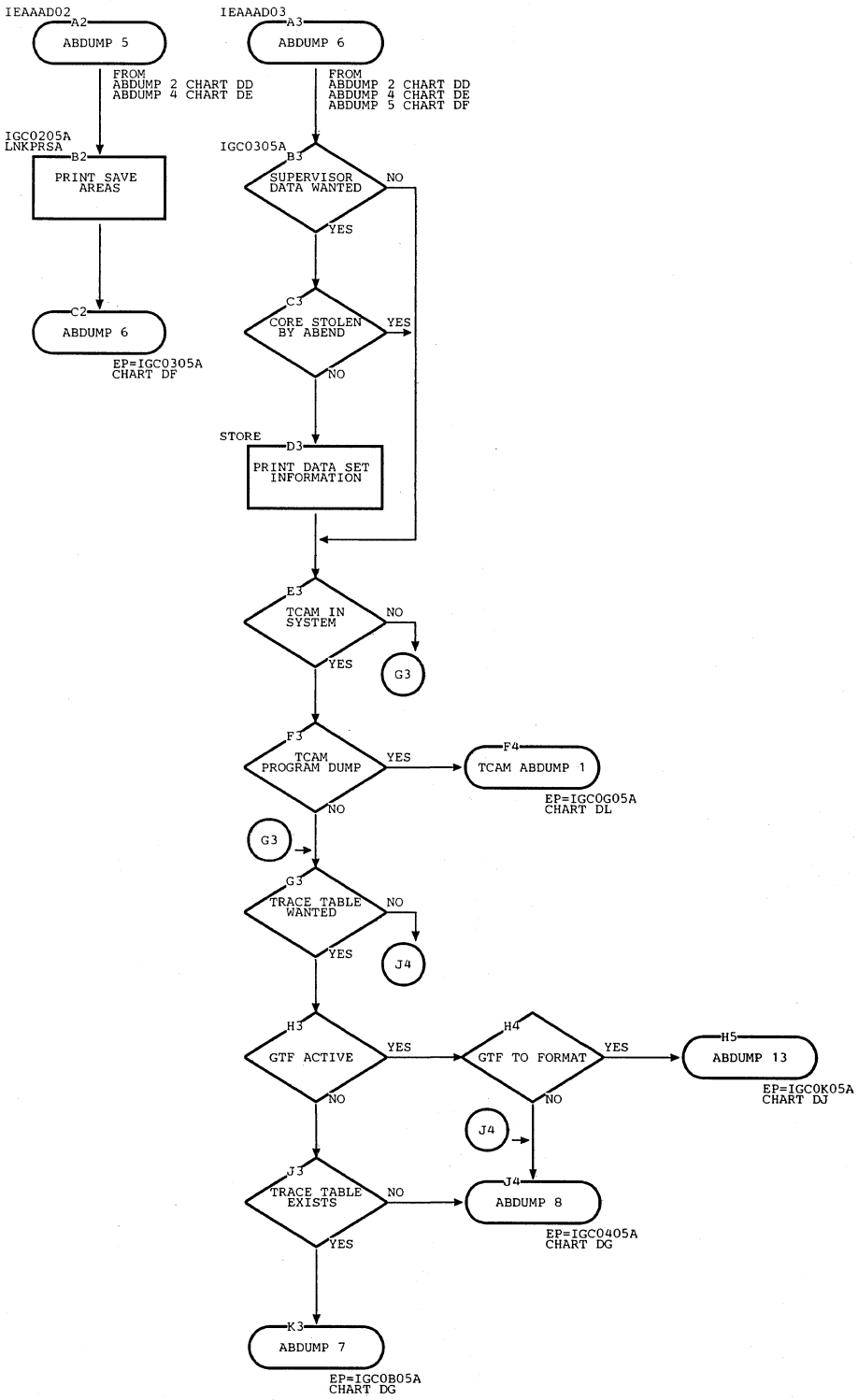


Chart DG. ABDUMP 7 and ABDUMP 8

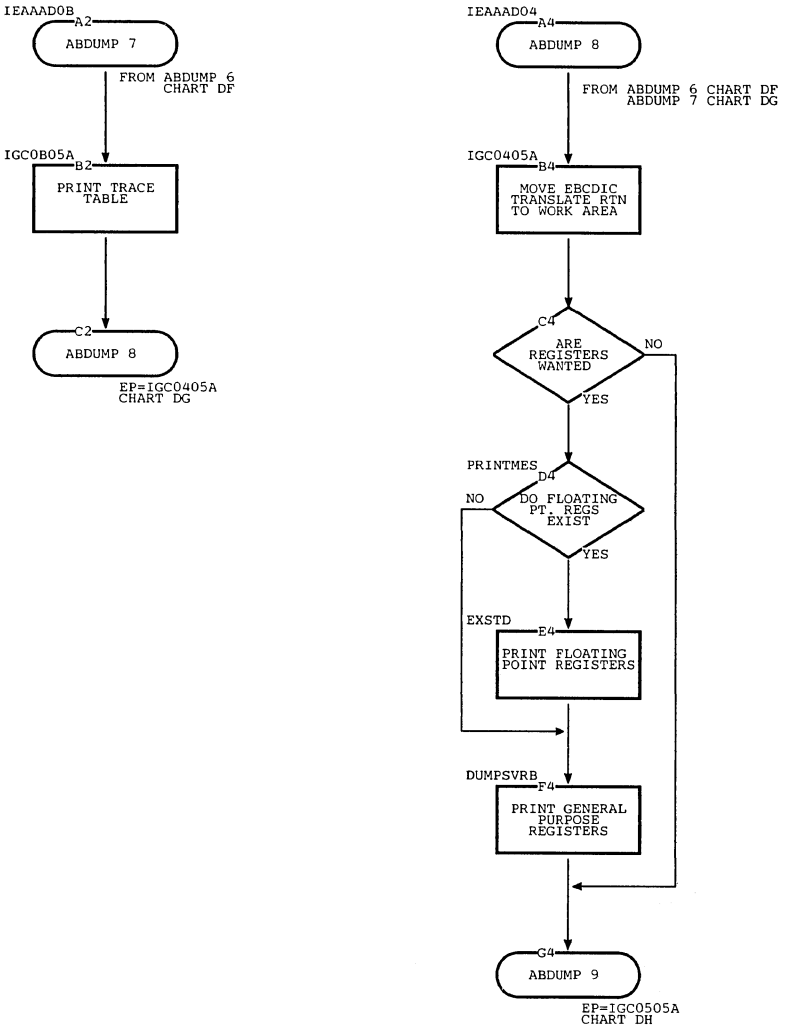


Chart DH. ABDUMP 9, ABDUMP 10, and ABDUMP 11

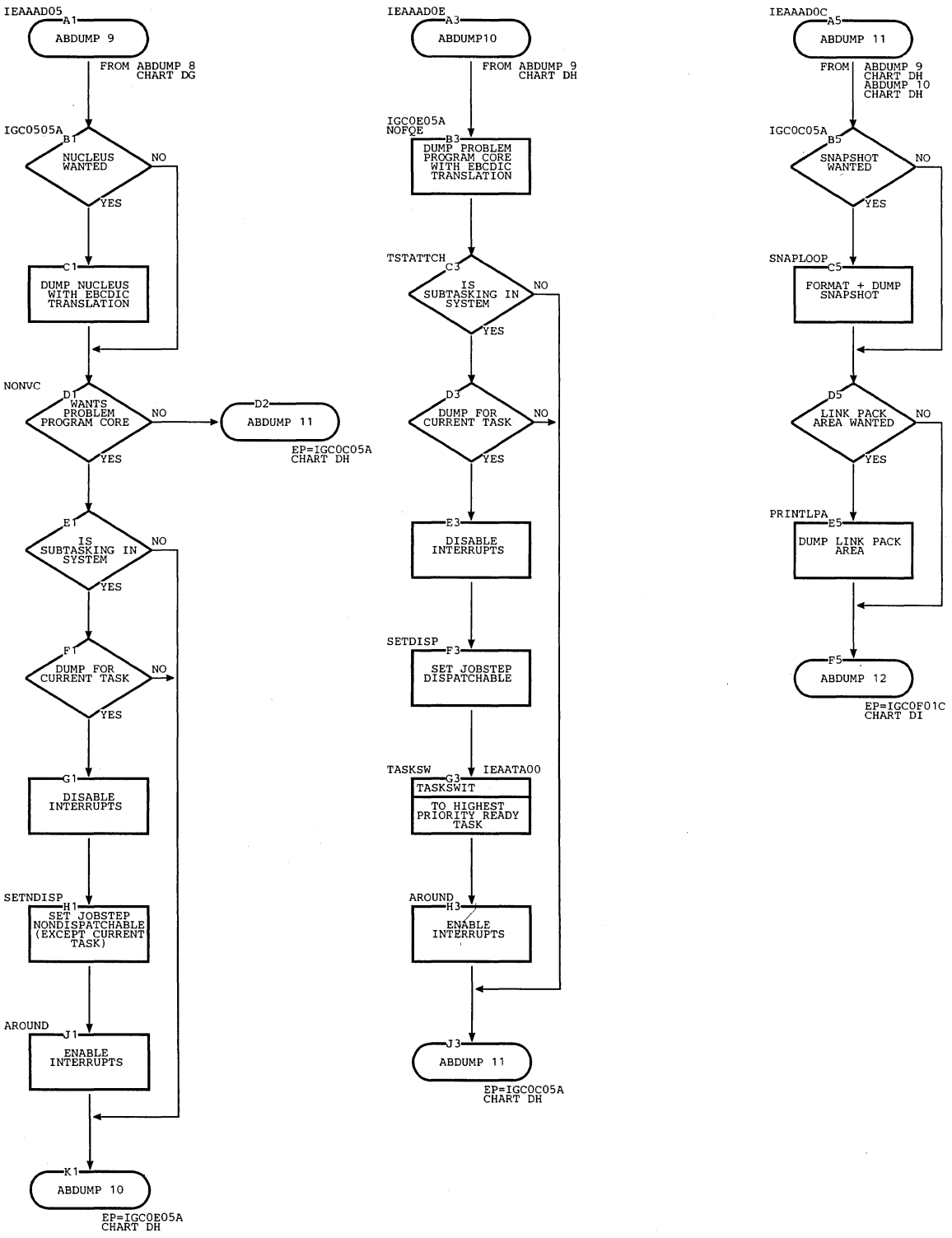


Chart DI. ABDUMP 12

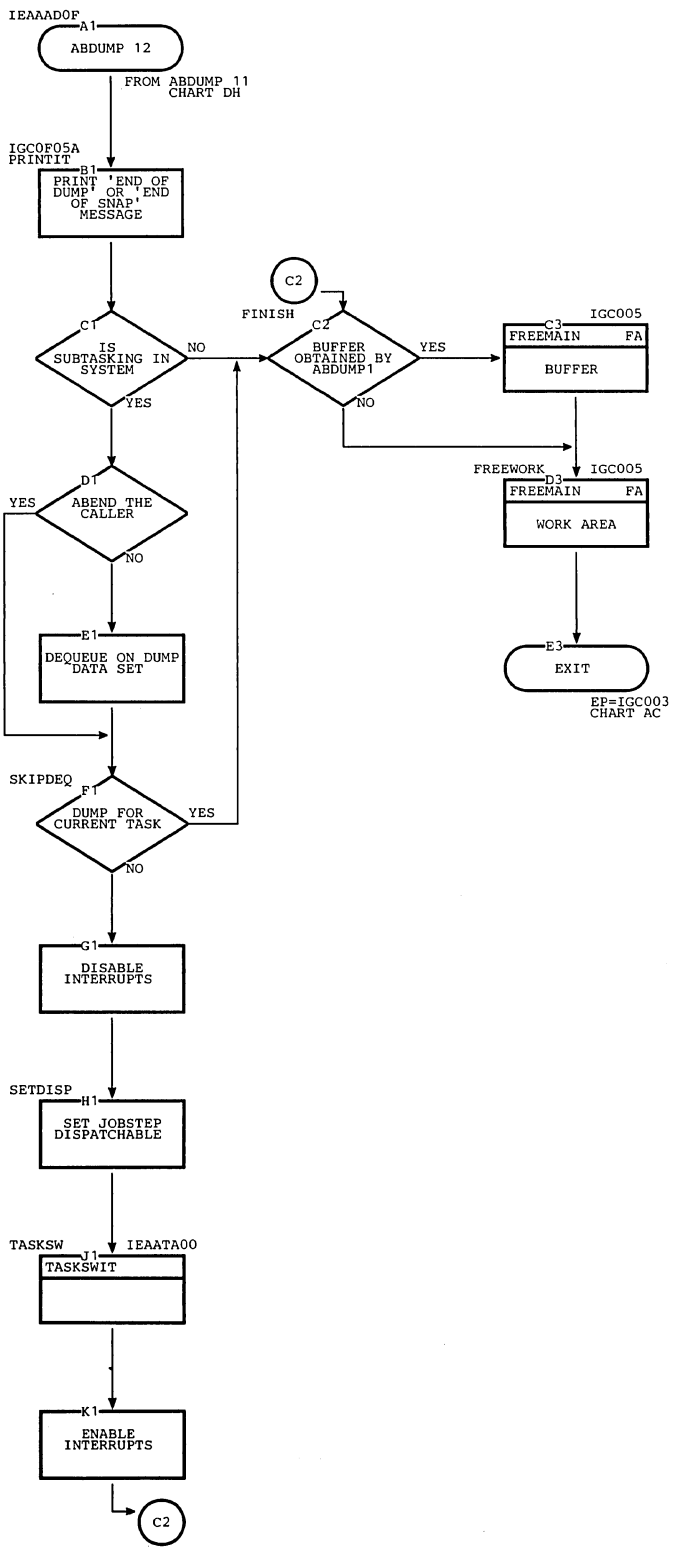


Chart DJ. ABDUMP 13

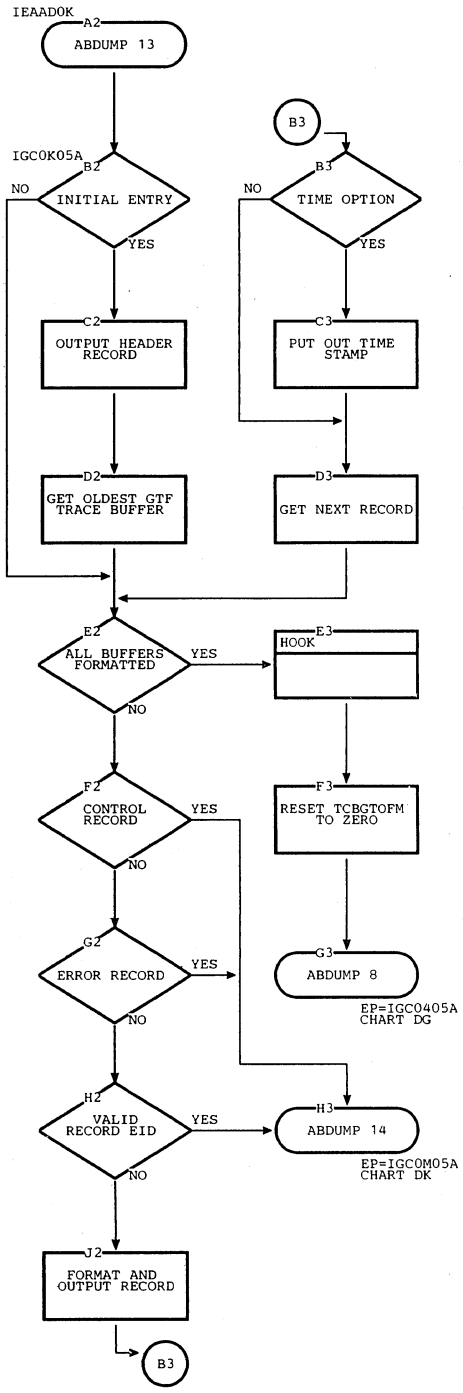


Chart DK. ABDUMP 14

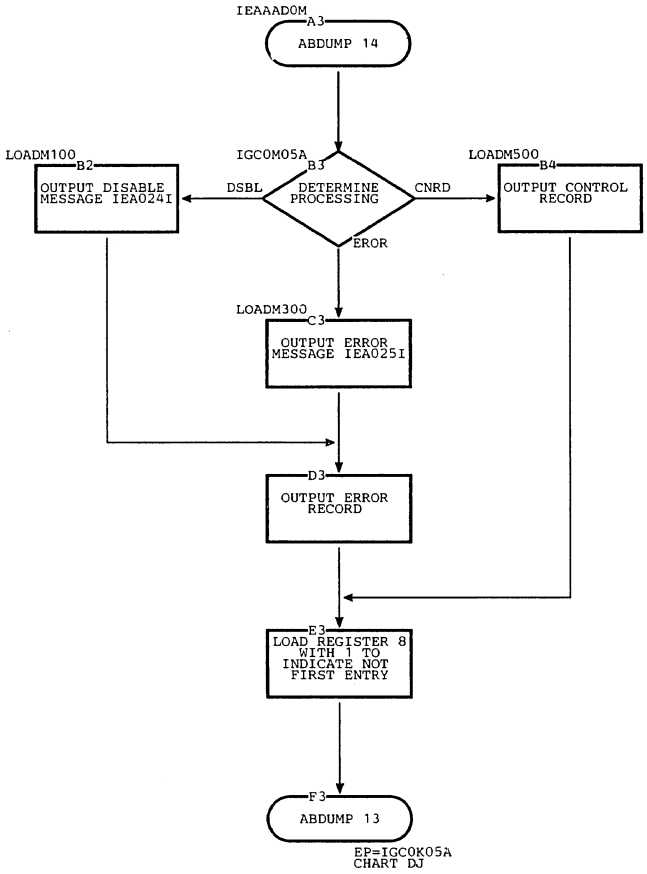


Chart DL. TCAM: ABDUMP 1 and ABDUMP 2

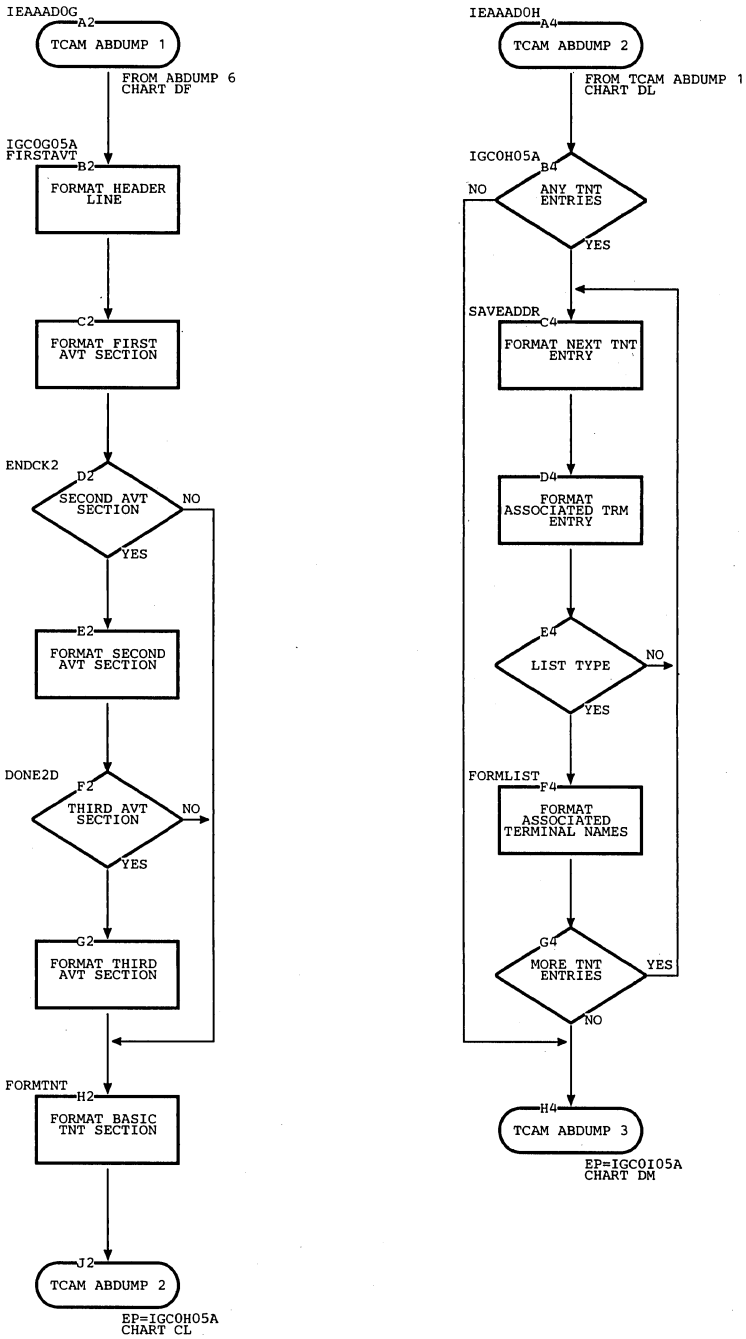


Chart DM. TCAM: ABDUMP 3 and ABDUMP 4

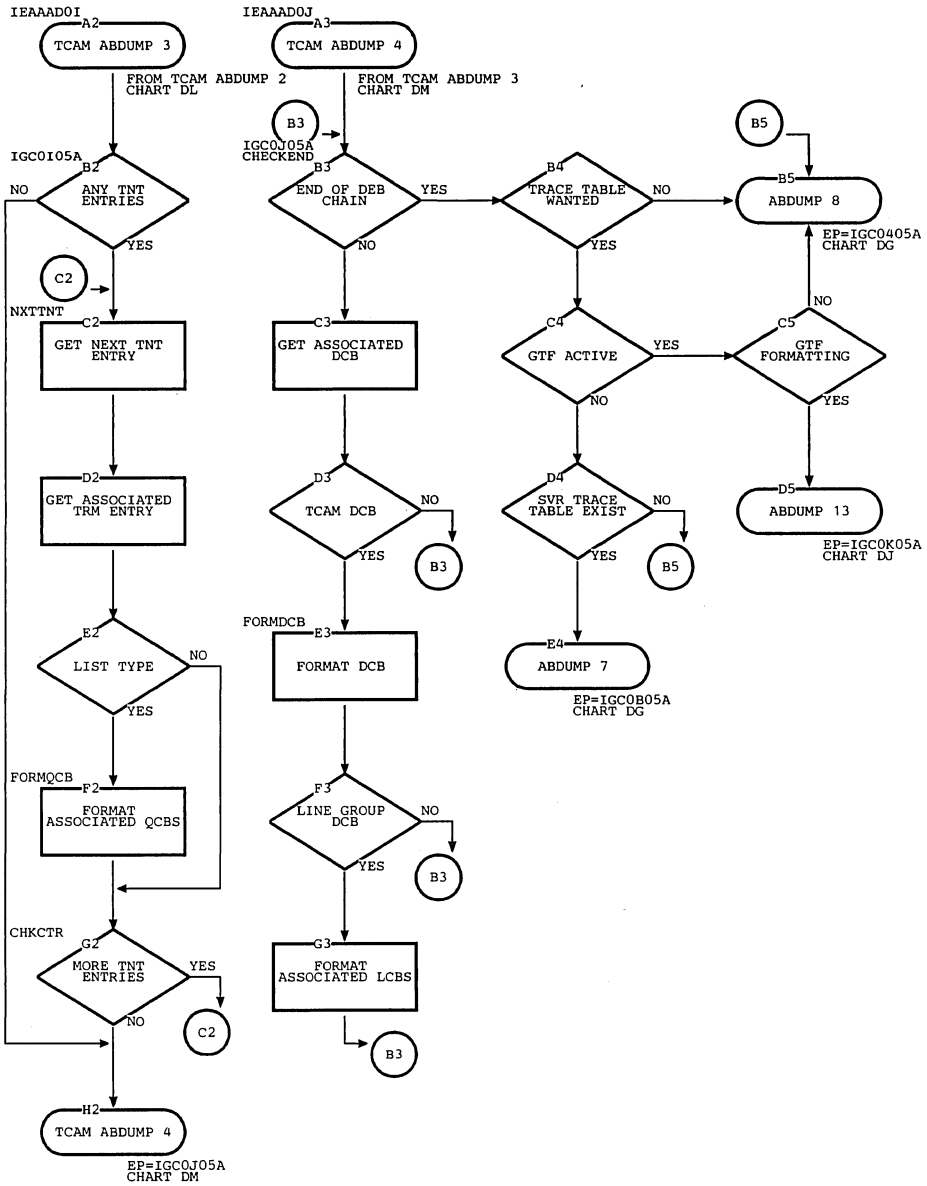


Chart EA. ATTACH without Subtasking

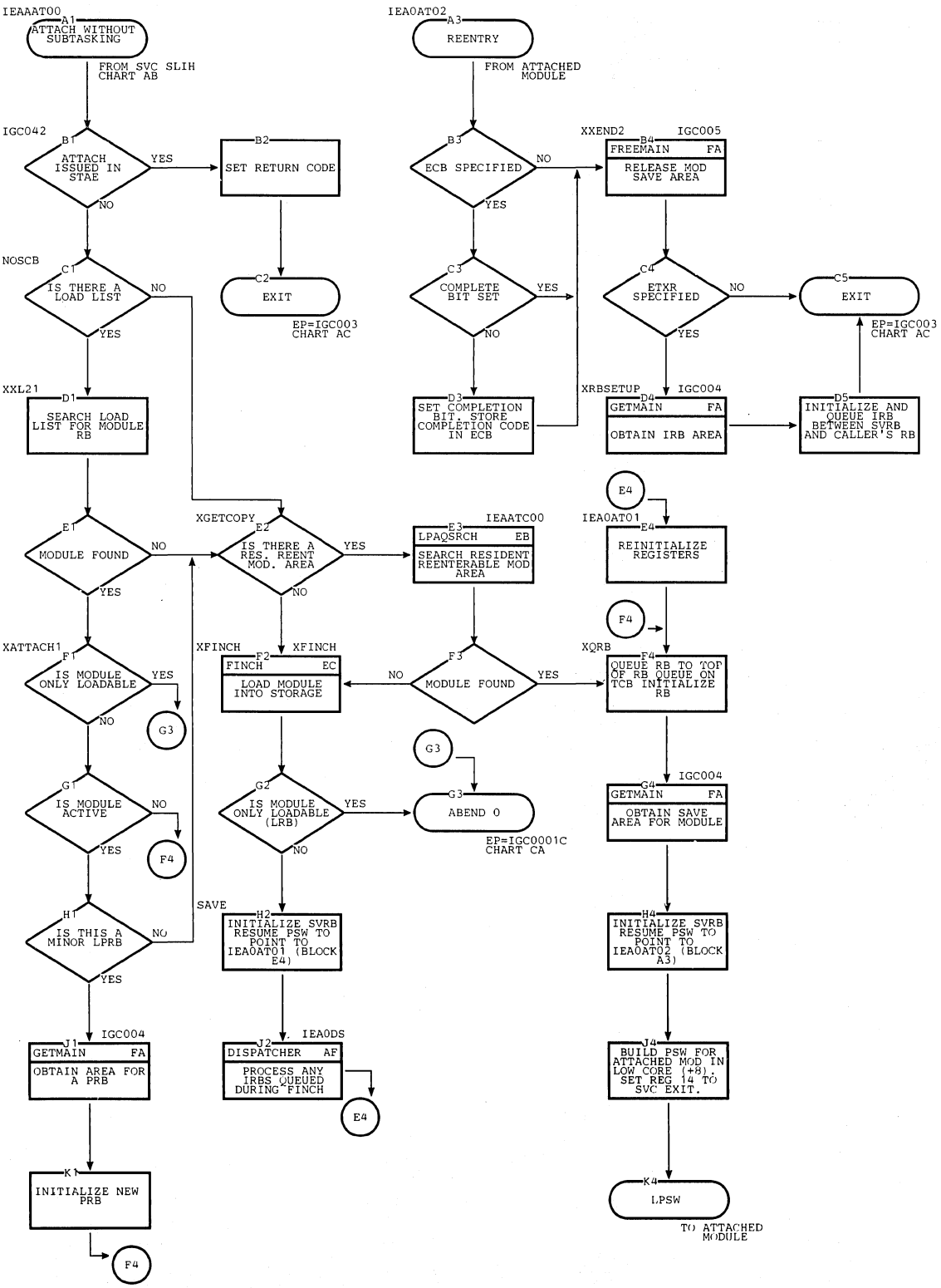


Chart EB. LINK, LOAD, XCTL (Part 1 of 3)

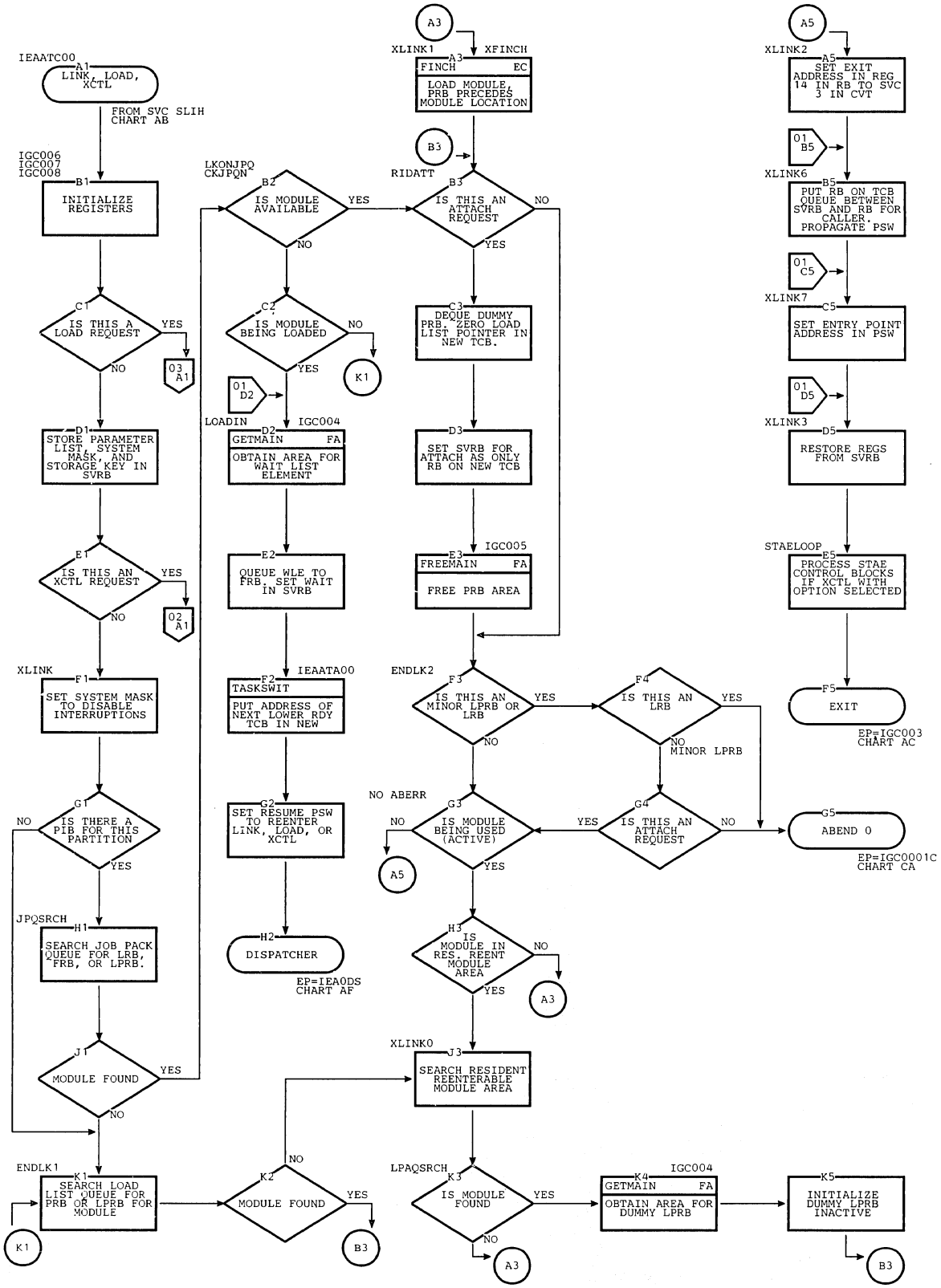


Chart EB. LINK, LOAD, XCTL (Part 2 of 3)

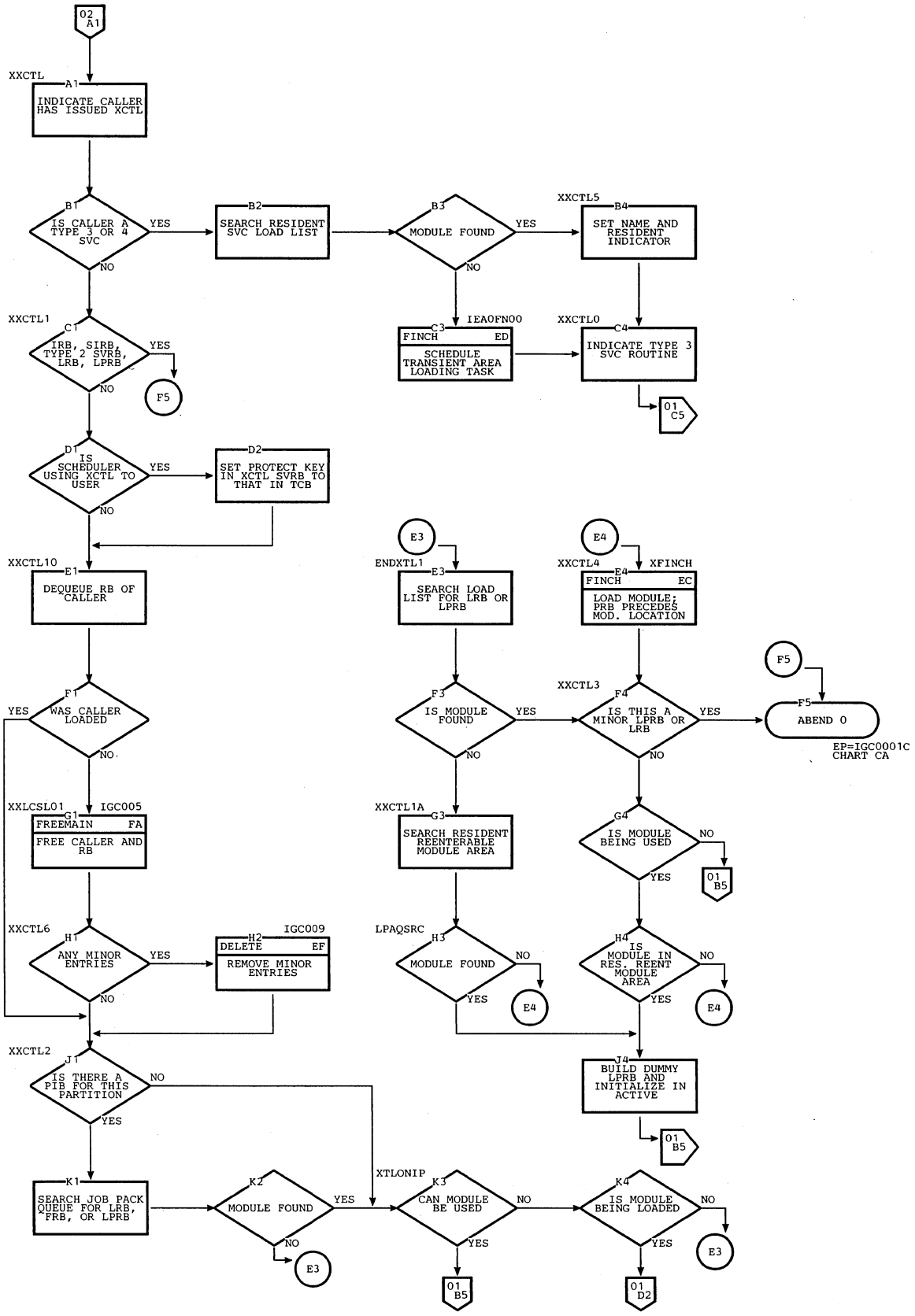


Chart EB. LINK, LOAD, XCTL (Part 3 of 3)

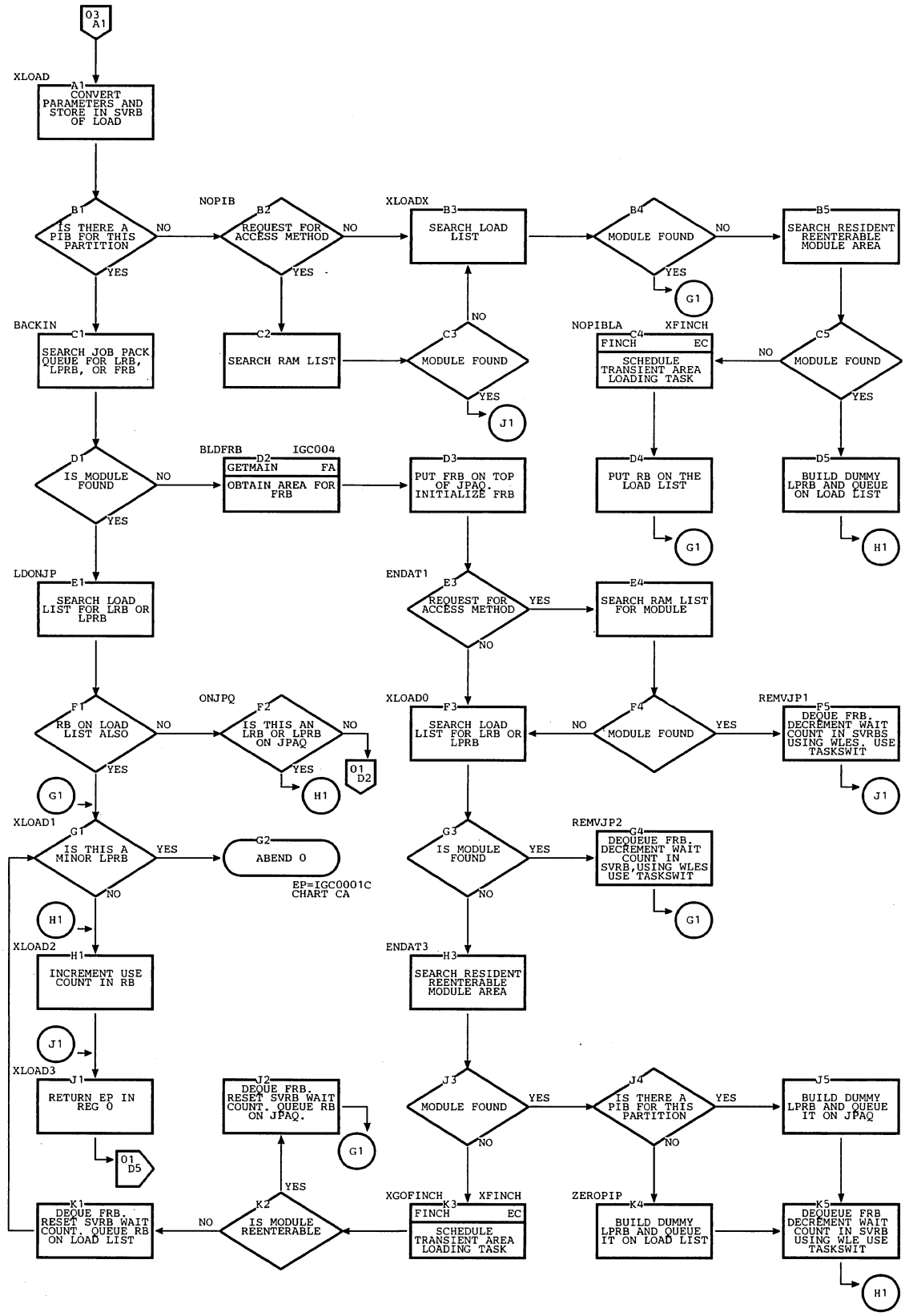


Chart EC. FINCH: Processing for LINK, LOAD, ATTACH, and Nontransient Area XCTL

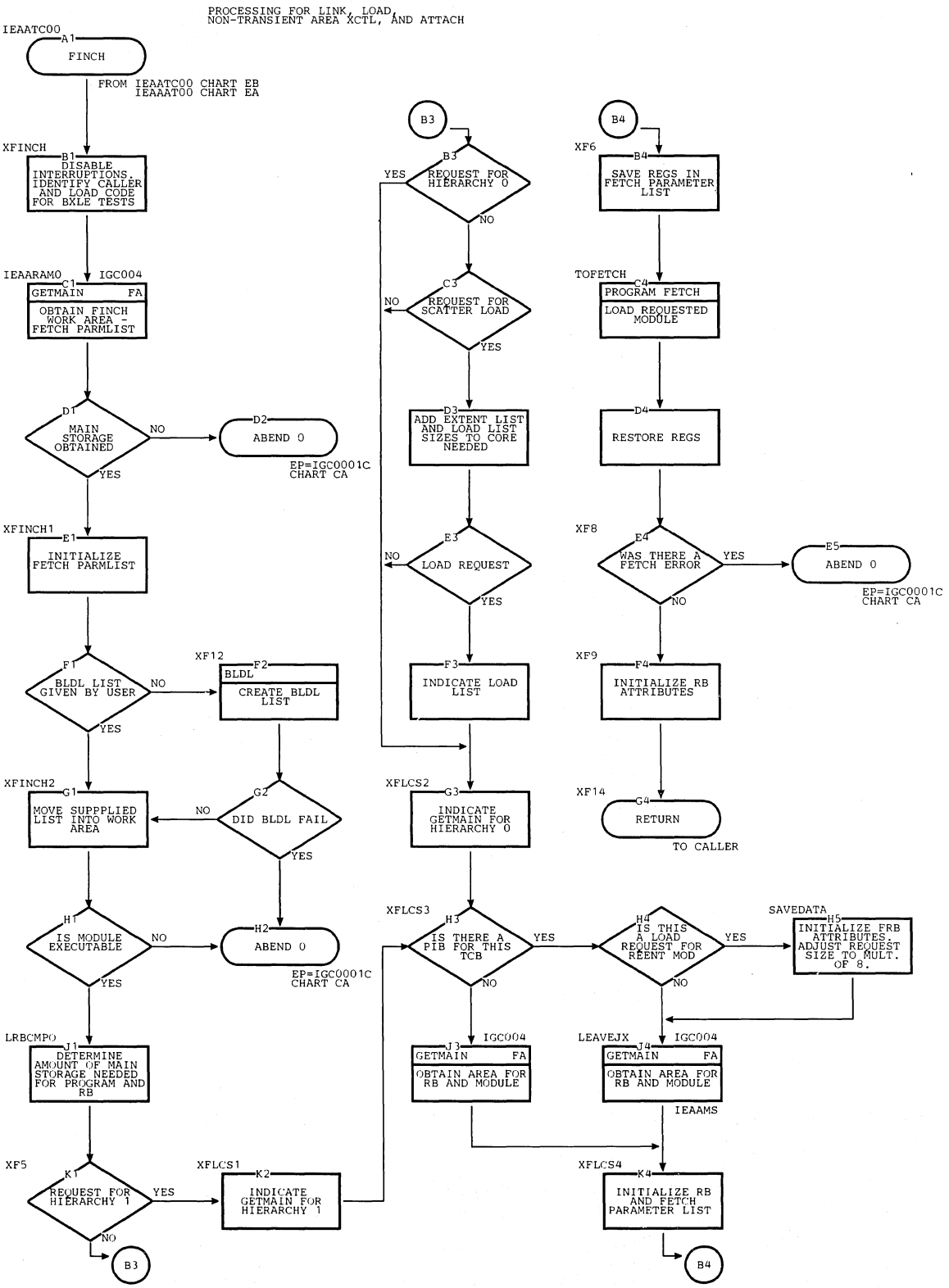


Chart ED. FINCH: Processing for SVC and I/O Transient Area Requests (Part 1 of 2)

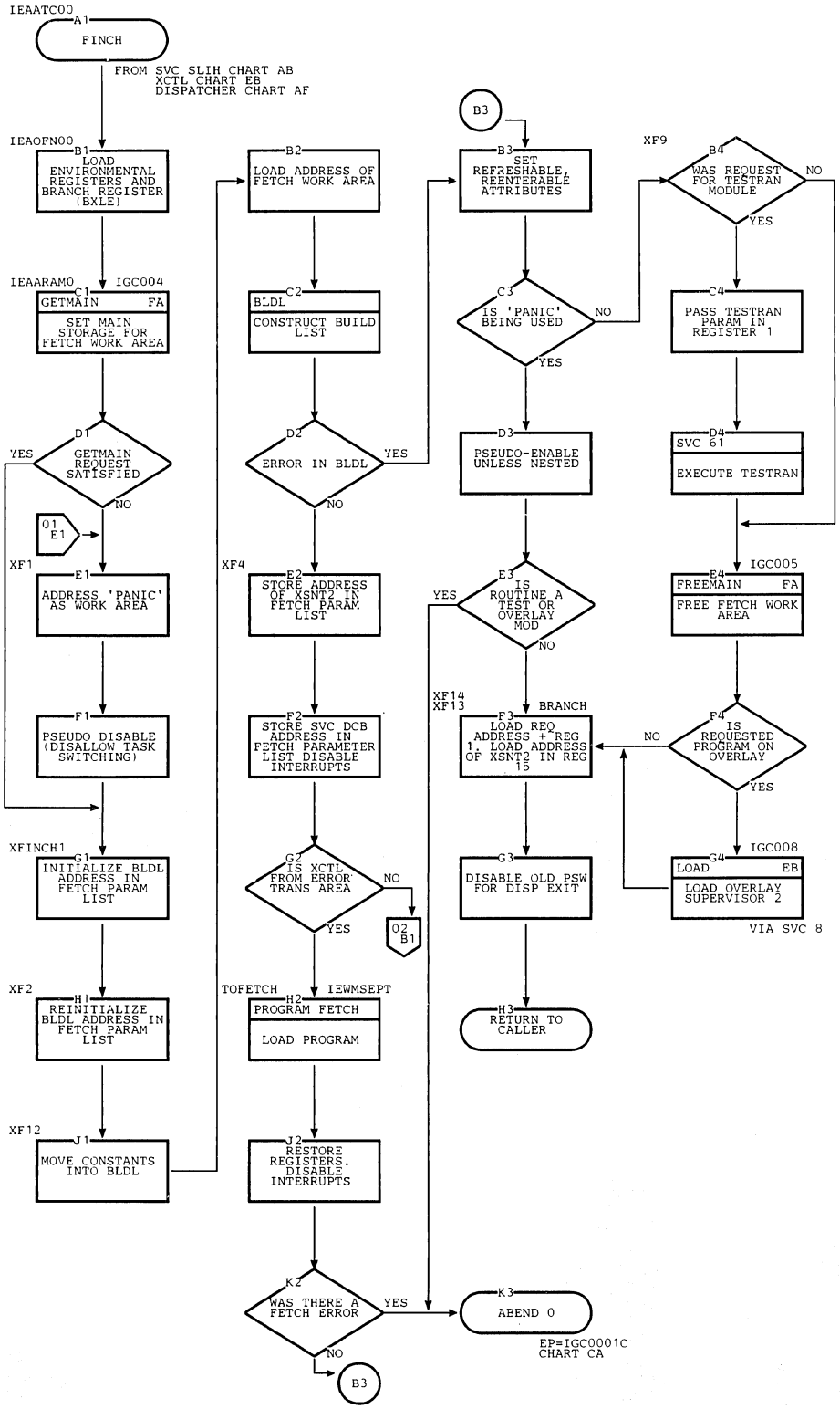


Chart ED. FINCH: Processing for SVC and I/O Transient Area Requests (Part 2 of 2)

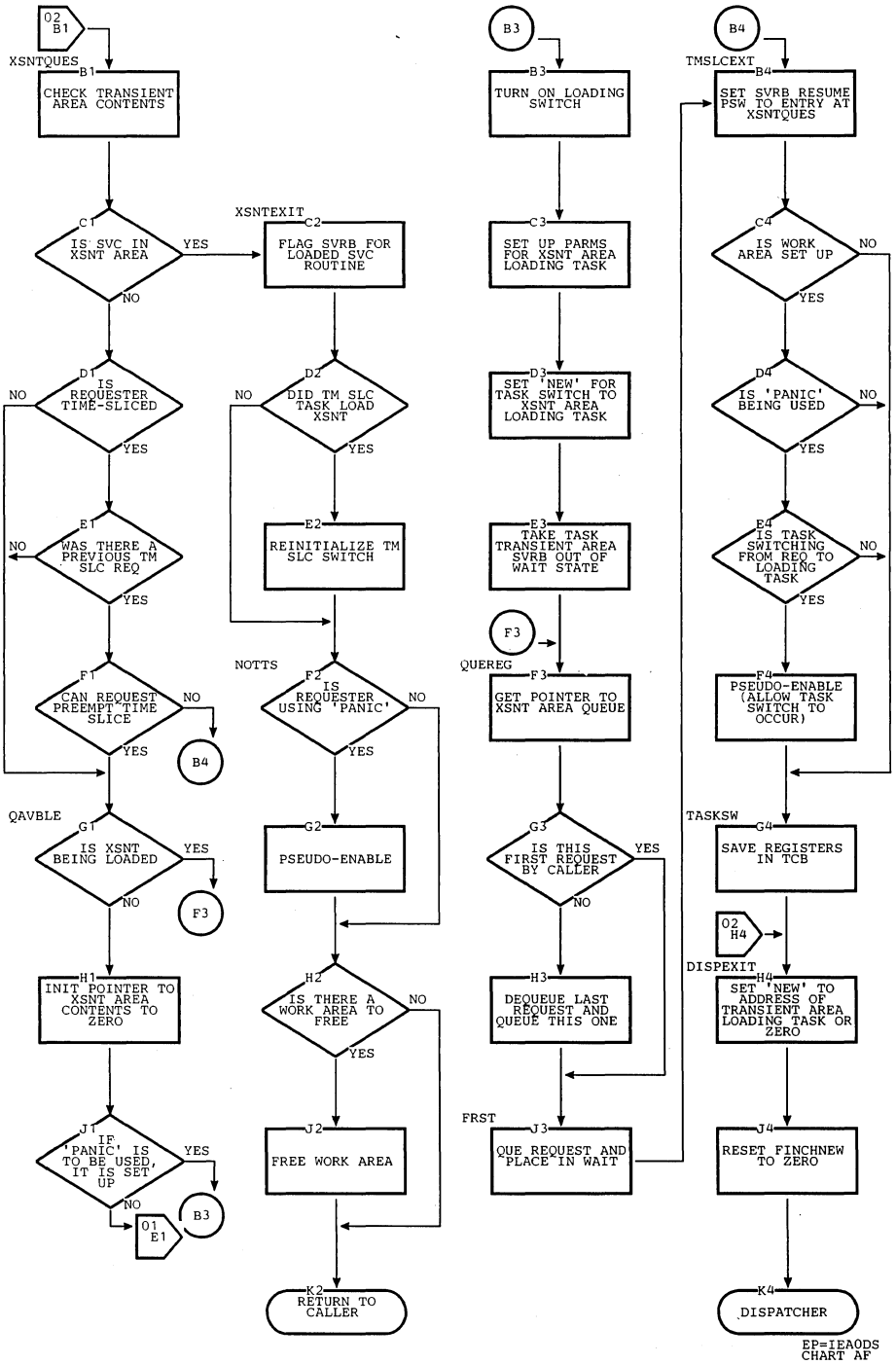


Chart ED. Transient Area Loading Task within FINCH

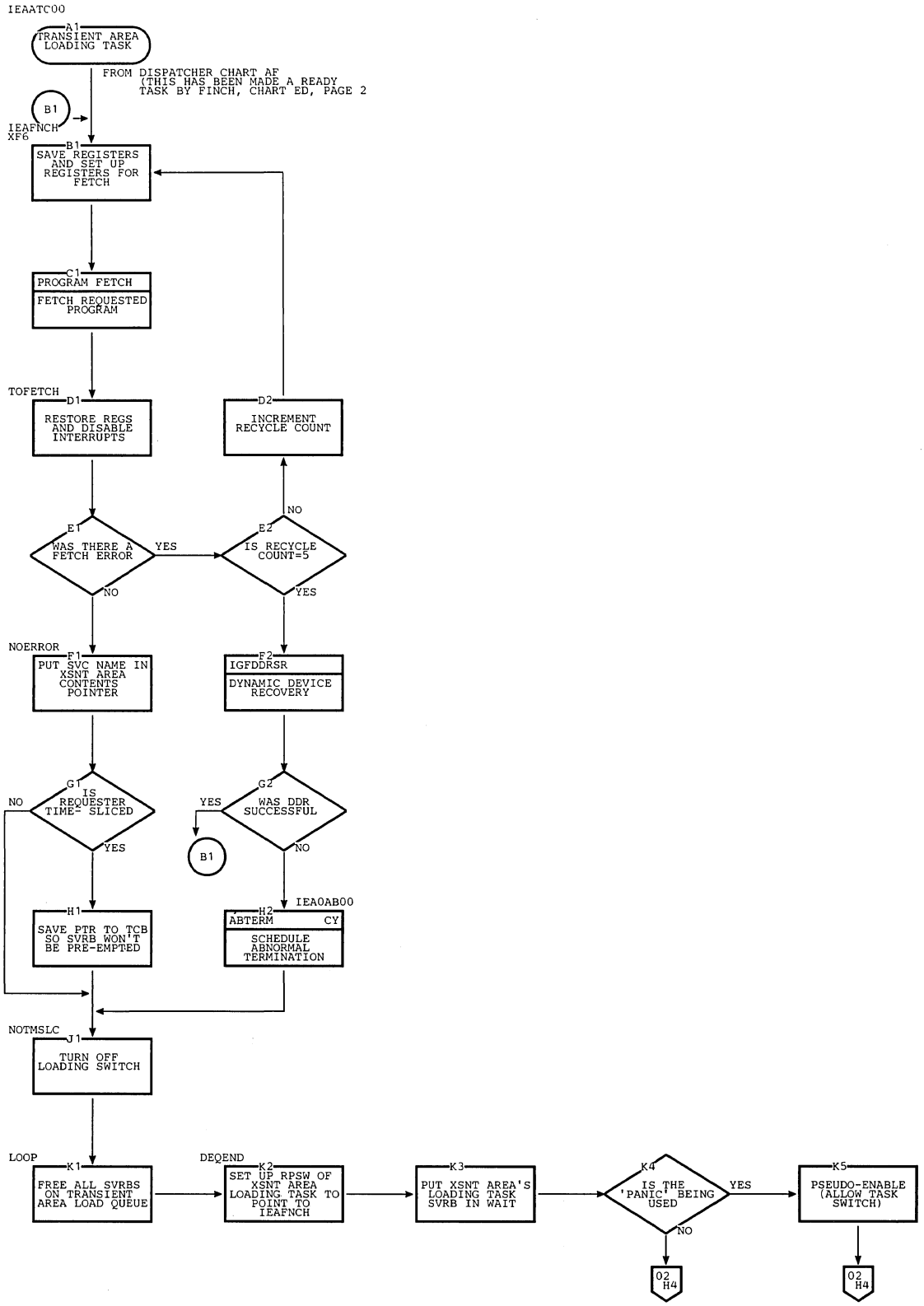


Chart EE. IDENTIFY

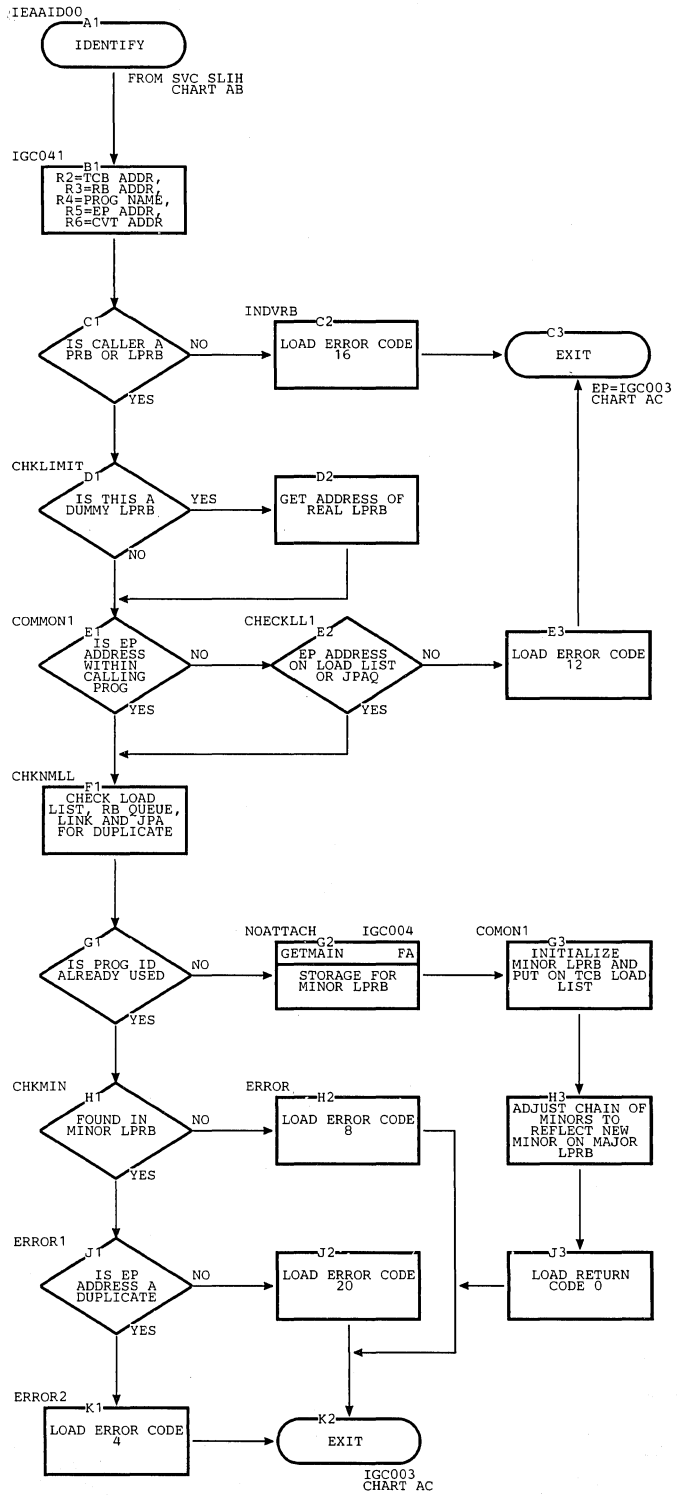


Chart EF. DELETE

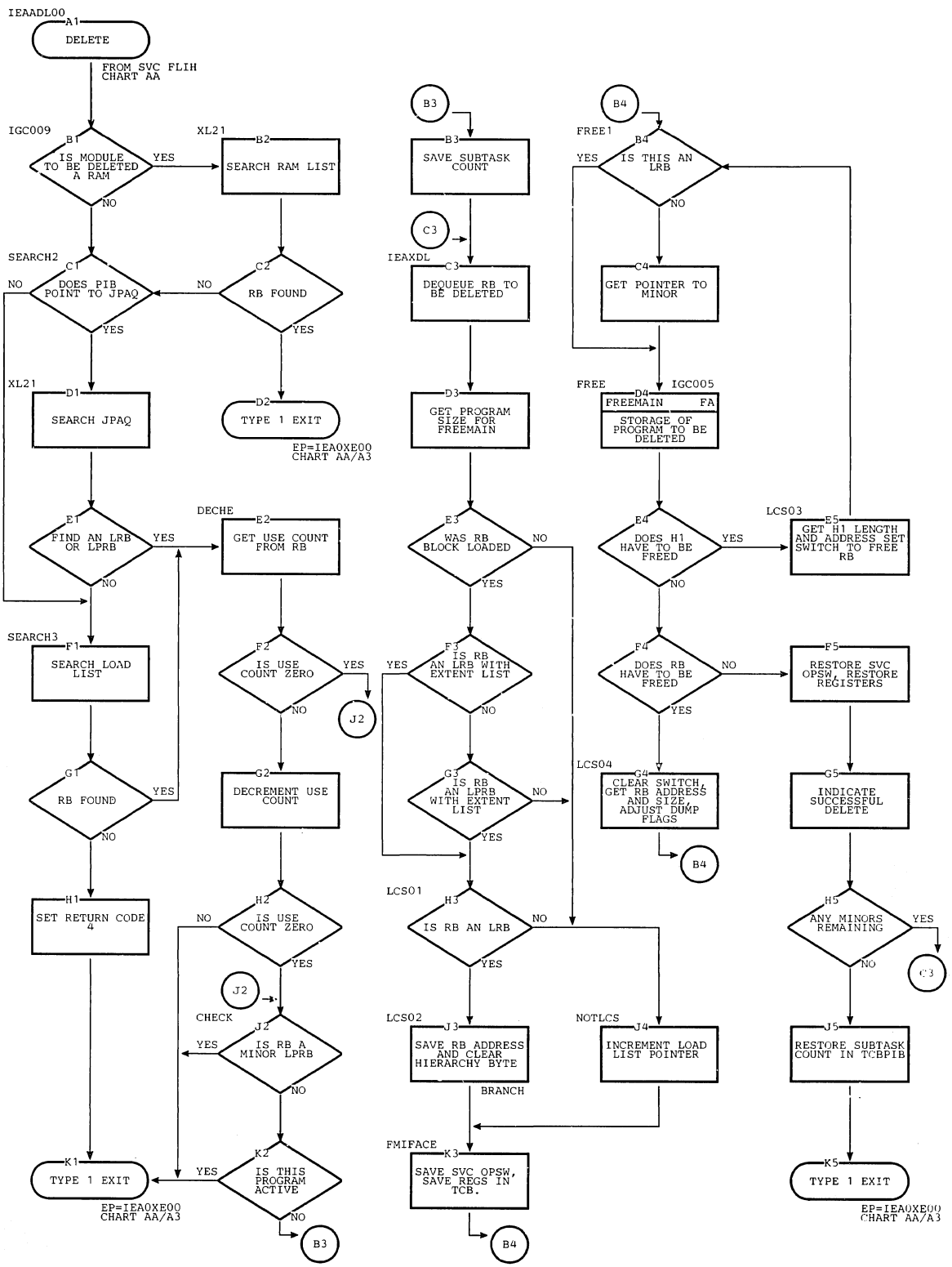


Chart FA. GETMAIN/FREEMAIN (Part 1 of 3)

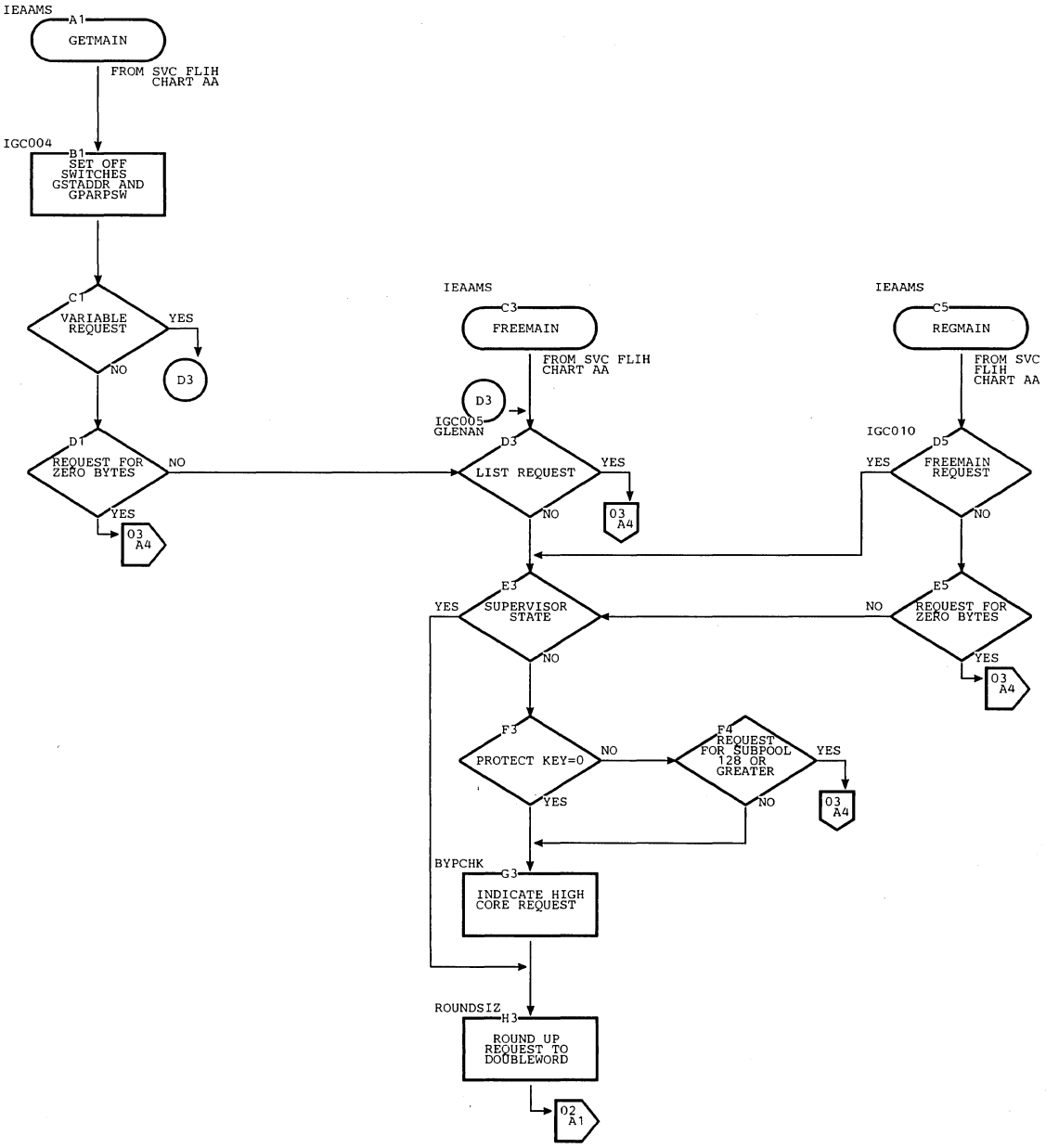


Chart FA. GETMAIN/FREEMAIN (Part 2 of 3)

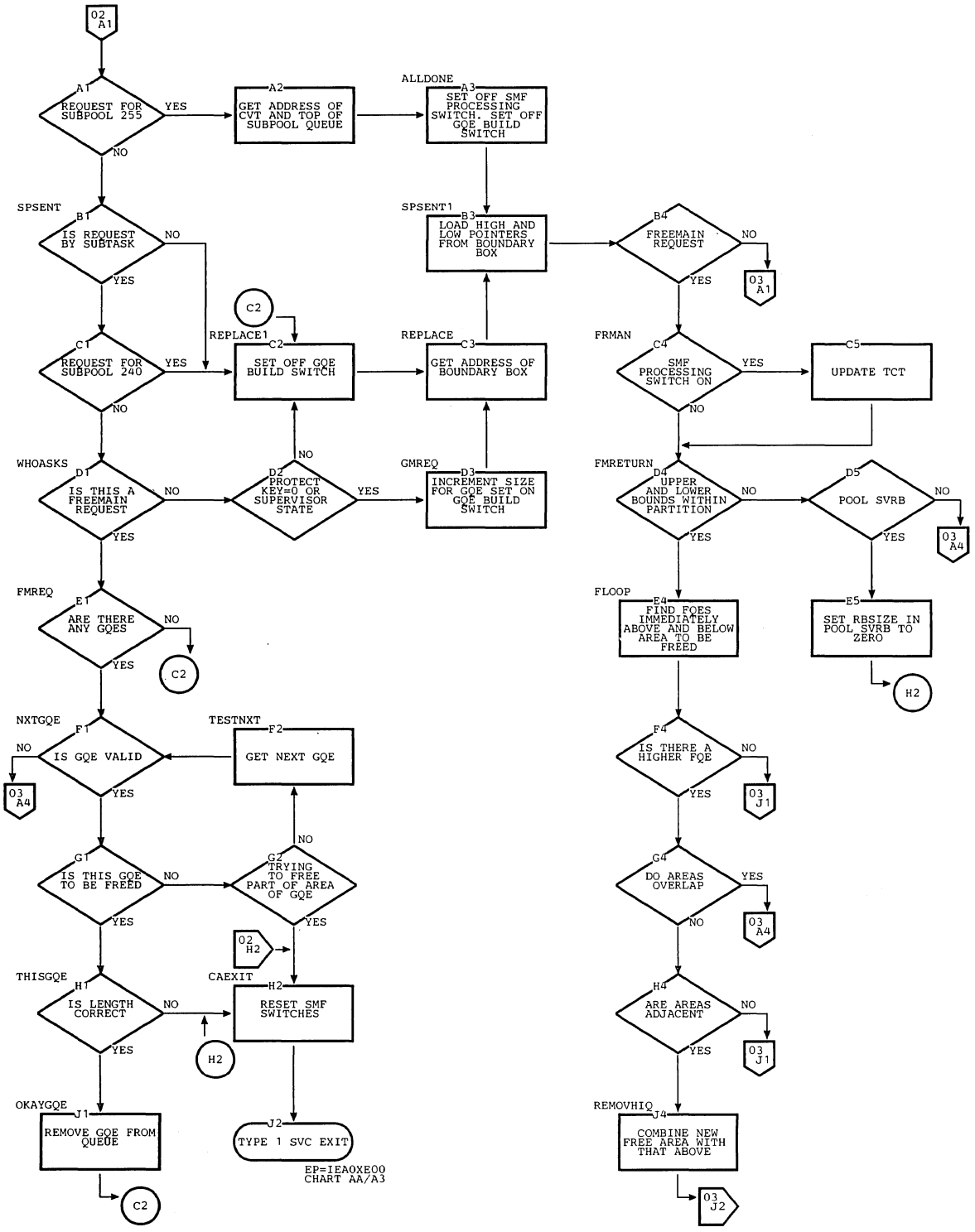


Chart FA. GETMAIN/FREEMAIN (Part 3 of 3)

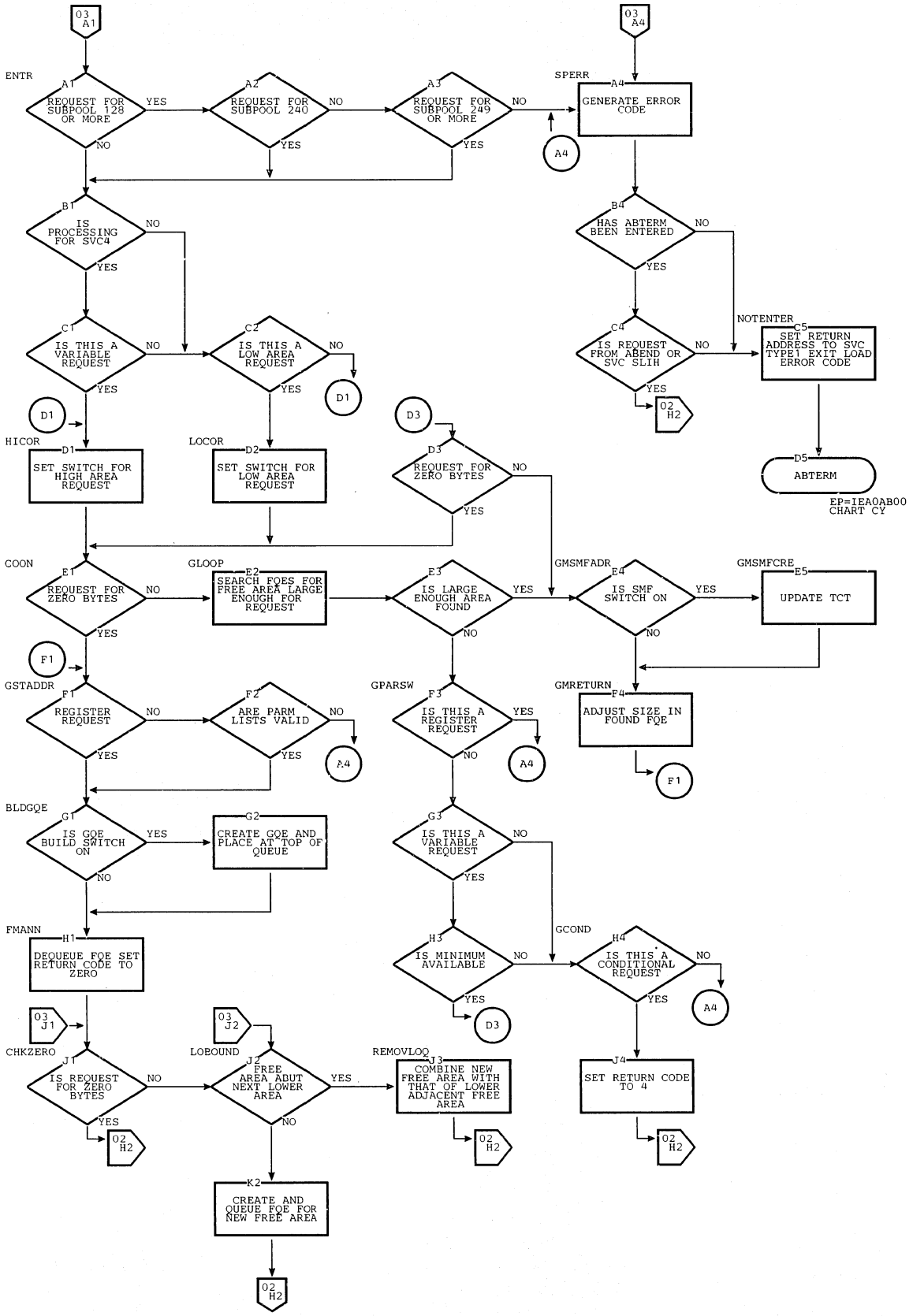


Chart GA. Communications Task Initialization

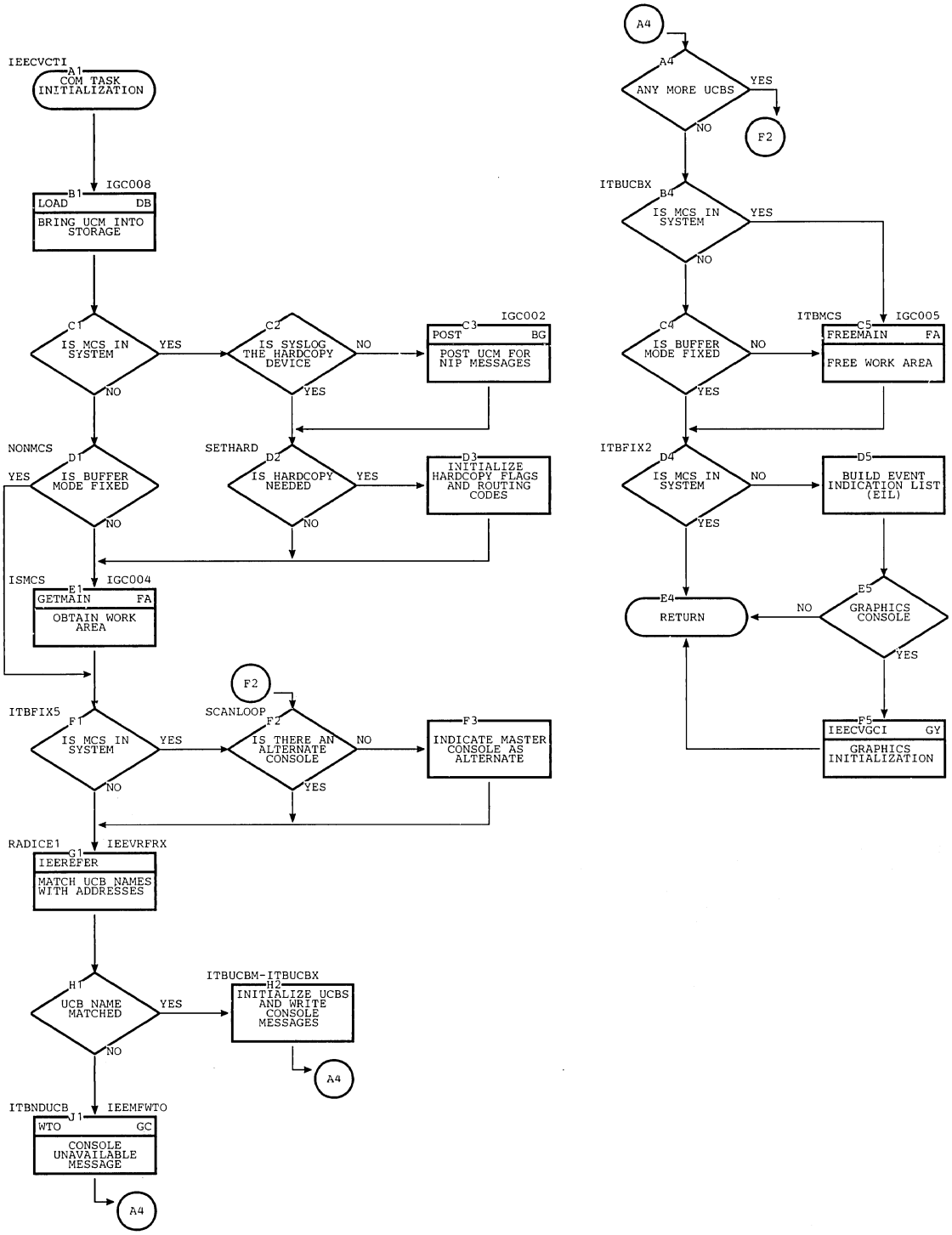


Chart GB. Communications Task External Interruption and Attention Handlers

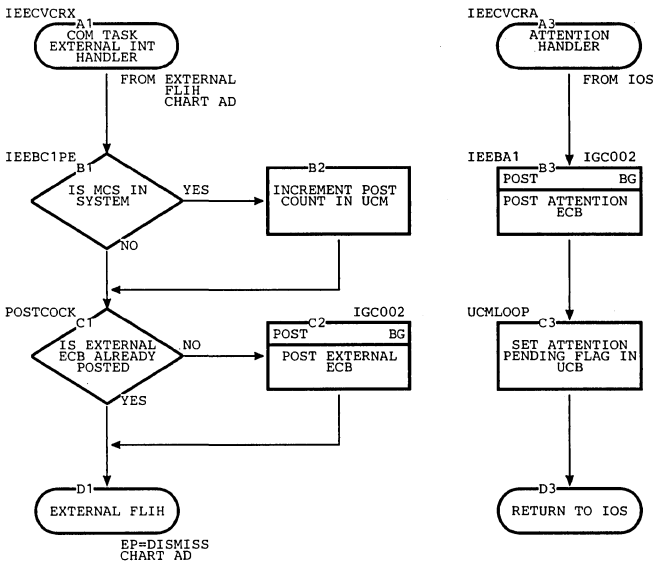


Chart GC. Communications Task Write-to-Operator without MCS

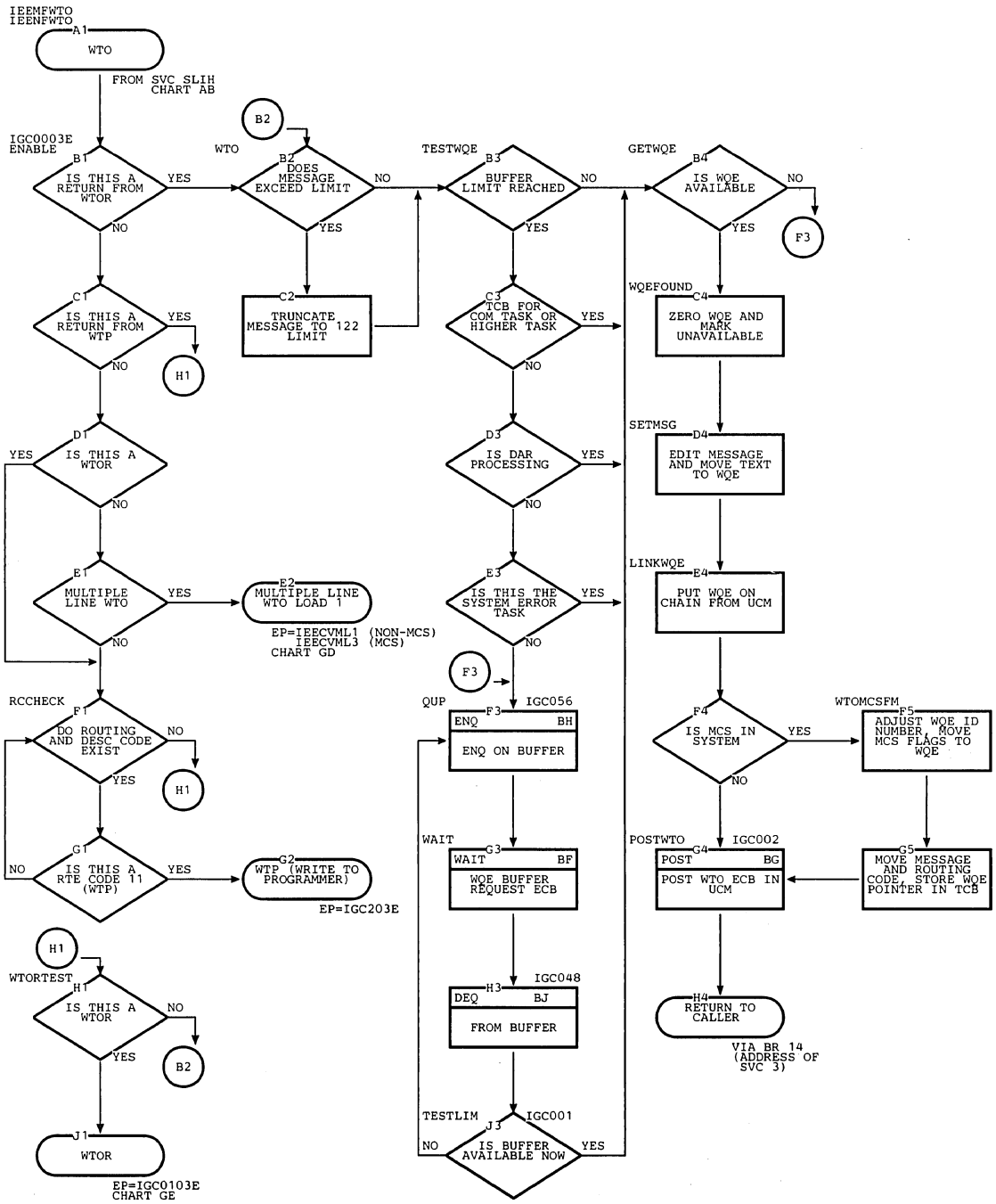


Chart GD. Multiple-Line Write-to-Operator (MLWTO) (Part 1 of 4)

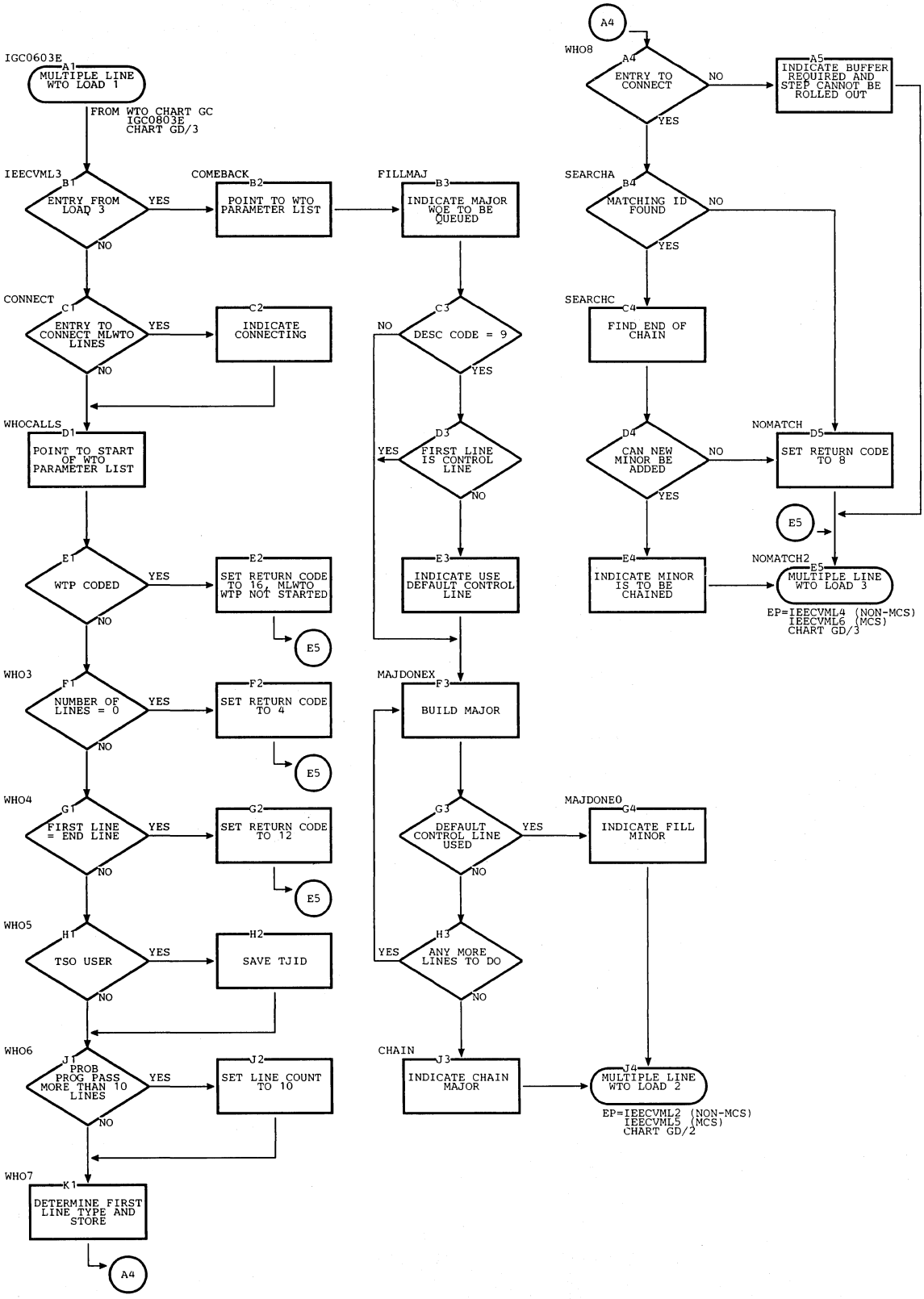


Chart GD. Multiple-Line Write-to-Operator (MLWTO) (Part 2 of 4)

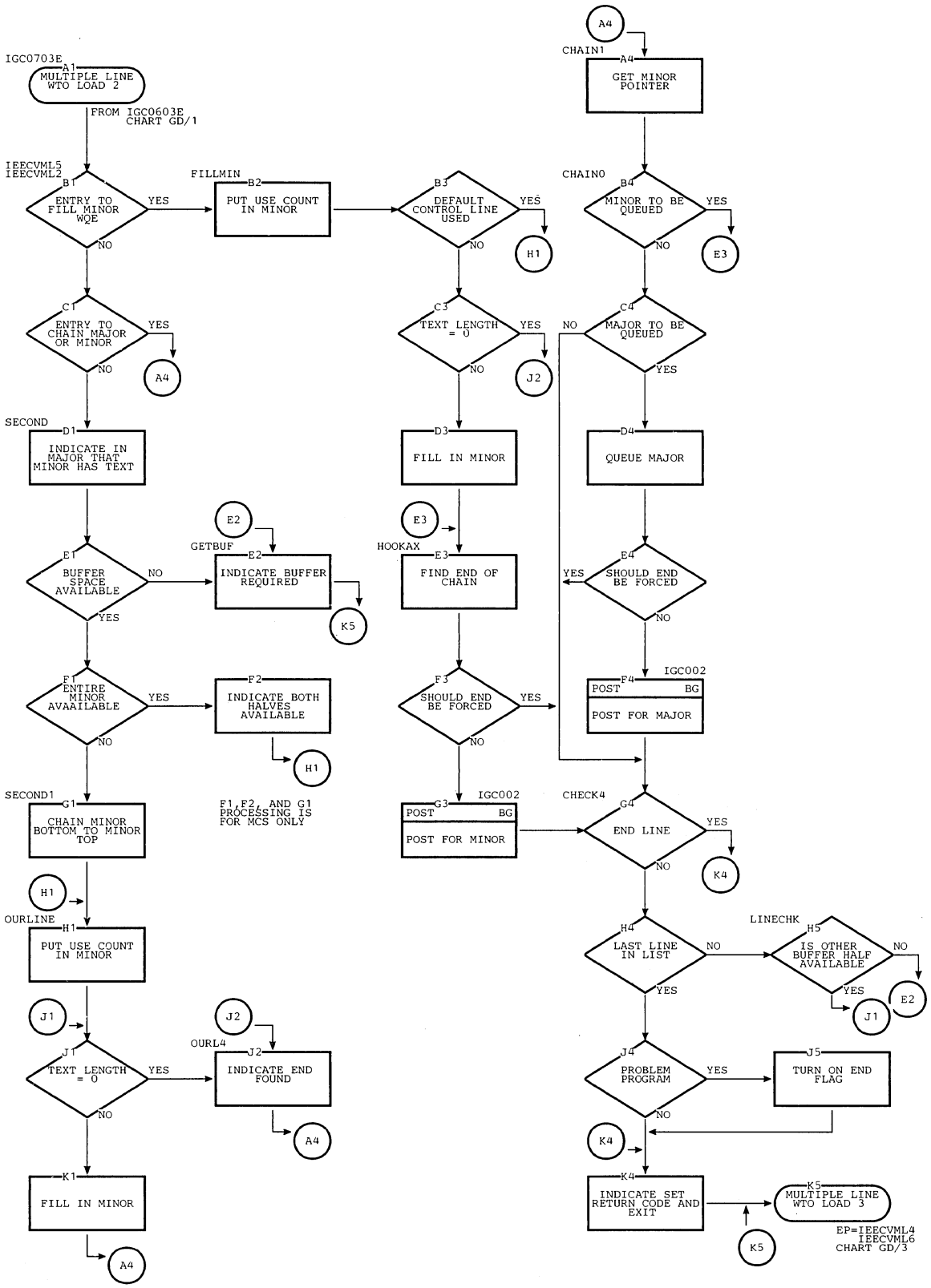


Chart GD. Multiple-Line Write-to-Operator (MLWTO) (Part 3 of 4)

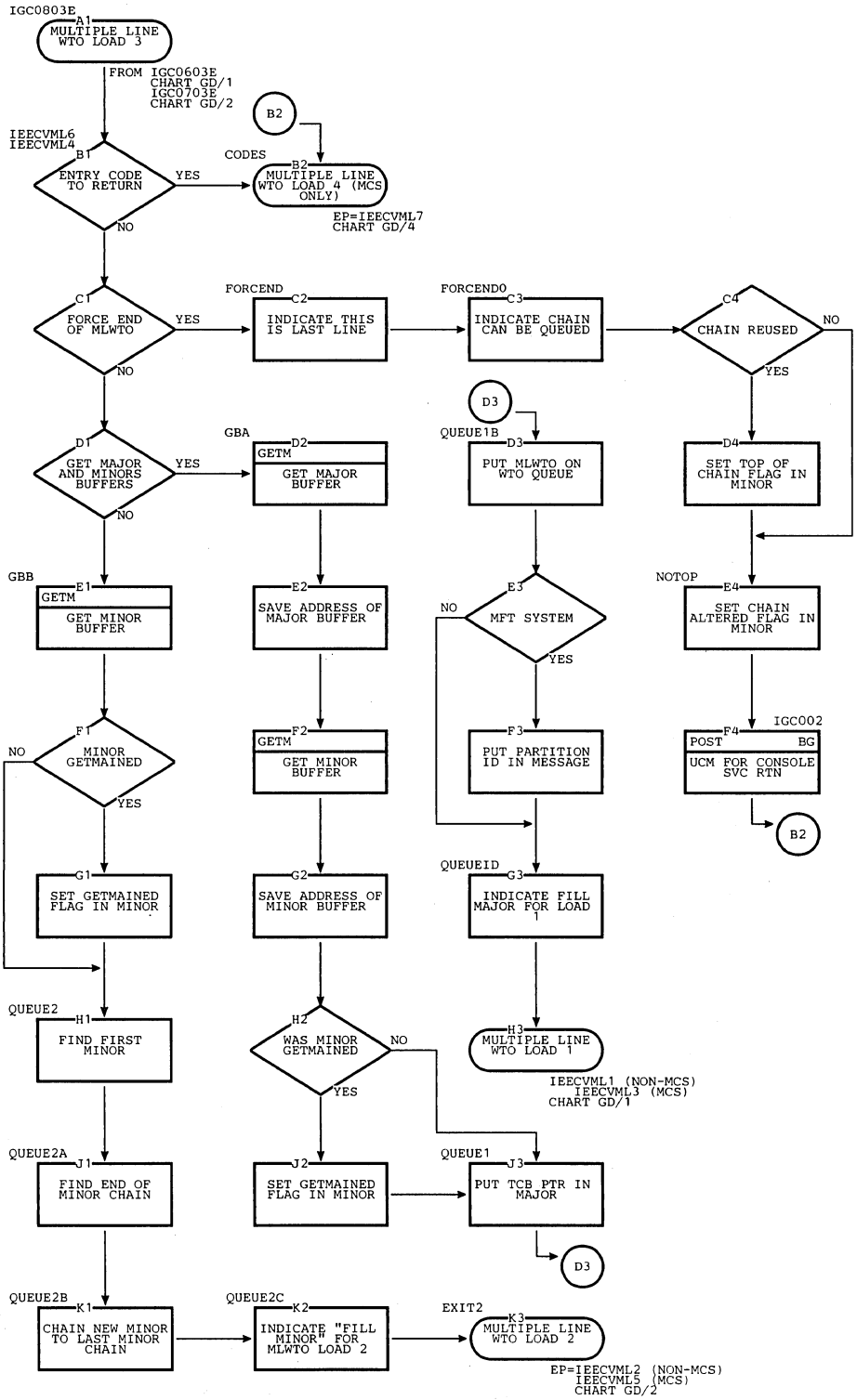


Chart GD. Multiple-Line Write-to-Operator (MLWTO) (Part 4 of 4)

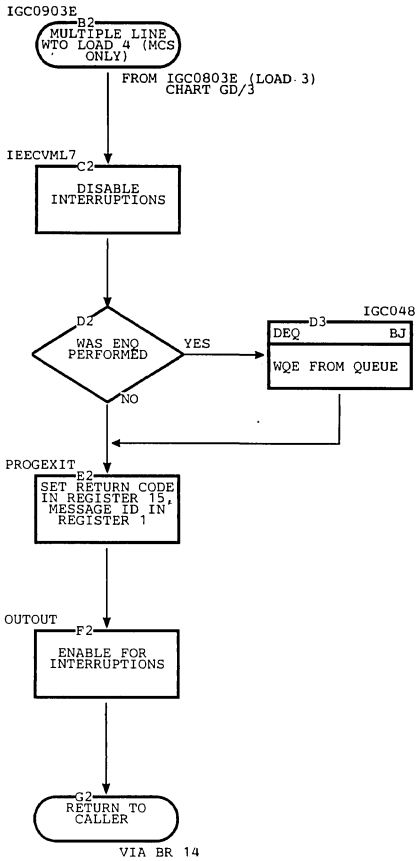


Chart GE. Communications Task Write-to-Operator with Reply

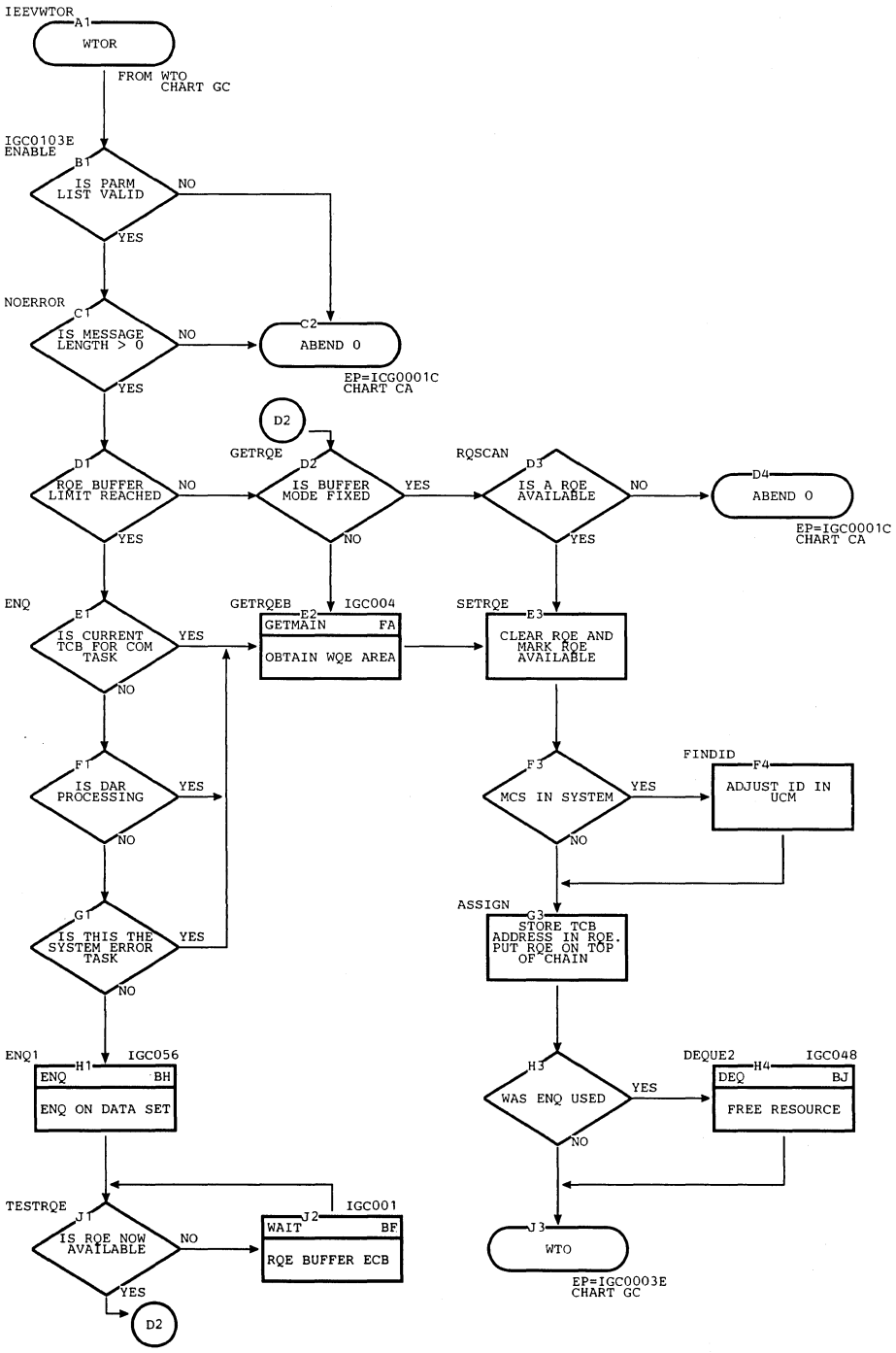


Chart GF. Communications Task Wait (non-MCS)

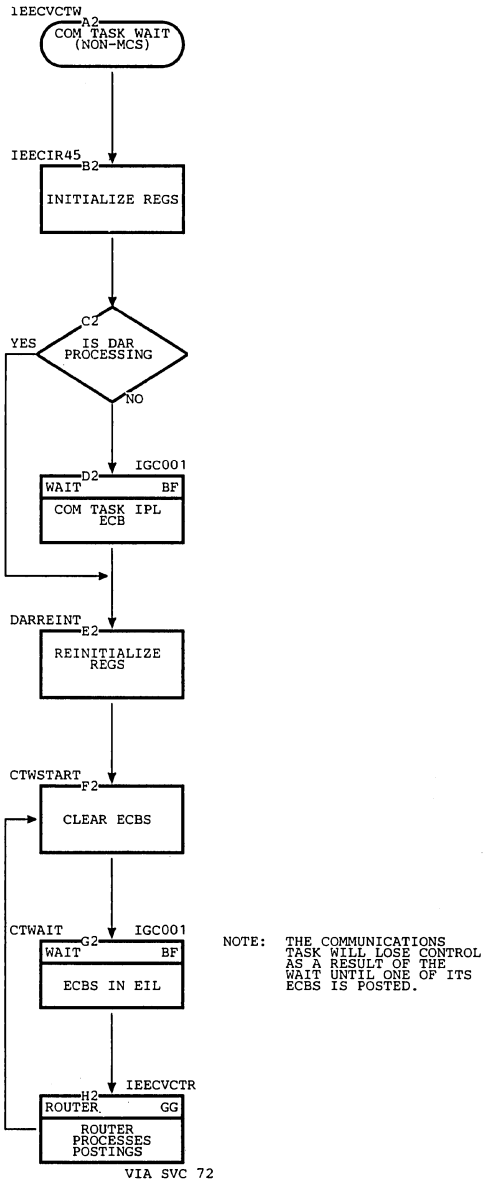
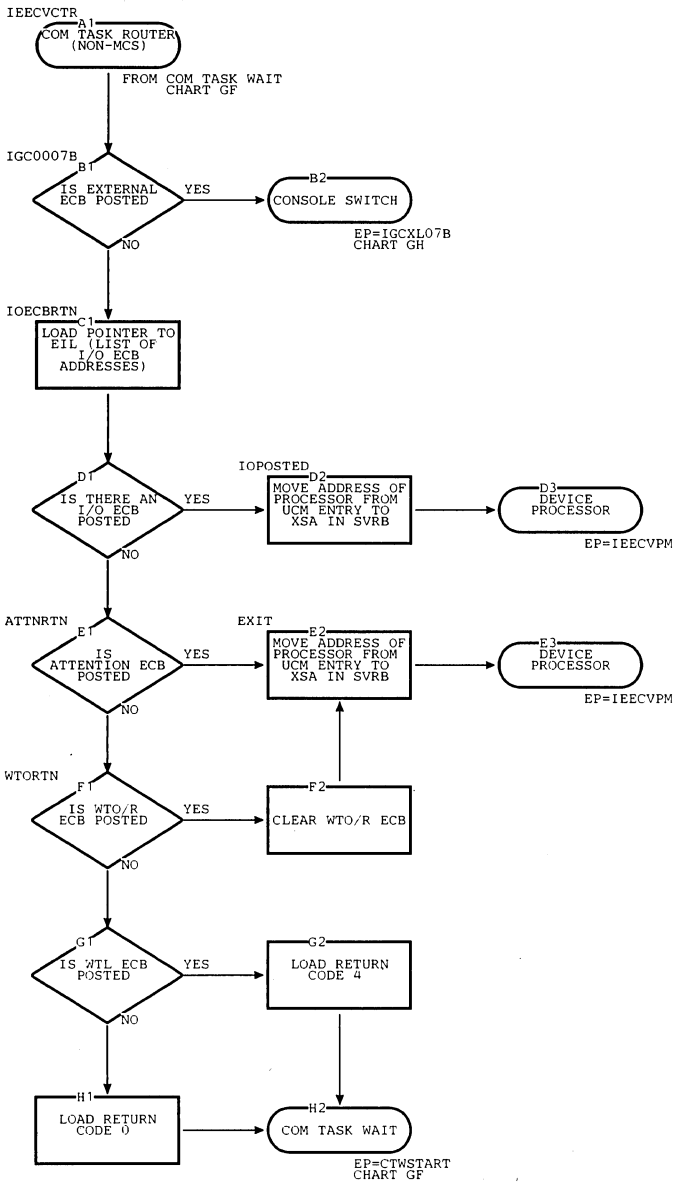


Chart GG. Communications Task Router (non-MCS)



NOTE: NON-MCS PROCESSOR FLOWCHARTS ARE GO AND GR
MCS PROCESSOR FLOWCHARTS ARE GP, GS, GT, AND GX.

Chart GH. Communications Task Console Switch (non-MCS)

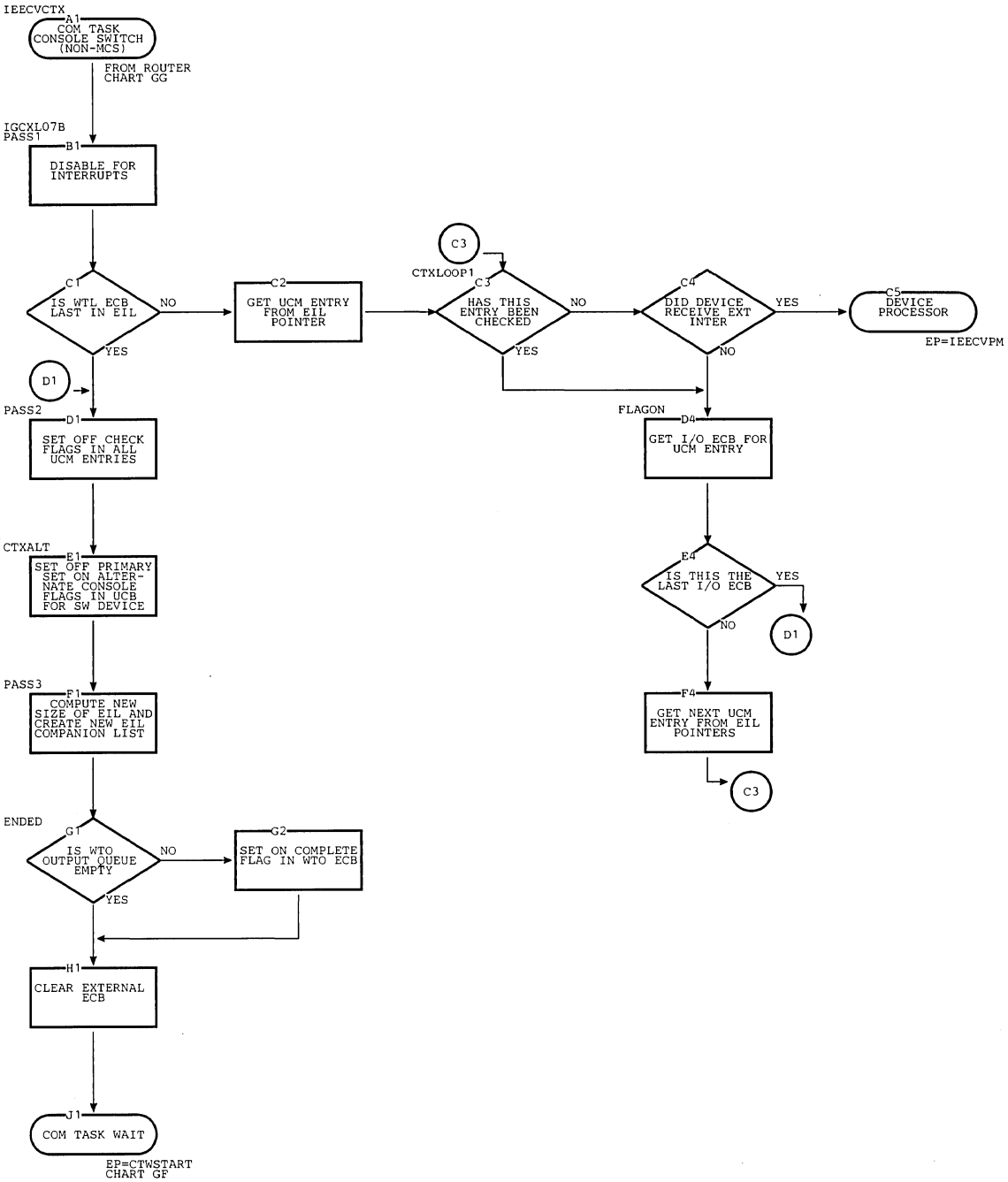


Chart GI. Communications Task Router (MCS)

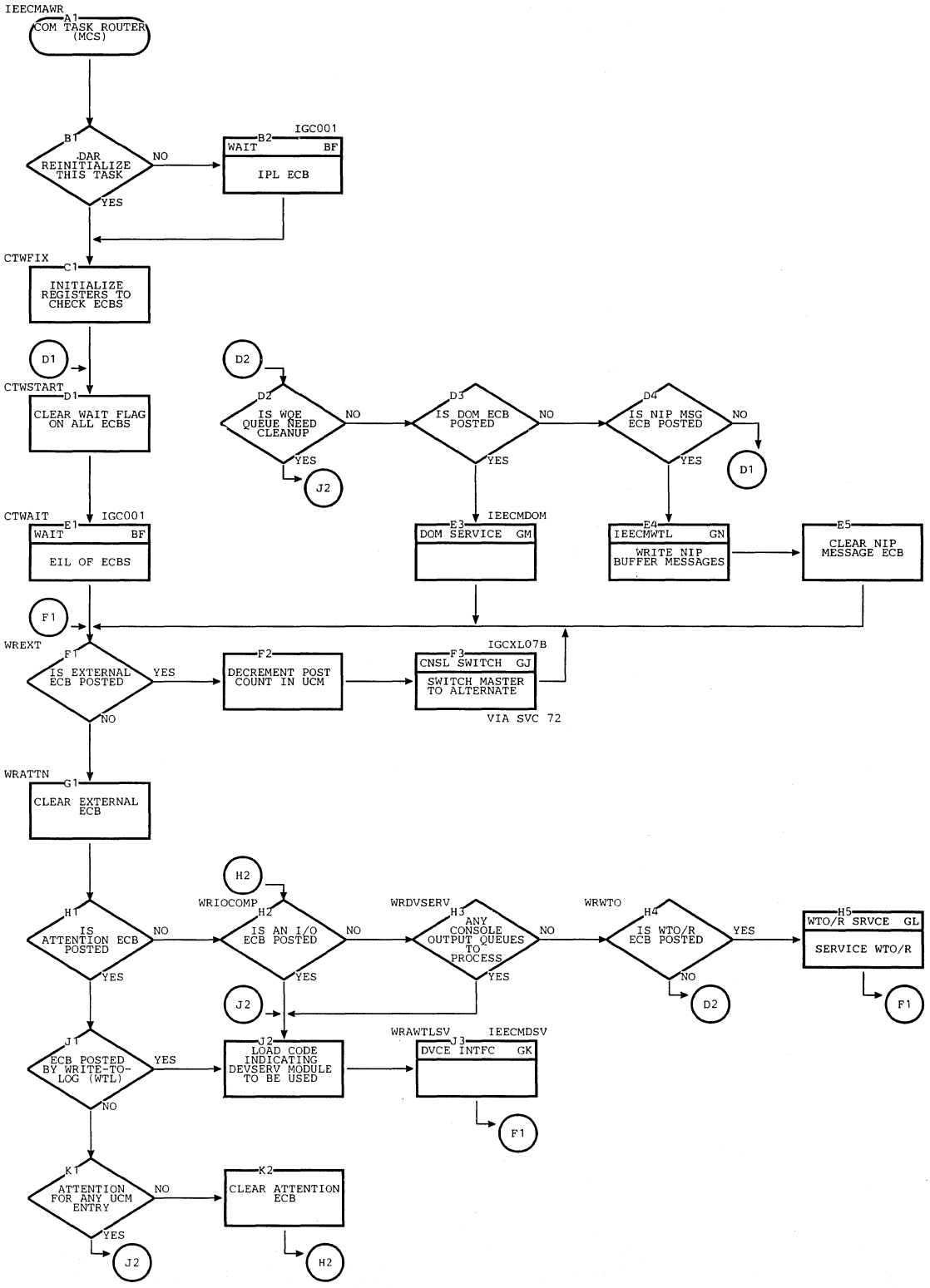


Chart GJ. Communications Task Console Switch Load 1 (MCS) (Part 1 of 4)

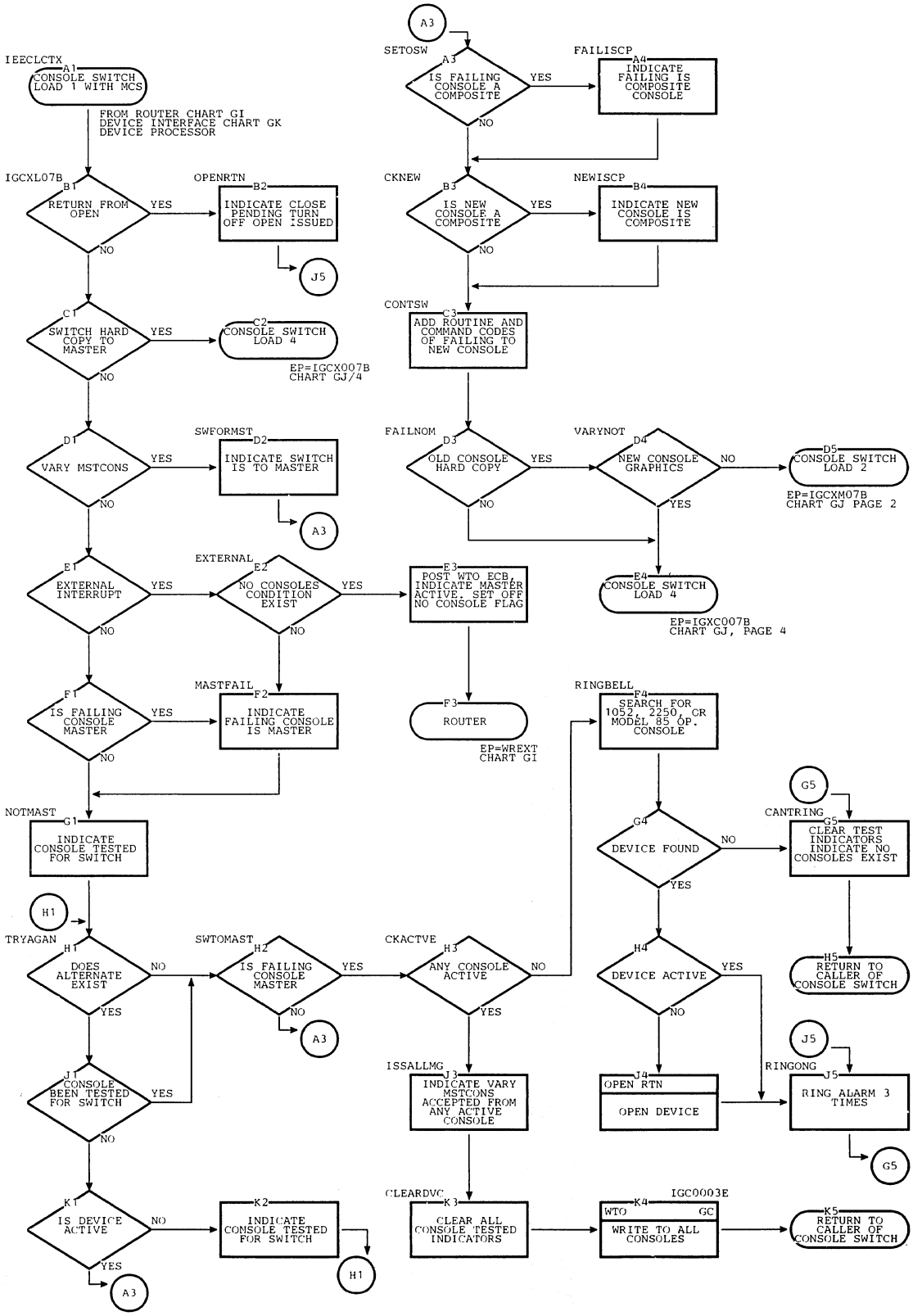


Chart GJ. Communications Task Console Switch Loads 2 and 3 (MCS) (Part 2 of 4)

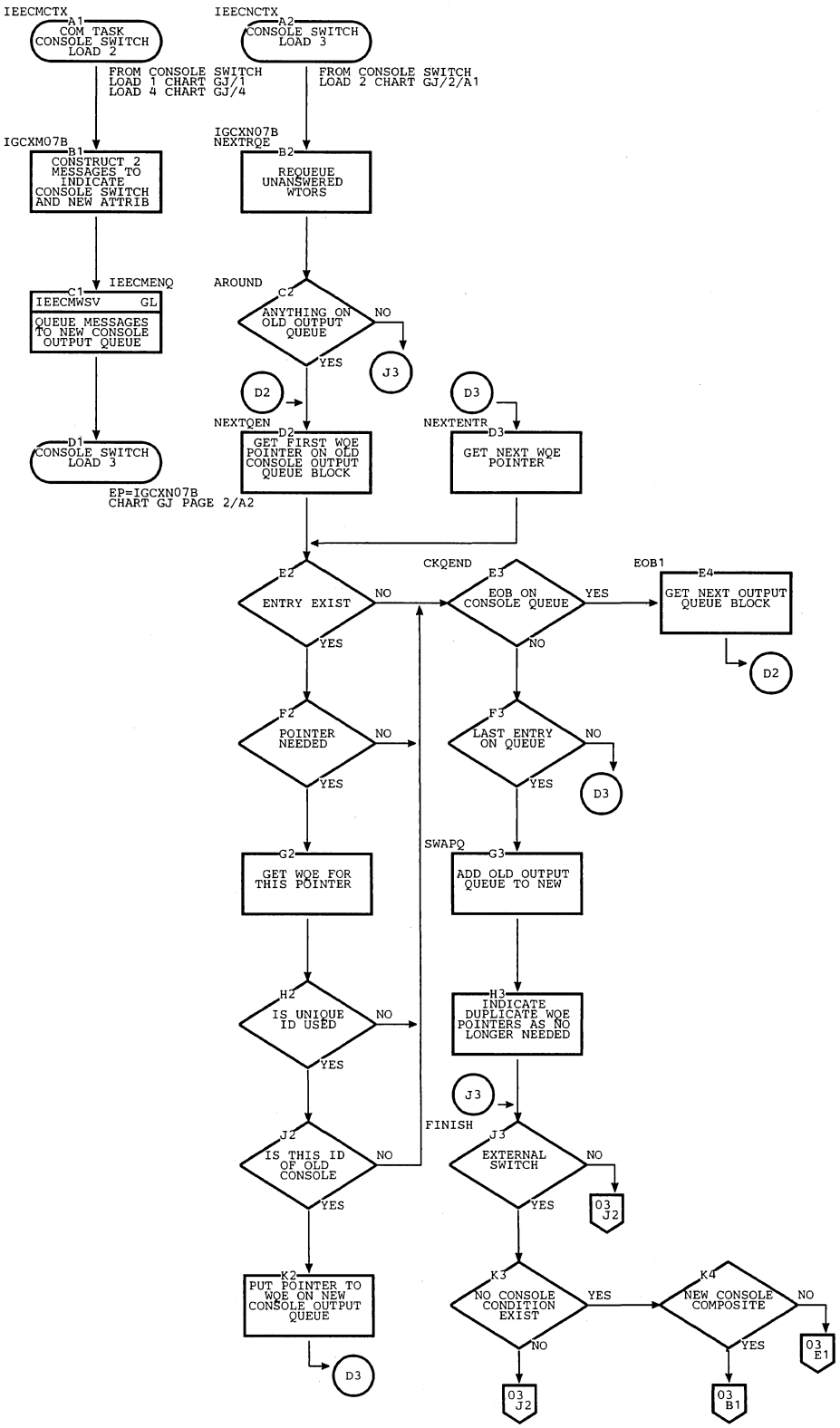


Chart GJ. Communications Task Console Switch Load 3 (MCS) (Part 3 of 4)

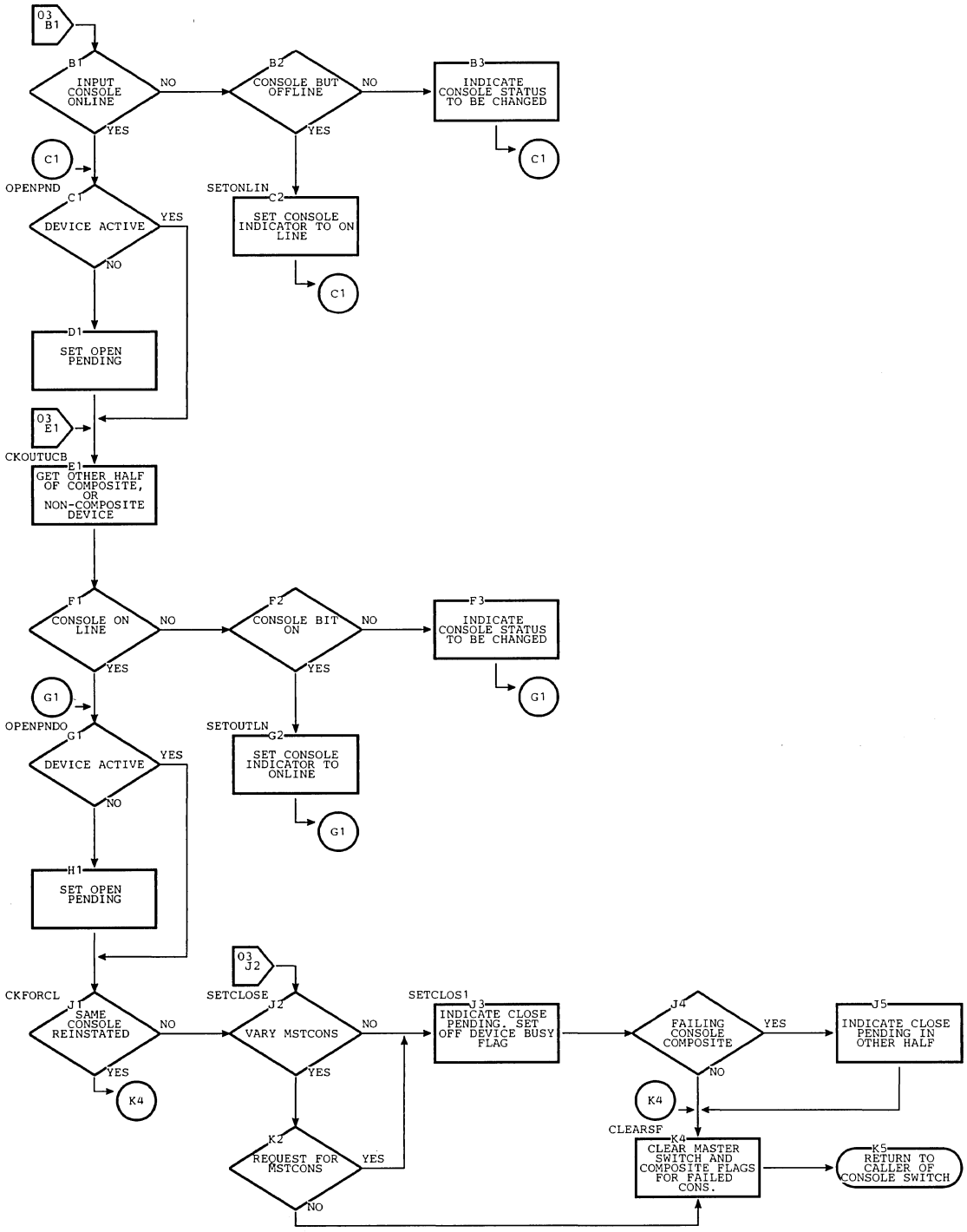


Chart GJ. Communications Task Console Switch Load 4 (MCS) (Part 4 of 4)

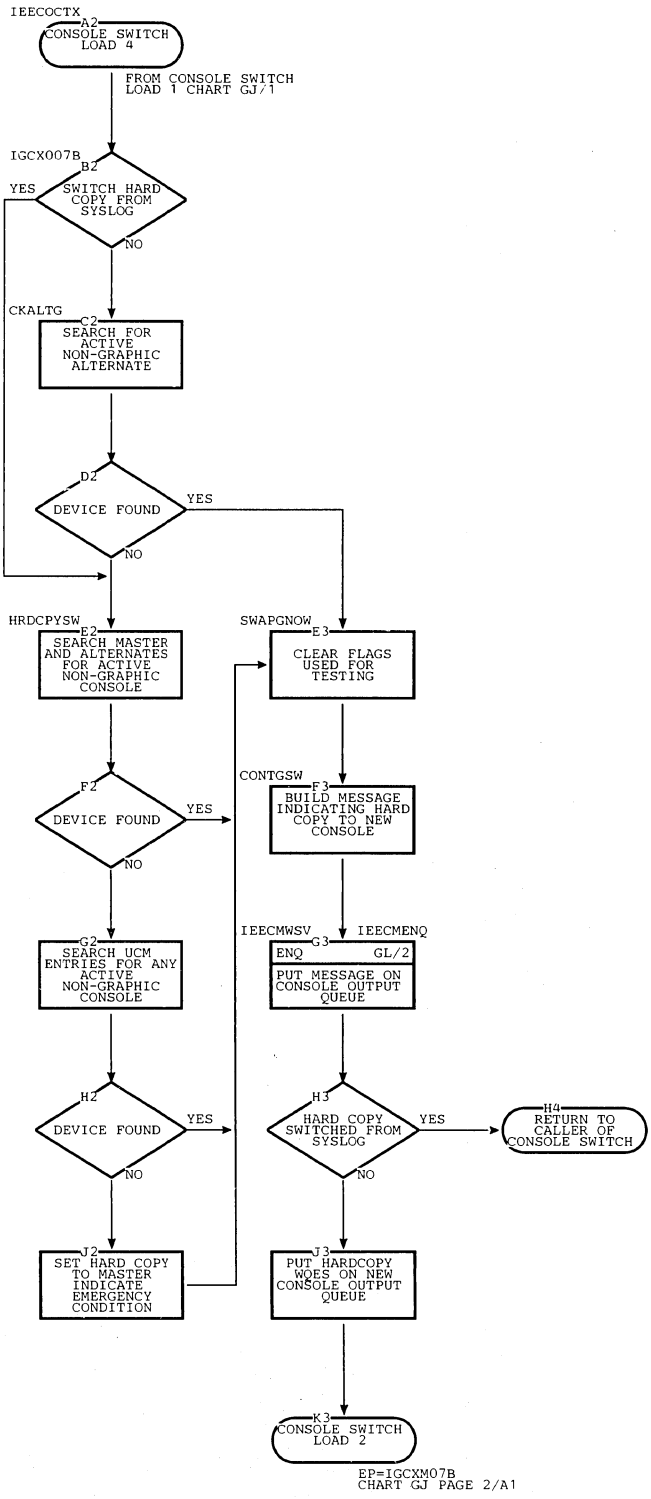


Chart GK. Communications Task Device Interface (MCS) (Part 1 of 3)

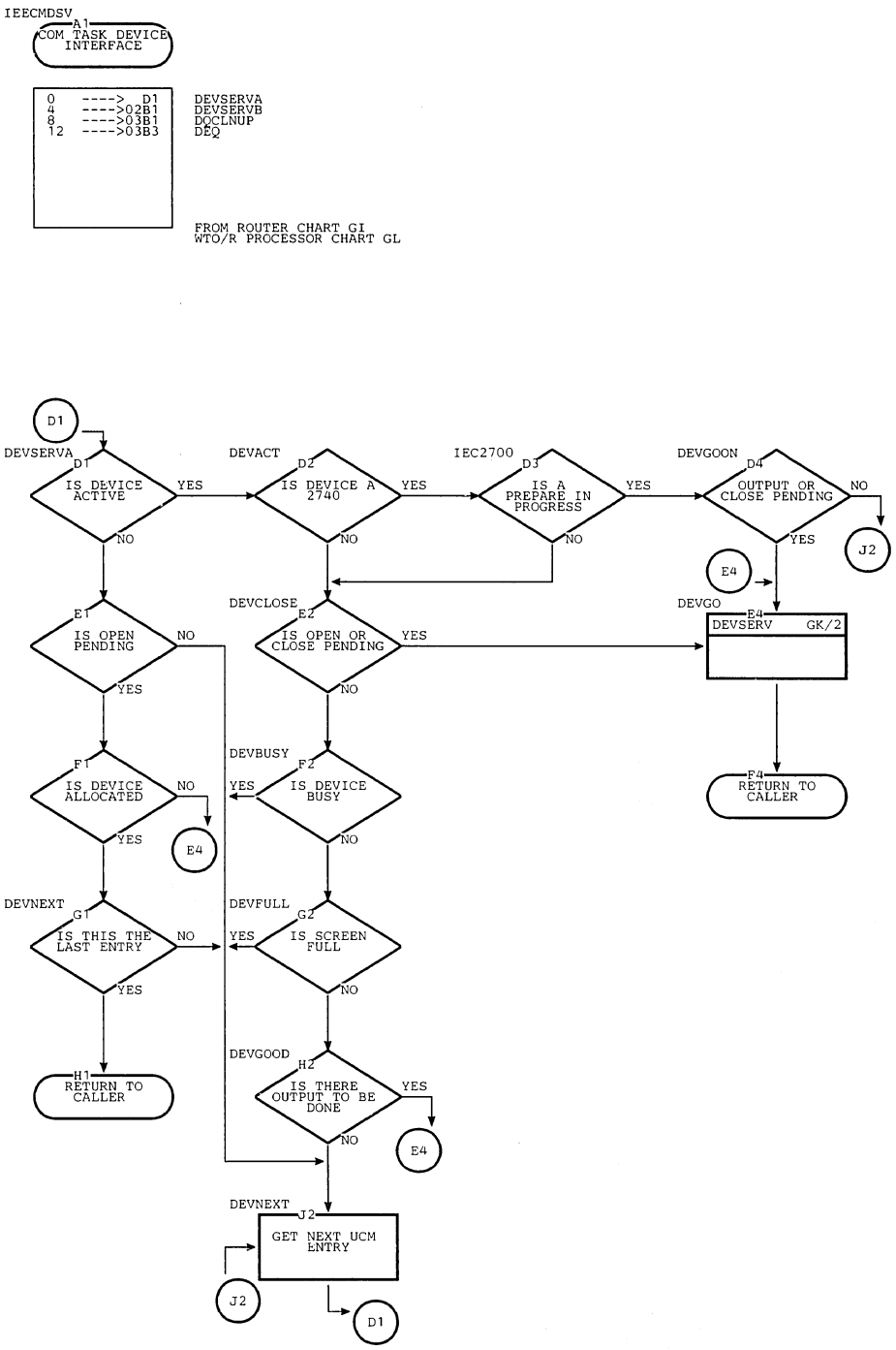


Chart GK. Communications Task Device Interface (MCS) (Part 2 of 3)

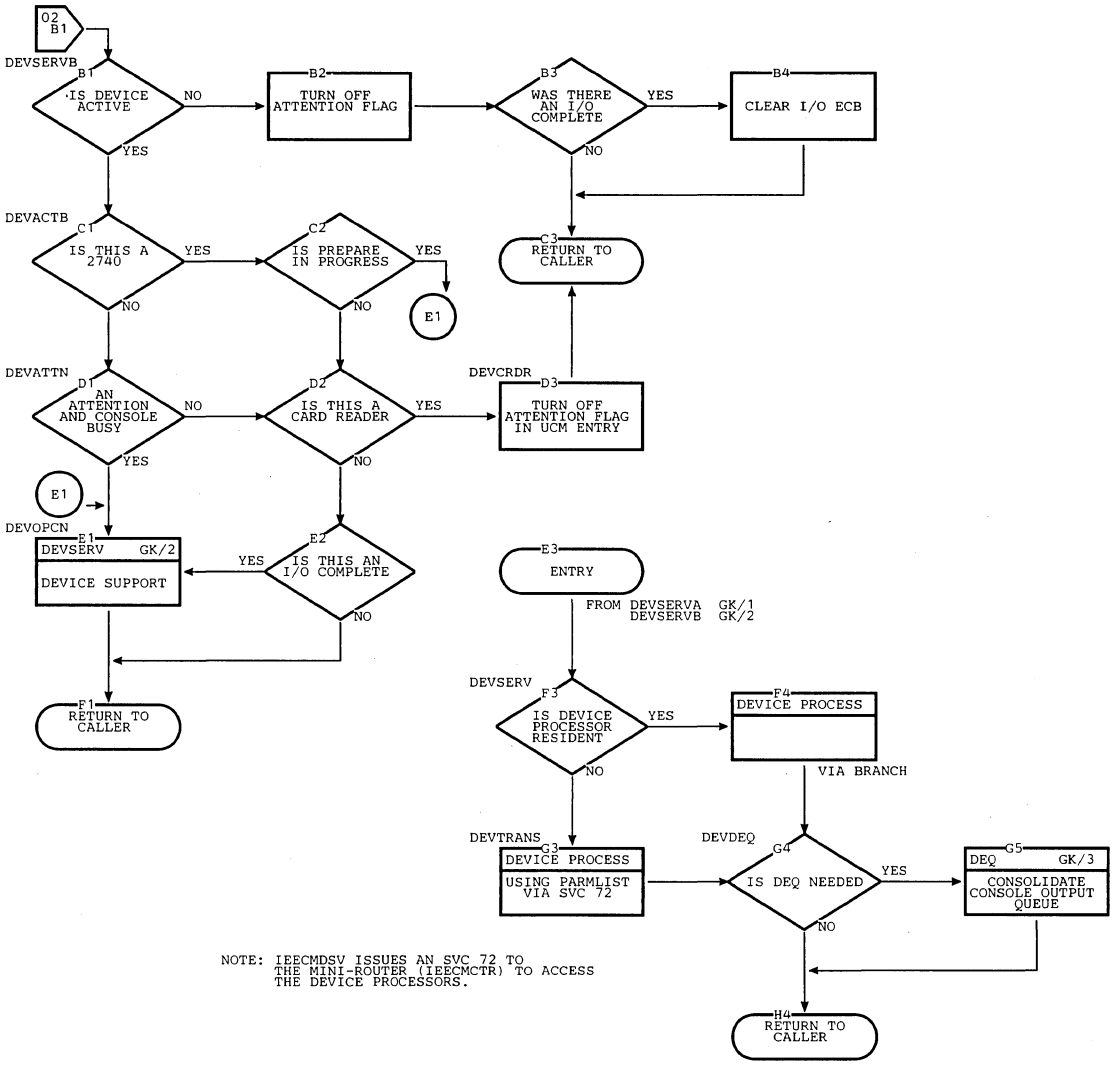


Chart GK. Communications Task Device Interface (MCS) (Part 3 of 3)

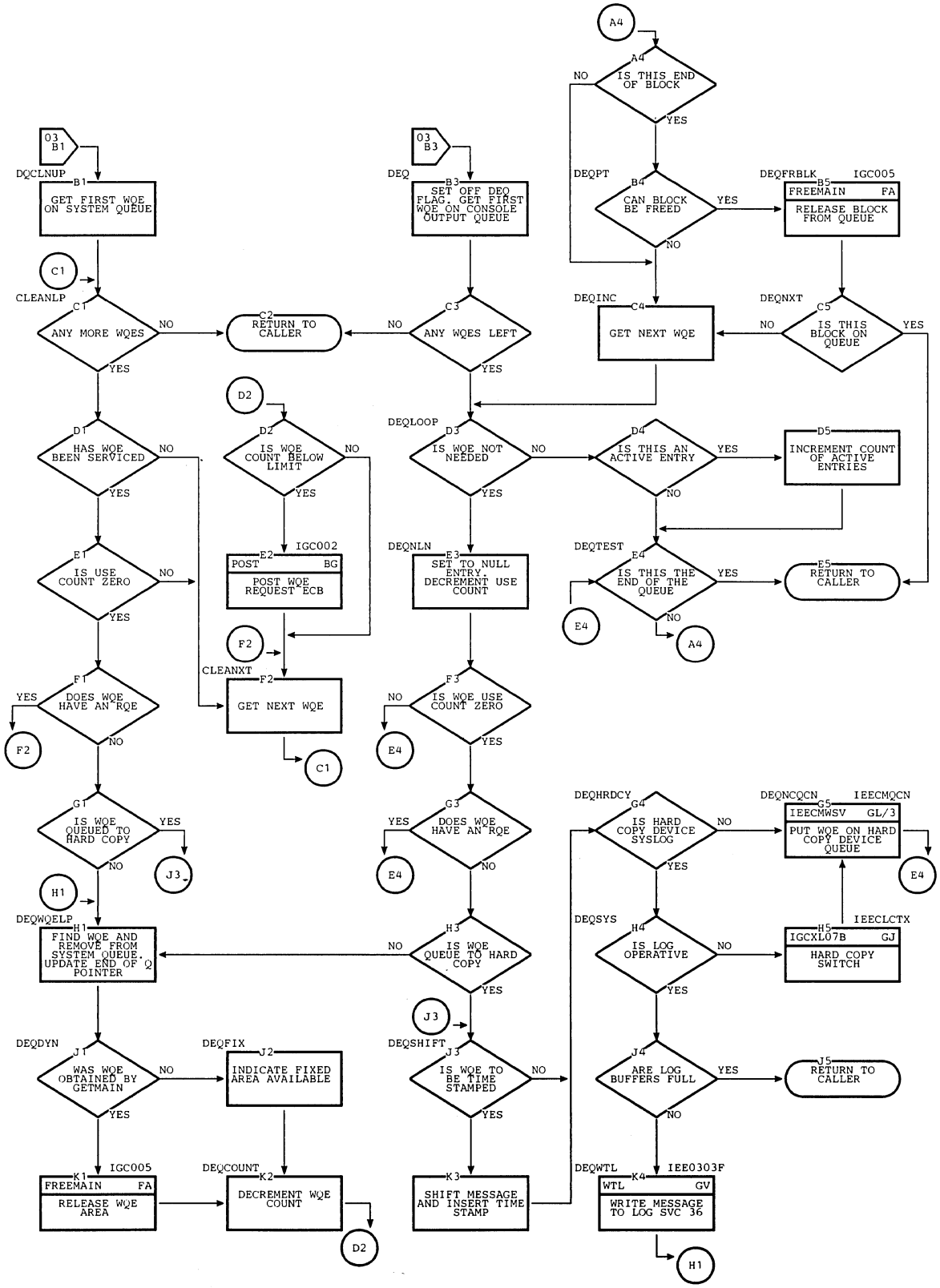


Chart GL. Communications Task WTO/R Processor (MCS) (Part 1 of 3)

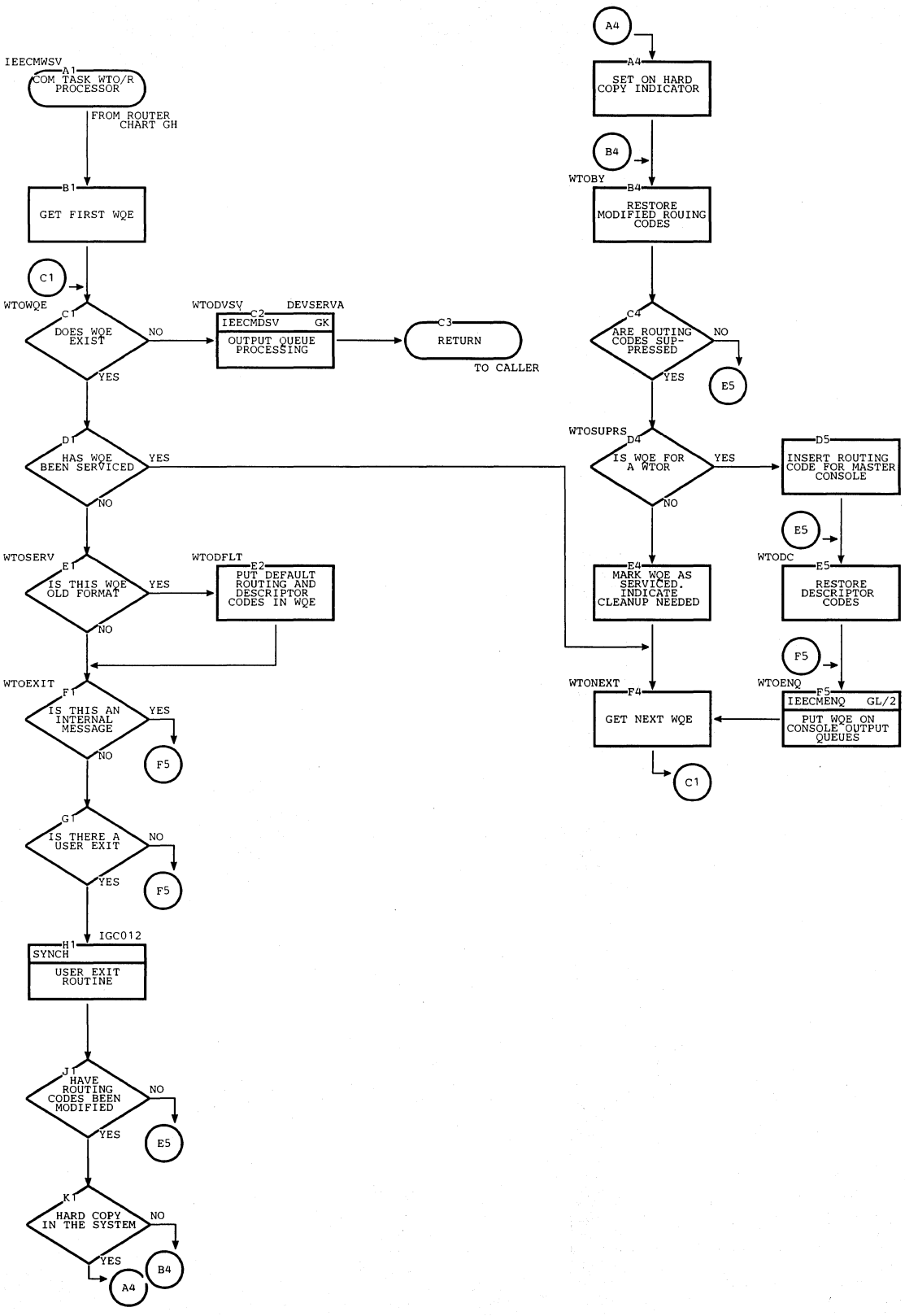


Chart GL. Communications Task WTO/R Processor (MCS) (Part 2 of 3)

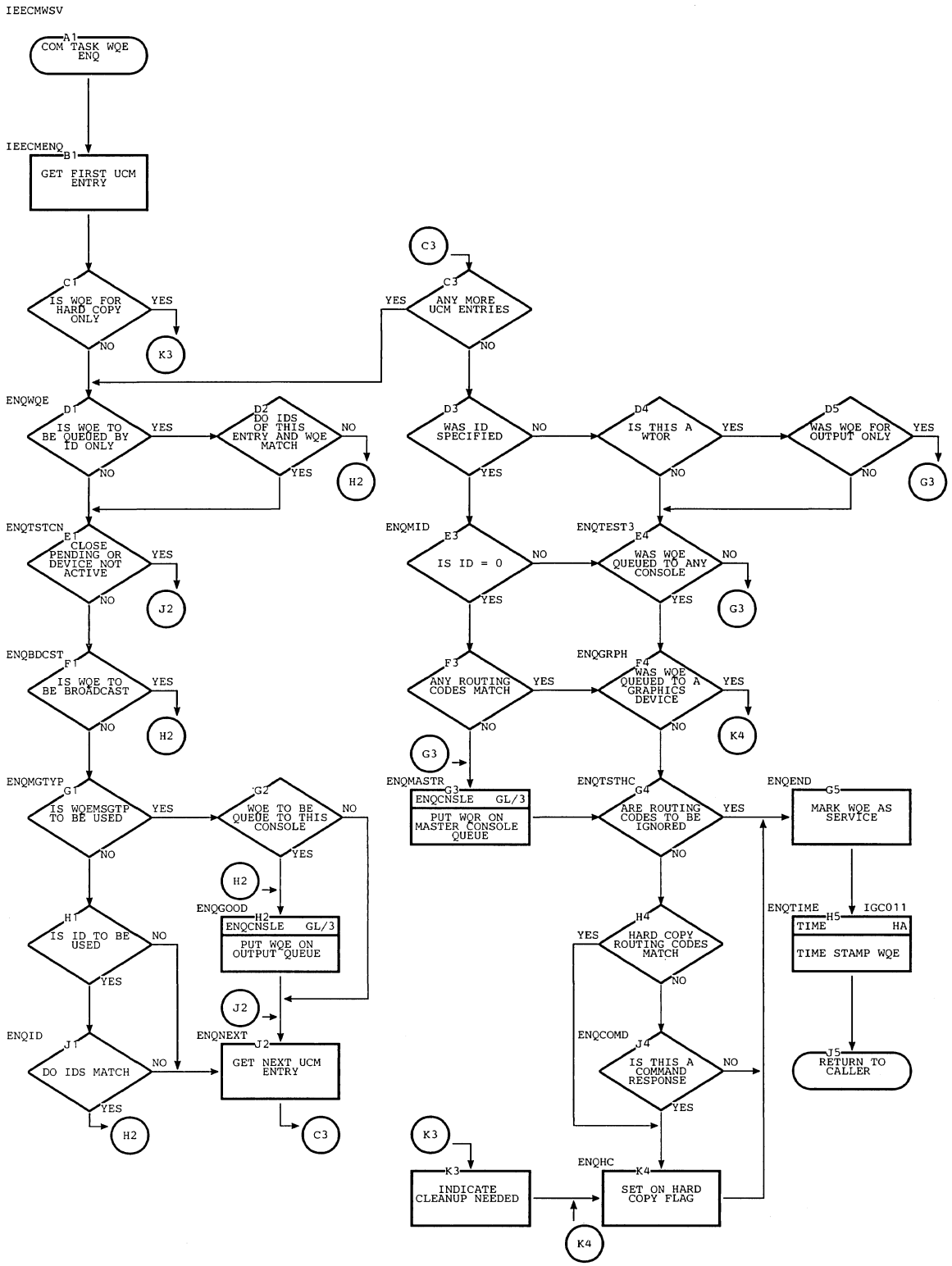


Chart GL. Communications Task WTO/R Processor (MCS) (Part 3 of 3)

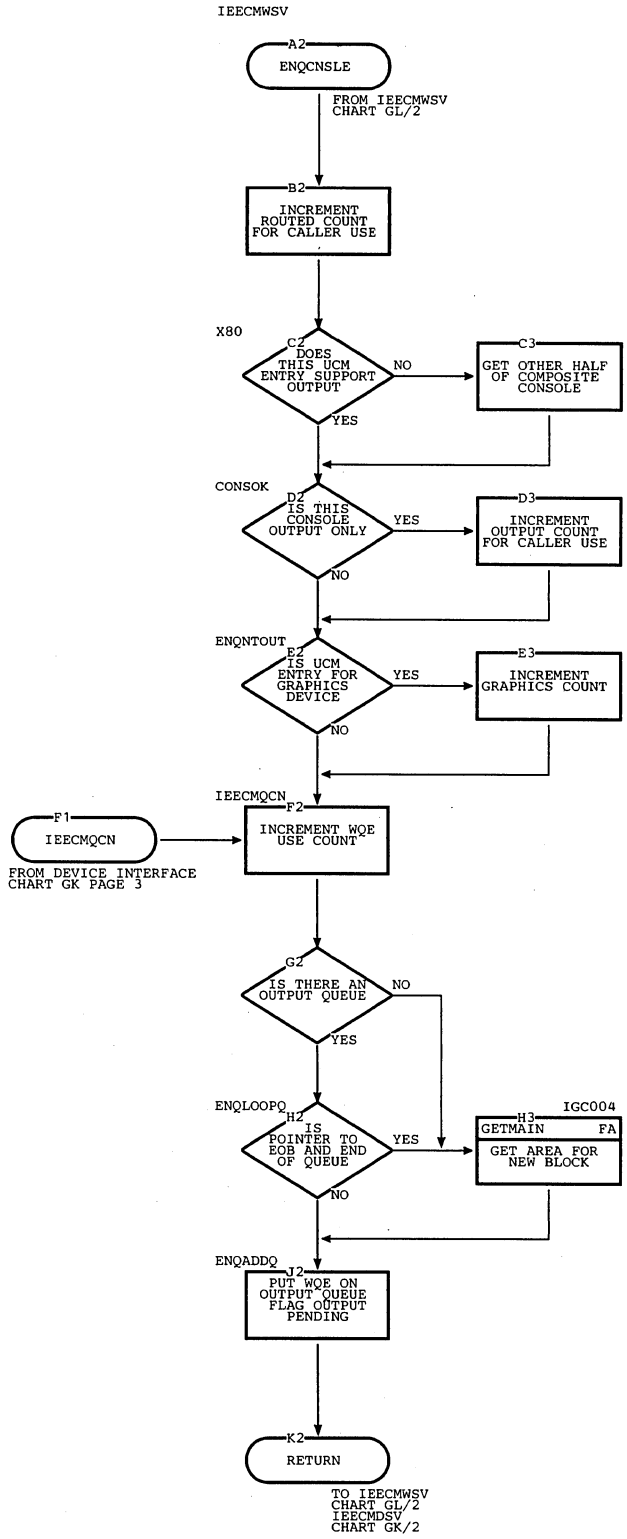


Chart GM. Communications Task Delete Operator Message (MCS)

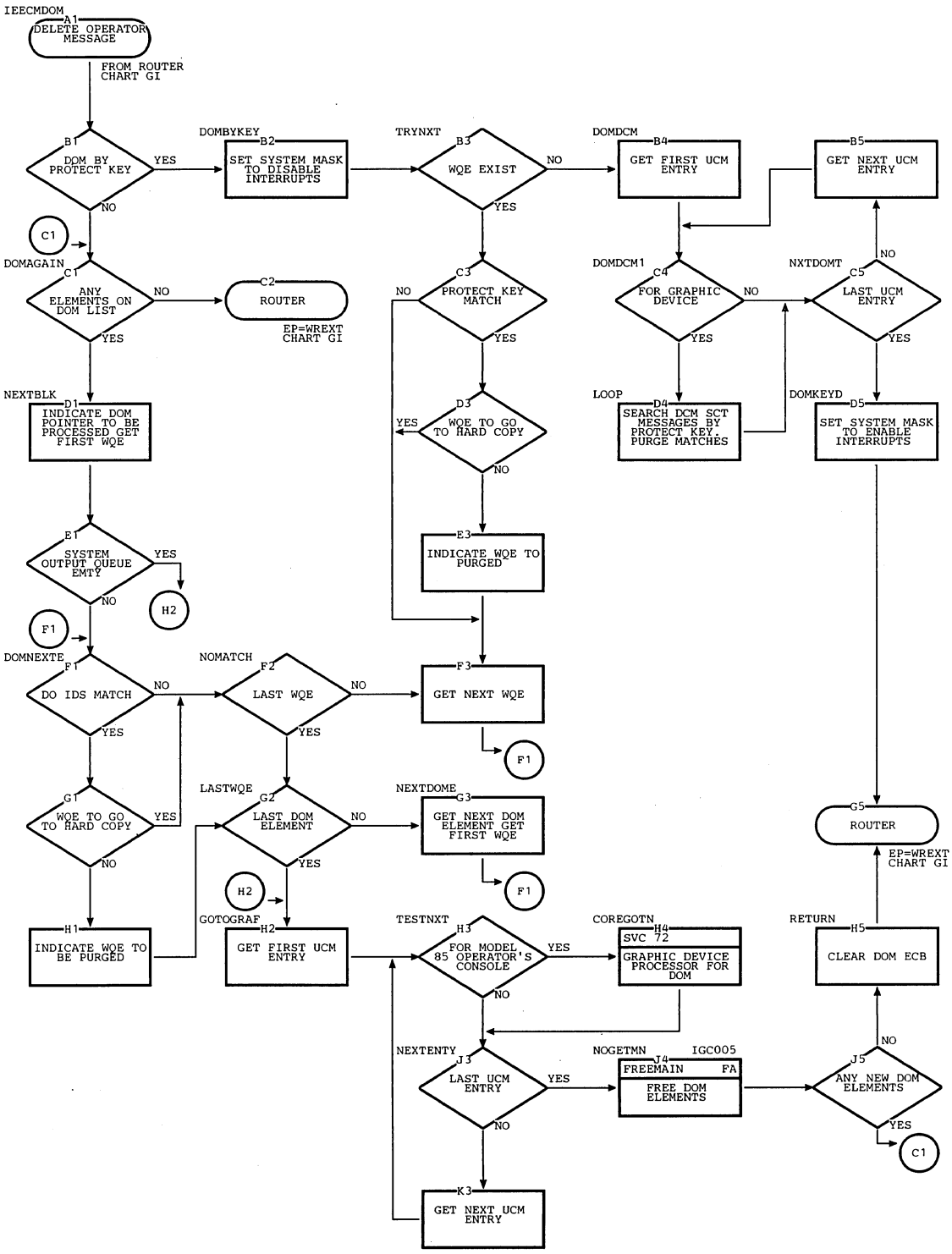


Chart GN. Communications Task NIP Message Buffer Writer

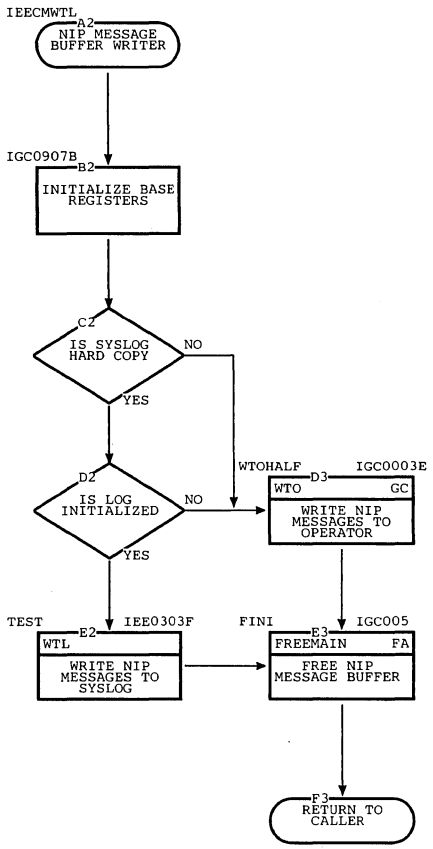


Chart GO. Communications Task 1052 Processor (non-MCS) (Part 1 of 2)

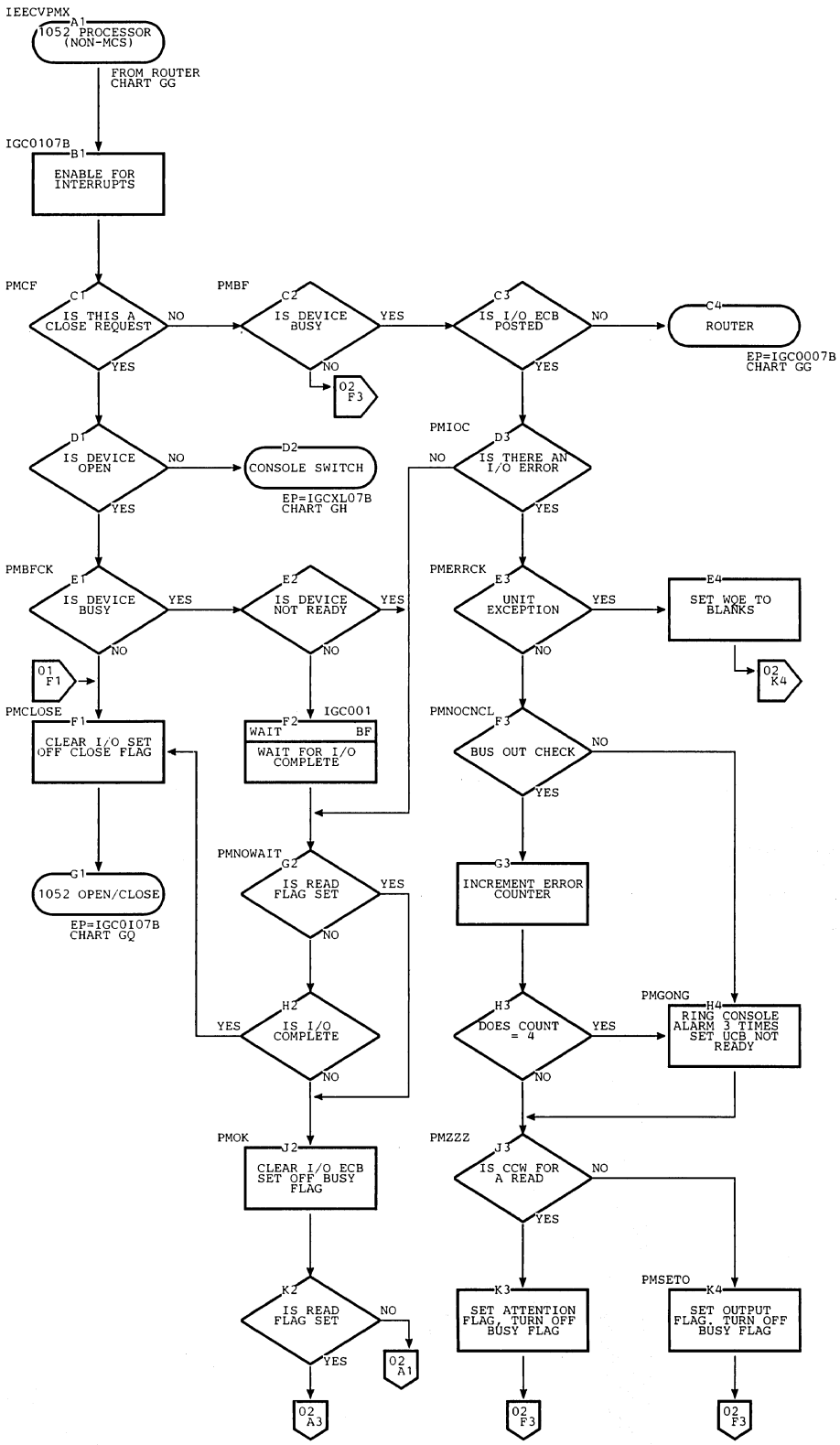


Chart GO. Communications Task 1052 Processor (non-MCS) (Part 2 of 2)

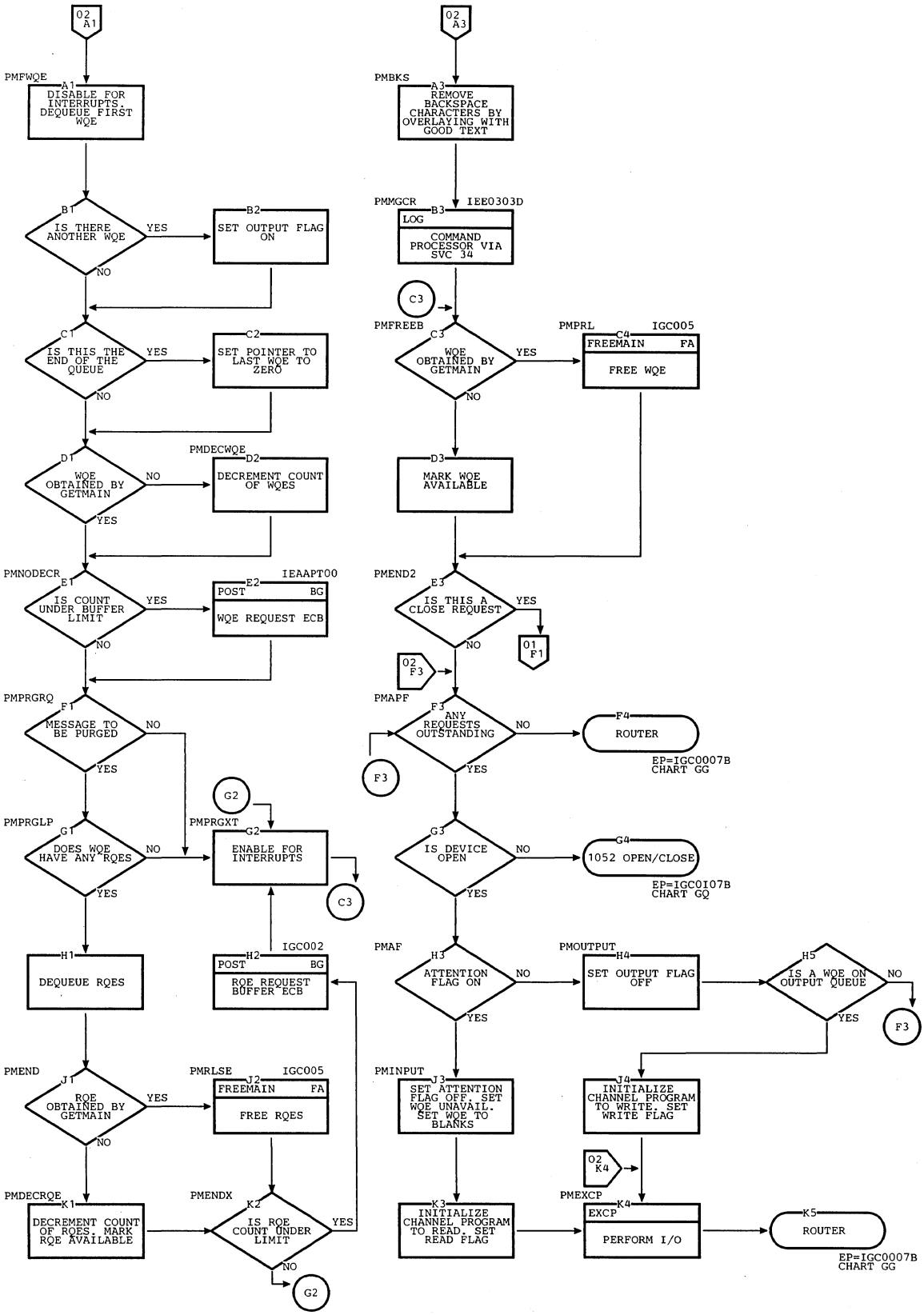


Chart GP. Communications Task 1052 Processor Load 1 (MCS) (Part 1 of 2)

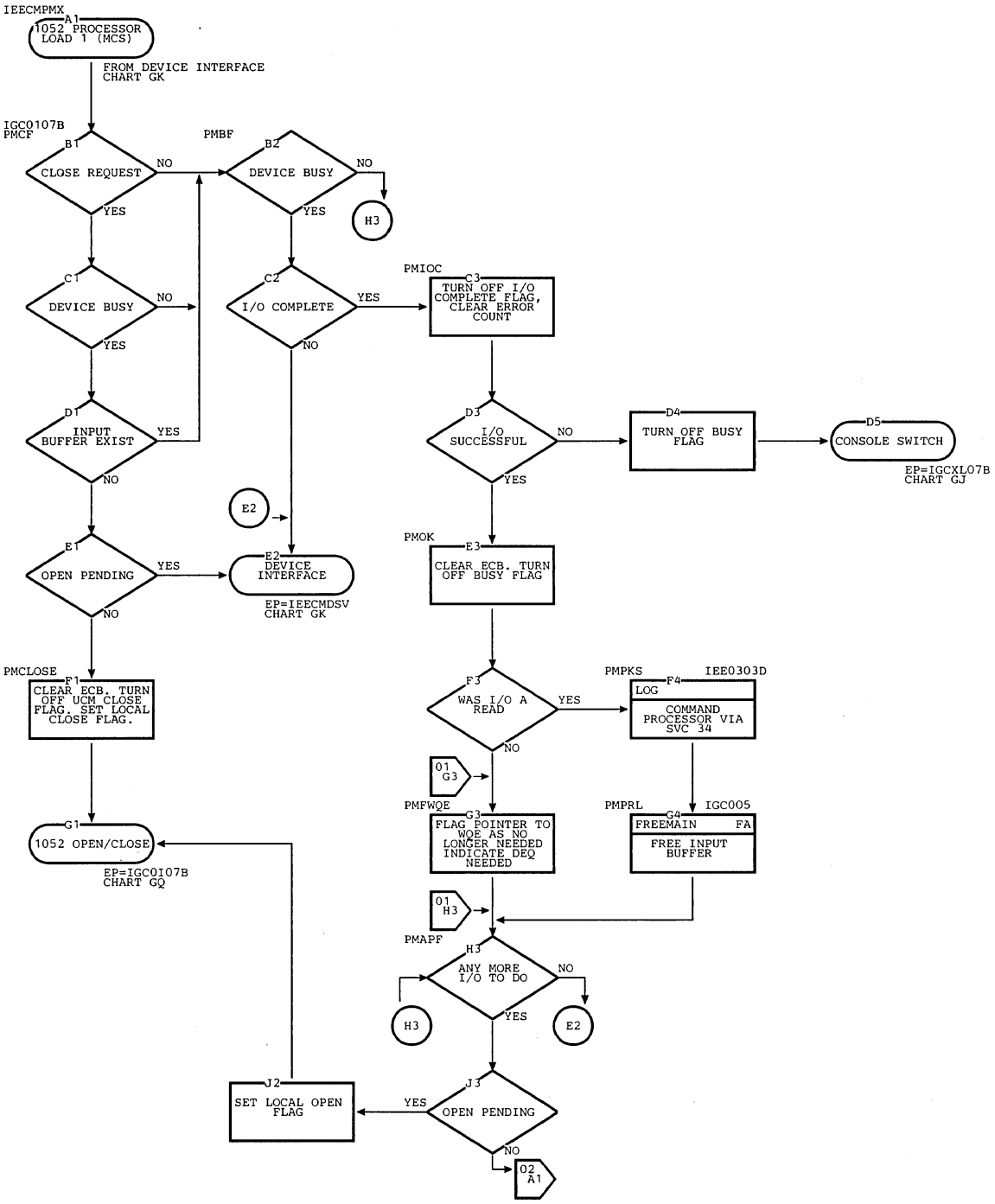


Chart GP. Communications Task 1052 Processor Load 1 (MCS) (Part 2 of 2)

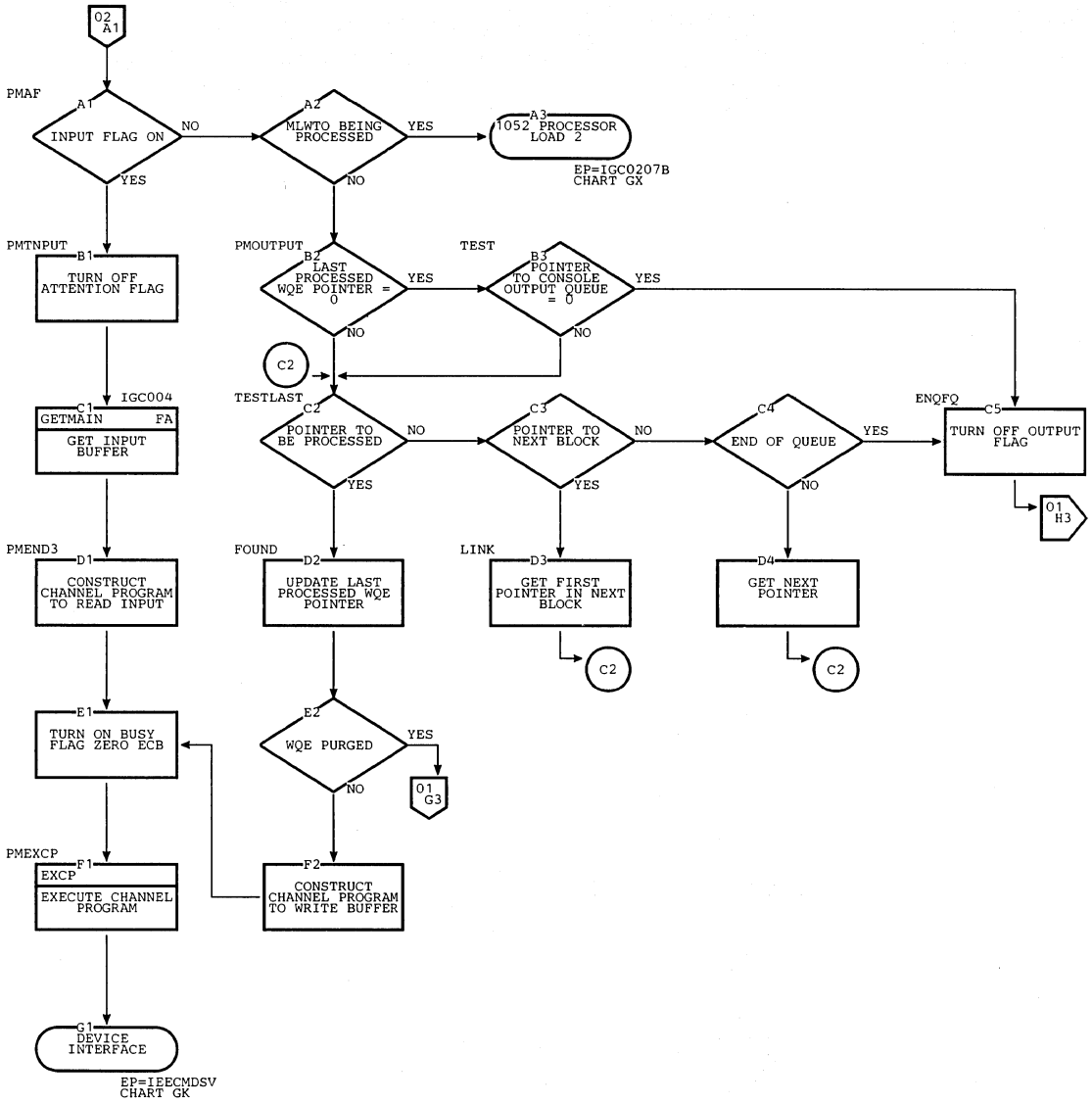


Chart GQ. Communications Task 1052 Open/Close

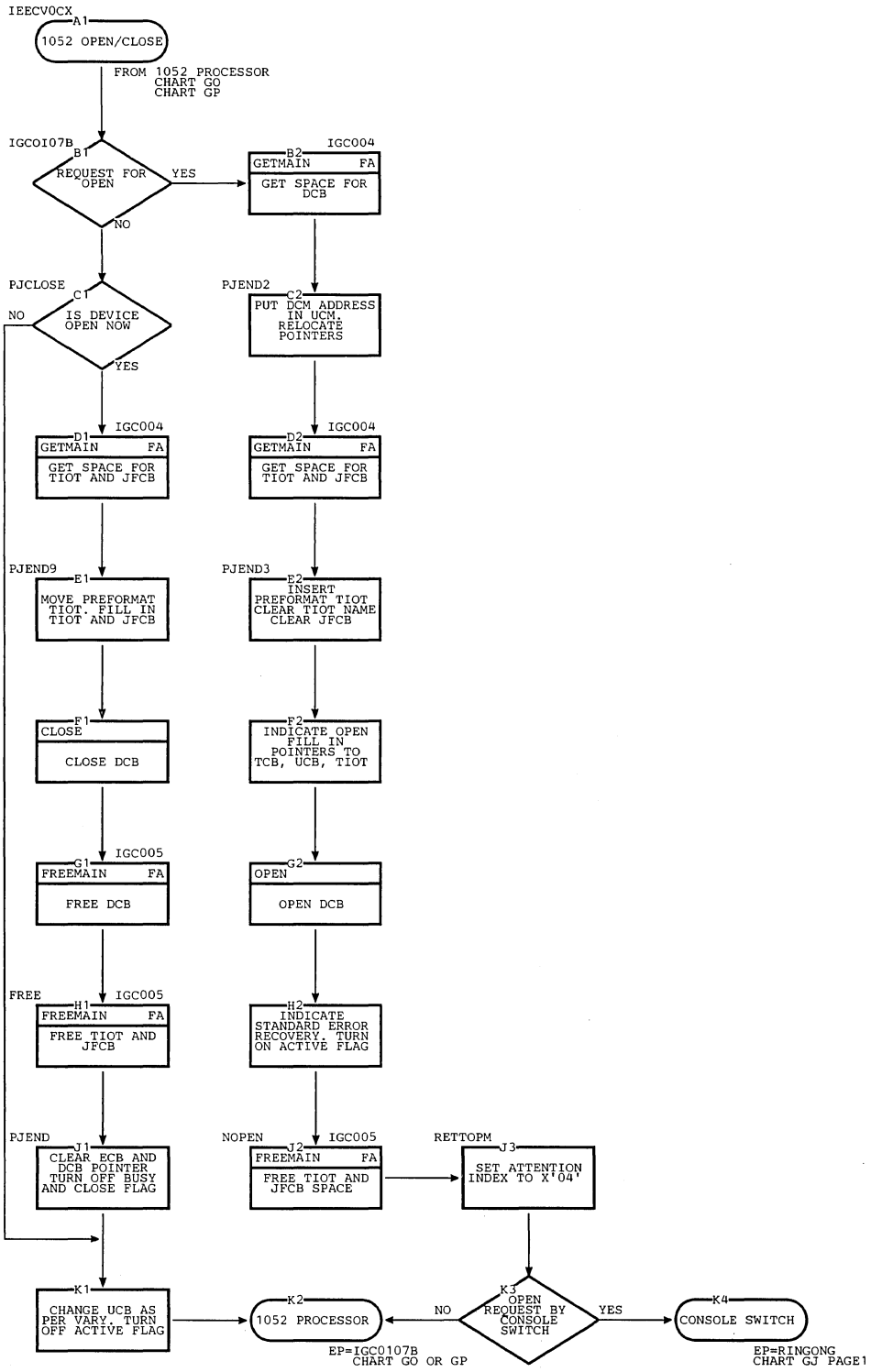


Chart GR. Communications Task Card Reader Processor (non-MCS)

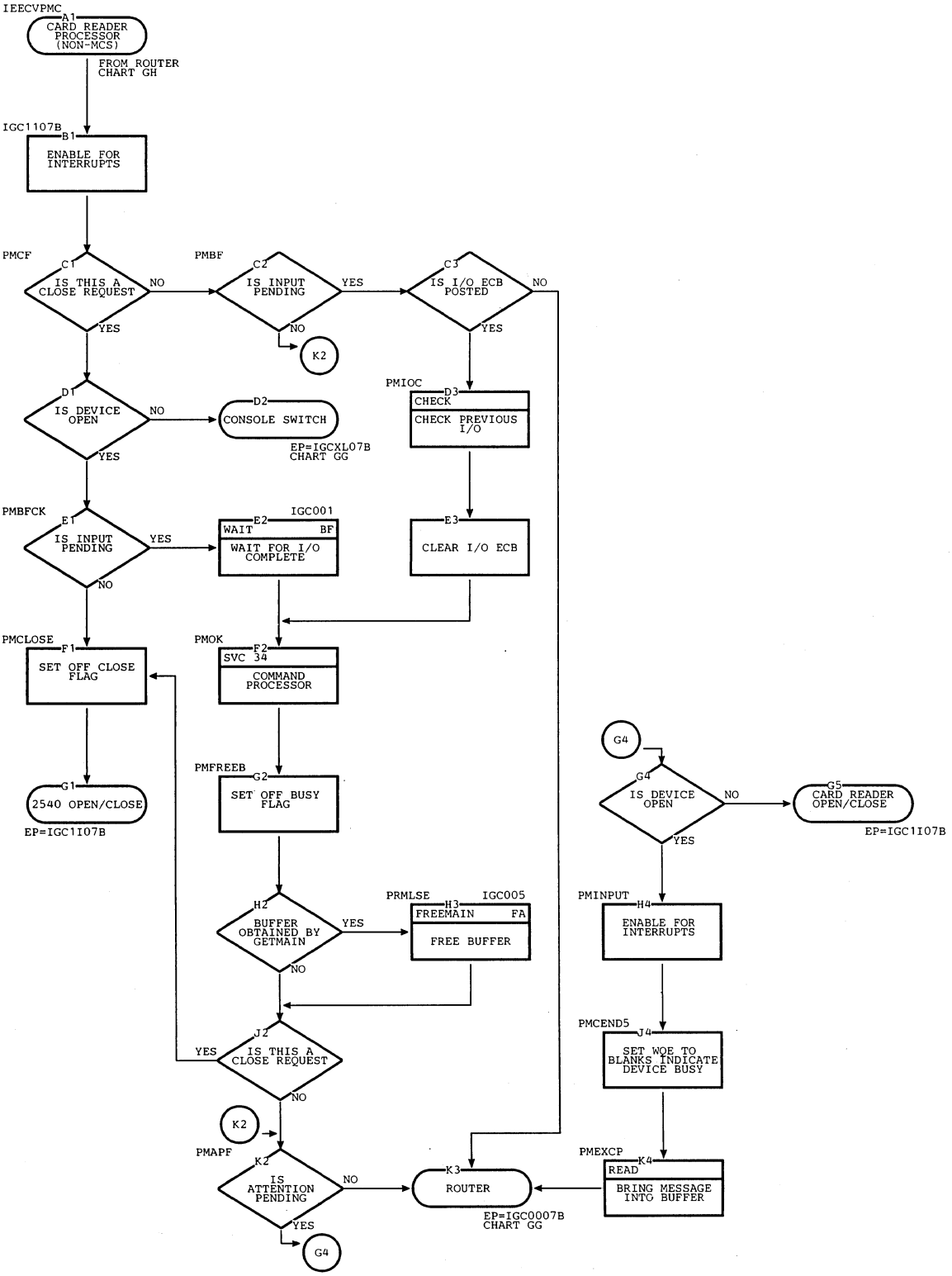


Chart GS. Communications Task Card Reader Processor (MCS)

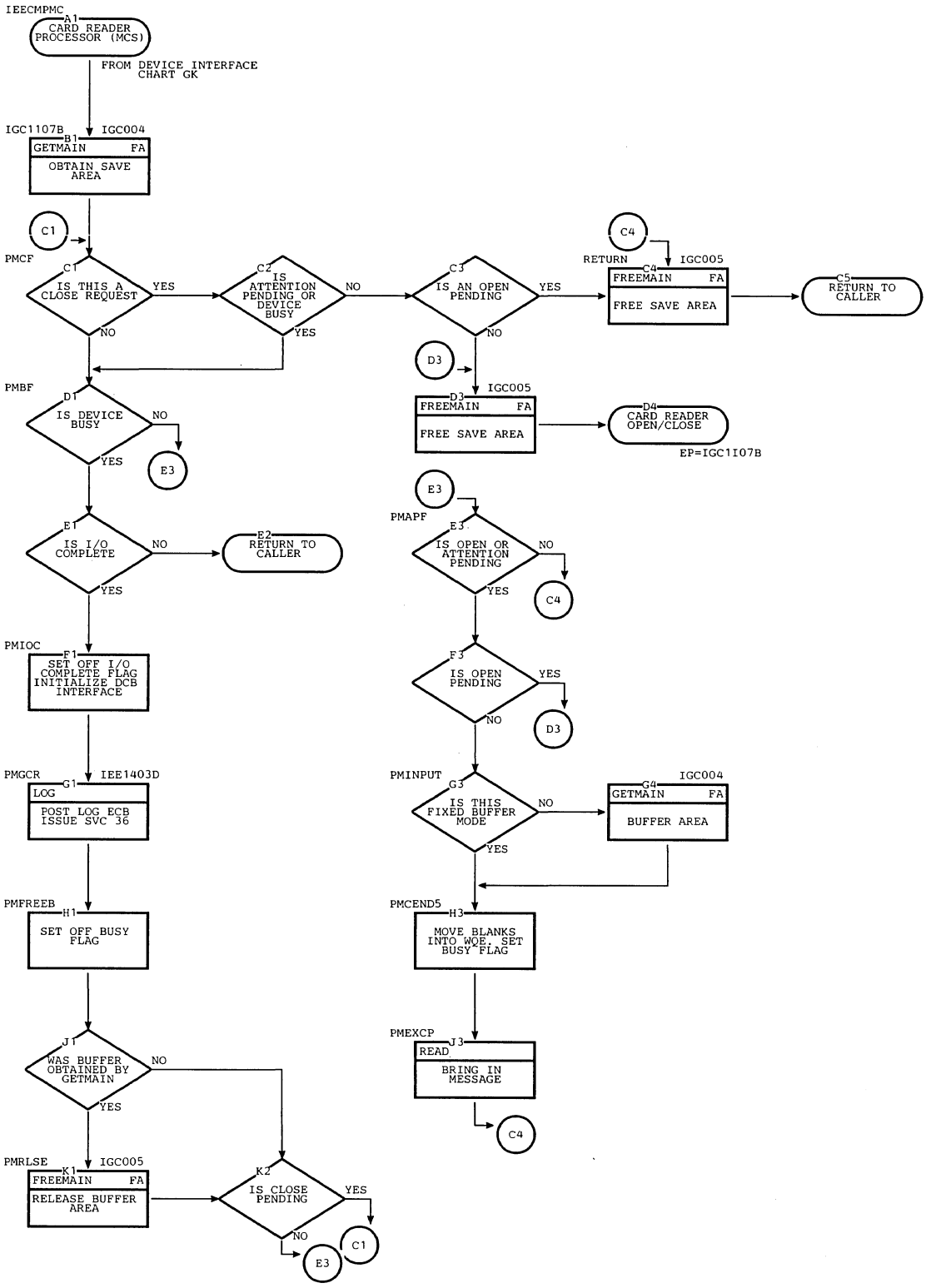


Chart GT. Communications Task 2740 Processor (MCS only) (Part 1 of 2)

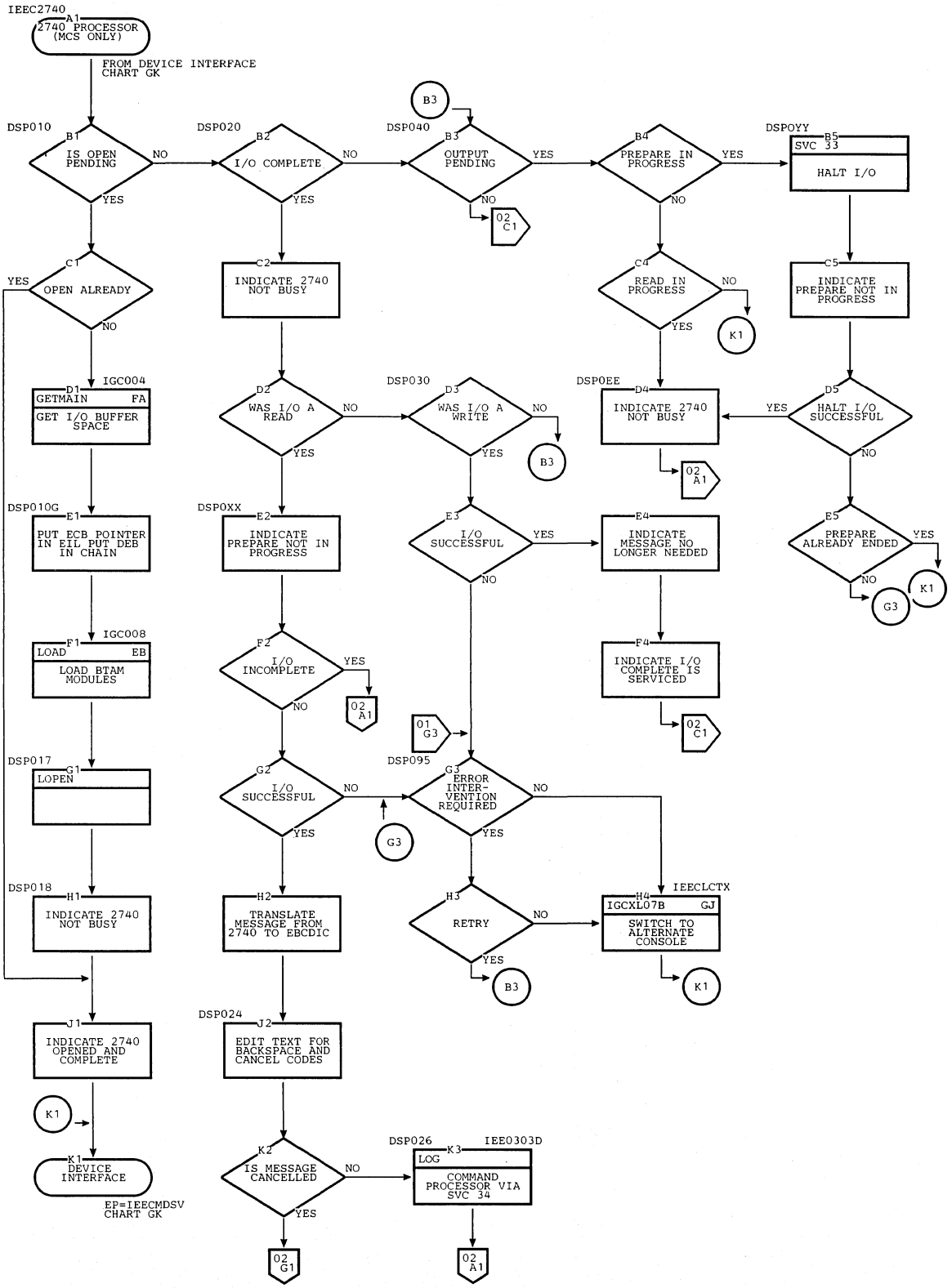


Chart GT. Communications Task 2740 Processor (MCS only) (Part 2 of 2)

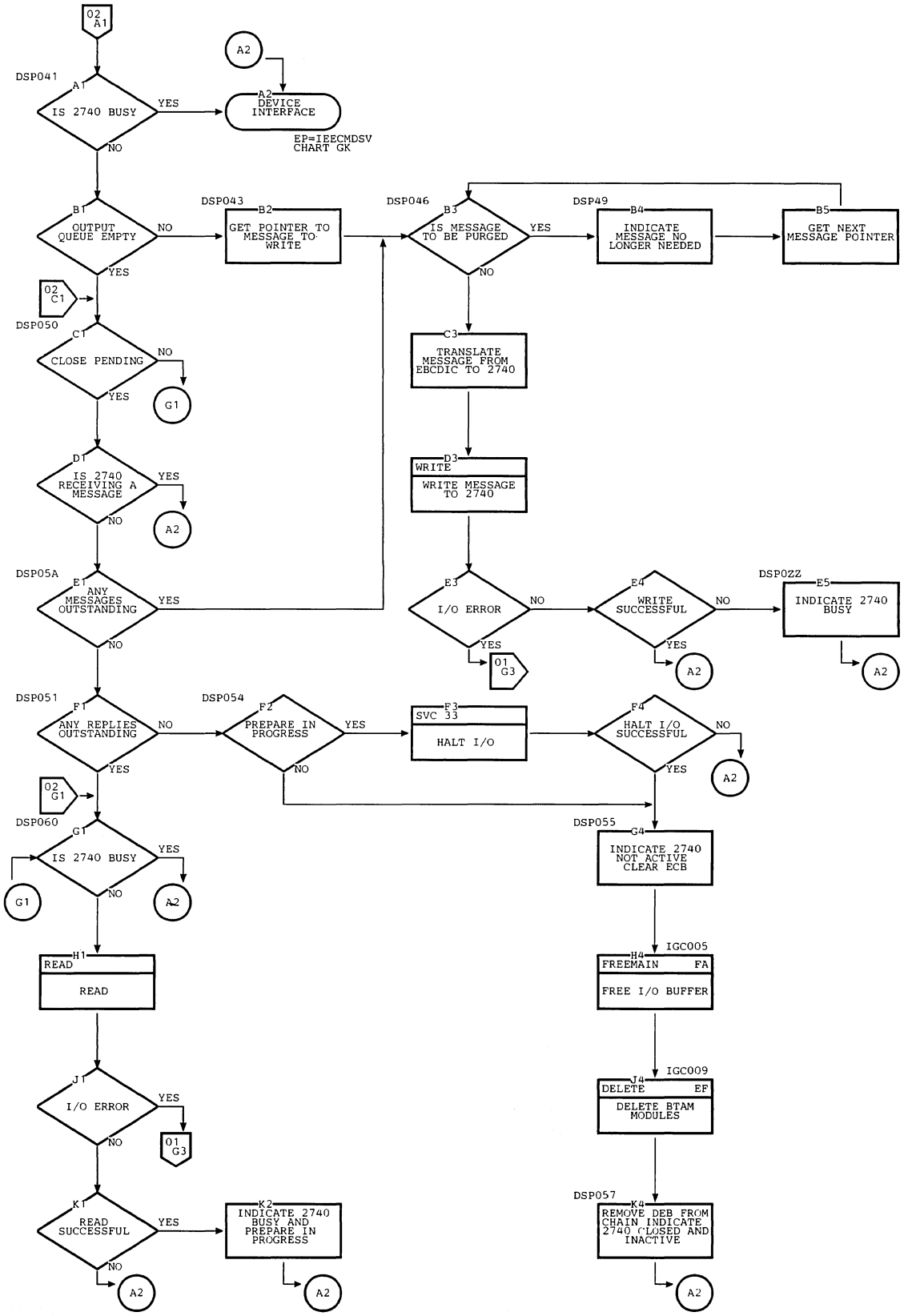


Chart GU. Communications Task Log Writer

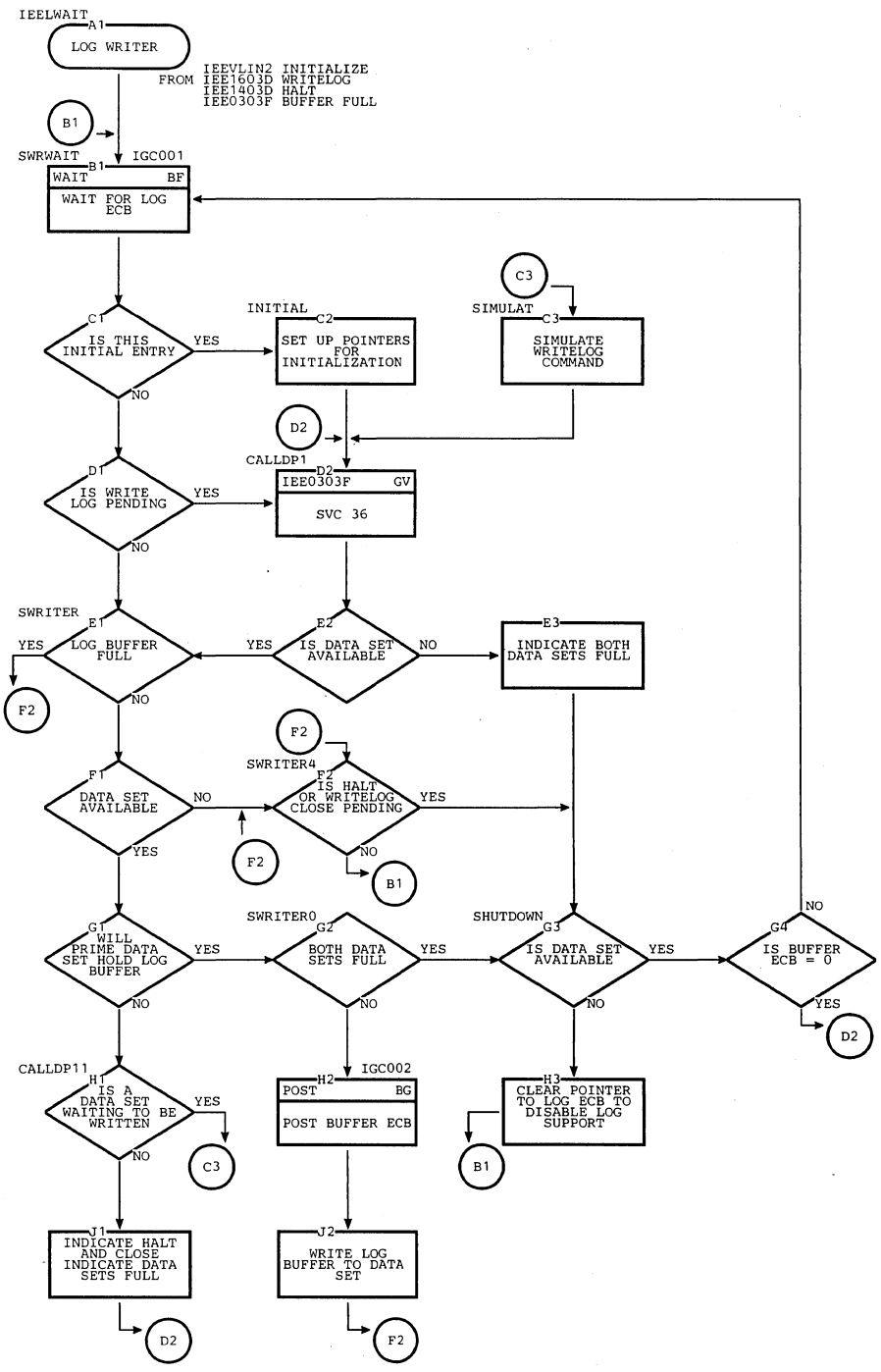


Chart GV. Communications Task LOG, Write-to-Log (Load 1)

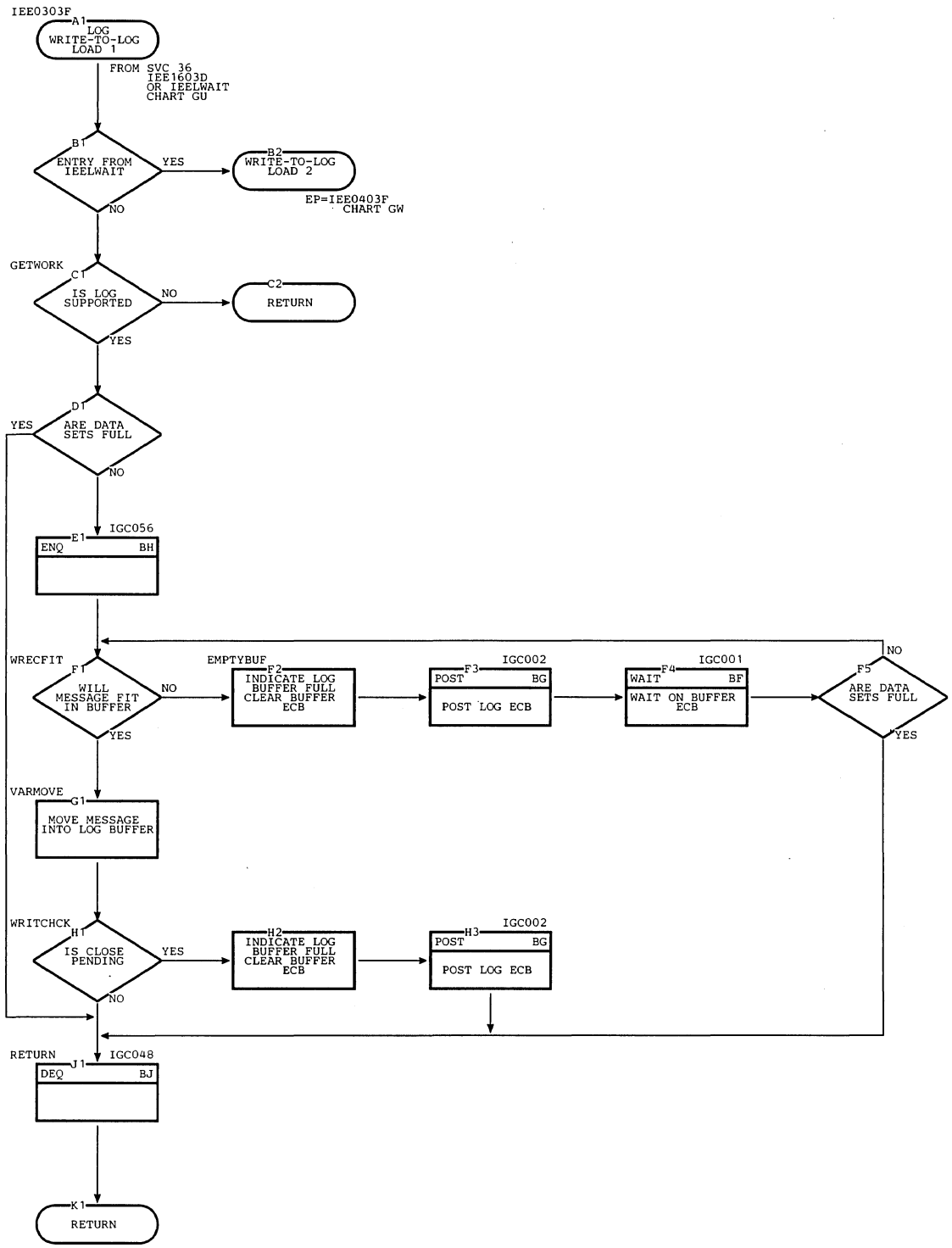


Chart GW. Communications Task LOG, Write-to-Log (Load 2)

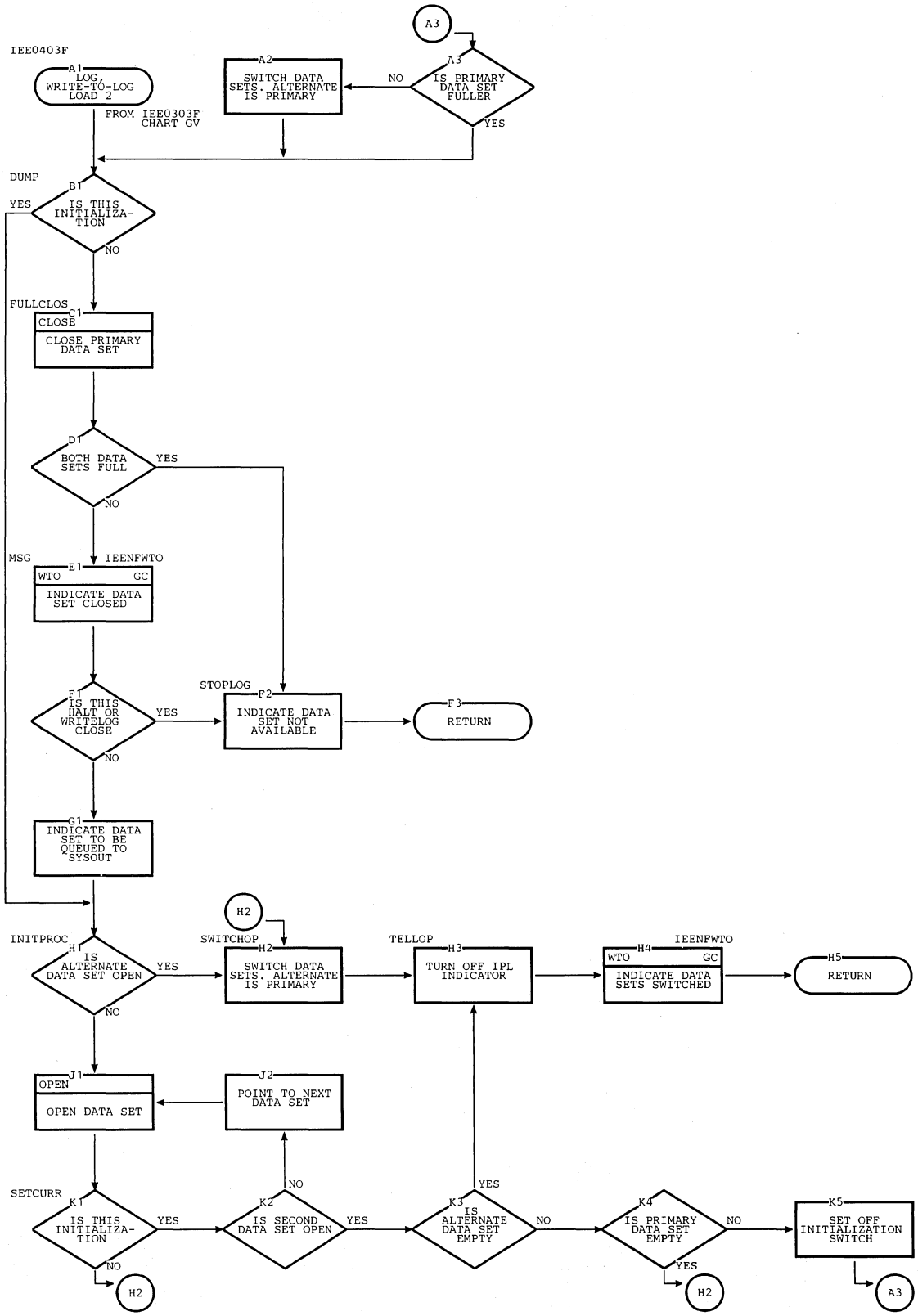


Chart GX. 1052 Processor Load 2 (Part 1 of 2)

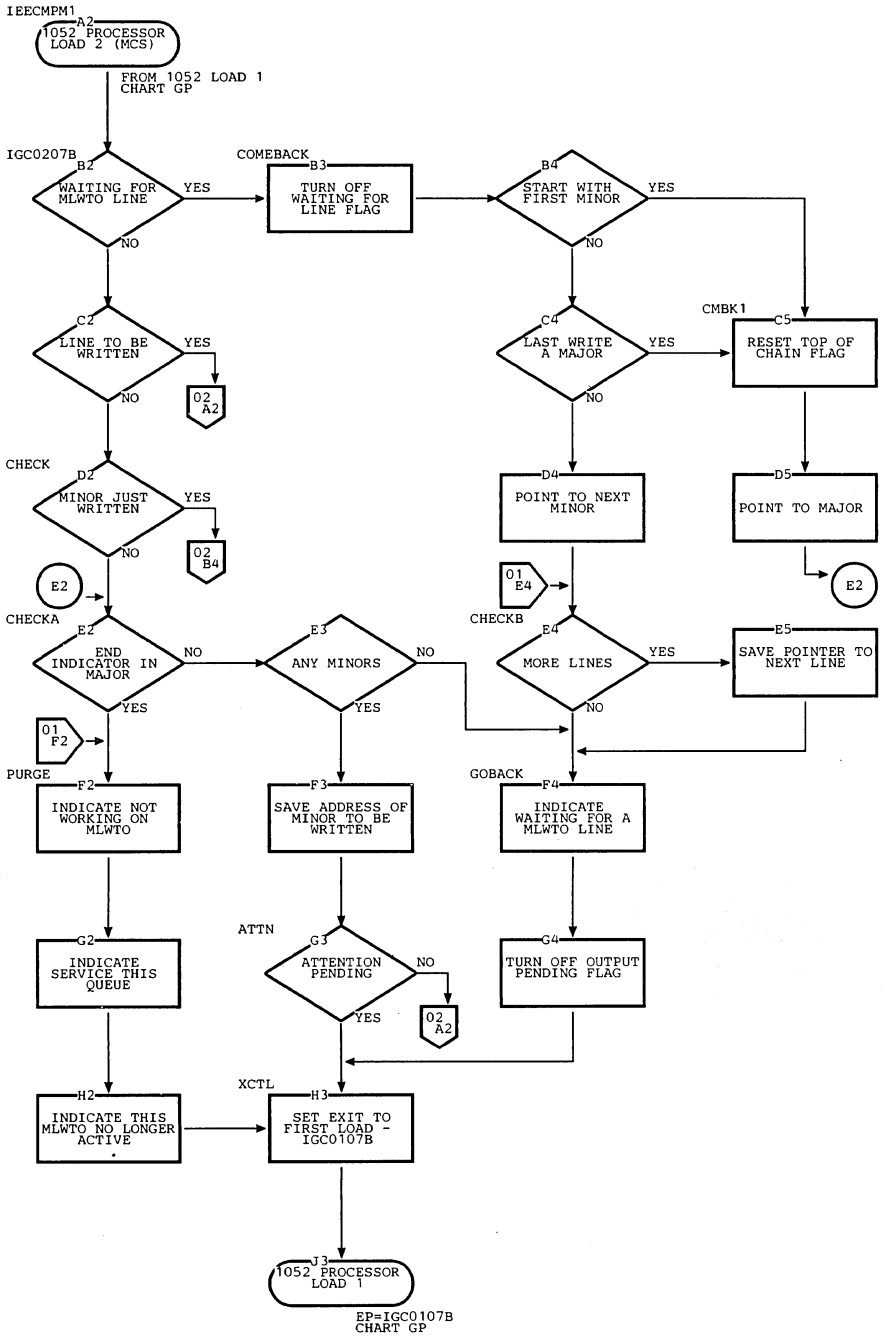


Chart GX. 1052 Processor Load 2 (Part 2 of 2)

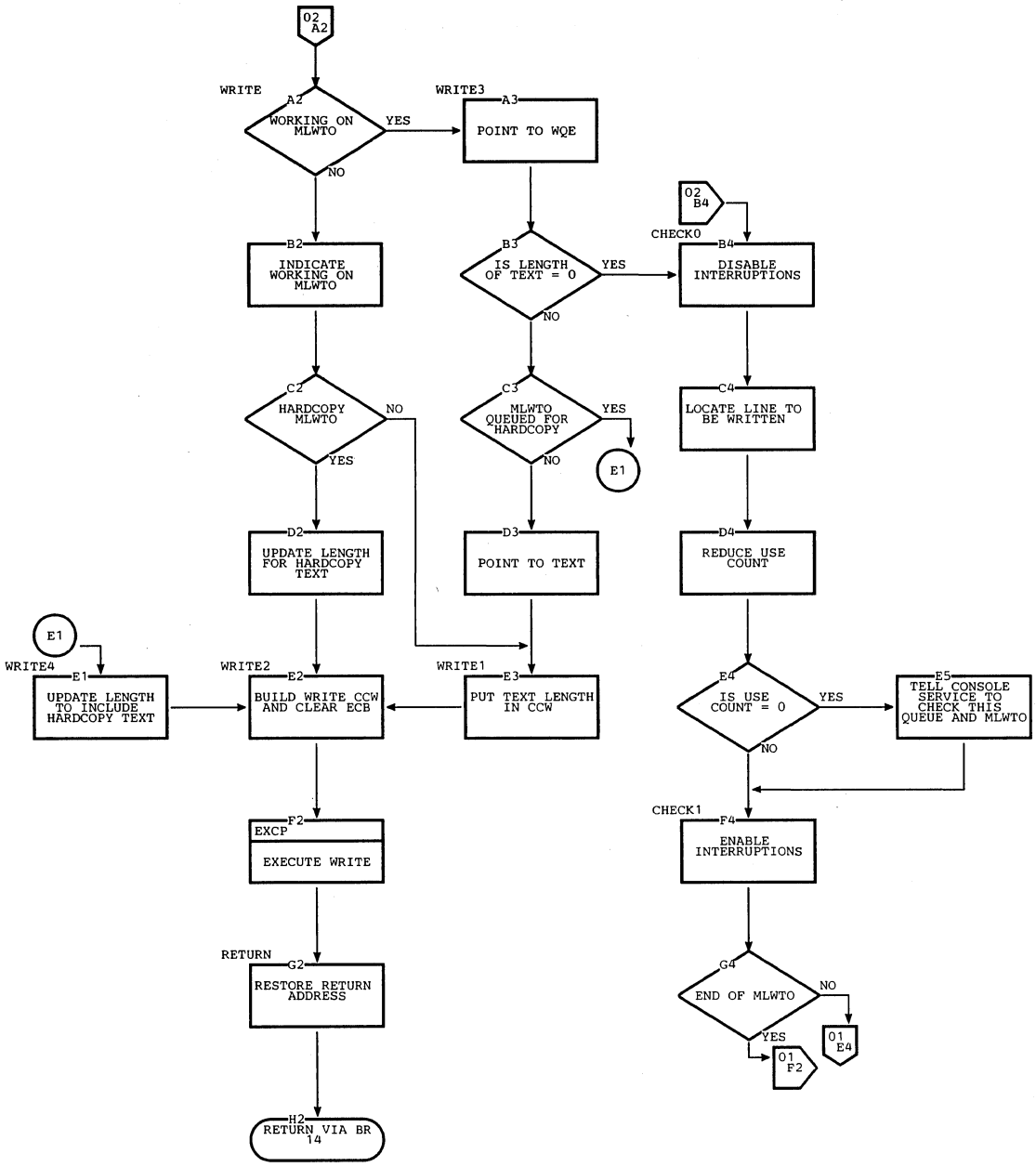
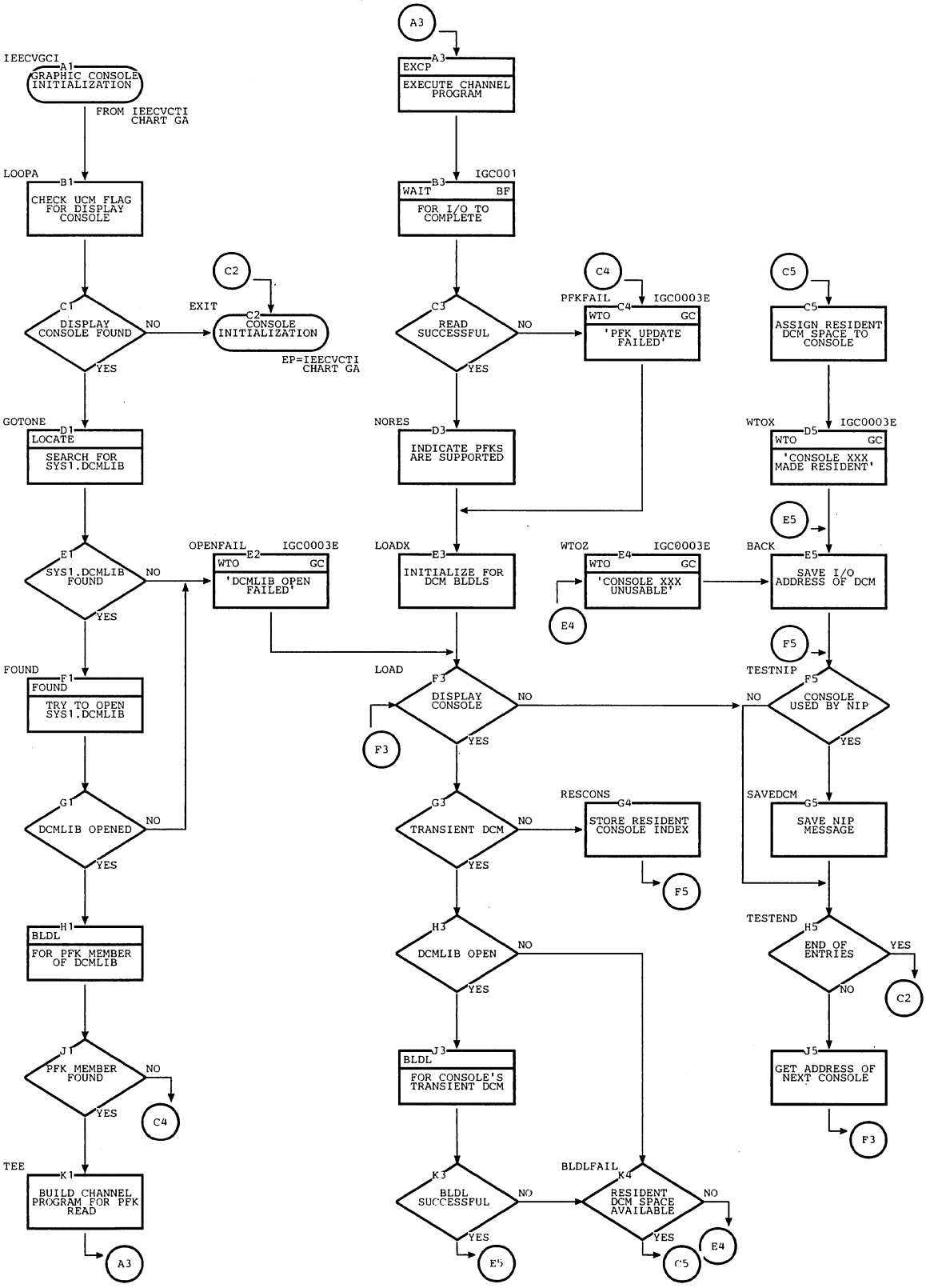


Chart GY. Graphic Console Initialization



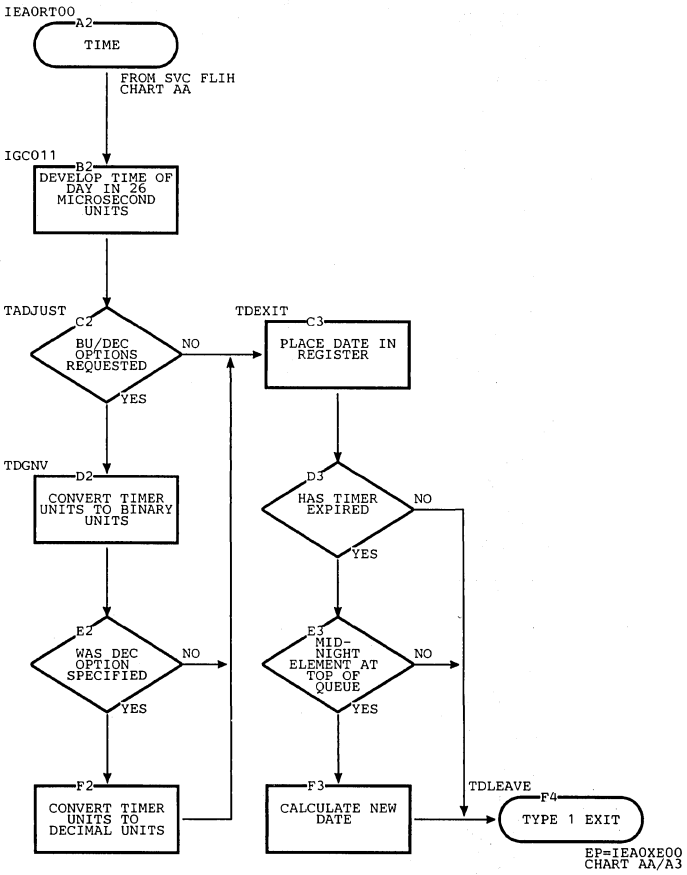

IEAORT00
A2
TIME
FROM SVC FLIH
CHART AA
IGC011
B2
DEVELOP TIME OF DAY IN 26 MICROSECOND UNITS
TADJUST
C2
BU/DEC OPTIONS REQUESTED
NO
YES
TDEXIT
C3
PLACE DATE IN REGISTER
TDGNV
D2
CONVERT TIMER UNITS TO BINARY UNITS
D3
HAS TIMER EXPIRED
NO
YES
E2
WAS DEC OPTION SPECIFIED
NO
YES
E3
MID-NIGHT ELEMENT AT TOP OF QUEUE
NO
YES
F2
CONVERT TIMER UNITS TO DECIMAL UNITS
F3
CALCULATE NEW DATE
TDLEAVE
F4
TYPE 1 EXIT
EP=IEA0XE00
CHART AA/A3

Chart HB. Timer Second-Level Interruption Handler (Part 1 of 2)

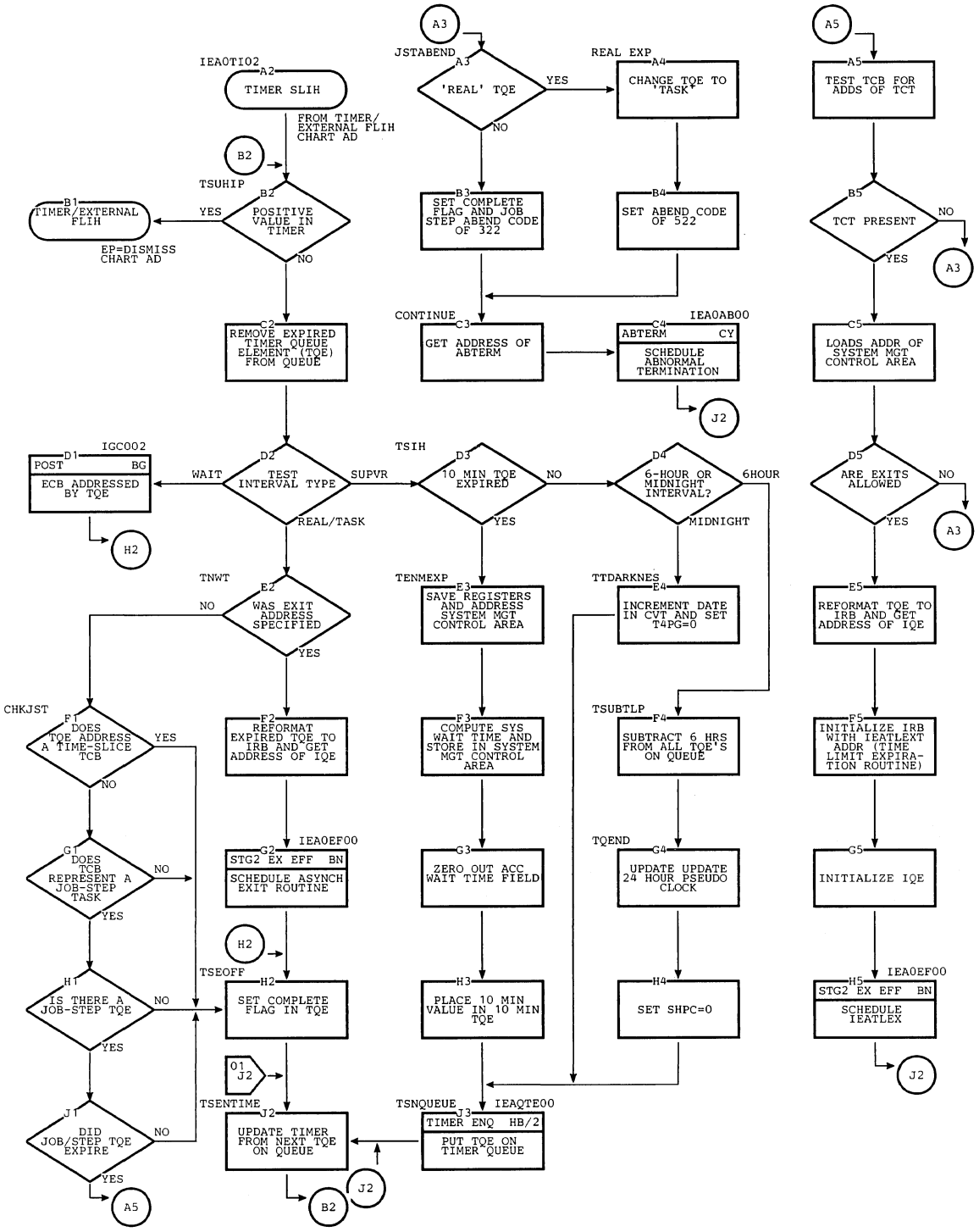


Chart HB. Timer Second-Level Interruption Handler (Part 2 of 2)

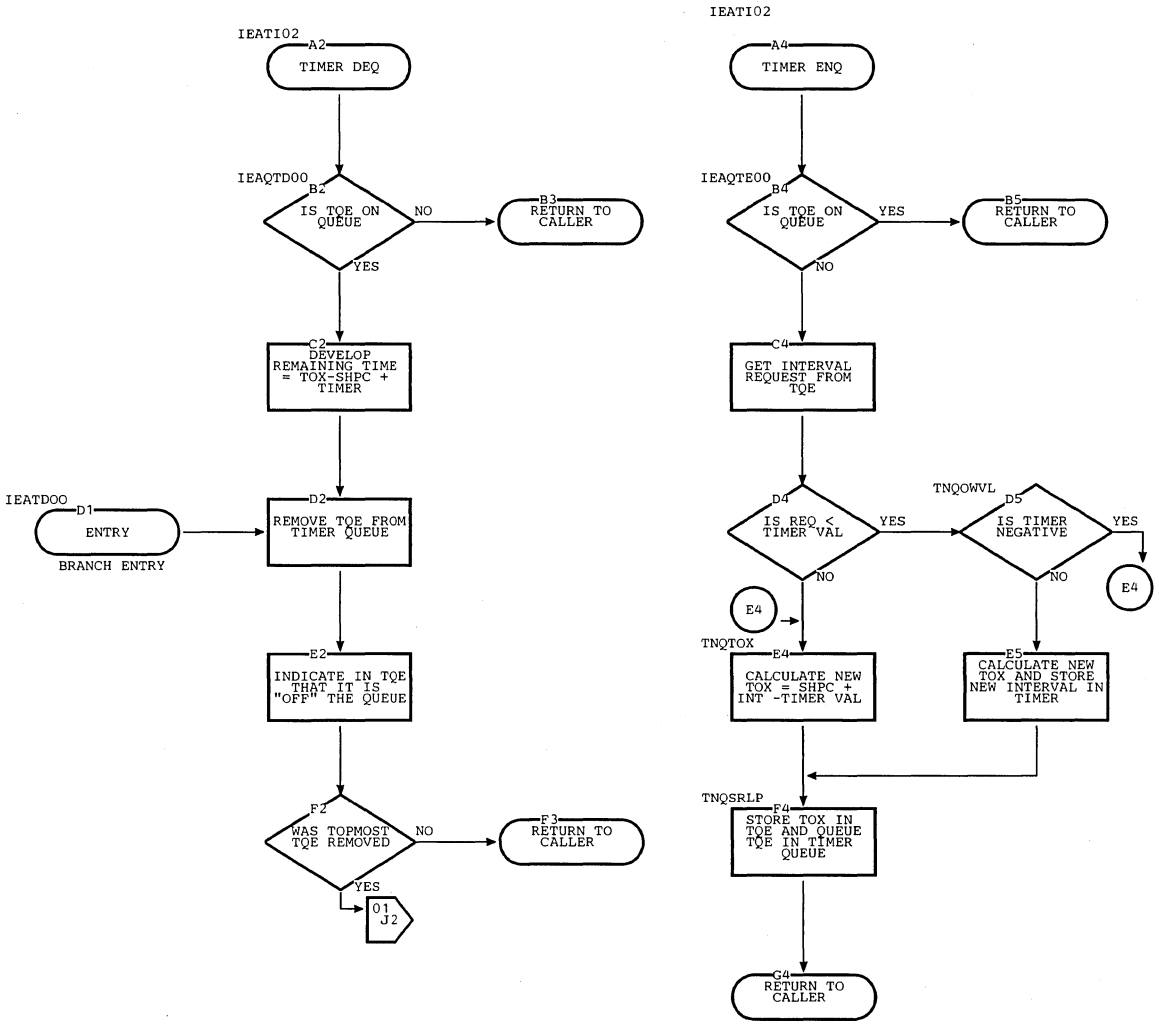


Chart HC. TTIMER

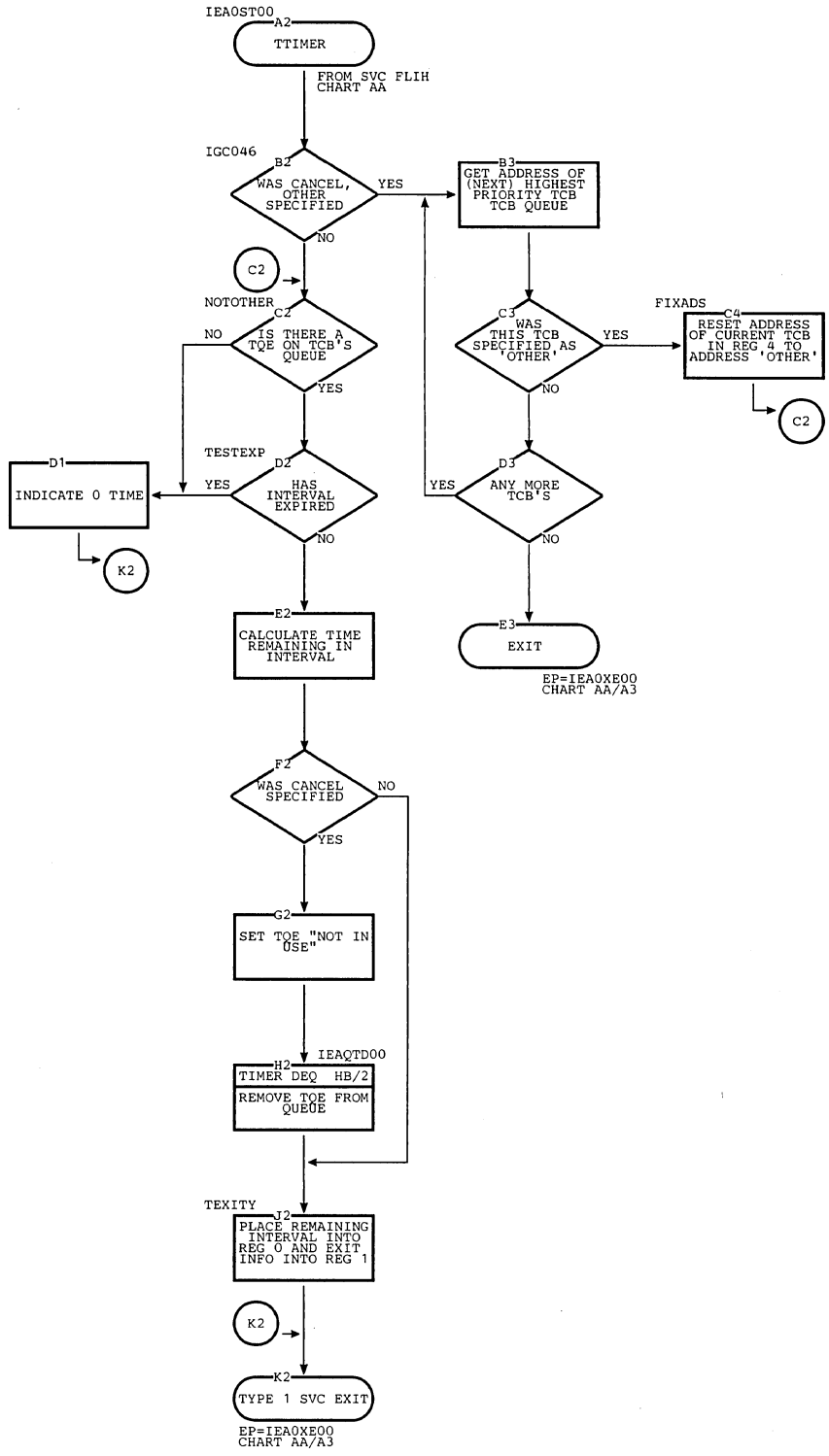


Chart HD. STIMER

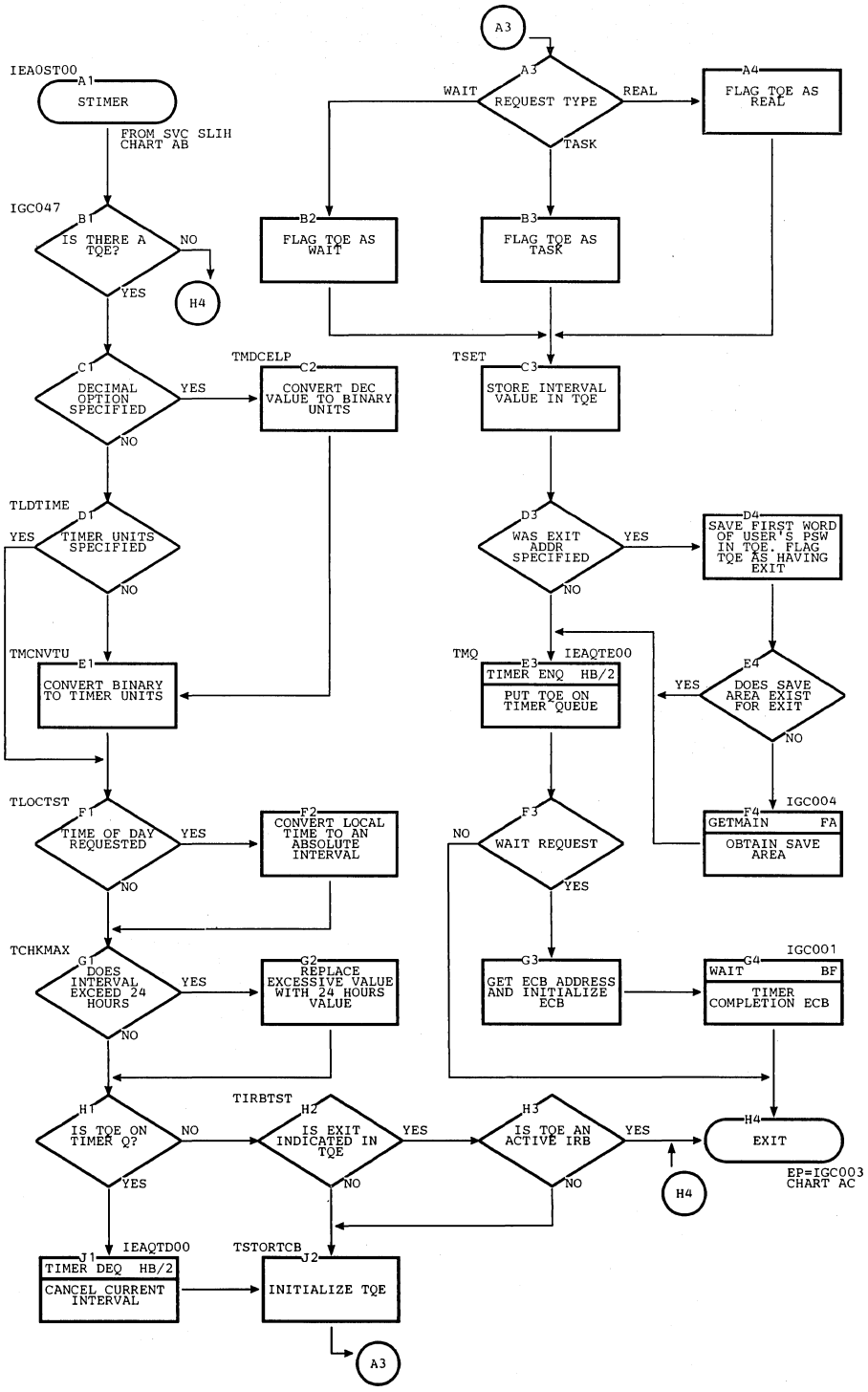


Chart HE. TIME with Time-of-Day Clock

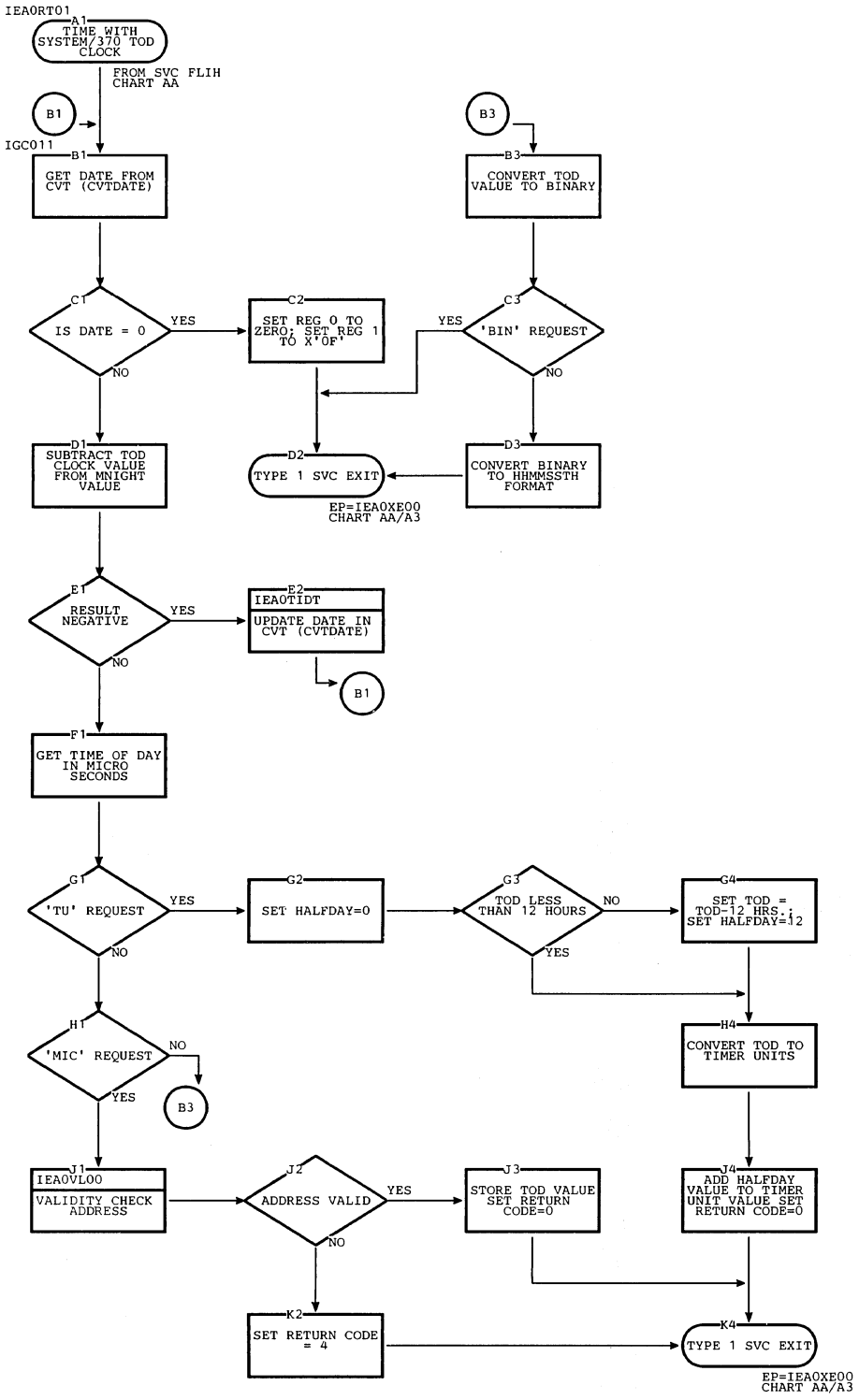


Chart HF. TTIMER with Time-of-Day Clock

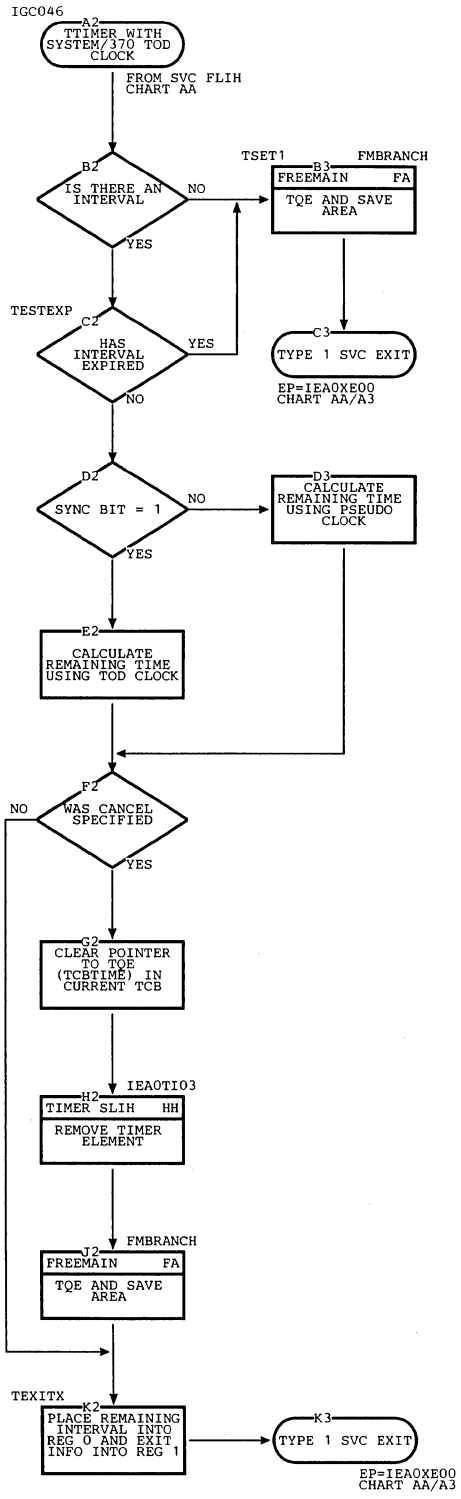


Chart HG. STIMER with Time-of-Day Clock

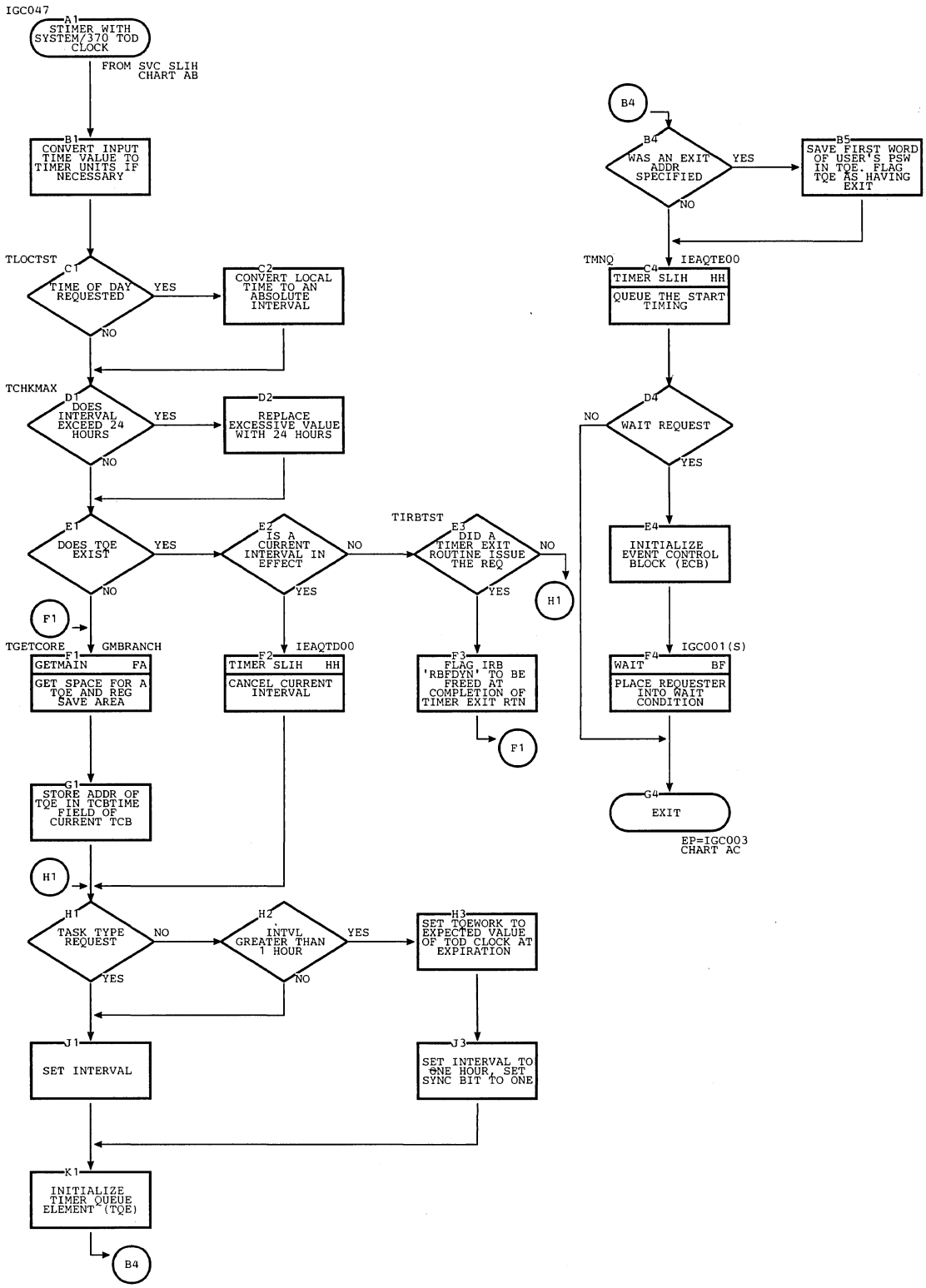


Chart HH. Timer Second-Level Interruption Handler with Time-of-Day Clock

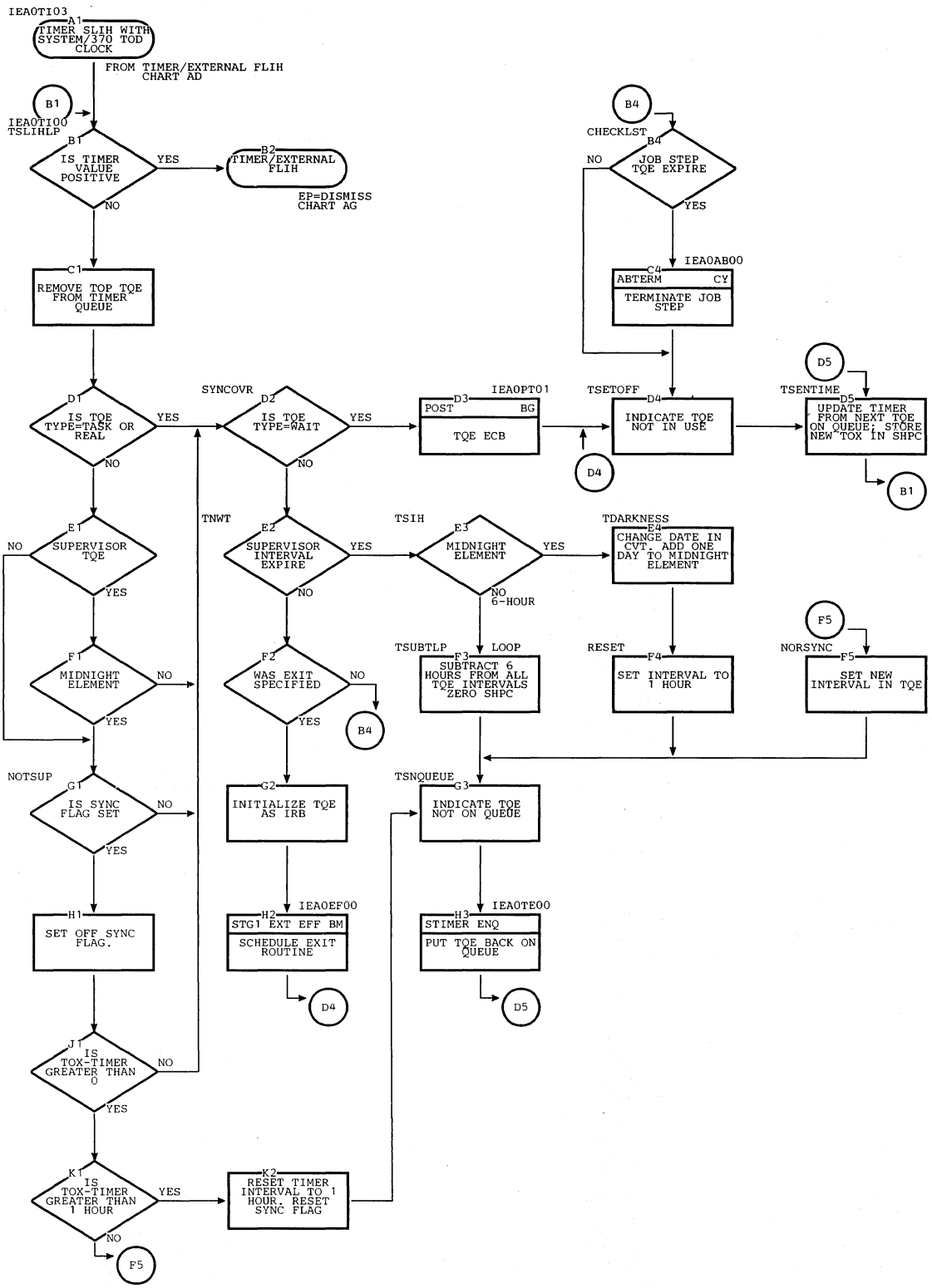


Chart IA. Overlay Supervision

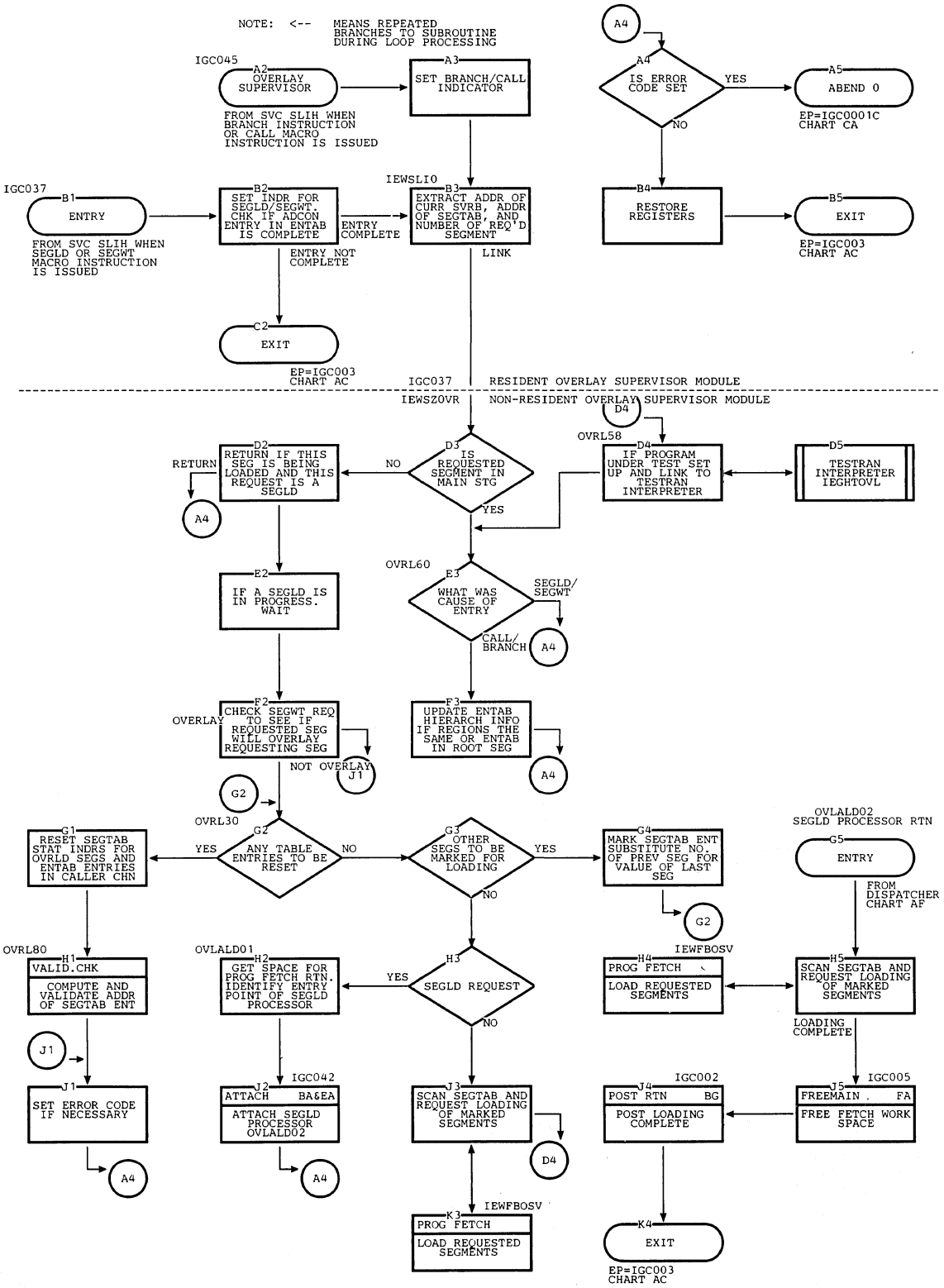


Chart JA. DIDOCS Processor 1, Load 1 (Part 1 of 2)

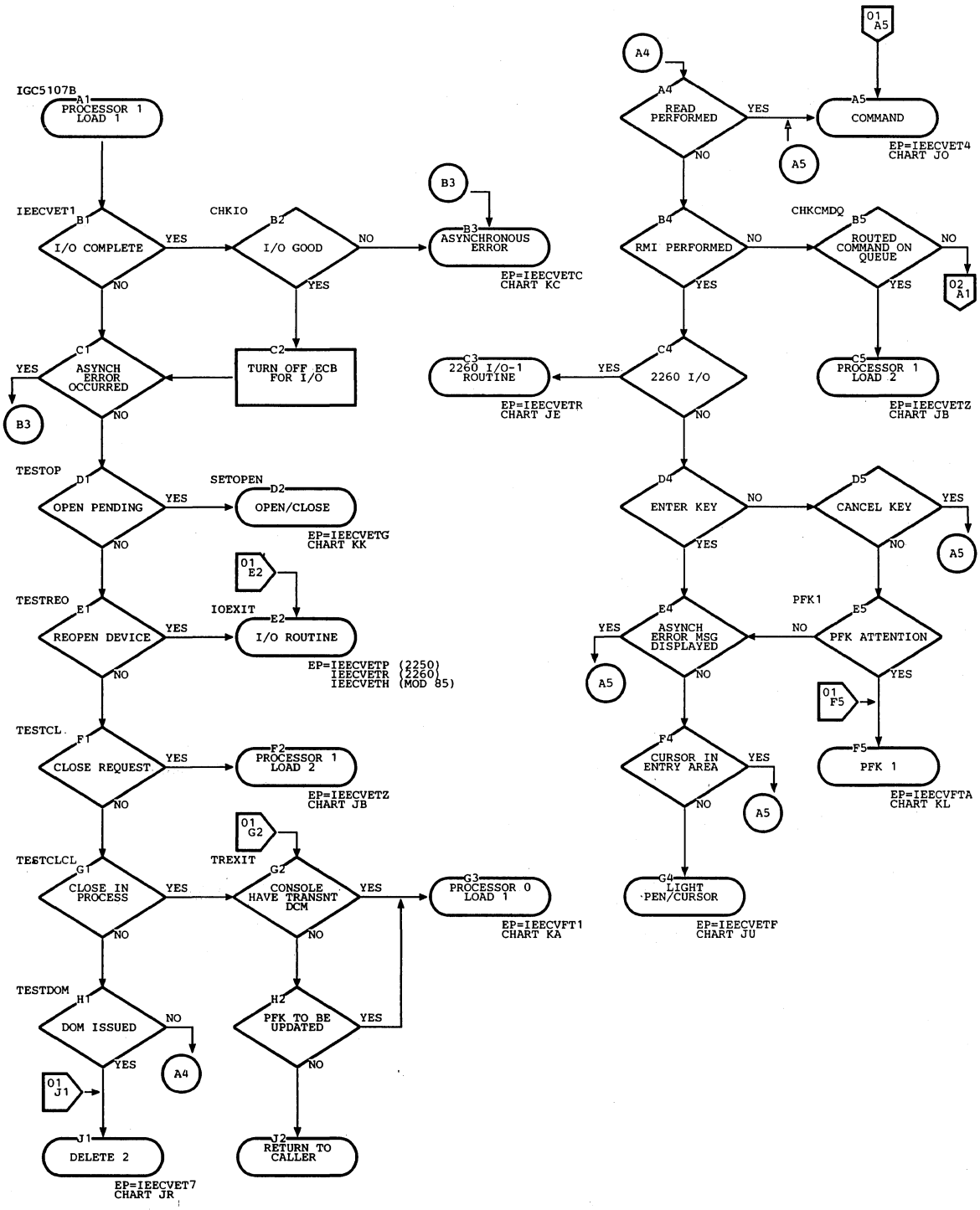


Chart JA. DIDOCS Processor 1, Load 1 (Part 2 of 2)

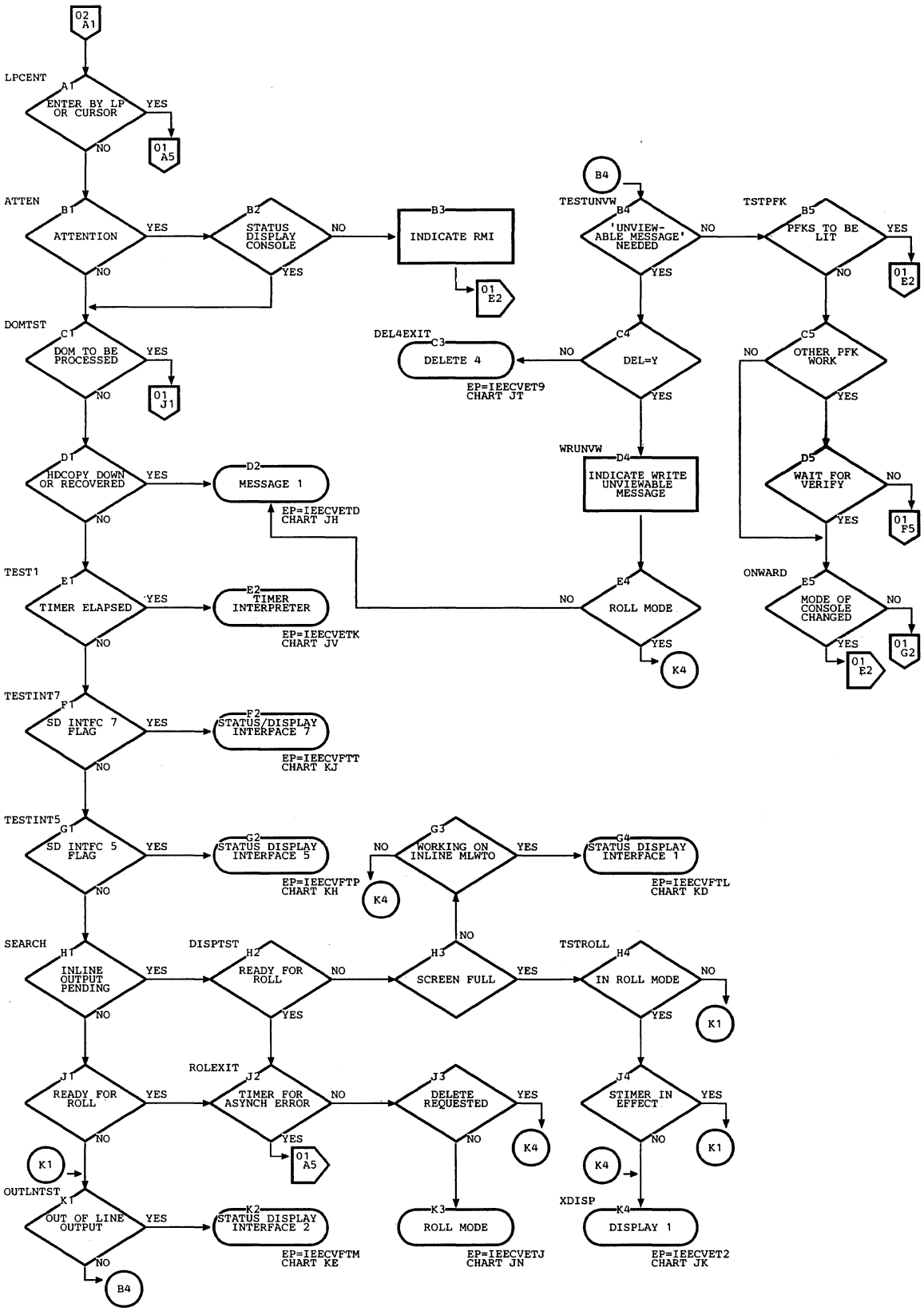


Chart JB. DIDOCS Processor 1, Load 2

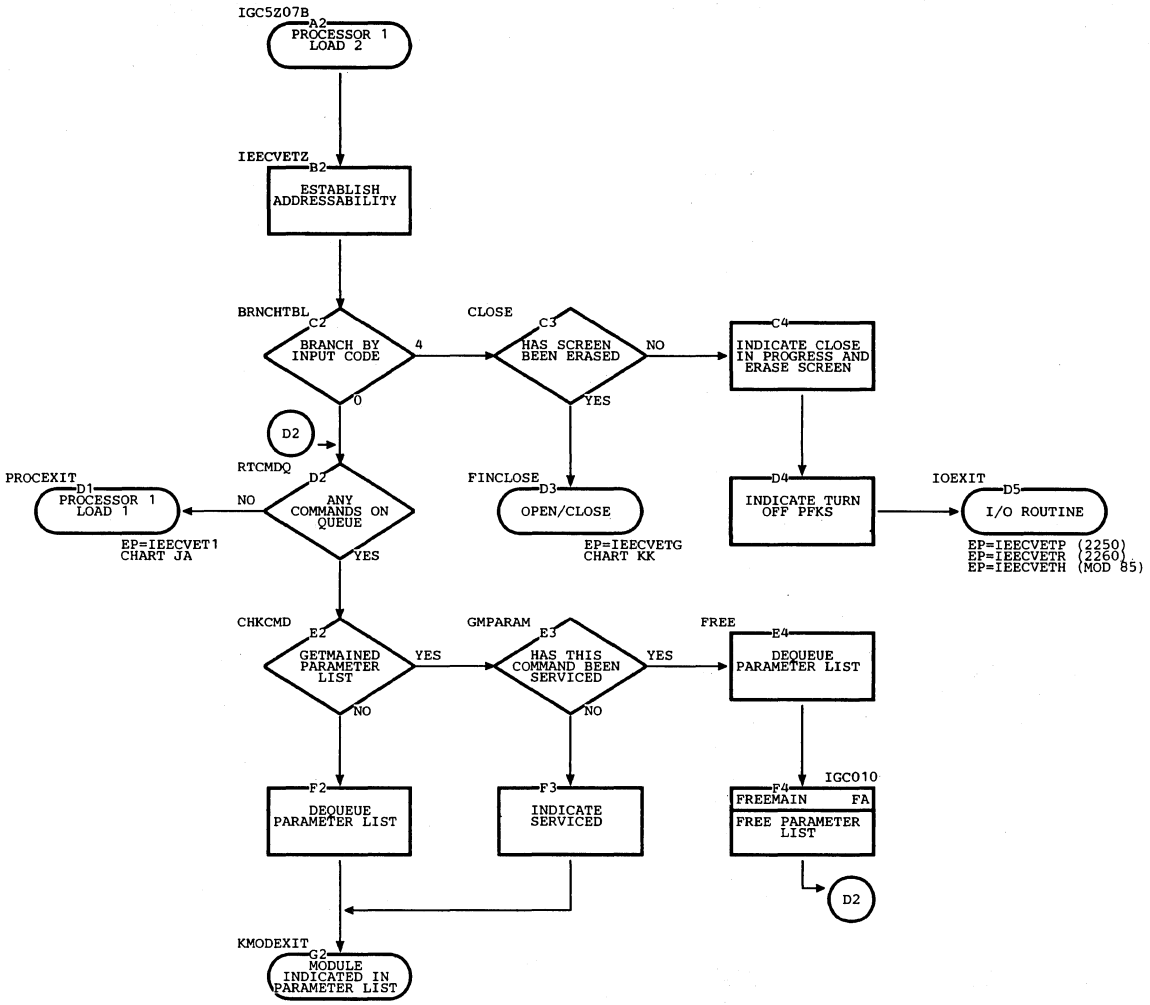


Chart JC. DIDOCS 2250 I/O 1 (Part 1 of 2)

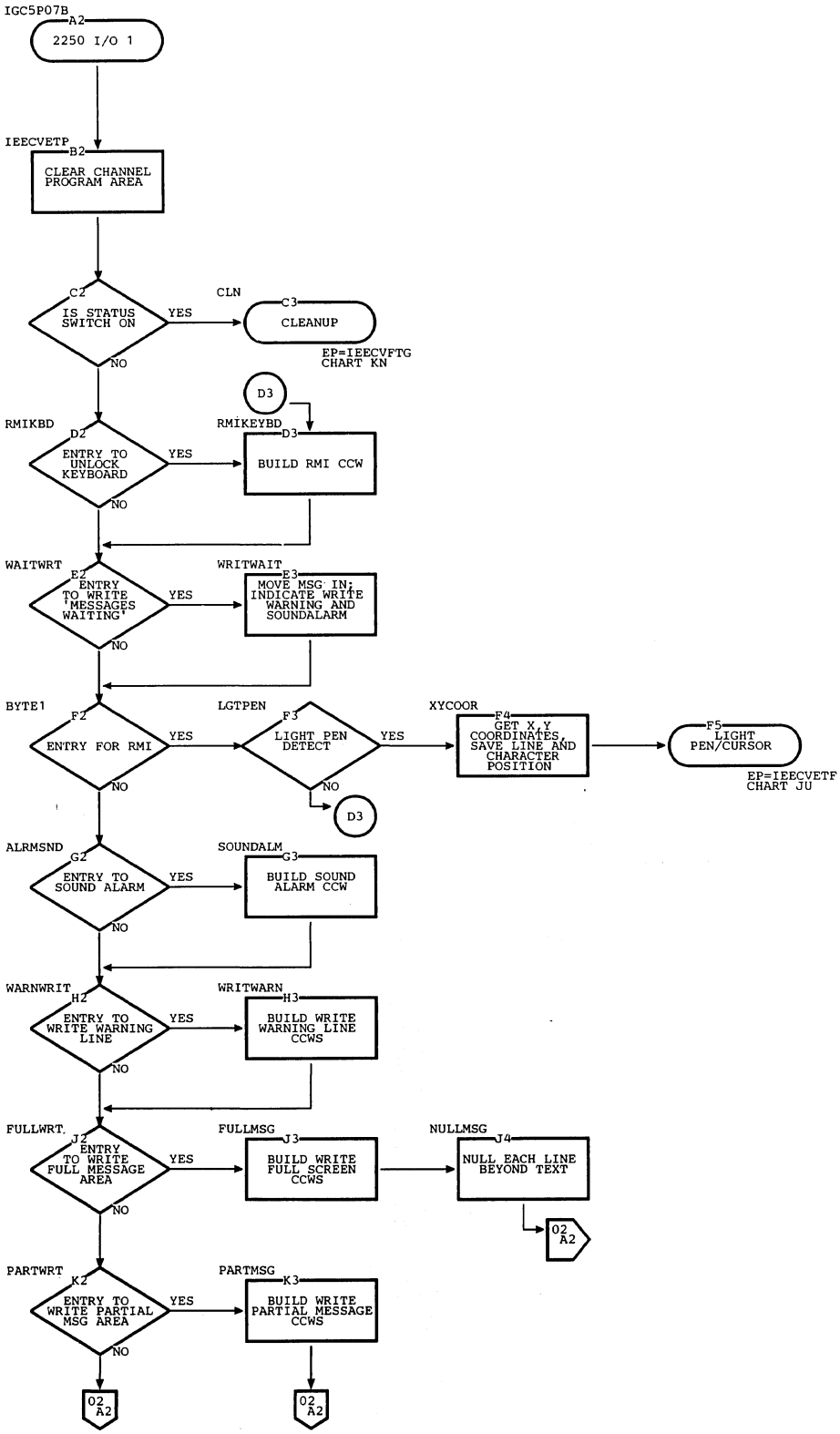


Chart JC. DIDOCS 2250 I/O 1 (Part 2 of 2)

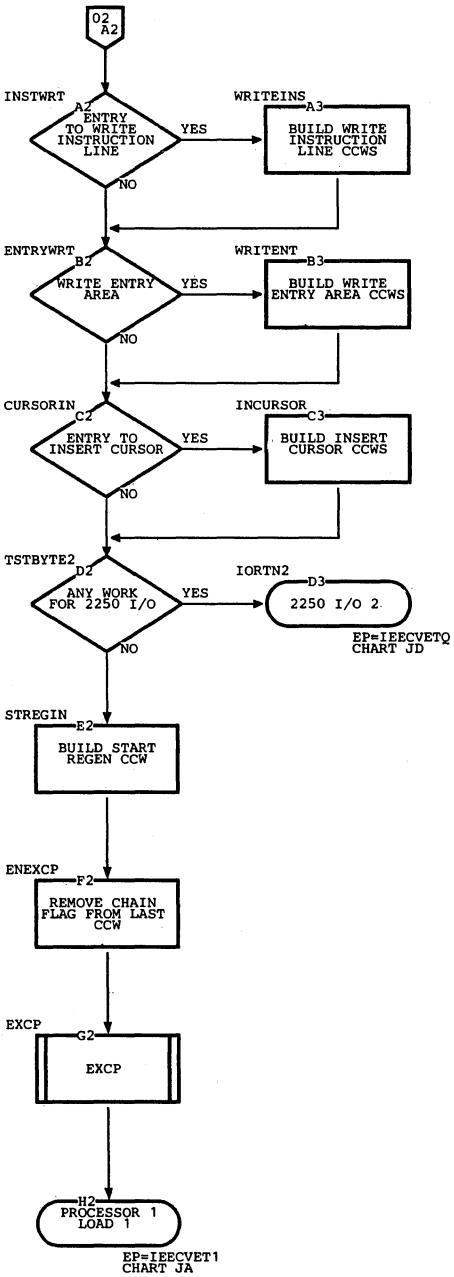


Chart JD. DIDOCS 2250 I/O 2

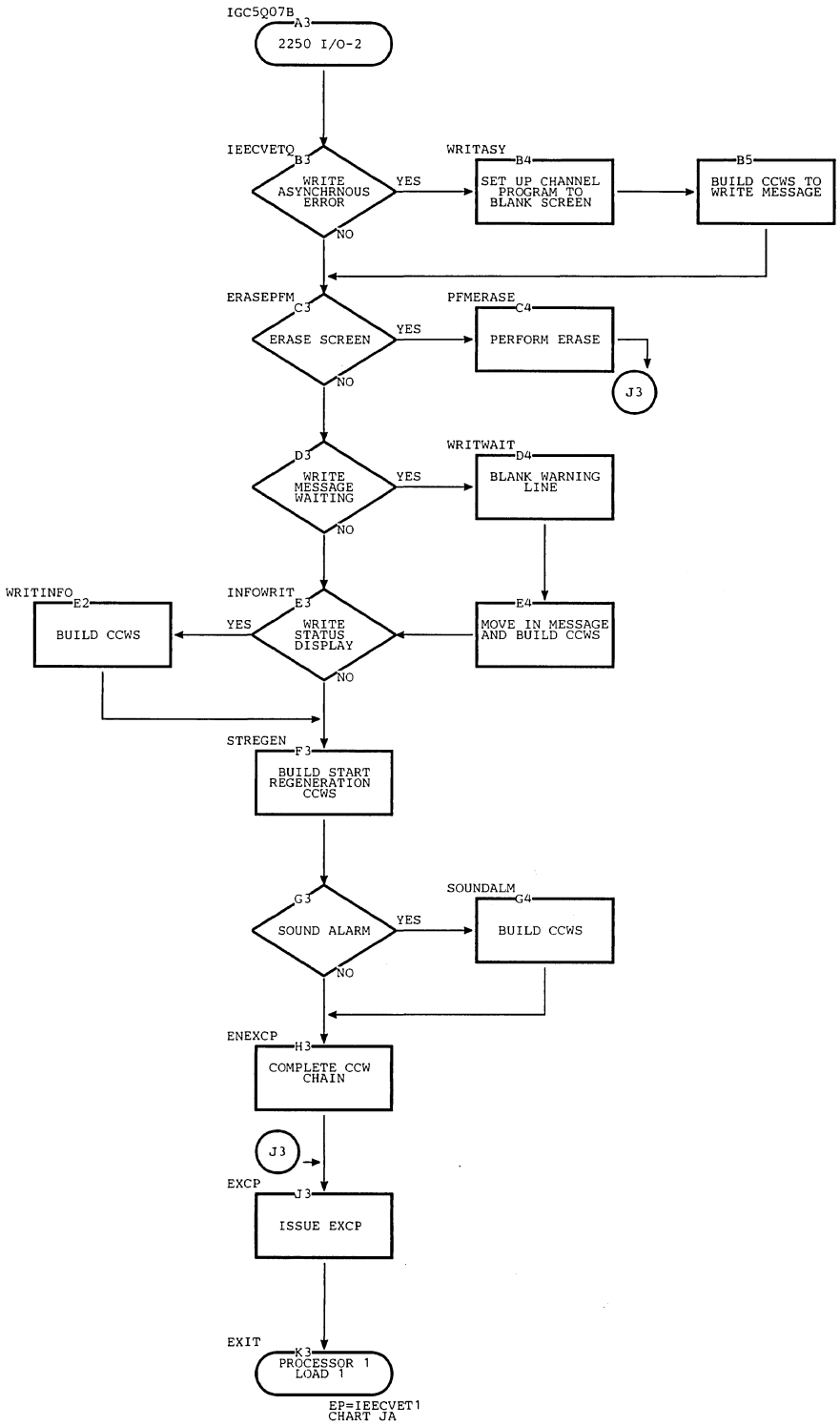


Chart JE. DIDOCS 2260 I/O 1

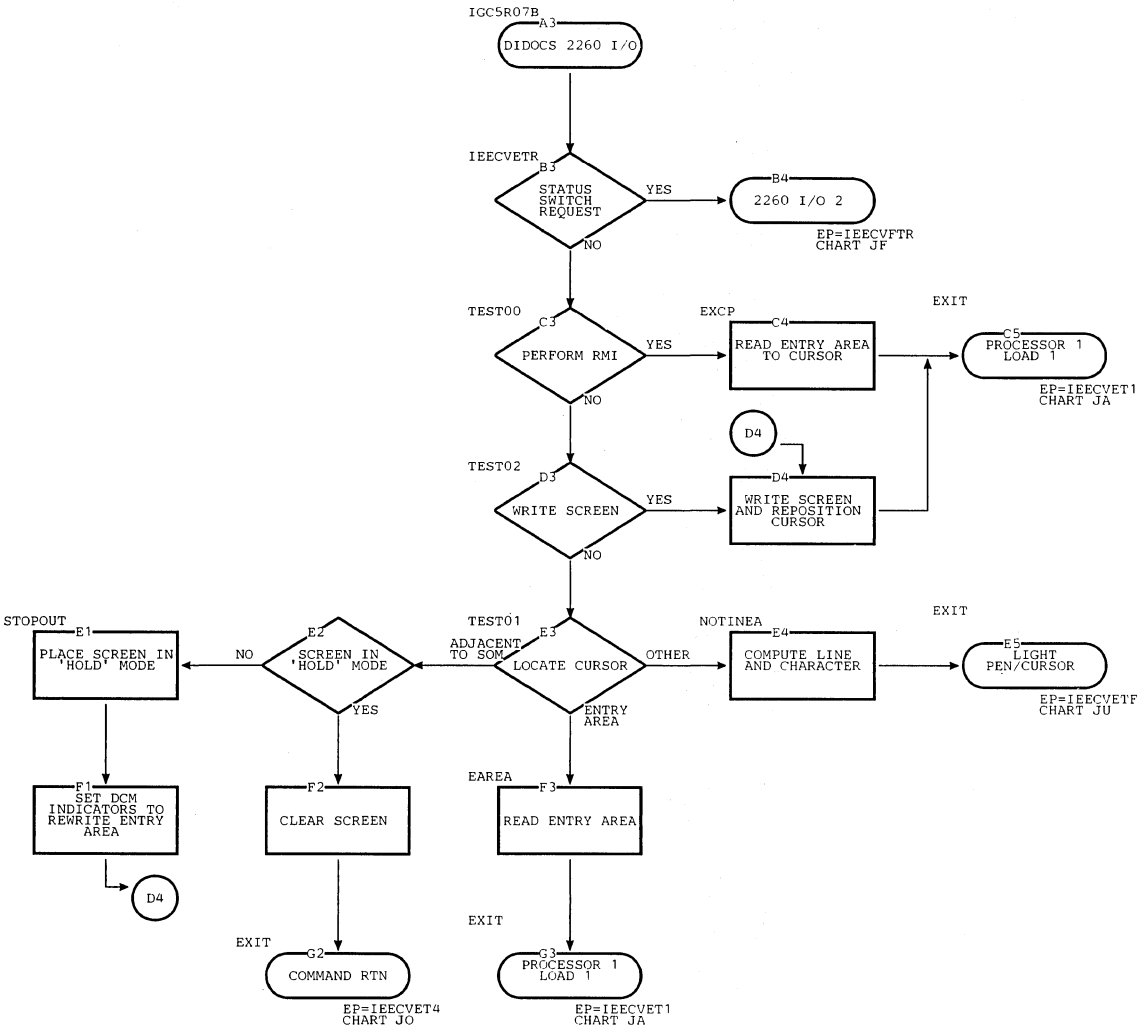


Chart JF. DIDOCS 2260 I/O 2

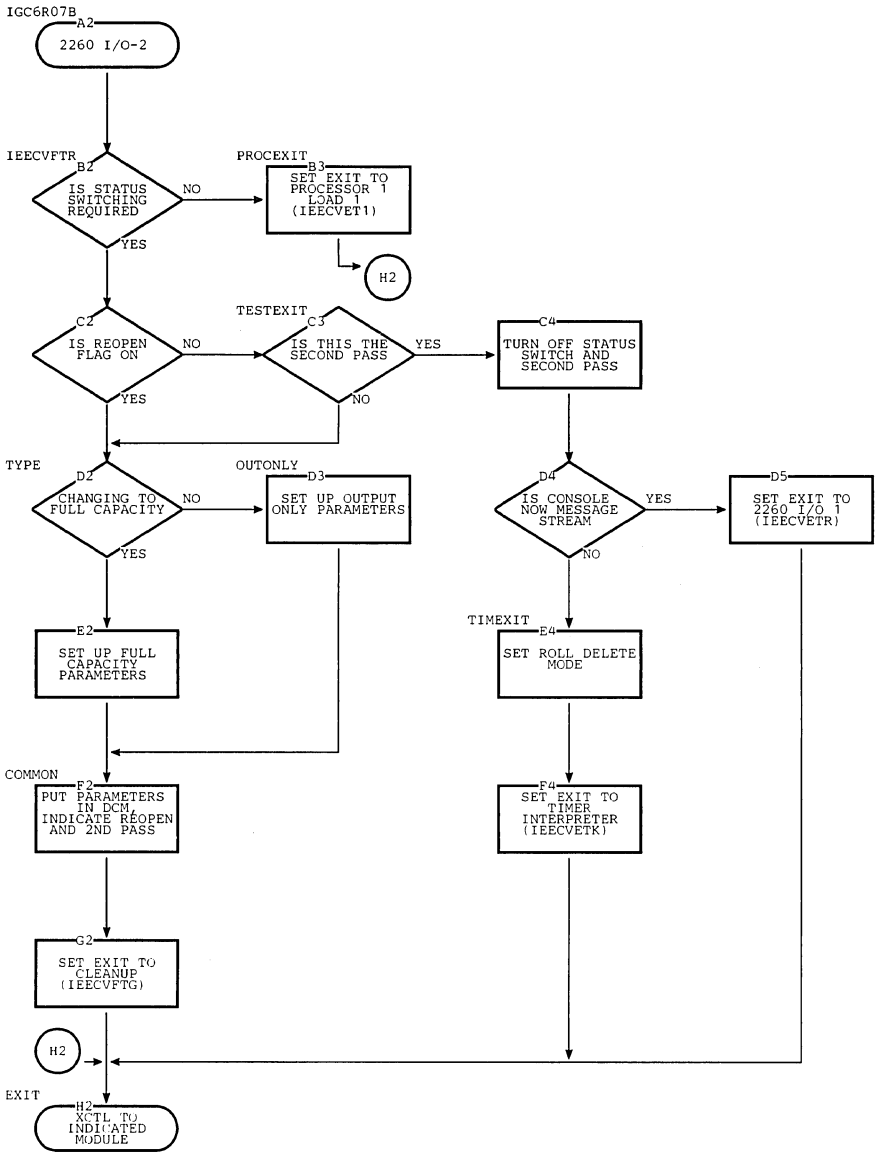


Chart JG. DIDOCS Model 85 I/O

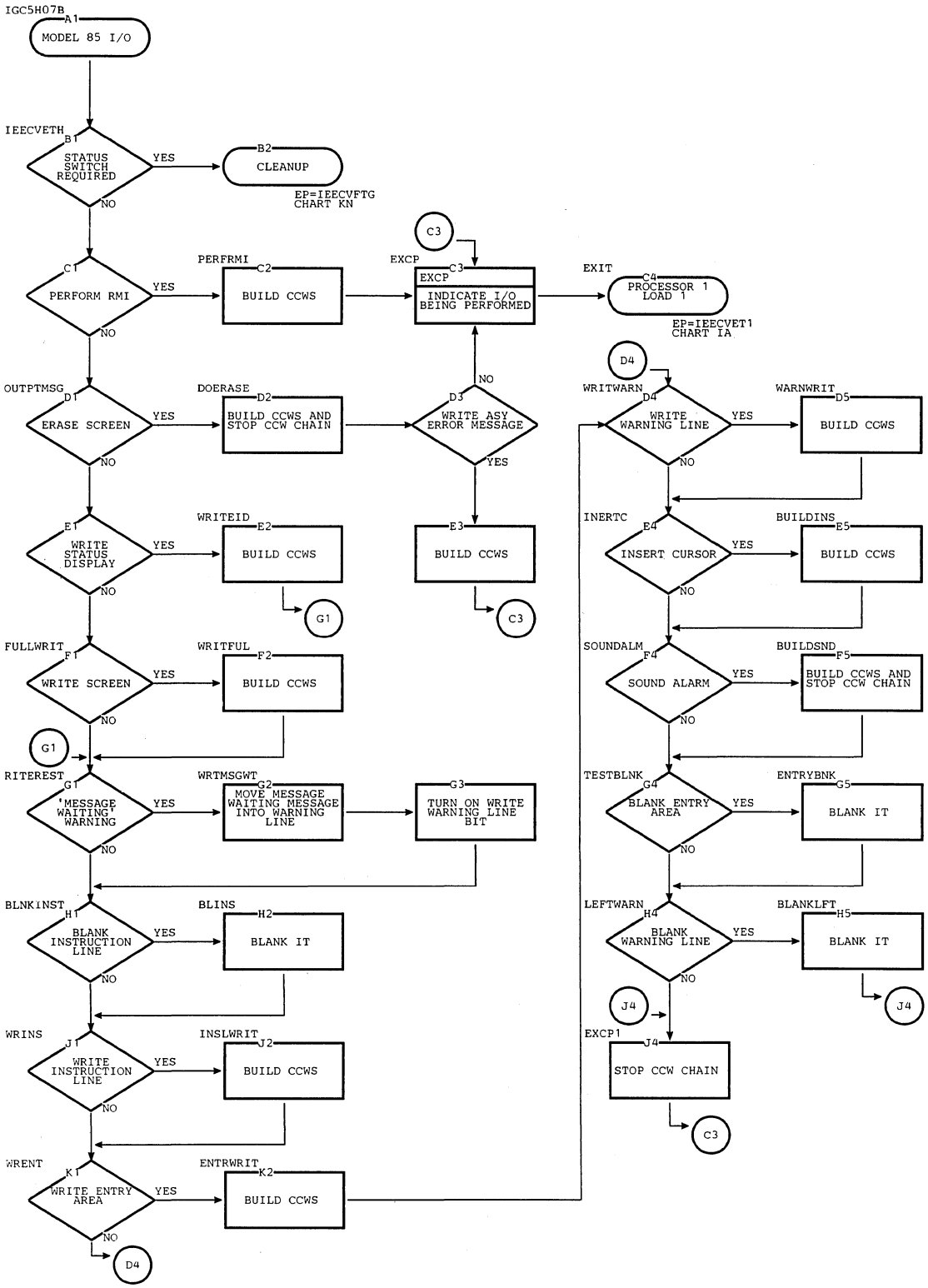


Chart JH. DIDOCS Message 1

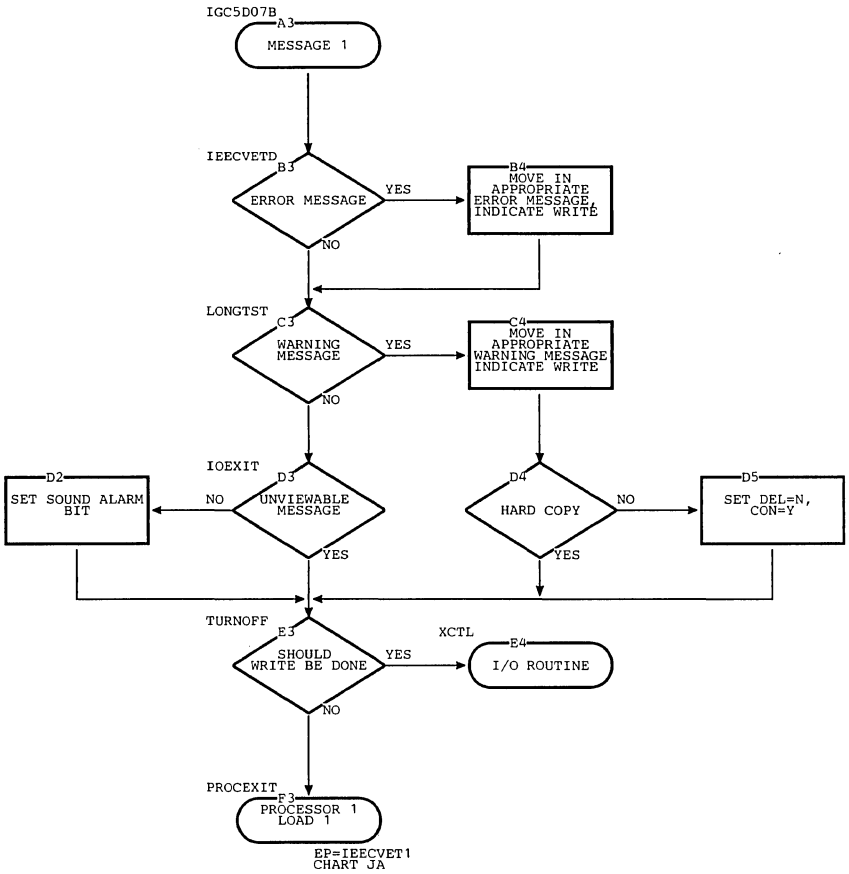


Chart JI. DIDOCS Message 2 (Part 1 of 2)

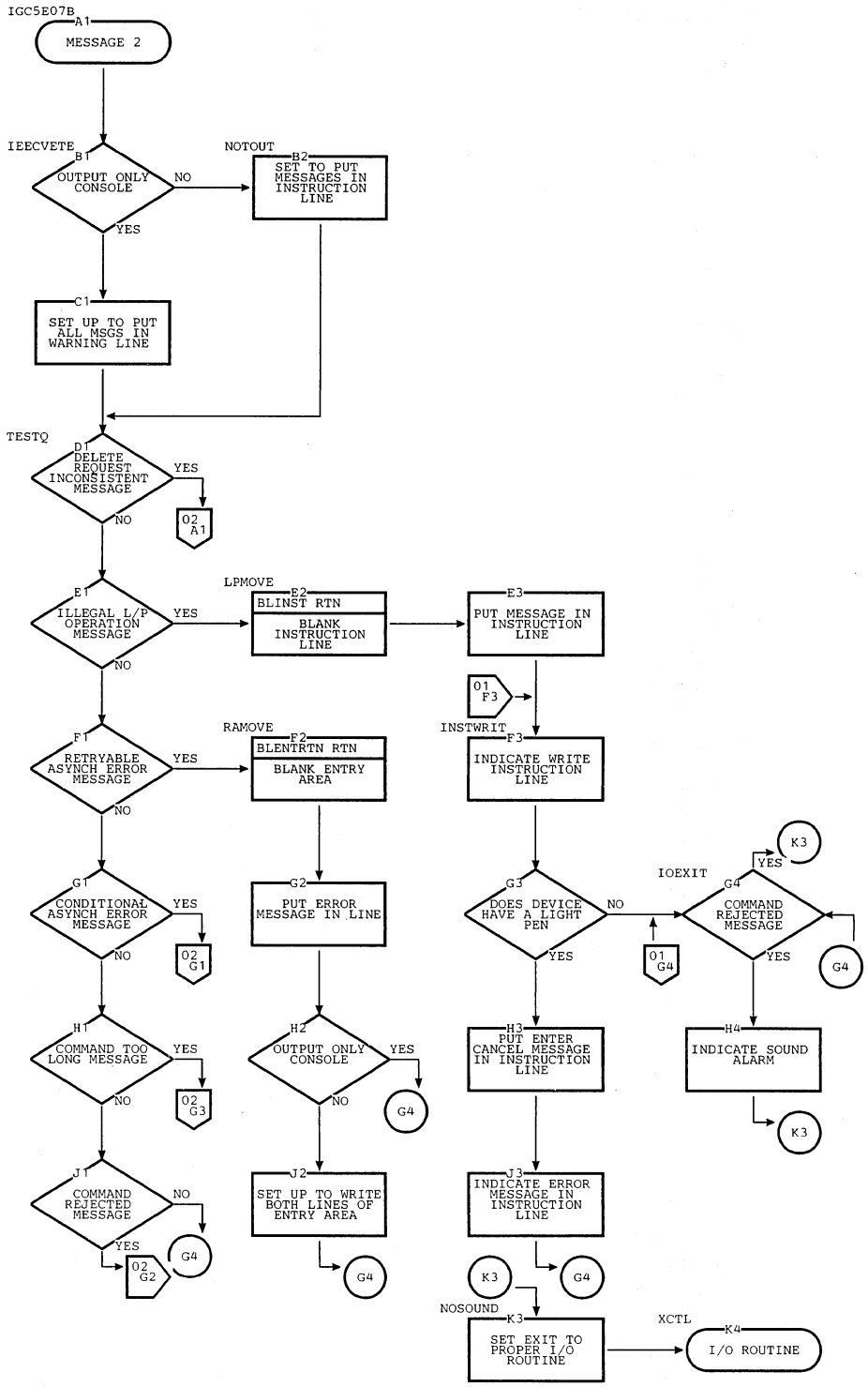


Chart JI. DIDOCS Message 2 (Part 2 of 2)

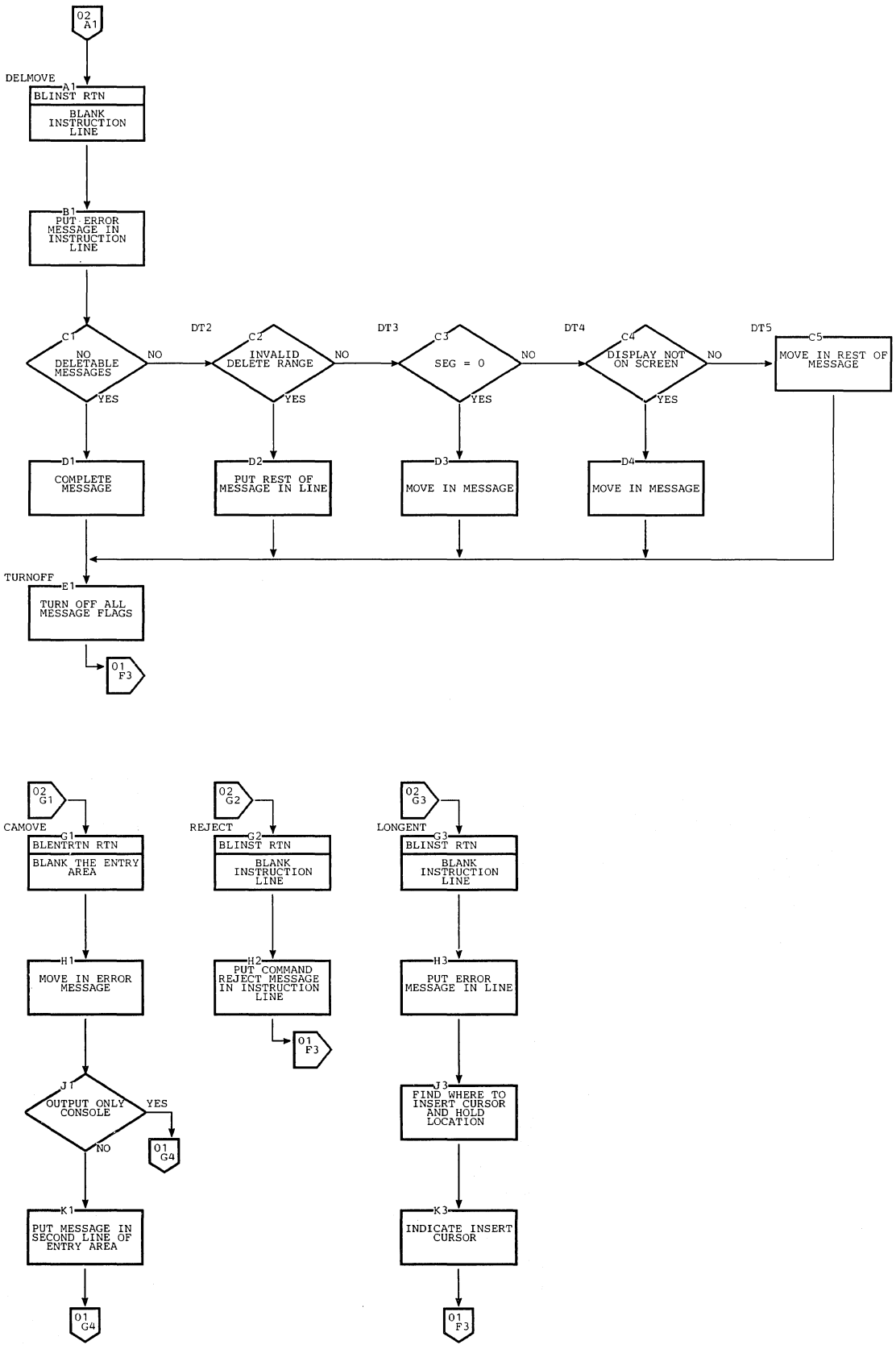


Chart JJ. DIDOCS Message 3

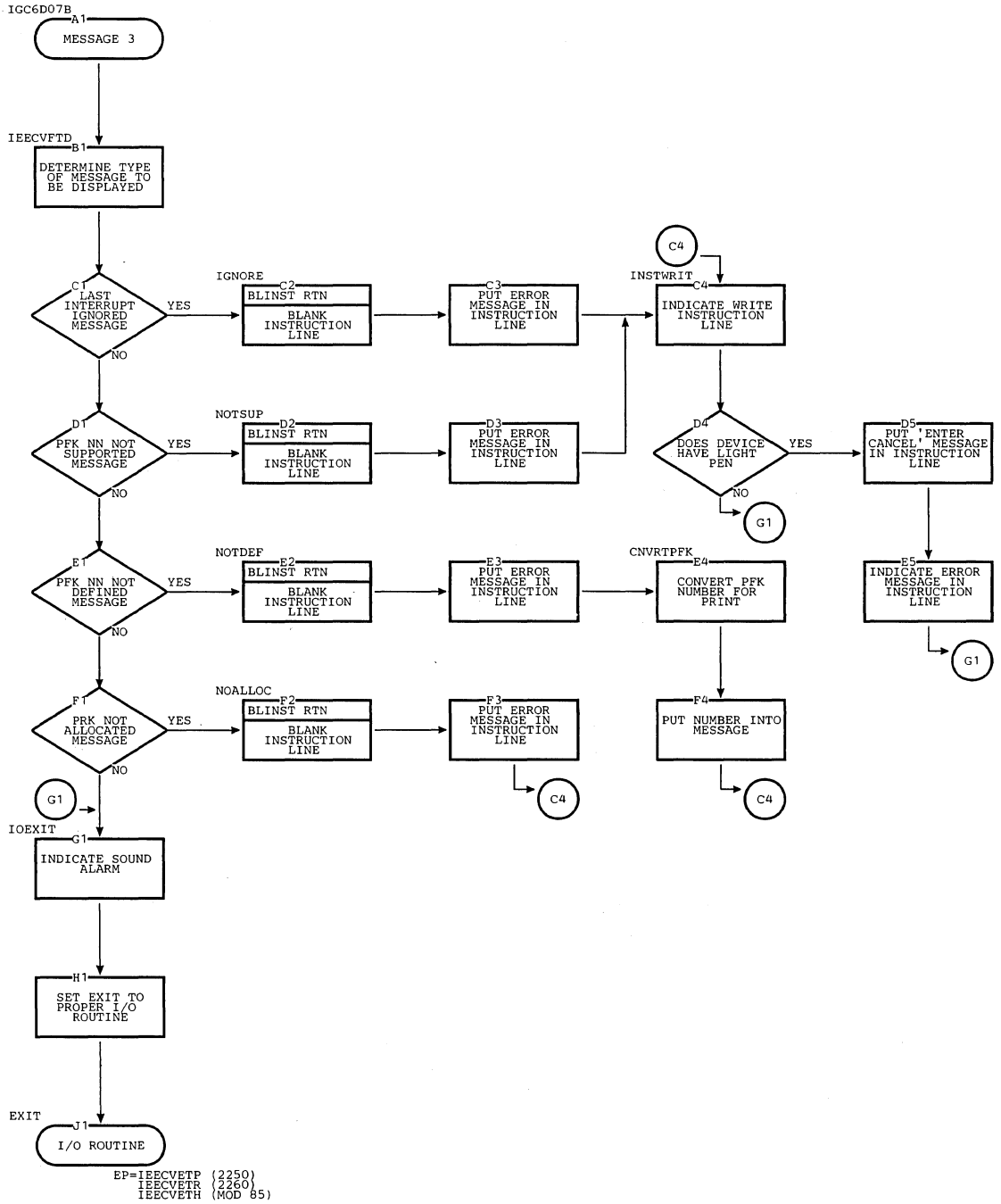


Chart JK. DIDOCS Display 1 (Part 1 of 2)

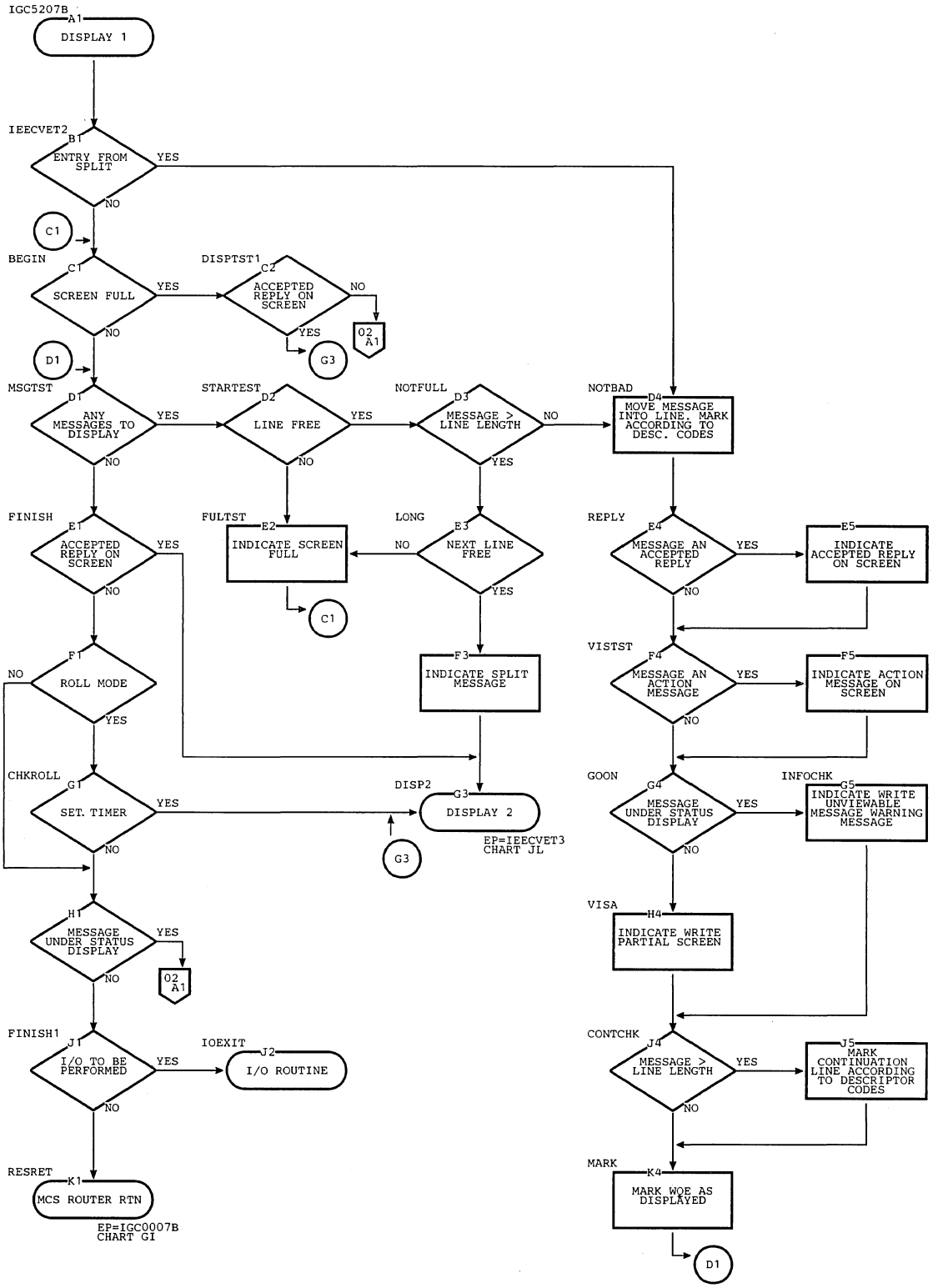


Chart JK. DIDOCS Display 1 (Part 2 of 2)

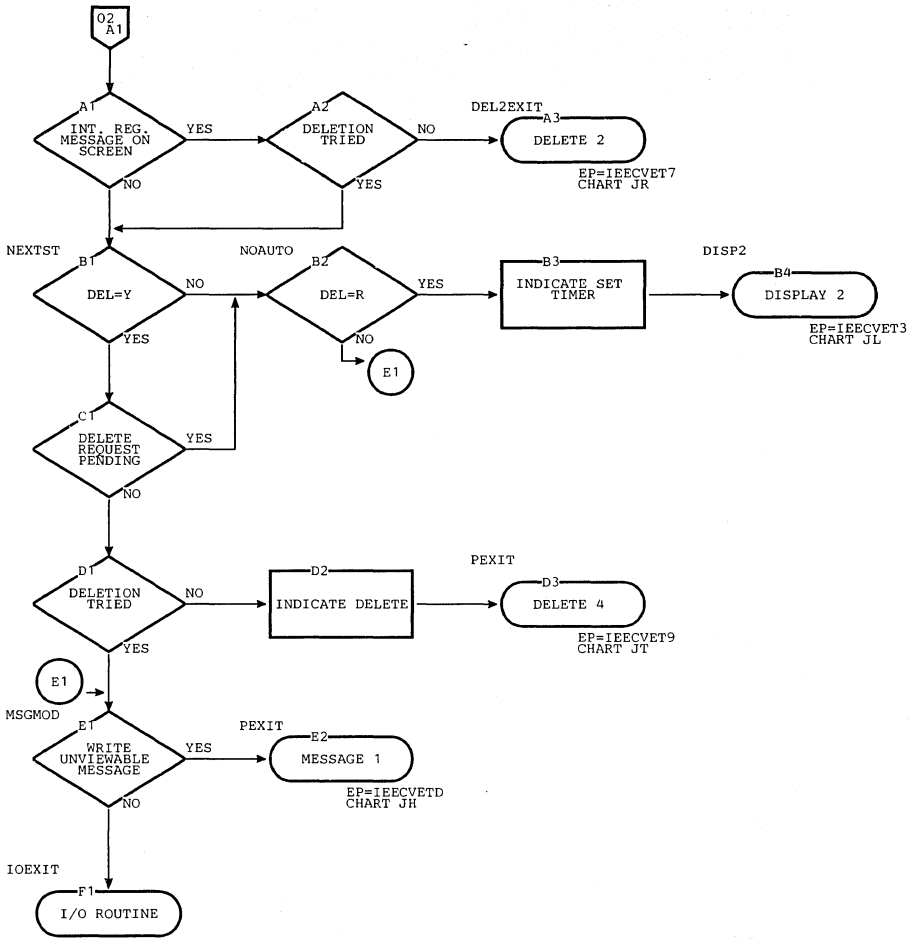


Chart JL. DIDOCS Display 2 (Part 1 of 2)

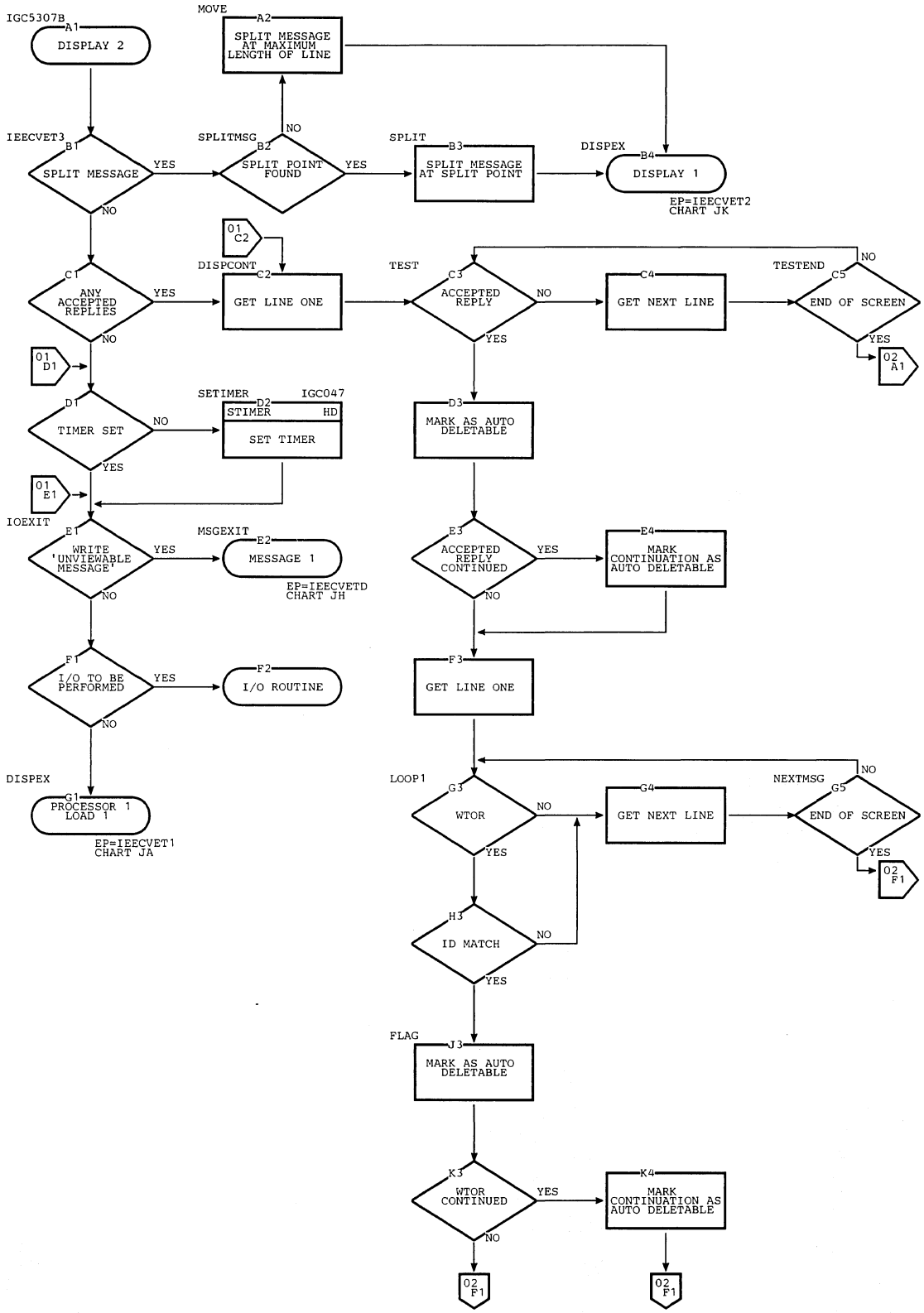
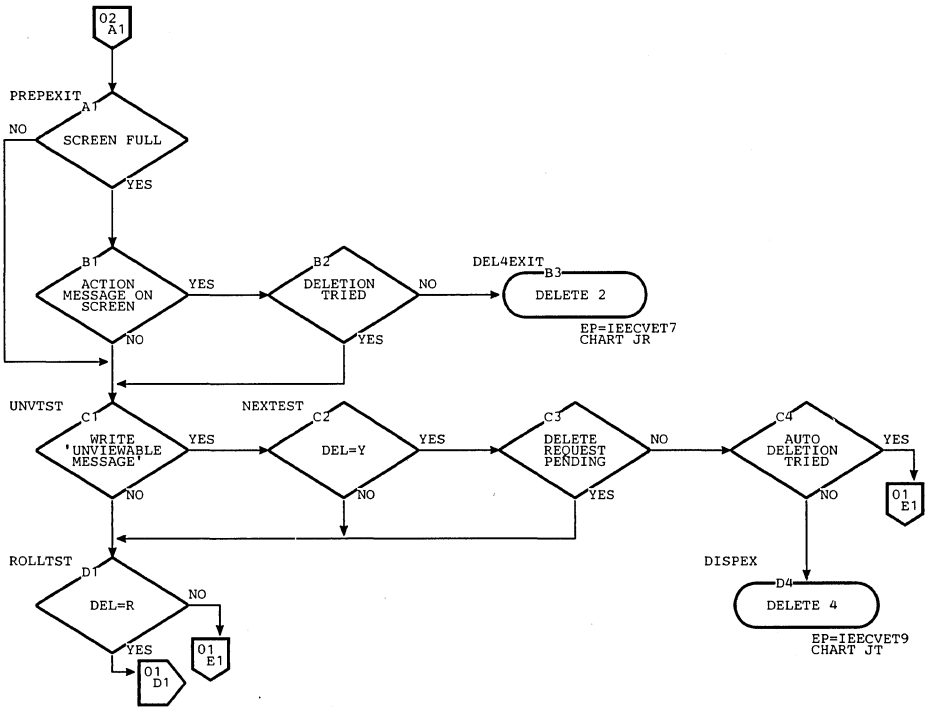


Chart JL. DIDOCS Display 2 (Part 2 of 2)



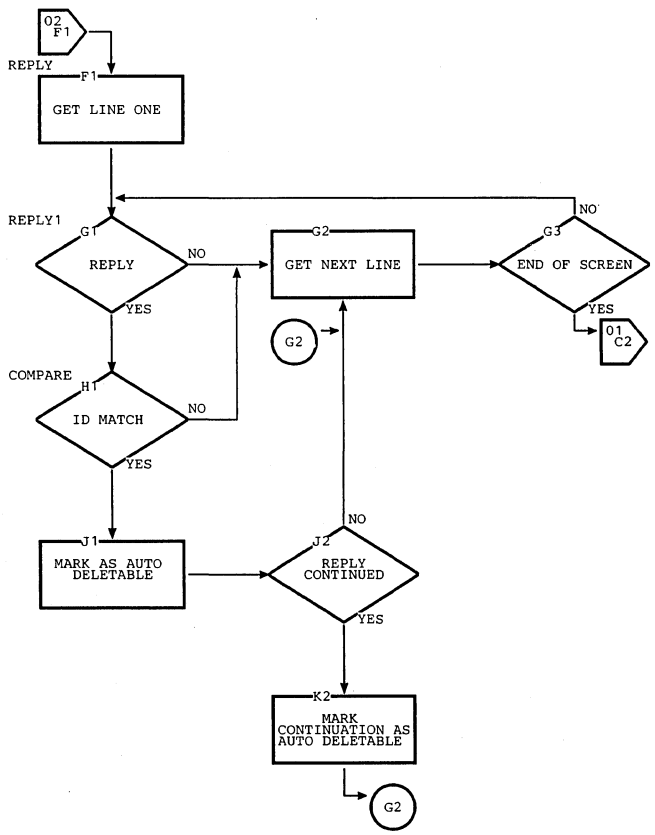


Chart JM. DIDOCS Display 3 (Part 1 of 2)

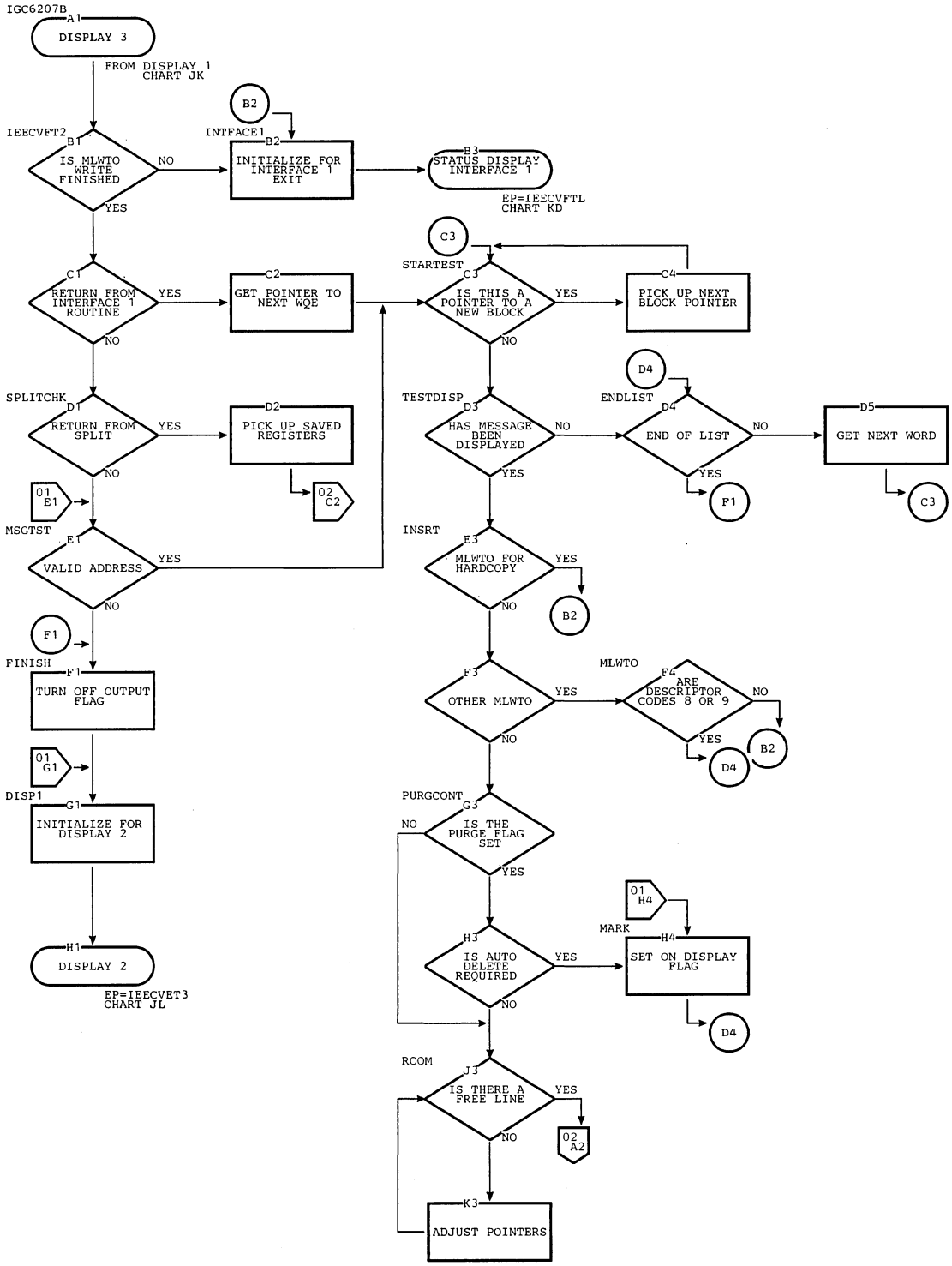


Chart JM. DIDOCS Display 3 (Part 2 of 2)

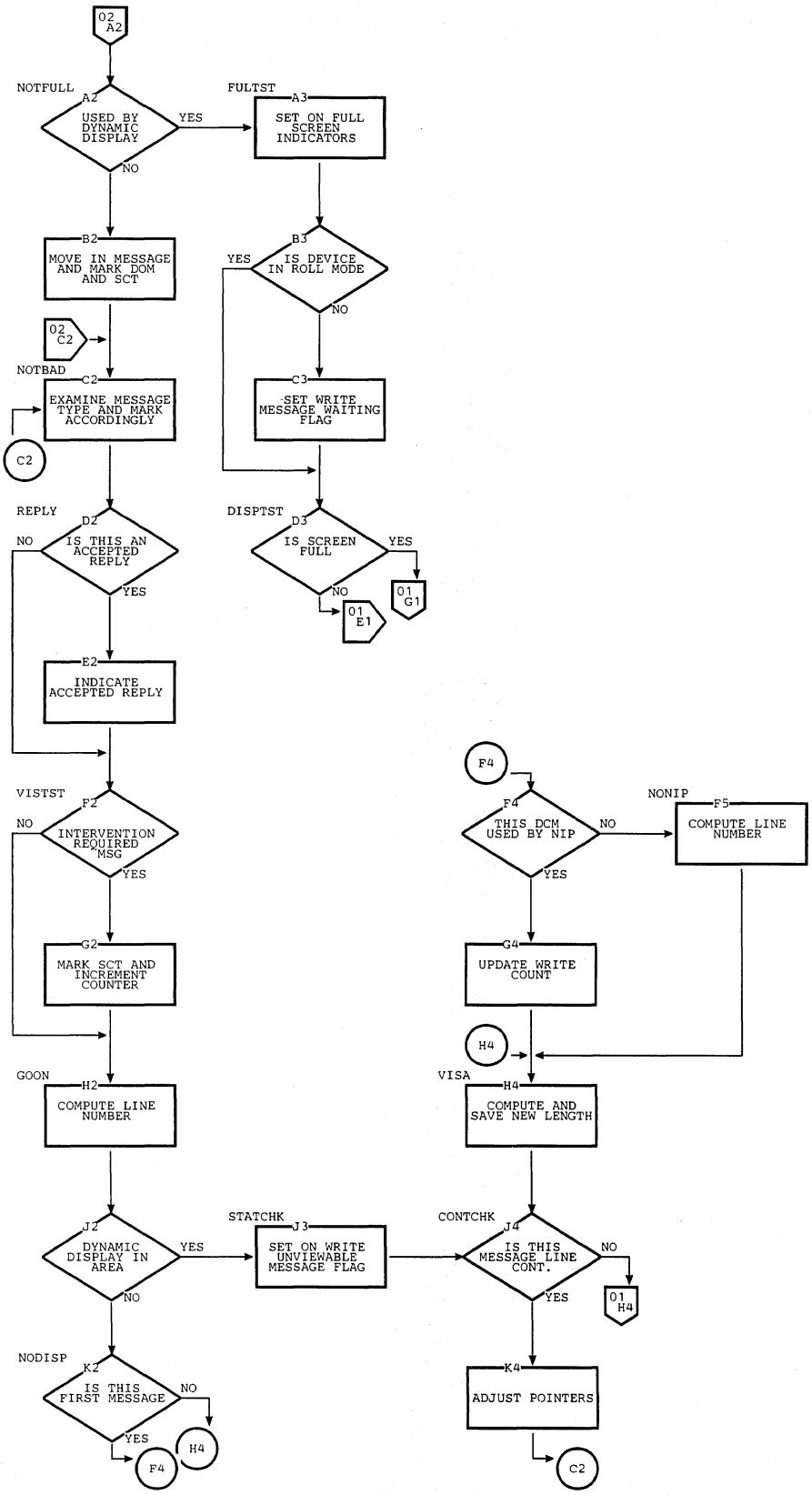


Chart JN. DIDOCS Roll Mode

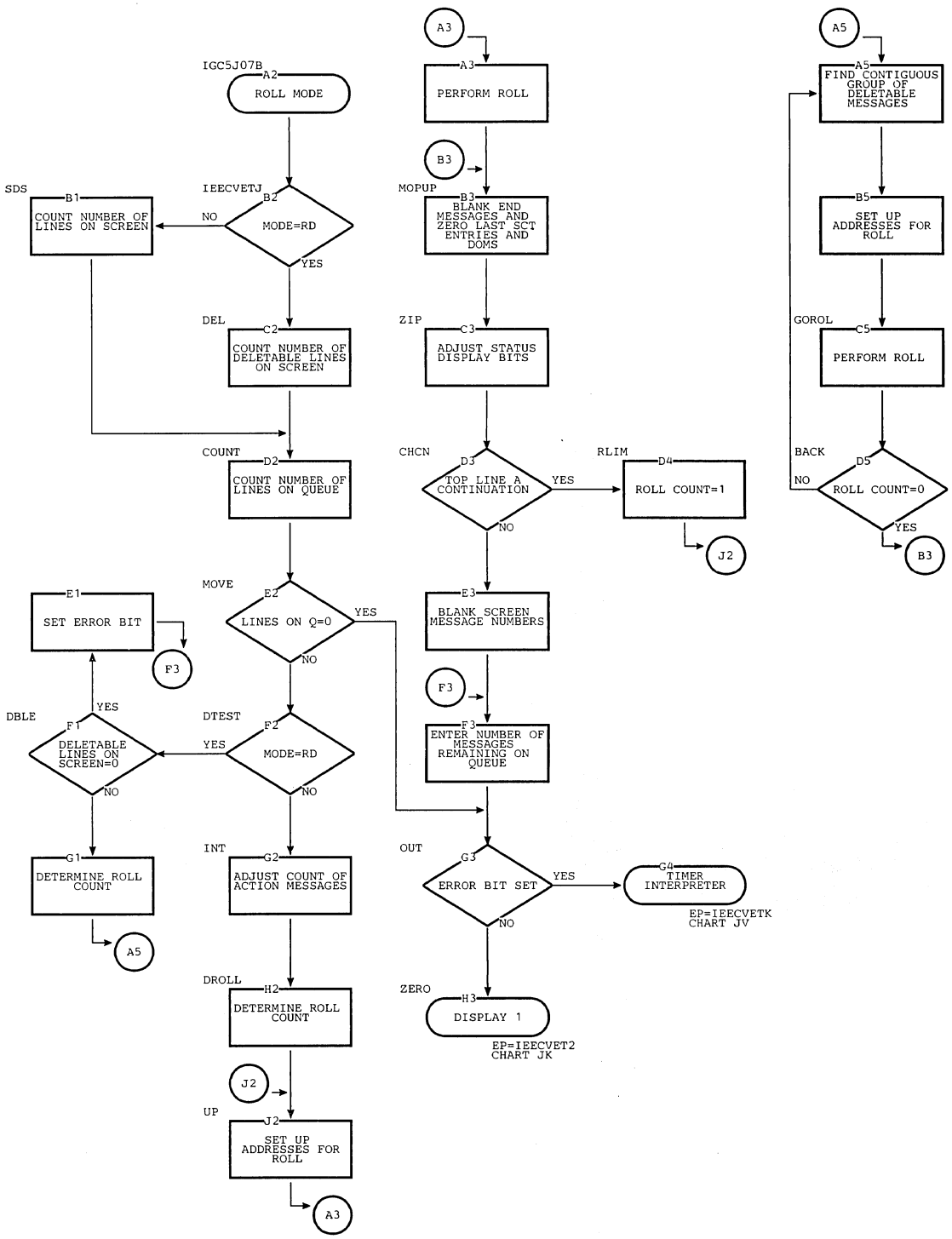


Chart JO. DIDOCS Command

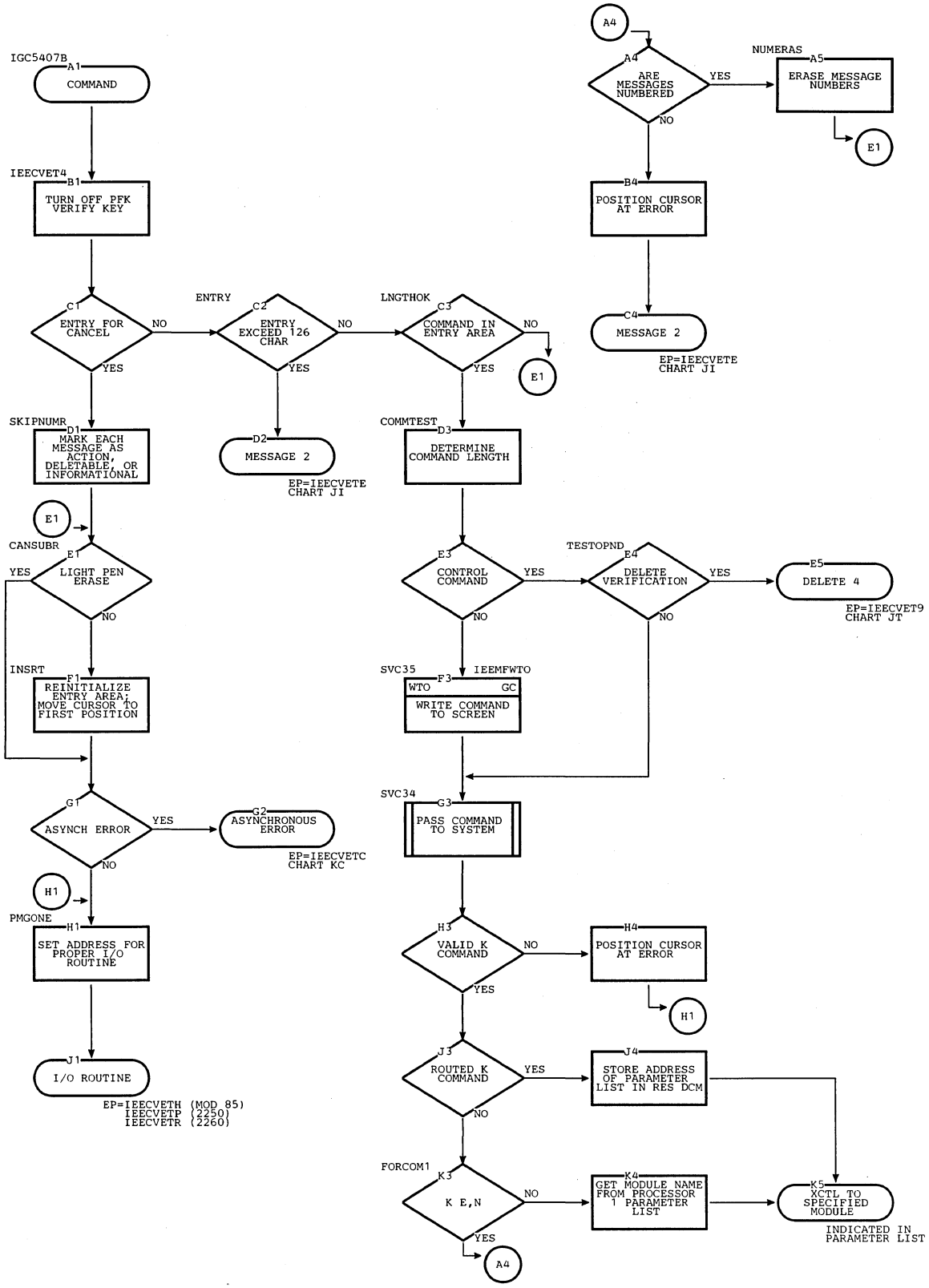


Chart JP. DIDOCS Options (Part 1 of 2)

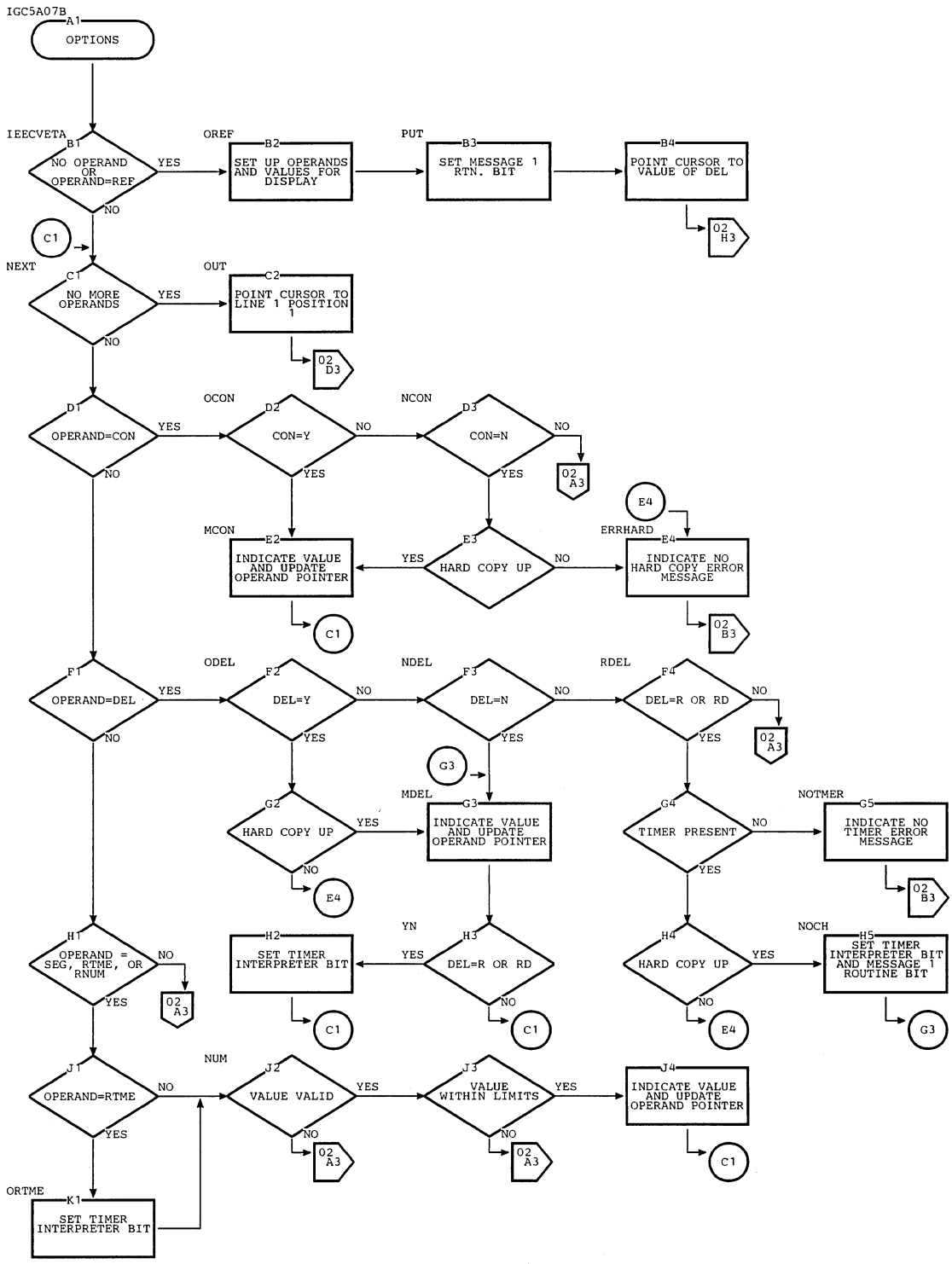


Chart JP. DIDOCS Options (Part 2 of 2)

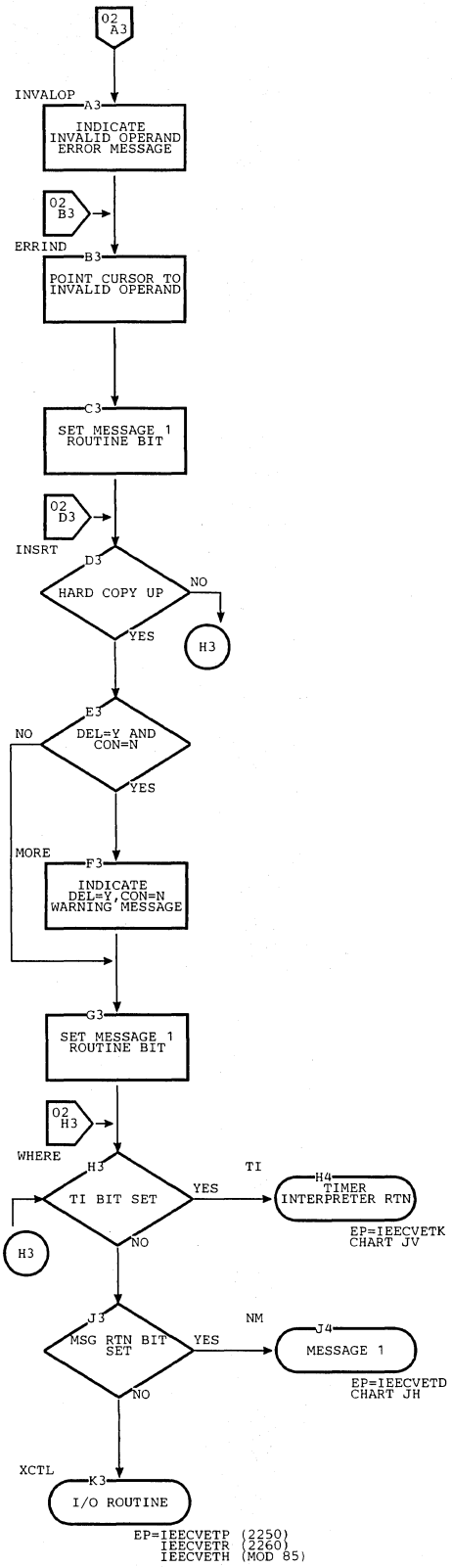


Chart JQ. DIDOCS Delete 1 (Part 1 of 2)

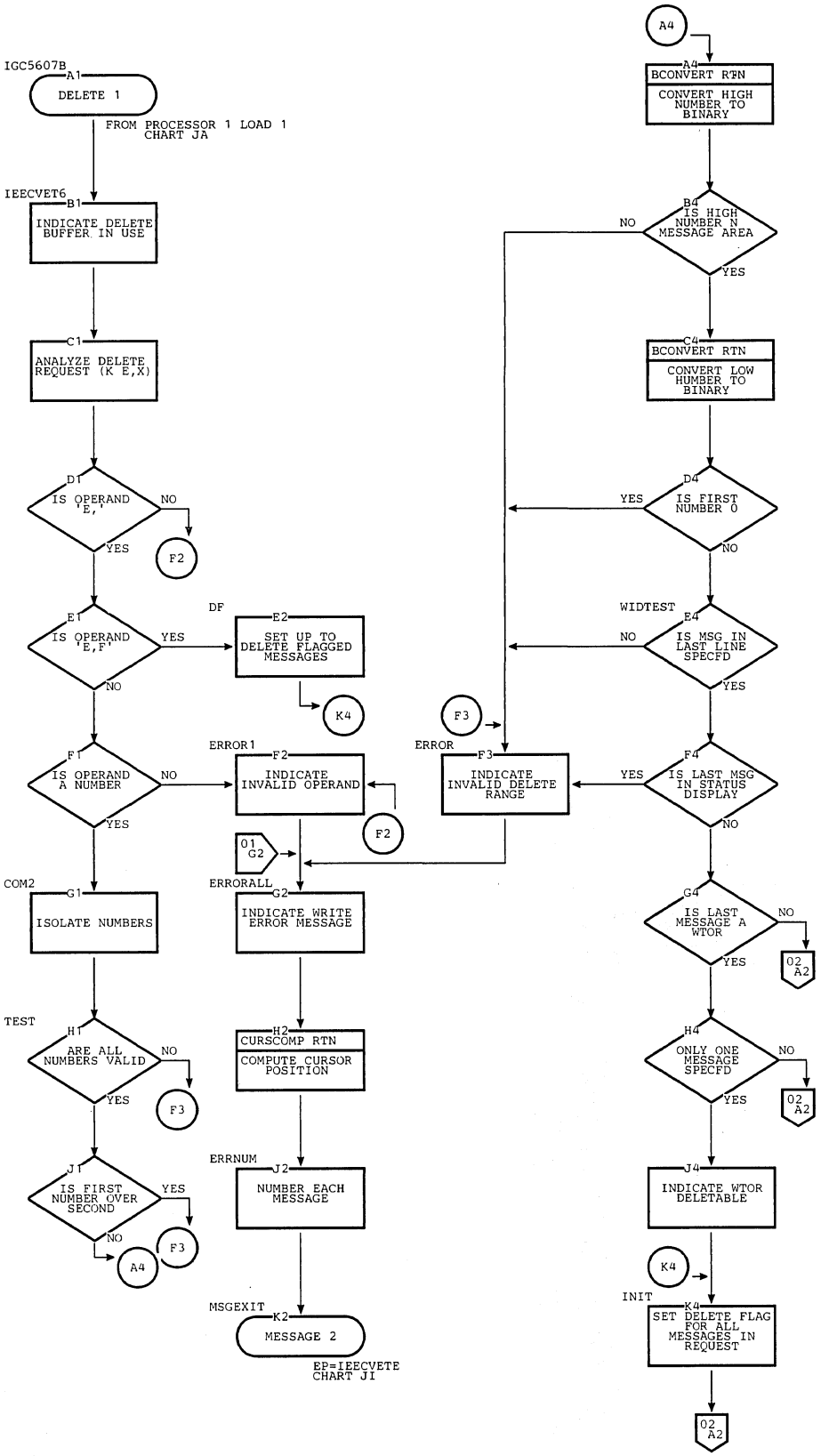


Chart JQ. DIDOCS Delete 1 (Part 2 of 2)

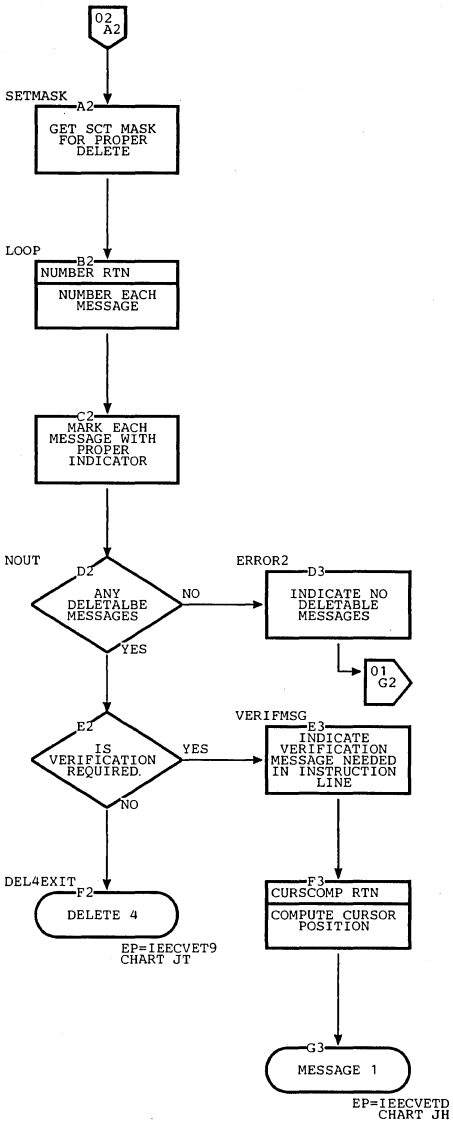


Chart JR. DIDOCS Delete 2

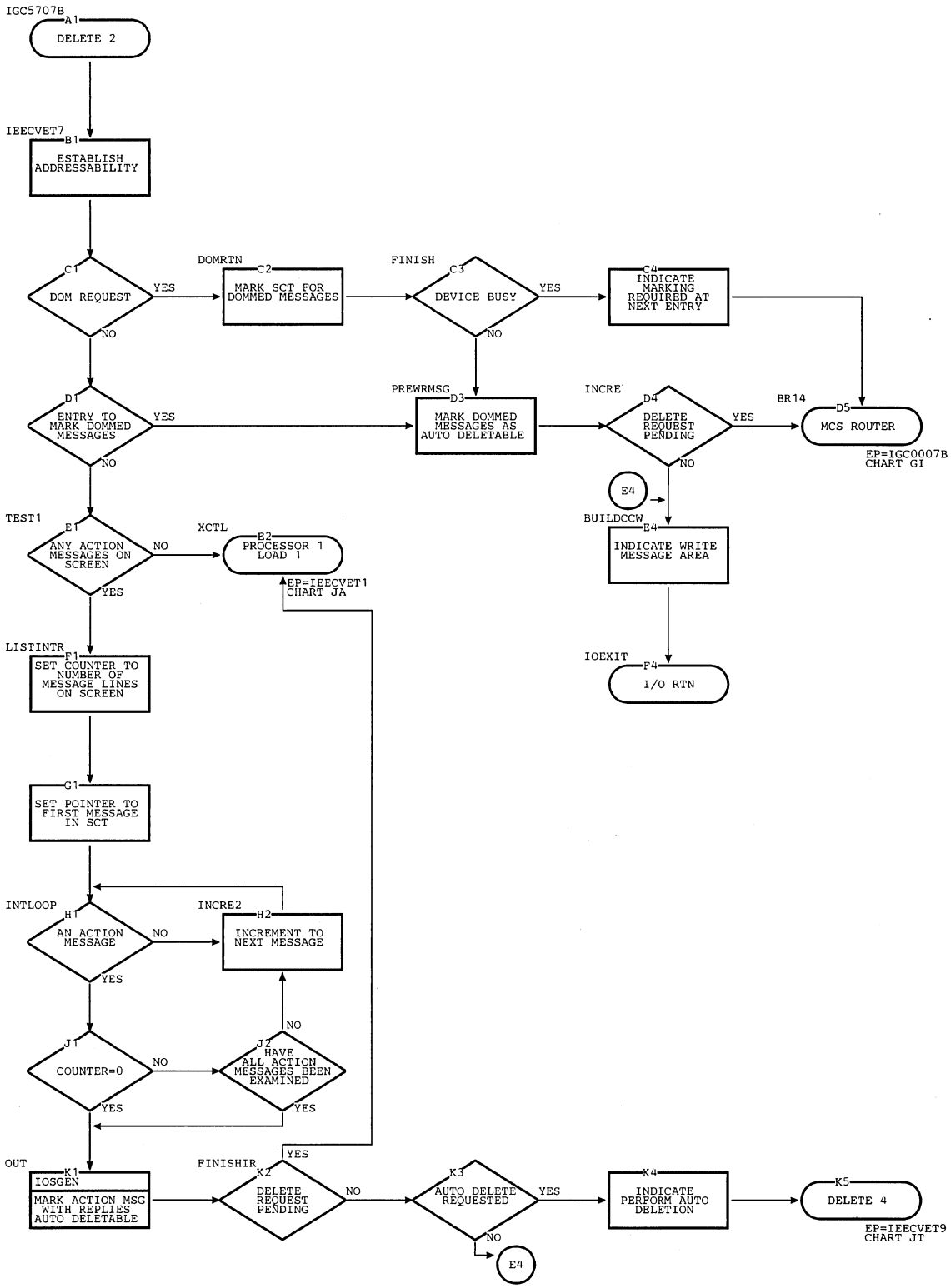


Chart JS. DIDOCS Delete 3 (Part 1 of 2)

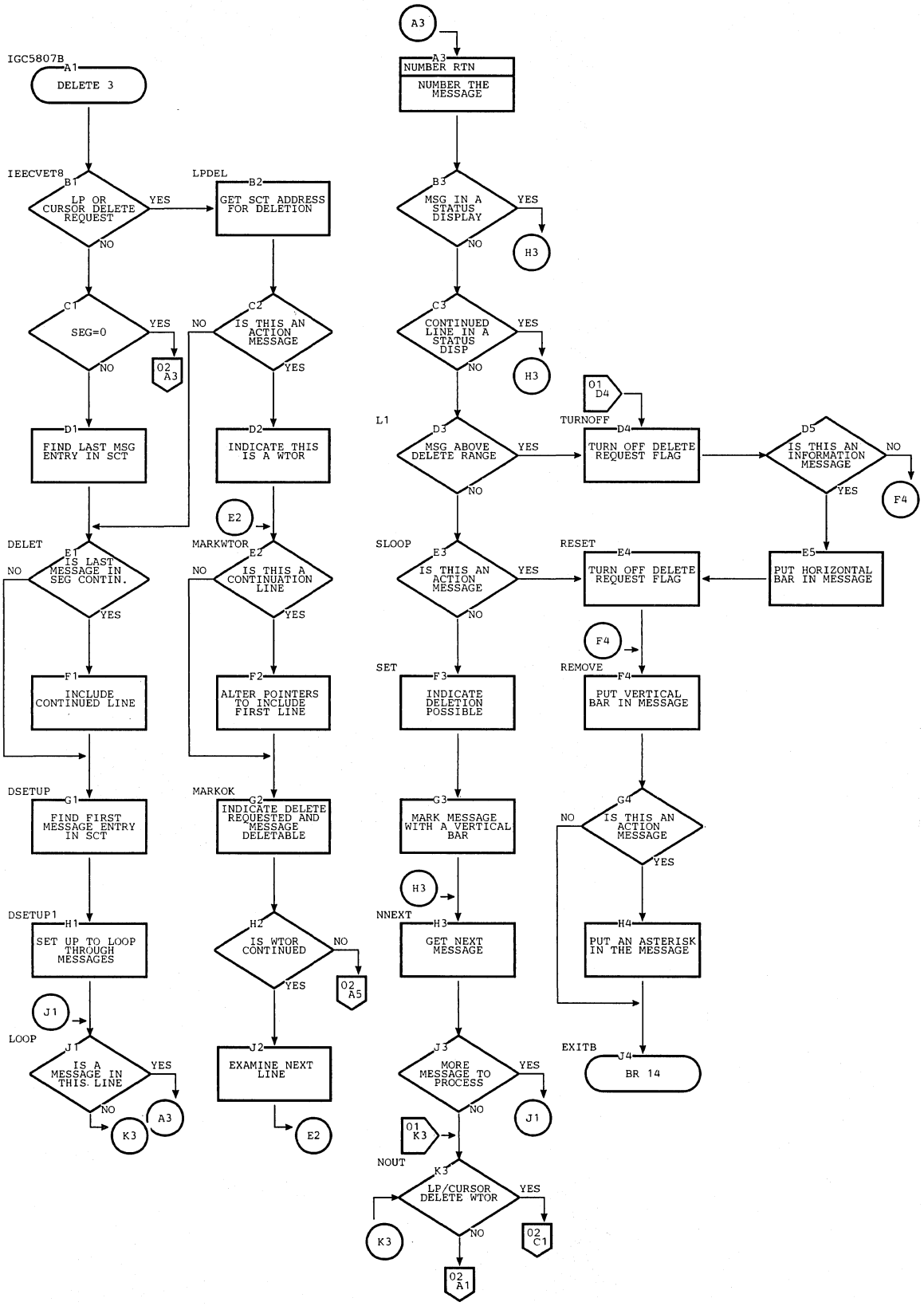


Chart JS. DIDOCS Delete 3 (Part 2 of 2)

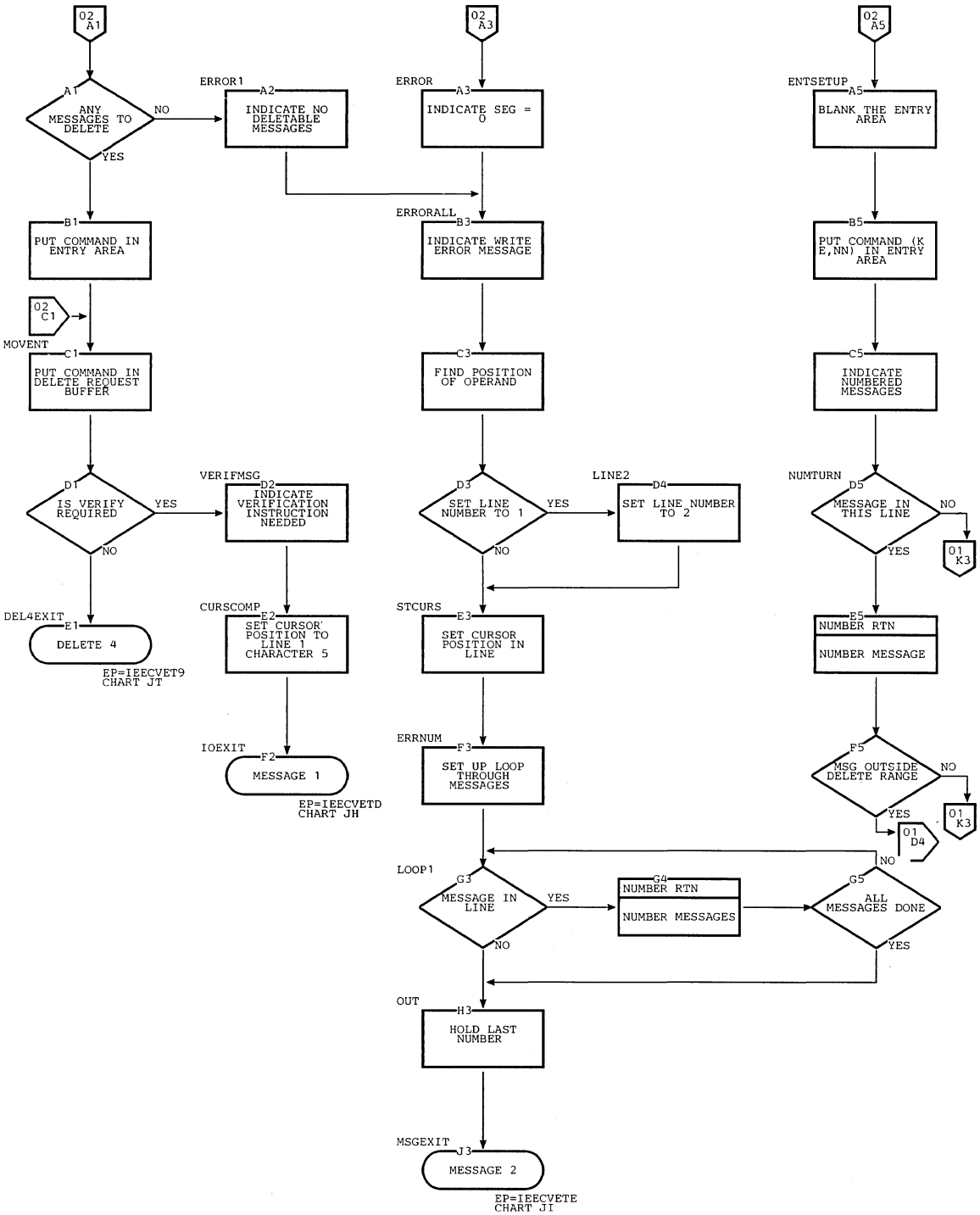


Chart JT. DIDOCS Delete 4

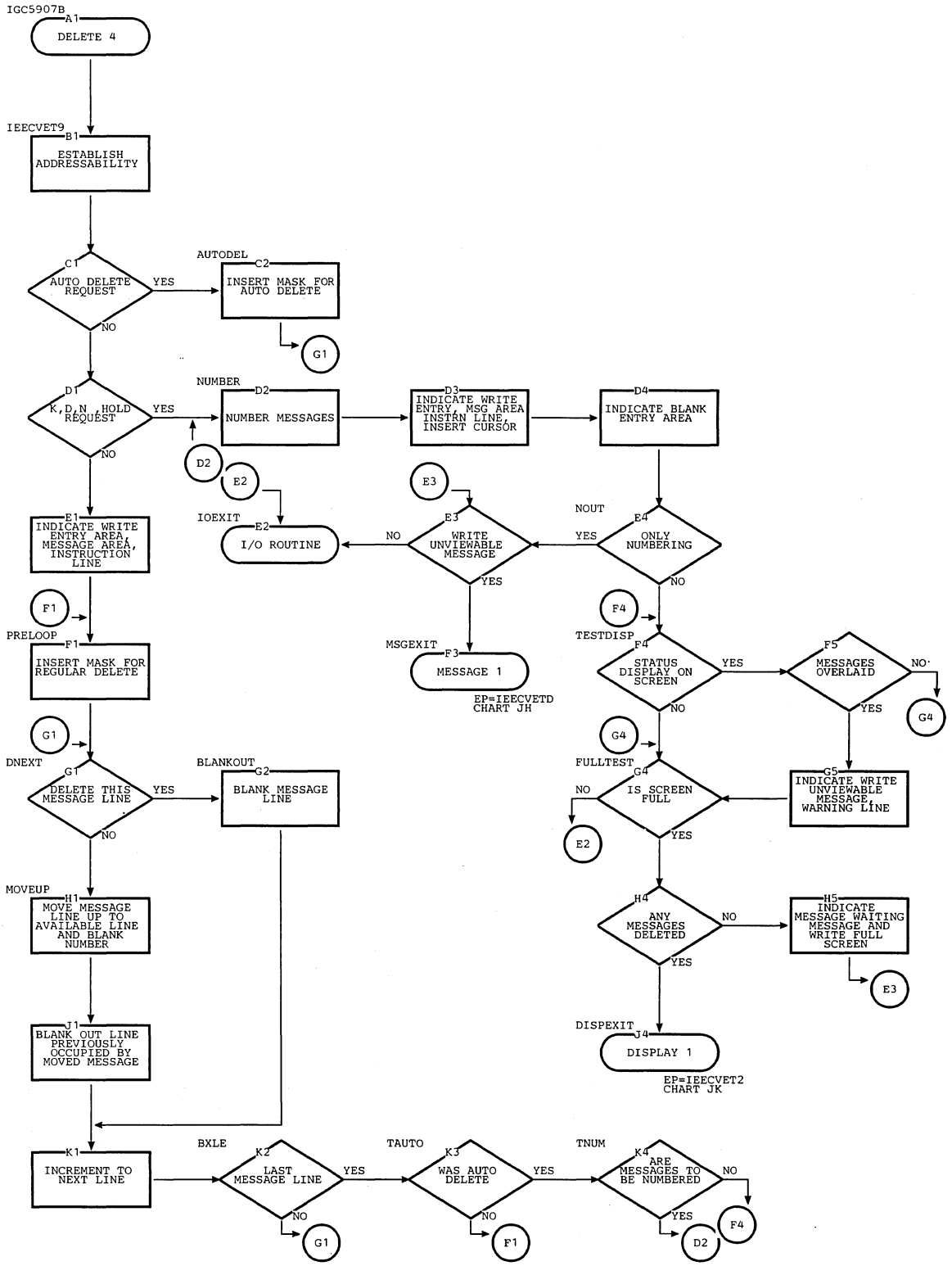


Chart JU. DIDOCS Light Pen/Cursor

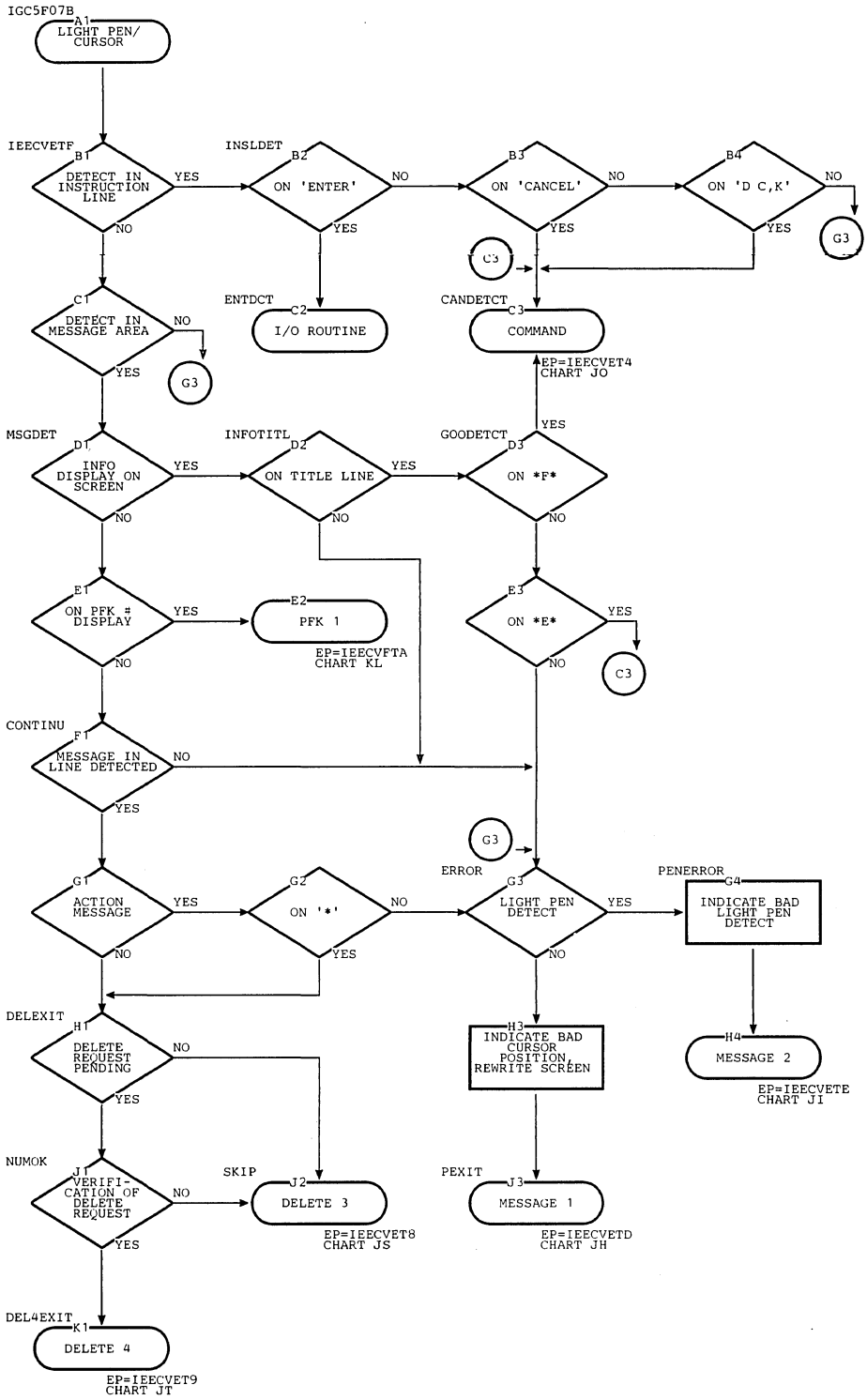


Chart JV. DIDOCS Timer/Interpreter (Part 1 of 2)

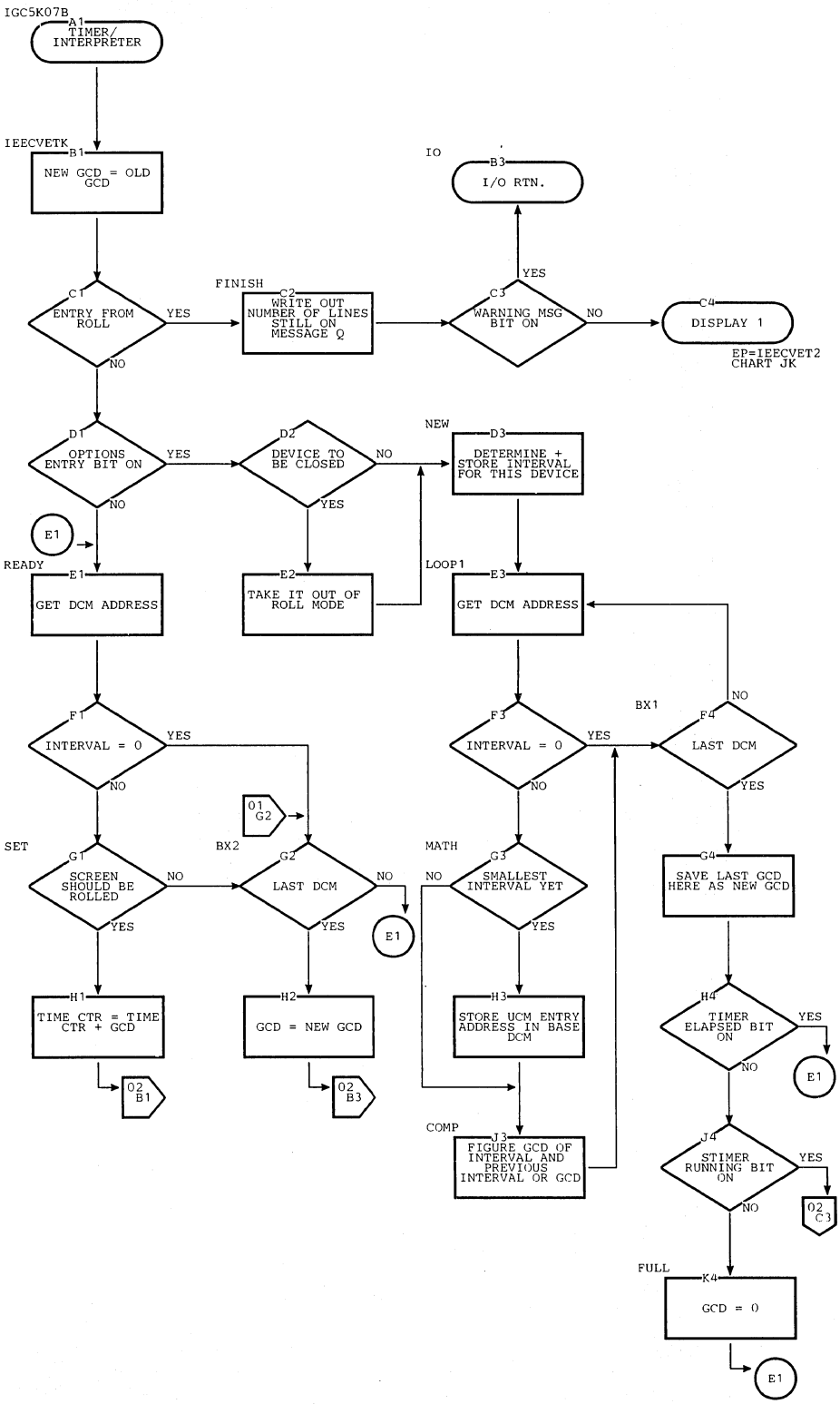


Chart JV. DIDOCS Timer/Interpreter (Part 2 of 2)

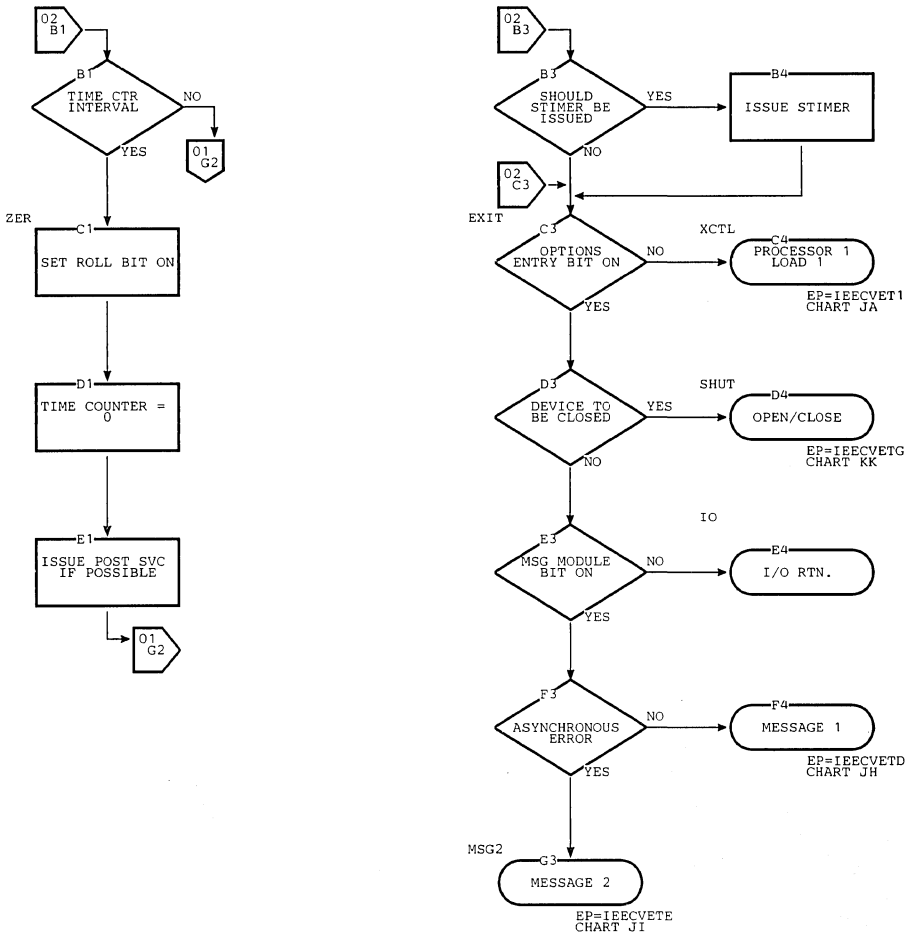


Chart KA. DIDOCS Processor 0 Load 1 (Part 1 of 2)

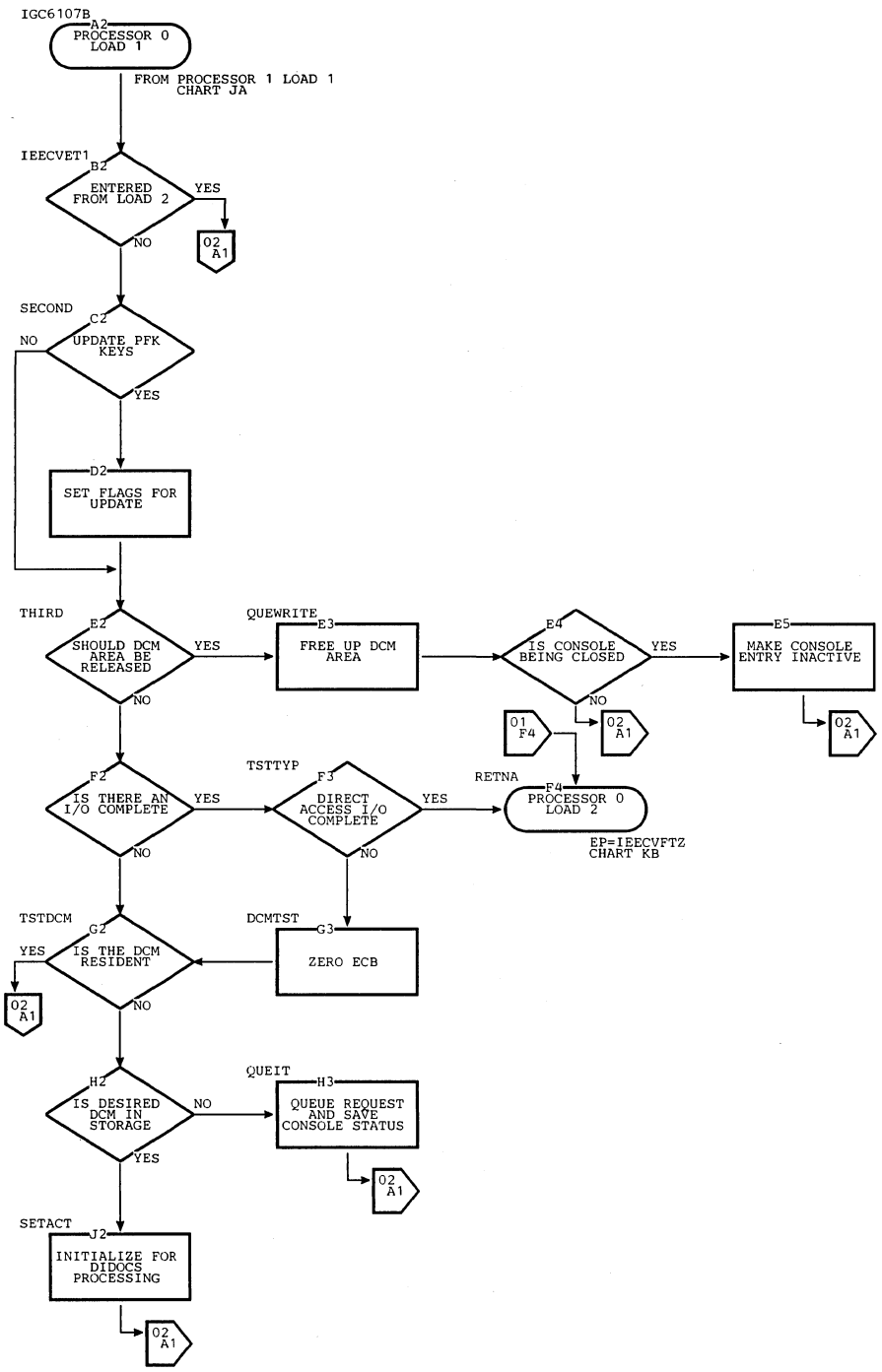


Chart KA. DIDOCS Processor 0 Load 1 (Part 2 of 2)

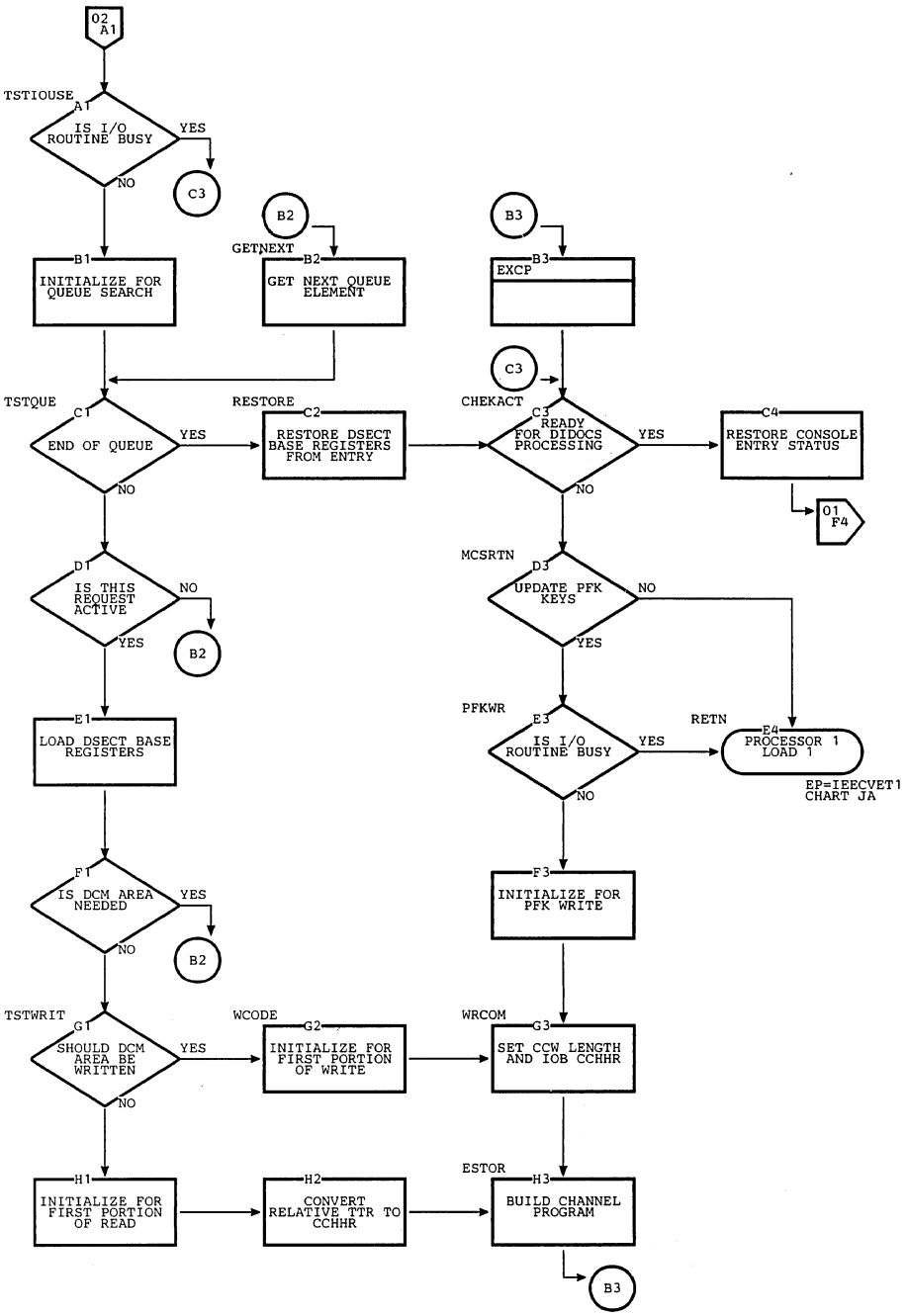


Chart KB. DIDOCS Processor 0 Load 2 (Part 1 of 2)

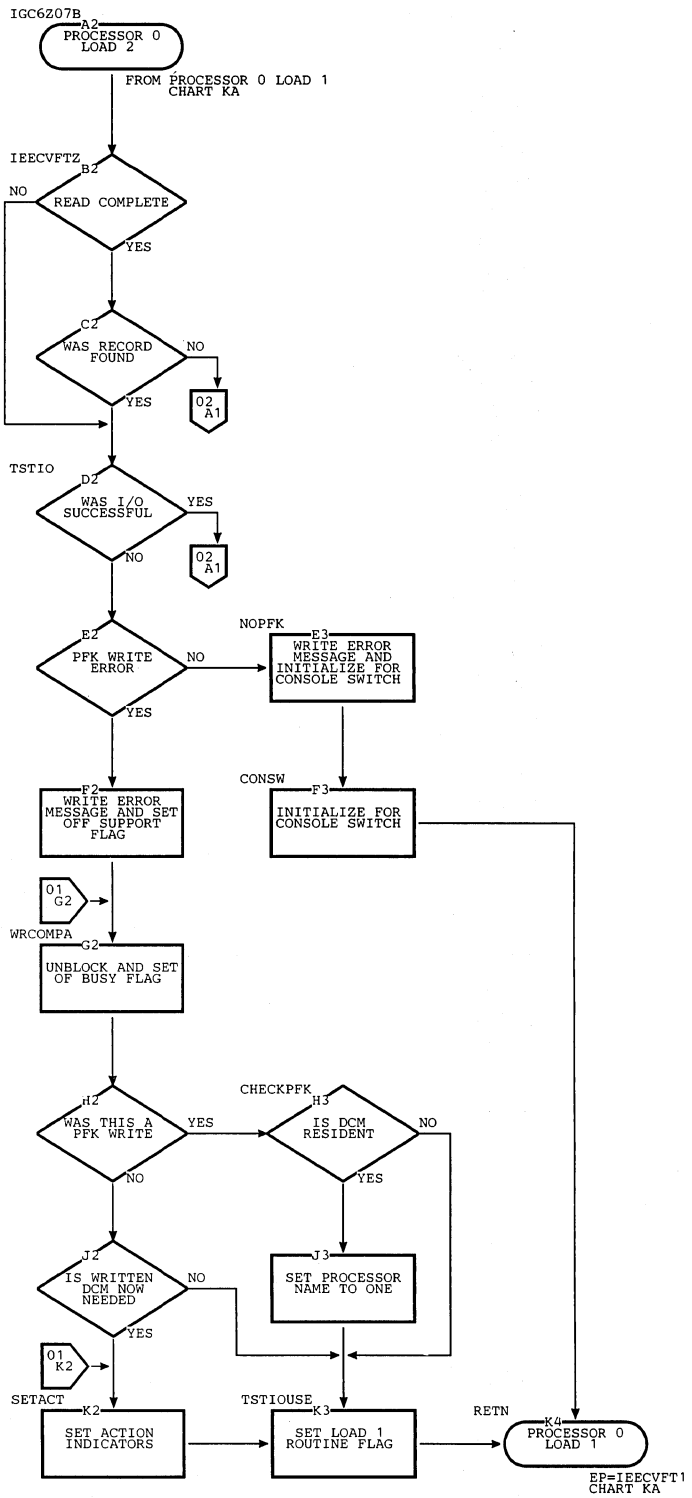


Chart KB. DIDOCS Processor 0 Load 2 (Part 2 of 2)

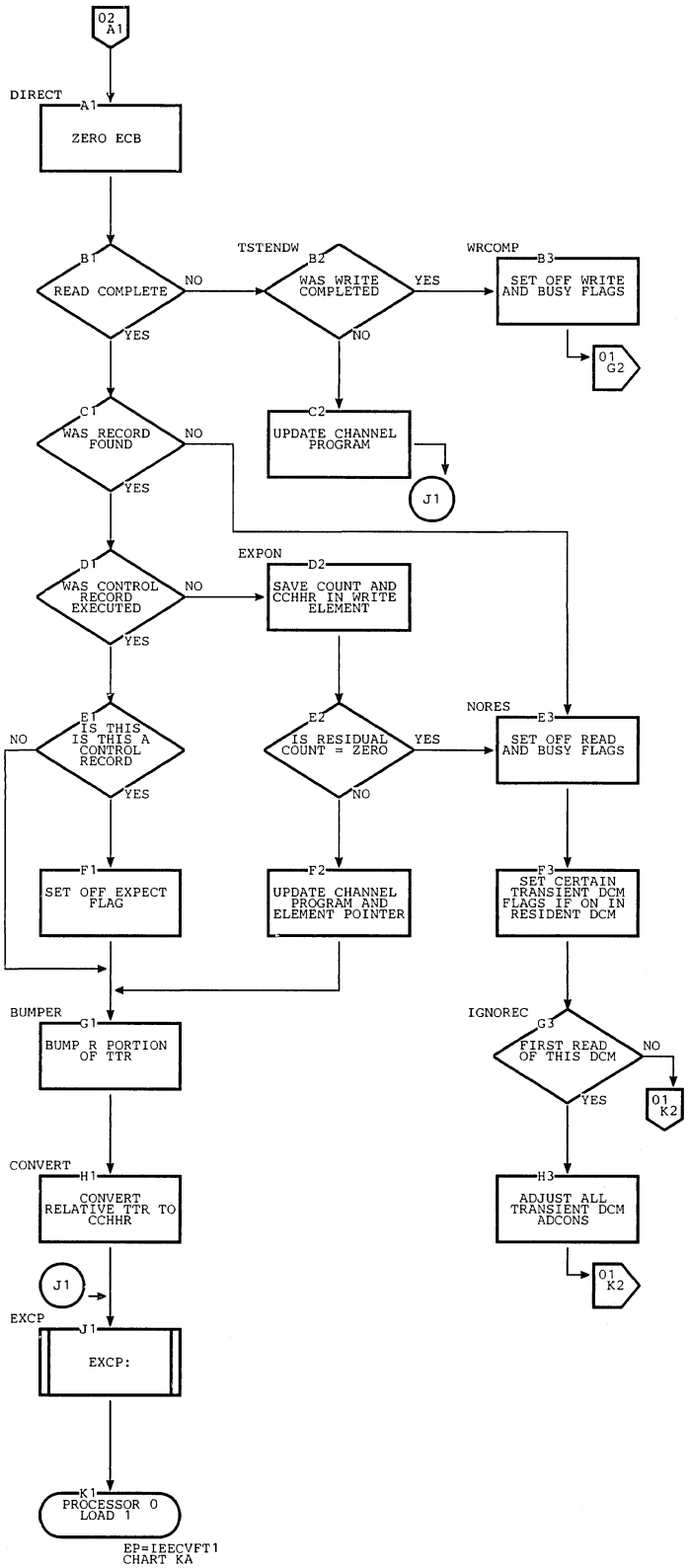


Chart KC. DIDOCS Asynchronous Error (Part 1 of 2)

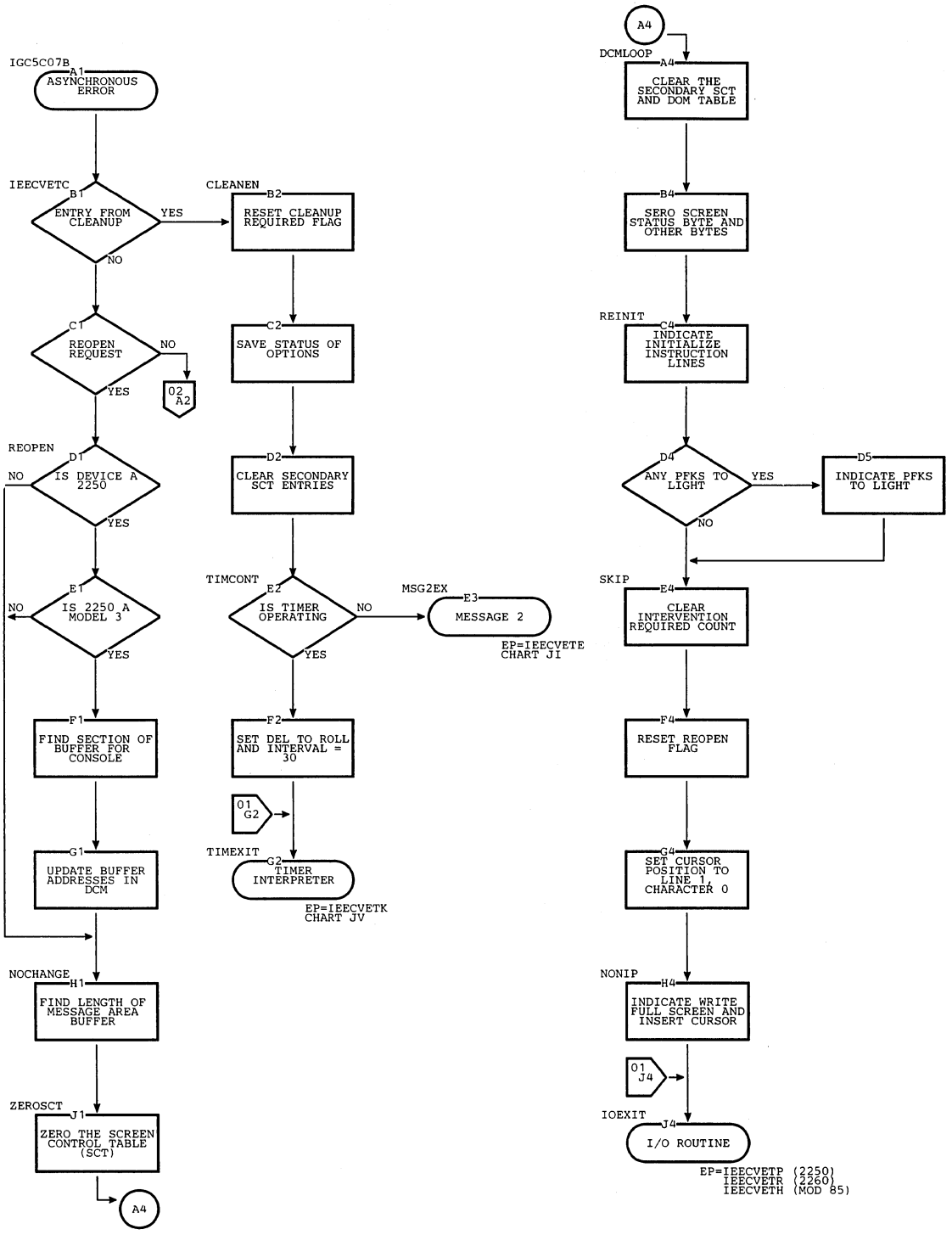


Chart KC. DIDOCS Asynchronous Error (Part 2 of 2)

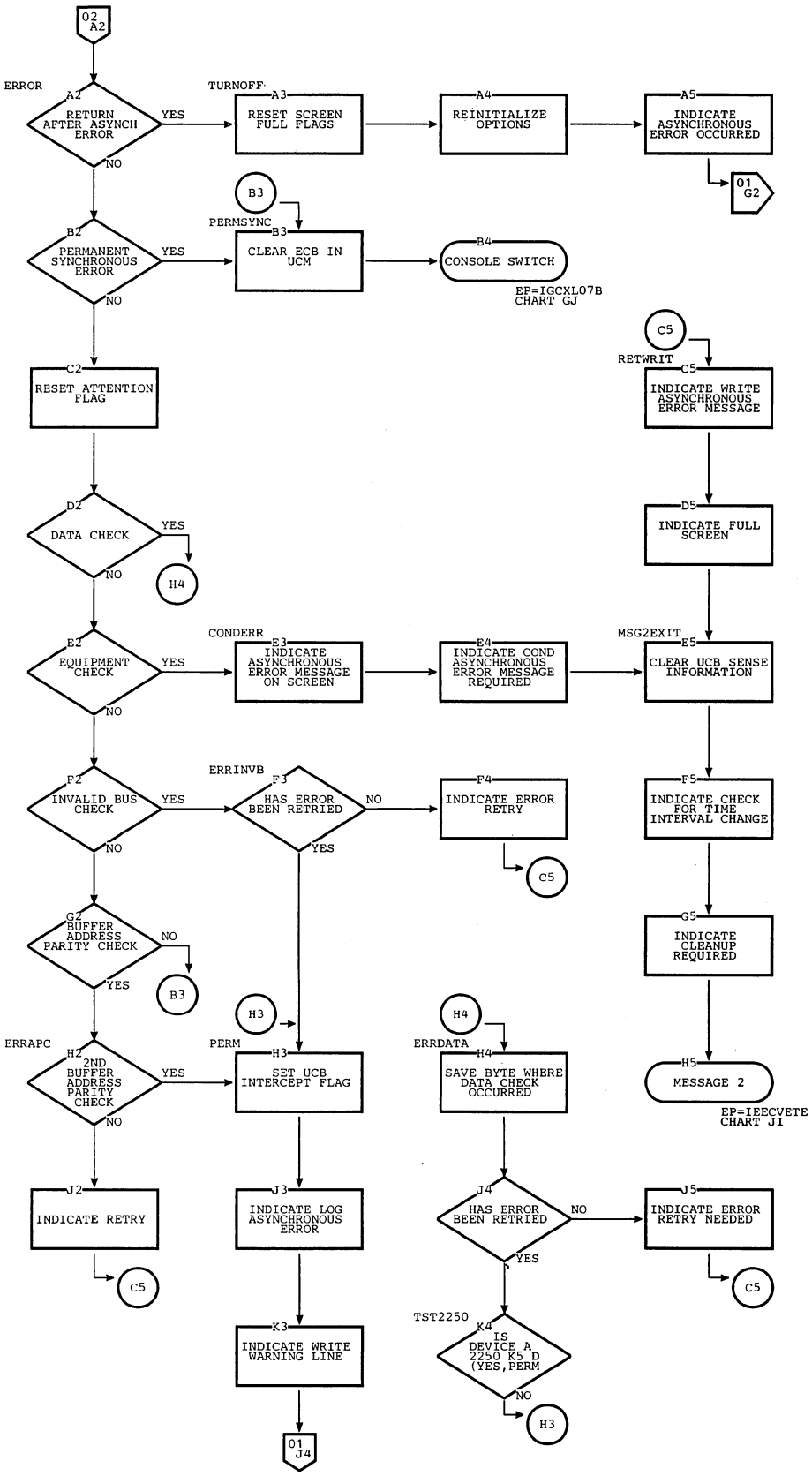


Chart KD. DIDOCS Status Display 1

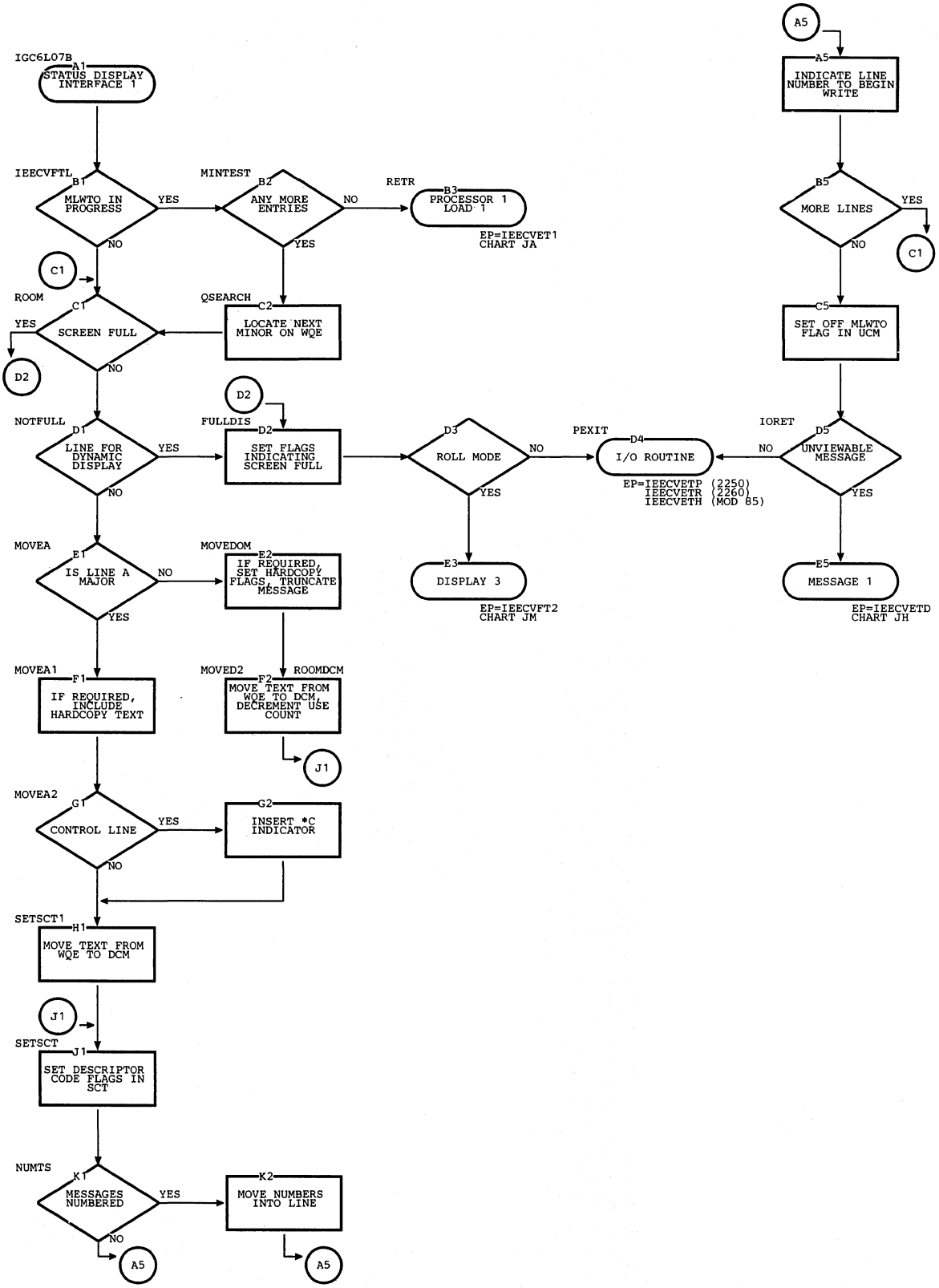


Chart KE. DIDOCS Status Display 2 (Part 1 of 2)

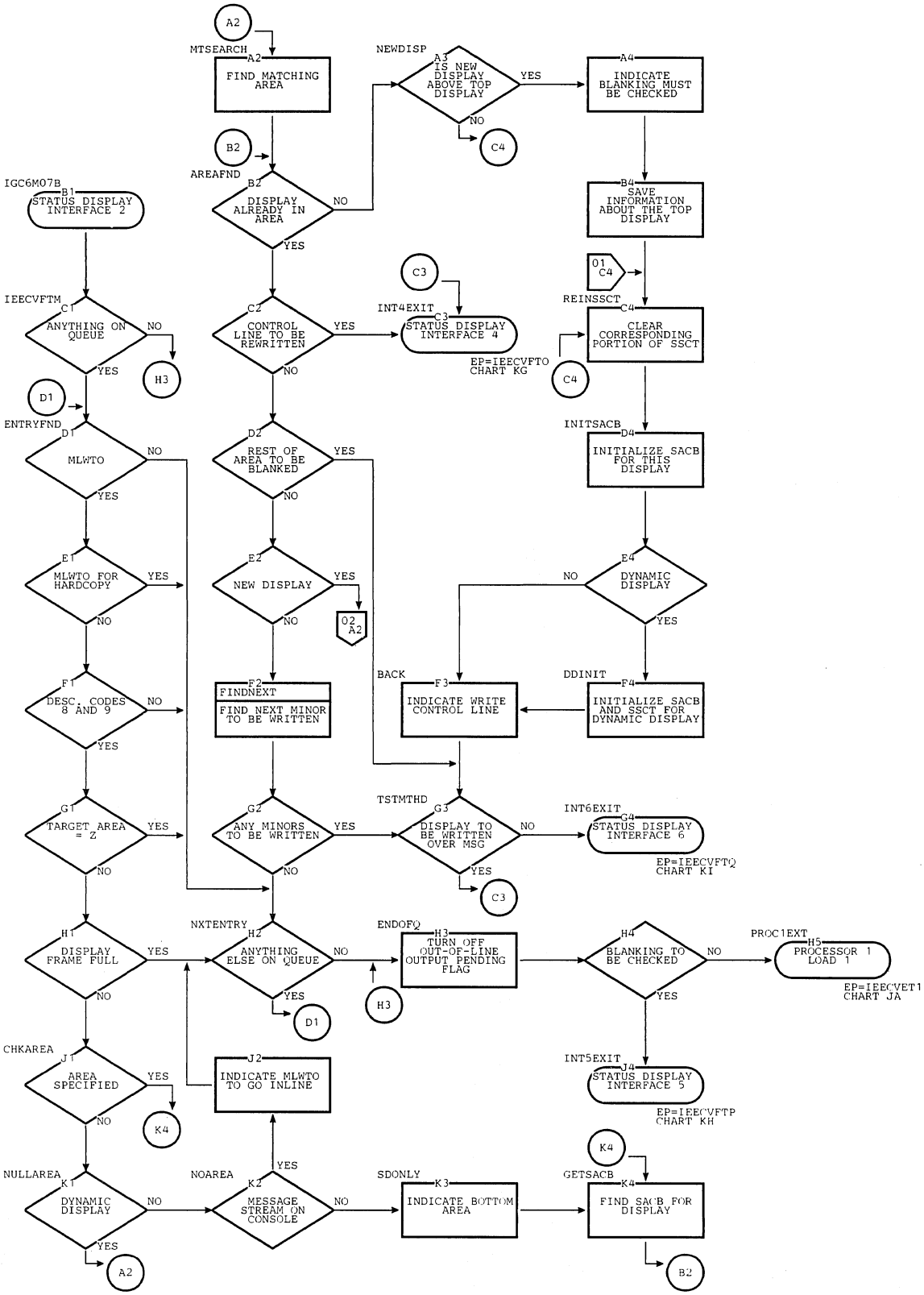


Chart KE. DIDOCS Status Display 2 (Part 2 of 2)

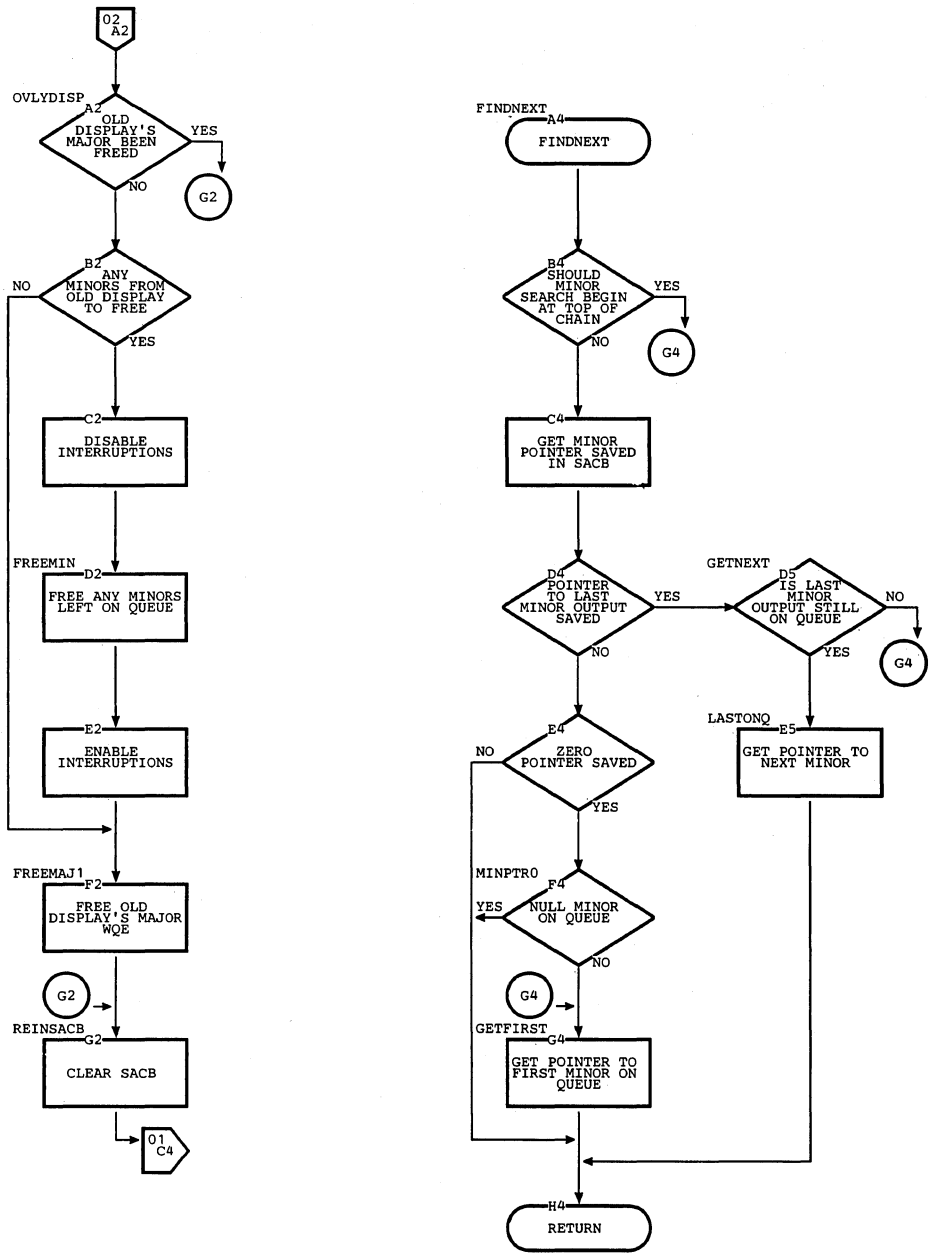


Chart KF. DIDOCS Status Display 3 (Part 1 of 2)

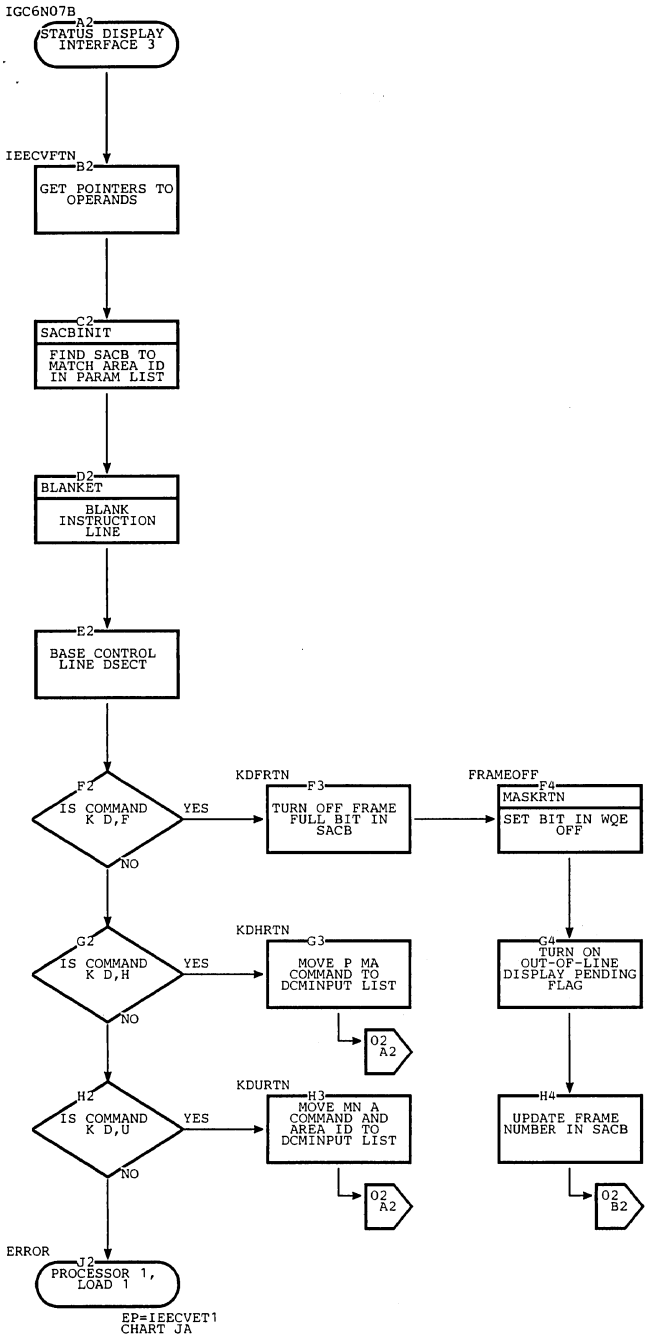


Chart KF. DIDOCS Status Display 3 (Part 2 of 2)

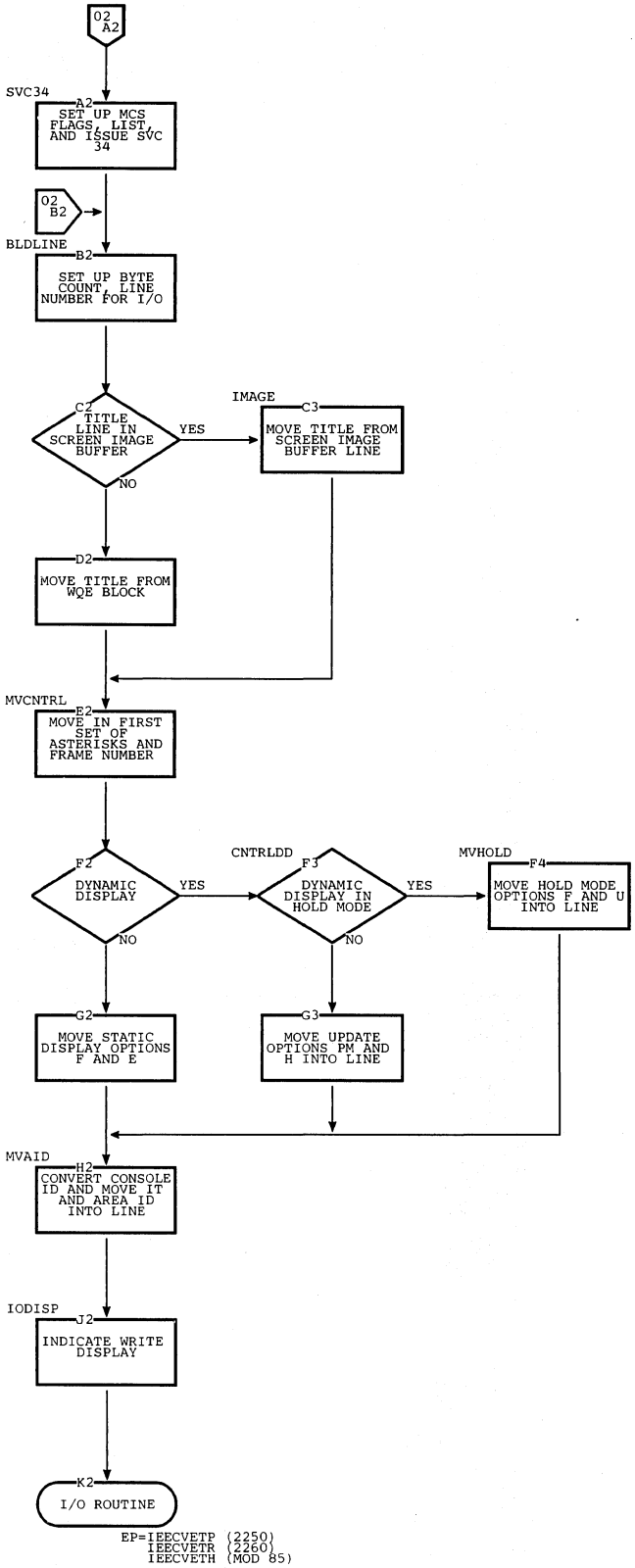


Chart KG. DIDOCS Status Display 4

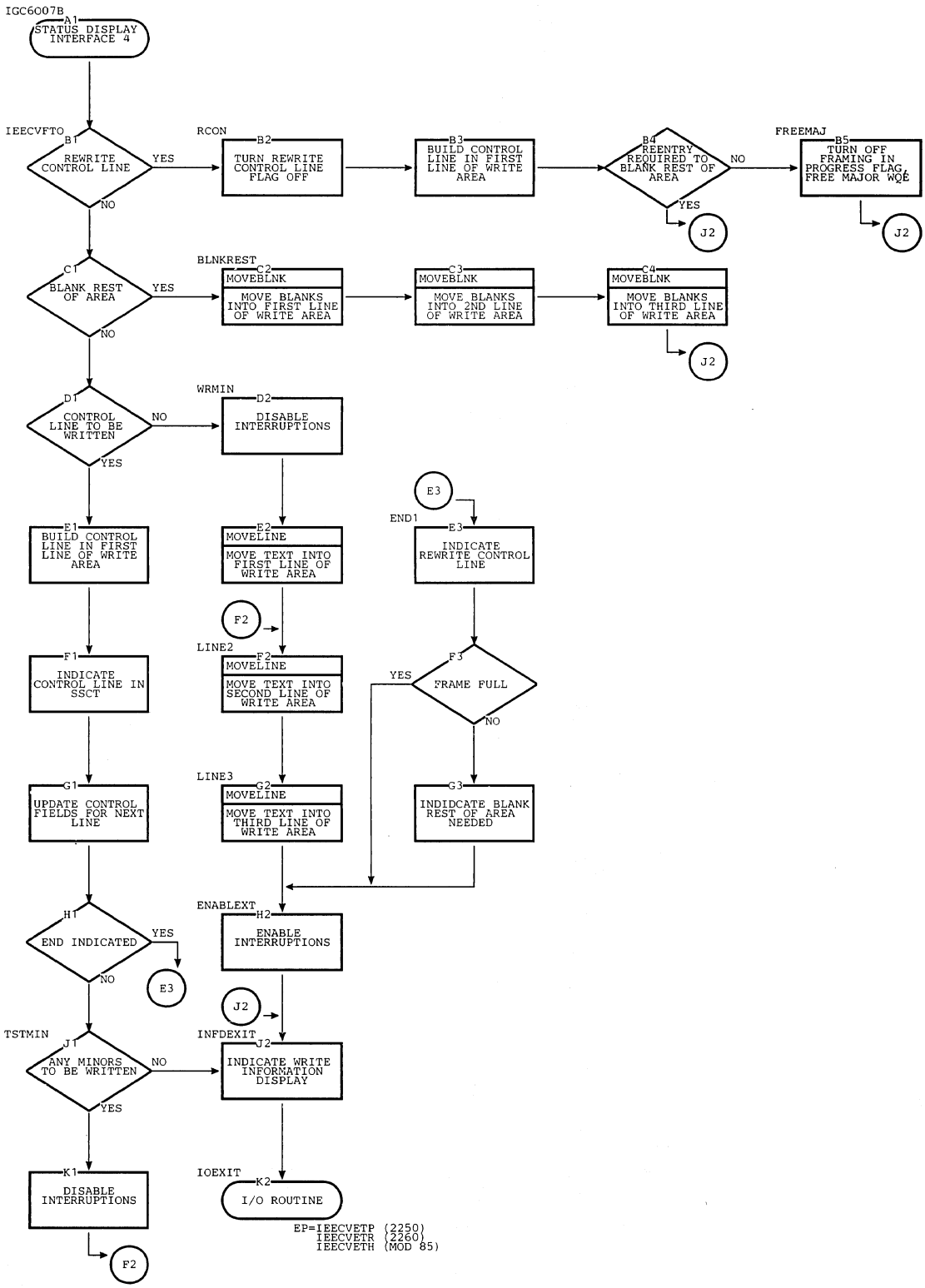


Chart KH. DIDOCS Status Display 5 (Part 1 of 3)

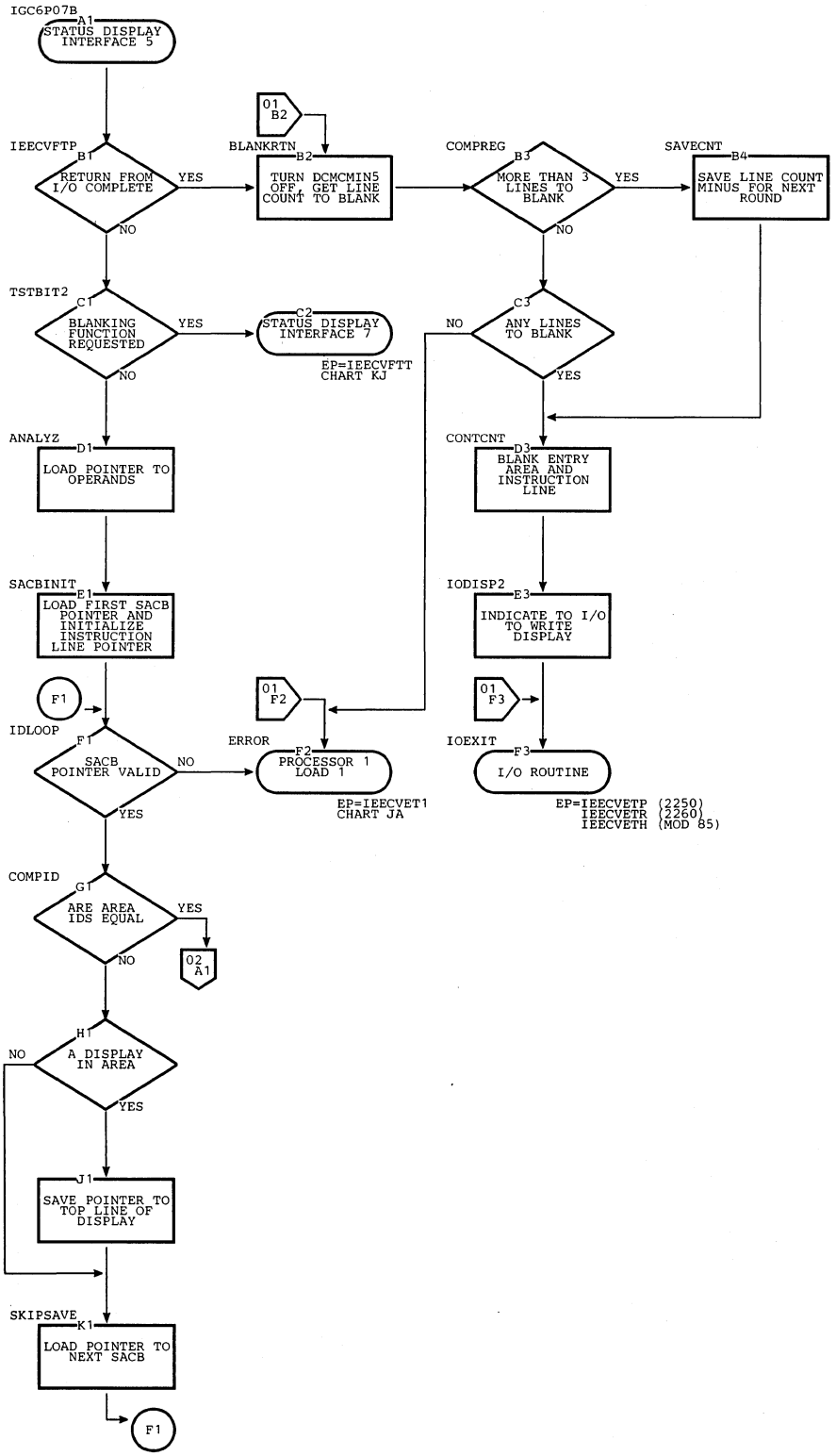


Chart KH. DIDOCS Status Display 5 (Part 2 of 3)

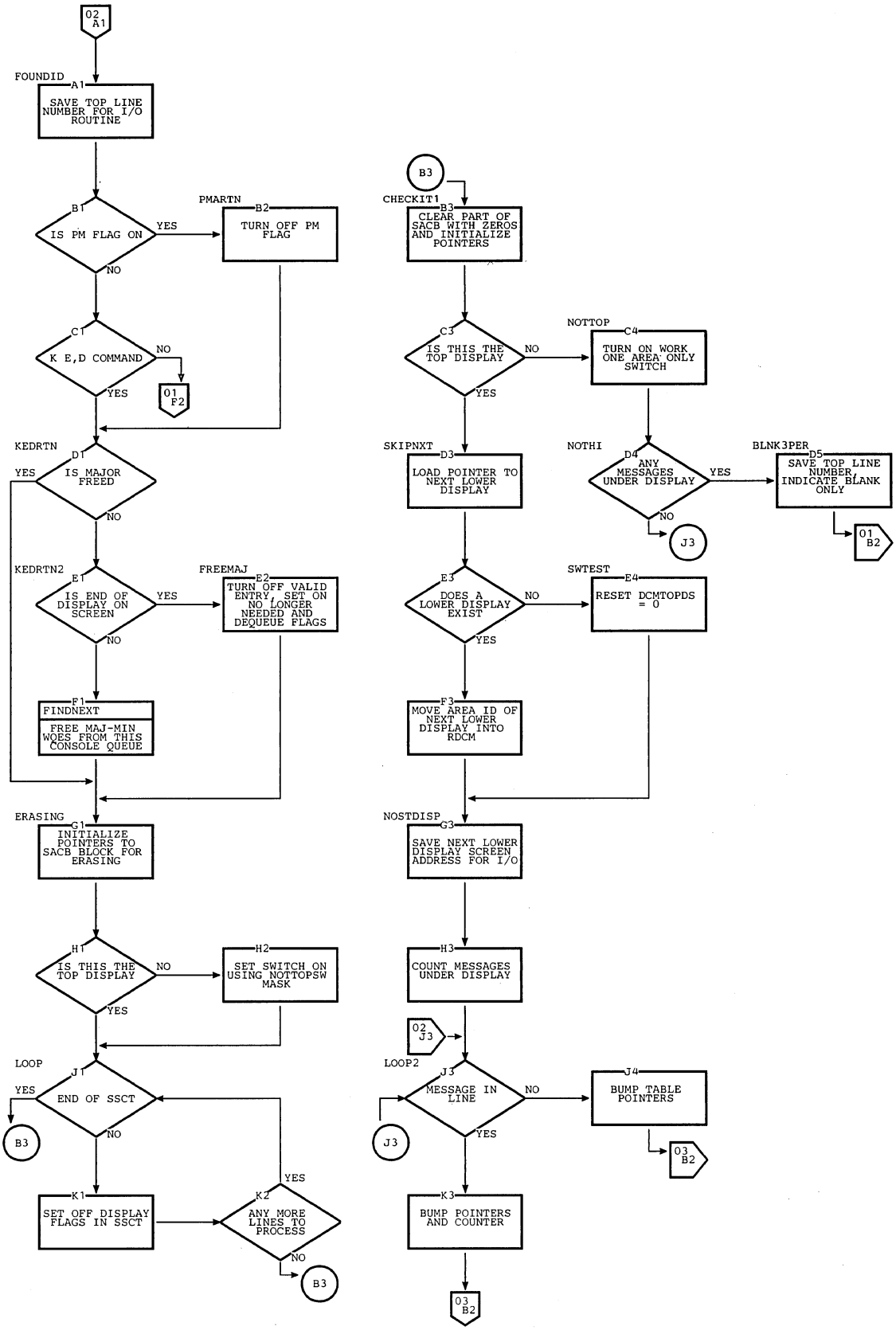


Chart KH. DIDOCS Status Display 5 (Part 3 of 3)

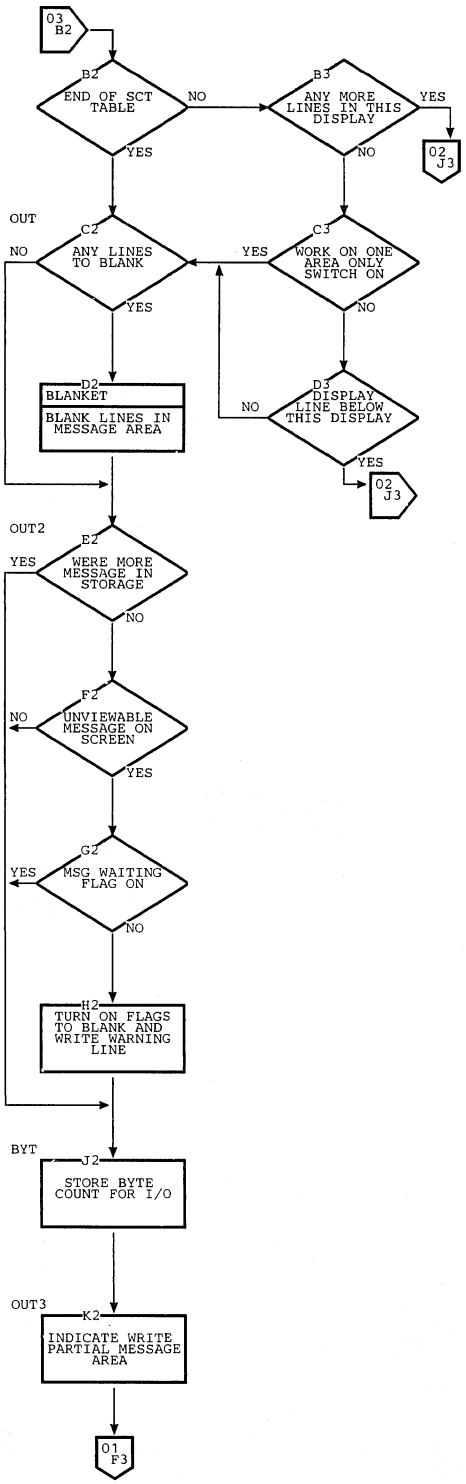


Chart KI. DIDOCS Status Display 6 (Part 1 of 2)

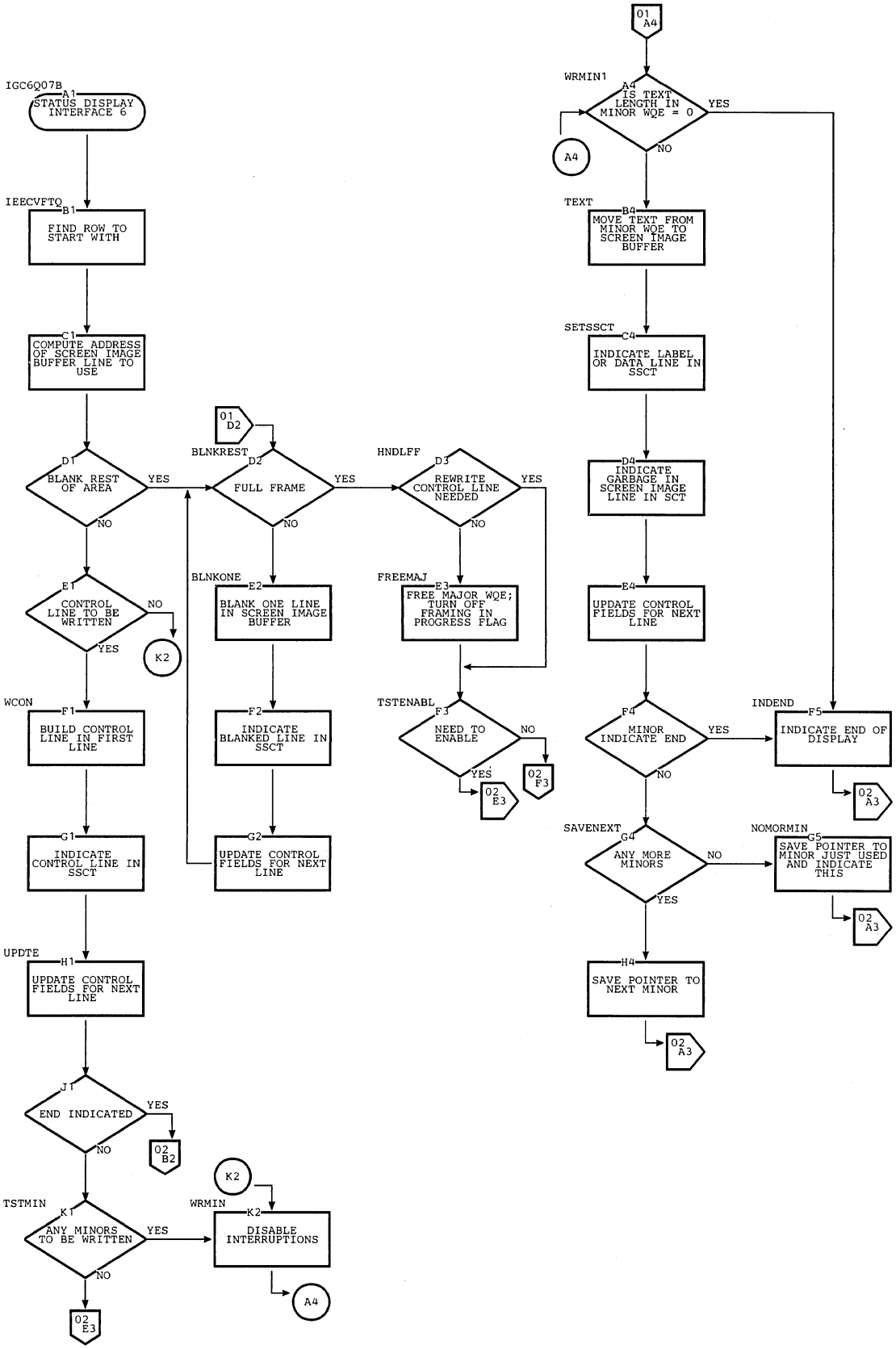


Chart KI. DIDOCS Status Display 6 (Part 2 of 2)

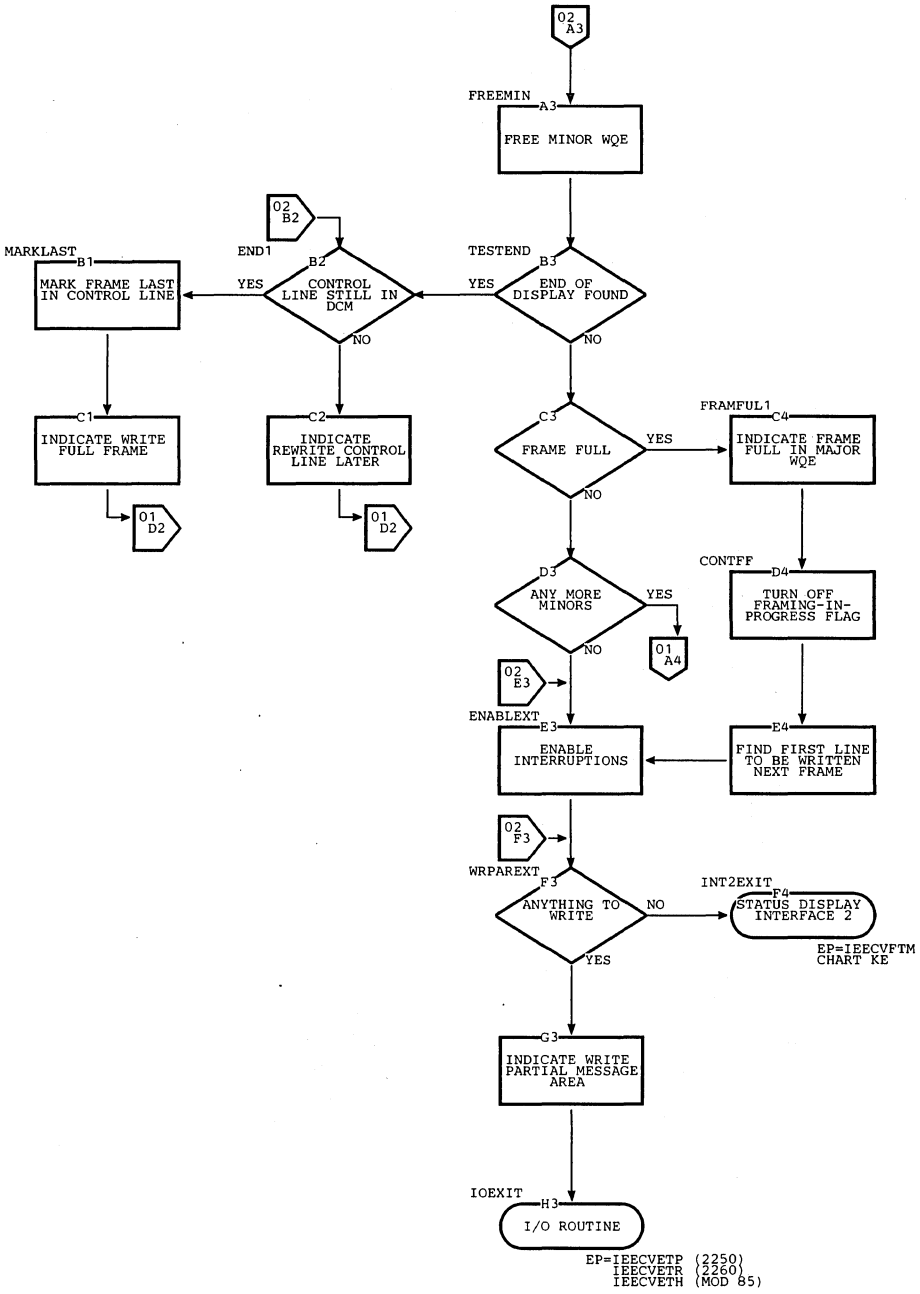


Chart KJ. DIDOCS Status Display 7

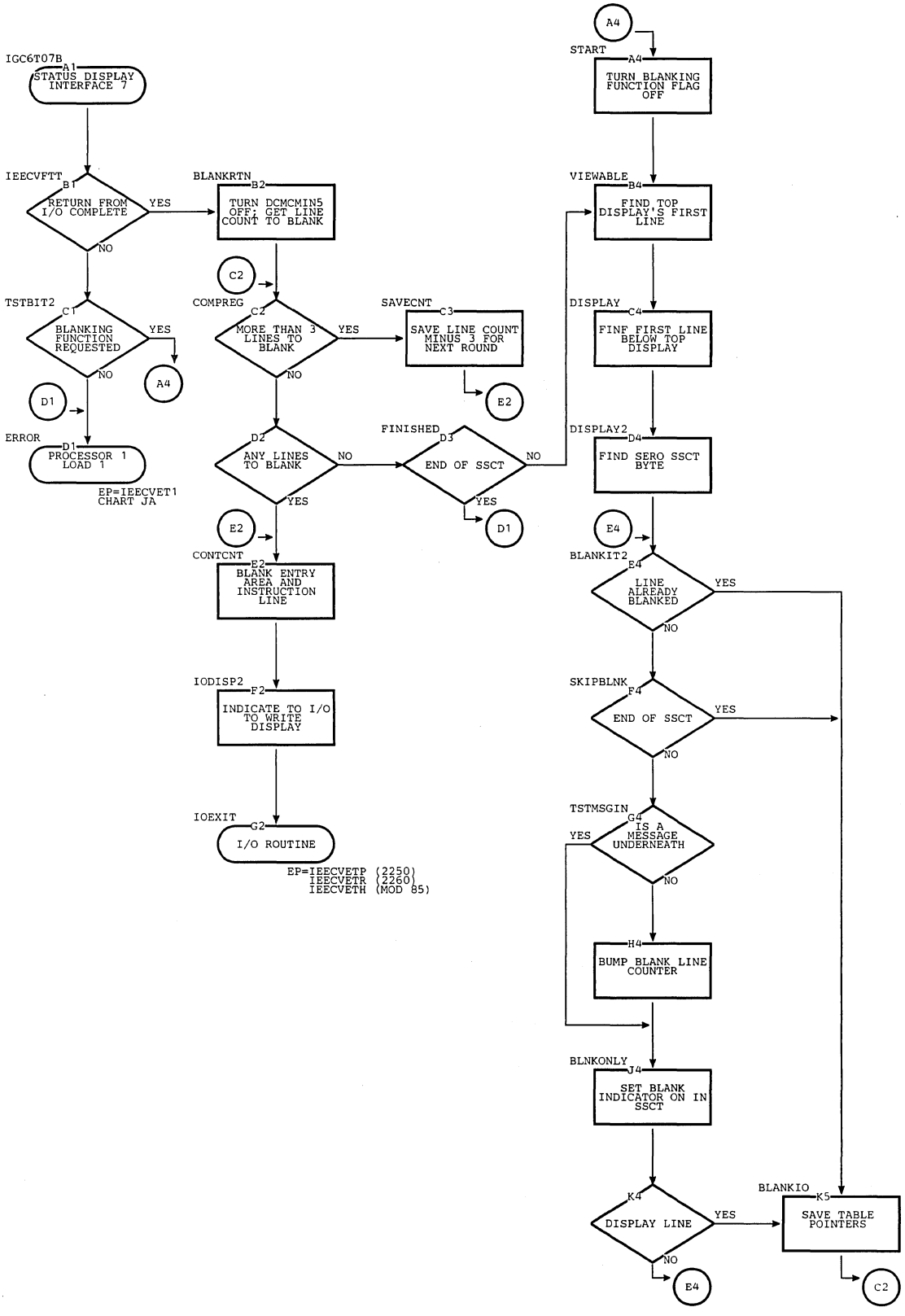


Chart KK. DIDOCS Open/Close

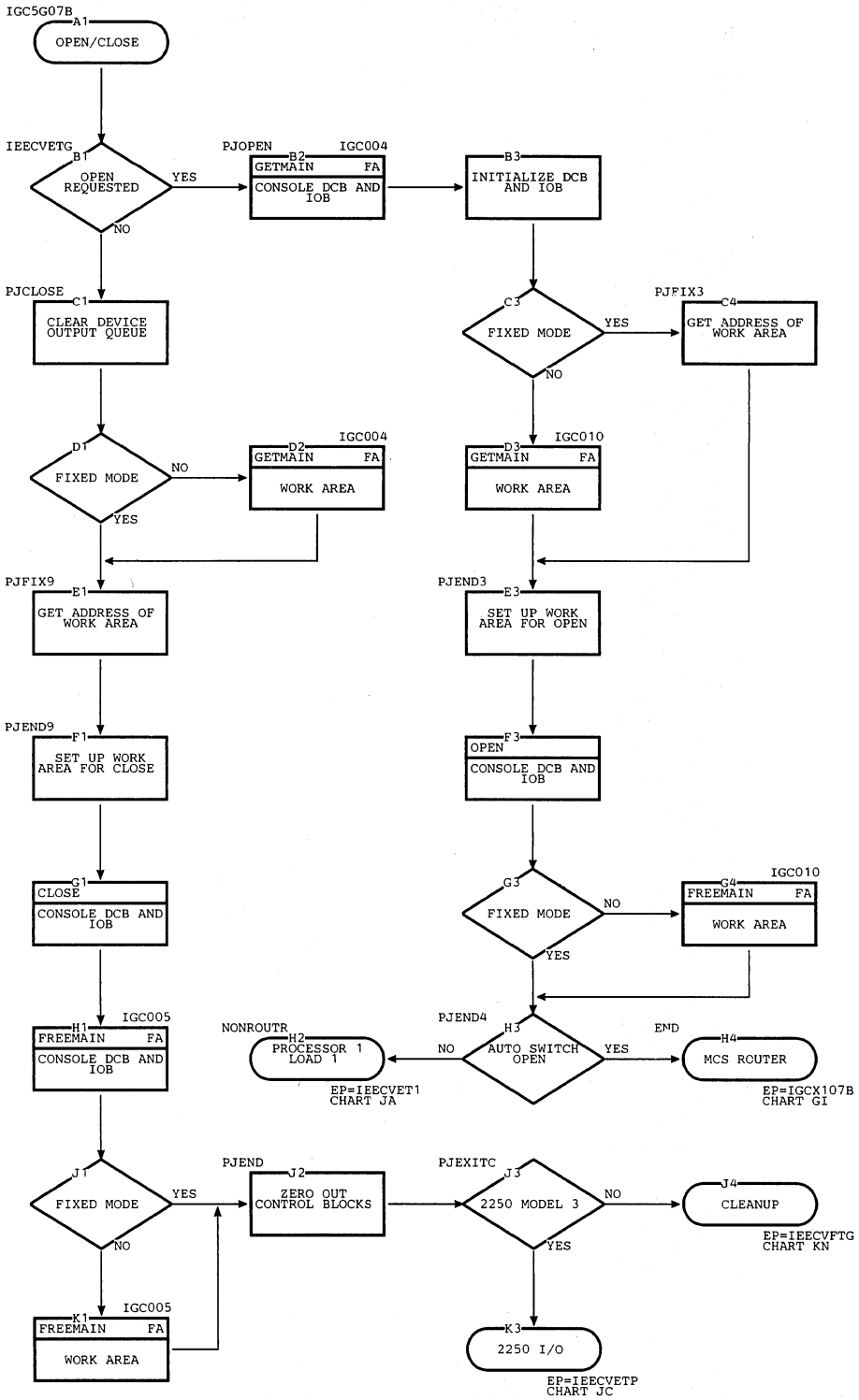


Chart KL. DIDOCS PFK 1 (Part 1 of 2)

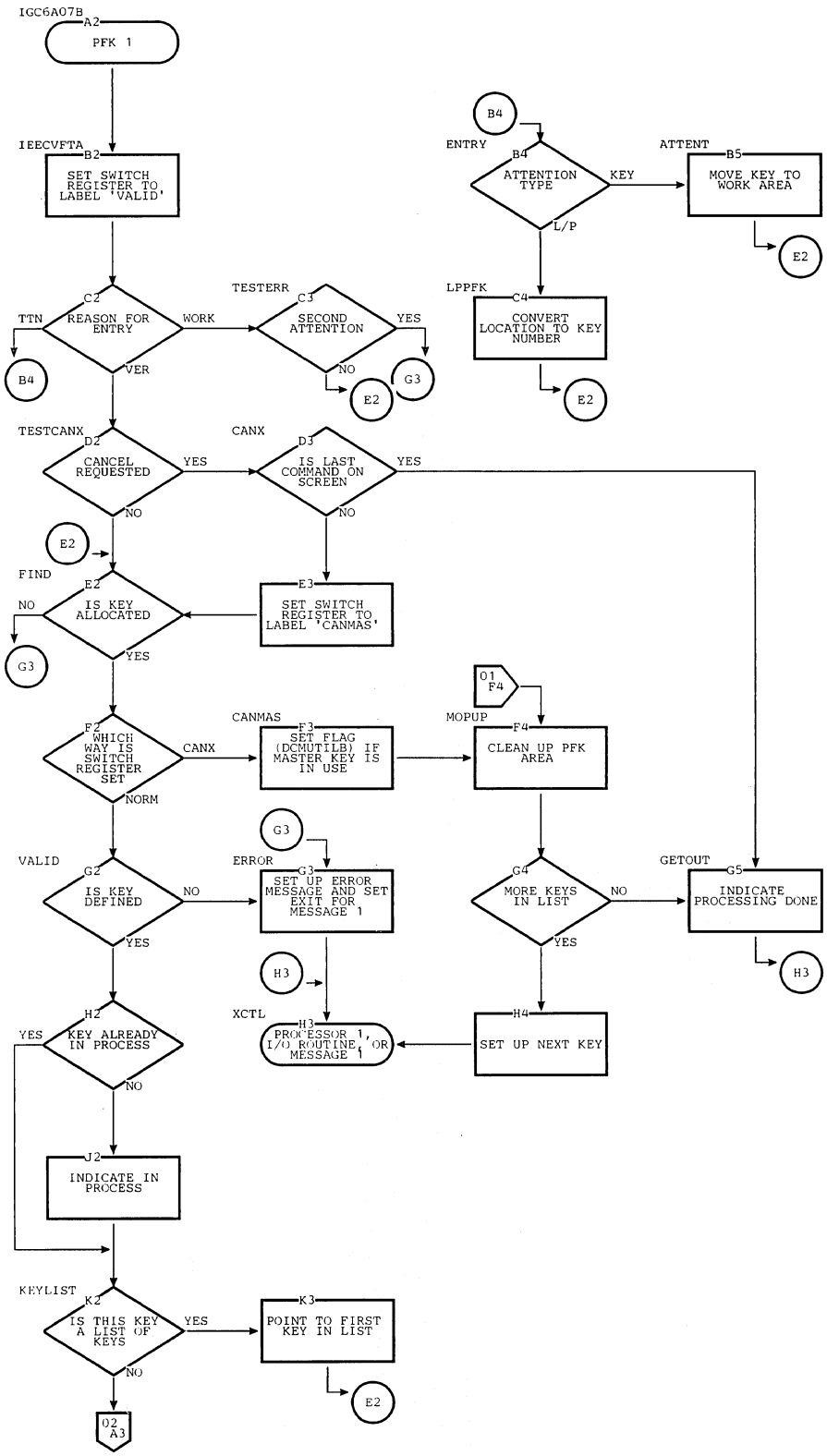


Chart KL. DIDOCS PFK 1 (Part 2 of 2)

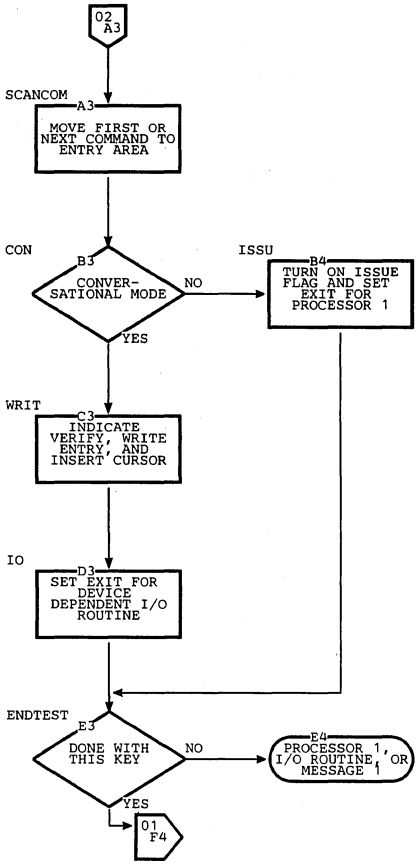


Chart KM. DIDOCS PFK 2

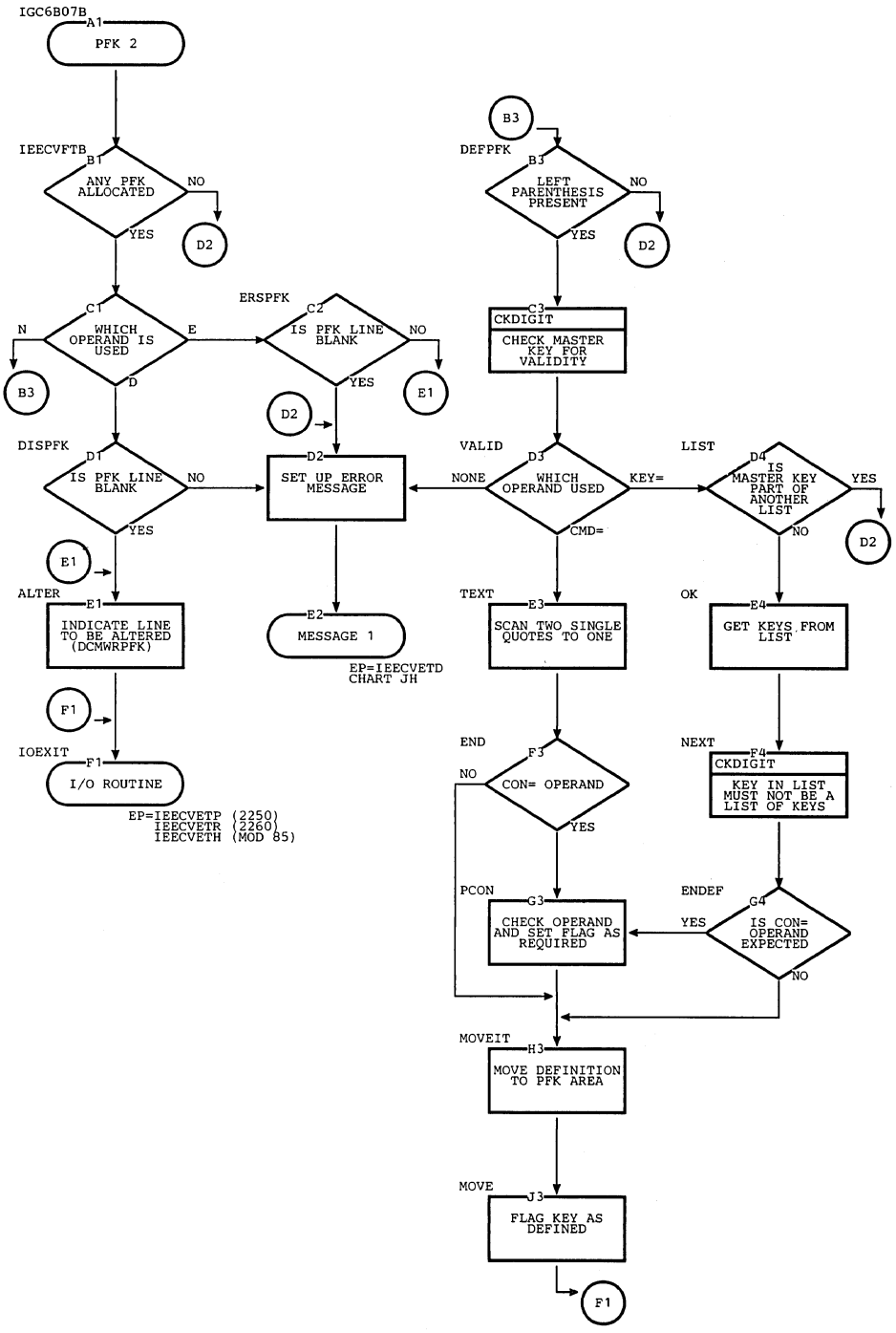


Chart KN. DIDOCS Cleanup (Part 1 of 2)

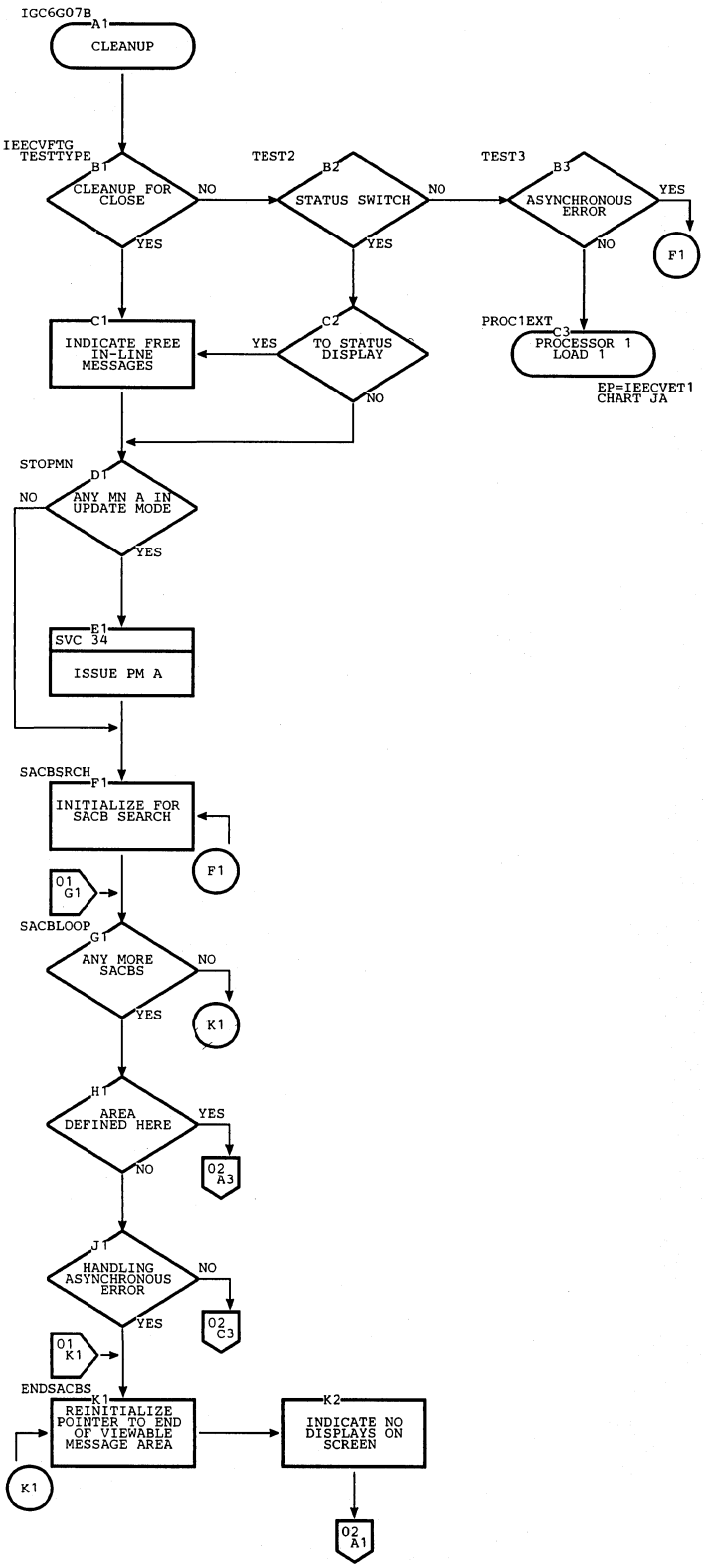


Chart KN. DIDOCS Cleanup (Part 2 of 2)

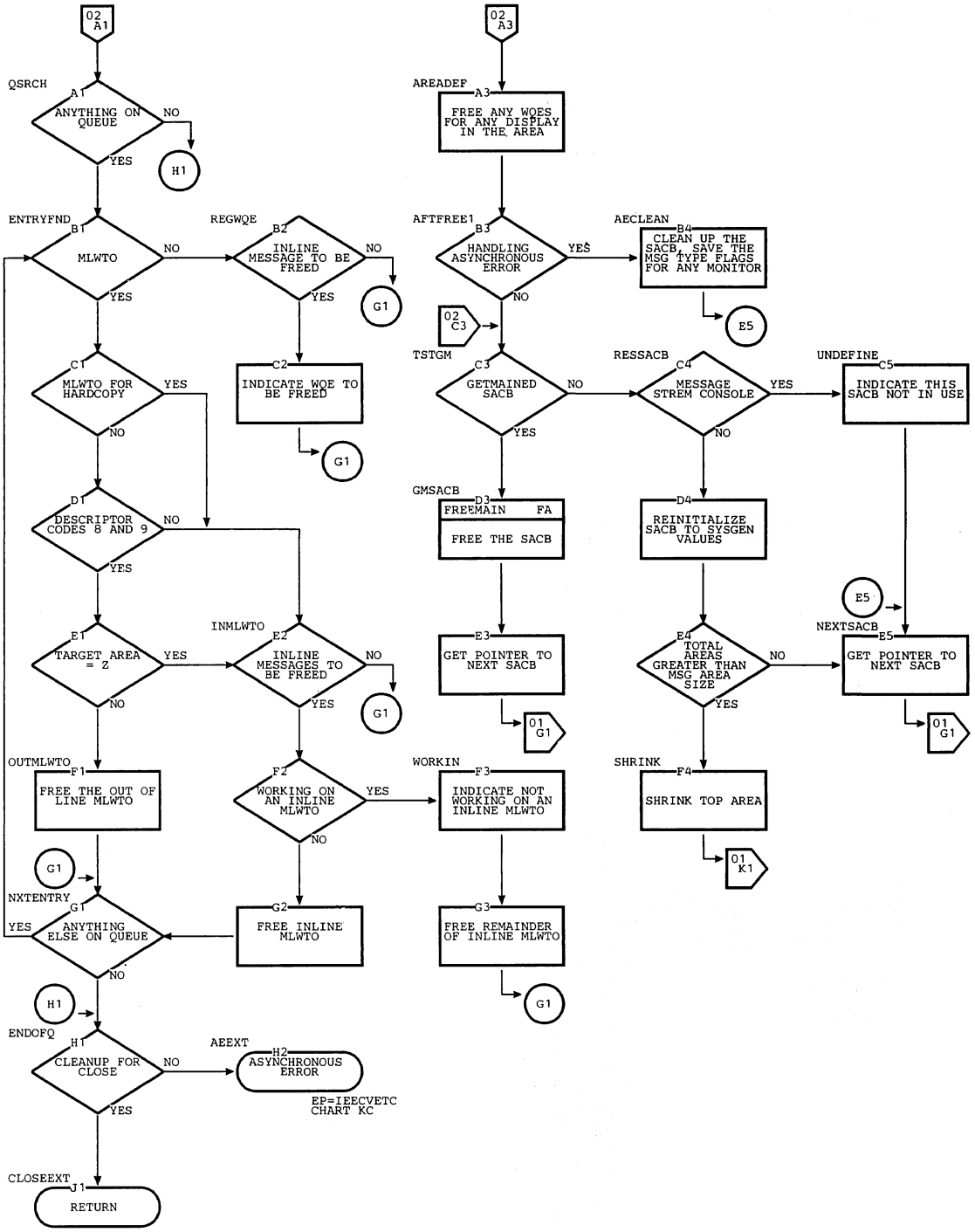


Chart LA. Checkpoint Modules

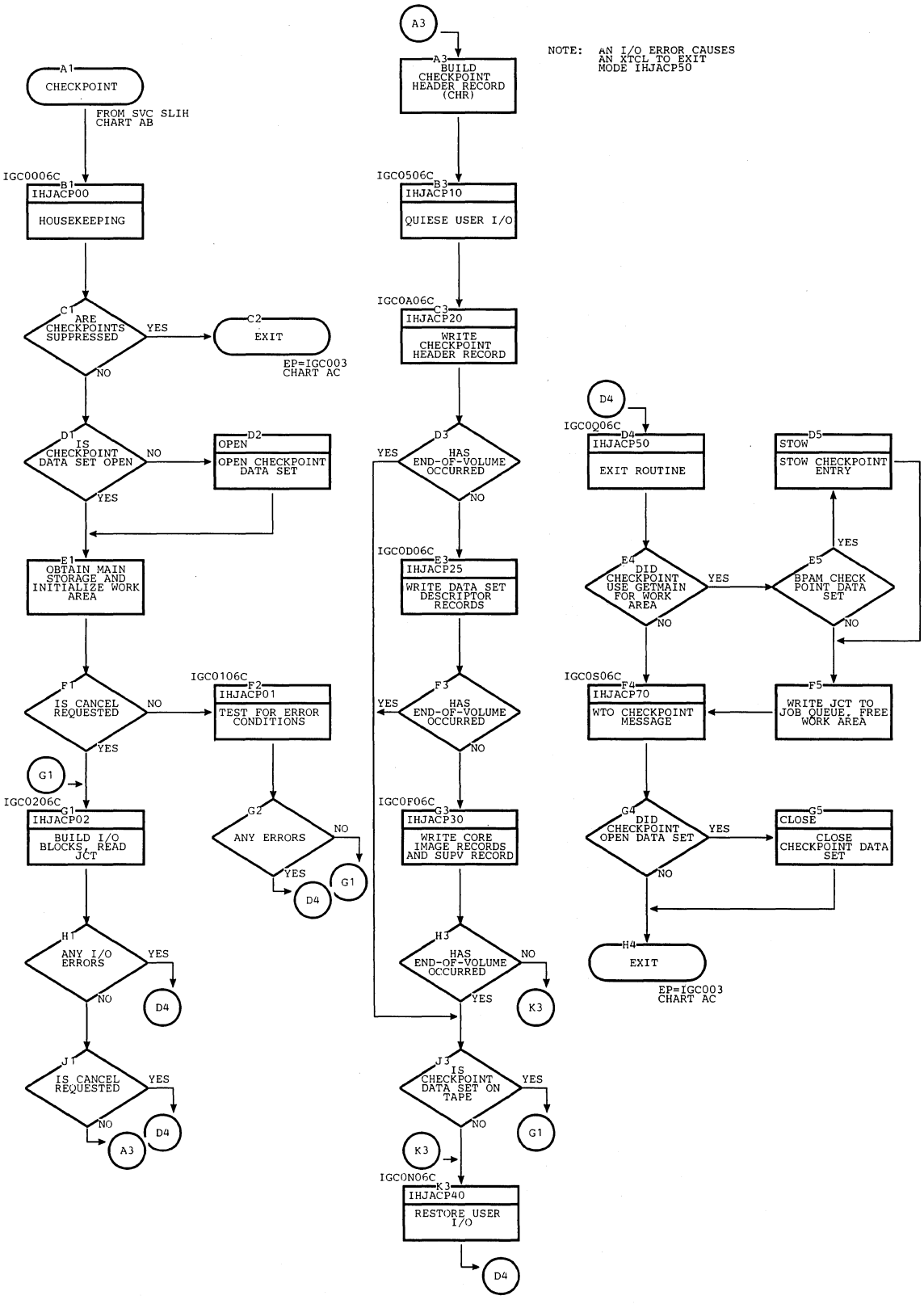


Chart LB. Restart Modules

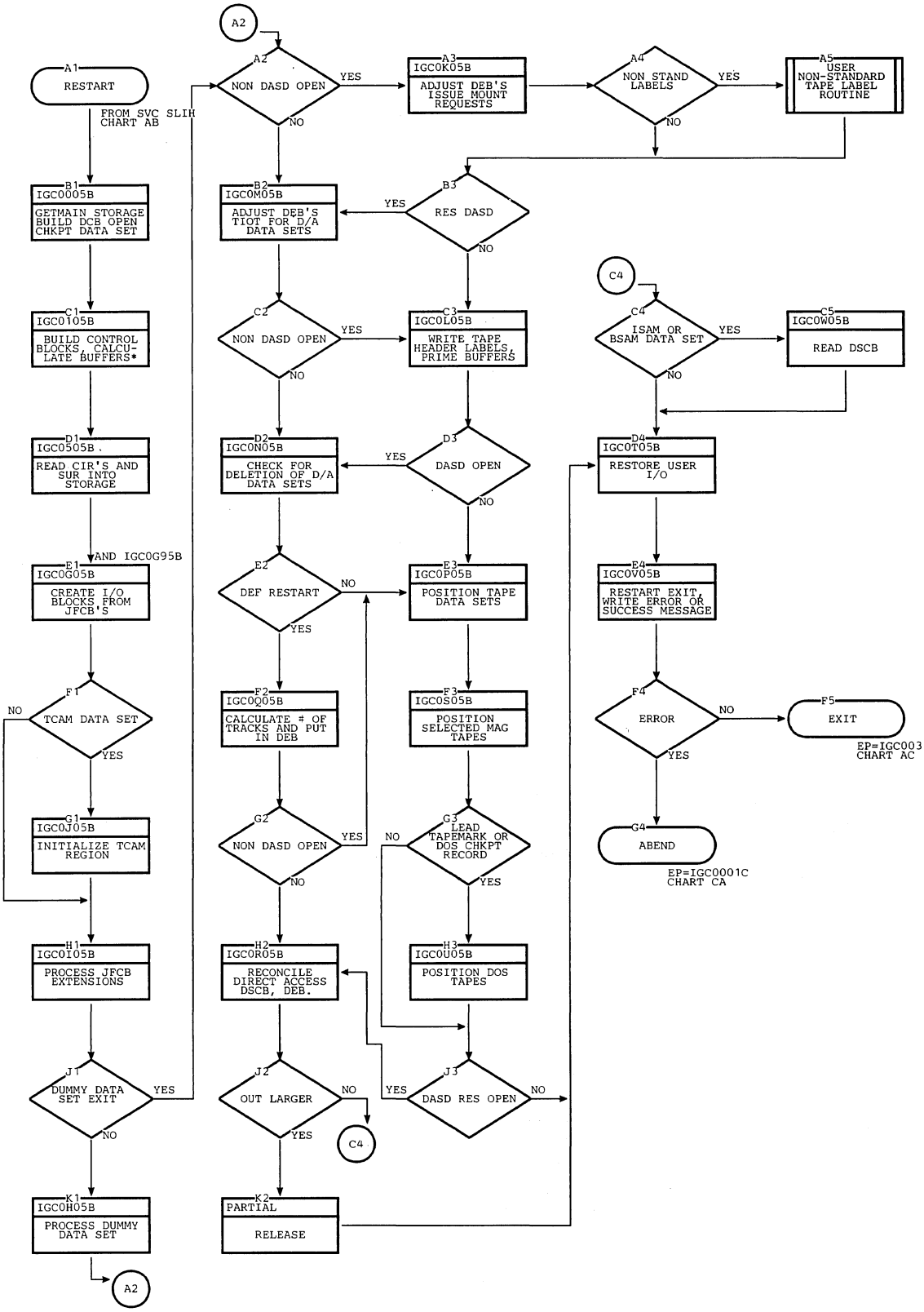


Chart MA. System Management Facility EXCP Counter (Part 1 of 2)

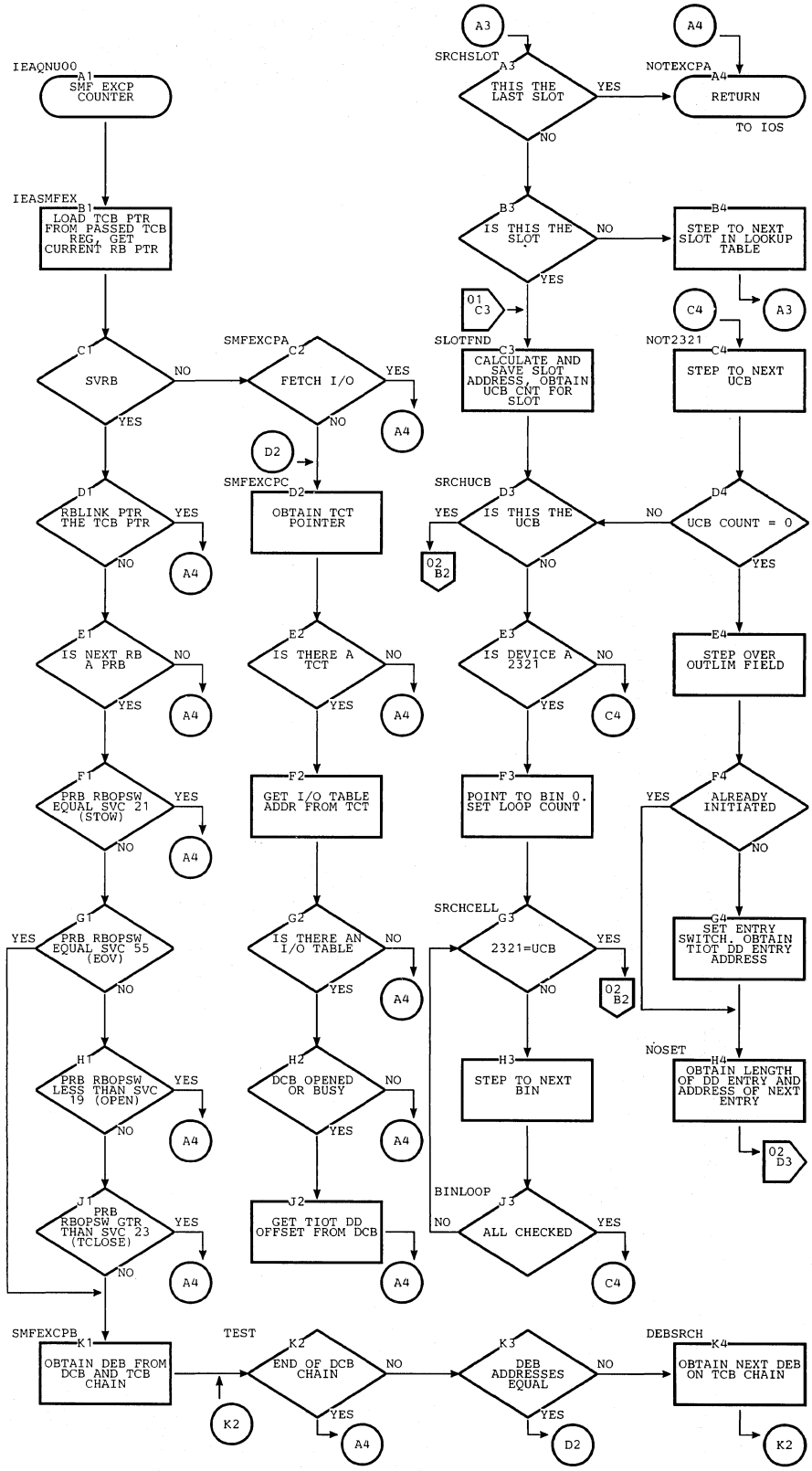


Chart MA. System Management Facility EXCP Counter (Part 2 of 2)

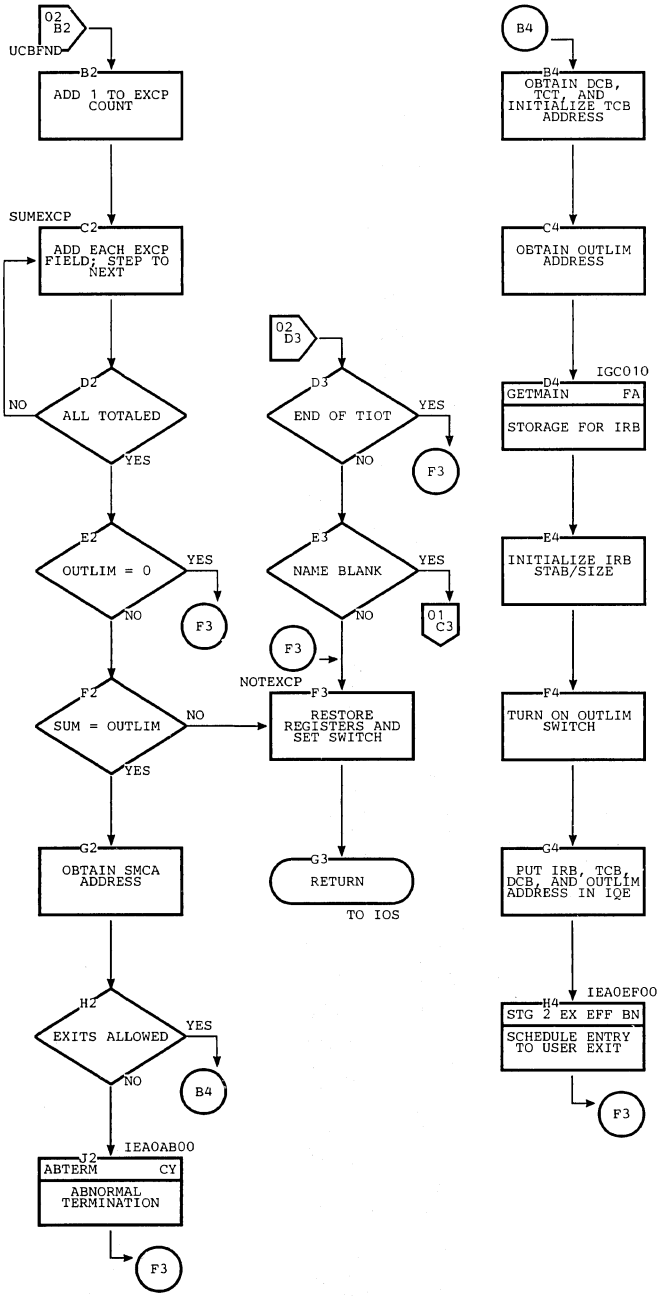
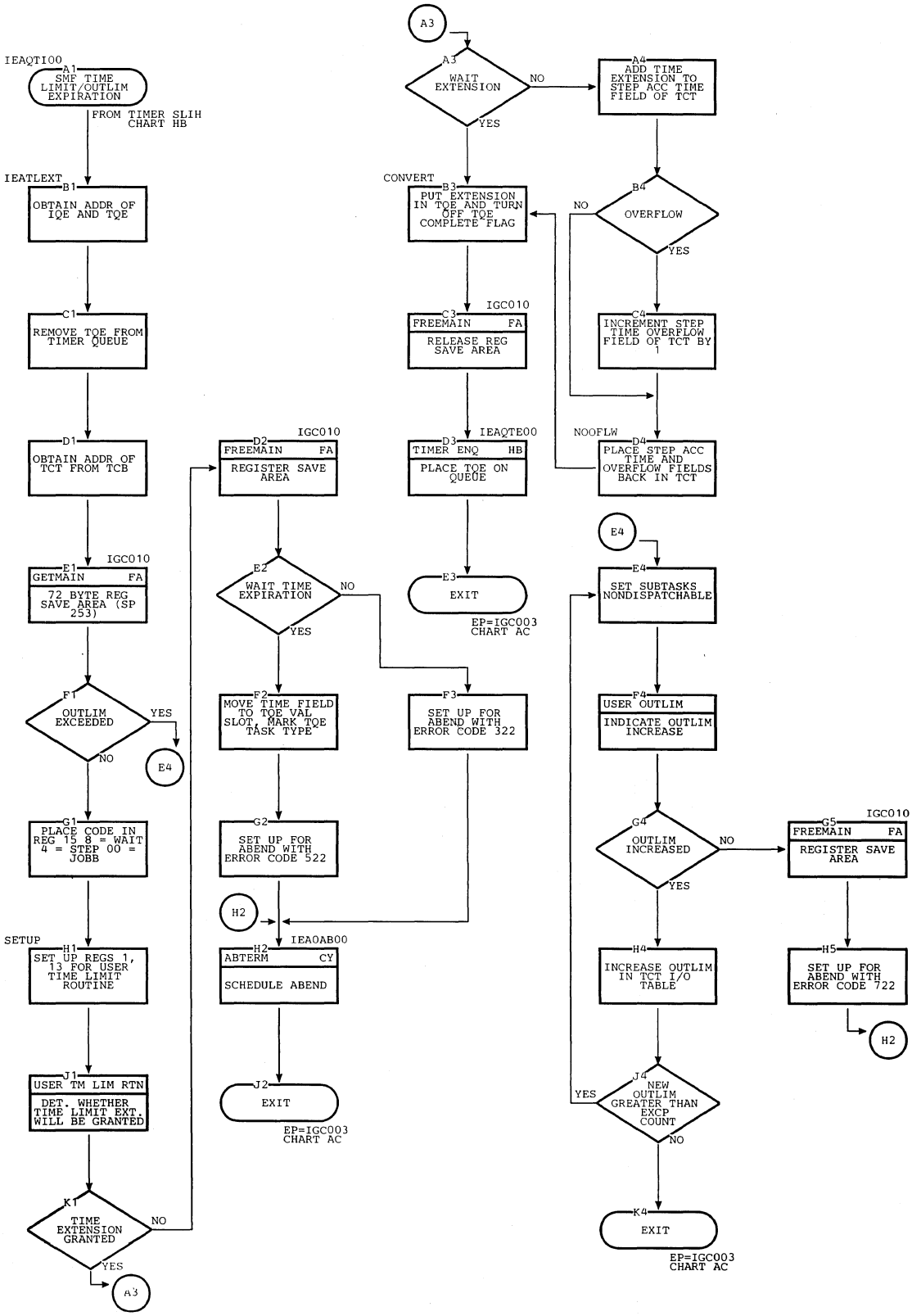


Chart MB. System Management Facility Time Limit Expiration



Module Name	Routine Name	Entry Point
IEAAIH00*	Program Check FLIH	IEAAPK00
	Input/Output FLIH	IEAAIO02
	SVC FLIH	IEAASC00
	Timer/External FLIH	IEAQEX00
	Dispatcher	IEA0DS
	SVC Type 1 Exit	IEA0XE00
IEAATA00	SVC SLIH	IEAATA00
	SVC EXIT (types 2, 3, 4)	IEAAXT
	Validity Check	IEA0VL00
	Task Switch Determination	TASKSWIT
*See also Task Supervision Figure 46.		

Figure 45. Interruption Supervision Modules

Module Name	Routine Name	Entry Point
IEAAIH00*	Stage 2 Exit Effector	IEA0EF00
	Stage 3 Exit Effector	IEAEXTEF
IEAAPT00	POST	IGC002
IEAAPX00	SPIE	IGC014
IEAAST00	STAE	IGC00060
IEAAWT00	WAIT	IGC001
IEAAXR00	EXTRACT without subtasking	IGC040
IEABXR00	EXTRACT with subtasking	IGC040
IEAGED02	DETACH	IGC062
IEAGENQ1	ENQ/DEQ	IGC048 IGC056
IEAGENQ2	ENQ/DEQ with Shared DASD	IGC048 IGC056
IEAQCB01	Nonexecutable; contains address of first major QCB.	IEAQCB01
IEAQAT00	ATTACH with subtasking	IGC042
IEAQCH00	CHAP	IGC044
IEAQEF00	Stage 1 Exit Effector	IGC043
*See also Interruption Supervision Figure 45.		

Figure 46. Task Supervision Modules

Module Name	Routine Name	Entry Point
IEAMAD00	ABDUMP 1	IGC0005A
IEAAAD01	ABDUMP 4	IGC0105A
IEAAAD02	ABDUMP 5	IGC0205A
IEAAAD03	ABDUMP 6	IGC0305A
IEAAAD04	ABDUMP 8	IGC0405A
IEAAAD05	ABDUMP 9	IGC0505A
IEAAAD0A	ABDUMP 3	IGC0A05A
IEAAAD0B	ABDUMP 7	IGC0B05A
IEAAAD0C	ABDUMP 11	IGC0C05A
IEAAAD0D	ABDUMP 2	IGC0D05A
IEAAAD0E	ABDUMP 10	IGC0E05A
IEAAAD0F	ABDUMP 12	IGC0F05A
IEAAAD0K	ABDUMP 13	IGC0K05A
IEAAAD0M	ABDUMP 14	IGC0M05A
IEAAAD0Y	SVCDUMP 1	IGC0005A
IEAAAD0Z	SVCDUMP 2	IGC0Z05A
IEAASPRG	Subsystem Purge	IEAASPRG
IEADTM22	DAR 1	IGC0221C
IEADTM23	DAR 2	IGC0321C
IEADTM24	DAR 3	IGC0421C
IEADTM25	DAR4	IGC0521C
IEAGAB00	ABTERM without trace table	IEA0AB00
IEAIAB00	ABTERM with trace table	IEA0AB00
IEANAB00	ABTERM with subtasking	IEA0AB00
IEANTM00	ABEND Normal Processor and Abnormal Router	IGC0001C
IEANTM01	ABEND/STAE Graphics Linkage	IGC0101C
IEANTM02	ABEND I/O Purge	IGC0201C
IEANTM03	ABEND Control Block Validity Check	IGC0301C
IEANTM04	ABEND Dump Test	IGC0401C
IEAMTM05	ABEND Open Dump Data Set	IGC0501C
IEANTM05	ABEND Open Dump Data Set (with subtasking)	IGC0501C
IEANTM06	ABEND Dump	IGC0601C
IEANTM07	ABEND Termination	IGC0701C
IEANTM08	ABEND Indicative Dump	IGC0801C
IEANTM09	ABEND Recursion Processor	IGC0901C
IEANTM0A	ABEND Steal Main Storage	IGC0A01C
IEACTM0B	ABEND WTOR Purge (with MCS)	IGC0B01C

Figure 47. Task Termination Modules (Part 1 of 2)

Module Name	Routine Name	Entry Point
IEANTM0B	ABEND WTOR Purge (without MCS)	IGC0B01C
IEANTM0C	ABEND Loading Program Purge (with subtasking)	IGC0C01C
IEANTM0D	ABEND Subtask Enqueue Purge (with subtasking)	IGC0D01C
IEANTM0E	ABEND Data Set Close	IGC0E01C
IEANTM0F	ABEND IQE Purge and Subtask Cleanup	IGC0F01C
IEANTM0G	ABEND Abnormal Termination and ABEND Recursion	IGC0G01C
IEANTM0H	ABEND WTO Recursion Messages	IGC0H01C
IEANTM0J	ABEND JPAQ and GQE Validity Check	IGC0J01C
IEANTM0M	ABEND Message List Processor	IGC0M01C
IEASTM11	ASIR Purge I/O	IGC0111C
IEASTM12	ASIR Purge IOBs and Close DCBs (non-ISAM, non-TAM)	IGC0211C
IEASTM13	ASIR Purge RB queue	IGC0211C
IEASTM14	ASIR Purge IOBs and Close DCBs (ISAM & TAM)	IGC0411C

Figure 47. Task Termination Modules (Part 2 of 2)

Module Name	Routine Name	Entry Point
IEAAAT00	Attach without subtasking	IGC042
IEAAID00	Identify	IGC041
IEAATC00	Link	IGC006
	XCTL	IGC007
	Load	IGC008
	FINCH	IEA0FN00
	Transient Area Loading Task	IEAFNCH
IEAADL00	Delete	IGC009

Figure 48. Contents Supervision Modules

Module Name	Routine Name	Entry Point
IEAAMS00	GETMAIN	IGC004
	FREEMAIN	IGC005

Figure 49. Main Storage Supervision Modules

Module Name	Routine Name	Entry Point
IEECLCTX	Console Switch 1 (MCS)	IGCXL07B
IEECMAWR	Router (MCS)	IEECMAWR
IEECMCTR	Mini-Router (MCS)	IGC0007B
IEECMCTX	Console Switch 2 (MCS)	IGCXM07B
IEECMDOM	Delete Operator Message	IEECMDOM
IEECMDSV	Device Interface (MCS)	IEECMDSV
IEECMPMC	2540 Processor (MCS)	IGC1107B
IEECMPMP	1403/1443 Processor (MCS)	IGC2107B
IEECMPMX	1052 Processor (MCS)	IGC0107B
IEECMPM1	1052 Processor, Load 2 (MCS)	IGC0207B
IEECMWSV	WTO Processor (MCS)	IEECMWSV
IEECMWTL	NIP Message Buffer Writer (MCS)	IGC0907B
IEECNCTX	Console Switch 3 (MCS)	IGCXN07B
IEECOCTX	Console Switch 4 (MCS)	IGCXO07B
IEECVCRA	I/O Attention Handler	IEEBA1
IEECVCRX	External Interruption Handler	IEEBC1PE
IEECVCTI	Initialization	IEECVCTI
IEECVCTR	Router (non-MCS)	IGC0007B
IEECVCTW	Wait (non-MCS)	IEECIR45
IEECVCTX	Console Switch (non-MCS)	IGCXL07B
IEECVML1	Multiple-line WTO (non-MCS) Load 1	IEECVML1
IEECVML2	Multiple-line WTO (non-MCS) Load 2	IEECVML2
IEECVML3	Multiple-line WTO (MCS) Load 1	IEECVML3
IEECVML4	Multiple-line WTO (non-MCS) Load 3	IEECVML4
IEECVML5	Multiple-line WTO (MCS) Load 2	IEECVML5
IEECVML6	Multiple-line WTO (MCS) Load 3	IEECVML6
IEECVOCC	2540 Open/Close	IGC1I07B
IEECVOCP	1403/1443 Open/Close	IGC2I07B
IEECVOCX	1052 Open/Close	IGC0I07B
IEECVPMC	2540 Processor (non-MCS)	IGC1107B
IEECVPMP	1403/1443 Processor (non-MCS)	IGC2107B
IEECVPMX	1052 Processor Load 1 (non-MCS)	IGC0107B
IEECVPM1	1052 Processor, Load 2 (non-MCS)	IGC0207B
IEEC2740	2740 Processor (MCS)	IEEC2740
IEENFWTO	WTO Non-MCS	IGC0003E
IEENFWTO	WTO MCS	IGC0003E
IEEVWTOR	WTOR Service	IGC0103E

Figure 50. Communications Task Modules

Module Name	Routine Name	Entry Point
IEA0RT00	Time	IGC011
IEA0ST00	TTIMER	IGC046
	STIMER	IGC047
IEA0TI0x*	Timer SLIH	IEA0TI02
	Timer Enqueue	IEAQTE00
	Timer Dequeue	IEAQTD00
*Module name is option dependent. See Figure 39.		

Figure 51. Timer Supervision Modules

Module Name	Routine Name	Entry Point
IEWSVOVR	Resident Overlay	IGC037
IEWSYOVR	Nonresident Overlay	IEWSZOVR
IEWSXOVR	Nonresident Overlay with SEGWT Checking	IEWSZOVR

Figure 52. Overlay Supervision Modules

Module Name	Routine Name	Entry Point
IGC0006C	Housekeeping 1	IGC0006C
IGC0106C	Housekeeping 2	IGC0106C
IGC0206C	Housekeeping 3	IGC0206C
IGC0506C	Check I/O	IGC0506C
IGC0A06C	Preserve 1	IGC0A06C
IGC0D06C	Preserve 2	IGC0D06C
IGC0F06C	Checkmain	IGC0F06C
IGC0N06C	Resume I/O	IGC0N06C
IGC0Q06C	Exit	IGC0Q06C
IGC0S06C	Message	IGC0S06C
IGC0L05B	SYSIN/SYSOUT Nondirect Access Processor	IGC0L05B
IGC0N05B	SYSIN/SYSOUT Direct Access Positioning 1	IGC0N05B
IGC0Q05B	SYSIN/SYSOUT Direct Access Positioning 2	IGC0Q05B
IGC0105B	Housekeeping 1	IGC0105B
IGC0205B	Housekeeping 2	IGC0205B
IGC0505B	Repmain 1	IGC0505B
IGC0605B	Repmain 2	IGC0605B
IGC0G05B	JFCB 1	IGC0G05B
IGC0G95B	Dummy Data Set Processor 1	IGC0G95B
IGC0H05B	Dummy Data Set Processor 2	IGC0H05B
IGC0I05B	JFCB 2	IGC0I05B
IGC0J05B	TCAM Data Set Processor	IGC0J05B
IGC0K05B	Mount/Verify Nondirect Access	IGC0K05B
IGC0M05B	Mount/Verify Direct Access	IGC0M05B
IGC0P05B	Data Set Processor 1	IGC0P05B
IGC0R05B	Data Set Processor 2	IGC0R05B
IGC0S05B	Data Set Processor 1A	IGC0S05B
IGC0T05B	Final Processing	IGC0T05B
IGC0U05B	Data Set Processor	IGC0U05B
IGC0V05B	Exit	IGC0V05B
IGC0W05B	ISAM/BDAM Data Set Processor	IGC0W05B

Figure 53. Checkpoint/Restart Modules

Module Name	Routine Name	Entry Point
IGC5A07B	Options	IEECVETA
IGC5C07B	Asynchronous Error	IEECVETC
IGC5D07B	Message 1	IEECVETD
IGC5E07B	Message 2	IEECVETE
IGC5F07B	Light Pen/Cursor	IEECVETF
IGC5G07B	Open/Close	IEECVETG
IGC5H07B	Model 85 I/O	IEECVETH
IGC5J07B	Roll Mode	IEECVETJ
IGC5K07B	Timer Interpreter	IEECVETK
IGC5P07B	2250 I/O 1	IEECVETP
IGC5Q07B	2250 I/O 2	IEECVETQ
IGC5R07B	2260 I/O 1	IEECVETR
IGC5Z07B	Processor 1, Load 2	IEECVETZ
IGC5107B	Processor 1, Load 1	IEECVET1
IGC5207B	Display 1	IEECVET2
IGC5307B	Display 2	IEECVET3
IGC5407B	Command	IEECVET4
IGC5607B	Delete 1	IEECVET6
IGC5707B	Delete 2	IEECVET7
IGC5807B	Delete 3	IEECVET8
IGC5907B	Delete 4	IEECVET9
IGC6A07B	PFK 1	IEECVFTA
IGC6B07B	PFK 2	IEECVFTB
IGC6D07B	Message 3	IEECVFTD
IGC6G07B	Cleanup	IEECVFTG
IGC6L07B	Status Display Interface 1	IEECVFTL
IGC6M07B	Status Display Interface 2	IEECVFTM
IGC6N07B	Status Display Interface 3	IEECVFTN
IGC6O07B	Status Display Interface 4	IEECVFTO
IGC6P07B	Status Display Interface 5	IEECVFTP
IGC6Q07B	Status Display Interface 6	IEECVFTQ
IGC6R07B	2260 I/O 2	IEECVFTR
IGC6T07B	Status Display Interface 7	IEECVFTT
IGC6Z07B	Processor 0, Load 2	IEECVFTZ
IGC6107B	Processor 0, Load 1	IEECVFT1
IGC6207B	Display 3	IEECVFT2

Figure 54. DIDOCS Modules

Entry Point	Name of Routine	Module Name	CSECT	Chart	SVC	Type
IEAAPK00	Program Check FLIH	IEAAIH00	IEAAIH00	AE		
IEAASC00	SVC FLIH	IEAAIH00	IEAAIH00	AA		
IEAASPRG	Subsystem Purge	IEAASPRG	IEAASPRG			
IEAATA00	SVC SLIH	IEAATA00	IEAATA00	AB		
IEAAXT	SVC EXIT (Types 2, 3, and 4)	IEAATA00	IEAATA00	AC		
IEAEXTEF	Stage 3 Exit Effector	IEAAIH00	IEAEXTEF	BO		
IEAFNCH	Transient Area Loading Task	IEAATC00	IGC006	ED		
IEAQCB01	Nonexecutable; points to address of first QCB	IEAQCB01	IEAQCB01	(Fig. 23)		
IEAQEX00	Timer/External FLIH	IEAAIH00	IEAAIH00	AD		
IEAQTD00	Timer DEQ	IEA0TI00	IEA0TI02	HB		
IEAQTE00	Timer ENQ	IEA0TI00	IEA0TI02	HB		
IEAXKALL	Extended Precision Floating Point Simulator for System/360 models	IEAXPALL	IEAXPALL			
IEAXKDXR	Extended Precision Floating Point Simulator for System/360 Models 85, 195 and System/370 models	IEAXPDXR	IEAXPDXR			
IEAXPALL	Extended Precision Floating Point Simulator for System/360 models	IEAXPALL	IEAXPALL			
IEAXPDXR	Extended Precision Floating Point Simulator for System/360 Models 85, 195 and System/370 models	IEAXPDXR	IEAXPDXR			
IEAXPSIM	Extended Precision Floating Point Simulator CPU Determination	IEAXPSIM	IEAXPSIM			
IEA0AB00	ABTERM without trace table	IEAGAB00	IEA0AB00			
IEA0AB00	ABTERM with trace table	IEAIAB00	IEA0AB00			
IEA0AB00	ABTERM with subtasking	IEANAB00	IEA0AB00	CY		
IEA0DS	Dispatcher	IEAAIH00	IEAAIH00	AF		
IEA0AT01	ATTACH without subtasking	IEAAAT00	IGC042	EA	42	2
IEA0AT02	ATTACH without subtasking	IEAAAT00	IGC042	EA	42	2
IEA0EF00	Stage 2 Exit Effector	IEAAIH00	IEAAIH00	BN		
IEA0FN00	FINCH	IEAATC00	IGC006	EC,ED		
IEA0TI02	Timer SLIH	IEA0TI02	IEA0TI02	HB,HH		

Figure 55. Supervisor Module Cross Reference (Part 1 of 8)

Entry Point	Name of Routine	Module Name	CSECT	Chart	SVC	Type
IEA0VL00	Validity Check	IEAATA00	IEAATA00			
IEA0XE00	SVC Exit (Type 1)	IEAAIH00	IEAAIH00	AA		
IEEBA1	Communication Task I/O Attention Handler	IEECVCRA	IEEBA1	GB		
IEEBC1PE	Communication Task External Interruption Handler	IEECVCRX	IEEBC1PE	GB		
IEECIR45	Communication Task Wait (non-MCS)	IEECVCTW	IEECIR45	GF		
IEECMAWR	Communication Task Router (MCS)	IEECMAWR	IEECMAWR	GI		
IEECMDOM	Communication Task Delete Operator Message	IEECMDOM	IEECMDOM	GM	87	3
IEECMDSV	Communication Task Device Interface	IEECMDSV	IEECMDSV	GK		
IEECMPM1	1052 Processor, Load 2 (MCS)	IEECMPM1	IGC0207B	GX		
IEECMWSV	Communication Task WTO Processor	IEECMWSV	IEECMWSV	GL		
IEECVCTI	Communication Task Initialization	IEECVCTI	IEECVCTI	GA		
IEECVETA	DIDOCS Options	IGC5A07B	IEECVETA	JP		
IEECVETC	DIDOCS Asynchronous Error	IGC5C07B	IEECVETC	KC		
IEECVETD	DIDOCS Message 1	IGC5D07B	IEECVETD	JH		
IEECVETE	DIDOCS Message 2	IGC5E07B	IEECVETE	JI		
IEECVETF	DIDOCS Light Pen/Cursor	IGC5F07B	IEECVETF	JU		
IEECVETG	DIDOCS Open/Close	IGC5G07B	IEECVETG	KK		
IEECVETH	DIDOCS Model 85 I/O	IGC5H07B	IEECVETH	JG		
IEECVETJ	DIDOCS Roll Mode	IGC5J07B	IEECVETJ	JN		
IEECVETK	DIDOCS Timer Interpreter	IGC5K07B	IEECVETK	JV		
IEECVETP	DIDOCS 2250 I/O 1	IGC5P07B	IEECVETP	JC		
IEECVETQ	DIDOCS 2250 I/O 2	IGC5Q07B	IEECVETQ	JD		
IEECVETR	DIDOCS 2260 I/O 1	IGC5R07B	IEECVETR	JE		
IEECVETZ	DIDOCS Processor 1, Load 2	IGC5Z07B	IEECVETZ	JB		
IEECVET1	DIDOCS Processor 1, Load 1	IGC5107B	IEECVET1	JA		
IEECVET2	DIDOCS Display 1	IGC5207B	IEECVET2	JK		
IEECVET3	DIDOCS Display 2	IGC5307B	IEECVET3	JL		
IEECVET4	DIDOCS Command Routine	IGC5407B	IEECVET4	JO		

Figure 55. Supervisor Module Cross Reference (Part 2 of 8)

Entry Point	Name of Routine	Module Name	CSECT	Chart	SVC	Type
IEECVET6	DIDOCS Delete 1	IGC5607B	IEECVET6	JQ		
IEECVET7	DIDOCS Delete 2	IGC5707B	IEECVET7	JR		
IEECVET8	DIDOCS Delete 3	IGC5807B	IEECVET8	JS		
IEECVET9	DIDOCD Delete 4	IGC5907B	IEECVET9	JT		
IEECVFTA	DIDOCS PFK 1	IGC6A07B	IEECVFTA	KL		
IEECVFTB	DIDOCS PFK 2	IGC6B07B	IEECVFTB	KM		
IEECVFTD	DIDOCS Message 3	IGC6D07B	IEECVFTD	JJ	72	4
IEECVFTG	Cleanup	IGC6G07B	IEECVFTG	KN		
IEECVFTL	Status Display Interface 1	IGC6L07B	IEECVFTL	KD		
IEECVFTM	Status Display Interface 2	IGC6M07B	IEECVFTM	KE		
IEECVFTN	Status Display Interface 3	IGC6N07B	IEECVFTN	KF		
IEECVFTO	Status Display Interface 4	IGC6O07B	IEECVFTO	KG		
IEECVFTP	Status Display Interface 5	IGC6P07B	IEECVFTP	KH		
IEECVFTQ	Status Display Interface 6	IGC6Q07B	IEECVFTQ	KI		
IEECVFTR	2260 I/O 2	IGC6R07B	IEECVFTR	JF	72	4
IEECVFTT	Status Display Interface 7	IGC6T07B	IEECVFTT	KJ		
IEECVFTZ	Processor 0, Load 2	IGC6Z07B	IEECVFTZ	KB		
IEECVFT1	Processor 0, Load 1	IGC6107B	IEECVFT1	KA		
IEECVFT2	Display 3	IGC6207B	IEECVFT2	JM		
IEECVGCI	Graphic Console Initialization	IEECVGCI	IEECVGCI			
IEECVML1	MLWTO (Non-MCS) Load 1	IEECVML1	IEECVML1		35	4
IEECVML2	MLWTO (Non-MCS Load 2	IEECVML2	IEECVML2		35	4
IEECVML3	MLWTO (MCS) Load 1	IEECVML3	IEECVML3		35	4
IEECVML4	MLWTO (Non-MCS) Load 3	IEECVML4	IEECVML4		35	4
IEECVML5	MLWTO (MCS) Load 2	IEECVML5	IEECVML5		35	4
IEECVML6	MLWTO (MCS) Load 3	IEECVML6	IEECVML6		35	4
IEECVOCC	Card Reader Open/Close	IEECVOCC	IGC1I07B			
IEECVOCP	Printer Open/Close	IEECVOCP	IGC2I07B			
IEECVOCX	1052 Open/Close	IEECVOCX	IGC0I07B	GQ		
IEECVPMC	Card Reader Processor (MCS)	IEECMPMC	IGC1107B	GS		
IEECVPMP	Printer Processor (MCS)	IEECMPMP	IGC2107B			

Figure 55. Supervisor Module Cross Reference (Part 3 of 8)

Entry Point	Name of Routine	Module Name	CSECT	Chart	SVC	Type
IEECVPMX	1052 Processor, Load 1 (MCS)	IEECMPMX	IGC0107B	GP		
IEECVPMC	Card Reader Processor (non-MCS)	IEECVPMC	IGC1107B	GR		
IEECVPMP	Printer Processor (non-MCS)	IEECVPMP	IGC2107B			
IEECVPMX	1052 Processor, Load 1 (non-MCS)	IEECVPMX	IGC0107B	GO		
IEECVPM1	1052 Processor, Load 2 (non-MCS)	IEECVPM1	IGC0207B			
IEEC2740	2740 Processor (MCS only)	IEEC2740	IEEC2740	GT		
IEWSZOVR	Basic Nonresident Overlay Supervisor with SEGWT Checking	IEWSXOVR	IEWSZOVR			
IEWSZOVR	Basic Nonresident Overlay Supervisor	IEWSYOVR	IEWSZOVR			
IGCXL07B	Communication Task Console Switch Load 1 (MCS)	IEECLCTX	IGCXL07B	GJ		
IGCXL07B	Communication Task Console Switch (non-MCS)	IEECVCTX	IGCXL07B	GH		
IGCXM07B	Communication Task Console Switch Load 2 (MCS)	IEECMCTX	IGCXM07B	GJ		
IGCXN07B	Communication Task Console Switch Load 3 (MCS)	IEECNCTX	IGCXN07B	GJ		
IGCXO07B	Communication Task Console Switch Load 4 (MCS)	IEECOCTX	IGCXO07B	GJ		
IGC001	WAIT Service	IEAAWT00	IGC001	BF		
IGC002	POST Service	IEAAPT00	IGC002	BG	2	1
IGC003	EXIT (Types 2, 3, and 4 SVC)	IEAATA00	IEAATA00	AC	3	1
IGC004	GETMAIN	IEAAMS	IEAAMS	FA	4	1
IGC005	FREEMAIN	IEAAMS	IEAAMS	FA	5	1
IGC006	LINK	IEAATC00	IGC006	EB	6	2
IGC007	XCTL	IEAATC00	IGC006	EB	7	2
IGC008	LOAD	IEAATC00	IGC006	EB	8	2
IGC009	DELETE	IEAADL00	IGC009	EF	9	2
IGC010	GETMAIN/FREEMAIN (R-form)	IEAAMS	IGC006	FA	10	1
IGC011	TIME	IEA0RT00	IGC011	HA,HE	11	1
IGC012	SYNCH	IEA0SY00	IGC012		12	2
IGC014	SPIE	IEAAPX00	IGC014	BK	14	2
IGC037	Resident Overlay Supervisor	IEWSVOVR	IGC037	IA	37	2

Figure 55. Supervisor Module Cross Reference (Part 4 of 8)

Entry Point	Name of Routine	Module Name	CSECT	Chart	SVC	Type
IGC040	EXTRACT (without subtasking)	IEAAXR00	IGC040	BD	40	1
IGC040	EXTRACT (with subtasking)	IEABXR00	IGC040	BE	40	1
IGC041	IDENTIFY	IEAAID00	IGC041	EE	41	2
IGC042	ATTACH (without subtasking)	IEAAAT00	IGC042	EA	42	2
IGC042	ATTACH (with subtasking)	IEAQAT00	IGC042	BA	42	2
IGC043	Stage 1 Exit Effector	IEAQEF00	IGC043	BM	43	2
IGC044	CHAP	IEAQCH00	IGC044	BC	44	1
IGC046	TTIMER	IEA0ST00	IGC047	HC,HF	46	1
IGC047	STIMER	IEA0ST00	IGC047	HD,HG	47	2
IGC048	DEQ	IEAGENQ1	IGC048	BJ	48	2
IGC048	DEQ (with shared DASD)	IEAGENQ2	IGC048	BJ	48	2
IGC056	ENQ	IEAGENQ1	IGC048	BH	56	2
IGC056	ENQ (with shared DASD)	IEAGENQ2	IGC048	BH	56	2
IGC062	DETACH	IEAGED02	IGC062	BB	62	2
IGC109	Type 3 and 4 ESR Extended SVC Router	IGC116	IGC116	AI	109	2
IGC116	Type 1 ESR Extended SVC Router	IGC116	IGC116	AI	116	1
IGC117	Type 2 ESR Extended SVC Router	IGC116	IGC116	AI	117	2
IGC0A01C	ABEND Load 10	IEANTM0A	IGC0A01C	CJ	13	4
IGC0A05A	ABDUMP 3	IEAAAD0A	IGC0A05A	DD		
IGC0A06C	Checkpoint Preserve 1	IGC0A06A	IGC0A06A	LA	63	4
IGC0B01C	ABEND Load 11 (non-MCS)	IEACTM0B	IGC0B01C		13	4
IGC0B01C	ABEND Load 11 (MCS)	IEANTM0B	IGC0B01C	CK	13	4
IGC0B05A	ABDUMP 7	IEAAAD0B	IGC0B05A	DG		
IGC0C01C	ABEND Load 12	IEANTM0C	IGC0C01C	CK	13	4
IGC0C05A	ABDUMP 11	IEAAAD0C	IGC0C05A	DH		
IGC0D01C	ABEND Load 13	IEANTM0D	IGC0D01C	CK	13	4
IGC0D05A	ABDUMP 2	IEAAD05A	IGC0D05A	DD		
IGC0D06C	Checkpoint Preserve 2	IGC0D06C	IGC0D06C	LA	63	4
IGC0E01C	ABEND Load 14	IEANTM0E	IGC0E01C	CL	13	4
IGC0E05A	ABDUMP 10	IEAAAD0E	IGC0E05A	DH		
IGC0F01C	ABEND Load 15	IEANTM0F	IGC0F01C	CM	13	4

Figure 55. Supervisor Module Cross Reference (Part 5 of 8)

Entry Point	Name of Routine	Module Name	CSECT	Chart	SVC	Type
IGC0F05A	ABDUMP 12	IEAAAD0F	IGC0F05A	DI		
IGC0F06C	Checkmain	IGC0F06C	IGC0F06C	LA	63	4
IGC0G01C	ABEND Load G	IEANTM0G	IGC0G01C	CB	13	4
IGC0G05B	Restart JFCB 1	IGC0G05B	IGC0G05B	LB	52	4
IGC0H01C	ABEND Load H	IEANTM0H	IGC0H01C	CO	13	4
IGC0H05B	Restart Dummy Data Set Processor	IGC0H05B	IGC0H05B	LB	52	4
IGC0I05B	Restart JFCB 2	IGC0I05B	IGC0I05B	LB	52	4
IGC0I07B	1052 Open/Close	IEECVOCX	IGC0I07B	GQ		
IGC0J01C	ABEND Load J	IEANTM0J	IGC0J01C	CP	13	4
IGC0J05B	Restart TCAM Data Set Processor	IGC0J05B	IGC0J05B	LB	52	4
IGC0K05A	ABDUMP 13	IEAAAD0K	IGC0K05A	DJ		
IGC0K05B	Restart Mount/Verify Nondirect Access	IGC0K05B	IGC0K05B	LB	52	4
IGC0L05A	ABDUMP 1	IEAMAD00	IGC0L05A	DC		
IGC0L05B	SYSIN/SYSOUT Nondirect Access Processor	IGC0L05B	IGC0L05B	LB	52	4
IGC0M01C	ABEND Load M	IEANTM0M	IGC0M01C	CN	13	4
IGC0M05A	ABDUMP 14	IEAAAD0M	IGC0M05A	DK		
IGC0M05B	Mount/Verify Direct Access	IGC0M05B	IGC0M05B	LB	52	4
IGC0N05B	SYSIN/SYSOUT Direct Access Positioning 1	IGC0N05B	IGC0N05B	LB	52	4
IGC0N06C	Checkpoint Resume I/O	IGC0N06C	IGC0N06C	LA	63	4
IGC0P05B	Nondirect Access Positioning	IGC0P05B	IGC0P05B	LB	52	4
IGC0Q05B	SYSIN/SYSOUT Direct Access Positioning 2	IGC0Q05B	IGC0Q05B	LB	52	4
IGC0Q06C	Checkpoint Exit	IGC0Q06C	IGC0Q06C	LA	63	4
IGC0R05B	Direct Access Positioning	IGC0R05B	IGC0R05B	LB	52	4
IGC0S05B	Data Set Processor 1A	IGC0S05B	IGC0S05B	LB	52	4
IGC0S06C	Checkpoint Message	IGC0S06C	IGC0S06C	LA	63	4
IGC0T05B	Restart Final Process	IGC0T05B	IGC0T05B	LB	52	4
IGC0U05B	Restart Data Set Processor	IGC0U05B	IGC0U05B	LB	52	4
IGC0V05B	Restart Exit	IGC0V05B	IGC0V05B	LB	52	4

Figure 55. Supervisor Module Cross Reference (Part 6 of 8)

Entry Point	Name of Routine	Module Name	CSECT	Chart	SVC	Type
IGC0W05B	Restart ISAM/BSAM Data Set Processor	IGC0W05B	IGC0W05B	LB	52	4
IGC0Z05A	SVC DUMP 2	IEAAAD0Z	IGC0Z05A	DB		
IGC0001C	ABEND Load 0	IEANTM00	IGC0001C	CA	13	4
IGC0003E	WTO MCS	IEEMFWTO	IGC0003E	GC	35	4
IGC0003E	WTO Non-MCS	IEENFWTO	IGC0003E	GC	35	4
IGC0005A	SVC DUMP 1	IEAAAD0Y	IGC0005A	DA		
IGC00060	STAE	IEAAST00	IGC00060	BL	60	3
IGC0006C	Checkpoint Housekeeping 1	IGC0006C	IGC0006C	LA	63	4
IGC0007B	Communication Task Mini-Router (MCS)	IEECMCTR	IGC0007B		72	4
IGC0007B	Communication Task Router (non-MCS)	IEECVCTR	IGC0007B	GG	72	4
IGC0101C	ABEND Load 1	IEANTM01	IGC0101C	CC	13	4
IGC0103E	Communication Task WTOR Service	IEEVWTOR	IGC0103E	GE		
IGC0105A	ABDUMP 4	IEAAAD01	IGC0105A	DE		
IGC0105B	Restart Housekeeping 1	IGC0105B	IGC0105B	LB	52	4
IGC0106C	Checkpoint Housekeeping 2	IGC0106C	IGC0106C	LA	63	4
IGC0107B	1052 Processor Load 1 (MCS)	IEECMPMX	IGC0107B	GP		
IGC0107B	1052 Processor Load 1 (non-MCS)	IEECVPMX	IGC0107B	GO		
IGC0111C	ASIR 1	IEASTM11	IGC0111C	CQ		4
IGC0201C	ABEND Load 2	IEANTM02	IGC0201C	CD	13	4
IGC0205A	ABDUMP 5	IEAAAD02	IGC0205A	DF		
IGC0205B	Restart Housekeeping 2	IGC0205B	IGC0205B	LB		
IGC0206C	Checkpoint Housekeeping 3	IGC0206C	IGC0206C	LA	63	4
IGC0207B	1052 Processor Load 2	IEECMPM1	IGC0207B	GX		
IGC0211C	ASIR 2	IEASTM12	IGC0211C	CR		4
IGC0221C	DAR 1	IEADTM22	IGC0221C	CU		
IGC0301C	ABEND Load 3	IEANTM03	IGC0301C	CD	13	4
IGC0305A	ABDUMP 6	IEAAAD03	IGC0305A	DF		
IGC0311C	ASIR 3	IEASTM13	IGC0311C	CS		4
IGC0321C	DAR 2	IEADTM23	IGC0321C	CV		
IGC0401C	ABEND Load 4	IEANTM04	IGC0401C	CE	13	4

Figure 55. Supervisor Module Cross Reference (Part 7 of 8)

Entry Point	Name of Routine	Module Name	CSECT	Chart	SVC	Type
IGC0405A	ABDUMP 8	IEAAAD04	IGC0405A	DG		
IGC0411C	ASIR 4	IEASTM14	IGC0411C	CT		4
IGC0421C	DAR 3	IEADTM24	IGC0421C	CW		
IGC0501C	ABEND Load 5 (without subtasking)	IEAMTM05	IGC0501C		13	4
IGC0501C	ABEND Load 5 (with subtasking)	IEANTM05	IGC0501C	CF	13	4
IGC0505A	ABDUMP 9	IEAAAD05	IGC0505A	DH		
IGC0505B	Repmain 1	IGC0505B	IGC0505B	LB	52	4
IGC0506C	Checkpoint I/O	IGC0506C	IGC0506C	LA	63	4
IGC0521C	DAR 4	IEANTM25	IGC0521C	CX		
IGC0601C	ABEND Load 6	IEANTM06	IGC0601C	CG	13	4
IGC0605B	Repmain 2	IGC0605B	IGC0605B	LB	52	4
IGC0701C	ABEND Load 7	IEANTM07	IGC0701C	CH	13	4
IGC0801C	ABEND Load 8	IEANTM08	IGC0801C	CH	13	4
IGC0901C	ABEND Load 9	IEANTM09	IGC0901C	CI	13	4
IGC0907B	Communication Task NIP Message Buffer Writer (MCS)	IEECMWTL	IGC0907B	GN		
IGC1I07B	2540 Open/Close	IEECVOCC	IGC1I07B		72	4
IGC1107B	2540 Processor (MCS)	IEECMPMC	IGC1107B	GS	72	4
IGC1107B	2540 Processor (non-MCS)	IEECVPMC	IGC1107B	GR	72	4
IGC2I07B	Printer Open/Close	IEECVOCP	IGC2I07B		72	4
IGC2107B	Printer Processor (MCS)	IEECMPMP	IGC2107B		72	4
IGC2107B	Printer Processor (non-MCS)	IEECVPMP	IGC2107B		72	4
SVE	SVC SLIH	IEAATA00	IEAATA00	AB		
TASKSWIT	Task Switch Determination	IEAATA00	IEAATA00	AB		
XFINCH	FINCH	IEAATC00	IGC006	EC,ED		

Figure 55. Supervisor Module Cross Reference (Part 8 of 8)

This section describes the format and contents of the control blocks, parameter lists, and other data areas used in MFT. The offsets of particular bytes from the first byte (byte 0) are given in decimal, with the hexadecimal displacement in parentheses.

ABDUMP PARAMETER LIST

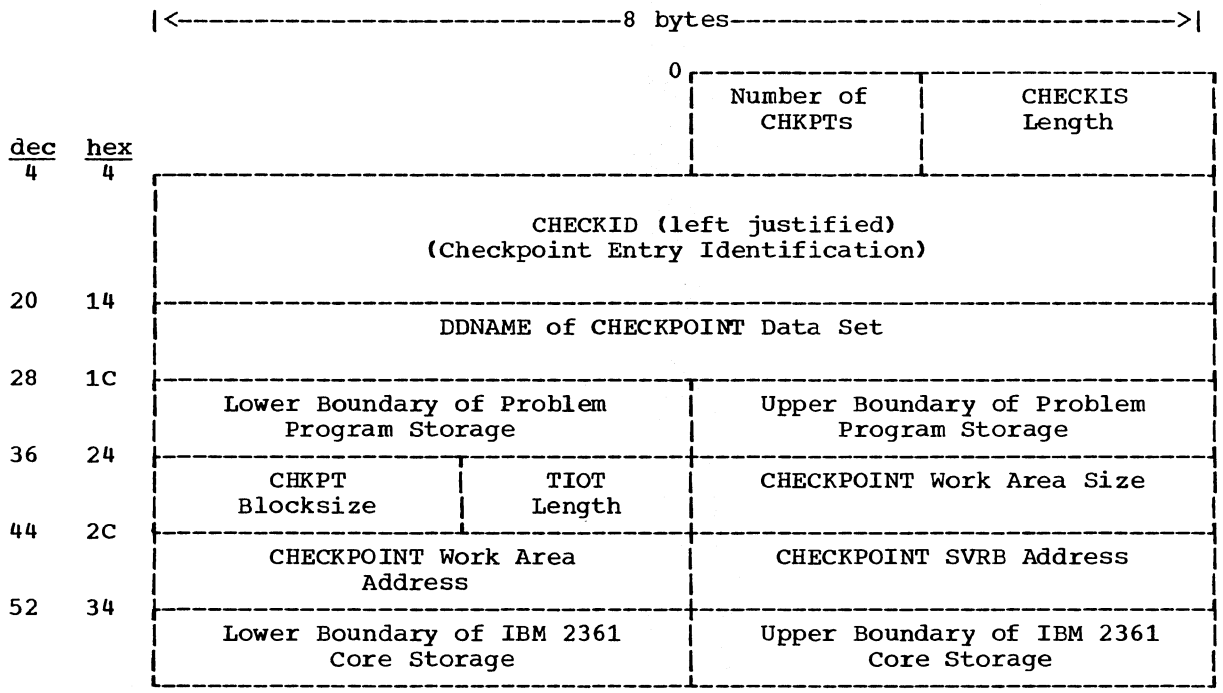
Offset	Bytes and Bit Pattern	Field Name	Field Description
0 (0)	1		Identification number.
1 (1)	1		Reserved.
2 (2)	2		Option Flags.
	First Byte		
	0...	PFABEND	Requested by ABEND.
	1...		Requested by SNAP.
	.1..	PFTCB	TCB address is given.
	..1.	PFSUPDAT	Display all supervisor data.
	...1	PFTRACE	Display trace table (if possible).
	.... 1...	PFNUC	Display the nucleus.
	.... .1..	PFSNAP	Snapshot list is given.
	.... ..1.	PFID	ID is given.
	x	PFQCB	Reserved.
3 (3)	Second Byte		
	1...	PFSAVE	Save area (see next flag).
	.0..	PFSAVE2	Display entire save area.
	.1..		Display headings only.
	..1.	PFREGS	Display registers on entry to ABEND or SNAP.
	...1	PFLPA	Display modules in the resident reenterable module area used by the task being dumped.
	.... 1...	PFJPA	Display all main storage assigned to the job step task.
	.... .1..	PFPSW	Display PSW on entry to ABEND or SNAP.
	.... ..1.	PFSPAIL	Display all main storage assigned to the job step task.
	x		Reserved.
4 (4)	1		Reserved.
5 (5)	3		Address of DCB.
8 (8)	1		
	0...		SNAP request.
	1...		SDUMP request.
9 (9)	3		Address of TCB.
12 (C)	1		Reserved.
13 (D)	3		Address of storage list.

BOUNDARY BOX

SQA boundary box is in the master scheduler resident data area at offset X'64'. Partition boundary box is pointed to by TCBMSS field.

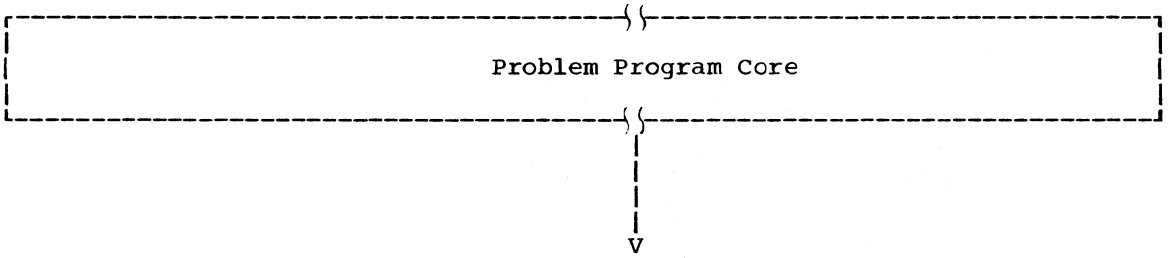
Offset	Bytes and Bit Pattern	Field Name	Field Description
-16 (-10)	4	QNXTREQ	Address of next QNXTREQ on Transient Area Request Queue when this task has a request pending.
-12 (-C)	4	QSVRB	Address of SVRB for this transient area request.
-8 (-8)	8		Reserved for ABEND appendages.
0 (0)	1		Reserved for LCS flags.
1 (1)	3		For job step task: address of hierarchy 0 FQE for highest free area. For subtask: address of last GQE created for subtask.
The following fields (offsets 4-23) exist only if the boundary box is for a job step task:			
4 (4)	4		Address of highest main storage address in hierarchy 0.
8 (8)	4		Address of lowest main storage address in hierarchy 0.
The following fields are included only with Main Storage Hierarchy Support:			
12 (C)	4		Address of hierarchy 1 FQE for highest free area.
16 (10)	4		Address of highest main storage address in hierarchy 1.
20 (14)	4		Address of lowest main storage address in hierarchy 1.

CHECKPOINT HEADER RECORD (CHR)



Note: The CHR is 400 bytes long and is padded with ones.

CORE IMAGE RECORD (CIR)



Direct copy of problem program storage, from the highest to the lowest address.

Blocksize 1. Is specified by the caller, or

2. If not specified, is the maximum for the device type.

COMMUNICATION VECTOR TABLE

Pointed to by main storage location 16.

Offset		Bytes and Bit Pattern	Field Name	Field Description
-6	(-6)	2	CVTMDL	The model number of the CPU in decimal. Each digit is represented by four bits; the significant digits are right justified within the halfword.
-4	(-4)	4	CVTRELNO	Release number in EBCDIC.
0	(0)	4	CVTTCBP	Address of a doubleword field (IEATCBP) within the Dispatcher, the first containing the next-to-be dispatched TCB address, the second containing the last-to-be-dispatched (current) TCB address. During task execution, both words are identical. When in a wait state, the first word is set to 0 until the waiting is over; then both words are once again identical. During task switching, the words may not be identical.
4	(4)	4	CVT0EF00	Address of Stage 2 Exit Effector.
8	(8)	4	CVTLINK	Address of the DCB for the SYS1.LINKLIB data set.
12	(C)	4	CVTJOB	Address of work queue control blocks used by the job scheduler.
16	(10)	4	CVTBUF	Address of the buffer of the resident console interruption routine.
20	(14)	4	CVTXAPG	Address of the I/O supervisor appendage table.
24	(18)	4	CVT0VL00	Address of entry point of the task supervisor's address validity check routine.
28	(1c)	4	CVTPCNVT	Entry point of the routine which converts a relative track address (TTR) to an absolute track address (MBBCCHHR).
32	(20)	4	CVTPRLTV	Entry point of the routine that converts an absolute track address (MBBCCHHR) to a relative track address (TTR).
36	(24)	4	CVTILK1	Address of the channel and control unit portion of the UCB lookup table.
40	(28)	4	CVTILK2	Address of the UCB address-list portion of the UCB lookup table.
44	(2c)	4	CVTXTLER	Entry point of an XCTL routine that brings system error routines into the error transient area.
48	(30)	4	CVTSYSAD	Address of the system residence volume entry in the UCB table.
52	(34)	4	CVTBTERM	Entry point of the ABTERM routine.
56	(38)	4	CVTDATE	Current date in packed decimal.
60	(3C)	4	CVTMSLT	Address of master scheduler resident linkage table. Note: The master scheduler resident data area consists of a data area followed by the master common area. The master common area is currently at the same offset (136 decimal, X'88') from the beginning of both the master resident core and the master

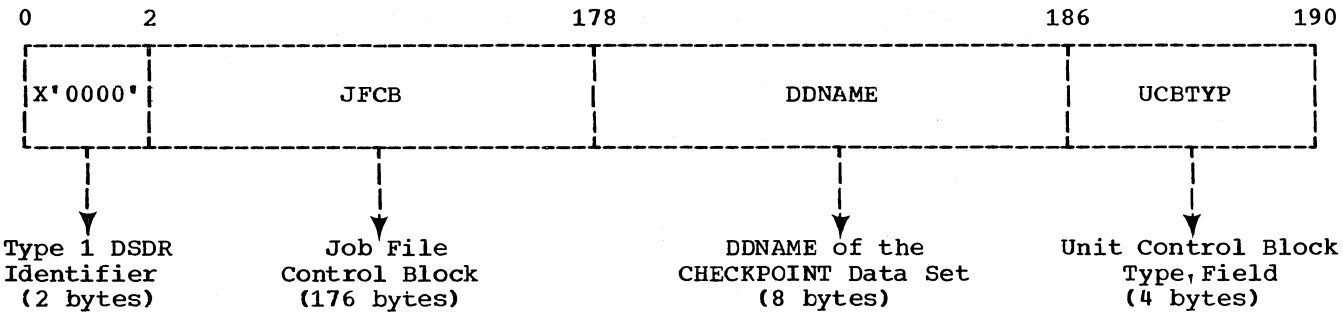
Offset	Bytes and Bit Pattern	Field Name	Field Description
60 (3C) (cont.)			scheduler resident data area, but its offset from the beginning of the master scheduler resident data area may change in future releases. Therefore, all references to the master common area should be made using the address in CVTMSLT, and all references to the data area of the master scheduler resident data area should be made using the address in CVTMSER, at offset 148 (X'94').
64 (40)	4	CVTZDTAB	Address of the I/O device characteristic table.
68 (44)	4	CVTXITP	Address of the error interpreter routine.
72 (48)	4	CVTDAR	Address of the I/O control block complex needed by ABEND's Damage Assessment Routine (DAR). If zeros, SYS1.DUMP data set has not been defined.
76 (4C)	4	CVT0FN00	Entry point of the FINCH routine.
80 (50)	2	CVTEXIT	An SVC 3 instruction (EXIT).
82 (52)	2	CVTBRET	A BR 14 instruction (used by data management routines).
84 (54)	4	CVTSVDCB	Address of the DCB for the SYS1.SVCLIB data set.
88 (58)	4	CVTTPC	Address of the 6-hour pseudo clock (SHPC), used by timer supervisor routines.
92 (5C)	4	CVTPBLDL	Address of BAL entry point to the BLDL routine.
96 (60)	4	CVTSJQ	Address of the selected job queue.
100 (64)	4	CVTCUCB	Address of the table that contains the current console's UCB addresses (Unit Control Module).
104 (68)	4	CVTQTE00	Address of the timer enqueue routine.
108 (6C)	4	CVTQTD00	Address of the timer dequeue routine.
112 (70)	4	CVTSTB	Address of the I/O device statistics table.
116 (74)	1	CVTDCB	System Configuration.
	0001 0000	CVT4MS1	MVT - Uniprocessing
	0001 0100	CVT4MPS	MVT - Multiprocessing
	0010 0000	CVT2SPS	MFT
	0100 0000	CVT1SSS	PCP
117 (75)	3	CVTDCBA	Address of the DCB for the SYS1.LOGREC (outboard recorder) data set for system environment recording.
120 (78)	4	CVTIOQET	Address of request element table.
124 (7C)	4	CVTIXAVL	Address of the I/O supervisor's freelist pointer (which contains the address of the next request element).
128 (80)	4	CVTNUCB	Lowest address not in the nucleus. If the protection option is specified in the system generation process, this is a 2K boundary. If the protection option is not specified in the system generation process, this is a doubleword boundary.
132 (84)	4	CVTFBOSV	Address of program fetch routine.
136 (88)	4	CVT0DS	Address of entry point of the Dispatcher.

Offset		Bytes and Bit Pattern	Field Name	Field Description
140	(8C)	4	CVTILCH	Address of the logical channel word table.
144	(90)	4	CVTIERLC	Address of the asynchronous exit queue.
148	(94)	4	CVTMSER	Address of master scheduler resident data area. Note: This field should be used to address the data area, but not the master common area, of the master scheduler resident data area. See the note under CVTMSLT, at offset 60 (X'3C').
152	(98)	4	CVT0PT01	Address of branch entry point of POST routine.
156	(9C)	4	CVTTRMTB	Address of terminal table present in systems that have QTAM routines.
160	(A0)	4	CVTHEAD	Address of the highest priority TCB in the queue of TCBs for executing tasks.
164	(A4)	4	CVTMZ00	Highest storage address for this machine (machine size).
168	(A8)	4	CVT1EF00	Reserved.
172	(AC)	4	CVTQOCR	Graphics interface task (GFX) field. If GFX is active: address of seventh word of GFX parameter list. If GFX is not active: 0 (Four bytes of binary 0's).
176	(BO)	4	CVTQMWR	Address of system output communications-data-area (CDA) which is stored on an external device used by the queue manager.
180	(B4)	2	CVTSNCTR	Serial number counter. Counter for assigning serial numbers to nonspecific, unlabeled magnetic tape volumes. (A binary number forming the XXX part of the volume serial number of the form LXXXYY.)
182	(B6)	1	CVTOPTA	
		1...	CVTCCH	Channel Check Handler (CCH) option is present in the system.
		.1..	CVTAPR	Automatic Path Retry selected.
		..1.	CVTDDR	Dynamic Device Reconfiguration selected.
		...1	CVTNIP	NIP is executing.
		.... 1...	CVTATTCH	Subtasking option (ATTACH) selected.
		.... .1..	CVTHIAR	Main Storage Hierarchy Support operative.
		.... ..1.	CVTASCII	ASCII option selected.
		1	CVTXPFP	Extended Precision Floating Point selected.
183	(B7)	1	CVTOPTB	
		1...	CVTPROT	Store Protection selected.
		..1.	CVTTOD	CPU has Time-of-Day Clock.
184	(B8)	4	CVTQCDSR	Without Link Library Option: Reserved. With Link Library Option: Address of the reenterable load module queue search routine.
188	(BC)	4	CVTQLPAQ	Without Link Library Option: Reserved. With Link Library Option: Address of Reenterable Load Module Queue.
192	(C0)	4		Reserved.
196	(C4)	4	CVTSMCA	Address of the System Management Control Area if the System Management Facilities option is present in the system.

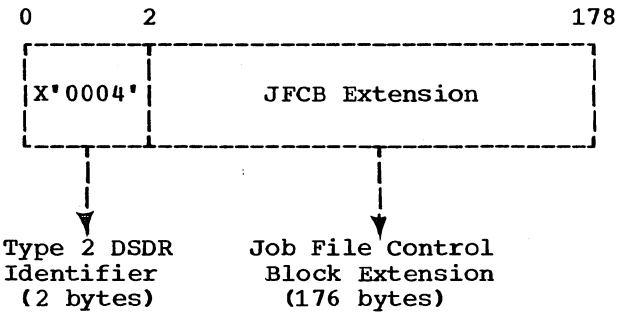
Offset		Bytes and Bit Pattern	Field Name	Field Description
200	(C8)	4		Reserved.
204	(CC)	4	CVTUSER	A field available to the user of System/360 Operating System.
208	(D0)	4	CVTMDLDS	Reserved.
212	(D4)	2	CVTQABST	An SVC 13 instruction (ABEND).
214	(D6)	2	CVTLNKSC	An SVC 6 instruction (LINK).
216	(D8)	4	CVTTSCE	Address of the time-slice control element (TSCE).
220	(DC)	4	CVTPATCH	Address of 200-byte patch area in nucleus.
224	(E0)	4	CVTRMS	Address of machine status block for RMS.
Bytes 228 through 231 are reserved.				
232	(E8)	4	CVT0SCR1	Address of the sector calculation routine for rotational position sensing (RPS).
236	(EC)	4	CVTGTF	Status flags and the address of the Class Mask Table for the Generalized Trace Facility (GTF).
236	(EC)	First Byte	CVTGTFST	Status flags for GTF.
		00..	CVTGTFS	GTF not active.
		01..		GTF starting.
		10..		GTF stopping.
		11..		GTF active.
		..1.	CVTSTATE	GTF is in control.
		...0	CVTMODE	Trace data is to be written in main storage (MODE=INT specified).
		...1		Trace data is to be written to auxiliary storage (MODE=EXT specified).
		.... 1...	CVTFORM	Trace data is to be formatted on abnormal termination.
		.... 0...		ABDUMP is to bypass formatting of trace data.
		.... .1..		CVTUSR - User requested trace data is to be included in the trace data set.
		.... ..1.		CVTMCTYP - The System/370 MONITOR CALL instruction is valid.
237	(ED)	3	CVTCMT	Address of the Class Mask Table (CMT).
240	(F0)	1	CVTTCMFG	TCAM flags.
		1...	CVTTCRDY	TCAM is ready to accept users.
		0...		TCAM is not ready or has abnormally terminated.
241	(F1)	3	CVTAQAVT	Address of first word of TCAM dispatcher. This word contains the address of the AVT.
244	(F4)	1	CVTTSKS	The maximum allowable number of entries in the TCB address table.
245	(F5)	3	CVTTAT	Address of the first entry in the TCB address table. The first entry is for Partition 0.
248	(F8)	1	CVTSYST	The number of entries in the TCB address table, for TCBs created during system generation.
Bytes 249 through 259 are reserved.				
260	(104)	4	CVTPURG	Address of the Subsystem Purge routine.
268			CVTQMSGA	Address of Message Information List module (IEAQMSGS).

DATA SET DESCRIPTOR RECORDS

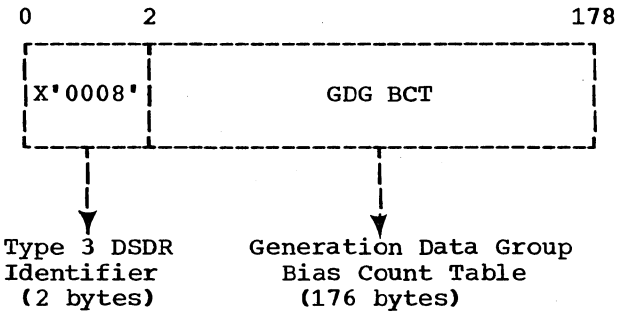
Type 1 DSDR



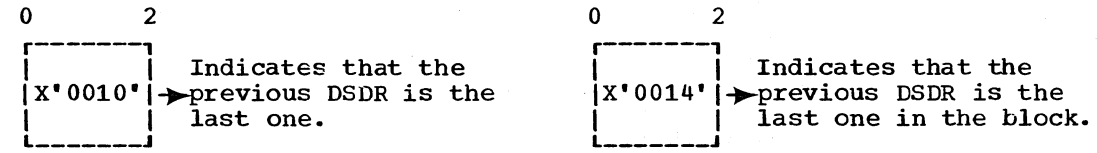
Type 2 DSDR



Type 3 DSDR



Special Identifiers



RESIDENT DISPLAY CONTROL MODULE (RDCM)

Pointed to by field UCMXB in UCM entry.

Offset		Bytes and Bit Pattern	Field Name	Field Description
0	(0)	4	DCMADTRN	Address of the main storage area assigned to the TDCM.
4	(4)	1	DCMFNDEX	Display console identification number assigned by the graphic console unit.
5	(5)	1	DCMRFLGS	Contains flags indicating the status of the display console as follows:
		1...		DCM read in process.
		.1..		Console uses transient DCMs.
		..1.		DCM write in process.
		...1		DOM must be tries.
		.... 1...		DCM was used by NIP.
		.... ..1.		First read of DCM.
		1		Processor 0 routing flag.
6	(6)	2	DCMLEN	Length of the transient portion of the DCM.
8	(8)	4	DCMADKP	Address of the routed K command parameter list.
12	(C)	1	DCMTOPAR	Area ID of the topmost display area defined for the screen.
13	(D)	1	DCMTOPDS	Area ID of the topmost display area that is in use (contains a display).
14	(E)	1	DCMRESTA	Attention and open flags:
		1...		Attention status has been saved
		.1..		Open status has been saved
15	(F)	1	DCMDEVTY	Device type flags:
		1...		2260
		.1..		2250
16	(10)	4	DCMADSDS	Pointer to the first screen area control block (this field contains zeros if no display areas are defined for the screen).
20	(14)	1	DCMRMS	Recovery management system (RMS) information.
21	(15)	3	DCMADRMS	Address of recovery management system (RMS) channel programs.
24	(18)	4	DCMWLAST	Address of the last console queue entry to be displayed.
28	(1C)	2	DCMRMSAL	Number of lines in the message area.
30	(1E)	2		Reserved.
32	(20)	4	DCMMSGSV	Address of a list of saved NIP messages.
36	(24)	4	DCMADPFK	Address of the resident program function keyboard (PFK) area.
40	(28)	2	DCMINTVL	Time interval for the DCM.
42	(2A)	2	DCMTMCTR	Time counter for the DCM.
44	(2C)	1	DCMR2FLG	Information flags conceiving the timer:
		1...		Full screen
		.1..		Unviewable message displayed

Offset	Bytes and Bit Pattern	Field Name	Field Description
44 (2D)	..1.		Timer flag
(cont.)	...1		Ready to roll
	.... 1...		Pending delete request
	.... .1..		Entry was from options
	.... ..1.		Timer elapsed
	1		TDCM is in core
45 (2D)	1	DCMR3FLG	
	1...		Changing status of output-only console
	.1..		Entry for K VARY
	..1.		Close in process
	...1		Asynchronous error message on screen

Screen Area Control Block (SACB)

The following fields comprise the screen area control block (SACB) portion of the RDCM. One complete screen area control block is created for each display area defined for the screen; if no display areas are defined, no screen control blocks are created. SACBs may be contiguous with the RDCM or may be chained to the RDCM by pointers.

Offset	Bytes and Bit Pattern	Field Name	Field Description
48 (30)	2	DCMPLN	Length of the display area in bytes.
50 (32)	2	DCMPLNPR	Length of the screen area control block prefix in bytes.
52 (34)	4	DCMACBNX	Address of the next screen area control block.
56 (38)	1	DCMAID	Alphabetic identifier assigned to the display area.
57 (39)	1	DCMASACB	Screen area control block flags with the following meanings:
	1...		SACB in use
	.1..		SACB space was obtained by GETMAIN
58 (3A)	2	DCMALN	Length of the display area in bytes.
60 (3C)	1	DCMATOP	Line number of the top line of the display area.
61 (3D)	1	DCMAROW	Line number of the line to be written next.
62 (3E)	2	DCMAFR	Frame number of the frame being displayed.
64 (40)	4	DCMAMJWQ	Address of the console queue entry for the major WQE.
68 (44)	4	DCMAMIN	Address of the console queue entry for the minor WQE being written.
72 (48)	4	DCMATIME	Time indicator written in the control line of the display being written.
76 (4C)	2	DCMAMT	Message type flags:
	..1.		MONITOR A
78 (4E)	1	DCMAFLG1	Flags pertaining to the display area status:
	1...		Display coming to area
	.1..		Display in area
	..1.		End of display on screen
	...1		Framing in progress
	.... 1...		Frame full
	.... .1..		Blanking to be done

Offset	Bytes and Bit Pattern	Field Name	Field Description
79 (4F)	1	DCMAFLG2	Flags pertaining to the display area status:
	1...		Saved pointer to last minor output
	.1..		Write control line
	..1.		Rewrite control line
	...1		Major WQE has been found
80 (50)	1	DCMADFLG	Flags pertaining to the display in the display area:
	1...		Dynamic display
	.1..		Dynamic display in hold mode
	..1.		Dynamic display with control line in SIB
81 (51)	3		Reserved.

TRANSIENT DISPLAY CONTROL MODULE (TDCM)

Pointed to by Resident DCM

The TDCM contains two sections: the device-independent section and the device-dependent section. The names and displacements of all fields in the device-independent section are identical for all devices, but the contents of the fields may vary depending on device type. The device-dependent section varies according to the device it is supporting.

Offset		Bytes and Bit Pattern	Field Name	Field Description
0	(0)	2	DCM length	Length of the TDCM in bytes.
2	(2)	2		Reserved.
4	(4)	1	DCMFLG1	Flags pertaining to the DCM.
		...1	DCMQUE	Queue this DCM for write.
		.... ..1.	DCMOUTPT	DCM updated for output-only.
5	(5)	3		Reserved.
8	(8)	4	DCMWTINT	DCMWTBUF initial value.
12	(3)	4		Reserved.
16	(10)	4	DCMPACK	Area used for packing decimal digits.
20	(14)	4	DCMCVBIN	Area used for convert to binary.
24	(18)	1	DCMTIMES	Timer routine flags.
		1...	DCMTIMER	Timer elapsed for this display.
		.1..	DCMOPTTI	Options to Timer-Interpreter.
		..1.	DCMGROLL	Display ready to roll.
		...1	DCMOTTMM	Options or T/I to message module.
		.... 1...	DCMSTTMR	STIMER should be issued.
		.... .1..	DCMTASYN	Timer set for asynchronous error message.
		.... ..1.	DCMOCTTI	OPEN/CLOSE to T/I routine.
		1	DCMRMTTI	Roll mode to T/I routine.
24	(19)	1		Reserved.
26	(1A)	2	DCMELGN	Entry area last character pointer.
28	(1C)	4	DCMBUFAD	Address of the buffer address table.
32	(20)	4	DCMDOMPK	Address of the first DOM Number.
36	(24)	4	DCMAMTAB	Address of the first screen control table entry.
40	(28)	4	DCMADSEC	Address of the first secondary screen control table entry.
44	(2C)	4	DCMADDRL	Address of the last screen control table entry.
48	(30)	4	DCMASCRN	Address of the screen image buffer.
52	(34)	4	DCMLSCRN	Address of the last line of the screen image buffer.
56	(38)	4	DCMWTBUF	Address of the last line of the message area in the screen image buffer.
60	(3C)	4	DCMAINS	Address of the insrruction line in the screen image buffer.
64	(40)	4	DCMAENTR	Address of the entry area in the screen image buffer.

Offset		Bytes and Bit Pattern	Field Name	Field Description
68	(44)	4	DCMAWARN	Address of the warning line in the screen image buffer.
72	(48)	4	DCMADCHP	Address of the channel program area.
76	(4C)	4	DCMPFKLN	Reserved.
80	(50)	4	DCMCXSVE	CXSA save area.
84	(54)	4	DCMADOPN	Address of the command operand.
88	(58)	20	DCMDSAV	Command save area and work area.
108	(6C)	2	DCMINLGN	Input length.
110	(6E)	2	DCMMCSFL	Space for MCS flags.
112	(70)	128	DCMINPUT	Space for input message text.
240	(F0)	8	DCMRQDEL	Buffer for pending delete requests.
248	(F8)	2	DCMLGNTH	Length of a line.
250	(FA)	2	DCMBAINC	Position of cursor in screen image buffer.
252	(FC)	2	DCMIRCTR	INTERVENTION REQUESTED message counter.
254	(FE)	2	DCMBADLN	Location in buffer to begin writing message.
256	(100)	2	DCMBYTCT	Number of bytes to write.
258	(102)	2	DCMADNUM	The line number to be assigned to the next line.
260	(104)	2	DCMAXLGN	Maximum line length.
262	(106)	2	DCMMSGAL	Number of lines in the message area.
264	(108)	2	DCMRMINC	RMI information.
266	(10A)	2	DCMSCTCN	Length of one SCT entry.
268	(10C)	2	DCMCORLN	Length of DCM line in main storage.
270	(10E)	2		Reserved.
272	(110)	1	DCMPFKNM	Number of the PFK key being processed.
273	(111)	1	DCMPFKKN	Number of the PFK key, in a list of PFK keys, that is being processed.
274	(112)	2	DCMDEL	Contains the current value of the DEL parameter of the CONTROL S command (Y or N).
276	(114)	1	DCMCON	Contains the current value of the CON parameter of the CONTROL S command (Y or N).
277	(115)	1	DCMSEG	Contains the current value of the SEG parameter of the CONTROL S command (numerical value).
278	(116)	1	DCMDL	Contains the current value of the DL parameter of the CONTROL S command (numerical value).
279	(117)	1	DCMRNUM	Contains the current value of the RNUM parameter of the CONTROL S command (numerical value).

Offset		Bytes and Bit Pattern	Field Name	Field Description
280	(118)	2	DCMRTME	Contains the current value of the RTME parameter of the CONTROL S command (numerical value).
282	(11A)	1	DCMSEGDF	The default value for SEG.
283	(11B)	1	DCMRNUMD	The default value for RNUM.
284	(11C)	2	DCMRTMED	The default value for RTME.
286	(11E)	1	DCMASKEN	The ENTER mask.
287	(11F)	1	DCMASKCN	The CANCEL mask.
288	(120)	1	DCMASKCR	The cursor mask.
289	(121)	1	DCMASKLP	The light pen mask.
290	(122)	1	DCMASKPF	The PFK mask.
291	(123)	1	DCMOPTST	Screen control option flags.
		1...	DCMOPTVR	Delete verification (Y=1; N=0).
		.1..	DCMOPTAD	Automatic deletion (Y=1; N=0).
		..1.	DCMOPTSG	Default SEG specified.
		...1	DCMOPRLL	Roll mode specified.
292	(124)	1	DCMCS	Open/Close request.
		1...	DCMCSC	Close requested.
		.1..	DCMCSO	Open requested.
293	(125)		DCMUTILT	Reserved.
294	(126)	1	DCMDSTAT	Current display status flags.
		1...	DCMDSTFL	Full screen.
		.1..	DCMUNVW	UNVIEWABLE MESSAGE written to screen.
		..1.	DCMDSTNM	Messages are numbered.
		...1	DCMDSTNH	Messages numbered - HOLD option.
		.... 1...	DCMDSINR	Intervention requested deletion tried.
		.... .1..	DCMDSAUT	Automatic deletion tried.
295	(127)	1	DCMMCSST	MCS Interface flags.
		1...	DCMDUSE	Status Display Support.
		.1..	DCMNOHRD	NO HARDCOPY message written.
		.... ..1.	DCMOOMSS	Message stream entry.
		1	DCMOOSDS	Status display entry.
296	(128)	1	DCMIOUNQ	Device I/O information flags.
		1...	DCMIO226	(2260) RMI performed.
		.1..	DCMRPCUR	(2260) Advance cursor to blanks.
		..1.	DCMFRSCN	(2260) Put output in hold mode.
		...1	DCMRDARM	(2250) Perform read after RMI.
		.... 1...	DCMW2250	(2250) Device is a 2250.
		.... .1..	DCMINNOR	(2250) Normal instruction line.
		.... ..1.	DCMINERR	(2250) Error instruction line.
297	(129)	1	DCMIOCM1	I/O Communications byte 1.
		1...	DCMDORMI	Issue RMI.
		.1..	DCMSOUND	Sound alarm.
		..1.	DCMWRWRN	Write warning line.
		...1	DCMWRMSG	Write full message area.
		.... 1...	DCMWRPAR	Write partial message area.
		.... .1..	DCMWRINS	Write instruction line.
		.... ..1.	DCMWRENT	Write entry area.
		1	DCMINSC	Insert cursor.

Offset		Bytes and Bit Pattern	Field Name	Field Description
298	(12A)	1	DCMIOCM2	I/O Communications byte 2.
		1...	DCMBLENT	Blank entry area.
		.1..	DCMBLWRL	Blank left half of warning line.
		..1.	DCMBLWRR	Blank right half of warning line.
		...1	DCMINSSH	Initialize and shift instruction line.
		.... 1...	DCMWINFD	Write informational display.
		.... .1..	DCMERASE	Perform erase.
		.... ..1.	DCMIOCRD	Perform read.
		1	DCMWRASY	Write asynchronous error message to mid-screen.
299	(12B)	1	CCMIOCM3	I/O Communications byte 3.
		1...	DCMOPRMI	RMI after OPEN to unlock keyboard.
		.1..	DCMSSRG	Suppress start regeneration.
		..1.		Reserved.
		...1	DCMWRPFK	Write PFK area.
		.... 1...	DCMPFKAT	PFK attention.
		.... .1..	DCMRDPFK	PFK area read.
		.... ..1.	DCMACPFK	Turn active PFK lights on.
		1	DCMLTPFK	Turn all PFK lights on.
300	(12C)	1	DCMLINEN	Line numbers to begin write.
301	(12D)	1	DCMCULNO	Line in entry area to insert cursor.
302	(12E)	1	DCMPOSCU	Position to insert cursor.
303	(12F)	1	DCMASYNC	Asynchronous error retry flags.
		1...	DCMASCAN	Asynchronous error message on screen.
		.1..	DCMASDA	Retry bit.
		..1.	DCMASIN	Retry bit.
		...1	DCMASBA	Retry bit.
		.... 1...	DCMASLOG	Log asynchronous error.
304	(130)	1	DCMCOM1	Communications byte 1.
		1...	DCMLPENT	Enter by light pen or cursor.
		.1..	DCMIOPRD	Read performed.
		..1.	DCMCOMRM	RMI performed.
		...1	DCMCONAU	Perform automatic deletion.
		.... 1...	DCMCOMRD	Perform regular delete.
		.... .1..	DCMCOMNM	Number messages.
		.... ..1.		Reserved.
		1	DCMCANCL	Indicate CANCEL to Command routine.
305	(131)	1	DCMCOM2	Communications byte 2.
		1...	DCMCM2I	Input to be processed.
		.1..	DCMSPLIT	Message to be split.
		..1.	DCMCOMAR	Accepted reply.
		...1	DCMCLOSE	Cleanup for close.
		.... 1...	DCMERPF	Erase performed; close device.
		.... .1..	DCMCMIN5	Return to Interface 5 for blanking.
		.... ..1.	DCMBLNK	Blanking required.
		1	DCMAE	Cleanup for asynchronous error.
306	(132)	1	DCMCOM3	Communications byte 3.
		1...	DCMCDSP3	Display 3 work complete.
		.1..	DCMRTPFK	Return to PFK routine.
		..1.	DCMVLPFK	Verifying last command.
		...1	DCMXINT1	Entry for Interface 1 routine.
		.... 1...	DCMOLUNV	Out-of-line message caused UNVIEWABLE MESSAGE.
		.... .1..	DCMPFKWR	Write PFK updates to library.
		.... ..1.		Reserved.
		1	DCMCMIN7	Return to Interface 7 for blanking.

Offset		Bytes and Bit Pattern	Field Name	Field Description
307	(133)	1	DCMCMSG1	Message Module Communications byte 1.
		1...	DCMMSGWT	Move in MESSAGE WAITING.
		.1..	DCMUNMSG	Move in UNVIEWABLE MESSAGE.
		...1	DCMCHOPT	Move in CHANGE OPTIONS.
		.... 1...	DCMELONG	Move in ENTRY TOO LONG.
		.... .1..	DCMWRCDL	Move in CON=N, DEL=Y.
		.... ..1.	DCMDELNT	Move in DEL UNCHANGED, NO TIMER.
		1	DCMMNHRD	Move in NO HARDCOPY.
308	(134)	1	DCMCMSG2	Message Module Communications byte 2.
		1...	DCMDLREQ	Move in DELETION REQUESTED.
		.1..	DCMRQINC	Move in REQUEST INCONSISTENT.
		..1.	DCMMSGCR	Move in INVALID CURSOR OPERATION.
		...1	DCMINVOP	Move in INVALID OPERAND.
		.... 1...	DCMCILLP	Move in ILLEGAL LIGHT PEN OPERATION.
		.... .1..	DCMDELRI	Move in DELETE REQUEST INCONSISTENT.
		.... ..1.	DCMASYRT	Move in ASYNCHRONOUS ERROR RETRYABLE.
		1	DCMASYCD	Move in ASYNCHRONOUS ERROR MAY BE RETRYABLE.
309	(135)	1	DCMCMSG3	Message Module Communications byte 3.
		1...	DCMCNRLL	Move in ROLL MODE.
		.1..	DCMCDLR1	Move in NO DELETABLE MESSAGES.
		..1.	DCMCDLR2	Move in INVALID RANGE.
		...1	DCMCDLR3	SEG equal to zero.
		.... 1...	DCMCDLR4	Display not on screen.
		.... .1..	DCMCDLR5	Invalid operand.
		.... ..1.	DCMNHCIN	Move NO HARDCOPY message to instruction line.
		1	DCMDTBSY	COMMAND REJECTED--TASK BUSY.
310	(136)	1	DCMCMSG4	Message Module Communications byte 4.
		1...	DCMPFKNA	Move in PFK NOT ALLOCATED.
		.1..	DCMPFKND	Move in PFK NOT DEFINED.
		..1.	DCMPFKNO	Move in NO PFK ALLOCATION.
		...1	DCMPFKIP	Move in PFK IN PROCESS.
311	(137)	1	DCMSVC34	SVC 34 Communications byte.
		1...	DCMMYCMD	Command to be handled by this console.
		.1..	DCMINVLD	Invalid K command.
		..1.	DCMTYPE1	K command is not routable.
312	(138)			Reserved.
313	(139)	1	DCMIORTN	Indicates which I/O routine handles I/O for the console.
314	(13A)	1	DCMTEST	Reserved for testing.
316	(13C)	2	DCMBASRT	Location in buffer for start of the channel program regeneration orders.
318	(13E)	2	DCMBAMI	Location of first message line for use by the channel program.
320	(140)	2	DCMBAPFK	Location of the PFK display line in the buffer for use by the channel program.
322	(142)	2	DCMBAINS	Location of the instruction line in the buffer for use by the channel program.
324	(144)	2	DCMBAENT	Location of the entry area in the buffer for use by the channel program.
326	(146)	2	DCMBAWRN	Location of the warning line in the buffer for use by the channel program.

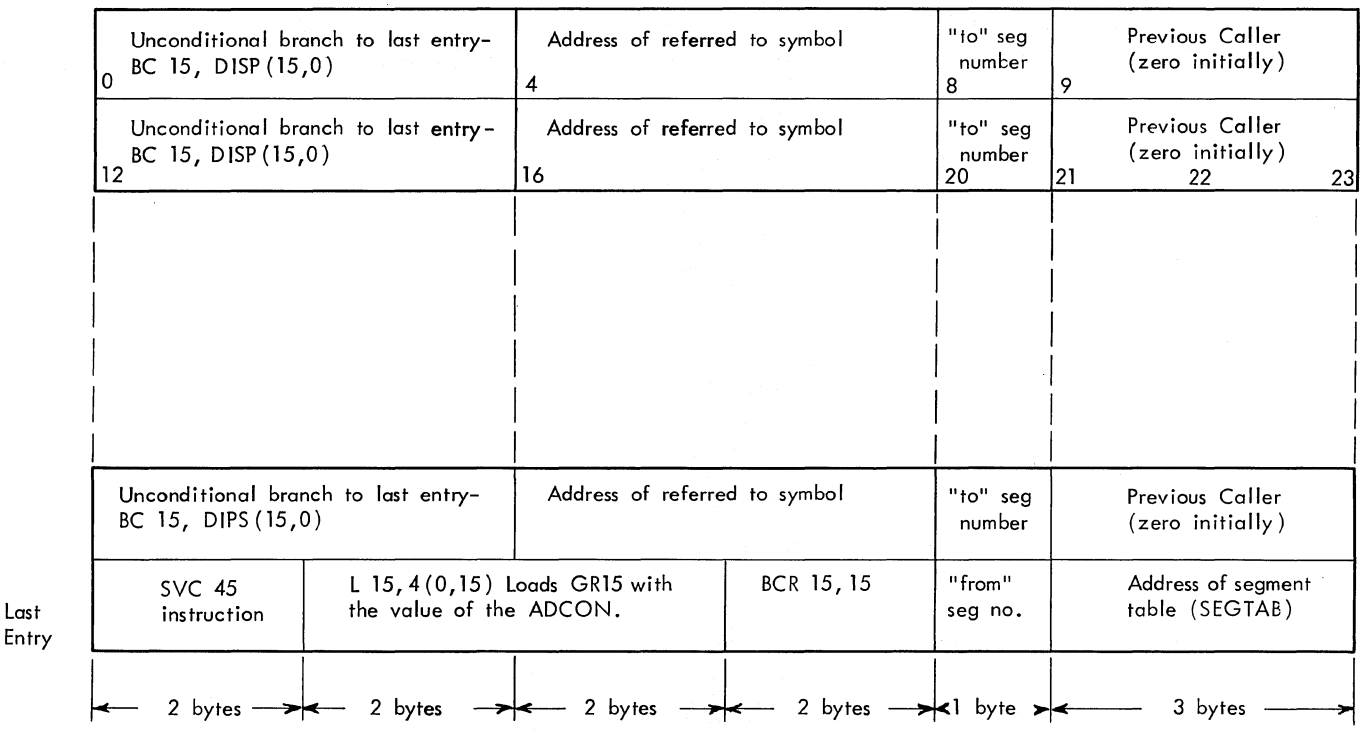
Offset	Bytes and Bit Pattern	Field Name	Field Description
328 (148)	Variable	Buffer address table	Addresses of the buffers for the various display control fields.
Variable	Channel Program Area		Work area for the channel program.
Variable	Screen Image Buffer		Contains a copy of the display screen.
Variable	DOM Identification Number		Contains information pertaining to delete-operator-message (DOM) messages.
Variable	Screen Control Table		Contains one two-byte entry for each line in the screen image buffer. Each entry contains flags indicating the content of the line.
	First Byte		
	1...	DCMMSGWR	WTOR Message displayed inline.
	.1..	DCMMSGIN	Message displayed inline.
	..1.	DCMMSGCN	Message continued on next line.
	...1	DCMMSGJK	To write out-of-line display.
	.... 1...	DCMMSGAD	Message can be deleted automatically.
	.... .1..	DCMMSGRD	Request has specified message to be deleted.
	.... ..1.	DCMMSGIF	Informational message inline.
	1	DCMMSGST	End of table indicator.
	Second Byte		
	1...	DCMMSGAC	Action message.
	.1..	DCMMSGC7	Descriptor code 7 message.
	..1.	DCMMSGDM	Message has been DOMed.
	...1	DCMMSGAR	Message is an accepted reply.
	.... 1...	DCMMSGIR	Intervention required message.
	.... .1..	DCMMSGCT	Continuation line.
	.... ..1.	DCMMSGPP	Issued by problem program.
	1	DCMMSGCL	Control line of inline MLWTO.
Variable	Secondary Screen Control Table		Contains a one-byte entry for each line in the message area of the screen. Each entry contains flags pertaining to out-of-line (display area) messages.
	1...	DCMSECCL	Control line of out-of-line display.
	.1..	DCMSECLL	Label line of out-of-line display.
	..1.	DCMSECDL	Data line of out-of-line display.
	...1	DCMSECBL	This line is blanked.
	.... ..1.	DCMSECDD	Line reserved for dynamic display.
	1	DCMSECST	End of table indicator.

The following table indicates the sizes in bytes of the fields of the "device dependent" portion of the TDCM.

Field	2260[1]	2260[2]	2250	M85
Buffer Address Table	0	0	10	8
CCW Area	80	80	112	72
Screen Image Buffer	960	960	3866	2800
DOM ID Numbers	44	44	188	120
Screen Control Table	22	22	94	60
Secondary Screen Control Table	11	11	47	30

[1]input/output
[2]output-only

ENTRY TABLE (ENTAB)



NOTE: DISP = is the displacement, in bytes, of this entry from the last entry.
"to" segment number -- is the number of the segment containing the symbol being referred to.
"from" segment number -- is the number of the segment that contains this entry table.

EVENT CONTROL BLOCK

Offset	Bytes and Bit Pattern	Field Description
0	1	Awaiting completion of an event:
	1...	W - Waiting for completion of an event.
		After completion of an event:
	.1..	C - The event has completed.
	..xx xxxx	Completion code.
		One of the following completion codes will appear at the completion of a channel program:
		Access Methods Other Than BTAM
	0111 1111	Channel program has terminated without error. (CSW contents useful.)
	0100 0001	Channel program has terminated with permanent error. (CSW contents useful.)
	0100 0010	Channel program has terminated because a direct access extent address has been violated. (CSW contents do not apply.)
	0100 0100	Channel program has been intercepted because of permanent error associated with device end for previous request. You may reissue the intercepted request. (CSW contents do not apply.)
	0100 1000	Request element for channel program has been made available after it has been purged. (CSW contents do not apply.)
	0100 1111	Error recovery routines have been entered because of direct access error but are unable to read home address or record 0. (CSW contents do not apply.)
		BTAM
	0111 1111	Completed normally
	0100 0001	Completed with an I/O error
	0100 1000	Enable command halted, or, I/O operation purged.
1	3	Awaiting completion of an event: Request block address.
		After completion of the event: 0 or remainder of completion code.

FREE QUEUE ELEMENT (FQE)

Pointed to by the boundary box for a job step task TCB, or by the FQE describing the next higher free area.

Offset	Bytes and Bit Pattern	Field Name	Field Description
0 (0)	4	FQEPTR	Address of FQE describing the next lower free area in main storage.
4 (4)	4	FQELNGTH	Number of bytes in described free area.

GOTTEN SUBTASK AREA QUEUE ELEMENT (GQE)

Pointed to by the boundary box for a subtask TCB, or by a subsequent GQE.

Offset	Bytes and Bit Pattern	Field Name	Field Description
0 (0)	4	GQEPTR	Address of previously created GQE.
4 (4)	4	GQELNGTH	Number of bytes in described area.

INTERRUPTION QUEUE ELEMENT (IQE)

Offset	Bytes and Bit Pattern	Field Name	Field Description
0 (0)	1		Reserved.
1 (1)	3	IQELNK	Address of the next IQE on the IQE queue.
4 (4)	4	IQEPARAM	The parameter that is to be passed to the asynchronous exit routine.
8 (8)	1		Reserved.
9 (9)	3	IQEIRB	Address of the IRB that is to be scheduled for this request.
12 (C)	1		Reserved.
13 (D)	3	IQETCB	Address of the TCB for this request.

MESSAGE INFORMATION LIST FOR TYPE-1 SVCS

Offset	Bytes and Bit Pattern	Field Name	Field Description
-4 (-4)	4		Address of end of Message Information List.
0 (0)	4	INFTCB	Address of the TCB for which the information is pertinent.
4 (4)	4	INFBADDR	Return address in the calling routine (from register 14) if entry to the type-1 SVC routine was by a branch instruction.
8 (8)	1	INFRCL	
	xxx.		Reason code (0=no reason code).
	...x xxxx		Number of bytes of variable data.
9 (9)	1	INFFLG	
	1...		INFBADDR contains a valid address.
10 (A)	2	INFCC	System completion code.
12 (C)	16	INFVAR	Information to be provided to the user. This field is always 16 bytes long, but the information may be less than 16 bytes.

Note: The above description is for one entry in the Message Information List. Each additional entry in the list will have the same format as offsets 0 through 27.

PROGRAM INTERRUPTION CONTROL AREA (PICA)

Pointed to by PIEPICA field.

Offset	Bytes and Bit Pattern	Field Name	Field Description
0 (0)	1		
0 (0)	 xxxx	PICAPRMK	Program mask to be used in the PSW when the programs of the task are executing.
1 (1)	3	PICAEXIT	Address of the user's program interruption exit routine to be given control when a program interruption of a specified type occurs.
4 (4)	2	PICAITMK	Bits 33 through 47 are used to indicate on which program interruption types the exit routine is to be used. The bits are numbered 0 through 15, left to right. A bit set to one indicates that the error handling routine is to be used.

PROGRAM INTERRUPTION ELEMENT (PIE)

Pointed to by TCBPIE field.

Offset	Bytes and Bit Pattern	Field Name	Field Description
0 (0)	1 1...		Flag byte. The task cannot accept further program interruptions. This bit is set whenever a user program interruption exit routine is entered.
1 (1)	3	PIEPICA	Address of the current PICA.
4 (4)	8	PIEPSW	Program interruption old PSW (stored at the program interruption).
12 (C)	4	PIEGR14	Save area for register 14.
16 (10)	4	PIEGR15	Save area for register 15.
20 (14)	4	PIEGR0	Save area for resgiser 0.
24 (18)	4	PIEGR1	Save area for register 1.
28 (1C)	4	PIEGR2	Save area for register 2.

MACHINE CHECK RECORD FOR SER0 AND SER1

SER0 and SER1 produce two types of record entries corresponding to the two types of errors processed: machine-check and channel errors. Record size varies with the type of record and with the machine model. The formats of the machine-check record produced by SER0 and SER1 is:

Offset	Bytes and Bit Pattern	Field Name	Field Description
0 (0)	1	CLASRC	
	1...		Machine Check Record.
	xxxx		MCH.
	xxxx ..1.		Converted MCH.
	xxxx 1...		SER1.
	xxxx 1..1		SER0.
	xxxx 1.1.		Converted SER1.
	xxxx 1.11		Converted SER0.
1 (1)	1	SYSREL	
	...x xxxx		OS.
	..1x xxxx		DOS.
	bits 3-7		
	0-1F		Release level 0-31.
2 (2)	4	Switches	
	Byte 0:		
	1...		More records follow.
	0...		Last record.
	.1..		Time-of-Day clock.
	..1.		Extended control mode.
	.... 1...		Time macro used (HHMMSS).
	...x .xxx		Unassigned.
	Byte 1:		
	1...		Short form of record.
	.1..		Record incomplete.
	..1.		System terminated.
	...1		First record of two record recording.
	.... 1...		Channel record included.
	.... .1..		Portion of data overlayed.
	.... ..1.		External machine check.
	x		Unassigned.
	Bytes 2-3:		Reserved.
6 (6)	1	RCDCNT	Bits 0-3 contain the sequence number of a physical record. Bits 4-7 contain the total number of physical records in this logical record.
7 (7)	1		Reserved.
8 (8)	4	DATE	The system date the record was made.
12 (C)	4	TIME	The system time the record was made.
16 (10)	1		Reserved.
17 (11)	3	CPUSER	CPU serial number (for System/370 only).
20 (14)	2	CPUID	CPU identifier (40, 50, etc.).
22 (16)	2	MCELLNG	Maximum Machine Check Extended Log length (for System/370 only).
24 (18)	8	PROGID	The module name of the program being processed and/or requesting service at the time of the error.

Offset	Bytes and Bit Pattern	Field Name	Field Description
32 (20)	8	JOBID	The name assigned to the job being executed at the time of the failure.
40 (28)	8	PSW	The Machine Check Old PSW.
48 (30)	Variable	LOGOUT	Register contents and hardware logout information. Logout size and format is model dependent.
	Variable	DAMAGE	Recovery Management Support's assessment of damage to the system.

CHANNEL ERROR RECORD FOR SER0 AND SER1

Offset	Bytes and Bit Pattern	Field Name	Field Description
0 (0)	1		Class and source.
	0010		Channel-check record.
	 1000		SER1.
	 1001		SER0.
1 (1)	1		System and release.
	000.		System/360.
	...x xxxx		Release level (0 through 31).
2 (2)	2		Flags:
	Byte 0		
	1...		Operator action message.
	.xxx xxxx		Reserved.
	Byte 1		
	1...		Message required.
	.1..		Record incomplete.
	..1.		System terminated.
	...1		Channel unsupported or failed to log.
	.... 1...		Invalid CUA.
	xxx		Reserved.
4 (4)	4		Reserved.
8 (8)	4		System date when failure occurred.
12 (C)	4		System time when failure occurred.
16 (10)	1		Reserved.
17 (11)	3		CPU serial number (not used for S/360).
20 (14)	2		CPU model number.
22 (16)	2		Reserved.
24 (18)	8		Name assigned to the job being executed at the time of failure.
32 (20)	16		List of eight devices (channel and unit addresses) that were busy when failure occurred.
48 (30)	8		CCW that was being executed when failure occurred.
56 (38)	8		CSW that was stored following the detection of the I/O failure.

Offset		Bytes and Bit Pattern	Field Name	Field Description
64	(40)	4		Reserved.
68	(44)	4		Device type at time of failure (obtained from UCB).
72	(48)	1		Flags used to identify channel at time of failure:
		0000 0001		Integrated multiplexer.
		0000 0010		Integrated selector.
		0000 0011		Integrated block multiplexer.
		0000 0101		Stand-alone selector 2860.
		0000 0110		Stand-alone multiplexer 2870.
		0000 0111		Block multiplexer 2880.
		0000 1010		Integrated file adapter.
73	(49)	3		Channel and unit address of the I/O device at time of failure.
76	(4C)	4		Reserved.
80	(50)	variable		Channel logout associated with the I/O failure causing the channel failure. Size is model and channel dependent.

Offset	Bytes and Bit Pattern	Field Name	Field Description
-12 (-C)	4	XRBQMAJ	LPRB: MFT with subtasking only. The address of the major RB for the program that contains the embedded entry point established by the IDENTIFY macro instruction that created this LPRB.
-8 (-8)	4	XRBSUC	Load list pointer: If this RB is for the first program loaded, this field is 0. Otherwise, this field contains the address of the XRBSUC field in the RB for the program loaded just prior to the program represented by this RB.
-4 (-4)	4	XRBPRE	Load list pointer: If this RB is for the most recently loaded program, this field contains the address of the TCBLLS field in the TCB. Otherwise, this field contains the address of the XRBSUC field in the RB for the program loaded immediately after the program represented by this RB.
0 (0)	8	XRBNM	Contents of this field depend on the use of this request block. The use of this request block is shown by bits 0-3 of byte 1 of the XSTAB field at offset 10 (X'A'). IRB For timer, 1st byte contains flags; for all other uses, contains no meaningful information. FRB, LPRB, LRB, PRB Program name. SIRB 8 character name of the error routine currently occupying the 400 byte I/O supervisor transient area. SVRB Type 2 SVC: No meaningful information.
0 (0)	4		Type 3 or 4 SVC: TTRN address, on the SVC library, of the load module. N, the concatenation number, is 0.
4 (0)	4		Four digit number of the form ysss. y - Number of the current phase of the routine. (First or only phase: y = 0.) sss - SVC number in unpacked decimal (signed) form.
8 (8)	2	XRBSZ	The number of contiguous doublewords occupied by the RB, the program (if applicable), and associated supervisor work areas. Does not include program size if a program extent list is present.
10 (A)	2	XSTAB	Flag bytes.
	Byte 0: xxxx		These bits are used to distinguish the LRB, LPRB, PRB, IRB, SIRB, and SVRB. These bits have the following definition:
	0000		PRB: The program was not loaded via a LOAD macro instruction and does not have minor entries identified via an IDENTIFY macro instruction.
	0001		PRB: The program was not loaded via a LOAD macro instruction and does have minor entries identified via an IDENTIFY macro instruction.
	0010		LPRB: The program was loaded via a LOAD macro instruction and does not have minor entries identified via an IDENTIFY macro instruction.

Offset	Bytes and Bit Pattern	Field Name	Field Description
10 (A) (cont.)	0011		LPRB: The program was loaded via LOAD macro instruction and does have minor entries identified via an IDENTIFY macro instruction.
	0100		IRB
	0101		FRB
	1000		SIRB
	1100		SVRB: The program is a type 2 SVC routine or a type 3 or 4 SVC routine that has not yet been loaded.
	1101		SVRB: The program is a type 3 or type 4 SVC routine that has been loaded.
	1110		LPRB: This block describes a minor entry identified via an IDENTIFY macro instruction.
	1111		LRB
	.... 1...		The type 3 or 4 SVC routine is resident.
	.... .1..		A checkpoint may be taken in a user exit from this SVC routine.
	.... ..1.		LRB, LPRB, PRB: The program was hierarchy block loaded. A program extent list exists.
	.... 1.1.		IRB: The IRB represents an End-of-task exit routine.
	1		Refreshable module.
	Byte 1:		FRB only:
	0...		Module being loaded is reenterable.
	1...		Module being loaded is not reenterable.
	.0..		The Finch routine has not executed a GETMAIN macro instruction.
	.1..		The Finch routine has executed a GETMAIN macro instruction.
	..xx xxxx		(Reserved bits)
	Byte 2:		All RBs except FRBs:
	1...		XRBLNK field points to the TCB.
	.1..		Active program.
	..1.		Registers 2-14 to be restored from XRBREG.
	...1		Reenterable or reusable program.
	.... 00..		IRB has no interrupt queue elements.
	.... 01..		IRB has interrupt queue elements which are request elements.
	.... 10..		This is a dummy LPRB, in a partition, for a program in the reenterable load module area. The LPRB for the program is in the reenterable load module area.
	.... 11..		IRB has interrupt queue elements which are not request elements.
	.... ..1.		Request block storage is to be freed when program returns.
	1		Wait on less than the number of specified events.
	0		Wait on a single event or all of the specified events.
12 (C)	4	XRBWLE	FRB only: Address of the most recent wait list element.
12	1	XRBUSE	Use count (the number of loads via the LOAD macro instruction less the number of deletes via the DELETE macro instruction).
13 (D)	3	XRBEP	Entry point address.
			End of LRB (Unless the program was hierarchy block loaded - XSTAB byte 0 bit 6 on.)
			Program Extent List (If the program was hierarchy block loaded, the following fields exist.)

Offset		Bytes and Bit Pattern	Field Name	Field Description
16	(10)	4	XLISTLH0	The length in bytes of that part of the program contained in hierarchy 0. This does not include the RB length.
20	(14)	4	XLISTLH1	The length in bytes of that part of the program contained in hierarchy 1.
24	(18)	4	XLISTAH0	The address of the program extent contained in hierarchy 0. It is not the address of the RB.
28	(1C)	4	XLISTAH1	The address of the program extent contained in hierarchy 1.
				End of the LRB Program Extent List
16	(10)	8	XRBPSW	User's old PSW.
16	(10)	4	XRREQ	FRB only: Address of the TCB for the task which requested that the module be loaded.
20	(14)	4	XRTLPRB	FRB only: Address of the LPRB build by Finch for the program which has been brought in by a LOAD macro instruction.
				End of the FRB
24	(18)	40	XBQ	IRB: Address of a 12 byte or 16 byte request element. LPRB: Address of an LPRB describing an entry identified by the IDENTIFY macro instruction. PRB: Address of an LPRB describing an entry identified by the IDENTIFY macro instruction. SIRB: Address of a 12-byte or 16-byte request element. SVRB: For type 3 and type 4 SVCs, this field contains the size of the program in bytes.
28	(1C)	1	XRBWT	Wait count.
29	(1D)	3	XRBLNK	Primary (active) queuing field. Address of the previous RB for the task. Address of the TCB if this is the first or only RB on the queue.
				End of LPRB and PRB
				(Unless the program was hierarchy block loaded - XSTAB byte 1 bit 6 on.)
				Program Extent List (If the program was hierarchy block loaded, the following fields exist.)
32	(20)	4	XLISTLH0	The length in bytes of the program extent contained in hierarchy 0. This does not include the RB length.
36	(24)	4	XLISTLH1	The length in bytes of the program extent contained in hierarchy 1.

Offset		Bytes and Bit Pattern	Field Name	Field Description
40	(28)	4	XLISTAH0	The address of the program extent contained in hierarchy 0. It is not the address of the RB.
44	(2C)	4	XLISTAH1	The address of the program extent contained in hierarchy 1.
				End of LPRB, PRB Program Extent List
32	(20)	64	XRBREG	IRB, SIRB, SVRB: Save area for 16 general registers (0-15)
96	(60)	nx8		SVRB: An extended save area, up to 6 doublewords, requested for SVC routine.

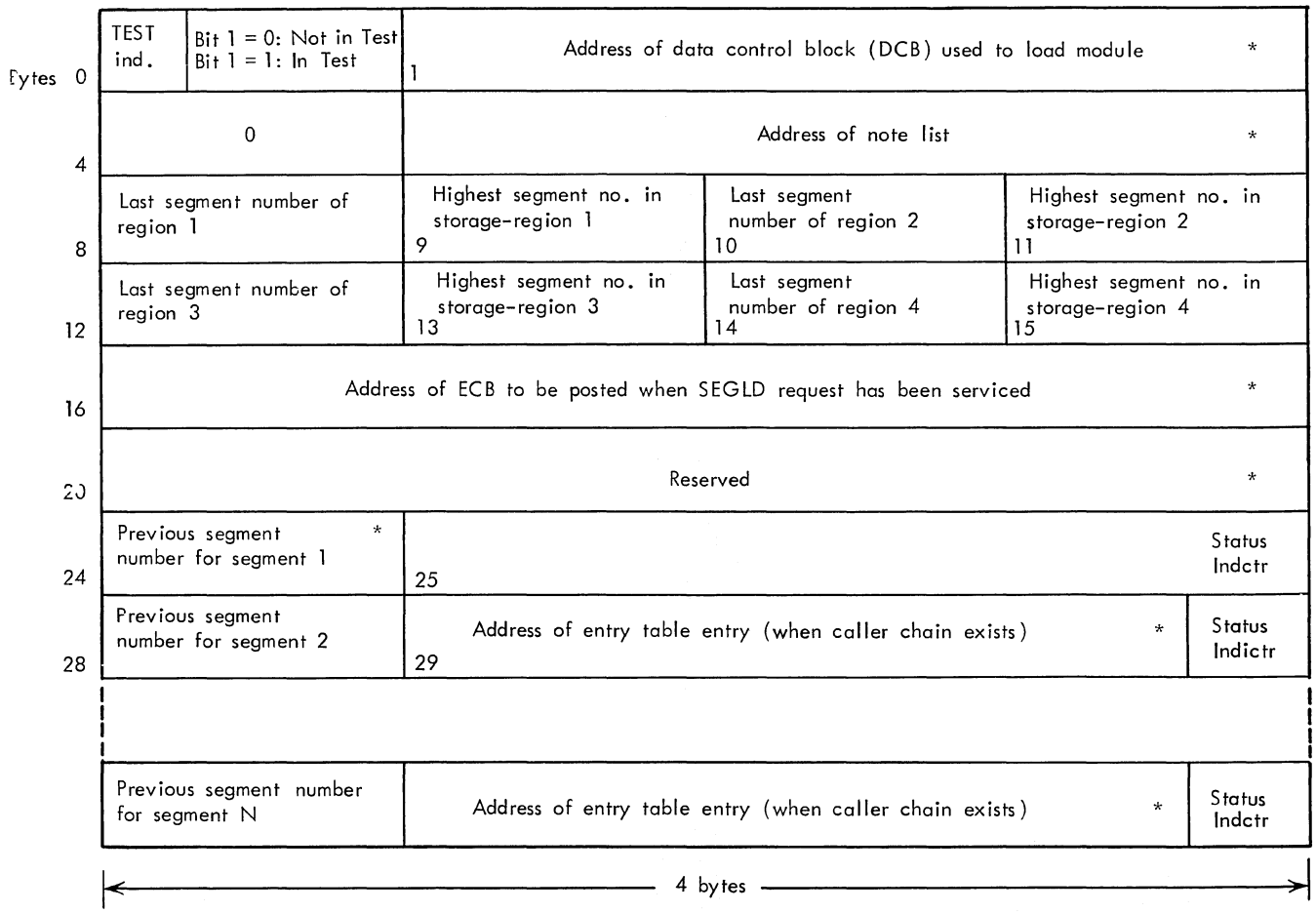
REPLY QUEUE ELEMENT

Offset	Bytes and Bit Pattern	Field Name	Field Description
0 (0)	4	RQERQE	Address of next reply queue element.
4 (4)	2	RQEID	Reply identification number.
6 (6)	1	RQEXA	
	0		Associated reply will be purged.
	...1		Cancel message exists.
	..1.		WTOR issued under protection key 0 (MCS only).
7 (7)	1	RQEAVAIL	Buffer flags (this field exists only with MCS).
	1000 0000		Buffer is free.
	0100 0000		Buffer is in use.
	0001 0000		Buffer was obtained dynamically.
	0000 1000		Buffer has been serviced.
8 (8)	4	RQETCB	Address of TCB of task that issued the message with which this reply is associated.
12 (C)	4	RQEXB	Non-MCS: address of purging message buffer.
12 (C)	4	RQEWQE	MCS: address of associated WQE.
16 (10)	1	RQELNGTH	Maximum length of reply.
17 (11)	3	RQEPTR	Address of user's buffer.
20 (14)	4	RQEECB	Address of user's ECB.

REQUEST QUEUE ELEMENT (RQE)

Offset	Bytes and Bit Pattern	Field Name	Field Description
0 (0)	2	RQELNK	Address of next RQE on the queue.
2 (2)	2	RQEUCB	
	xxxx xxx.		Address of UCB.
	1		This RQE represents a request for a system error routine that operates under an SIRB.
4 (4)	1		Free List indicator.
	1111 1111		Indicates that the RQE is on the Free List (X'00' indicates not on Free List).
5 (5)	3	RQEIOB	Address of the associated IOB.
8 (8)	1	RQEPRI	Dispatching priority of the requesting task.
9 (9)	3	RQEDEB	Address of the associated DEB.
12 (C)	1		Protection key of requesting task.
13 (D)	3	RQETCB	Address of the TCB with which the I/O request is associated.

SEGMENT TABLE (SEGTAB)



STAE CONTROL BLOCK (SCB)

Pointed to by TCBNSTAE field.

Offset	Bytes and Bit Pattern	Field Name	Field Description
0 (0)	1		STAE flag byte for previous SCB for this task; or 0 if this is the first SCB created for this task.
1 (1)	3		Address of previous SCB or, if none, address of TCB.
4 (4)	4		Address of the user-written STAE exit routine.
8 (8)	1		I/O flags.
	.... .1..		Allow asynchronous interruptions.
	00		Quiesce I/O.
	01		Halt I/O.
	10		Bypass I/O intervention.
9 (9)	3		Address of the parameter list to be passed to the STAE exit routine.
12 (C)	1		
	1...		SCB will not be canceled by the exit routine when XCTL is issued.
	.1..		ISAM/TAM Switch.
13 (D)	3		Address of the request block of the task that issued the STAE.

STAE PARAMETER LIST

Pointed to by STAE control block.

Offset	Bytes and Bit Pattern	Field Name	Field Description
0 (0)	1		Flags.
	1...		STAI parameter list (valid in MVT only).
	0...		STAE parameter list.
	.... .1..		Allow asynchronous interruptions.
	00		Quiesce I/O.
	01		Halt I/O.
	10		Bypass I/O processing.
1 (1)	3		Address of the STAE exit routine.
4 (4)	4		Address of the STAE exit routine parameter list.

STORAGE UTILIZATION BLOCK

Offset	Bytes and Bit Pattern	Field Name	Field Description
0 (0)	1	SUBDAIO	
	1...		I/O routine is in use.
1 (1)	1	SUBFLGS	
	1...		Exit to Processor 1 routine.
	.1..		Initialize SUB on first entry.
	.... 1...		Write PFK updates.
2 (2)	2	SUBNUM	Number of transient consoles.
4 (4)	4	SUBDOM	Address of SUBKEY field in SUB.
8 (8)	4	SUBPBASE	Address of the base DCB.
12 (C)	4	SUBTIOT	Address of the task I/O table (in the Transient DCM control block of the SUB).
16 (10)	4	SUBSPACE	Address of the SUB work area (SUBAREA).

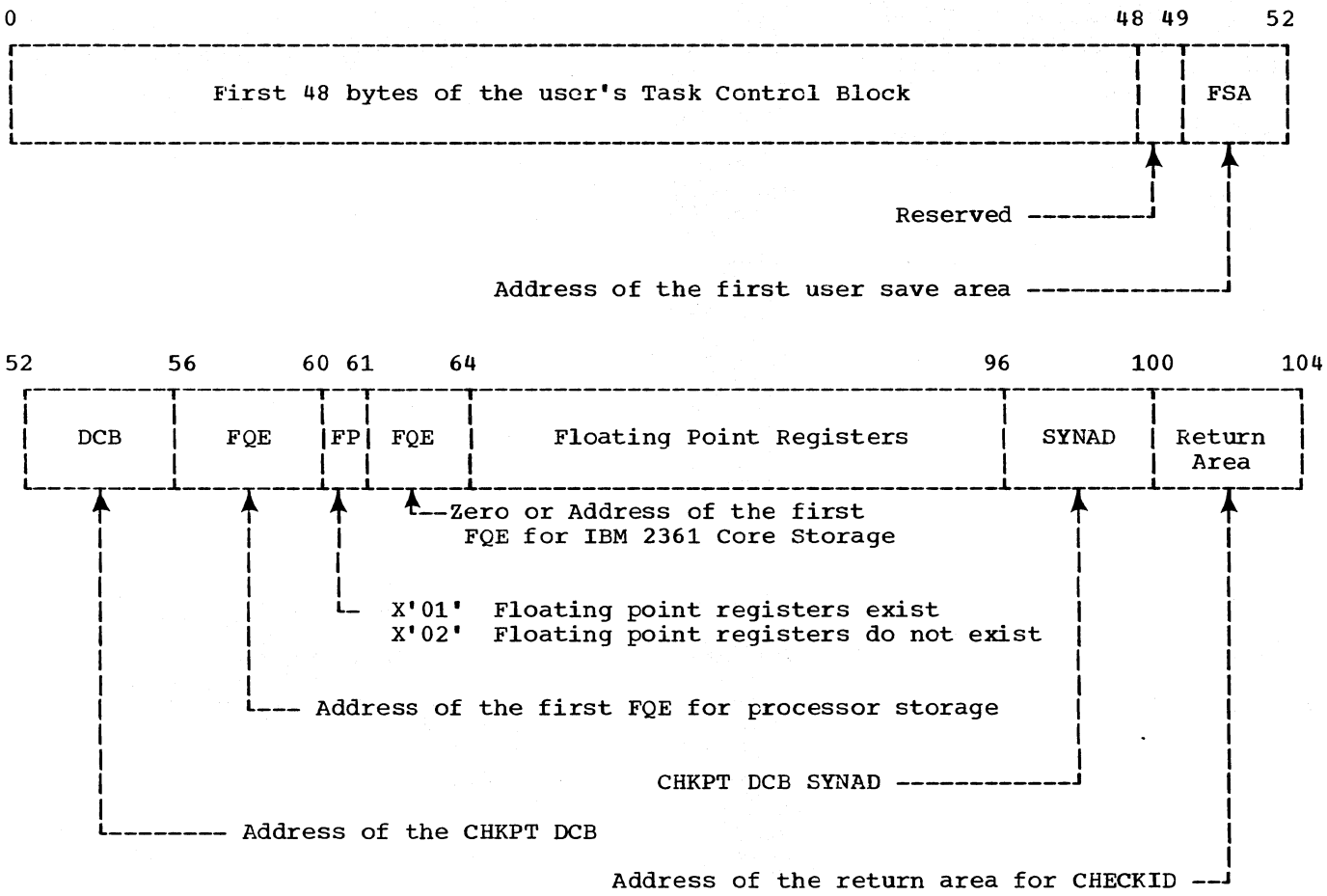
The following fields are generated only if transient DCMs have been defined for the system.

20 (14)	4	SUBQUE	Address of the DCM request queue (SUBREQ).
24 (18)	4	SUBBLDL	Address of the DCM BLDL list (SUBTTR).
28 (1C)	4	SUBBLK	Address of the DCB (in the transient DCM control block).
32 (20)	4	SUBPFKAD	Address of the PFK area.
36 (24)	2(n)	SUBKEY	Delete Operator Message (DOM) elements; each DOM element is two bytes long, there is one DOM element for each display console in the system.
	36	SUBAREA	Work area required by the SUB (CXSA save area, work area, two register save areas).
	4(n)	SUBREQ	DCM request elements; there is one 4-byte element for each console in the system that has a transient DCM.
	8(n)	SUBTTR	DCM BLDL elements; there is one 8-byte element for each transient DCM in the system.

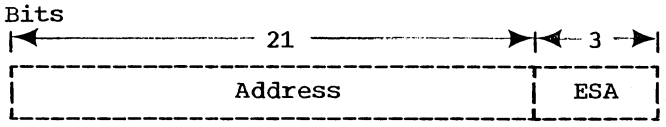
Each of the following areas is expanded only if the associated type of display console is included in the system:

64	SUB2260	RMS/SER channel programs for the 2260 display consoles.
44	SUB2250	RMS/SER channel programs for the 2250 and Model 91 display operator consoles.
24	SUB5450	RMS/SER channel programs for the Model 85 and Model 165 operator consoles.
364	SUBDCB	Transient DCM control block; contains information pertaining to transient DCMs. This area is included only if transient DCMs are in the system.
142	SUBDCMB	Base DCM; contains executable code which gains control whenever an STIMER interval elapses. This routine sets flags in the DCM and POSTs the DCM for the Timer/Interpreter routine.

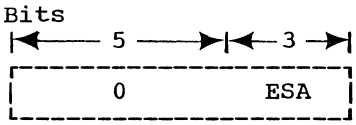
SUPERVISOR RECORD (SUR)



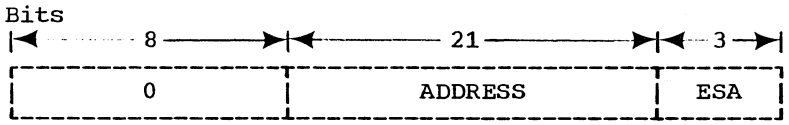
SVC TABLE



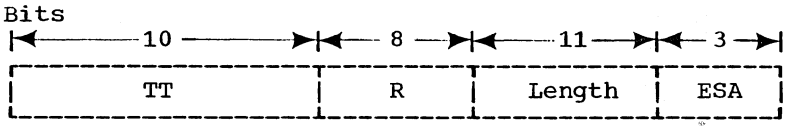
3-byte entries for type 1 and 2 SVC routines



1-byte entries for type 3 and 4 SVC routines



4-byte entries for type 1 and 2 SVC routines



4-byte entries for type 3 and 4 SVC routines

TASK CONTROL BLOCK

Pointed to by CVTHEAD or by TCBTCB field in next higher priority TCB.

Offset		Bytes and Bit Pattern	Field Name	Field Description
-32	(-20)	30	TCBFRS	Floating point register save area.
0	(0)	4	TCBRBP	Address of the RB for executing program.
4	(4)	4	TCBPIE	Address of the Program Interrupt Element (PIE).
8	(8)	4	TCBDEB	Address of the queue of DEBs.
12	(C)	4	TCBTIO	Address of the Task I/O Table (TIOT).
16	(10)	4	TCBCMP	Task completion code. A flag byte field containing indicators used or set by the ABEND SVC.
		Byte 0:		
		1...		A dump has been requested.
		.1..		Indicates step ABEND request.
		..1.		Some problem program storage was overlaid by the second load of ABEND. A first load overlay is indicated in TCBFLGS field.
		...x		Reserved.
		.... 1...		A double ABEND has occurred.
		.... .1..		A dump message (WTO) is to be issued to the operator.
		.... ..1.		Scheduler is to print an indicative dump.
		1		An ABEND message is provided to be printed by ABDUMP.
17	(11)	Bytes 1-3:		System completion code in first 12 bits; user completion code in last 12 bits. These codes are explained in the manual Messages and Codes.
20	(14)	1	TCBFLTRN	A byte used for flags as described:
		...x xxxx		Reserved.
		1...		Both TESTRAN and decimal simulator programs being used on a Model 91.
		.1..		Suppress taking checkpoints for this step.
		..1.		Job step TCB: This is a graphics foreground job or the graphic job processor.
21	(15)	3	TCBTRN	TESTRAN: Address of control core table.
24	(18)	1		Reserved.
25	(19)	3	TCBMSS	Job Step TCB: Address of the boundary box. Subtask TCB: Address of last created GQE.
28	(1C)	1	TCBPKF	Storage protection key for this task.
		xxxx		Storage protection key.
		 0000		Must be zeros.
29	(1D)	5	TCBFLGS	Flag byte fields.
		Byte 0:		
		1...		Abnormal termination in progress.
		.1..		Normal termination in progress.
		..1.		ABEND was initiated by the resident abnormal termination routine.
		...1		GTF Trace has been suspended.
		.... ..1.		Problem-program storage has been overlaid to process ABEND.
		1		Prohibit queuing of asynchronous exits for this task.
		.... xx..		Reserved.
30	(1E)	Byte 1:		
		1...		System task: ABEND prohibited for this task.

Offset		Bytes and Bit Pattern	Field Name	Field Description
30 (cont.)	(1E)	..1.		Task has stopped Trace Table.
		...1		Task has issued a system-must-complete and set all other tasks in the system non-dispatchable.
		.... 1...		Task has issued a step-must-complete and turned off all other tasks in the step.
		1		This task is a member of a time-sliced group.
		.x.. .xx.		Reserved.
31	(1F)	Byte 2:		
		xx.x ...x		Reserved bits.
		..1.		Exit Effector: System error routines already operating for this task.
		.... 1...		Floating point registers exist.
		.... .1..		Job scheduler routines in process.
		.... ..1.		XCTL routine is changing the storage protection key in the PSW from zero to that used by the problem program.
32	(20)	Byte 3:		Reserved.
33	(21)	Byte 4:		(If any bit in this byte is 1, the task is nondispatchable.)
		1		Primary nondispatchability bit. This bit is set to 1 if any of the secondary nondispatchability bits (offset 173 through 175) is set to 1. This bit is set to 0 if a secondary nondispatchability bit is set to 0 and and all other secondary nondispatchability bits are 0.
		.... 1...		Another task is in system-must-complete status.
		.... .1..		Another task in this job step is in must-complete status.
		xxxx ..x.		Reserved bits.
34	(22)	1	TCBLMP	Number of resources for which this task is enqueued.
35	(23)	1	TCBDSP	Dispatching priority for this task.
36	(24)	4	TCBLLS	Address of the most recently added RB on the list of programs loaded via the LOAD macro instruction.
40	(28)	4	TCBJLB	Address of a JOBLIB DCB.
44	(2C)	4	TCBJSTCB	Subtask TCB: Address of Job Step TCB.
44	(2C)	4		Other TCBs: Zero.
48	(30)	64	TCBGRS	General register save area.
112	(70)	1	TCBIDF	TCB identifier field.
113	(71)	3	TCBFSA	Address of the first problem program save area.
116	(74)	4	TCBTCB	Address of next TCB of lower priority.
120	(78)	4	TCBTME	Address of the timer element.
124	(7C)	1	TCBPIB	Partition type:
		00..		System task partition.
		01..		Reader partition.
		10..		Writer partition.
		11..		Processing program partition.
		..1.		Large partition.
		..0.		Small partition.
		...1		CPU timing stopped by FINCH until transient area is loaded.
		.... ..1.		Writer partition, used by ABEND. Required by transient writer, but used also by resident writer.

Offset		Bytes and Bit Pattern	Field Name	Field Description
124 (7C) (cont.)		1		Scheduler in control. Bit turned off when TIOT written on SYSJOBQE. Used by ABEND.
		.... xx..		Reserved bits.
125	(7D)	3		Address of the partition information block (PIB).
128	(80)	4	TCBNTC	Address of the previous TCB on the originating task's subtask queue. The TCB for the last subtask has 0's in this field.
132	(84)	4	TCBOTC	Address of the TCB for the originating task.
136	(88)	4	TCBLTC	Address of the last TCB on the subtask queue for this task. The TCB for the last subtask has 0 in this field.
140	(8C)	4	TCBIQE	Address of an interruption queue element (IQE) for scheduling the ETXR routine for the originating task.
144	(90)	4	TCBECB	Address of the ECB that will be posted by the task termination routines when normal or abnormal termination occurs.
148	(94)	4	TCBXTCB	Reserved.
152	(98)	1	TCBFTLMP	Limit priority.
153	(99)	3	TCBFTFLG	Without Subtasking: Reserved. With Subtasking: Flag Bytes.
		Byte 0:		
		.... .1..		Top task in tree of abnormally terminating tasks.
		.... ..1.		Abnormal termination dump has been completed.
		1		Task in enqueued on dump data set.
		xxxx x...		Reserved.
154	(9A)	Byte 1:		
		1...		OPEN in process for the dump data set.
		.... .1..		Dump data set is open for job step.
		.... ..1.		SYSABEND data set.
		.... ..0.		SYSUDUMP data set.
		.xxx x..x		Reserved.
155	(9B)	Byte 2:		
		...1		A valid message recursion has occurred in ABEND.
		.... .1..		No abnormal termination dumps can be provided within this job step.
		xxx. x.xx		Reserved.
156	(9C)	4		Reserved.
160	(A0)	1	TCBNSTAE	Flags internal to STAE routine.
161	(A1)	3		Address of the current STAE control block.
164	(A4)	4	TCBTCT	Address of the timing control table if the system management facilities option is present.
168	(A8)	4	TCBUSER	A field available to the user.
172	(AC)	1	TCBDAR	Damage Assessment Routine (DAR) flags.
		1...		Primary DAR recursion - DAR failure while writing core image dump.
		.1..		Secondary DAR recursion - DAR failure while attempting to reinstate failing Partition.
		..1.		A dump only has been requested.

Offset	Bytes and Bit Pattern	Field Name	Field Description
172 (AC) (cont.)	...1		A recursion is permitted in CLOSE after DAR processing is completed.
	.... 1...		Problem-program storage has been overlaid to execute DAR.
	.... .1..		WTO in progress for 'Reinstatement Failure' message.
	1		SVC DUMP is executing for this task.
	.... ..x.		Reserved bits.
173 (AD)	3	TCBNDSP	Secondary nondispatchability bits.
		TCBNDSP1	If any bit in these bytes in 1, the primary nondispatchability bit (byte 33, bit 7) is 1, and the task is nondispatchable.
	xx..		Damage assessment routine bits.
	1...		The task is temporarily nondispatchable.
	.1..		The task is permanently nondispatchable.
	..xx		Recovery management support and system error recovery bits.
	..1.		The task is temporarily nondispatchable.
	...1		The task is permanently nondispatchable.
	 xxxx		Reserved bits.
174 (AE)		TCBNDSP2	
	1...		ABDUMP is processing (MFT with subtasking).
	1		The dump data set is being opened (MFT with subtasking).
	.xxx xxx.		Reserved.
175 (AF)		TCBNDSP3	
	1...		Task has been terminated (MFT with subtasking).
	.1..		Task to be terminated by ABEND (MFT with subtasking).
	..xx xxxx		Reserved.
176 (B0)	4		Reserved.
180 (B4)	1	TCBRECDE	ABEND recursion reconfiguration flags.
	1...	TCBREC	Valid ABEND recursion.
	.000 0001	TCBOPEN	OPEN of the SYSABEND or SYSUDUMP data set for the job step.
	.000 0010	TCBCLOSD	CLOSE of the direct SYSOUT on tape.
	.000 0011	TCBCLOSE	CLOSE of open data set.
	.000 0100	TCBCLOSF	Forced close of DCBs (graphics).
	.000 0101	TCBGREC	Graphics (GFX) interface in control.
	.000 0110		Reserved.
	.000 0111	TCBADUMP	ABDUMP in process for this task.
	.000 1000		Reserved.
	.000 1001	TCBMESG	Message recursion.
	.000 1010		Reserved.
	.000 1011		Reserved.
	.000 1100		Reserved.
	.000 1101		Reserved.
	.000 1110	TCBTCAMR	TCAM message control program reinitialization.
	.000 1111	TCBSAVCD	Reserved.
	.001 0000	TCBTYP1W	Invalid ABEND recursion from type-1 SVC WTP message.
	.001 0001		
	.		
	.		Reserved.
	.		
	.011 0011		
	.011 0100	TCBTYP1R	Valid return from type-1 SVC WTP message.
	.011 0101		
	.		
	.		Reserved.
	.		
	.111 1111		
181 (B5)	3	TCBJSCB	Address of the job step control block.

TIMER QUEUE ELEMENT (TQE)

Offset		Bytes and Bit Pattern	Field Name	Field Description
0	(0)	1	TQEFLGS	Flags field.
		1000		Element is off the queue.
		0100		LTOD requested.
		0001		Exit specified.
		 1000		REAL request.
		 0100		SUPVSR request.
		 0110		WAIT request.
		0100 0101		Midnight element.
		 0000		TASK request.
		..x.		Reserved.
1	(1)	3	TQETCB	Address of TCB
4	(4)	1		Reserved.
5	(5)	3	TQEFLNK	Address of next timer queue element.
8	(8)	1		Reserved.
9	(9)	3	TQEBLNK	Address of preceding timer queue element.
12	(C)	4	TQEVAL	Time of expiration or time remaining.
16	(10)	4	TQELHPSW	First word of current PSW (used when TQE serves as an IRB).
20	(14)	4		Reserved.
24	(18)	4	TQESAADR	Address of processing program save area.
28	(1C)	1		Reserved.
29	(1D)	3	TQEEXIT	Address of timer asynchronous exit routine.
32	(20)	4	TQEWORK	Expected TOD Clock value at expiration.
36	(24)	1		Flags field.
		1000 0000		Synchronized TQE.
36	(24)	64	TQEGRS	Register save area when TQE is used as an IRB.
96	(60)	4	TQEECB	ECB when WAIT operand is specified in STIMER macro instruction.
100	(64)	16	TQEIQE	IQE when TQE serves as IRB.

TIME-SLICE CONTROL ELEMENT (TSCE)

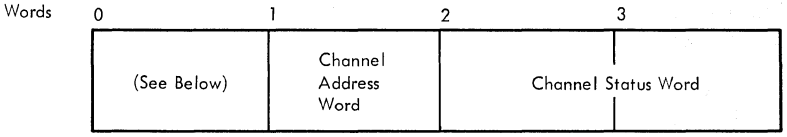
Pointed to by CVTTSCE field.

Offset		Bytes and Bit Pattern	Field Name	Field Description
0	(0)	1		Dispatching priority of time-slice group.
1	(1)	3		Address of highest-priority TCB that is a member of the time-slice group.
4	(4)	1		Reserved.
5	(5)	3		Address of the lowest-priority TCB that is a member of the time-slice group.
8	(8)	1		Reserved.
9	(9)	3		Address of the next TCB to be dispatched when the time-slice group has the highest priority.
12	(C)	1		Reserved.
13	(D)	3		Length of time slice: in milliseconds before NIP; in timer units after NIP (26.04166 microseconds per timer unit).

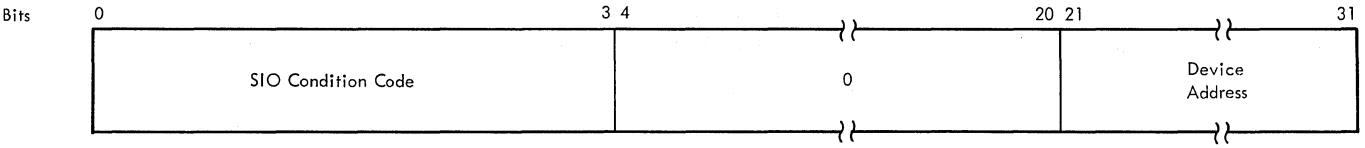
TRACE TABLE RECORD FORMAT

NOTE: Each entry is four words

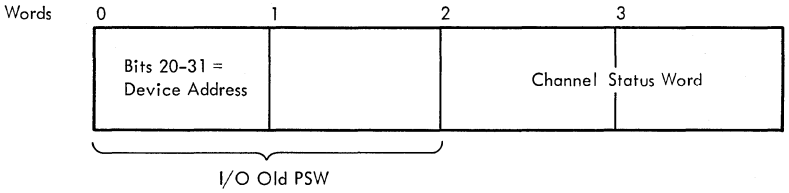
SIO Instruction:



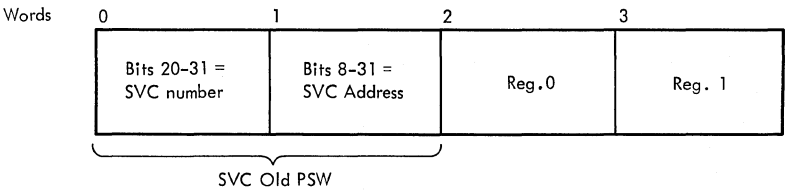
First Word of SIO Entry:



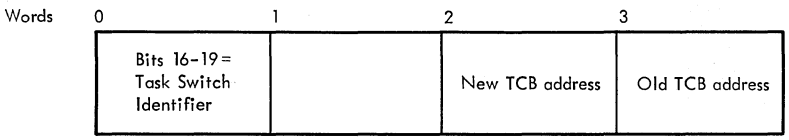
I/O Interruption:



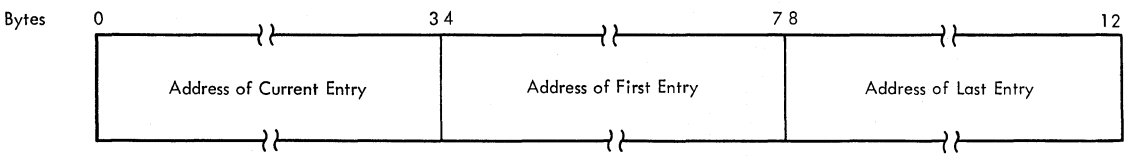
SVC Interruption:



Task Switch



The addresses of the trace table are contained in a 12-byte field whose address is at hex loc 14. The format of the field is:



UNIT CONTROL MODULE (IEECUCM)

The Unit Control Module consists of several distinct sections. These sections appear in main storage in the following order:

1. UCM Prefix
2. MCS Prefix
3. UCM Base
4. Event Indication List (EIL)
5. One or more UCM Entries
6. Message text

While the format remains approximately the same, the contents of the UCM depends on options and devices specified during system generation. The offsets given reflect a UCM that includes all of the options.

UCM PREFIX

Pointed to by UCMXECB-8 (in UCM Base).

Offset		Bytes and Bit Pattern	Field Name	Field Description
0	(0)	2	UCM2WID	The first 4 bytes of the UCM prefix are used in MVT to contain the TJID. Because TSO is not available with MFT, these bytes are functionally not used.
2	(2)	2	UCM2RID	See above.
4	(4)	4	UCM2PST	Address of branch entry to POST (IEA0PT01).

MCS PREFIX TO UCM BASE (MCS ONLY)

Pointed to by UCMEXB-4 (in UCM Base).

Offset		Bytes and Bit Pattern	Field Name	Field Description
0	(0)	4	UCMMCENT	Address of master console UCM entry.
4	(4)	72	UCMSAVE0	Resident and communications save area.
76	(4C)	4	UCMDOME	Address of first DOM element.
80	(50)	4	UCMWTOX	Address of WTO/R exit routine.
84	(54)	2	UCMFLGS	Control flags:
		Byte 0		
		x...	UCMSYSA	Reserved.
		.1..	UCMSYSB	Hardcopy support required.
		..1.	UCMSYSC	Commands to hard copy.
		...1	UCMSYSD	Console switch for master console.
		.... 1...	UCMSYSE	No consoles exist.
		.... .1..	UCMSYSF	Graphics consoles exist.
		.... ..1.	UCMSYSG	Hardcopy device SYSLOG.
		1	UCMSYSH	Timer present and operative.

Offset		Bytes and Bit Pattern	Field Name	Field Description
85	(55)	Byte 1		
		1...	UCMSYSI	WQE housekeeping needed.
		.1..	UCMSYSJ	Hard copy to be done.
		..1.	UCMSYSK	New console composite.
		...1	UCMSYSL	OPEN being issued to ring console alarm.
		.... 1...	UCMSYSM	Failing console is composite.
		.... .1..	UCMSYSN	Graphic console present.
		.... ..1.	UCMSYSO	Used by WTL.
86	(56)	2	UCMOWTOR	Default values for old WTOR macros.
88	(58)	4	UCMCMID	Current message ID.
92	(5C)	4	UCMHCUCM	Address of hardcopy UCM entry, or zero.
96	(60)	1	UCMXCT	External request count.
97	(61)	3	UCMUEXIT	Address of user exit data, or zero.
100	(64)	2	UCMHRDRT	Hardcopy routing code assignments.
102	(66)	2		Reserved.
104	(68)	4	UCMXSA	Address of UCM3WD (offset X'70').
108	(6C)	4		Reserved.
112	(70)	16	UCM3WD	Parameter list area for SVC 72.
128	(80)	4	UCMQRTN	Address of IEECMENQ.
132	(84)	4	UCMRUTCK	Route checking field
136	(88)	4	UCMDOMRT	Address of IEECMDOM.
140	(8C)	4	UCMTPPTR	Address of 2740 save area (UCMTPSAV).
144	(90)	4	UCMNPECB	NIP ECP; posted when NIP's hard copy can be written.
148	(94)	4	UCMLOGAD	Address of System Log TCB.
152	(98)	4	UCMDTINT	Dynamic display time interval.
156	(9C)	1	UCMSDS1	Status display flags.
		1...	UCMSDS1A	STCMDS to hard copy.
		.1..	UCMSDS1B	INCMDS to hard copy.
157	(9D)	3	UCMSDS2	Reserved.
160	(A0)	4		Address of UCM extension.
164	(A4)	4		Address of MCS prefix.

UNIT CONTROL MODULE (UCM) BASE

Pointed to by CVTCUCB field.

Offset		Bytes and Bit Pattern	Field Name	Field Description
-8	(-8)	4		Address of UCM Prefix.
-4	(-4)	4		Address of MCS Prefix.
0	(0)	4	UCMXECB	External interruption ECB.
4	(4)	4	UCMAECB	Attention interruption ECB.
8	(8)	4	UCMOECB	WTO/R request ECB.
12	(C)	4	UCMDECB	DOM request ECB.
16	(10)	4	UCMRECB	RMS ECB.
20	(14)	4	UCMLSTP	Address of event indication list.
24	(18)	4	UCMWTOQ	Address of first WQE (system output queue).
28	(1C)	4	UCMRPYQ	Address of the first RQE.
32	(20)	4	UCMRPYI	Reply ID assignment pattern (100 bit positions used).
45	(2D)	1	UCMRQLM	ID limit.
46	(2E)	2	UCMWQLM	WQE buffer limit.
48	(30)	4	UCMRQECB	Reply-request-waiting ECB.
52	(34)	4	UCMWQECB	Buffer-request-waiting ECB.
56	(38)	2	UCMRQNR	Current RQE count.
58	(3A)	2	UCMWQNR	Current WQE count.
60	(3C)	4	UCMWQEND	Address of last WQE or zero.
64	(40)	4	UCMPXA	Address of communications task TCB (IEECVTCB).
68	(44)	1	UCMMODE	Mode flags.
		.... 1...	UCMAMFA	Accept VARY command with MSTCONS operand from any MCS secondary console.
		.... .1..	UCMOGCE	Only graphics consoles exist.
		.... ..1.	UCMMCS	MCS generated with system.
		1	UCMFIX	MFT mode.
		0		MVT mode.
69	(45)	1	UCMCORE	WTO purge routine switches.
70	(46)	1	UCMMODEL	System model number.
71	(47)	1	UCMIMCR	Used by console initialization routine for error handling.
72	(48)	4	UCMVEA	Address of first UCM entry.
76	(4C)	4	UCMVEZ	Size of UCM entry.
80	(50)	4	UCMVEL	Address of last UCM entry.
84	(54)	56	UCMSAVE3	Save area for ED2.
140	(8C)	64	UCMSAVE4	Save area for CRA.

Offset		Bytes and Bit Pattern	Field Name	Field Description
204	(CC)	4	UCMR9SV	Save area for ED2.
208	(D0)	4	UCMWEA	Address of UCMWQE (head of WQE queue).
212	(D4)	4	UCMWEZ	Address of UCMWSIZ (length of queue).
216	(D8)	4	UCMWEL	Address of UCMWEND (end of WQE queue).
220	(DC)	4	UCMREA	Address of UCMRQE (head of RQE queue).
224	(E0)	4	UCMREZ	Address of UCMRSIZ (length of queue).
228	(E4)	4	UCMREL	Address of UCMREND (end of RQE queue).
232	(E8)	232	UCMOPENX	Work area for OPEN/CLOSE.

The following fields are included only in systems without MCS.

464	(1D0)	144	UCMREAD	Input buffer.
608	(260)	72	UCMSAVE1	Save area.
680	(2A8)	72	UCMSAVE2	Save area.
752	(2F0)	136	UCMINDCB	Space for input DCB.
888	(378)	116	UCMOUDCB	Space for output DCB.

UCM EVENT INDICATION LIST (EIL)

Pointed to by UCMLSTP (in UCM Base).

Offset		Bytes and Bit Pattern	Field Name	Field Description
0	(0)	1	UCMEIL	Length in doublewords of EIL.
1	(1)	1	UCMRPYL	Last assigned reply ID.
2	(2)	6		Reserved.
8	(8)	4		Address of external ECB (UCMXECB).
12	(C)	4		Address of attention ECB (UCMAECB).
16	(10)	4		Address of WTO/R ECB (UCMOECB).
20	(14)	4		Address of DOM ECB (UCMDECB).
24	(18)	4		Address of RMS ECB (UCMRECB).
28	(1C)	Variable (8-128)		I/O ECB addresses. This list consists of one word for each console device specified during system generation. Last entry has high-order byte set to X'80'.

UCM ENTRY INDIVIDUAL DEVICE MAP

Pointed to by UCMVEA if first entry; by UCMVEL if last.

Offset		Bytes and Bit Pattern	Field Name	Field Description
0	(0)	4	UCMECB	Address of I/O completion ECB.
4	(4)	4	UCMSBR	Address of resident processor module.
8	(8)	4	UCMDCB	Address of DCB.
12	(C)	4	UCMUCB	Address of UCB.
16	(10)	8	UCMNAME	Name of processing module.
24	(18)	1	UCMSTS	Status flags.
		1...	UCMAF	Attention pending.
		.1..	UCMPF	Output pending.
		..1.	UCMBF	Device busy.
		...1	UCMCF	Close pending.
		.... 1...	UCMTA	Open pending.
		.... .1..	UCMTB	Dequeue appropriate output queue entries.
		1	UCMTC	Working on MLWTO.
25	(19)	1	UCMATR	Attribute flags.
		1...	UCMOF	WTO support.
		.1..	UCMIF	Attention support.
		..1.	UCMXF	External interruption support.
		...1	UCMUF	Device active.
		...0		Device inactive.
		.... 1...	UCMLF	Load flag.
		0000 0100	UCMAT04	Device status to change.

The following fields are included with MCS only:

26	(1A)	1	UCMID	Unique entry ID.
27	(1B)	1	UCMXA	Reserved.
28	(1C)	4	UCMXB	Reserved.
32	(20)	4	UCMRTCD	Routing codes assigned to this console.
36	(24)	4	UCMOUTQ	Address of the output queue.
40	(28)	2	UCMAUTH	Command code authorization.
		Byte 0		
		1...	UCMAUTH1	Command group 1 (SYS).
		.1..	UCMAUTH2	Command group 2 (I/O).
		..1.	UCMAUTH3	Command group 3 (CONS).
		Byte 1		Reserved.
42	(2A)	2	UCMDISP	Disposition flags.
		Byte 0		
		1...	UCMDISPA	Master console.
		.1..	UCMDISPB	Hardcopy device/console.
		..1.	UCMDISPC	Graphics.
		...1	UCMDISPD	Output only.
		.... 1...	UCMDISPE	Console has full I/O capability.
		.... .1..	UCMDISPF	Console is message stream only.
		.... ..1.	UCMDISPG	Console is status display only.
		1	UCMDISPH	MCS.

Offset		Bytes and Bit Pattern	Field Name	Field Description
44	(2C)	4	UCMALTEN	Address of alternate input UCM entry.
48	(30)	4	UCMOAOEN	Address of alternate output UCM entry.
52	(34)	4	UCMWLAST	Address of last WQE serviced on output queue.
56	(38)	4	UCMCOMPC	Address of other device if this is a composite console.
60	(3C)	2	UCMMSG	Message type flags.
		Byte 0		
		1...	UCMMSGA	Display job names requested.
		.1..	UCMMSGB	Display status requested.
		..1.	UCMMSGC	Monitor active.
		.... 1...	UCMMSGE	Show requested.
		.... .1..	UCMMSGF	Monitor SESS requested.
		Byte 1		Reserved.
62	(3E)	1	UCMXOR	Reserved.
63	(3F)	1	UCMDEVC	Device control flags.
		1...	UCMDEVA	Full screen on graphics console.
		.1..	UCMDEVB	Prepare command issued.
		..1.	UCMDEVCC	Tested for console switch.
		...1	UCMDEVD	DOM issued.
		.... 1...	UCMDEVE	I/O complete.
		.... .1..	UCMDEVF	Modified DCM for DOM.
		.... ..1.	UCMDEVG	Halt I/O on 2740.
64	(40)	4	UCMMLAST	Address of last minor WQE serviced.
68	(44)	1	UCMSDS5	Status display flags.
		1...	UCMSDS5A	MLWTO needed to keep writing.
		.1..	UCMSDS5B	Inline output pending.
		..1.	UCMSDS5C	Out-of-line output pending.
		...1	UCMSDS5D	Transient DCM blocked.
		.... 1...	UCMSDS5E	Transient DCM locked.
		.... .1..	UCMSDS5F	UCMMLAST field valid.
		.... ..1.	UCMSDS5G	I/O hardware in output-only status.
69	(45)	3	UCMRCT	Address of routing control table.
72	(48)	144	UCMINPUT	

UCM MESSAGE TEXT AREA (INCLUDED WITH MCS AND USER EXIT ONLY)

Pointed to by UCMTPPTR (in MCS Prefix).

Offset		Bytes and Bit Pattern	Field Name	Field Description
0	(0)	128	UCMMSTXT	Message text.
128	(80)	4	UCMROUTC	Routing codes.
132	(84)	4	UCMDESCD	Descriptor codes.
136	(88)	72	UCMXTSAV	Save area for IEECMWSV interface.
208	(D0)	72	UCMTPSAV	Save area for 2740 Device Support Processor (included only when 2740 is used as a system console).

WRITE QUEUE ELEMENT (WQE) SINGLE-LINE

Offset		Bytes and Bit Pattern	Field Name	Field Description
0	(0)	1	WQEUSE	Use count.
1	(1)	3	WQELKPTR	Address of next WQE.
4	(4)	4	WQENBR	Message length.
8	(8)	128	WQETXT	Message text (the message may not occupy all of the 128 bytes).
136	(88)	1	WQEXA	Disposition flags.
		1...	WQEPURGE	Purge.
		.1..	WQEQFHC	Queue for hard copy.
		..1.	WQERQE	RQE exists for this WQE.
		...1	WQEQDFHC	Queued for hard copy.
		.... 1...		WQE is for a WTOR.
137	(89)	2		Reserved.
139	(8B)	1	WQEAVAIL	Buffer status flags.
		1...	WQEBUFA	Buffer is free.
		.1..	WQEBUFB	Buffer is in use.
		..1.	WQEBUFC	Buffer is reserved.
		...1	WQEBUFD	Buffer obtained dynamically.
		.... 1...	WQEBUFE	Buffer has been serviced.
140	(8C)	4	WQEXB	Reserved.
144	(90)	1	WQERTCT	Routed WQE count.
145	(91)	3	WQESEQN	ID sequence number.
148	(94)	2	WQEMCSF	MCS flags.
		Byte 0		
		1...	WQEMCSA	Routing or descriptor codes exist.
		.1..	WQEMCSB	UCM entry ID passed in register 0.
		..1.	WQEMCSC	Command response (includes hard copy).
		...1	WQEMCSD	WQEMSGTP field in WQE to be used for message identification.
		.... 1...	WQEMCSE	Accepted reply to WTOR.
		.... .1..	WQEMCSFF	Broadcast to all active consoles.
		.... ..1.	WQEMCSG	Queue for hard copy only.
		1	WQEMCSH	Queue to UCM entry passed in register 0.
		Byte 1		
		1...	WQEMCSI	Time stamp exists in message text.
		.... .1..	WQEMCSN	Bypass hard copy queuing.
		.... ..1.	WQEMCSO	Reserved for DOM function.
		1	WEQMCSP	Reserved for graphics.
150	(96)	2	WQEMSGTP	Message type flags.
		Byte 0		
		1...	WQEMSGTA	Display jobnames.
		.1..	WQEMSGTB	Display status.
		Byte 1		Reserved.
152	(98)	2	WQEROUT	Routing Codes.
154	(9A)	2		Reserved.
156	(9C)	1	WQEUCMID	Unique UCM entry ID.

Offset		Bytes and Bit Pattern	Field Name	Field Description
157	(9D)	1	WQEPKE	TCB protection key of WTO issuer (MCS).
158	(9E)	2		Reserved.
160	(A0)	2	WQEDESCD	Descriptor codes.
162	(A2)	2		Reserved.
164	(A4)	4	WQETIME	Timer element.

Offset		Bytes and Bit Pattern	Field Name	Field Description
0	(0)	1	WMJUC	Use count.
1	(1)	3	WMJNXT	Address of the next WQE.
4	(4)	1	WMJMLW	MLWTO flags.
		1...	WMJMLWA	Message displayed.
		.1..	WMJMLWB	Major WQE.
		..1.	WMJMLWC	Minor WQE.
		...1	WMJMLWD	Chain altered.
		.... 1...	WMJMLWE	WTL issued.
		.... ..1.	WMJMLWG	Chain to be serviced.
		1	WMJMLWH	Minor queued has no text.
5	(5)	1	WMJAREA	Display area ID.
6	(6)	2	WMJTXTL	Text length.
8	(8)	72	WMJTXT	Text of first line of message.
80	(50)	40	WMJRES	Reserved.
120	(78)	2	WMJSER	Existing line type.
		Byte 0		
		1...	WMJSERA	C Line in major.
		.1..	WMJSERB	1 label found.
		..1.	WMJSERC	2 LABELS FOUND.
		...1	WMJSERD	Last type was C line.
		.... 1...	WMJSERE	Last type was L line.
		Byte 1		Reserved.
122	(7A)	2	WMJLTYP	Line type of message in text field.
		Byte 0		
		1...	WMJLTYPA	Control line.
		.1..	WMJLTYPB	Label line.
		..1.	WMJLTYPC	Data line.
		...1	WMJLTYPD	End indicator.
		Byte 1		Reserved.
124	(7C)	4	WMJNXTM	Address of the first minor WQE.
128	(80)	4	WMJTCB	Address of the TCB of task that issued WTO.
132	(84)	4	WMJMSGN	Message ID assigned to MLWTO.
136	(88)	1	WMJMDISP	Disposition flags.
		1...	WMJNDISPA	Purge this WQE.
		.1..	WMJMDISPB	Queue for hard copy
		..1.	WMJMDISPC	This WQE has an RQE.
		...1	WMJMDISPD	Queued for hard copy.
		.... 1...	WMJMDISPE	WQE if for a WTOR.
137	(89)	2	WMJTJID	Terminal job ID.
139	(8B)	1	WMJBUF	Buffer status flags.
		1...	WMJBUFA	WQE available.
		.1..	WMJBUFB	WQE in use.
		...1	WMJBUFD	WQE acquired by GETMAIN.
		.... 1...	WMJBUFE	WQE serviced.
140	(8C)	4	WMJRORI	Reserved.

Offset		Bytes and Bit Pattern	Name	Meaning
0	(0)	3	WMJMUC	Use count.
1	(1)	1	WMJMNXT	Address of the next WQE.
4	(4)		WMJMMLW	Flags concerning the multiple-line message represented by the WQE:
		1...	WMJMMLWA	Entire first minor available.
		.1..	WMJMMLWB	Major.
		..1.	WMJMMLWC	Minor.
		...1	WMJMMLWD	Chain altered.
		.... 1...	WMJMMLWE	WTL issued.
		.... .1..	WMJMMLWF	(For IEECMWSV) start at top of chain.
		.... ..1.	WMJMMLWG	Chain to be serviced.
		1	WMJMMLWH	Minor queued has no text.
5	(5)	1	WMJMAREA	Identifier of the display area to which the display is to be routed.
6	(6)	2	WMJMTXTL	Length of the text field.
8	(8)	6	WMJMTS	Time stamp for hardcopied messages.
14	(E)	1	WMJMPAD	Reserved.
15	(F)	4	WMJMRR	Routing codes for hardcopied messages.
19	(13)	1	WMJMPAD1	Reserved.
20	(14)	72	WMJMTXT	Text of the message to be passed to the operator.
92	(5C)	18	WMJMRESA	Reserved.
110	(68)	2	WMJSER	Existing line types
		Byte 0		
		1...	WMJSERA	C line.
		.1..	WMJSERB	1 label found.
		..1.	WMJSERC	2 labels found.
		...1	WMJSERD	Last type was C line.
		.... 1...	WMJSERE	Last type was L line.
112	(70)	8	WMJMCONS	Framing indicators (for display consoles).
120	(78)	2	WMJMRESB	Reserved.
122	(7A)	2	WMJMLTYP	Flags indicating the line type of the line in the text field:
		Byte 0		
		1...	WMJMLTYA	Control line.
		.1..	WMJMLTYB	Label line.
		..1.	WMJMLTYC	Data line.
		...1	WMJMLTYD	End indicator.
124	(7C)	4	WMJMMIN	Address of the first minor WQE chained to the major WQE.
128	(80)	4	WMJMTCB	Address of the TCB of the task that issued the multiple-line WTO macro instruction.
132	(84)	4	WMJMMSGN	Message number assigned to the multiple-line WTO message.

Offset		Bytes and Bit Pattern	Field Name	Field Description
136	(88)	1	WMJMDSP	Disposition flags with the following meanings:
		1...	WMJMDSPA	Purge the WQE.
		.1..	WMJMDSPB	Queue for hardcopy.
		..1.	WMJMDSPC	This WQE has an RQE.
		...1	WMJMDSPD	Queued for hardcopy.
		.... 1...	WMJMDSPE	WQE is for WTOR.
137	(89)	2	WMJMTJID	Reserved.
139	(8B)	1	WMJMBUF	Flags indicating the stauts of the buffer used by the WQE:
		1...	WMJMBUFA	Buffer space available.
		.1..	WMJMBUFB	Buffer in use.
		...1	WMJMBUFD	Buffer acquired by GETMAIN.
		.... 1...	WMJMBUFE	WQE serviced.
140	(8C)	4	WMJMRORI	Reserved.
144	(90)	1	WMJMRTCT	Routed WQE count.
145	(91)	3	WMJMSEQ	Sequence number assigned to the message.
148	(94)	1	WMJMCS1	Flags pertaining to multiple console support (MCS):
		1...	WMJMCS1A	Routing or descriptor codes exist.
		.1..	WMJMCS1B	UCM Entry ID passed in register 0.
		..1.	WMJMCS1C	Command response (hard copy).
		...1	WMJMCS1D	WMJMMT1 field indicates message type.
		.... 1...	WMJMCS1E	Accepted reply to WTOR.
		.... .1..	WMJMCS1F	Queue to all active consoles.
		.... ..1.	WMJMCS1G	Queue to hard copy only.
		1	WMJMCS1H	Queue to UCM Entry passed in register 0.
149	(95)	1	WMJMCS2	Flags pertaining to multiple console support (MCS):
		1...	WMJMCS2A	Time stamp in message text.
		.1..	WMJMCS2B	WQE represents a multiple-line message.
		.... .1..	WMJMCS2F	Bypass hardcopy queuing.
		.... ..1.	WMJMCS2G	Reserved for DOM.
		1	WMJMCS2H	Reserved for graphics.
150	(96)	1	WMJMMT1	Flags indicating the type of message the WQE represents:
		1...	WMJMMT1A	DISPLAY JOBNAMES
		.1..	WMJMMT1B	DISPLAY STATUS
		..1.	WMJMMT1C	MONITOR ACTIVE
		.... 1...	WMJMMT1E	Show requested
		.... .1..	WMJMMT1F	MONITOR SESS
151	(97)		WMJMMT2	Reserved.
152	(98)		WMJMRTC	Routing codes assigned to the message.
156	(9C)		WMJMUID	UCM Entry ID.
157	(9D)		WMJMTID	TCB key of the task that issued WTO/R.
160	(A0)		WMJMDEC	Descriptor codes assigned to the message.
164	(A4)		WMJMTIM	Timer element.

WRITE QUEUE ELEMENT (WQE) MINOR NON-MCS

Offset		Bytes and Bit Pattern	Field Name	Field Descriptions
0	(0)	1	WMNUC	Use count for the minor WQE. This use count is set equal to the use count of the major WQE when the MLWTO is queued. As each console finishes with the line represented by the minor WQE, the use count is decremented by 1. When the use count equals zero, the WQE is ready to be removed from the system.
1	(1)	3	WMNEXT	Address of the next minor WQE.
4	(4)	1	WMNMLW	Flags pertaining to the message represented by the minor WQE:
		1...	WMNMLWA	Message displayed
		.1..	WMNMLWB	Major
		..1.	WMNMLWC	Minor
		...1	WMNMLWD	Chain altered
		.... 1...	WMNMLWE	WTL issued
		.... ..1.	WMNMLWG	Chain to be serviced
		1	WMNMLWH	Minor GETMAINed
5	(5)	2	WMNLT	Flags indicating the line type of the text field:
		Byte 0		
		1...	WMNLTA	Control line
		.1..	WMNLTB	Label line
		..1.	WMNLTC	Data line
		...1	WMNLTD	End indicator
		Byte 1		Reserved
7	(7)	1	WMNTXL	Length of the message in the text field.
8	(8)	4	WMNHCT	Hardcopy identifier.
12	(C)	72	WMNTXT	Message to be passed to the operator.
84	(54)	52	WMNRESA	Reserved.
136	(88)		WMNDISP	Disposition flags with the following meanings:
		1...	WMNDISPA	Purge this queue
		.1..	WMNDISPB	Queue for hardcopy
		..1.	WMNDISPC	This WQE has an RQE
		...1	WMNDISPD	Queued for hardcopy
		.... 1...	WMNDISPE	This WQE is for a WTOR
137	(89)	2	WMNTJID	Reserved.
139	(8B)	1	WMNBUF	Flags indicating the status of the buffer used by the WQE:
		1...	WMNBUFA	Buffer available
		.1..	WMNBUFB	Buffer in use
		...1	WMNBUFD	Buffer acquired by GETMAIN
		.... 1...	WMNBUFE	WQE serviced
140	(8C)	4	WMNRORI	Reserved

WRITE QUEUE ELEMENT (WQE) MINOR MCS

Offset		Bytes and Bit Pattern	Field Name	Field Descriptions
0	(0)	1	WMNMUC1	Use count for the message in the Text 1 field.
1	(1)	3	WMNMNX1	Address of the second half of the WQE WMNMUC2).
4	(4)	1	WMNMML1	Multiple-line WTO flags pertaining to the message in the Text 1 field:
		.1..	WMNMML1B	Major
		..1.	WMNMML1C	Minor
		...1	WMNMML1D	Chain altered
		.... 1...	WMNMML1E	WTL issued
		.... .1..	WMNMML1F	Emergency Minor for ABEND/PURGE
		.... ..1.	WMNMML1G	Chain to be serviced
		1	WMNMML1H	Minor GETMAINed
5	(5)	2	WMNMLT1	Flags indicating the line type of the message in the Text 1 field:
		Byte 0		
		1...	WMNMLT1A	Control
		.1..	WMNMLT1B	Label line
		..1.	WMNMLT1C	Data line
		...1	WMNMLT1D	End indicator
		Byte 1		Reserved
7	(7)	1	WMNMTL	Length of the message in the Text 1 field.
8	(8)	4	WMNHCT1	Hardcopy ID for the message in the Text 1 field.
12	(C)	72	WMNMTXT1	First message to be passed to the operator by this WQE.
84	(54)	1	WMNMUC2	Use count for the message in the Text 2 field.
85	(55)	3	WMNMNX2	Address of the next minor WQE.
88	(58)	1	WMNMML2	Multiple-line WTO flags pertaining to the message in the Text 2 field (see flags field at offset X'4', above).
89	(59)	2	WMNMLT2	Flags indicating the line type of the message in the Text 2 field (see flags field at offset X'5', above).
91	(5B)	1	WMNMTL2	Length of the message in the Text 2 field.
92	(5C)	4	WMNHCT2	Hardcopy ID for the message in Text 2 field.
96	(60)	72	WMNMTXT2	Contains the second message to be passed to the operator by this WQE.

Offset		Bytes and Bit Pattern	Field Name	Field Description
-8	(-8)	1		Length of reply.
-7	(-7)	3		Address of requester's reply buffer.
-4	(-4)	4		Address of requester's reply ECB.
0	(0)	1		Reserved.
1	(1)	1		Output message length.
2	(2)	2		MCS Flags
		Byte 0		
		1...		Routing and descriptor codes exist.
		.1..		UCM entry ID passed in register 0.
		..1.		Command response.
		...1		Message type flags exist.
		.... 1...		This is a reply to a WTOR.
		.... .1..		Broadcast to all active consoles.
		.... ..1.		Queue for hard copy only.
		1		Queue to UCM entry (ID passed in register 0).
		0000 0000		No routing or descriptor fields exist.
		Byte 1		
		1...		Time stamp exists in message text.
		.... 1...		Multiple-line WTO
		.... .1..		Bypass hard copy queuing.
		.... ..1.		Reserved for DOM function
4	(4)	128		Message ID and message text.
132	(84)	2		Descriptor codes.
134	(86)	2		Routing codes.
136	(88)	1		Message type flags.
		1...		Display jobnames.
		.1..		Display status.
137	(89)	1		Reserved.
138	(8A)	2		An SVC 35 instruction.

This section provides the ABEND and ABDUMP entry/exit tables, and input (register contents and parameter lists) to the modules that service the macro instructions. This information supplements that found in the Programmer's Guide to Debugging.

INPUT TO ABEND MODULES

Register 1
- Option Flags (byte 1)
- ABEND Completion Code (bytes 2-4)

REGISTER PARAMETERS

Register	Bytes and Bit Patterns	Field Descriptions
1	1	
	1...	Request is for a dump of all main storage assigned to the task. If a //SYSABEND DD statement is provided, all control blocks for the task will be displayed.
	.1..	The job step of the active task is to be abnormally terminated.
	3	A completion code not greater than 4095.

INPUT TO THE ATTACH MODULE

Register 1
(when user specifies PARAM= operand, otherwise unaltered): Address of Optional User Parameter List.

Register 15
ATTLIST (Address of ATTACH Parameter List)

ATTACH Parameter List

Offset		Bytes and Bit Patterns	Field Name	Field Descriptions
0	(0)	1	ATTDE	
		0000 0000		EP or EPLOC specified.
		1000 0000		DE specified.
1	(1)	3	ATTEP	For EP or EPLOC: address of symbolic entry point name. For DE: address of the directory entry.
4	(4)	1	ATTHI	
		0000 0000		No hierarchy was specified.
		0000 0001		Hierarchy zero was specified.
		0000 0010		Hierarchy one was specified.
5	(5)	3	ATTDCB	Address of the DCB for the PDS containing the entry point name. An address of zero indicates that the JOBLIB or LINKLIB DCB is to be used. If the operand is omitted, zero is used.
8	(8)	1		Reserved.
9	(9)	3	ATTECB	Address of the ECB to be used by the control program to indicate the termination of the new task. The return code or completion code will also be placed in the ECB.
12	(C)	1	ATTGSLI	
		0000 0000		Bytes 13-15 contain a subpool number.
		0000 0001		Bytes 13-15 contain the address of a list of subpool numbers.
13	(D)	3	ATTGSLV	A subpool number, or the address of a list. The subpool(s) specified are assigned to the new task and can no longer be used by the originating task. If list is specified, the first byte of the list contains the number of remaining bytes in the list.
16	(10)	1	ATTSSLI	
		0000 0000		Bytes 17-19 contain a subpool number.
		0000 0001		Bytes 17-19 contain a list of subpool numbers.
17	(11)	3	ATTSSLV	A subpool number or the address of a list of subpool numbers. The subpool(s) may be shared by the new task and the originating task. If list is specified, the first byte of the list contains the number of remaining bytes in the list.
20	(14)	1		Reserved.
21	(15)	3	ATTETXR	Address of the end-of-task exit routine to be given control when the new task terminates.

Offset		Bytes and Bit Patterns	Field Name	Field Descriptions
24	(18)	2	ATTDPMOD	The signed number to be algebraically added to the current dispatching priority of the originating task. The result is assigned as the dispatching priority of the new task, unless it is greater than limit priority of the new task, in which case the limit priority of the new task is assigned as the dispatching priority. If the field is 0, the smaller of the new task's limit priority and the dispatching priority of the originating task is assigned.
26	(1A)	1	ATTLPMOD	The number to be subtracted from the limit priority of the originating task. The result is assigned as the limit priority of the new task. If the field is zero, the limit priority of the originating task is assigned.
27	(1B)	1	ATTKEYS	
		x...		Reserved.
		.0..		Propagate the JSCB field from the originating TCB.
		.1..		If the originating task's protect key is zero, move the specified JSCB address (see byte 37) into the attached TCB. Otherwise, propagate the originating task's TCBJSCB field.
		..0.		Subpools 251 and 252 and the job pack queue pointer (TCBJPQ) of the originating task are not to be given to the attached subtask.
		..1.		Subpools 251 and 252 and the job pack queue pointer (TCBJPQ) of the originating task are to be given to the attached subtask.
		...0		The attached task is to have a protect key equal to that of the originating task.
		...1		The attached task is to have a protect key of zero.
		.... 0...		Subpool zero is to be shared with other tasks.
		.... 1...		Subpool zero is not to be shared.
		.... .0..		A save area of 72 bytes is to be obtained for the task.
		.... .1..		No save area is to be obtained.
		.... ..0.		Propagate the TCBJSTCB field from the originating task.
		.... ..1.		The TCBJSTCB of the new task is to point to the new task.
		0		The new task is to operate in problem-program mode.
		1		The new task is to operate in supervisor mode.
28	(1C)	8	ATTNAME	The entry point name if EP was specified. This field contains blanks if EPLOC or DE was specified.
36	(24)	1		Reserved. (This field is present only when the JSCB operand is specified.)
37	(25)	3	ATTJSCB	Address of the job step control block.

Optional User Parameter List

Offset		Bytes and Bit Pattern	Field Name	Field Description
0	(0)	4		First address parameter.
4	(4)	4		Second address parameter.
n		1		
		1...		This is the last entry in the list.
n+1		3		Last address parameter.

INPUT TO THE CHAP MODULE

Register 0
CHAPVAL, Dispatching Priority Change Value

Register 1
CHAPARM, Zero or Address of user parameter list which contains address of TCB.

Register Contents

Register	Bytes and Alignment	Field Description
0	1	
	0000 0000	A positive value was specified.
	0001 0001	A negative value was specified. Register 0 contains the complement of the absolute value of the negative number.
	3	The value to be added to the tasks Dispatching Priority.

INPUT TO THE DELETE MODULE

Register 0
DELADDR, Address of DELETE parameter list.

DELETE Parameter List (Created by User)

Delete Parameter List

Offset	Bytes and Bit Pattern	Field Name	Field Description
0 (0)	8	DELNAME	The eight byte entry point name of module to be deleted. The field must be padded with blanks if the name is less than eight characters in length.

INPUT TO THE DEQ MODULE

Register 1
DEQLIST, Address of DEQ Parameter List

DEQ Parameter List
(generated by the ENQ macro instruction) The DEQ Parameter List may be extended in multiples of 3 or 4 words (because of parameter field ADUCB).

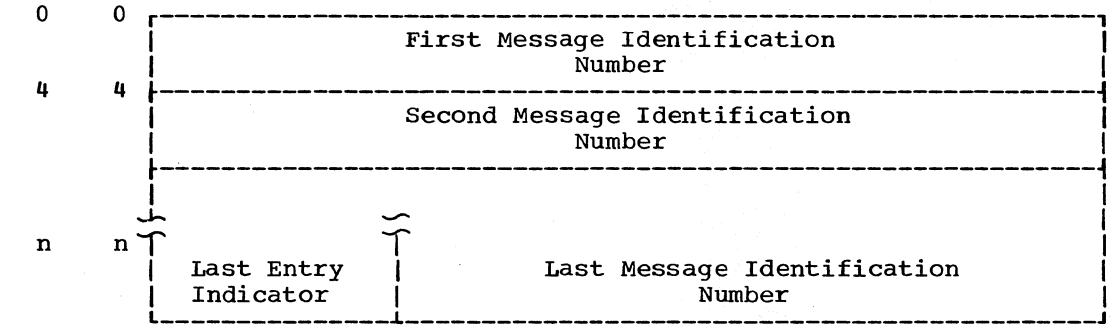
Offset		Bytes and Bit Pattern	Field Name	Field Description
0	(0)	1	LISTEND	
		1111 1111		This is the last entry in the list.
1	(1)	1	LMINOR	The length, in bytes, of the Minor Resource Name.
2	(2)	1	PARMCDS	
		0...		Set to zero with a DEQ request.
		.0.. 0...		The request is for a resource of this job step.
		.0.. 1...		The request is for a reserved resource.
		.1.. 0...		The request is for a resource known to the system.
		.1.. 1...		The request is for a resource known across systems.
		..1		"Reset System Must Complete" is requested.
		...1		"Reset Step Must Complete" is requested.
		.... .000		This is an unconditional request to release the resource. The task will be abnormally terminated if it has not been assigned the resource.
		.... .001		This is a conditional request to release the resource only if it has been assigned to the task.
3	(3)	1	RETURN	This field is used by the control program to return the return code for conditional requests.
4	(4)	1		Reserved.
5	(5)	3	ADMAJOR	The address of an 8-byte field containing the Major Resource Name.
8	(8)	1		Reserved.
9	(9)	3	ADMINOR	The address of an 1-255 byte field containing the Minor Resource Name.
n	(n)	1		Reserved. (This field is only present when a reserved resource is to be dequeued.)
		3	ADUCB	The address of a word containing the UCB address for the device to be dequeued. (This field is only present when a reserved device is to be dequeued.)

INPUT TO THE DOM MODULE

Register 0
DOMFLG, DOM Message or Message List Indicator

Register 1
DOMMSG, DOM Message Identification Number or Address of Optional User Parameter List

Optional User Parameter List



Register Parameters

Register	Bytes and Bit Pattern	Field Descriptions
0	1	
	1...	Indicates that a list of identification numbers has been specified.
	0...	Indicates that only one identification number has been specified (register 1).
1	4	Address of a list of identification numbers; or a 24-bit right-adjusted identification number of the message to be deleted.

Parameter List

Offset		Bytes and Bit Pattern	Field Description
0	(0)	4	24-bit, right-adjusted identification number of the message to be deleted.
n	(n)	1	
		1...	Indicates last message in the list.

INPUT TO THE ENQ MODULE

Register 1
ENQLIST, Address of ENQ Parameter List

ENQ Parameter List

Offset		Bytes and Bit Pattern	Field Name	Field Description
0	(0)	1	LISTEND	
		1111 1111		This is the last entry in the list.
1	(1)	1	LMINOR	The length, in bytes, of the Minor Resource Name.
2	(2)	1	PARMCDS	
		0...		This request is for exclusive control of the resource.
		1...		This request is for shared control of the resource.
		.0.. 0...		The resource is used only by this job step.
		.1.. 0...		The resource may be used by more than one job step.
		.1.. 1...		The request is for a resource known across systems.
		..1.		"Set System Must Complete" is requested.
		...1		"Set Step Must Complete" is requested.
		.... .111		The request is for the availability status of the resource(s), but not actual control.
		.... .110		The request is to change the attribute of the resource from shared to exclusive, only if the requester can have immediate exclusive control (requester is already enqueued for the resource).
		.... .011		Control of the resource(s) is requested only if the task can have immediate control of it.
		.... .001		Control of the resource(s) is requested only if the task has not previously requested it.
		.... .000		This is for an unconditional request for the resources. If any of them are not available, the task is put into a wait state until they are available.
3	(3)	1	RETURN	This field is used by the control program to return the return codes for conditional requests.
4	(4)	1		Reserved.
5	(5)	3	ADMAJOR	The address of an 8-byte field containing the Major Resource Name of the resource requested.
8	(8)	1		Reserved.
9	(9)	3	ADMINOR	The address of an 1-255 byte field containing the Minor Resource Name of the resource requested.

INPUT TO THE EXTRACT MODULE

Register 1
EXTRPRM, Address of EXTRACT Parameter List.

EXTRACT Parameter List

Offset		Bytes and Bit Pattern	Field Name	Field Descriptions
0	(0)	4	EXTRLST	Address of the user's answer area in which the extracted fields are placed.
4	(4)	4	EXTRTCB	Address of the TCB from which the fields are to be extracted. A zero in this field indicates that the current TCB is the subject.
8	(8)	1	EXTRFLG	Flags indicating which TCB fields are to be extracted.
		1...		GRS -- Address of the general register save area.
		.1..		FRS -- Address of the floating point register save area.
		..1.		TAX -- Contents of TCB+148 (X'94'); this is presently reserved.
		...1		AETX -- Address of the end-of-task exit routine specified in the ATTACH macro instruction.
		.... 1...		PRI -- Limit and dispatching priorities of the task; these will be placed in the last two bytes of the answer area.
		.... .1..		CMC -- Task completion code; zero if task is not complete.
		.... ..1.		TIOT -- Address of the Task I/O Table.
		1		COMM -- Address of Communications Parameter Area.
9	(9)	3		Reserved.

INPUT TO FREEMAIN MODULE

Register Type Request

Register 0
FRESP, Number of the Subpool from which Area is to be Freed (1 byte).
FRELNGTH, Number of Bytes to be Freed (3 bytes).

Register 1
FREAD, The beginning address of the area to be freed.

Element and Variable Type Request

Register 1
FREPARM, Address of FREEMAIN Parameter List.

FREEMAIN Parameter List

Offset		Bytes and Bit Pattern	Field Name	Field Descriptions
0	(0)	1		Reserved.
1	(1)	3	FRELNGTH	Zero indicates a variable type request. For element type requests, this is the length to be freed.
4	(4)	1		Reserved.
5	(5)	3	FREADD	The address of a list containing the beginning addresses of the area(s) to be freed. The high order bit in the last fullword in the list is set to 1.
8	(8)	1	FRETYPE	
		0000 0000		Indicates a request for the release of a single area of storage.
		1...		Indicates a request for the release of one or more areas of storage.
		11..		Indicates a variable type request for the release of one area of storage. The address and the length are in the length list.
9	(9)	1	FRESPID	The number of the subpool from which the storage is to be freed.

INPUT TO GETMAIN MODULE

Register Type Request

Register 0
GETSP, Subpool Number (byte 1).
GETLNGTH, Length Requested (bytes 2-4).

Register 1
GETID, Negative to indicate GETMAIN request.

Element and Variable Type Request

Register 1
GETPARM, Address of GETMAIN Parameter List.

GETMAIN Parameter List

Offset		Bytes and Bit Pattern	Field Name	Field Descriptions
0	(0)	1		Reserved.
1	(1)	3	GETLNGTH	For element requests, this is the length in bytes of the amount of storage requested. For list and variable requests, it is the address of a list of requested lengths. The high order bit of the last word is set to one to indicate the end of the list.
4	(4)	1	GETHID	The Hierarchy from which the storage is to be obtained.
5	(5)	3	GETADD	Reserved.
8	(8)	1	GETYPE	
		0010 0000		Conditional single element request.
		0000 0000		Unconditional single element request.
		1010 0000		Conditional list request.
		1000 0000		Unconditional list request.
		1110 0000		Conditional variable request.
		1100 0000		Unconditional variable request.
9	(9)	1	GETSPID	The number of the subpool from which the requested core is to be allocated.

Length Parameter List

Offset		Bytes and Bit Pattern	Field Name	Field Descriptions
0	(0)	1		Reserved.
1	(1)	3	GETLNG1	The length, in bytes, of the main storage requested. For a variable request, this field contains the minimum storage requested.
4	(4)			Reserved.
5	(5)	3	GETLNG2	The length, in bytes, of the next area of main storage requested. For a variable request, this is the maximum amount of storage requested.
n	(n)	1		
		1000 0000		This indicates that this is the last word in the list.

User Supplied Address List

Offset	Bytes	Field Name	Field Descriptions
0 (0)	1		Reserved.
1 (1)	3	GETADD1	GETMAIN is to return the address of the first area allocated in this field.
4 (4)			Reserved.
5 (5)	3	GETADD2	For a variable request, it is used to return the length of the area allocated.

INPUT TO IDENTIFY MODULE

Register 0
IDEP, Address of Entry Point Symbolic Name.

Register 1
IDENTRY, Main Storage Address of the Entry Point Being Added.

Entry Point Symbolic Name
(supplied by user [EPLOC=], or generated by macro [EP=])

Offset	Bytes	Field Name	Field Descriptions
0 (0)	8	IDNAME	The eight-byte symbolic entry point name to be added to the module. The name may be padded with blanks, if necessary.

INPUT TO LINK MODULE

Register 1
LINPARM, Address of Optional User Parameter List.

Register 15
LINKPARM, Address of LINK Parameter List.

User Optional Parameter List

Offset		Bytes and Bit Pattern	Field Name	Field Descriptions
0	(0)	1		Reserved.
1	(1)	3		The first user-supplied address parameter.
n	(n)	1		
		1...		VL=1 specified. This indicates that this is the last word in the parameter list.
n+1	(n+1)	3		The last user-supplied address parameter.

LINK Parameter List

Offset		Bytes and Bit Pattern	Field Name	Field Descriptions
0	(0)	1	LINKFLG	
		0000 0000		EP or EPLOC forms specified.
		1000 0000		DE form specified.
1	(1)	3	LINKEP	For the EP or EPLOC forms, this field contains the address of a doubleword containing the symbolic entry point name for the called routine. For the DE form, this field contains the address of the DE list entry.
4	(4)	1	LINKHI	
		0000 0000		No hierarchy was specified.
		0000 0001		Hierarchy zero was specified.
		0000 0010		Hierarchy one was specified.
5	(5)	3	LINKDCB	The address of the specified DCB. Zero indicates that the DCB address was not specified. The JOBLIB and LINKLIB DCBs will be assumed.

Symbolic Name

Offset		Bytes and Bit Pattern	Field Name	Field Descriptions
0	(0)	8	LINKNAME	This field contains the symbolic entry point name of the called routine. If the name is less than 8 characters in length, it is padded with blanks.

INPUT TO LOAD MODULE

Register 0
LOADDE, Directory Entry Indicator (byte 1).
LOADEP, Address of Symbolic Entry Point Name; or Two's Complement of Address (bytes 2-4).

Register 1
LOADHI, Hierarchy Indicator (byte 1).
LOADDCB, Address of DCB (bytes 2-4).

Register Parameters

Register	Bytes and Bit Pattern	Field Descriptions
0	1	
	0000 0000	The EP= or EPLOC= operands were specified.
	1111 1111	The DE= operand was specified.
	3	If EP= or EPLOC= was specified, this field contains the address of the Symbolic Entry Point Name. If the name is less than 8 characters in length, it is padded with blanks. If DE= was specified, this field contains the two's complement form of the address of the directory entry. (Note: The LOAD service routine (SVC 8) converts the DE address back to a positive format.)
1	1	
	0000 0000	No hierarchy was specified.
	0000 0001	Hierarchy zero was specified.
	0000 0010	Hierarchy one was specified.
	3	Address of the DCB. Zero indicates that the JOBLIB or LINKLIB DCB is to be used.

Symbolic Entry Point Name
(supplied by user [EPLOC= or DE=], or by Macro [EP=])

Offset	Bytes and Bit Pattern	Field Name	Field Descriptions
0 (0)	8	LOADNAM	This field contains the symbolic entry point name of the called routine. If the name is less than eight characters in length, the field is padded with blanks.

INPUT TO THE POST MODULE

Register 0
POSTCDE, Code to be Posted in the ECB.

Register 1
POSTECB, if high-order bit is on, points to an address in main storage that contains the address of an ECB.

Register 11
address of ECB on branch entry to POST.

INPUT TO THE RETURN MODULE

Register Contents

Register	Bytes	Field Descriptions
13	4	Address of the save area in which registers were previously saved.
14	4	Address in the calling program to which control is to be returned. This register must be loaded by the user if register 14 is not within the range of registers specified to be restored by the macro.
15	3	Return code if the RC=number operand is not used. Use of the RC=(15) operand will override a request to restore register 15 from the save area.

INPUT TO SAVE MODULE

Register 13 (user-supplied)
SAVADDR, Address of Save Area.

Register 15
MODIDAD, Address of SAVE Macro Instruction. This is used to locate the module identifier which is created at SAVE+4.

Module Identifier Parameter List

Offset	Bytes	Field Name	Field Descriptions
0	1	SAVCNT	Length, in bytes, of the identifier name.
1		SAVID	The name given by the user as the identify operand in the macro. This may be 1 to 70 characters long.

INPUT TO SNAP MODULE

Register 1
SNAPARM, Address of SNAP Parameter List

SNAP Parameter List

Offset		Bytes and Bit Pattern	Field Name	Field Descriptions
0	(0)	1	SNAPID	ID number to be printed in the identification heading of the dump.
1	(1)	1		Reserved.
2	(2)	1	SNAPFLG1	
		1...		Always set to one by Snap macro.
		.0..		Dump is for active task.
		.1..		Dump is for specified subtask.
		..0.		Do not include the control blocks in the dump.
		..1.		Include the control blocks in the dump.
		...0		Do not include the trace table in the dump.
		...1		Include the trace table in the dump.
		.... 0...		Do not include the nucleus.
		.... 1...		Include all the nucleus except the trace table.
		.... .0..		No list of addresses was supplied.
		.... .1..		A list of addresses specifying the storage areas to be dumped has been supplied.
		.... ..0.		No ID number was specified.
		.... ..1.		An ID number is to be printed in the dump.
		0		No queue control blocks or queue elements are to be printed.
		1		The queue control blocks and queue elements are to be printed.
3	(3)		SNAPFLG2	
		0...		Do not provide a back trace of the save areas.
		1...		Provide linkage information and a back trace of the save areas.
		.0..		SAH not specified.
		.1..		SAH specified; provide linkage information.
		..0.		Do not provide register contents.
		..1.		Provide the contents of the general registers at the time that the SNAP was issued.
		...0		Do not provide LPA information.
		...1		Provide the contents of the Link Pack Area.
		.... 0...		Do not provide JPA information.
		.... 1...		Provide the contents of the Job Pack Area.
		.... .0..		Do not provide the PSW.
		.... .1..		Provide the Program Status Word at the time that the SNAP was issued.
		.... ..0.		Do not supply the subpools.
		.... ..1.		Provide all subpools (0-127).
		0		Always set to 0.
4	(4)	1		Reserved.
5	(5)	3	SNAPDCB	The address of the DCB for the data set to contain the dump.
8	(8)	1		Reserved.
9	(9)	3	SNAPTCB	Zero indicates dump is for the active task. Otherwise address of the TCB for which the dump is to be taken.
12	(C)	1		Reserved.
13	(D)	3	SNAPSTOR	Zero indicates no addresses were specified for the areas to be dumped. Otherwise, address of the storage parameter list containing the starting and ending addresses of the areas to be dumped.

INPUT TO SPIE MODULE

Register 1
SPIADD, Address of PICA.

PICA Parameter List

Offset		Bytes and Bit Pattern	Field Name	Field Descriptions
0	(0)	1	SPIMSK	Program Mask.
1	(1)	3	SPIEXIT	The address of the user-specified exit routine.
4	(4)	2	SPINT	Program interruption mask.

INPUT TO STAE MODULE

Register 0
STAEOPT, Create/Overlay Option Indicator.

Register 1
STAEXTL, XCTL Indicator (byte 1).
STAELST, Address of STAE Parameter List (bytes 2-4).

Register Parameters

Register	Bytes and Bit Pattern	Field Description
0	1	Create/Overlay Indicator.
	0000 0000	A new STAE parameter list is to be created.
	0000 0100	Cancel the most recent STAE request.
	0000 1000	Overlay the previous STAE parameter list with the parameters passed in this request.
1	1	XCTL indicator.
	0000 0000	Indicates that the STAE macro instruction is to be canceled if an XCTL is issued by this program. If the XCTL operand is not specified, this option is assumed.
	1000 0000	Indicates that the STAE macro instruction is not to be canceled if an XCTL is issued by this program.
	3	The address of the STAE parameter list.

STAE Parameter List

Offset		Bytes and Bit Pattern	Field Name	Field Description
0	(0)	4	STAEXIT	Address of STAE exit routine.
4	(4)	4	STAEPRM	Address of parameter list to be passed to STAE exit routine.

INPUT TO STIMER MODULE

Register 0
STIFLG (byte 1).
STIEXIT, Address of Exit Routine (bytes 2-4).

Register 1
STIVAL, Address of STIMER Parameter List.

Register Parameters

Register	Bytes and Bit Pattern	Field Descriptions
0	1	
	0000	TUINTVL specified - the time interval is specified in timer units.
	0001	BINTVL specified - the time interval is specified as a binary number.
	0011	DINTVL specified - the time interval is specified as a decimal number.
	0111	TOD specified - the time of day is specified in the form HHMMSSth.
	 0000	The interval is to be decremented only when the task is active.
	 0001	The interval is to be decremented continuously and the current task is to be placed in a wait condition until the interval expires.
	 0011	The interval is to be decremented continuously.
	3	The address of a user exit routine to be given control when the time interval expires. Zero indicates that no exit address was specified.
1	1	Reserved.
	3	For DINTVL and TOD, this is the address of a doubleword containing the time value. For BINTVL and TUINTVL this is the address of a fullword containing the time interval.

STIMER Parameter List (depends on BINTVL, TUINTVL, DINTVL, or TOD operand specified)

Offset	Bytes and Bit Pattern	Field Name	Field Descriptions
0 (0)	4	STIMTU	Contains the time interval as an unsigned 32-bit binary number where the low-order bit represents one timer unit (26.04 microseconds).
0 (0)	4	STIMBIN	Contains the time interval as an unsigned 32-bit binary number where the low-order bit represents 0.01 seconds.
0 (0)	8	STIMDEC	Contains the time interval as unpacked decimal digits in the form HHMMSSth where: HH is hours. MM is minutes SS is seconds t is tenths of seconds h is hundredths of seconds.
0 (0)	8	STIMTOD	Contains the time of day when the interval is to be completed. The interval is specified as unpacked decimal digits in the form HHMMSSth.

INPUT TO TIME MODULE

Register 1
TIMUNIT, Type of Unit Requested.

Register Parameters

Register	Bytes and Bit Pattern	Field Descriptions
1	3	Reserved.
	1	
	0000 0000	The time of day is to be returned in register 0 as an unsigned 32-bit binary number. (The least significant bit represents 26.04 microseconds.)
	0000 0001	The time of day is to be returned as an unsigned 32-bit binary number. (The least significant digit represents 0.01 seconds.)
	0000 0010	The time is to be returned as packed decimal digits in the form HHMMSSth where: HH is hours MM is minutes SS is seconds t is tenths of seconds h is hundredths of seconds.

INPUT TO TTIMER MODULE

Register 1
TTICODE, Type of Request.

Register Parameters

Register	Bytes and Bit Pattern	Field Descriptions
1	3	Reserved.
	1	
	0000 0000	The time remaining in the current task's time interval is to be returned in register 0 (the interval is not to be canceled).
	0000 0001	The current task's time interval is to be canceled.
	0000 0010	The time interval of a related task is to be canceled.

INPUT TO WAIT MODULE

Register 0
WAITYP, Wait Indicator (bytes 1-3).
WAITNUM, Number of Events (byte 4).

Register 1
WAITFLG, Element/List Indicator (byte 1).
WAITECB, Address of ECB or ECB List (bytes 2-4).

Register Parameters

Register	Bytes and Bit Pattern	Field Descriptions
0	3	Zero indicates the Wait Macro was used.
	1	The number of events which are to be awaited. This number must be less than 256.
1	1	
	0000 0000	One ECB has been specified.
	0000 0001	A list of ECBs has been specified.
	3	The address of an ECB or a list of ECB addresses. (If a list is specified, the address is in complemented form. The high-order bit of the last entry must be set to one.)

INPUT TO WTO MODULE

Register 0
(used only when QREG0 is specified in MCSFLAG operand)
UCMID, Console ID.

Register 1
WTOPRM, Address of WTO Parameter List.

WTO Parameter List

Offset		Bytes and Bit Pattern	Field Name	Field Descriptions
0	(0)	2	WTOLNG	The length, in bytes, of the WTO message + 4.
2	(2)	2	WTOMCSF	MCS Codes.
		Byte 0:		
		0...		The WTODESC and WTOROUT fields do not exist.
		1...		The WTODESC and WTOROUT fields exist.
		.1..		The message is to be queued to the console whose source ID is passed in Register 0.
		..1.		The WTO is an immediate command response.
		...1		Indicates that the WTOMSGT field exists.
		.... 1...		The WTO macro instruction is a reply to a WTOR macro instruction.
		.... .1..		This message should be broadcast to all active consoles.
		.... ..1.		This message is to be queued for hard copy only.
		1		This message is to be queued unconditionally to the console whose source ID is passed in register 0.
3	(3)	Byte 1:		
		1...		Time is not appended to the message.
		.xxx x...		Reserved.
		.... .1..		This message is not to be queued for hard copy (ignored if issuer does not have a zero protect key).
		xx		Reserved.
4	(4)	Variable	WTOMSG	WTO message to be written on operator's console or WTP message for programmer.
n		2	WTODESC	Descriptor Codes.
		Byte 0:		
		1...		System Failure - another IPL of the system is required.
		.1..		Immediate operator action required.
		..1.		Eventual action required.
		...1		System Status is indicated by the message.
		.... 1...		Immediate command response - for error and nonerror messages that are written as a result of an operator or system command.
		.... .1..		Job Status indicated by the message.
		.... ..1.		Application Program/Processor - for messages issued by problem programs and processors executed as problem programs.
		1		Operator request for status information.
n+1		Byte 1:		
		1...		Out-of-line message.
		.xxx xxxx		Reserved.
n+2		2	WTOROUT	Routing codes.

Offset	Bytes and Bit Pattern	Field Name	Field Descriptions
	Byte 0:		
	1...		Master Console.
	.1..		Master Console Informational.
	..1.		Tape Pool Console.
	...1		Direct Access Pool Console.
	.... 1...		Tape Library Pool Console.
	.... .1..		Disk Library Pool Console.
	.... ..1.		Unit Record Pool Console.
	1		Teleprocessing Control.
	Byte 1:		
	1...		System Security.
	.1..		System Error/Maintenance.
	..1.		Programmer Information Console.
	...1		Reserved.
	.... xxx.		User Routing Codes.
	x		Reserved.
n+4	1	WTOMSGT	Message Type Flags.
	Byte 0:		
	1...		Display Jobnames.
	.1..		Display Status.
	..1.		Monitor Active.
	...x xxxx		Reserved.

INPUT TO MULTIPLE-LINE WTO

Register 0
(used only when QREG0 is specified in MCSFLAG operand). UCMID, console identifier.

Register 1
WTOPRM, address of the WTO parameter list.

MLWTO Parameter List

Offset	Bytes and Bit Pattern	Field Name	Field Description
0 (0)	1 byte	WTOMK1	Marker designating start of the first field of the multiple-line WTO.
1 (1)	1 byte	WTOLNT1	The length, in bytes, of WTOTXT1 + 4.
2 (2)	2	WTOMCSF	MCS codes.
	Byte 0		
	0...		The WTODESC and WTOROUT fields do not exist.
	1...		The WTODESC and WTOROUT fields exist.
	.1..		The message is to be queued to the console whose source ID is passed in register 0.
	..1.		The WTO is an immediate command response.
	...1 1111		Indicates that the WTOMSGT field exists.
	.... 1...		The WTO macro instruction is a reply to a WTOR macro instruction.
	.... .1..		This message should be broadcast to all active consoles.
	.... ..1.		This message is to be queued for hard copy only.
	1		This message is to be queued unconditionally to the console whose source ID is passed in register 0.
3 (3)	Byte 1		
	1...		Time is not appended to the message.
	.xxx x...		Reserved.
	.... .1...		This message is not to be queued for hard copy (ignored if user does not have a zero protect key).
	xx		Reserved.
4 (4)	VARIABLE	WTOTXT1	Text of the first line of the multiple-line WTO message.
n+ 4	2	WTODESC	Descriptor Codes.
	Byte 0		
	1...		System Failure - another IPL of the system is required.
	.1..		Immediate operator action required.
	..1.		Eventual action required.
	...1		System Status is indicated by the message.
	.... 1...		Immediate command response - for error and nonerror messages that are written as a result of an operator or system command.
	.... .1..		Job Status indicated by the message.
	.... ..1.		Application Programmer/Processor - for messages issued by problem programs and processors executed as problem programs.
	x		Operator's request for status information.
n+5	Byte 1		
	1...		Out-of-line message.
	.xxx xxxx		Reserved.

Offset	Bytes and Bit Pattern	Field Name	Field Description
n+6	2	WTOROUT	Routing codes.
	Byte 0		
	1...		Master Console.
	.1..		Master Console Informational.
	..1.		Tape Pool Console.
	...1		Direct Access Pool Console.
	... 1....		Tape Library Pool Console.
	... .1..		Disk Library Pool Console.
	.... ..1.		Unit Record Pool Console.
	1		Teleprocessing Control.
	Byte 1		
	1...		System Security.
	.1..		System Error/Maintenance.
	..1.		Programmer Information Console.
	...1		Reserved.
	.... xxx.		User Routing Codes.
	x		Reserved.
n+8	1	WTOMSGT	Message Type Flags:
	Byte 0		
	1...		Display Jobnames.
	.1..		Display Status.
	..1.		Monitor Active.
	...x xxxx		Reserved.
n+10	2 bytes	WTOLT1	Line type of WTOTXT1 field.
n+12	1 byte	WTOAID	Area identifier of display area to which message is to be routed.
n+13	1 byte	WTOLCT	Number of lines in the entire multiple-line WTO.
n+14	1 byte	WTOMK2	Second line marker (contains zero).
n+15	1 byte	WTOLNT2	Length, in bytes, of WTOTXT2 + 4.
n+16	2 bytes	WTOLT2	Line type of WTOTXT2 field.
n+18	Variable	WTOTXT2	Second line of text of the multiple-line WTO.
	1 byte	WTOMK3	Next line marker (contains zero).
	1 byte	WTOLNT3	Length of WTOTXT3 field.
	2 bytes	WTOLT3	Line type of WTOTXT3 field.
	Variable	WTOTXT3	Third line of text of the multiple-line WTO.

Each subsequent entry in the parameter list will consist of a WTOMKn field, a WTOLNTn field, a WTOLTn field, and a WTOTXTn field.

INPUT TO WTOR MODULE

Register 1
WTORPRM, Address of WTOR Parameter List.

WTOR Parameter List

Offset		Bytes and Bit Pattern	Field Name	Field Descriptions
0	(0)	1	WTORRPL	The length, in bytes, of the area to receive the reply.
1	(1)	3	WTORRPA	The address of the area for the reply.
4	(4)	4	WTORECB	The address of the ECB to be posted by the control program when the reply is completed.
8	(8)	2	WTORLNG	The length, in bytes, of the WTOR message.
10	(A)	2	WTORMCS	MCS codes.
		Byte 0:		
		0...		The WTORDSC and WTORROU parameter fields do not exist.
		1...		The WTORDSC and WTORROU parameter fields exist.
		.1..		The message is to be queued to the console whose source ID is passed in register 0.
		..1.		The WTOR is an immediate command response.
		...1		Indicates that the WTORMGT parameter field exists.
		.... 1...		The WTO macro instruction is a reply to a WTOR macro instruction.
		.... .1..		This message should be broadcast to all active consoles.
		.... ..1.		This message is to be queued for hard copy only.
		1		This message is to be queued unconditionally to the console whose source ID is passed in Register 0.
		Byte 1:		
		1...		Time is not appended to the message.
		.xxx x...		Reserved.
		.... .1..		This message is not to be queued for hard copy (if it went to paper-producing, nongraphic device).
		xx		Reserved.
12	(C)	Variable	WTORMSG	Message to be written on operator's console.
n		2	WTORDSC	Descriptor codes.
		Byte 0:		
		1...		System Failure - another IPL of the system is required.
		.1..		Immediate operator action required.
		..1.		Eventual action required.
		...1		System Status is indicated by the message.
		.... 1...		Immediate command response - for error and nonerror messages that are written as a result of an operator or system command.
		.... .1..		Job Status indicated by the message.
		.... ..1.		Application Program/Processor - for messages issued by problem programs and processors executed as problem programs.
		1		Operator request for status information.
n+1		Byte 1:		
		1...		Out-of-line message.
		.xxx xxxx		Reserved.

Offset	Bytes and Bit Pattern	Field name	Field Descriptions
n+2	2	WTORROU	Routing codes.
	Byte 0:		
	1...		Master Console.
	.1..		Master Console Informational.
	..1.		Tape Pool Console.
	...1		Direct Access Pool Console.
	.... 1...		Tape Library Pool Console.
	.... .1..		Disk Library Pool Console.
	.... ..1.		Unit Record Pool Console.
	1		Teleprocessing Control.
n+3	Byte 1:		
	1...		System Security.
	.1..		System Error/Maintenance.
	..1.		Programmer Information Console.
	...1		Reserved.
	.... xxx.		User Routing Codes.
	x		Reserved.
n+4	1	WTORMGT	Message Type Flags.
	1...		Display Jobnames.
	.1..		Display Status.
	..1.		Monitor Active.
	...x xxxx		Reserved.
n+5	1		Reserved.

INPUT TO XCTL MODULE

Register 15

XCTLPARM, Address of XCTL Parameter List.

XCTL Parameter List

Offset		Bytes and Bit Pattern	Field Name	Field Description
0	(0)	1	XCTLFLG	
		0000 0000		EP or EPLOC specified.
		1000 0000		DE form specified.
1	(1)	3	XCTLEP	For the EP or EPLOC forms, this field contains the address of a doubleword containing the entry point name of the called routine. For the DE form, this field contains the address of the DE list entry.
4	(4)	1	XCTLHI	
		0000 0000		No Hierarchy specified.
		0000 0001		Hierarchy 0 specified.
		0000 0010		Hierarchy 1 specified.
5	(5)	3	XCTLDCB	The address of the DCB. Zero indicates that a DCB was not specified. The Linklib or Joblib DCB will be used.

Symbolic Entry Point Name

Offset		Bytes and Bit Pattern	Field Name	Field Description
0	(0)	8	XCTLNAM	This field contains the entry point name of the called routine. If the name is less than eight characters in length, the field is padded with blanks. When EPLOC is specified, this field is set up by the user. When EP is specified, this field is generated by the macro.

ABEND ENTRY/EXIT TABLE (Part 1 of 2)

Module	Entries	Exits	Reason for Exit
ABEND0 IEANTM00	SLIH ABEND 11	ABEND G SCHEDULER ABEND 11 ABEND 12	Abnormal termination Job step normal termination, no WTORs outstanding Normal end, WTOR purge needed or MCS in system Subtask normal end, no WTOR purge needed
ABEND 1 IEANTM01	ABEND 0 ABEND G	GRAPHICS ASIR 1 ABEND 11	GJP hook Valid STAE Normal ABEND
ABEND 2 IEANTM02	ABEND 11 ABEND G	DAR 1 ABEND 3 ABEND 9	System task or task in must complete status Normal ABEND Valid recursion
ABEND 3 IEANTM03	ABEND 2	ABEND 4 ABEND J	Normal ABEND
ABEND 4 IEANTM04	ABEND 3 ABEND 9 ABEND 10 ABEND H ABEND J	ABEND 5 ABEND 7 ABEND 8 ABEND 10 ABEND 12 ABEND 13 ABEND 15	Full dump Job step, no subtasks Job step, indicative dump Insufficient storage Subtask ABEND, no dump Job step with subtasks, ENQ purge needed Job step with subtasks, no ENQ purge needed
[1]ABEND 5 IEANTM05	ABEND 4	ABEND 6 ABEND 9	SYSABEND or SYSUDUMP statement present in TIOT Failure in OPEN or dump
ABEND 6 IEANTM06	ABEND 5	ABEND 7 ABEND 9 ABEND 12 ABEND 13 ABEND 15	Job step with no subtasks OPEN or dump failed Subtask ABEND, dump complete Job step with subtasks, ENQ purge needed Job step with subtasks, no ENQ purge needed
ABEND 7 IEANTM07	ABEND 4 ABEND 6 ABEND 8 ABEND 15 TCAM ABEND	SCHEDULER TCAM ABEND	Process TCAM abnormal termination
ABEND 8 IEANTM08	ABEND 4	ABEND 7 ABEND 13 ABEND 15	Job step with no subtasks Job step with subtasks, ENQ purge needed Job step with subtasks, no ENQ purge needed
ABEND 9 IEANTM09	ABEND 2 ABEND 5 ABEND 6	ABEND 4 ABEND H	Valid recursion
ABEND 10 IEANTM0A	ABEND 4	ABEND 4	In all cases

[1]IEAMTM05 in MFT without subtasking.

ABEND ENTRY/EXIT TABLE (Part 2 of 2)

Module	Entries	Exits	Reason for Exit
[1]ABEND 11 IEANTM0B	ASIR ABEND 0 ABEND 1	ASIR ABEND 0 ABEND 2 ABEND 12	ASIR was caller, WTOR purge done Jobstep normal end, WTOR purge done Normal ABEND Subtask normal end, WTOR purge done
[2]ABEND 12 IEANTM0C	ABEND 0 ABEND 4 ABEND 6 ABEND 11	ABEND 13	Subtask ABEND
[2]ABEND 13 IEANTM0D	ABEND 4 ABEND 6 ABEND 8 ABEND 12	ABEND 15	Terminate subtasks
[2]ABEND 14 IEANTM0B	ABEND 15	SVC EXIT	
[2]ABEND 15 IEAQTM0F	ABEND 4 ABEND 6 ABEND 8 ABEND 13	ABEND 7 ABEND 14	Close data sets
ABEND G IEANTM0G	ABEND 0 ABEND M	SVC EXIT ASIR 1 DAR 1 DAR 2 ABEND 1 ABEND 2 ABEND M	 STAE Exit routine entry DAR Recursion Invalid DAR recursion Abnormal Termination processing Valid CLOSE recursion after DAR Message List processing
ABEND H IEANTM0H	ABEND 9	ABEND 4	
ABEND J IEANTM0J	ABEND 3	ABEND 4	
ABEND M IEANTM0M	ABEND G	ABEND G	

[1]IEACTM0B in an MCS system.
[2]For MFT without subtasking, modules 12, 13, 14, and 15 are not included in the system.

ABDUMP ENTRY/EXIT TABLE

Module	Entries	Exits	Reason for Exit
SVC DUMP 1 IEAAAD0Y	SVC SLIH (expansion of SNAP)	ABDUMP 1 SVC DUMP 2 EXIT	Call is not for SVC DUMP Dump is to tape Dump in progress; caller not in supervisor state; or after normal dump to non-tape device
SVC DUMP 2 IEAAAD0Z	SVC DUMP 1	EXIT	
ABDUMP 1 IEAMAD00	SVC DUMP 1	ABDUMP 2	In all cases
ABDUMP 2 IEAAAD0D	ABDUMP 1	ABDUMP 3 ABDUMP 5 ABDUMP 6	Supervisor data is requested Save area trace requested Neither requested
ABDUMP 3 IEAAAD0A	ABDUMP 2	ABDUMP 4	In all cases
ABDUMP 4 IEAAAD01	ABDUMP 3	ABDUMP 5 ABDUMP 6	Save area trace requested No save area trace
ABDUMP 5 IEAAAD02	ABDUMP 2 ABDUMP 4	ABDUMP 6	In all cases
ABDUMP 6 IEAAAD03	ABDUMP 2 ABDUMP 4 ABDUMP5	ABDUMP 7 ABDUMP 8 ABDUMP 13 TCAM ABDUMP 1	Trace table requested Trace table not requested GTF Trace Table to be formatted TCAM control program being dumped
ABDUMP 7 IEAAAD0B	ABDUMP 6	ABDUMP 8	In all cases
ABDUMP 8 IEAAAD04	ABDUMP 6 ABDUMP 7	ABDUMP 9	In all cases
ABDUMP 9 IEAAAD05	ABDUMP 8	ABDUMP 10 ABDUMP 11	Problem program storage requested Problem program storage not requested
ABDUMP 10 IEAAAD0E	ABDUMP 9	ABDUMP 11	In all cases
ABDUMP 11 IEAAAD0C	ABDUMP 9 ABDUMP 10	ABDUMP 12	In all cases
ABDUMP 12 IEAAAD0F	ABDUMP 11	RETURN TO CALLER	In all cases
TCAM ABDUMP 1 IEAAAD0G	ABDUMP 6	TCAM ABDUMP 2	In all cases
TCAM ABDUMP 2 IEAAAD0H	TCAM ABDUMP 1	TCAM ABDUMP 3	In all cases
TCAM ABDUMP 3 IEAAAD0I	TCAM ABDUMP 2	TCAM ABDUMP 4	In all cases
TCAM ABDUMP 4 IEAAAD0J	TCAM ABDUMP 3	ABDUMP 7 ABDUMP 8 ABDUMP 13	Trace table requested Trace table not requested GTF Trace Table to be formatted
ABDUMP 13 IEAAAD0K	ABDUMP 6 ABDUMP 14 TCAM ABDUMP 4	ABDUMP 8 ABDUMP 14	GTF trace table formatting completed Trace error or control record encountered
ABDUMP 14 IEAAAD0M	ABDUMP 13	ABDUMP 13	Trace error or control record processed

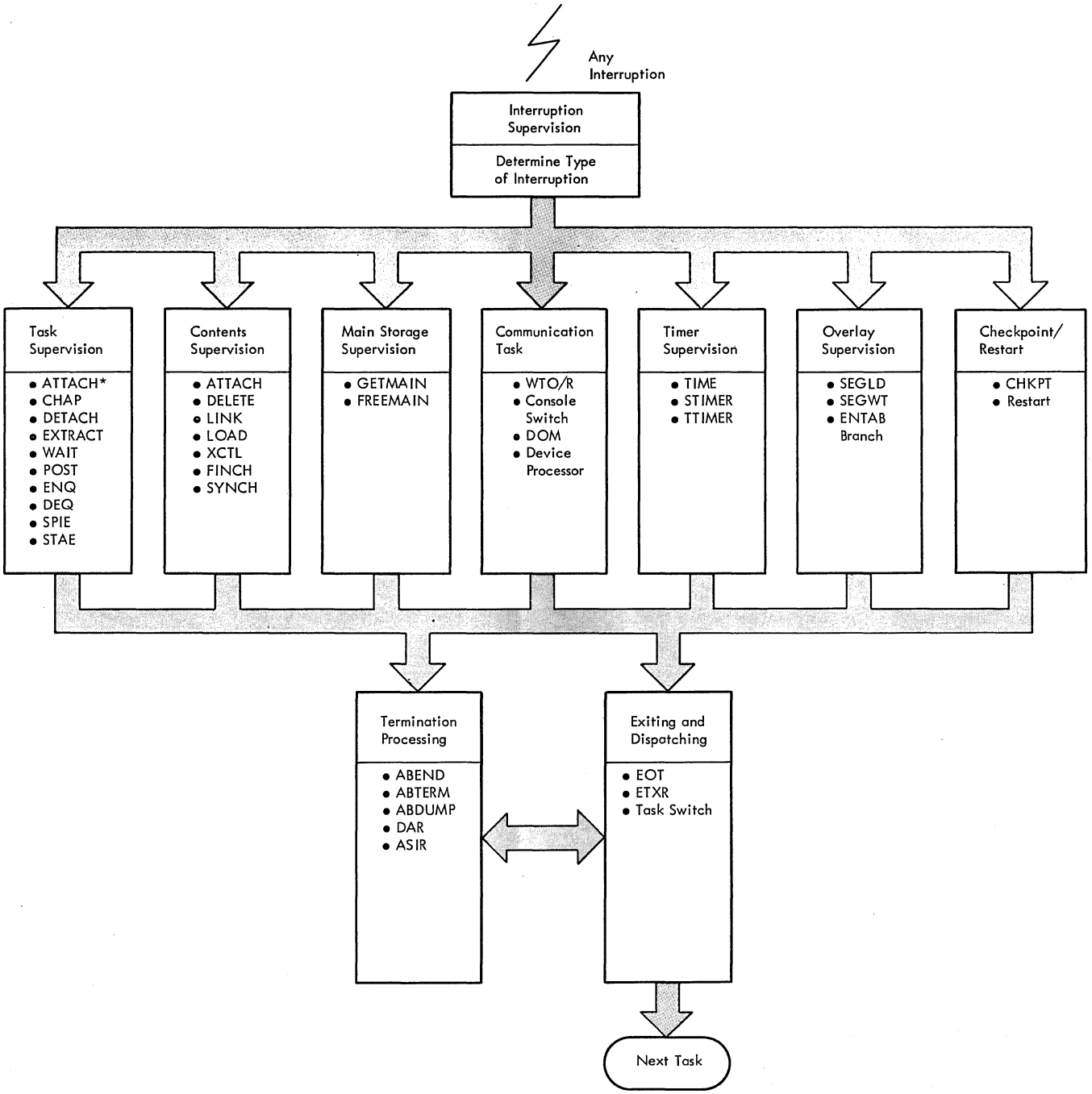


Diagram 01. The MFT Supervisor

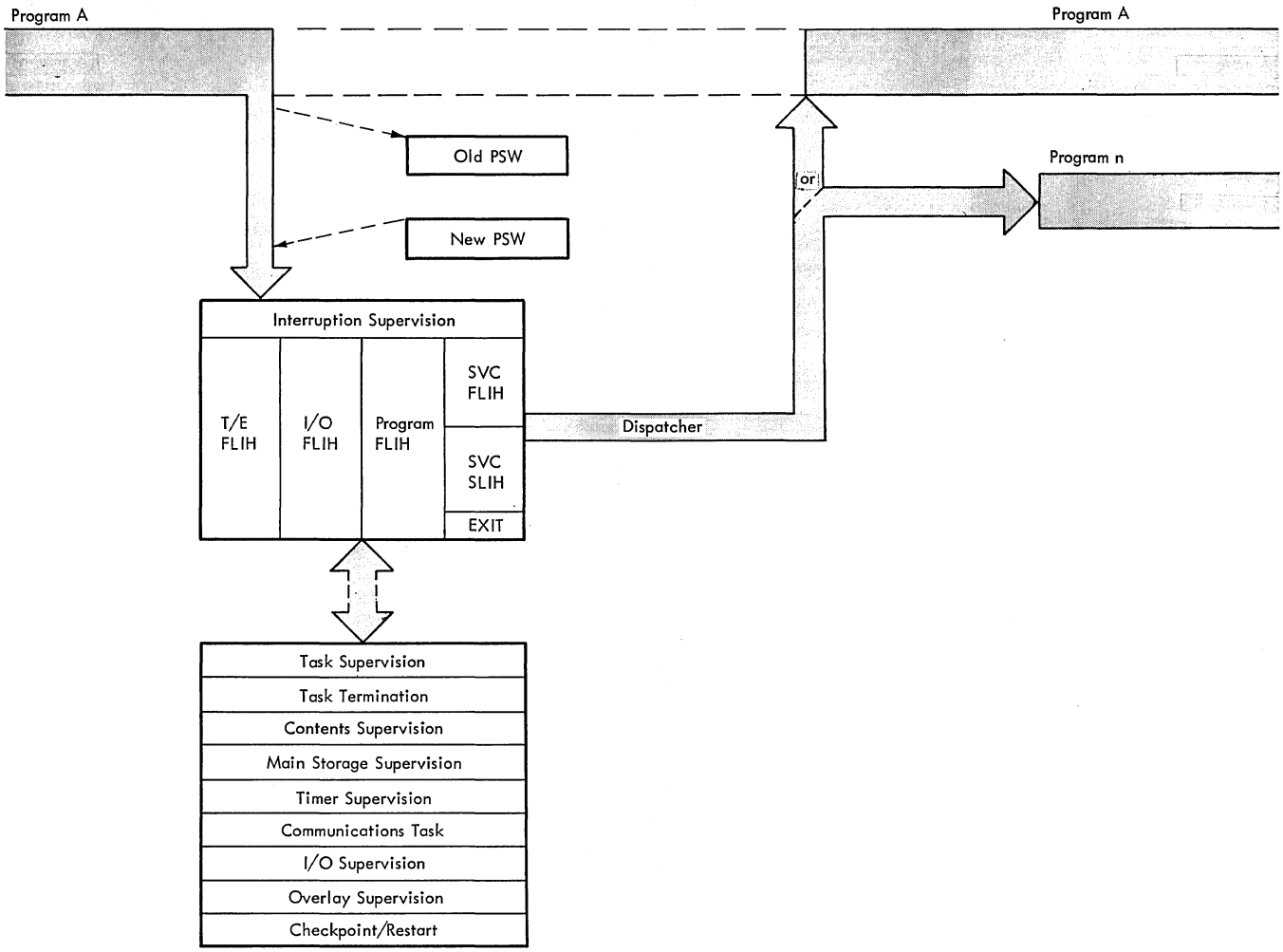


Diagram 02. Interruption Supervision

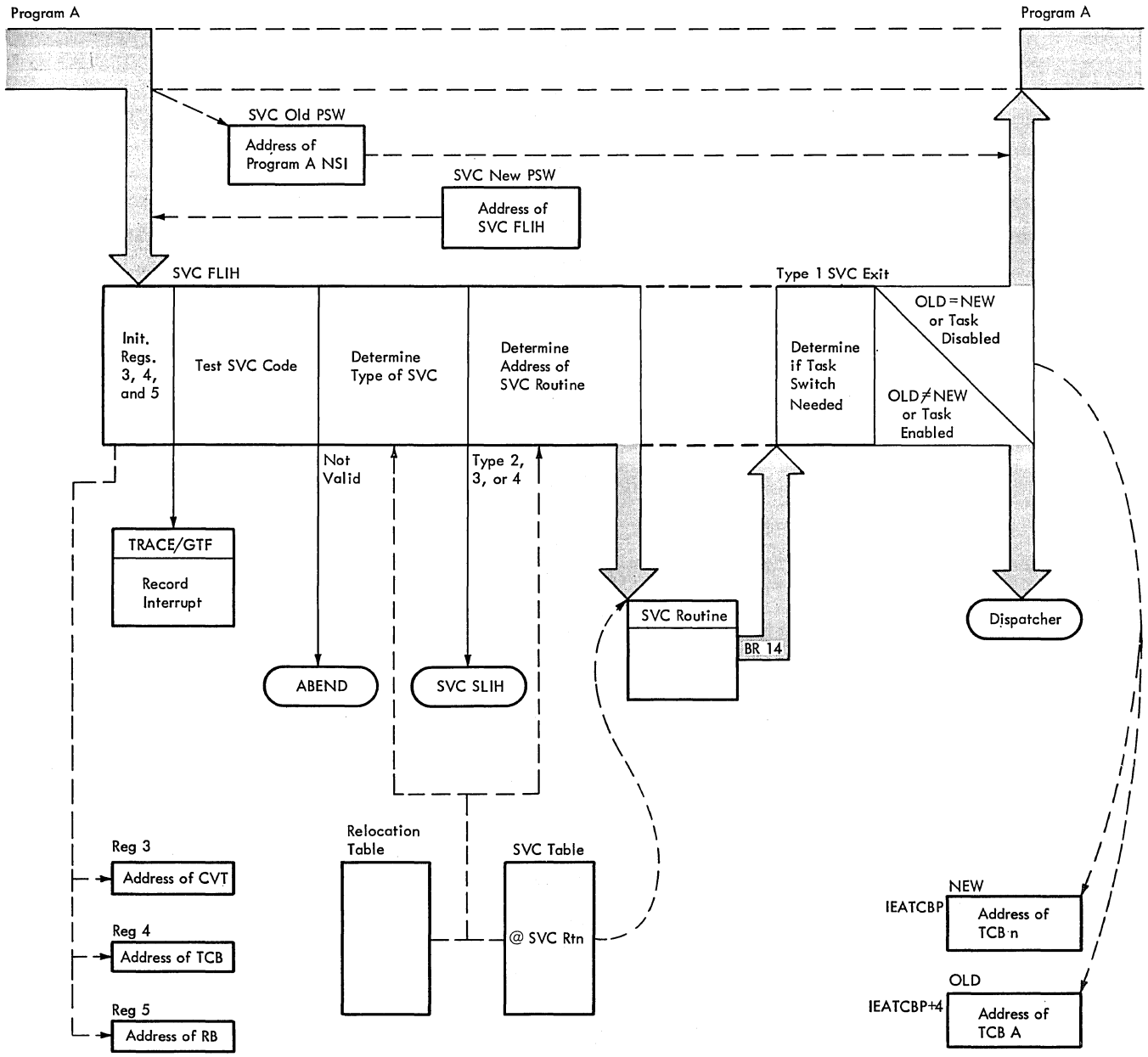


Diagram 03. SVC FLIH and Type-1 Exit

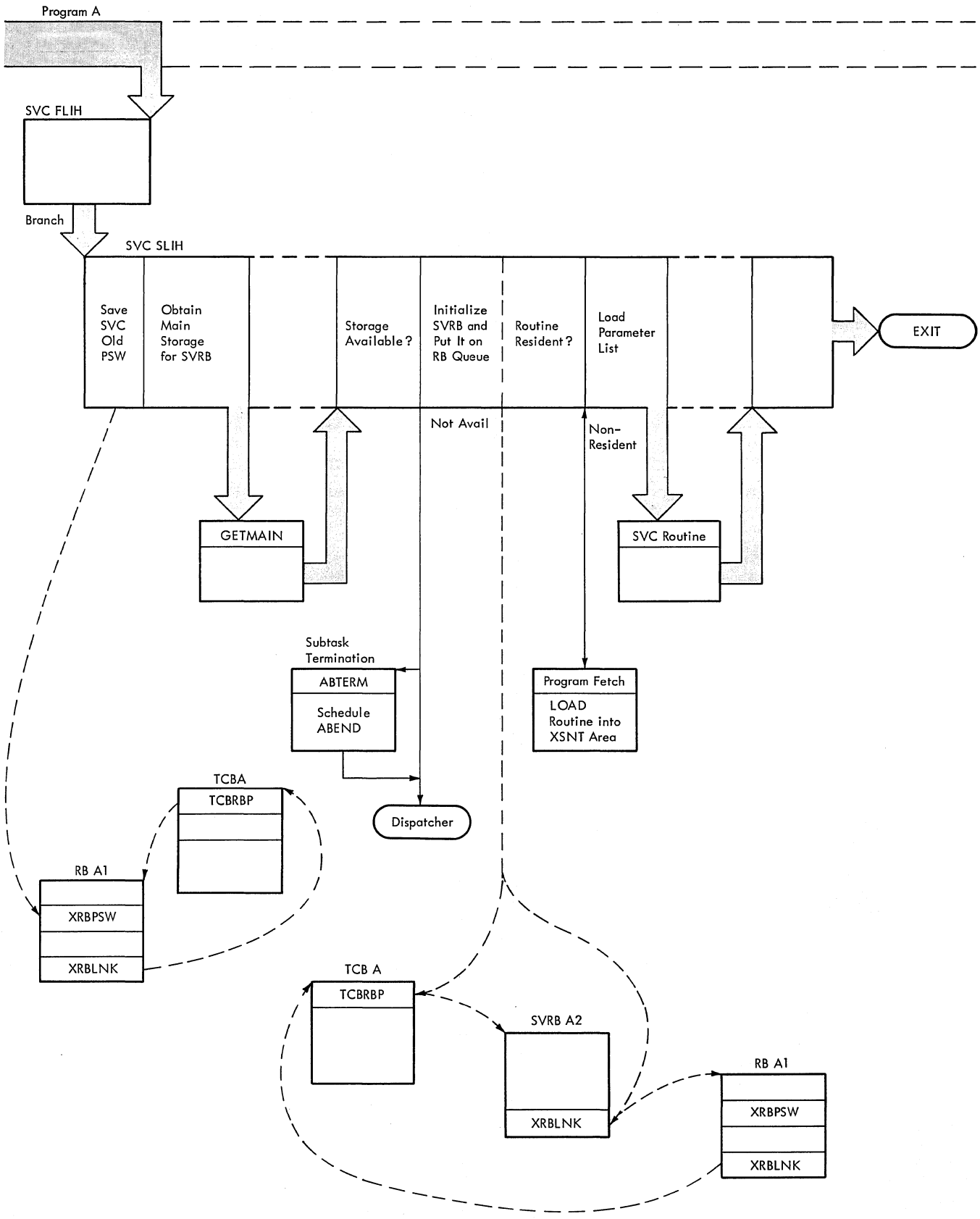


Diagram 04. SVC SLIH

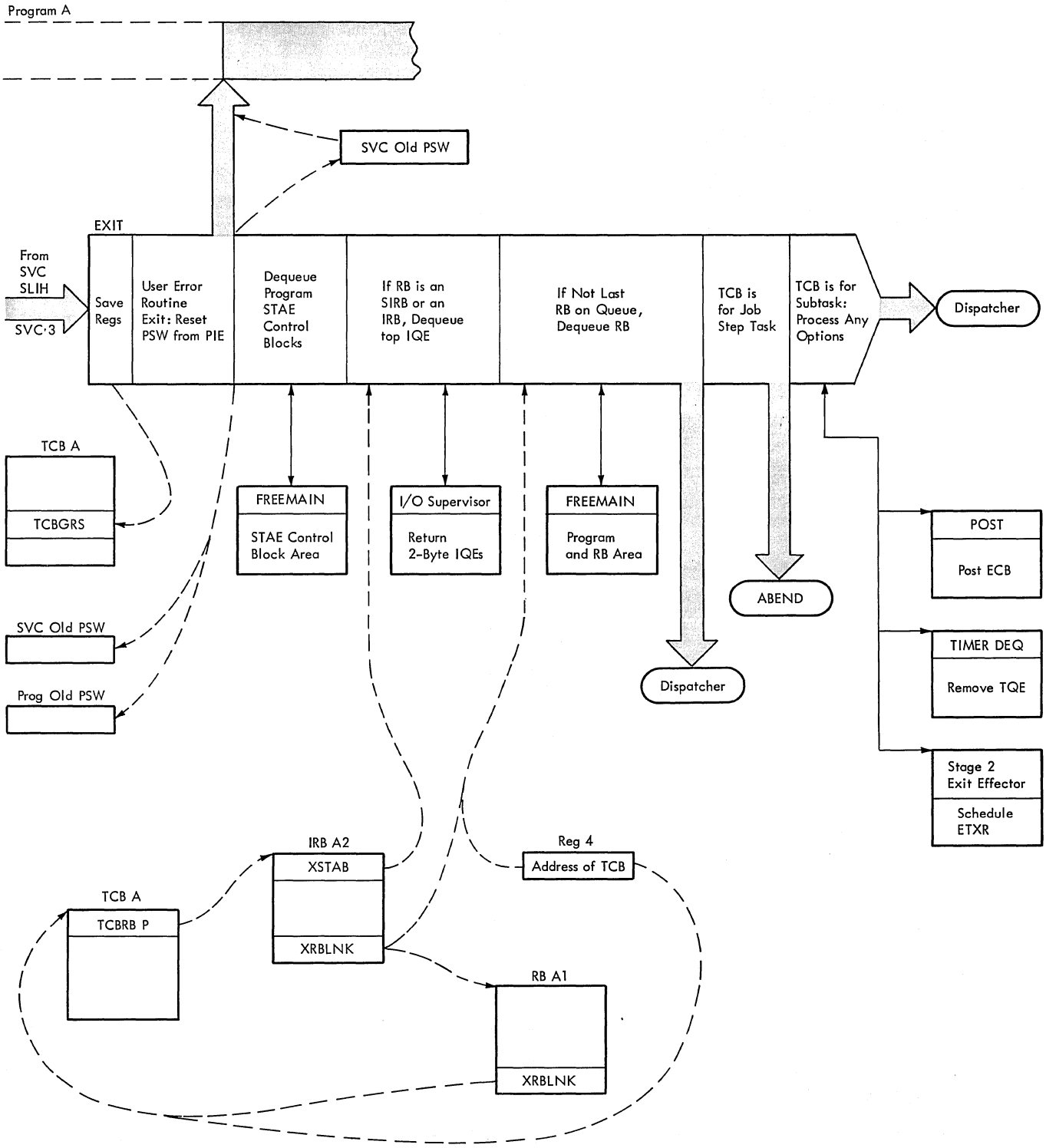


Diagram 05. EXIT

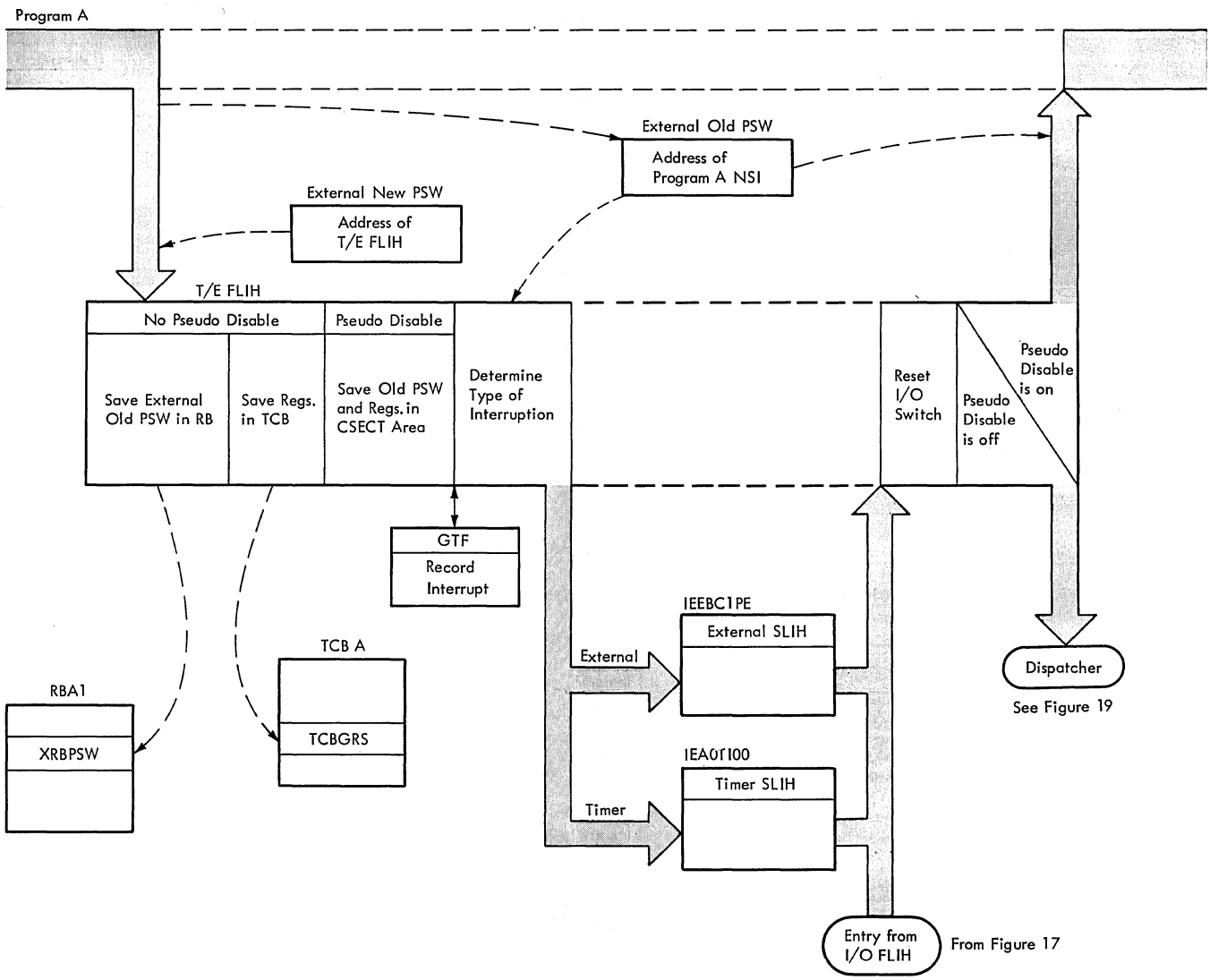


Diagram 06. Timer/External FLIH

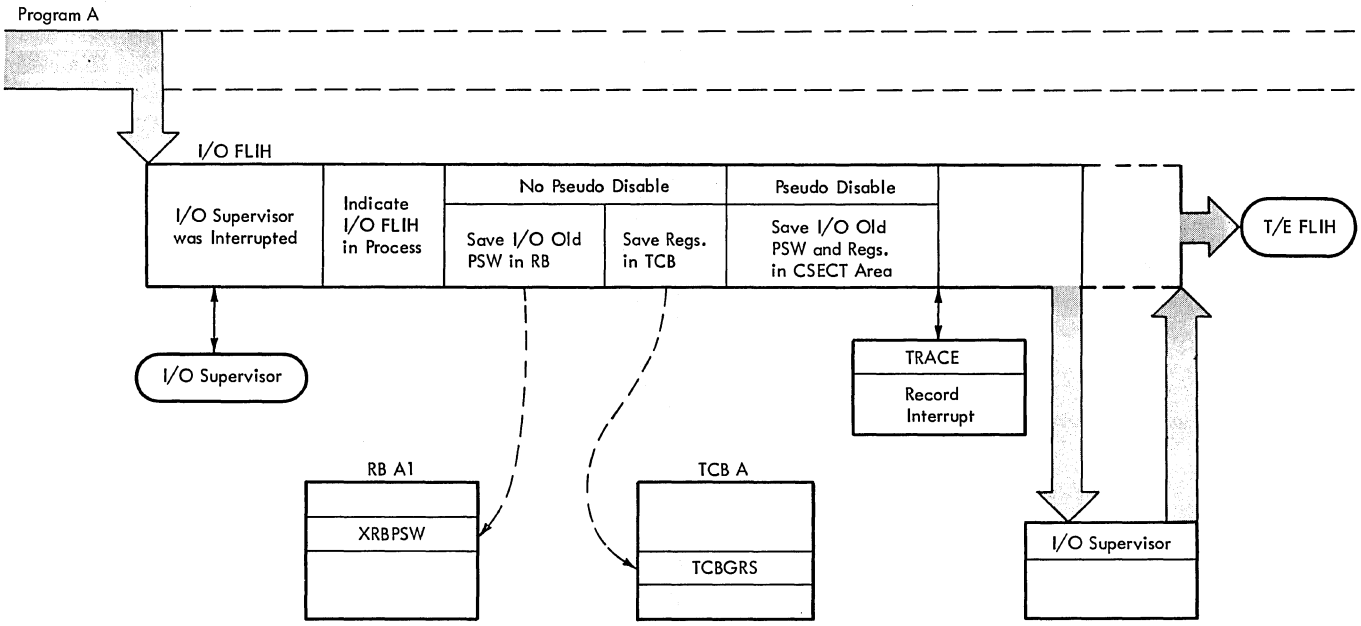


Diagram 07. Input/Output FLIH

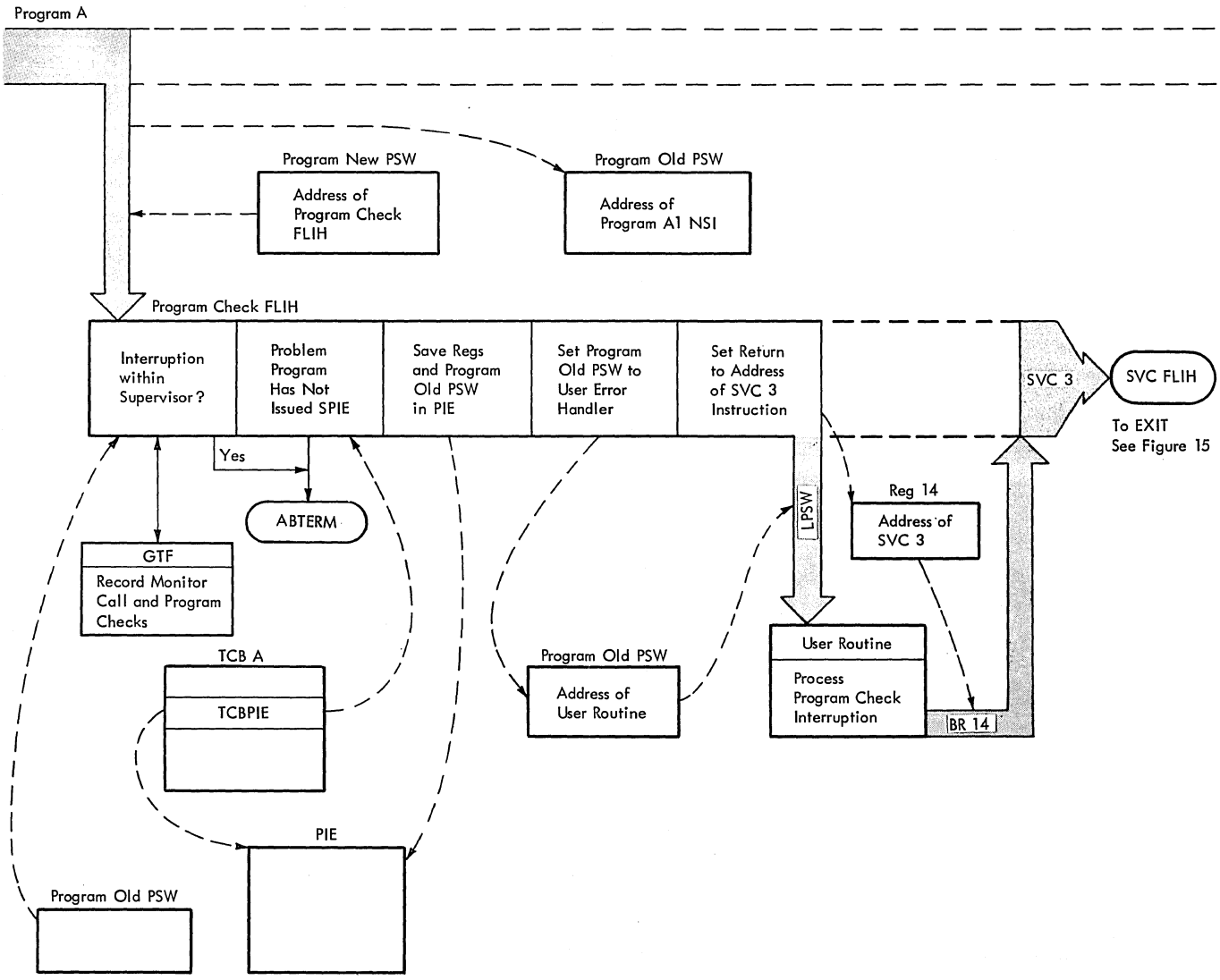


Diagram 08. Program Check FLIH

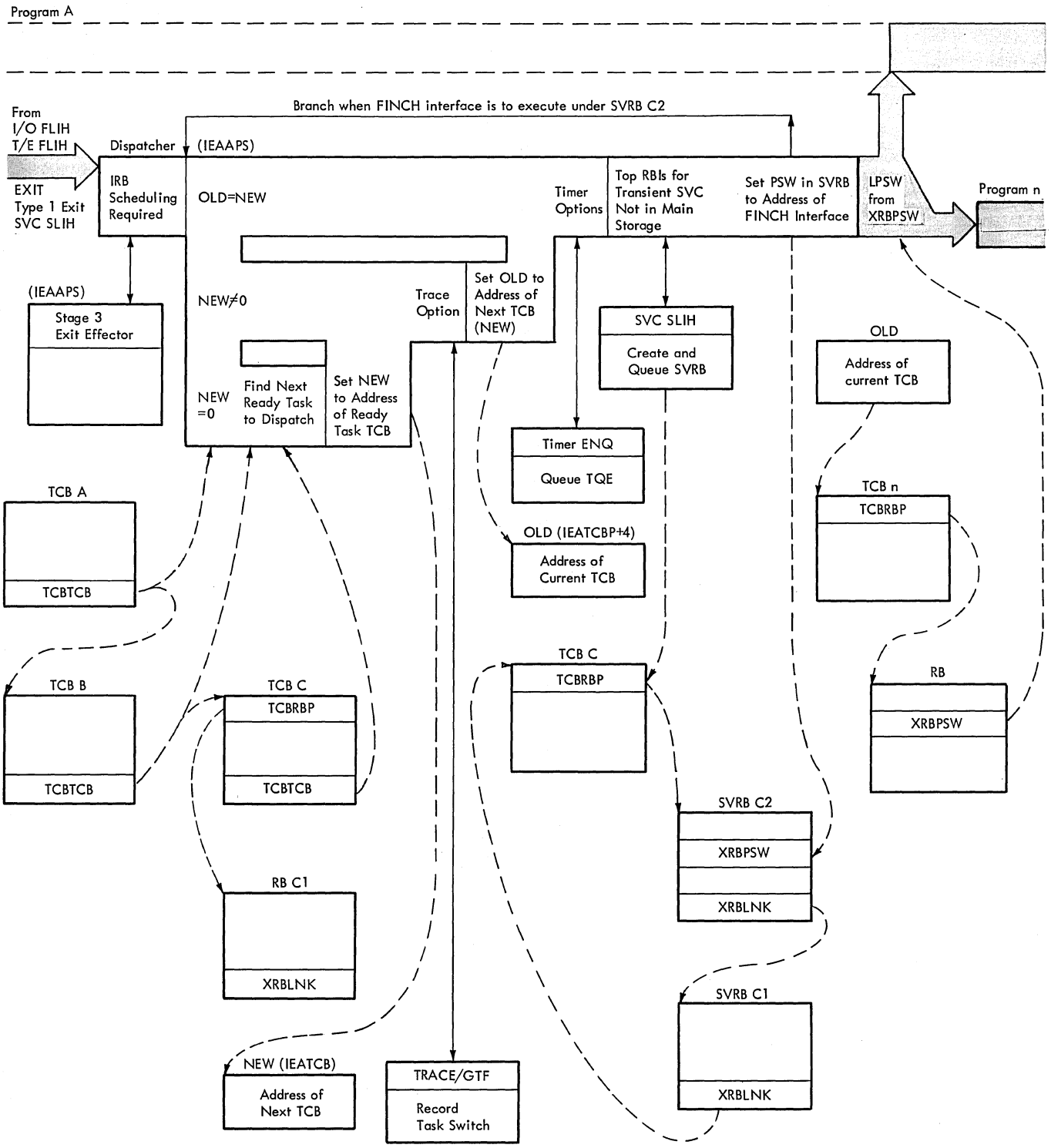


Diagram 09. Dispatcher

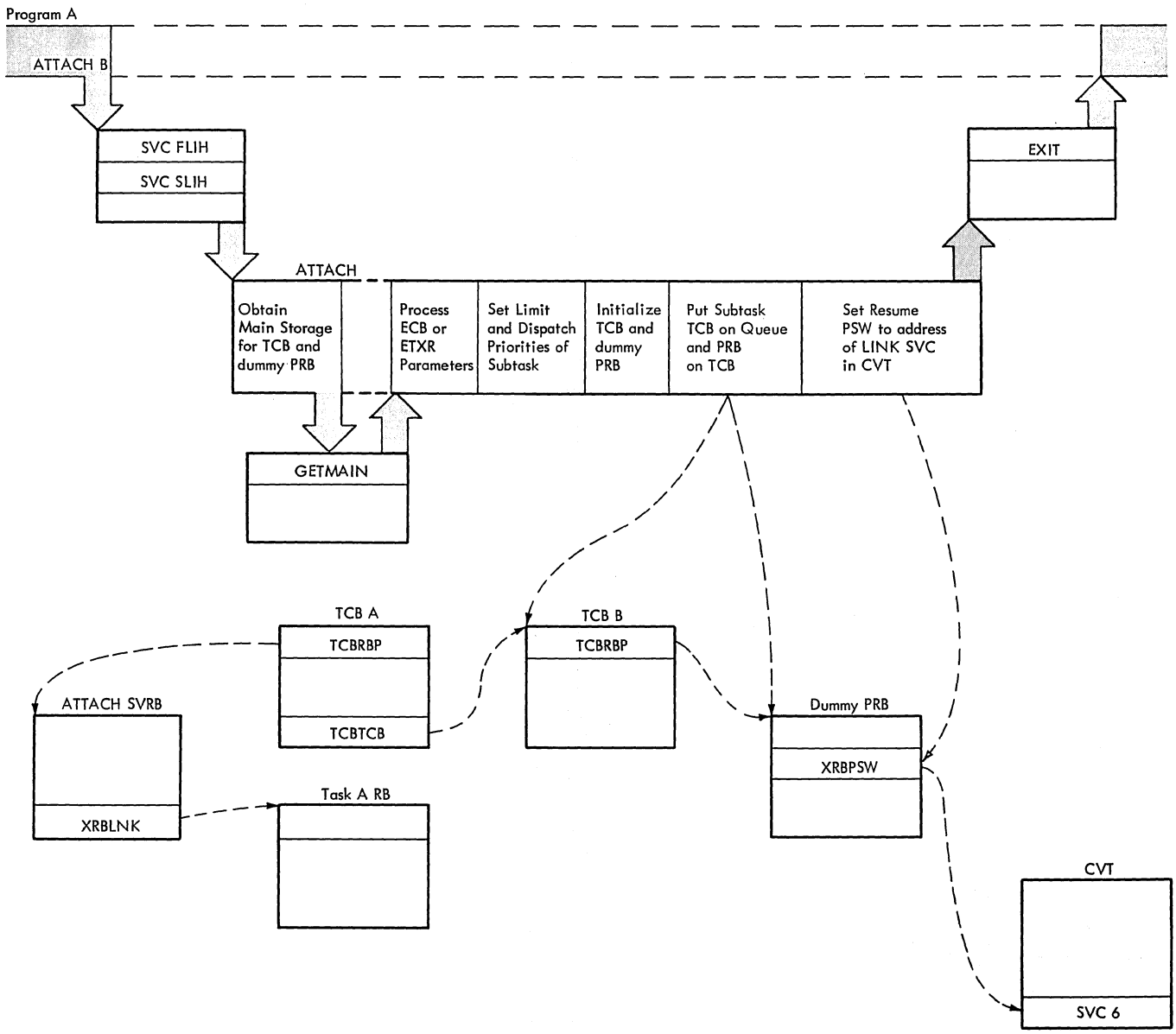


Diagram 10. Attaching a Subtask (Part 1)

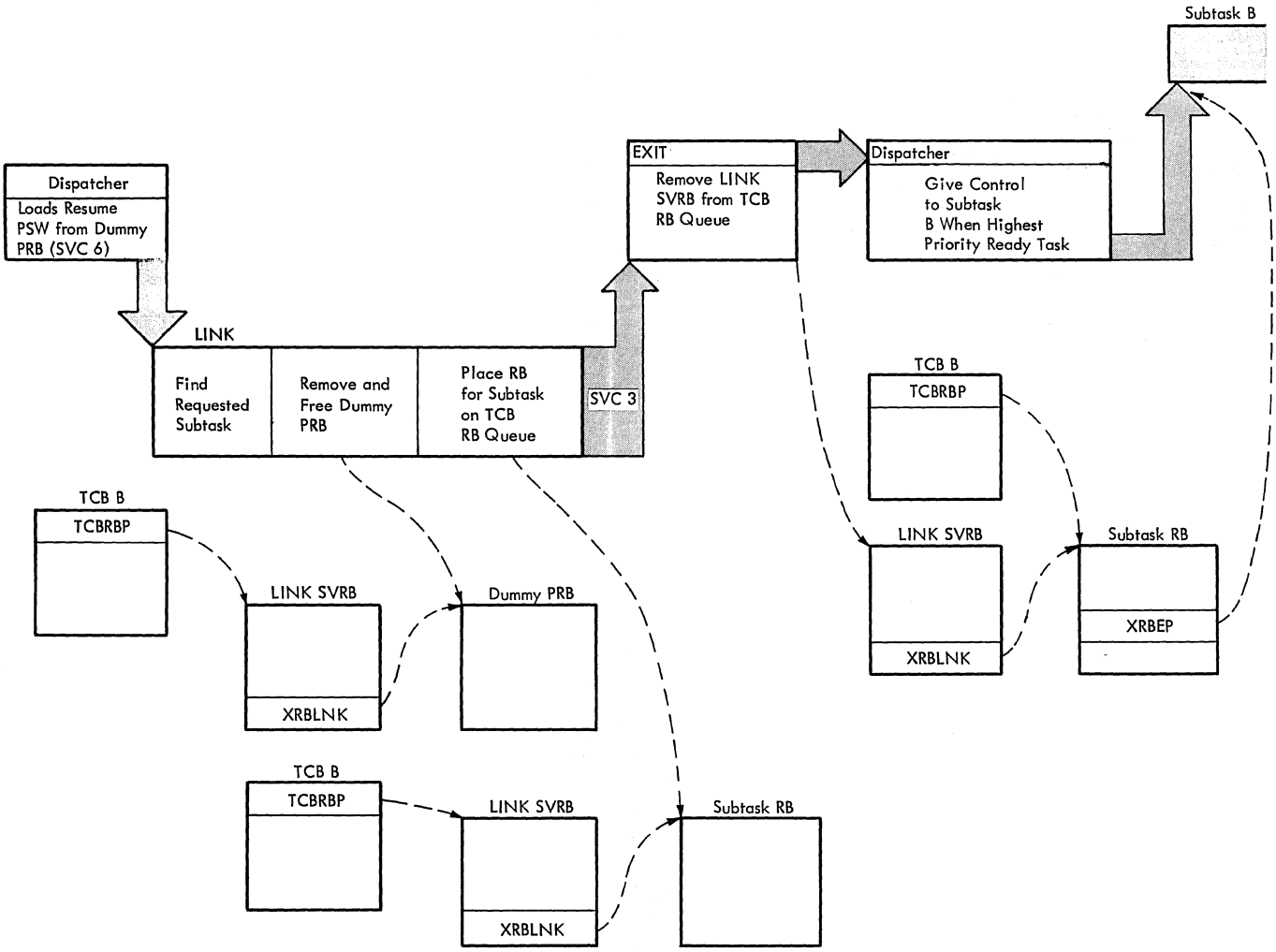


Diagram 11. Attaching a Subtask (Part 2)

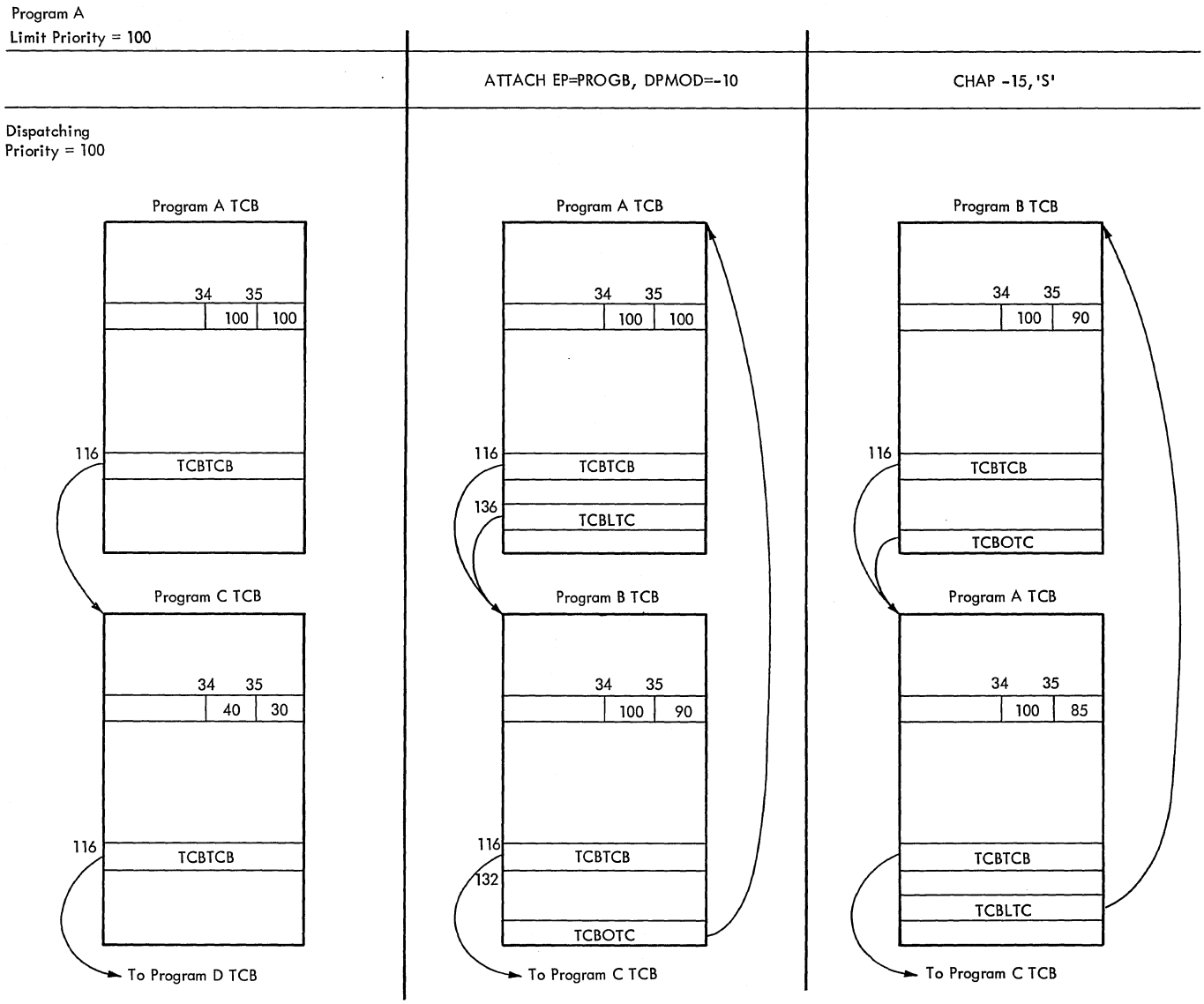


Diagram 12. Creating and Changing Dispatching Priorities

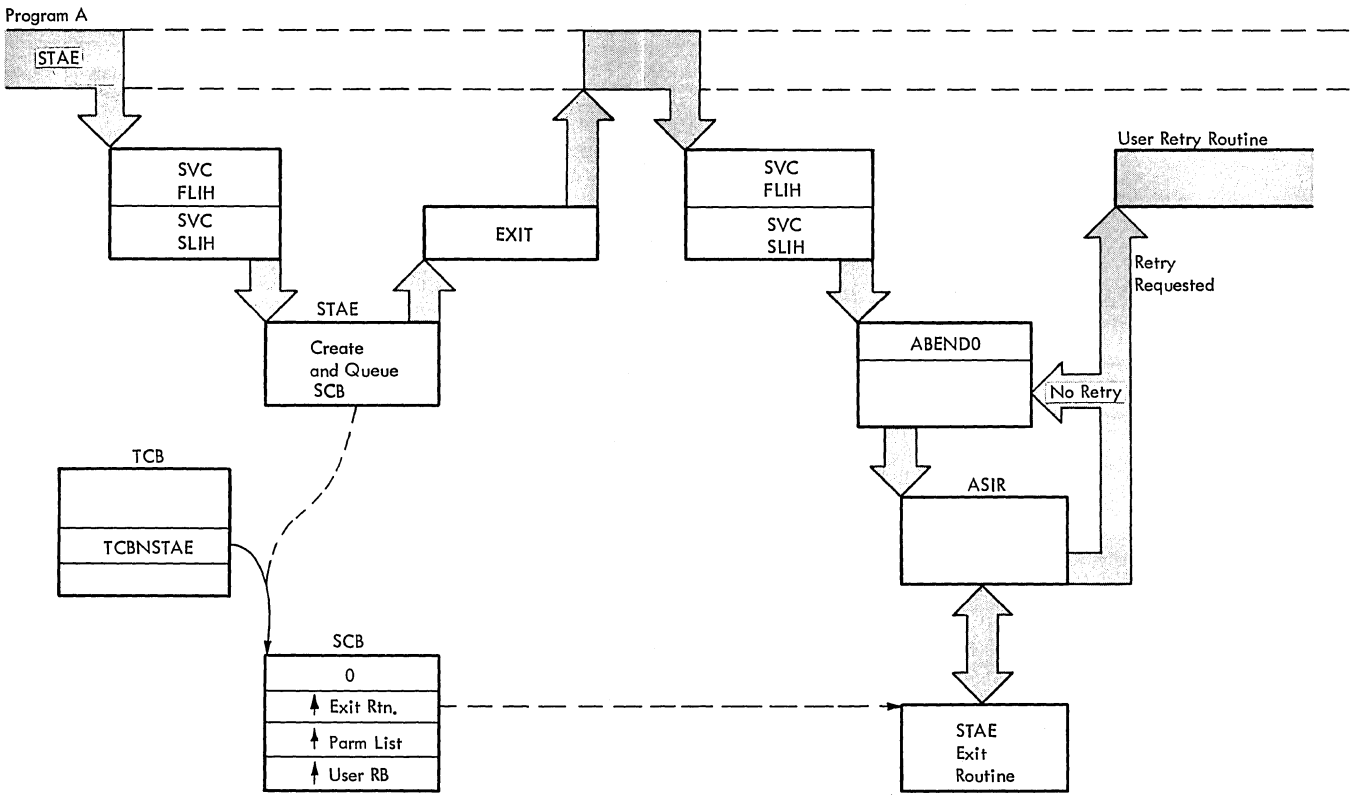


Diagram 13. Processing for a Task Asynchronous Exit

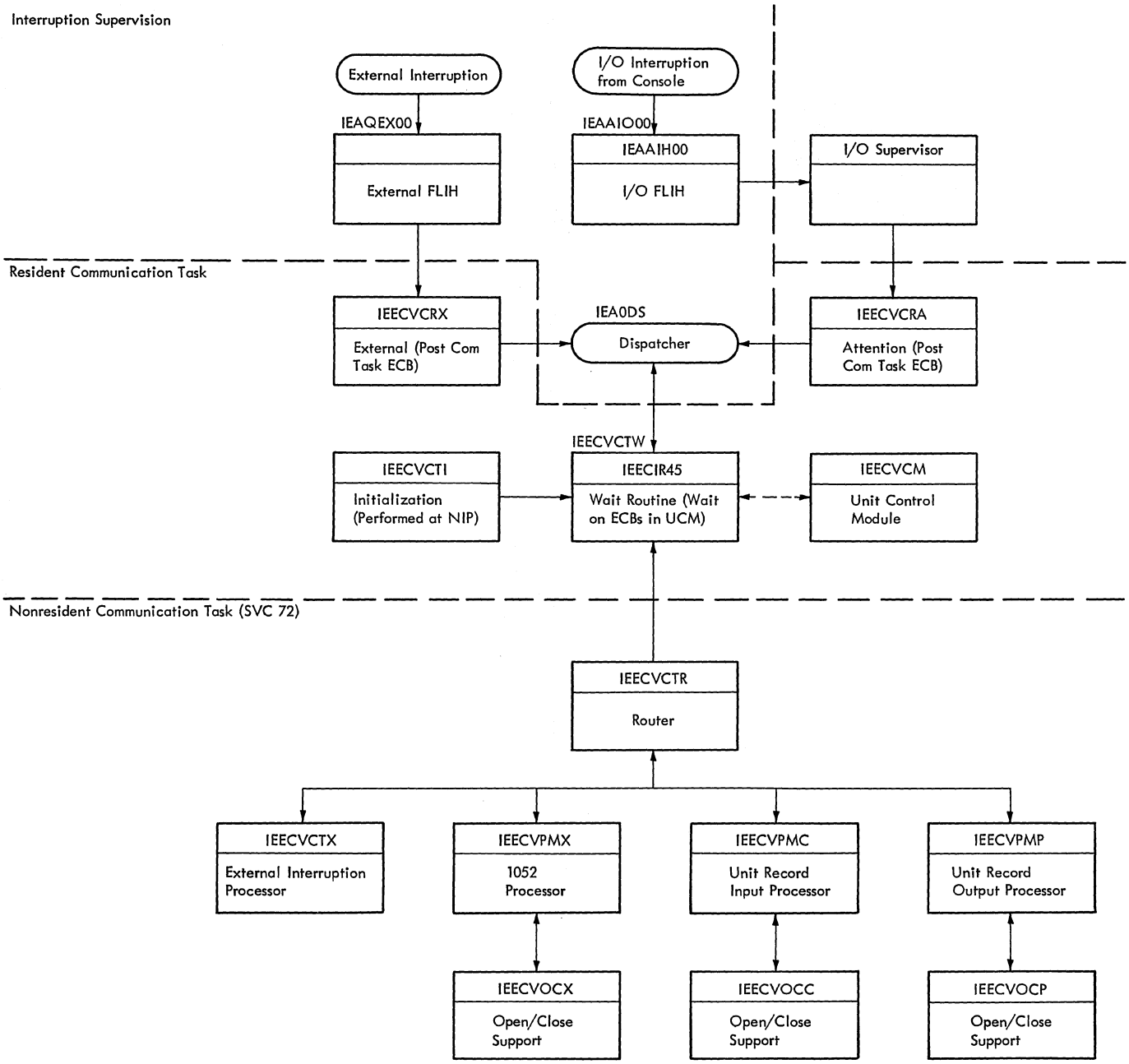


Diagram 14. Console Communications without MCS: Input

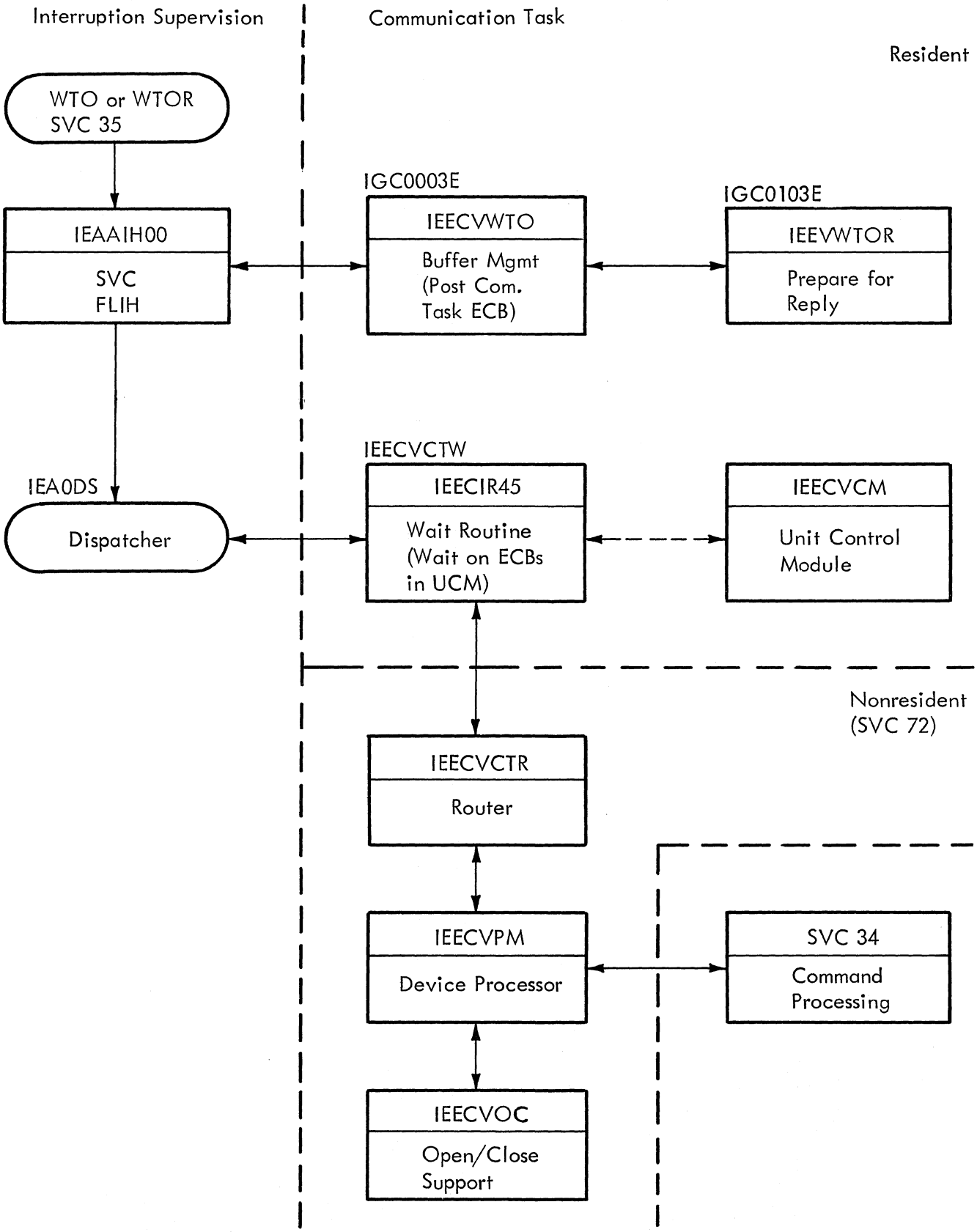


Diagram 15. Console Communications without MCS: Output

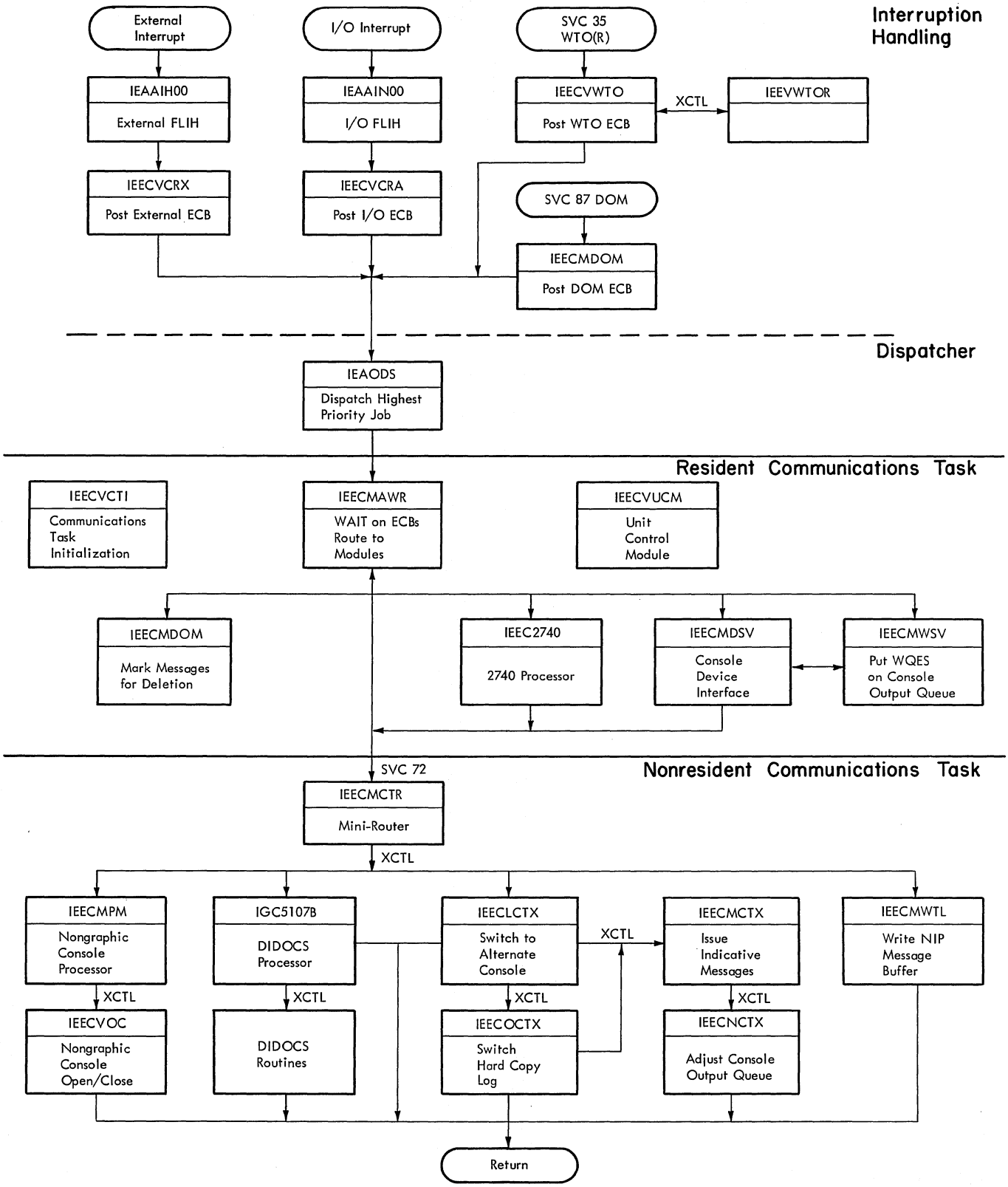


Diagram 16. Communications Task with Multiple Console Support

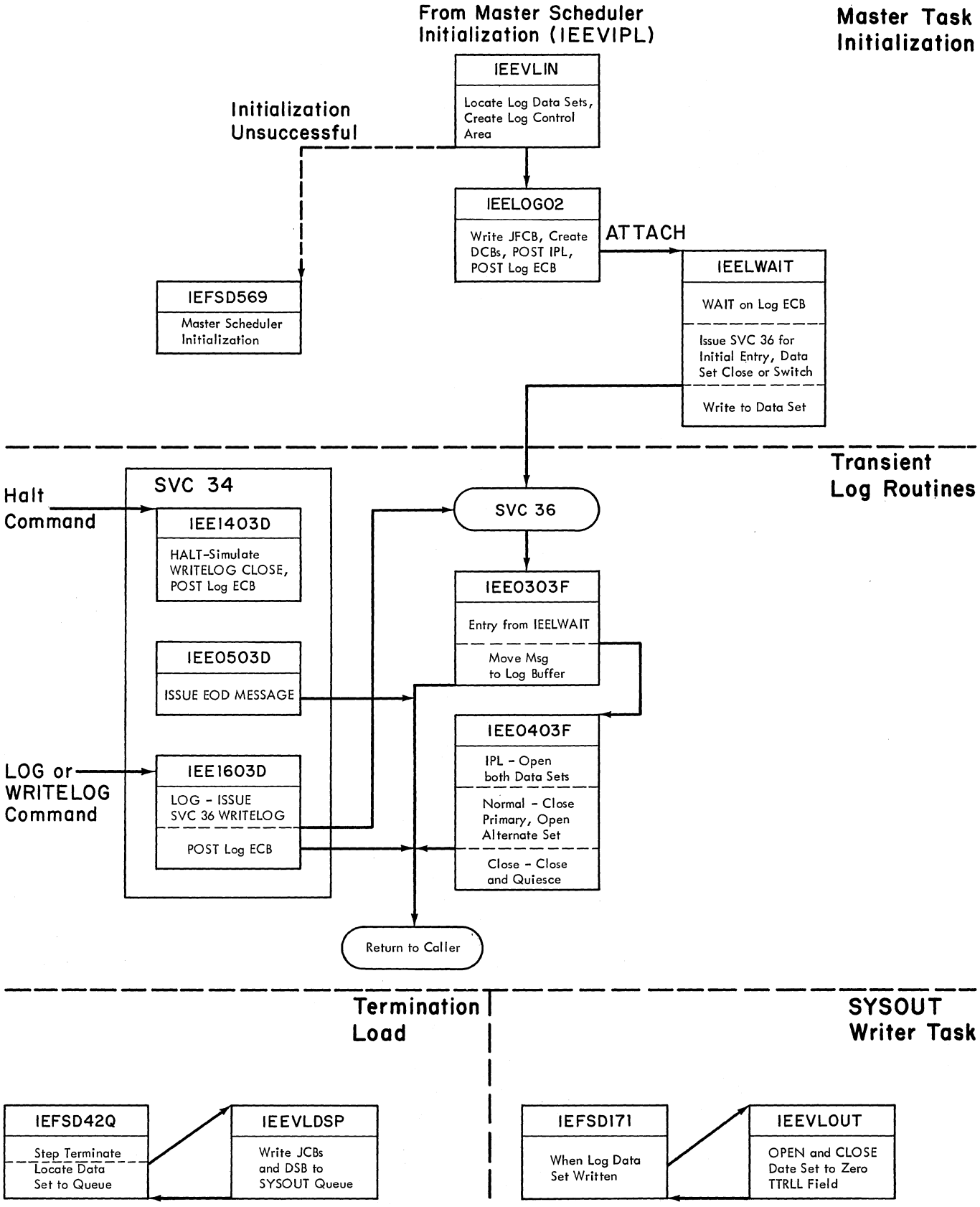


Diagram 17. Log Function

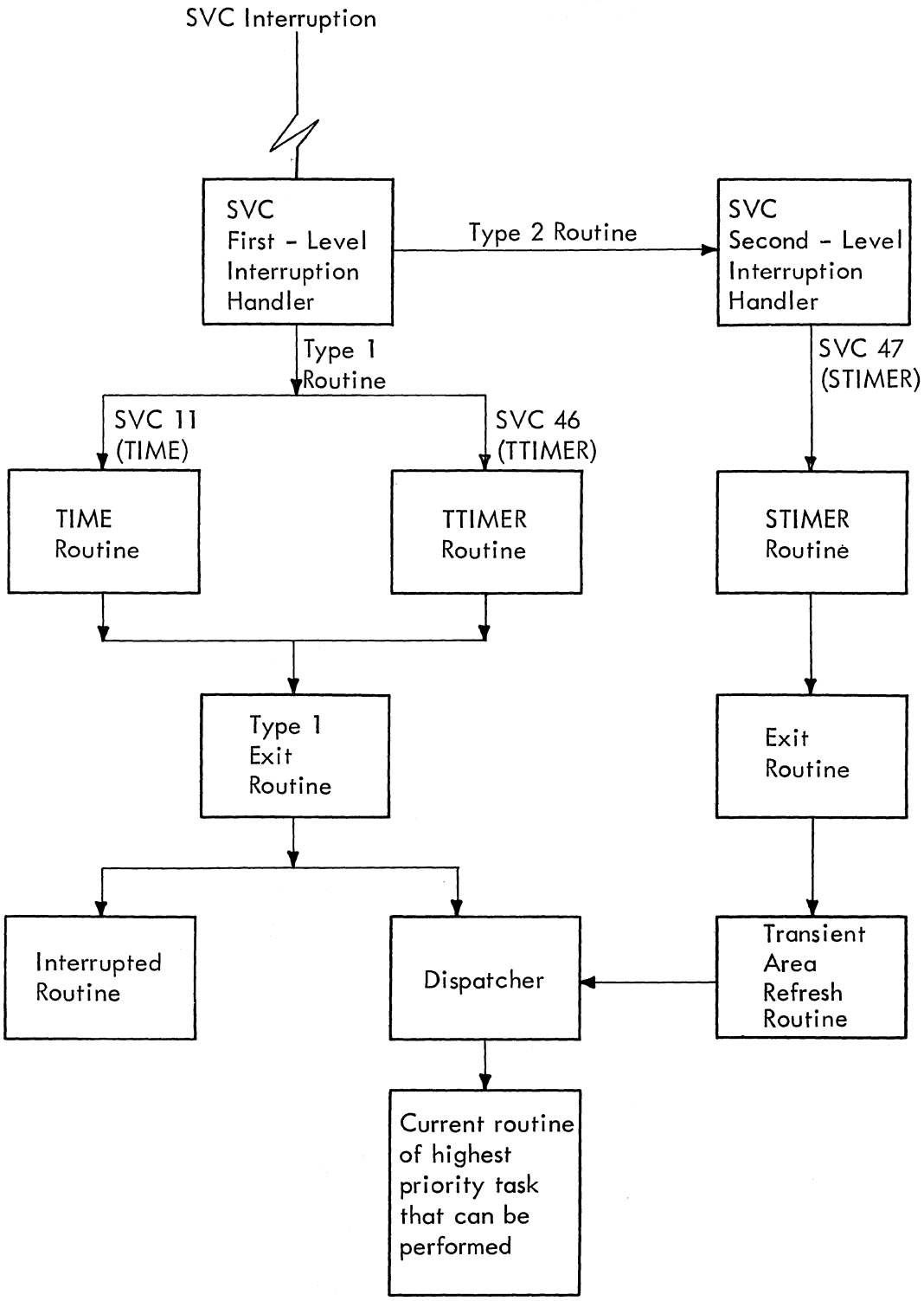


Diagram 18. Timer SVC Interruption Handling

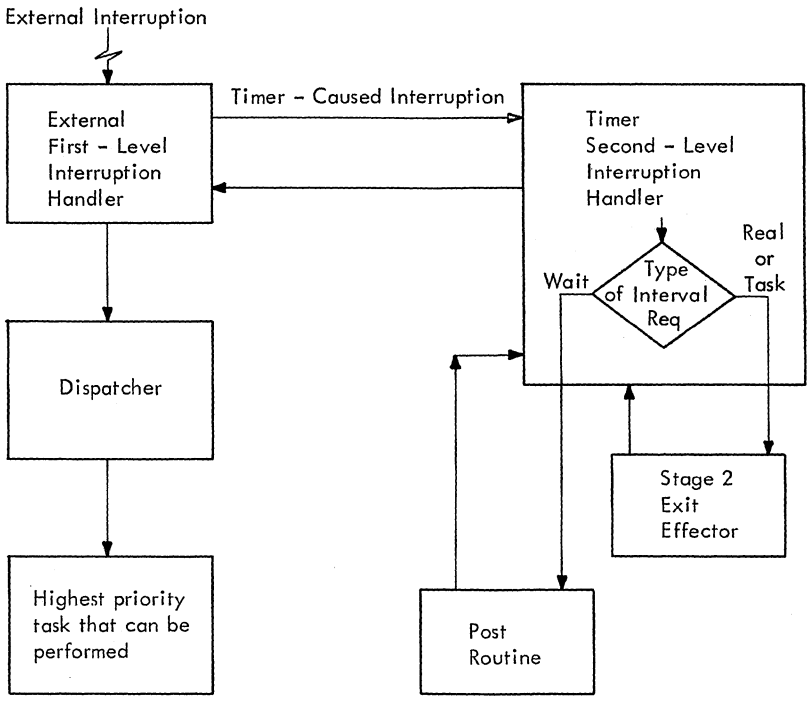


Diagram 19. Timer Interruption Handling

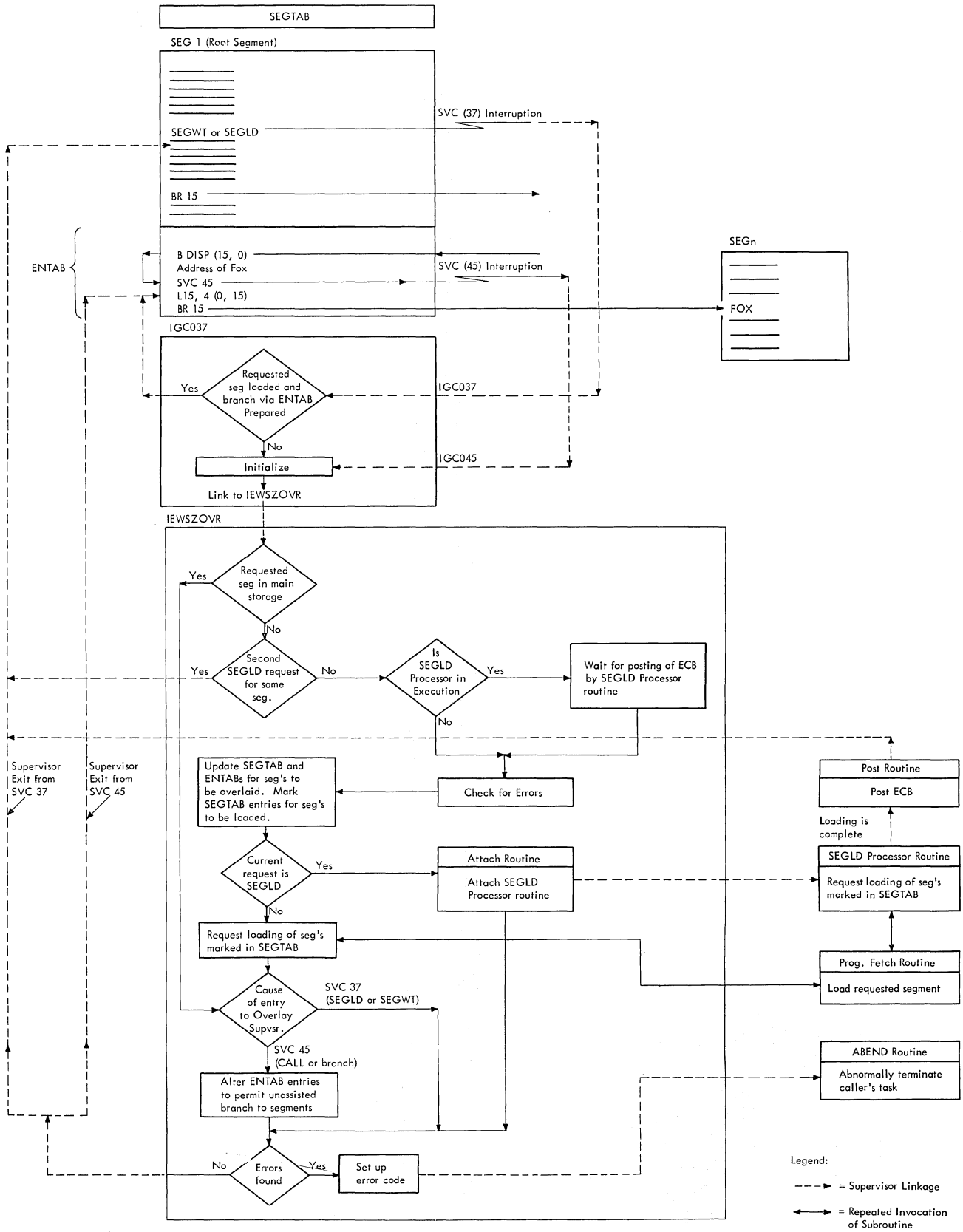


Diagram 20. Functional Flow of Overlay Supervision

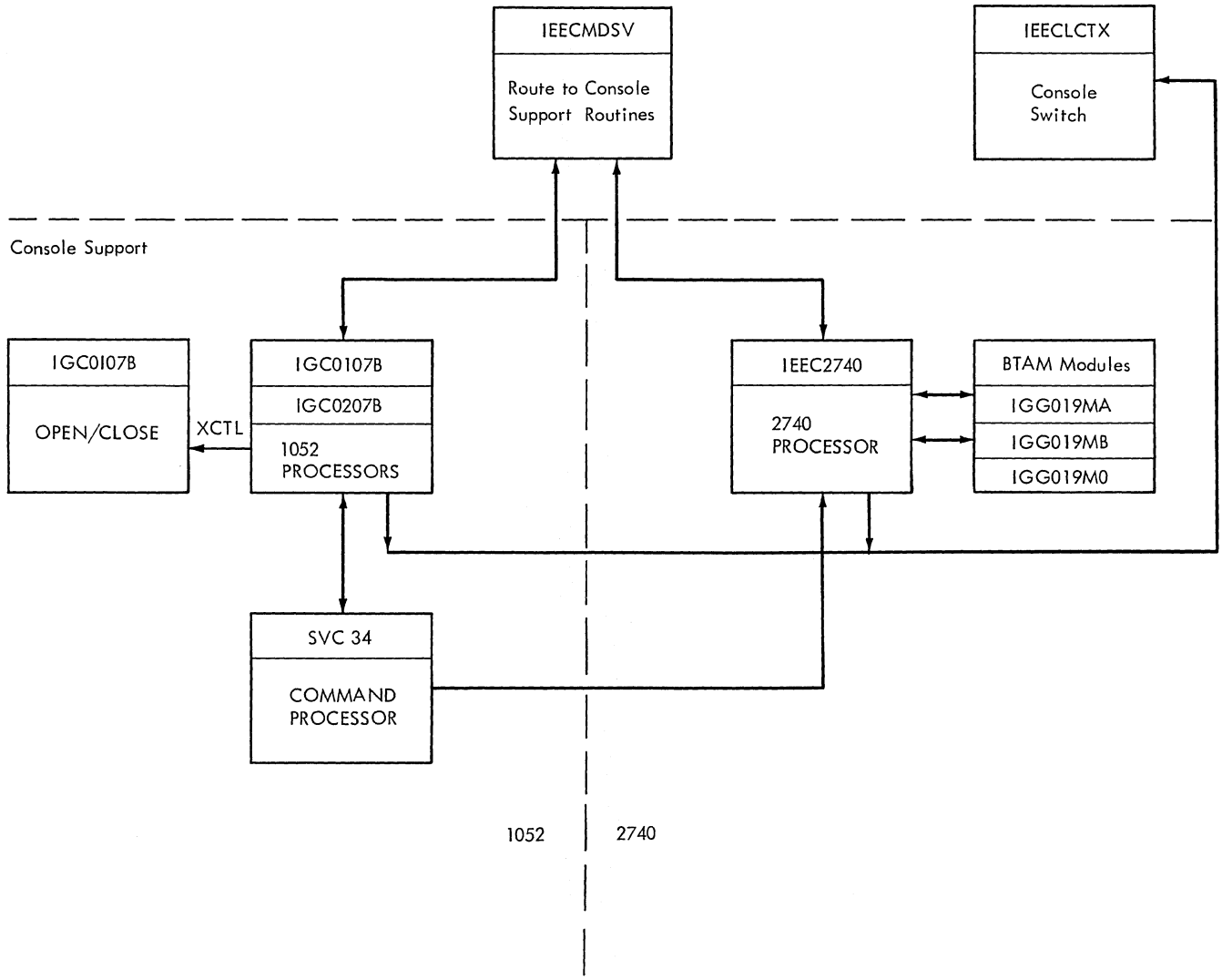


Diagram 21. 1052 and 2740 Console Support Routines with MCS

Where more than one page reference is given, the major reference is first.

ABEND
  entry/exit table 458
  functional description 30
  recursion processing 31
ABEND 0
  entry point 375
  flowchart 202
  module description 90
  module synopsis 151
ABEND 1
  entry point 375
  flowchart 205
  module description 91
  module synopsis 151
ABEND 2
  entry point 375
  flowchart 206
  module description 91
  module synopsis 151
ABEND 3
  entry point 375
  flowchart 206
  module description 91
  module synopsis 151
ABEND 4
  entry point 375
  flowchart 207
  module description 92
  module synopsis 151
ABEND 5
  entry point 376
  flowchart 208
  module description 92
  module synopsis 151
ABEND 6
  entry point 376
  flowchart 209
  module description 92
  module synopsis 151
ABEND 7
  entry point 376
  flowchart 210
  module description 93
  module synopsis 152
ABEND 8
  entry point 376
  flowchart 210
  module description 93
  module synopsis 152
ABEND 9
  entry point 376
  flowchart 211
  module description 93
  module synopsis 152
ABEND 10
  entry point 373
  flowchart 212
  module description 94
  module synopsis 152
ABEND 11
  entry point 373
  flowchart 213
  module description 94
  module synopsis 152
ABEND 12
  entry point 373
  flowchart 213
  module description 94
  module synopsis 152
ABEND 13
  entry point 373
  flowchart 213
  module description 94
  module synopsis 152
ABEND 14
  entry point 373
  flowchart 214
  module description 94
  module synopsis 153
ABEND 15
  entry point 373
  flowchart 215
  module description 94
  module synopsis 153
ABEND G
  entry point 374
  flowchart 203
  module description 91
  module synopsis 153
ABEND H
  entry point 374
  flowchart 217
  module description 95
  module synopsis 153
ABEND J
  entry point 374
  flowchart 218
  module description 95
  module synopsis 153
ABEND M
  entry point 374
  flowchart 216
  module description 95
  module synopsis 153
ABEND/STAE Interface Routines (see also ASIR 1, ASIR 2, etc.)
  functional description 32
ABDUMP
  entry point 373-376
  entry/exit table 461
  flowchart 231-241
  functional description 32
  module descriptions 98-101
  module synopsis 155-157
  parameter list format 377
ABTERM
  entry point 369
  flowchart 227
  functional description 30
  module description 95

module synopsis 154
Alternate Path Retry 60
ASIR 1
entry point 375
flowchart 219
functional description 27
module description 96
module synopsis 153
ASIR 2
entry point 375
flowchart 220
functional description 27
module description 96
module synopsis 154
ASIR 3
entry point 375
flowchart 221
functional description 27
module description 96
module synopsis 154
ASIR 4
entry point 376
flowchart 222
functional description 27
module description 96
module synopsis 154
ATTACH with Subtasking
entry point 369,373
flowchart 185
functional description 17
module description 78
module synopsis 149
ATTACH without Subtasking
entry point 369,373
flowchart 242
functional description 33,36
module description 102
module synopsis 158

BLDL 105
table 6
boundary box 39,78
format of 378
transient area queue 37,108
BXLE instruction in FINCH 105

CANCEL parameter in TTIMER 52
channel-check handler 60,16
CHAP
entry point 373
flowchart 188
functional description 20
module description 79
module synopsis 149
checkpoint
Check I/O
entry point 374
module description 126
Checkmain
entry point 374
module description 126
Exit
entry point 374
module description 126
flowchart 360
functional description 62
Housekeeping 1
entry point 375
module description 125
Housekeeping 2
entry point 375
module description 125
Housekeeping 3
entry point 375
module description 125
Message
entry point 374
module description 126
Preserve 1
entry point 373
module description 126
Preserve 2
entry point 373
module description 126
Resume I/O
entry point 374
module description 126
checkpoint header record 379
code
command authority 42,116
completion 30,83,90
routing 42,116
command scheduling control block (CSCB) 39
communications task
(see also DIDOCS, device support modules)
attention interruption handler
entry point 370
flowchart 256
functional description 41
module description 115
module synopsis 159
console switch
entry point 372
flowchart 265
functional description 41
module description 115
module synopsis 160
external interruption handler
entry point 370
flowchart 256
functional description 41
module description 115
module synopsis 159
initialization module
entry point 270
flowchart 255
functional description 41
module description 113
module synopsis 159
Log function 42
Log writer flowchart 288
LOG, Write-to-Log flowchart 289,290
router module
entry point 375
flowchart 264
functional description 41
module description 115
module synopsis 159
task control block 41,7,42
unit control module
format of 422
wait module
entry point 370
flowchart 263

functional description 41
module description 115
module synopsis 159
communications task with Multiple Console Support
console switch routines
entry point 372
flowchart 267-270
functional description 42
module description 116
module synopsis 162
parameter list 117
delete operator message (DOM) routine
entry point 370
flowchart 277
functional description 42
module description 118
module synopsis 163
device interface routine
entry point 370
flowchart 271
functional description 42
module description 117
module synopsis 162
event indication list 41,42
extended save area 64
graphic console initialization
entry point 371
flowchart 293
module description 113
module synopsis 160
mini-router
entry point 375
functional description 42
module description 116
module synopsis 162
NIP message buffer writer
entry point 376
flowchart 278
functional description 42
module description 118
module synopsis 163
router
entry point 370
flowchart 266
functional description 42
module description 116
module synopsis 162
WTO/R processor module
entry point 370
flowchart 274
functional description 42
module description 118
module synopsis 162
communications vector table
format 380
use 18,46,68
completion code
ABEND and 30,90
POST and 83
console
alternate 116
device support modules (see device support modules)
output queue 42,98
switch modules (see communications task)
core image record 379
damage assessment
modules (see DAR 1, DAR 2, etc.)
recursion 32,97
DAR 1
entry point 375
flowchart 223
functional description 32
module description 96
module synopsis 154
DAR 2
entry point 375
flowchart 224
functional description 32
module description 97
module synopsis 154
DAR 3
entry point 376
flowchart 225
functional description 32
module description 97
module synopsis 154
DAR 4
entry point 376
flowchart 226
functional description 32
module description 97
module synopsis 155
data set descriptor record 384
date and time
option 45
request for 40,121
DD name
SYSABEND 31,91
SYSUDUMP 31,91
DELETE
entry point 372
flowchart 251
functional description 37
module description 102
module synopsis 158
DEQ
entry point 373
flowchart 195
functional description 24
module description 85
module synopsis 150
DETACH
entry point 373
flowchart 187
functional description 20
module description 79
module synopsis 149
device independent display operator console support (see DIDOCS)
device
release 25
reserve 23
device support modules
Card Reader open/close
entry point 371,376
module description 120
module synopsis 160
Card Reader processor
entry point 371,372,376
flowchart 284,285
module description 119
module synopsis 160,163
Printer open/close

entry point 371,176
module description 119
module synopsis 161
Printer processor
entry point 371,372,376
module description 119
module synopsis 160,163
1052 open/close
entry point 371
flowchart 283
module description 119
module synopsis 160
1052 processor loads 1 and 2
entry point 372,370,375
flowchart 279-282,291,292
module description 119
module synopsis 160,163
2740 processor
entry point 372
flowchart 286
module description 120
module synopsis 164
(see also DIDOCS)
DIDOCS
Asynchronous error module
entry point 370
flowchart 340
functional description 63
module description 137
module synopsis 166
Cleanup
entry point 371
flowchart 358
functional description 64
module description 143
module synopsis 169
Command module
entry point 370
flowchart 324
functional description 63
module description 138
module synopsis 168
Delete 1
entry point 371
flowchart 327
functional description 63
module description 139
module synopsis 168
Delete 2
entry point 371
flowchart 329
functional description 63
module description 139
module synopsis 168
Delete 3
entry point 371
flowchart 330
functional description 63
module description 139
module synopsis 168
Delete 4
entry point 371
flowchart 332
functional description 63
module description 140
module synopsis 168
Display 1
entry point 370
flowchart 317
functional description 63
module description 138
module synopsis 167
Display 2
entry point 370
flowchart 319
functional description 63
module description 138
module synopsis 168
Display 3
entry point 371
flowchart 321
functional description 63
module description 138
module synopsis 171
extended save area 64
Light pen/cursor module
entry point 370
flowchart 333
functional description 64
module description 140
module synopsis 166
Message 1
entry point 370
flowchart 313
functional description 63
module description 137
module synopsis 166
Message 2
entry point 370
flowchart 314
functional description 63
module description 137
module synopsis 166
Message 3
entry point 371
flowchart 316
functional description 63
module description 137
module synopsis 169
Model 85 I/O module
entry point 370
flowchart 312
functional description 63
module description 136
module synopsis 166
Open/Close module
entry point 370
flowchart 354
functional description 63
module description 135
module synopsis 166
Options module
entry point 370
flowchart 325
functional description 63
module description 139
module synopsis 166
PFK 1
entry point 371
flowchart 355
functional description 64
module description 140
module synopsis 168
PFK 2
entry point 371
flowchart 357

functional description 64
module description 140
module synopsis 169
Processor 0 load 1
entry point 371
flowchart 336
functional description 63
module description 134
module synopsis 170
Processor 0 load 2
entry point 371
flowchart 338
functional description 63
module description 134
module synopsis 170
Processor 1 load 1
entry point 370
flowchart 304
functional description 63
module description 134
module synopsis 167
Processor 1 load 2
entry point 370
flowchart 306
functional description 63
module description 135
module synopsis 167
Roll Mode module
entry point 370
flowchart 323
functional description 63
module description 138
module synopsis 166
Status display 1
entry point 371
flowchart 342
functional description 64
module description 141
module synopsis 169
Status display 2
entry point 371
flowchart 343
functional description 64
module description 141
module synopsis 169
Status display 3
entry point 371
flowchart 345
functional description 64
module description 142
module synopsis 169
Status display 4
entry point 371
flowchart 347
functional description 64
module description 142
module synopsis 170
Status display 5
entry point 371
flowchart 348
functional description 64
module description 142
module synopsis 170
Status display 6
entry point 371
flowchart 351
functional description 64
module description 142
module synopsis 170
Status display 7
entry point 371
flowchart 353
functional description 64
module description 142
module synopsis 170
Timer/Interpreter module
entry point 370
flowchart 334
functional description 64
module description 140
module synopsis 167
2250 I/O 1 module
entry point 370
flowchart 307
functional description 63
module description 135
module synopsis 167
2250 I/O 2 module
entry point 370
flowchart 309
functional description 63
module description 136
module synopsis 167
2260 I/O 1 module
entry point 370
flowchart 310
functional description 63
module description 136
module synopsis 167
2260 I/O 2 module
entry point 371
flowchart 311
functional description 63
module description 136
direct access device
shared 4,24
Dispatcher 12,1,9
entry point 369
flowchart 180
functional description 15
module description 71
module synopsis 148
dispatching priority
changing 20,79
specifying 78
display control module (DCM) 64
format of 385,388
dummy program request block (PRB) 34
ATTACH and 18,78
LINK and 104
dump
data set 31,90
indicative 31,92
dynamic area
contents of 6
Dynamic Device Reconfiguration 61

elapsed time
determining 46
end-of-task
system routine 18,29,78
user routine 27,18,36
ENQ
count 84,24
entry point 373

flowchart 193
functional description 23
module description 83
module synopsis 150
entry table 53,54
format 394
ETXR parameter 27,78,102
event control block (ECB)
ATTACH routine 18,78
communications task
attention 41,42,115
DOM 42
external 41,42,115
Recovery Management Support 42
WTO 41,42,113
format 395
POST routine and 23,82,83
WAIT routine and 22,82
event indication list (EIL) 41,42,113
exit
type 1 SVC routine 12,13
entry point 370
flowchart 173
functional description 14
module description 68
module synopsis 147
type 2, 3, and 4 SVC routine 12
user specified routine 14,78
EXIT
entry point 369,372
flowchart 176
functional description 14,29
module description 69
module synopsis 148
Extended Precision Floating Point Decimal Simulator 4
entry point 369,373
functional description 66
module description 144
Extended SVC Router
description 69
flowchart 184
EXTRACT
entry point 373
flowchart 189,190
functional description 17
module description 80
module synopsis 149

FINCH 11
ATTACH and 102
Dispatcher and 72
entry point 369,376
flowchart 246-249
functional description 35
module description 104
module synopsis 158
SVC SLIH and 69
FINCH request block (FRB) 33
creating 36,103
format of 402
queuing 36
removing from queue 94
free queue element (FQE) 39,111
format of 396
FREEMAIN 11
entry point 372
flowchart 253
functional description 40
module description 112
module synopsis 159

Generalized Trace Facility 4,65
GETMAIN
entry point 372
flowchart 253
functional description 39
module description 111
module synopsis 159
gotten subtask-area queue element (GQE)
creating 40,112
format 396
releasing 40,112

hiearchy support (see Main Storage Hierarchy Support)

IDENTIFY
entry point 373
flowchart 250
functional description 37
module description 105
module synopsis 158
indicative dump 31,93
input/output (I/O)
error transient area 5,36
first-level interruption handler (FLIH) 12,42
entry point 365
flowchart 178
functional description 15
module description 70
module synopsis 147
interruption 11,15
New PSW 15
Old PSW 15
interruption
input/output 1,11,15
machine-check 1,11,16
program check 1,11,15
pseudo-disabled 14,35,71
supervisor call 1,11
timer/external 1,11,15
types 1,11
interruption queue element (IQE)
creating 27,78
format of 396
releasing 70,94
interruption request block (IRB) 34
creating 27,36,78,102
format of 402
interval
converting time of expiration to 49,121
timer 45,30

job pack area queue 102-105
contents of 34
job-step timing 46,50,52,77

limit priority
changing 20,79
specifying 78
LINK
attaching a subtask 19
entry point 372
flowchart 243
functional description 35,36
module description 104
module synopsis 158
LOAD
entry point 372
flowchart 243
functional description 35,36
module description 103
module synopsis 158
loaded program list 102-105
contents 34
loaded program request block (LPRB) 34,103,104
format of 402
minor 34,102
loaded request block (LRB) 34,104
format of 402

machine-check
handler 16,60
interruption 16
New PSW 16
main storage
allocating 39,111
area 6
expanding 6
fixed area 5
freeing 40,112
partitions 6
system queue area 6,39,78,111
transient areas 5
Main Storage Hierarchy Support 4,6,39
master scheduler
resident data area 39
message information list format 397
minor loaded program request block (LPRB) 34
Model 85 CRT Display 63
Model 165 Display Console 63
Multiple Console Support
(see communications task)
Multiple-line WTO modules
entry point 371
flowchart 258
module descriptions 114-115
module synopsis 163-164
"must complete" status 24,84
"reset must complete" 86

NEW (TCB pointer) 15
ATTACH and 19
DEQ and 86
Dispatcher and 15,70-77
EXIT and 70
LINK and 104
POST and 83
SVC SLIH and 69
nonresident (type 3 and 4) SVC routines 12
Nucleus Initialization Program 13,19

OLD (TCB pointer) 15,68
Online Test Executive Program (OLTEP) 31
Overlay supervision
entry point 372
flowchart 303
nonresident module 124
resident module 124

PANIC 14,35
partition
task control block 7,19
POST
entry point 372
flowchart 192
functional description 21
module description 82
module synopsis 149
priority
changing 20,79
dispatching 19,78
limit 20,78
Program Fetch 35,105
program-check
first-level interruption handler 12,14
entry point 369
flowchart 179
functional description 14
module description 71
module synopsis 148
New PSW 14
Old PSW 14,71
program interruption control area (PICA) 71,87
creating 25
format of 397
program interruption element (PIE) 71,87
creating 25,87
format of 398
program mask 25,87
program request block (PRB) 34
creating 37
dummy 18,19,78,79,104
format of 402
program status word (PSW)
ASIR 3 and 96
ATTACH and 18,19,102
current 11,14
Dispatcher and 72
new 1,11,14
old 1,11,14
resume 15
SPIE and 87
STAE and 32
protection key
validity checking 15
pseudo clock
local time 46,49
six hour 46,48,49
twentyfour hour 46,49
pseudo-disabled interruption supervision 14,35

qname 20,84
queue control block (QCB) 4
major 20-25,84-87

minor 20-25,84-87
queue element (QEL) 20-24,84-87

RAM=
entry 6
REAL parameter in STIMER 50
record entry format
channel error 400
machine-check 399
recursion
ABEND 31
CLOSE 31
DAR 31
invalid 31
message 31
OPEN 31
relocation dictionary (RLD) 54
reply queue element 41,115
format of 406
request
for main storage
conditional 39,111
unconditional 111
for resources
exclusive 21,83
shared 21,83
request block
active queue 34,28
format 402
on job pack area queue 34
on loaded program list 34
on resident reenterable module area queue 35
use count 37,103
(see also specific request block)
request queue element (RQE)
creating 27
format of 406
RESERVE
processing in shared DASD 24
"reset must complete" parameter 86
resident access method queue 35,103,104
resident reenterable module area queue 35,103,104
routines 6
resident option
type 3 and 4 SVC modules 13
resident SVC load list 14,35
entries 411
resource
exclusive request for 21
queues 21
shared request for 21
restart
automatic 62
deferred 62
Direct Access Positioning
entry point 374
module description 129
Exit
entry point 374
module description 130
Final Processing
entry point 374
module description 130
flowchart 361
functional description 62
Housekeeping 1
entry point 375
module description 127
Housekeeping 2
entry point 375
module description 127
JFCB 1
entry point 374
module description 128
JFCB 2
entry point 374
module description 128
Mount/Verify Direct Access
entry point 374
module description 128
Mount/Verify Nondirect Access
entry point 379
module description 128
Nondirect Access Positioning
entry point 374
module description 129
Repmain 1
entry point 376
module description 127
Repmain 2
entry point 379
module description 127
RET parameter in ENQ 23,85
RETURN 103
rname 20,84
RSVC=
entry 6

SCB
creation 25,88
format 408
SEGLD (overlay supervision) 53
segment table 53
format 407
SEGWT (overlay supervision) 53
SER0 131
flowchart 182
SER1 132
flowchart 183
Shared Direct Access Device 4,24
SPIE
entry point 372
flowchart 197
functional description 25
module description 87
module synopsis 150
STAE
control block 25,88
format 408
entry point 375
flowchart 198
functional description 25
module description 88
module synopsis 150
parameter list 408
format 408
retry routine 26,31,88
Stage 1 Exit Effector
entry point 373
flowchart 199
functional description 29
module synopsis 150

Stage 2 Exit Effector
entry point 369
flowchart 200
functional description 29
module synopsis 150
Stage 3 Exit Effector
entry point 369
flowchart 201
functional description 29
module synopsis 150
step deletion routines
GO 30,93
SMALLGO 30,93
STIMER
entry point 373
flowchart 298,301
functional description 49
module description 121-123
module synopsis 165
REAL parameter 49
TASK parameter 49
WAIT parameter 49
storage utilization block format 409
subpools
invalid 39
specifying 39,111
subtask
active 19
creation 18,78
task control block (TCB) 78,94
subtasking 17
supervisor call (SVC)
exit routine (see exit)
first-level interruption handler
entry point 369
flowchart 173
functional description 14
module description 68
module synopsis 147
interruption 1,14
New PSW 14
nonresident routines 12,36
Old PSW 14,68,82
record 410
relocation table 12
resident type 3 and 4 option 13
routine types 11-12
second-level interruption handler
entry point 369,376
flowchart 174
functional description 14
module description 68
module synopsis 148
table 12
extension 13
format 411
transient area 4,36
supervisor request block (SVRB) 34
creation 14,68
format of 402
SVC DUMP 1
entry point 375
flowchart 228
module description 97
module synopsis 157
SVC DUMP 2
entry point 375
flowchart 230
module description 97
module synopsis 157
SYNCH
entry point 372
functional description 37
module description 105
SYSABEND 31,92
System Environment Recording, Editing, and Printing (SEREP) 16,60
system interruption request block (SIRB) 34,29,70
format 402
System Log Function 42
System Management Facility 4,52
EXCP Counting routine
flowchart 362
functional description 67
module description 145
Output Limit Expiration routine
flowchart 364
functional description 67
module description 145,123
Wait Time Collection routine
flowchart 179
functional description 67
module description 144
system output queue 42,118
system queue area (SQA) 4,40,111
SYSUDUMP 31,92
SYS1.LINKLIB 6
SYS1.NUCLEUS 6
SYS1.SVCLIB 6

task control block (TCB)
address table 19,78
communications task 6,42
creation of subtask 18,78
extracting information from 17,80
format 412
master scheduler 6
Multiple Console Support Write-to-Log 6
partition 19
pointer to current 15,70
queue 19,6,20,70
recovery management 6
subtask 4,19,78
system error task 6,29
System Management Facility 6
transient area task 6,37
task input/output table (TIOT) 31,80,92
location of 6
TASK parameter in STIMER 50
termination (see also ABEND)
abnormal
intercepting 25
scheduling 30
normal 14,30
TIME
entry point 372
flowchart 294,299
functional description 46
module description 121
module synopsis 165
time
elapsed 46
of day 46,121
of expiration 48-49

Time-of-Day Clock 45-51
timer
  enqueue subroutine 51
    entry point 369
    flowchart 296
  expiration processing 50
  interval 30,45
  measuring intervals 49
  queue maintenance 48
  second-level interruption handler
    entry point 369
    flowchart 295,302
    functional description 48
    module description 121
    module synopsis 165
  units 47
  values 47
timer/external
  first-level interruption
   handler 12,15,50
    entry point 369
    flowchart 178
    functional description 12
    module description 70
    module synopsis 147
  interruption 11,15
  New PSW 15
  Old PSW 15,70
timer queue element (TQE)
  creating 48
  format of 417
  initializing 50
  midnight 48,51
  pointer to 48
  synchronized 48
  ten minute 52,66
time-slice
  control element 19,76,79
    format 418
  processing 4,19,76,70
timing control table 52,67
trace
  routine 4,65
  table 4,65
    record format 419
transient area 36
  entry point 369
  deferring requests for 37
  input/output error 4,36
  loading task 37,107
    flowchart 249
  refresh routine 29,72
  request queue 36,107
  SVC 36,4,30,72
TTIMER
  entry point 373
  flowchart 297,300
  functional description 51
  module description 121
  module synopsis 165

unit control block 25,64
unit control module 41,64
  base
    format 422
  entry 36
    format 424
  prefix
    format 420
use count (in IRB) 78
user error handling routine 25,15
user exit routine
  scheduling 27
user parameter area 5
user save area 5

validity check subroutine
  entry point 370
  functional description 14
  module synopsis 148

WAIT
  count 20,81-82
  entry point 372
  flowchart 191
  functional description 20
  module description 81
  module synopsis 149
  multiple-event 82,83
WAIT parameter in STIMER 50
Wait List Element 36,103,104
write queue element (WQE) 41,64,113
  format of 426-432
Write-to-Operator (WTO)
  entry point 375
  flowchart 257
  functional description 41
  module description 113
  module synopsis 161
  multiple-line (see Multiple-line WTO)
Write-to-Operator with Reply (WTOR)
  entry point 375
  emergency buffer 4
  flowchart 262
  functional description 41
  module description 115
  module synopsis 159
Write-to-Programmer (WTP) 41
WTO (see also Write-to-Operator)
  buffer 4
  macro expansion 433

XCTL
  entry point 372
  flowchart 243
  functional description 35,36
  module description 104
  module synopsis 158

2250 Display Station, Models 1 and 3 63
2260 Display Station, Model 1 with 2848
 Display Control 63
2361 Core Storage 4,39

IBM System/360 Operating System
MFT Supervisor

Order No. GY27-7236-1

READER'S COMMENT FORM

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance or additional publications will delay response, however. *For more direct handling of such requests, please contact your IBM representative or the IBM Branch Office serving your locality.*

How did you use this publication?

- ☐ As an introduction
- ☐ As a reference manual
- ☐ For another purpose (explain)_____
- ☐ As a text (student)
- ☐ As a text (instructor)

_____

Please comment on the general usefulness of the book; suggest additions, deletions, and clarifications; list specific errors and omissions (give page numbers):

What is your occupation?_____

Number of latest Technical Newsletter (if any) concerning this publication:_____

Please include your name and address in the space below if you wish a reply.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

Cut or Fold Along Line

Your comments, please . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Fold Fold

First Class
Permit 40
Armonk
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department 636
Neighborhood Road
Kingston, New York 12401

Fold Fold

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

Cut or Fold Along Line

MFT Supervisor Printed in U.S.A. GY27-7236-1

MFT Supervisor Printed in U.S.A. GY27-7236-1

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]