File No. S360-36
Form Y28-6659-3

Program Logic

IBM System/360 Operating System
MVT Supervisor

Program Number 360S-CI-535

This publication describes the internal logic of the MVT supervisor. The MVT supervisor is one part of the control program of the IBM System/360 Operating System. The supervisor controls the basic computing system and programming resources needed to perform several data processing tasks concurrently. Specifically, it was designed to:

- Handle interruptions.
- Supervise tasks.
- Control programs in main storage.
- Control main storage itself.
- Supervise the timer.
- Supervise console communications and the system log.
- Handle checkpoint restarts.
- Supervise exiting procedures.
- Supervise termination procedures.

Program Logic Manuals are intended for use by IBM customer engineers involved in program maintenance, and by system programmers involved in altering the program design. Program logic information is not necessary for program operation and use; therefore, distribution of this manual is limited to persons with program maintenance or modification responsibilities.

The information in this publication applies only to systems capable of multiprogramming with a variable number of tasks (MVT).

Restricted Distribution

RESTRICTED DISTRIBUTION: This publication is intended primarily for use by IBM personnel involved in program design and maintenance. It may not be made available to others without the approval of local IBM management.

Fourth Edition (November, 1968)

This edition corresponds to Release 17 of System/360 Operating System and obsoletes Y28-6659-2. It provides information concerning the Main Storage Hierarchy Support, Model 91, Checkpoint/Restart, and Multiprocessing special features. It also provides information concerning the STAE macro instruction and fail soft changes to the ABEND routine.

New or modified text is indicated by a vertical line to the left of the change. The symbol • to the left of an illustration caption indicates a new or modified illustration.

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 637, Neighborhood Road, Kingston, New York 12401

The information in this publication is organized to enable you to read selectively: for an overview of a function performed by the MVT Supervisor, or for the details of how that function is performed. The "Introduction" section describes the general operation of the MVT Supervisor. The following sections: "Interruption Handling," "Task Supervision," "Contents Supervision," and so on, describe functions first in general terms, and then in detail.

Many special features have been made available under the System/360 Operating System, and more are planned for future releases. Release 15 included the Rollout/Rollin feature; Release 16 added the Time-Slicing, Shared Direct Access Storage Device, and 2250 System Operator's Console features; Release 17 adds the Multiprocessing, Main Storage Hierarchy Support, and Model 91 features. A brief description of the purpose and function of each is provided in the Introduction section. Modifications to MVT Supervisor operation are indicated throughout the document; additional detailed information if required, is contained in Section 11: Special Features.

Your reading of this PLM will be aided by the reference information that appears in the sections at the back of the book. The sections consist of "Control Blocks and Tables," "Program Organization," and "Flow Charts." The tables in the "Program Organization" section tabulate varied information about each supervisor routine: entry point name, routine name, module name, library name, invoking macro instruction, etc.

Note: The area of main storage used exclusively by system routines is called, in this manual, "supervisor queue area" or "supervisor queue space." In MVT Control Program Logic Summary it is called "system queue area."

PREREQUISITE PUBLICATIONS

IBM System/360 Operating System:

MVT Control Program Logic Summary, Form Y28-6658

Supervisor and Data Management Services, Form C28-6646

Supervisor and Data Management Macro Instructions, Form C28-6647

Messages and Codes, Form C28-6631 (useful for the formats and meanings of error codes)

PUBLICATIONS TO WHICH THE TEXT REFERS

IBM System/360 Principles of Operation, Form A22-6821
(referred to in "Interruption Handling")

IBM System/360 Operating System:

Job Control Language, Form C28-6539
(referred to in "Abnormal Termination")

Linkage Editor, Program Logic Manual, Form Y28-6610
(referred to in "Contents Supervision")

Linkage Editor, Form C28-6538
(referred to in "Contents Supervision")

MVT Job Management, Program Logic Manual, Form Y28-6660
(referred to in "Task Supervision" and "Checkpoint/Restart")

I/O Supervisor, Program Logic Manual, Form Y28-6616
(referred to in the description of the Program Fetch routine in "Contents Supervision", in the description of the rollout/rollin Start Transfer routine in "Main Storage Supervision", and in "Interruption Handler")

Machine-Check Handler, Program Logic Manual, Form Y27-7155
(referred to in "Interruption Handler")

System Generation, Form C28-6554
(referred to in "Interruption Handler")

Utilities, Program Logic Manual, Form Y28-6614 (referred to in "Interruption Handler")

IBM System/360 Component Descriptions, Form A26-5988
(referred to in the description of the rollout/rollin Start Transfer routine in "Main Storage Supervision")

SECTION 1: INTRODUCTION 13
Task Supervision 14
Contents Supervision 16
Main Storage Supervision 16
Timer Supervision 17
Console Communications and System Log 17
Recording and Using Checkpoints 17
Exiting Procedures 18
Task Termination 18
Special Features 18
Time Slicing 18
Shared Direct Access Device 18
2250 System Operator's Console 18
Rollout/Rollin 18
Multiprocessing 18
Main Storage Hierarchy Support 19
Summary 19

SECTION 2: INTERRUPTION HANDLING 21
SVC Interruption Handling 21
Main Functions 21
Saving the Status of the Interrupted Calling Routine 21
Ensuring That Both CPUs in a Multiprocessing System do not Perform Supervisor Routines Simultaneously 22
Determining Whether a Supervisor Request Block is Needed 23
Constructing, Initializing, and Queuing the SVRB 23
Determining if the SVC Routine is Normally Resident in Main Storage 24
Fetching a Nonresident Routine From Auxiliary Storage 25
Deferring the Request 28
Restarting Deferred Requests 29
Minor Functions of SVC Interruption Handling 29
Program Interruptions 29
Multiprocessing Program Interruption Handler 30
Model 91 Program Interruption Handler 31
Handling Decimal Instructions 31
Entry From the Testran Interpreter 31
External Interruptions 31
Uniprocessing System 31
Multiprocessing System 32
Input/Output Interruptions 33
Machine Interruptions 33
System Environment Recording 34
SER0 35
SER1 36
Machine-Check Handler for Model 65 (MCH/65) 37

SECTION 3: TASK SUPERVISION 39
Services Directly Related to a Task Control Block 39
Attaching a Subtask 39
Obtaining Storage Space 40
Initializing the IQE, IRB, and TCB 41
Propagating Fields From the TCB of the Attaching Program 41
Placing Parameter Information in the Fields of the Subtask TCB . 42
Special Processing for Time Slicing 42
Allocating Subpools of Main Storage to the Subtask 43
Placing the Subtask TCB on Its Queues 43
Indicating to the Dispatcher the Need for a Task Switch 44
Preparation for the Dispatching of the Caller and the Fetching of the Specified Program 44
Changing the Priority of a Task 45
Extracting Information From a Task Control Block 47

Detaching a Subtask 49
Services Indirectly Related to a Task Control Block 50
Specifying a Program Interruption Exit Routine 51
Synchronizing a Program With One or More Events 52
Causing a Program to Wait for One or More Events 53
Indicating the Occurrence of an Event and Restarting a Waiting Program 55
Serializing the Use of a Resource 58
Types of Resource Requests 58
Description of the Resource Queues 58
Requesting One or More Resources 61
Signaling That the Use of One or More Resources is Complete . . . 66
Scheduling a User Exit Routine 70
The Stage 1 Exit Effector (CIRB Routine) 72
The Stage 2 Exit Effector 72
The Stage 3 Exit Effector 72
The Exit Routine 74
Specifying A Task Asynchronous Exit Routine 76
The STAE Service Routine 76
ABEND/STAE Interface 1 Routine (ASIR1) 77
ABEND/STAE Interface 2 Routine (ASIR2) 78
ABEND/STAE Interface 3 Routine (ASIR3) 79
Services Internal to the Supervisor 80
Testing and Indicating the Need for a Task Switch 80
Testing the Validity of User-Supplied Addresses 81
Changing the Status of Tasks 82
Setting or Resetting the Nonrolloutable Status 82
Setting or Resetting the "Must Complete" Status 82
Setting or Resetting Nondispatchability 82
Determining the Relative Dispatching Priorities of Tasks 83
Testing the Dispatchability of Tasks 84
Initiating An External Interruption In A Second CPU 84

SECTION 4: CONTENTS SUPERVISION 85
The Common Functions of Contents Supervision 85
General Description of the Common Functions 85
Allocation of an Available Module 86
Deferring the Request for an Unavailable Module 86
Preparing to Fetch a Module 86
Fetching the Module 87
Updating the Contents Directory 87
Restarting Deferred Requests 87
Detailed Descriptions of the Common Functions 87
Searching for the Module 87
Creating a Contents Directory Entry 88
Testing Module Status 88
Fetching the Module 89
Performing Alias Processing 90
Deferring a Request 90
Restarting Deferred Requests 90
Scheduling Execution of the Module 91
Special Functions of Contents Supervision 92
Final LOAD Processing 92
Special XCTL Processing 92
Processing if the Requestor is a User Program or a User Exit Routine 92
Processing if the Requestor is an SVC Routine 93
Informing the Supervisor of an Embedded Module Entry Point 97
Informing the Supervisor That a Loaded Module Is No Longer Needed in Main Storage 99
Supervising the Loading of Segments of an Overlay Module 100
Preparatory Linkage Editor Functions 100
Functions of the Overlay Supervisor 100
Linkage to the Overlay Supervisor 101
Types of Processing 101
Determining the Segments That Must Be Loaded 101
Controlling the Loading of Needed Segments 103
Preparation for an Unassisted Branch to the Loaded Segment . . .103

Passing of Control 105
Fetching Routines and Modules to Main Storage 105
Fetching SVC and I/O Error-Handling Routines 106
Fetching Nonresident Modules 106

SECTION 5: MAIN STORAGE SUPERVISION 117
Interruption Handling for Main Storage Supervision 118
Allocating Main Storage 119
Allocating a Region 119
Allocating Space within a Region 122
Processing if the Requested Space is Available 122
Processing if the Requested Space is not Available 123
Allocating a Borrowed Region Through Rollout 127
Determining Whether Rollout Should Be Performed 128
Obtaining the Needed Space from Unassigned Storage 129
Obtaining a Job Step Suitable to Be Rolled Out 129
Processing If a Job Step Suitable for Rollout Cannot Be Found . .131
Processing If a Suitable Job Step Can Be Found 131
Transferring the Contents of the Selected Region to the Rollout Data Set 135
Allocating the Borrowed Region to the Requestor's Job Step . . .137
Processing If I/O Error Occurred During Rollout 138
Exiting From the Rollout/Rollin Module 138
Allocating Space in the System Queue Area 138
Subpool 253 Allocation 139
Subpool 254 Allocation 139
Subpool 255 Allocation 139
Freemain Routine 139
Freeing Space Assigned to a Region 139
Freeing Space Within a Region 140
Freeing One or More Borrowed Regions Through Rollin 141
Freeing the Borrowing Job Step's PQE 141
Determining Whether the Rolled-Out Job Step Should Be Rolled In .141
Transferring the Rolled-Out Job Step to Main Storage 141
Restarting Deferred I/O Requests for the Rolled-In Job Step . .142
Restarting Deferred Operator Replies for the Rolled-In Job Step .143
Making Dispatchable the Tasks of the Rolled-In Job Step 143
Performing Final Common Housekeeping 144
Scheduling Deferred Rollout Requests 144
Freeing Space in the System Queue Area 145

SECTION 6: TIMER SUPERVISION 147
Timer SVC Interruption Handling 147
Time Routine 148
STIMER Routine 148
The Timer Queue 149
Converting Times of Expiration to Intervals 149
Building Timer Queue Elements 150
Timer Interruption Handling 150
Determining What Actions Are to be Performed 150
Returning 6-Hour and Midnight Elements to the Queue 151
TTIMER Routine 152
Determining Remaining Time 153
Canceling an Interval 153

SECTION 7: CONSOLE COMMUNICATIONS AND SYSTEM LOG 155
Supporting Console Communications 155
Reply Processing 156
Supporting the System Log 157

SECTION 8: CHECKPOINT/RESTART 159
Checkpoint (SVC 63) 159
Parameter and environment check 160
Parameter Check (IGC0006C) 160
Environment Check (IGC0106C) 161
JCT Processing (IGC0206C) 161
CANCEL Processing 161
Purging I/O Requests 162

Describing Data Set Status 162
Writing Out the CHR (IGC0A06C) 162
Building and Writing DSDRs (IGC0D06C) 162
Copying the Region 163
Writing CIRs (IGC0F06C) 163
Building and Writing SURs (IGC0G06C and IGC0H06C) 163
Restoring I/O Requests 163
Checkpoint Exit Routine 163
General Clean-up (IGC0Q06C) .163
Message Module (IGC0S06C) 164
Restart (SVC 52) 164
Obtaining and formatting storage 165
Obtaining Storage (IGC0005C) .165
Checkpoint Data Set Initialization (IGC0105B) 165
Restoring the step to Main Storage 165
Restoring Main Storage (IGC0505B) 166
SUR Processing (IGC0605B, IGC0705B, IGC0805B, IGC0905B) 166
JFCB Processing 166
Table Build Module (IGC0G05B) 166
Table Complete Module (IGC0I05B) 166
Mounting and Verifying Volumes 166
Non-Direct Access Processing (IGD0K05B): 166
Direct Access Mount/Verify Module (IGC0M05B) 167
Positioning Open Data Sets 167
Restarting I/O Requests 168
Restart Exit Routine 168

SECTION 9: EXITING PROCEDURES 169
Handling Return From Type-1 SVC Routines 169
Preparing for Return From Programs Other Than Type-1 SVC Routines . .169
Preparing for Return From a User Program Check Routine 170
Preparing for Return From Programs Controlled by RBs 170
If the Returning Routine Is an SVC Routine 171
If the Returning Routine Is a User Program 171
If the Returning Program Is a User Exit Routine 173
Common Processing 174
The Transient Area Refresh Routine 174
Dispatching (Performing the Actual Return of Control) 175
Determining and Giving Control to the Current Routine of the Task Next to be Dispatched 175
Normal Dispatcher Processing (Without Time-Slicing) 176
Dispatcher Processing with Time-Slicing (Differences) 177
Completing the Scheduling of User Exit Routines 177
Handling Task and Job Step Timing 177
Handling Task Timing 178
Handling Job Step Timing 178

SECTION 10: TERMINATION PROCEDURES 181
Normal Termination (EOT Routine) 181
Abnormal Termination 185
Scheduling an Abnormal Termination (ABTERM) 186
Processing if Specified Task Has Already Been Terminated 186
Processing if the Task Has Already Been Scheduled for Abnormal Termination 186
Processing if the Specified Task is the Job Step Task 188
Processing if the Specified Task is not the Job Step Task 191
Preparation for ABTERM Processing After a Program Interruption (ABTERM Prologue) 192
Dumping Selected Areas of Main Storage (ABDUMP) 193
Processing During ABDUMP1 (Entry Point IGC0005A) 194
Processing During ABDUMP2 (Entry Point IGC0105A) 196
Processing During ABDUMP3 (Entry Point IGC0205A) 196
Processing During ABDUMP4 (Entry Point IGC0305A) 196
Processing During ABDUMP5 (Entry Point IGC0405A) 197
Processing During ABDUMP6 (Entry Point IGC0505A) 198
Processing During ABDUMP11(Entry Point IGC0B05A) 199
Processing During ABDUMP7 (Entry Point IGC0605A) 199
Processing During ABDUMP8 (Entry Point IGC0705A) 200

Processing During ABDUMP9 (Entry Point IGC0805A) 200
Cleanup in the Where-to-Go Routine 201
Performing Abnormal Termination (ABEND Routine) 201
Processing During ABEND1 (Entry Point IGC0001C) 203
Processing During ABEND2 (Entry Point IGC0401C) 203
Processing During ABEND3 (Entry Point IGC0A01C) 205
Processing During ABEND4 (Entry Point IGC0101C) 209
Processing During ABEND5 (Entry Point IGC0201C) 214
Processing During ABEND6 (Entry Point IGC0301C) 220
Permitting the System to Quiesce (System Quiesce Routine) 226

SECTION 11. SPECIAL FEATURES 229
2250 System Operator's Console 229
Communication Task Control Flow 229
Unit Control Module Modification 229
Console Device Support Routines 231
Model 91 Decimal Simulator (IEAXDS00) Routine 231
Relationship to the Operating System 231
Simulator Organization 233
Simulator Control (DECENT) Routine 233
Simulator Routine for Add, Subtract, Zero-and-Add Decimal Instruction (DECASP) 236
Simulator Routine for Multiply Decimal Instruction (DECMP) 237
Simulator Routine for Divide Decimal Instruction (DECDP) 239
Simulator Routine for Compare Decimal Instruction (DECCP) 241
Analyzer/End Routine 241

SECTION 12: CONTROL BLOCKS AND TABLES 243
SVC Table 245
Communications Vector Table (CVT) 246
Task Control Block (TCB) 250
Supervisor Request Block (SVRB) -- for Resident Routine 254
Supervisor Request Block (SVRB) -- for Nonresident Routine 255
Interruption Request Block (IRB) 256
System Interruption Request Block (SIRB) 257
Program Request Block (PRB) 258
Trace Table (Uniprocessing Systems) 260
Trace Table (Multiprocessing Systems) 261
Transient Area Control Table (TACT) 262
Program Interruption Element (PIE) 263
Program Interruption Control Area (PICA) 263
STAE Control Block (SCB) 264
Event Control Block (ECB) 265
Parameter List Element (for the ENQ/DEQ routines) 266
Major Queue Control Block (QCB) .267
Minor Queue Control Block (QCB) .267
Queue Element (QEL) 268
Interruption Queue Element (IQE) 269
Request Queue Element (RQE) 270
Contents Directory Element (CDE) 271
Load List Element (LLE) 272
Partitioned Data Set Directory Entry 272
Scatter Extent List 275
Block Extent List and Note List 276
Scatter/Translation Record 277
Program Fetch Work Area -- (Displacements in Bytes) 278
Program Fetch Buffer Table 279
Control Record 280
Relocation Dictionary (RLD) Record 281
Control and Relocation Dictionary Record 282
Segment Table 283
Entry Table 284
Subpool Queue Element (SPQE) 285
Descriptor Queue Element (DQE) .285
Free Queue Element (FQE) 286
Allocated Queue Element (AQE) 286
GOVRFLB (Origin list for Main Storage Queues) 287
Partition Queue Element (PQE) 288

Dummy Partition Queue Element (DPQE) 289
Relationship of Dummy PQE to TCB and PQE Chain 289
Free Block Queue Element (FBQE) 289
Rollout I/O Queue Element (RIQE) 290
Reply Queue Element 290
SVC Purge Parameter List 291
Timer Queue Element (TQE) 292
Secondary Communications Vector Table 294
ABDUMP Parameter List 295
Time-Slice Control Element (TSCE) 296
Display Control Module (DCM) 297
Description of Fields 298
Multiprocessing Communications Vector Table (MPCVT) 300
Vary Queue Element (VQE) 300
Fail Soft Storage Element Map (FSSEMAP) 301
Sample Dump 302

SECTION 13: CHARTS 313

SECTION 14: PROGRAM ORGANIZATION 459
Module Directory 459
Routine Synopses 483

INDEX 493

Figure 1-1. Overall Flow of the Supervisor 13
Figure 1-2. Flow of Control After the ATTACH Macro Instruction is Issued 15
Figure 1-3. Processing After an SVC Interruption 20
Figure 2-1. Status Saving by the SVC Interruption Handlers 22
Figure 2-2. A Request Block Queue 24
Figure 2-3. The Transient Area Queues 26
Figure 3-1. Queue Relationships Among a TCB, IQE, IRB, and End-of-Task Exit Routine 41
Figure 3-2. TSCE Pointers 46
Figure 3-3. The Task Control Block Queue 47
Figure 3-4. The Handling of Shared and Exclusive Requests 59
Figure 3-5. The Resource Queues 60
Figure 3-6. Determining if the Next Waiting Requestor Should be Readied 68
Figure 3-7. Scheduling of Asynchronous Exit Routines 71
Figure 4-1. Subroutine CDSEARCH Uses the Load List and the Job Pack Queue in its Search for the Module's Name 88
Figure 4-2. Further Search by the Common Subroutines of Contents Supervision if the Module's CDE is not in the Job Pack Queue 89
Figure 4-3. Manipulation of the Caller's RB Queue During Servicing of an XCTL Request 94
Figure 4-4. The Transient Area Queues 95
Figure 4-5. Finding an Extent List by Searching the Job Pack Queue or the Load List 98
Figure 4-6. Organization of an Overlay Module 100
Figure 4-7. Functional Flow of Overlay Supervision 102
Figure 4-8. Use of the Caller's ENTAB to Branch to a Segment . . .103
Figure 4-9. Organization of SEGTAB Entries for a Single-Region Overlay Structure 105
Figure 4-10. Relationships of Program Fetch Routine to Other Routines for the Fetch of an SVC Routine or an I/O Error Routine 107
Figure 4-11. Control Blocks and Tables Used by the Program Fetch Routine 108
Figure 4-12. Relationship of Program Fetch Routine to Other Routines for the Fetch of a Module or Overlay Segment Module or Overlay Segment 109
Figure 4-13. Extent List 109
Figure 4-14. Note List as It Exists in Main Storage 110
Figure 4-15. Typical Load-Module Logical Format on Direct-Access Device 111
Figure 4-16. Overall Control Flow During the Loading of a Module or Segment 112
Figure 5-1. GETMAIN/FREEMAIN SVC Instructions 118
Figure 5-2. Main Storage Supervision Interruption Handling 118
Figure 5-3. Element Relationships: Region Allocation 121
Figure 5-4. List Structure for List Form of GETMAIN Macro Instruction 122
Figure 5-5. Element Relationships for Intra-Region Allocation . . .123
Figure 5-6. Position of Rollout/Rollin TCB on TCB Queue 124
Figure 5-7. Relationship of the Rollout/Rollin TCB, PRB, and IRB During Scheduling of the Rollout/Rollin Task 125
Figure 5-8. Scheduling of Rollout: Overall Flow 125
Figure 5-9. Steps in the Scheduling of the Rollout/Rollin Task . .126
Figure 5-10. Interfaces Between Rollout Module and SVC Purge Routine 133
Figure 5-11. How IOBs for Deferred I/O Requests are Queued 134
Figure 5-12. Element Relationships for System Queue Area Allocation 139

Figure 6-1. Timer SVC Interruption Handling 147
Figure 6-2. Positioning of Elements on the Timer Queue 149
Figure 6-3. Timer Queue Element (TQE) 151
Figure 6-4. Timer Interruption Handling 152
Figure 7-1. Console Support: Input 156
Figure 7-2. Console Support: Output. 156
Figure 7-3. Log Functions 158
Figure 8-1. Checkpoint Processing Routines 160
Figure 8-2. CHECKPOINT Header Record (CHR) 162
Figure 8-3. Restart Processing Routines 165
Figure 8-4. Data Set Descriptor Records (DSDRs) 168
Figure 9-1. The Transient Area Queues 172
Figure 9-2. Locating the Initiator TCB Associated with the Task Next to be Dispatched 179
Figure 10-1. Scheduling of the ABEND Routine by the ABTERM Routine .185
Figure 10-2. A Tree of Subtasks and a Possible Sequence of - Examination 188
Figure 10-3. Format of the Completion Code and the Dump Option Flag in the Parameter Register 193
Figure 10-4. Pointers Used During the Save Area Trace 198
Figure 10-5. Task Relationships During an Abnormal Termination . . .214
Figure 10-6. Preparation for the Dispatching of ABEND5 for the Selected Task 218
Figure 11-1. Control Flow of 2250 System Operator's Console Support .230
Figure 11-2. Relationship of the Decimal Simulator Routine (IEAXDS00) to the Operating System 232
Figure 11-3. Decimal Simulator (IEAXDS00) Routine Organization and Flow of Control 234
Figure 11-4. Storage Protection Checking 235
Figure 11-5. Example of Division by Decimal Simulator 240

TABLES

Table 2-1. Control Block Fields Used by the SVC Second-Level Interruption Handler 38
Table 3-1. Initialization of the Interruption Queue Element 41
Table 3-2. Processing if a Requested Resource is not in Use 62
Table 3-3. Processing if a Requested Resource is in Use 62
Table 3-4. Return Codes for the ENQ Routine 63
Table 3-5. TCB Flags That are Set if a Task is in "Must Complete" Status 65
Table 3-6. Error Conditions When use of a Resource is Signaled Complete 69
Table 3-7. Mask Bit Numbers Used in the STATUS Macro Instruction 83
Table 4-1. Types of Processing During Overlay Supervision 104
Table 4-2. Processing of Segment Table Entries 106
Table 4-3. Channel-Program Switching After a Program-Controlled Interruption 113
Table 4-4. Program Fetch Return Codes 114
Table 4-5. Termination Processing According to Module Type 115
Table 5-1. Subpool Numbers Used for Requesting Space 120
Table 6-1. Actions Taken After Timer Expiration 152
Table 10-1. ABTERM Processing 187
Table 10-2. The TCB Nondispatchability Flags 189
Table 11-1. Organization of the Decimal Simulator (IEAXDS00) Routine 233
Table 14-1. Module Directory .460
Table 14-2. Directory of Entry Point Names and Flowchart Identifications .471
Table 14-3. Table of Routines Invoked by SVC Instructions 481

CHARTS

Chart 00. Overall Control Flow of MVT Supervisor 314
Chart AA. SVC First-Level Interruption Handler 315
Chart AB. SVC First-Level Interruption Handler (Multiprocessing System 316
Chart AC. SVC Second-Level Interruption Handler 317
Chart AD. Transient Area Availability Check Routine 318
Chart AE. Transient Area Fetch Routine 319
Chart AF. Program Check First-Level Interruption Handler 320
Chart AG. Program Check First-Level Interruption Handler (Multiprocessing System) 321
Chart AH. Program Check First-Level Interruption Handler for Model 91 322
Chart AI. External First-Level Interruption Handler (Uniprocessing System) 323
Chart AJ. External First-Level Interruption Handler (Multiprocessing System) 324
Chart AK. I/O First-Level Interruption Handler 325
Chart AL. I/O First-Level Interruption Handler (Multiprocessing System) 326
Chart AM. SER0 Routine 327
Chart AN. SER1 Routine (for Models 40, 50, 60, 75) 328
Chart AO. SER1 Routine (for Model 91) 329
Chart AP. SER1 Routine (for Model 91) 330
Chart BA. Attach Routine 331
Chart BB. Attach Routine 332
Chart BC. Attach Routine 333
Chart BD. Attach Routine (With Time-Slicing) 334
Chart BE. Chap Routine 335
Chart BF. Chap Routine (With Time-Slicing) 336
Chart BG. Chap Routine (With Time-Slicing) 337
Chart BH. Extract Routine 338
Chart BI. Detach Routine 339
Chart BJ. SPIE Routine 340
Chart BK. Wait Routine 341
Chart BL. Wait Routine (Job Step Timing) 342
Chart BM. Post Routine 343
Chart BN. Post Routine (Job Step Timing) 344
Chart BO. ENQ Routine 345
Chart BP. DEQ Routine 346
Chart BQ. DEQ Routine (Shared DASD) 347
Chart BR. Stage 1 Exit Effector 348
Chart BS. Stage 2 Exit Effector 349
Chart BT. Stage 3 Exit Effector 350
Chart BU. Stage 3 Exit Effector 351
Chart BV. Task Switching Routine (Uniprocessing System) 352
Chart BW. Task Switching Routine (Multiprocessing System) 353
Chart BX. STAE Service Routine 354
Chart BY. ABEND/STAE Interface 1 Routine (ASIR1) 355
Chart BZ. ABEND/STAE Interface 2 Routine (ASIR2) 356
Chart B0. ABEND/STAE Interface 3 and 4 Routines (ASIR3, ASIR4) . . .357
Chart CA. Link, Load, XCTL, and SYNCH Processing 358
Chart CB. Link, Load, XCTL, and SYNCH Processing 359
Chart CC. Link, Load, XCTL, and SYNCH Processing 360
Chart CD. Identify Routine .361
Chart CE. Delete Routine 362
Chart CF. Program Fetch Routine 363
Chart CG. Program Fetch Routine 364
Chart CH. Program Fetch Routine (Main Storage Hierarchies) 365
Chart CI. Overlay Supervisor 366
Chart DA. GETMAIN/FREEMAIN Routine 367
Chart DB. GETPART/FREEPART Routine 368
Chart DC. Rollout Criterion Routine 369

Chart DD. Rollin Criterion Routine .370
Chart DE. Rollout/Rollin I/O Routine 371
Chart DF. SVC Purge Interface 372
Chart DG. SVC Restore Interface .373
Chart DH. Rollout/Rollin GETSTEP Routine 374
Chart DI. Rollout/Rollin TESTSTEP Routine 375
Chart DJ. Rollout/Rollin Reply Restore Routine 376
Chart EA. Time Routine 377
Chart EB. STIMER Routine 378
Chart EC. TTIMER Routine 379
Chart ED. Timer Second-Level Interruption Handler 380
Chart EE. Timer Second-Level Interruption Handler
Dequeue and Engueue Subroutines 381
Chart EF. Timer Second-Level Interruption Handler
(Job Step Timing) .382
Chart FA. Checkpoint Housekeeping 1 Routine 383
Chart FB. Checkpoint Housekeeping 2 Routine 384
Chart FC. Checkpoint Housekeeping 3 Routine 385
Chart FD. Check I/O Routine 386
Chart FE. Preserve 1 and 2 Routines 387
Chart FF. Checkmain 1 and 2 Routines 388
Chart FG. Checkmain 3 Routine .389
Chart FH. Resume I/O and Checkpoint Exit Routines 390
Chart FI. Checkpoint Message Module 391
Chart FJ. Restart Housekeeping 1 and 2 Routines 392
Chart FK. Repmain 1 and 2 Routines 393
Chart FL. Repmain 3 and 4 Routines 394
Chart FM. Repmain 5 Routine 395
Chart FN. Repmain 5 Routine 396
Chart FO. JFCB Processor 1 and 2 Routines 397
Chart FP. Mount/Verify 1 (Non Direct-Access) Routine 398
Chart FQ. Mount/Verify 2 (Direct-Access) Routine 399
Chart FR. Sysin/Sysout Data Set Processor 1 and 2 Routines 400
Chart FS. Data Set Processor 1 (Non Direct-Access) Routine 401
Chart FT. Data Set Processor 2 (Direct-Access) Routine 402
Chart FU. Access Method-Disposition and Restart Exit Routines . . .403
Chart GA. Type-1 Exit Routine .404
Chart GB. Exit Routine 405
Chart GC. Exit Routine 406
Chart GD. Transient Area Exit Routine 407
Chart GE. Transient Area Refresh Routine 408
Chart GF. CDEXIT Routine 409
Chart GG. Dispatcher (Uniprocessing System) 410
Chart GH. Dispatcher (Uniprocessing System With Job Step
and Task Timing) 411
Chart GI. Dispatcher (Uniprocessing System With Job Step
and Task Timing) 412
Chart GJ. DJSEARCH Subroutine (Uniprocessing System) 413
Chart GK. Dispatcher (Uniprocessing System With Time-Slicing) . . .414
Chart GL. Dispatcher (Uniprocessing System With Time-Slicing) . . .415
Chart GM. Dispatcher (Uniprocessing System With Time-Slicing) . . .416
Chart GN. Dispatcher (Multiprocessing System) 417
Chart GO. Dispatcher (Multiprocessing System) 418
Chart GP. DJSEARCH Subroutine (Multiprocessing System) 419
Chart HA. EOT Routine 420
Chart HB. Purge Timer Routine .421
Chart HC. Release Main Storage Routine 422
Chart HD. Release Loaded Programs Routine 423
Chart HE. ABTERM Routine 424
Chart HF. ABTERM Routine -- SETSUBS Subroutine 425
Chart HG. ABTERM Prologue Routine 426
Chart HH. ABDUMP Routine 427
Chart HI. ABEND1 428
Chart HJ. ABEND2 429
Chart HK. ABEND3 430
Chart HL. ABEND3 431
Chart HM. ABEND3 -- Steal Core Subroutine 432
Chart HN. ABEND4 433

Chart HO. ABEND5 434
Chart HP. ABEND5 435
Chart HQ. ABEND6 436
Chart HR. ABEND6-RBREMOVE Subroutine 437
Chart HS. ABEND6-RBREMOVE Subroutine 438
Chart HT. System Quiesce Routine 439
Chart IA. 2250 Processor Routine 440
Chart IB. 2250 Processor Routine 441
Chart IC. 2250 Processor Routine 442
Chart ID. 2250 Processor Routine 443
Chart IE. Display Routine 444
Chart IF. I/O-1 Routine 445
Chart IG. I/O-1 Routine 446
Chart IH. I/O-2 Routine 447
Chart II. Option Routine 448
Chart IJ. Open/Close Routine 449
Chart IK. Asynchronous Error Routine 450
Chart IL. Simulator Control Routine (Model 91) 451
Chart IM. Compare Decimal Routine (Model 91) 452
Chart IN. Add/Subtract/Zero-and-Add Decimal Routine (Model 91) . . .453
Chart IO. Add/Subtract/Zero-and-Add Decimal Routine (Model 91) . . .454
Chart IP. Multiply Decimal Routine (Model 91) 455
Chart IQ. Divide Decimal Routine (Model 91) 456
Chart IR. Analyzer/End Routine (Model 91). 457

The MVT supervisor is one part of the control program of IBM System/360 Operating System; it controls the basic computing system and programming resources needed to perform several data processing tasks concurrently. The entire control program is introduced in the publication IBM System/360 Operating System: MVT Control Program Logic Summary, Form Y28-6658.

Job steps, designated by the job management routines as tasks, are carried out 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 required for the performance of 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.

All supervisor activity begins with an interruption. In IBM System/360 the interruption is a machine characteristic; it is the means by which the supervisor gets control of the CPU to provide resources for the performance of tasks. An interruption may be planned (specifically requested in the program currently being executed by the CPU) or unplanned (caused by an event that may be either related or unrelated to the task currently being performed).

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 System/360 interval timer, the console interrupt key, or the direct control feature.
- Input/output interruption: the signal that an input/output event has occurred.
- Program interruption: a signal that a program has attempted an invalid action.
- Machine-check interruption: the signal that a machine error has occurred.

Overall operation of the supervisor is shown in Figure 1-1. The program being executed in the performance of task A has been interrupted, possibly because it contained a request for a supervisor service, possibly because an input/output operation has been completed for an entirely different task.

The interruption-handling portion of the supervisor (represented by the top box in Figure 1-1) analyzes the interruption, based on control information passed to it at the time of the interruption. Each of the five interruption types has associated with it two program status words (PSWs) called "old" (OPSW) and "new" (NPSW). The

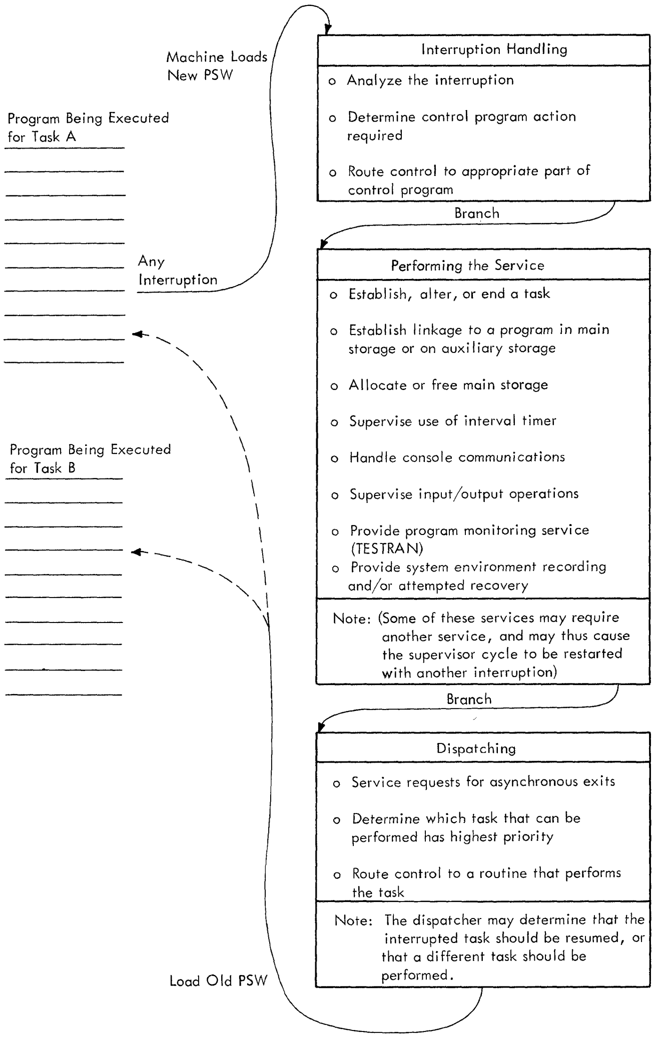

Machine Loads New PSW
Program Being Executed for Task A
Any Interruption
Program Being Executed for Task B
Load Old PSW
Interruption Handling
o Analyze the interruption
o Determine control program action required
o Route control to appropriate part of control program
Branch
Performing the Service
o Establish, alter, or end a task
o Establish linkage to a program in main storage or on auxiliary storage
o Allocate or free main storage
o Supervise use of interval timer
o Handle console communications
o Supervise input/output operations
o Provide program monitoring service (TESTRAN)
o Provide system environment recording and/or attempted recovery
Note: (Some of these services may require another service, and may thus cause the supervisor cycle to be restarted with another interruption)
Branch
Dispatching
o Service requests for asynchronous exits
o Determine which task that can be performed has highest priority
o Route control to a routine that performs the task
Note: The dispatcher may determine that the interrupted task should be resumed, or that a different task should be performed.

Figure 1-1. Overall Flow of the Supervisor

OPSW contains the information needed by the supervisor to analyze the interruption. The NPSW contains the address of the appropriate interruption handling routine.

When an interruption occurs, the CPU stores the contents of the current PSW in the OPSW for that type of interruption, and loads the NPSW. By loading the NPSW, 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 perform the services required as a result of the interruption.

The supervisor itself performs many of the services that are requested through an interruption (these services are represented by the middle box in Figure 1-1). Services that the supervisor provides may be grouped into these general categories:

- Task supervision. The supervisor creates tasks at the request of the job management routines or in response to a request to attach a subtask to an already existing task. The supervisor determines in what order tasks are to be performed.

- Contents supervision. The supervisor keeps records of all programs in main storage, and assigns these programs to perform tasks. The Program Fetch routine brings requested programs into main storage from secondary storage.

- Main storage supervision. The supervisor assigns main storage needed to perform job steps and tasks within job steps.

- Timer supervision. The supervisor controls the use of the System/360 interval timer.

- Console communications and system log. The supervisor provides the means for the operator to directly communicate with the system, and for a program to write a console message to the operator. It also provides support for the system log, which is a repository of statistical information about system usage.

- Recording and using checkpoints. On request, the supervisor writes records of a task's main storage region and the necessary task control information so that the task may be restarted from that point at a later time.

- Exiting procedures. The supervisor provides routines that prepare for the return of control from a completed program.

- Task termination. The supervisor provides for normal and abnormal termination of tasks.

- Recovery management. The optional service routines SER0 and SER1 provide for the recording of information related to a machine malfunction. The optional Machine-Check Handler for Model 65 (MCH/65), besides recording environmental data, attempts to analyze the malfunction and restore the system to normal operation.

After a control program service has been performed, the supervisor determines what task is to be performed next. The supervisor Dispatcher routine (represented by the bottom block in Figure 1-1) returns control to a processing program (or possibly to a supervisor routine). As seen in Figure 1-1, the program to which control passes need not be the one that was interrupted. The Dispatcher may determine that as a result of the interruption, task B, which has a higher priority than task A, should be performed next.

TASK SUPERVISION

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

When the operating system is generated, certain key TCBs are built into the system. These TCBs represent: the master scheduler task of job management, the system error task, the rollout/rollin task,[1] the communications task, and one transient area fetch task for each transient area. All other task control blocks are constructed by the supervisor Attach routine, at the request of either the control program or a user program. The Master Scheduler can attach up to fifteen Initiator/Terminator tasks, one for each storage protection key available. Initiator/Terminator routines attach job step tasks and subtasks. An entire tree structure of related tasks may thus be formed.

All the TCBs in the system are chained together, according to dispatching priority, to form the TCB queue. The transient area fetch TCBs are at the top of the queue, followed in order by the system error TCB, the rollout/rollin TCB,[1] the communications TCB, and the master scheduler TCB. The dispatching priorities of

[1] This task is included if the rollout feature is selected at system generation.

other tasks are assigned by the supervisor according to the parameters given in ATTACH macro instructions. When several TCBs with the same priority appear in the TCB queue, they are ordered first-in, first-out.

Figure 1-2 shows the flow of control that results from the issuance of the ATTACH macro instruction. This flow is typical of the processing that might follow a supervisor macro instruction.

The Attach routine, like other SVC routines, is entered as a result of an SVC interruption. The SVC interruption handling routines analyze the interruption, determine what service is required, and then branch to the Attach routine. The Attach routine obtains main storage space for a TCB by issuing the GETMAIN macro instruction. This causes another SVC interruption, called a nested interruption because it is an interruption to the processing of an interruption. The nested interruption is handled by the supervisor in exactly the same way as the original interruption for the ATTACH macro instruction, except that this time the interruption handler branches to the GETMAIN routine. When the required storage has been allocated, the Attach routine regains control. It initializes certain fields in the TCB and places it on the TCB queue.

After the Attach routine has initialized the control block that represents the new task, it branches to the Dispatcher. The Dispatcher examines the TCB queue to find the highest-priority task that is ready to be performed. This task may or may not be the one that was being performed at the time of the original SVC interruption. For example, if task B has been attached as a subtask of task A, and if B has a higher dispatching priority than A, then task B will be performed before task A is resumed.

The supervisor controls the order in which tasks are performed. This control is accomplished by the Dispatcher, working through the TCB queue. The highest-priority task represented on the TCB queue may not be the one to be performed; it may be waiting for some event (through the WAIT macro instruction, for example) or for a resource that has been serialized (through the ENQ macro instruction). The TCB queue serves as a record of the status of every task in the system.

When the time-slicing feature is included in the system, the dispatcher will contain special code for time-slicing implementation. The dispatcher controls time slicing through the time-slice control element (TSCE); there is one TSCE, assembled at system generation, for each time-slice group.

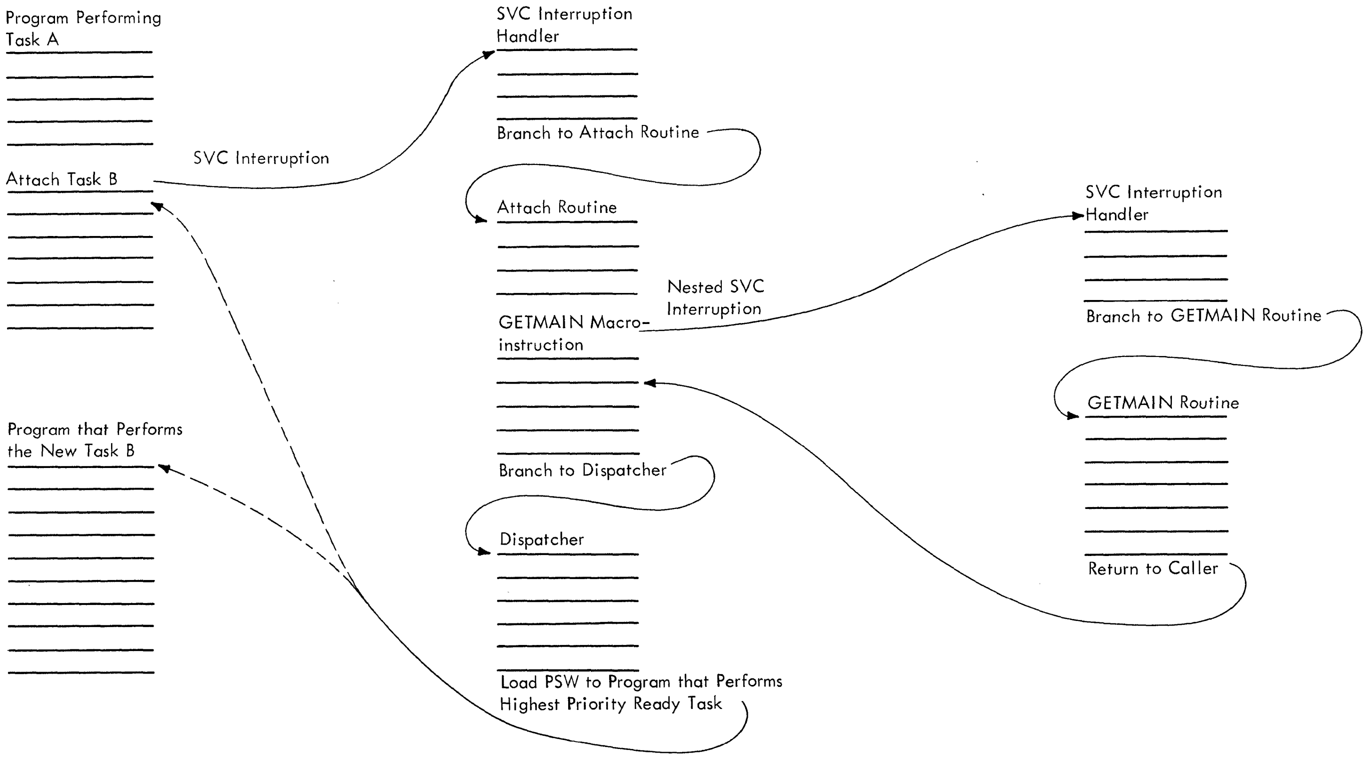


Figure 1-2. Flow of Control After the ATTACH Macro Instruction is Issued

CONTENTS SUPERVISION

Contents Supervision is accomplished through a structure of queues that are very closely related to the TCB queue. These are the request block queues.

Request blocks (RBs) represent levels of control within a task. Contents supervision routines construct an RB for the first level of control in the performance of a new task; this RB and RBs for subsequent levels are chained on the TCB's RB queue. For example, if program A issues a LINK macro instruction specifying program B, the Contents Supervision routines will construct a new RB to represent program B's level of control. When program B has completed its processing, control can pass back to program A. The supervisor uses the RB queue as a record of control levels; it can pass control to succeeding levels and, as the routines complete their operation, pass control back up the line, regardless of the number of times a task is interrupted.

There are four types of RBs:

- Program request blocks (PRBs), which represent nonsupervisory routines that must be executed in the performance of a task. PRBs are created by the contents supervision routines that perform the Attach, Link, or XCTL functions.
- Interruption request blocks (IRBs), which control routines that must be executed in the event of asynchronous interruptions. IRBs are created in advance of an interruption by the CIRB routine at the user's request, but not placed on an RB queue until an interruption actually occurs.
- System interruption request block (SIRB), which is used only for the system error task. There is only one SIRB in the system.
- Supervisor request blocks (SVRBs), which represent supervisor routines. SVRBs are created by the SVC interruption handling routines. They are queued just as PRBs are.

Contents Supervision routines construct only one type of RB: namely, the PRB.

The supervisor maintains a record of all programs in main storage -- their attributes, locations, and use statuses. This record is called the contents directory. The contents directory is made up of three separate queues: (1) the link pack area control queue (LPACQ); (2) the job pack area control queue (JPACQ); (3) the load list.

The LPACQ is a record of every program in the system link pack area. This area contains reenterable routines specified by the control program or by the user. It is loaded by the nucleus initialization program. The routines in the system link pack area can be used repeatedly to perform any task of any job step in the system.

The entries in the LPACQ are contents directory entries (CDEs). When a program represented in the LPACQ is requested for a task, it will be represented in that task's RB queue by a PRB; the address of this PRB will be inserted in the CDE.

There is a JPACQ for each job step in the system that uses a program not in the link pack area. The JPACQ, like the LPACQ, is made up of CDEs. It describes routines in a job step region brought into main storage by contents supervision routines to perform a task in the job step. The routines in the job pack area can be either reenterable or not reenterable. Routines in the job pack area cannot be used to perform a task that is not part of the job step.

The load list represents routines that are brought into a job pack area or found in the link pack area by the contents supervision routines that perform the Load function. The entries in the load list are load list elements, not CDEs. Each load list element is associated with a CDE in the JPACQ or LPACQ; the programs represented in the load list are thus also represented in one of the other contents directory queues.

MAIN STORAGE SUPERVISION

The MVT supervisor controls tasks through the TCB queue and the RB queues; it controls programs through the RB queues and the contents directory. A third major function, controlling main storage, is accomplished through a system of main storage queues.

When the job management routines designate a job step as a task, they request a region of main storage to be used in performing that task. The size of the region is specified by the user; the region contains the job pack area for the step, and all additional working space needed.

All requests for main storage are handled by the GETMAIN SVC routine. The supervisor maintains main storage queues to reflect storage assignments; the GETMAIN routine simply adjusts these queues to reflect new assignments.

When there are no job steps in the system, all of the dynamic area of main storage is treated as one region. It is represented to the supervisor by a free block queue element (FBQE) at the beginning of the area and a partition queue element (PQE) in supervisor queue space. The PQE contains the address of the FBQE, and therefore the address of the beginning of the free area; the FBQE contains the extent of the free area. When space is requested for a job step, the GETMAIN routine subtracts the requested area from the free area and builds a new FBQE and PQE for the new region. The address of the PQE is placed in the TCB of the job step task.

After job step initialization, a program performing a task may request main storage by issuing the GETMAIN macro instruction. The GETMAIN SVC routine allocates the storage only within the region[1] assigned to the job step being performed or within the supervisor queue area.[2] The supervisor maintains a separate chain of queue elements for allocation within a region. This chain keeps track of subpools within the region. A subpool is all of the main storage requested under a label called a subpool number. The storage in a subpool does not need to be contiguous. The chief advantages of subpools are that the storage be shared between tasks, and that all of the storage identified by a subpool number can be released with one FREEMAIN macro instruction.

The supervisor FREEMAIN service routine is used to free main storage space when it is no longer needed to perform a task. Space assigned to a job step, space within a region, and space within the supervisor queue space are all freed by the FREEMAIN routine. The routine makes space available by adding elements to chains in which are recorded all free areas in main storage, and by adjusting the queues of allocated space.

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 permits selective access to either the processor storage (hierarchy 0) or 2361 Core Storage (hierarchy 1) portions of main storage. If 2361 Core Storage is not included and a region is defined to exist in two hierarchies, a two-part region is established within processor storage. The two parts are not necessarily contiguous. A hierarchy parameter (HIARCHY=) in the GETMAIN and GETPOOL macro instructions permits specification of either hierarchy as desired.

TIMER SUPERVISION

The System/360 interval timer is a 32-bit word in lower main storage, that automatically keeps decrementing as long as the system is running and the interval timer switch is on. The supervisor timer service routines enable the programmer to obtain the date and time of day, measure periods of time, or schedule activity for a specific time of day. These routines, performed as a result of the macro instructions TIME, STIMER, and TTIMER, are handled just like any other SVC routines.

The timer queue is a chain of timer queue elements; each element represents an interval request. These queue elements are constructed by the STIMER service routine. The chain is ordered so that the request for the next interval to expire is at the top of the queue. When a requested interval expires, a timer interruption occurs.

The Timer Interruption Handler routine of the supervisor removes the top element from the timer queue and determines what action is to be taken. Examples are scheduling a timer exit or making a task ready to be performed.

CONSOLE COMMUNICATIONS AND SYSTEM LOG

Console communications includes both messages to the operator from a program and messages to the system from the operator. The SVC routines WTO and WTOR (Write to Operator and Write to Operator with Reply) perform output (system to operator) processing. Operator intervention at the console causes an external interruption, which is also handled by the console communications service routines.

The system log is a pair of data sets used by the system for recording statistical information. The supervisor log support routines perform input and output services related to the log.

RECORDING AND USING CHECKPOINTS

The supervisor provides routines to allow a job to be restarted after an abnormal termination. The Checkpoint routine creates a series of records at points in the problem program where the programmer wishes a reexecution to begin. These rec-

[1] If the rollout feature is in the system and rollout can be performed, the GETMAIN routine can allocate space to the job step from a temporary region obtained through rollout.

[2] Storage is allocated in the supervisor queue area only if the requestor is a supervisor routine.

ords include a copy of the task's main storage region, descriptions of data sets, and system control information. The Restart routine uses these records to restore the task to main storage, mount, verify and position its data sets, and give it control at the point where the checkpoint entry was written.

EXITING PROCEDURES

The supervisor provides routines that prepare for the return of control from a completed program and perform the actual return of control. Control may return to a main-line program or to a supervisor routine. The exiting procedures determine what type of program has completed its execution, and perform different clean-up operations for the different types.

The Dispatcher routine is entered to return control to a program belonging to the highest priority ready task. The Dispatcher, as we have previously noted, works through the TCB queue. (There is one case in which the Dispatcher is not entered to return control: when the completed program is a type-1 SVC routine that has not indicated the need for a task switch.)

TASK TERMINATION

The supervisor performs the processing needed when a task is terminated, either normally or abnormally. The termination processing includes releasing system resources that were assigned to perform the task.

The End of Task (EOT) routine performs normal termination processing. Abnormal termination processing is performed by the ABTERM and ABEND routines. The ABDUMP routine provides a dump of TCBs and main storage related to the terminating task.

SPECIAL FEATURES

Time Slicing

The time-slicing mechanism operates within the dispatcner. A priority is assigned to a group of tasks which are to be time-sliced; time-slicing occurs only among the tasks in the group and only when the priority level of the group is the highest priority level that has a ready task. Each task in 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 of higher priority than that of the group becomes ready.

The group of tasks to be time-sliced, the length of the time slice, and the priority of the time-sliced tasks, will be specified by the installation. Any task in the system that is not defined within the group to be time sliced will be dispatched under the current priority structure, i.e., when it is the highest priority ready task, and until it either waits or a task of higher priority becomes active.

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, are the basis for control of direct access storage device sharing between systems. The shared DASD feature provides the control program functions needed to control device reservation and release. Essentially this feature controls the use of a serially reusable resource, the shared data and device.

2250 System Operator's Console

The IBM 2250 Display Unit Model 1 can be used as a system operators console with the IBM System/360 Models 50, 65, and 75; it is standard with the Model 91. If this feature is selected, 2250 System Operator's Console programming support is provided for displaying system states and reference information, and system and problem program messages to the operator.

Console support is part of the Communications Task in Operating System/360 with MVT.

Rollout/Rollin

Rollout/rollin allows the temporary, dynamic expansion of a particular job step beyond its originally specified region size. A job step's region size can be based on a minimum requirement, rather than a maximum. When a job step needs more main storage, this feature attempts to obtain unassigned storage; failing that, another job step is rolled out, i.e., its entire region is transferred to secondary storage, and its storage is made available to the first job step. When released by the first job step, the additional storage is again available as unassigned storage, if that was its source, or to receive the job step to be transferred back into main storage (rolled in).

Multiprocessing

The multiprocessing feature, available with Model 65, enables a single control program to use the productive capability of two CPUs (CPUA and CPUB) so that two tasks can be executed simultaneously.

A multiprocessing system can operate in two modes: multisystem mode or partitioned mode. In the partitioned mode, a multiprocessing system operates as two independent systems, each CPU having its own control program, main storage units and I/O devices. In the multisystem mode, all programs are run under one control program, both CPUs having access to all of main storage, and all I/O devices, except teleprocessing equipment and the 1052 console, being attached to both CPUs. Each CPU has its own 4K byte prefixed storage area and can interrupt the other CPU through a direct hardware connection.

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 (identified as hierarchy 0) or 2361 Core Storage (identified as hierarchy 1) portions of main storage. If 2361 Core Storage is not included in the system and a region is defined to exist in two hierarchies, a two-part region is established within processor storage. The two parts are not necessarily contiguous. Normally, all storage requested by programs of a given step or task is assigned from its region, although the rollout/rollin feature does provide the capability of acquiring temporary additional storage. If the Main Storage Hierarchy Support option has been selected, a region may be defined to consist of two parts: the first located in hierarchy 0 and the second located in hierarchy 1.

SUMMARY

The supervisor is a collection of programs for handling interruptions and providing services for them. These interruptions are the basic method by which the control program manages data processing tasks. The supervisor functions are largely performed by routines that manipulate a network of control queues -- the TCB queue, the RB queues, the contents directory, the main storage queues, and the timer queue.

The processing after a timer/external, input/output, program, or machine interruption is generally straight forward. Figure 1-1 reflects what happens after one of these interruptions. The processing after an SVC interruption is a little more complicated. Figure 1-3 provides a more detailed, although still simplified, picture of this processing.

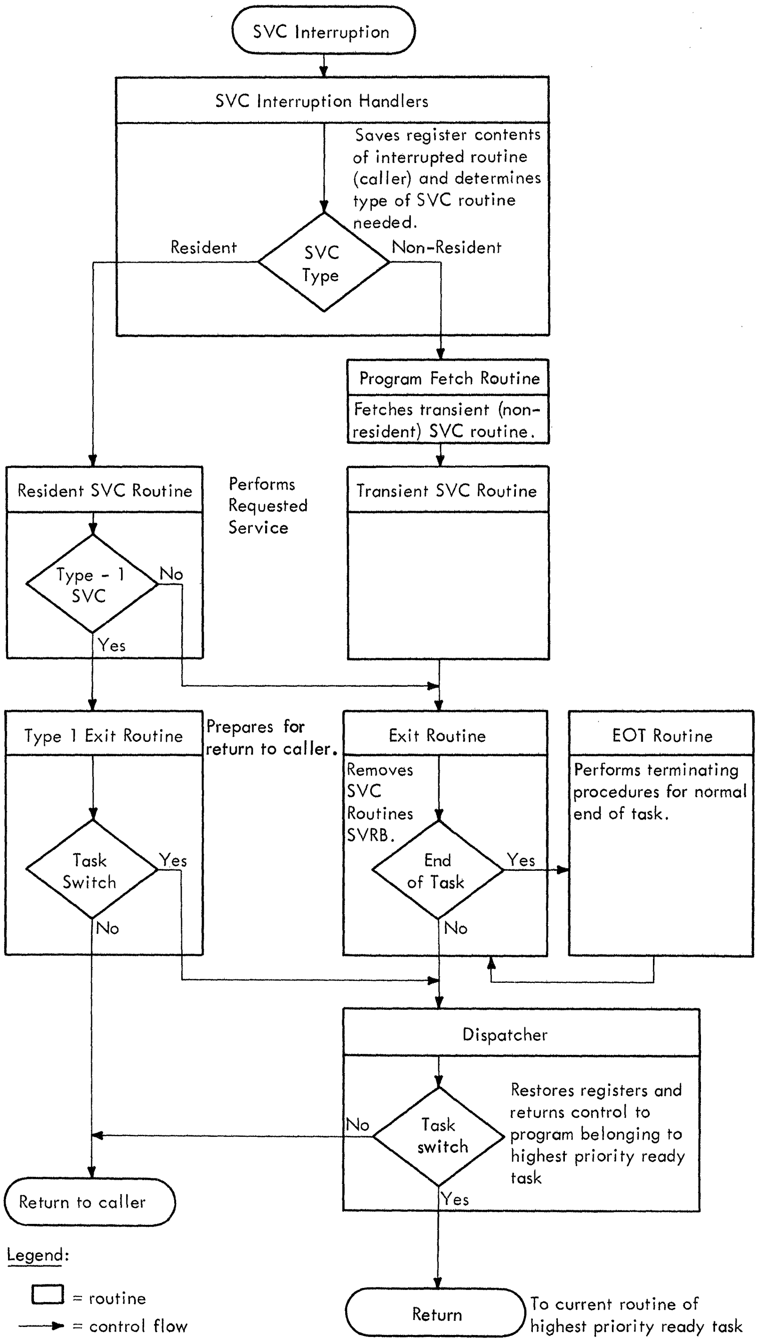

SVC Interruption
SVC Interruption Handlers
Saves register contents of interrupted routine (caller) and determines type of SVC routine needed.
Resident
SVC Type
Non-Resident
Program Fetch Routine
Fetches transient (non-resident) SVC routine.
Resident SVC Routine
Performs Requested Service
Transient SVC Routine
Type - 1 SVC
No
Yes
Type 1 Exit Routine
Prepares for return to caller.
Exit Routine
Removes SVC Routines SVRB.
EOT Routine
Performs terminating procedures for normal end of task.
Task Switch
Yes
No
End of Task
Yes
No
Dispatcher
Restores registers and returns control to program belonging to highest priority ready task
No
Task switch
Yes
Return to caller
Legend:
= routine
= control flow
Return
To current routine of highest priority ready task

Figure 1-3. Processing After an SVC Interruption

The supervisor handles all five types of interruptions.

- For SVC interruptions, the supervisor determines what SVC service is required, and routes control to the appropriate service routine.

- For timer/external interruptions, the supervisor determines the cause of the interruption and branches either to a timer service routine or to an external service routine.

- For input/output interruptions, the supervisor branches to the Input/Output Supervisor, which performs input/output error handling and services.

- For program interruptions, the supervisor either terminates the task in which the interruption occurred, or branches to a user error handling routine.

- For machine interruptions, the supervisor either places the machine in the wait state, or branches to an optional recovery management program.

This chapter contains a detailed description of all five types of interruption handling.

SVC INTERRUPTION HANDLING

When a system or user program issues a macro instruction, the last machine instruction of the resulting macro-expansion at execution time is often an SVC instruction. The SVC instruction causes the computer to produce an SVC interruption. The part of the supervisor that receives immediate control is called the SVC interruption handler.

Main Functions

The SVC interruption handlers perform the following main functions:

- Save the register contents and SVC old PSW for the interrupted calling program or routine.

- In a multiprocessing system, ensure that both CPUs do not perform disabled supervisor routines simultaneously.

- Determine whether a supervisor request block (SVRB) should be constructed to restart the needed SVC routine if it is interrupted or if it must wait.

- If necessary, construct an SVRB, place in it information about the routine, and queue the SVRB to the request block queue for the current task.

- Determine if the needed SVC routine is normally resident in main storage.

- Pass control to a resident SVC routine.

- Fetch a nonresident routine from auxiliary storage and prepare for the passing of control.

- Defer a request for a routine that cannot be fetched.

- When possible, restart deferred requests.

In addition, the SVC interruption handlers perform two minor functions. They place in the so-called "environmental" registers the addresses of three control blocks needed by all SVC routines -- the communications vector table, the current task control block, and the current request block. They also set up the return address to which the SVC routine will return control when it is complete.

The SVC interruption handlers are divided physically and functionally into two parts, the SVC First-Level Interruption Handler, abbreviated SVC FLIH, and the SVC Second-Level Interruption Handler, abbreviated SVC SLIH. The SVC First-Level Interruption Handler saves register contents and the SVC old PSW, and determines the type and location of the needed SVC routine. For certain commonly used SVC routines, the SVC FLIH also branches directly to the routine to begin its execution. For other SVC routines, further processing by the SVC SLIH is needed. This processing includes the construction of a supervisor request block (SVRB) to control an interruptable routine, and the fetching of the routine if it is nonresident.

Saving the Status of the Interrupted Calling Routine

The SVC interruption handlers must save the register contents and old PSW belonging to the calling routine. The purpose is to permit later return of control to the caller at its next executable instruction. The current PSW, containing the address of the next executable instruction, is stored by the machine in hexadecimal location 20 in lower main storage. (Refer to IBM System/360 Principles of Operation, section entitled "Interruptions.") The caller's register contents are saved by the SVC First-Level Interruption Handler in a special SVC save area in lower main storage, called IEASCSAV. Figure 2-1, parts A and B, show the saving of the caller's PSW and register contents, respectively, by the machine and by the SVC FLIH.

The caller's register contents and old PSW remain in lower main storage or are moved to other save areas, depending on the type of SVC routine that will be executed. If the routine is not allowed to issue, directly or indirectly, an SVC instruction, no SVC interruption can occur that would cause the saved status information to be overlaid. Accordingly, the caller's register contents and old PSW can safely remain in their lower main storage save areas. But if the needed SVC routine can cause a new SVC interruption, the status information can be overlaid and therefore must be moved to new save areas. Such overlaying of status information can occur if the SVC routine issues an SVC instruction, or a macro instruction that expands into an SVC instruction, or is interrupted and loses control tọ a routine of another task that issues an SVC instruction.

Ensuring That Both CPUs in a Multiprocessing System do not Perform Supervisor Routines Simultaneously

In a multiprocessing system, the SVC FLIH routine determines whether the second CPU is performing a disabled supervisor routine by testing the supervisor lock and CPU identity bytes in the multiprocessing CVT. If the lock byte is not set, neither CPU is performing a disabled supervisor

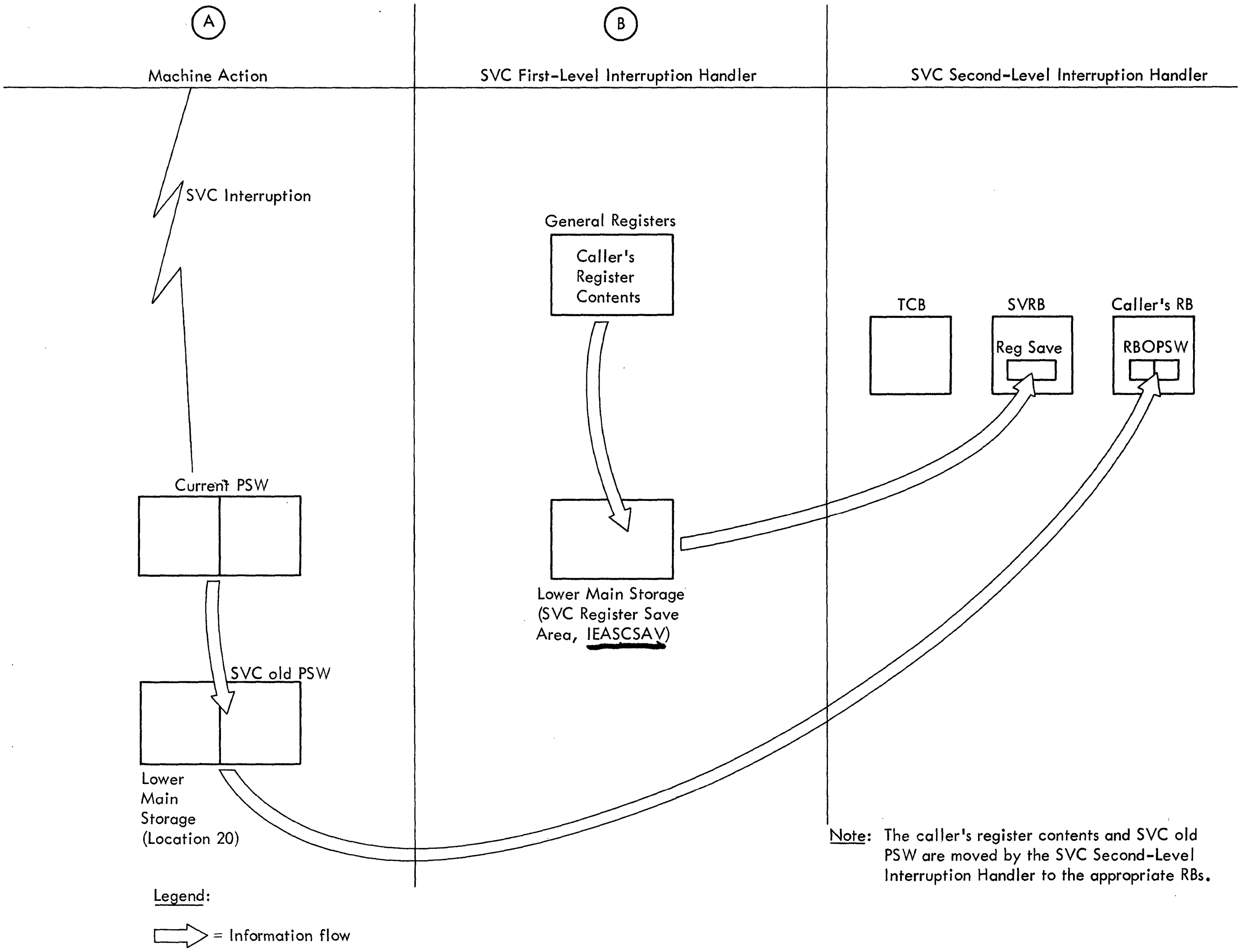

A
B
Machine Action
SVC First-Level Interruption Handler
SVC Second-Level Interruption Handler
SVC Interruption
General Registers
Caller's Register Contents
TCB
SVRB
Caller's RB
Reg Save
RBOPSW
Current PSW
Lower Main Storage (SVC Register Save Area, IEASCSAV)
SVC old PSW
Lower Main Storage (Location 20)
Note: The caller's register contents and SVC old PSW are moved by the SVC Second-Level Interruption Handler to the appropriate RBs.
Legend:
= Information flow

Figure 2-1. Status Saving by the SVC Interruption Handlers

routine. The SVC FLIH routine sets the supervisor lock byte and places the CPUID in the CPU identity byte before proceeding further. If the supervisor lock byte is set, the CPU identity byte is tested to determine which CPU set the byte. If set by the executing CPU, the SVC routine proceeds; if not, the SVC old PSW is set (backed up) to reissue the SVC[1] instruction.

If the SVC old PSW is enabled for external interruptions, the SVC[1] instruction is reissued via the Type-1 SVC Exit routine which restores registers and loads the SVC old PSW. The SVC[1] instruction is thus reexecuted until the second CPU releases the supervisor lock byte.

If the SVC old PSW is disabled for external interruptions, the executing CPU branches to the External FLIH routine so that external signals from the second CPU (such as a malfunction alert) can be received. The SVC[1] instruction will be reissued when the calling task is next dispatched. Before the External FLIH routine is entered, the status of the task that issued the SVC is saved. The registers are stored in the TCB, the current RB is set from the SVC old PSW to reissue the SVC instruction, the External FLIH bit is set in FLRETFLG to indicate that the registers have been saved, and the External old PSW is set equal to the SVC old PSW. The External FLIH routine tests the supervisor lock byte until it is reset by the second CPU. Between repeated tests of the lock byte, the CPU is enabled for external interruptions. After the supervisor lock byte has been reset and any external interruptions which may have occurred have been processed, the External FLIH routine branches to the Dispatcher. When the calling task is dispatched, it reissues the SVC[1] instruction.

Determining Whether a Supervisor Request Block is Needed

~~The decision to create a supervisor request block depends on whether the needed SVC routine can issue an SVC instruction~~. Certain commonly used resident SVC routines, called type-1 routines, are not permitted to issue an SVC instruction. Other types of routines are permitted to issue an SVC instruction. The SVC FLIH examines the SVC table to determine the type of routine that is needed.

There are two parts in the SVC table, one containing entries for IBM-supplied SVC routines, the other containing entries for user-supplied SVC routines. The number and type of routines specified in the two parts depends on the particular system that the user generates at system generation time. There is one entry for each SVC routine. Each entry contains descriptive information, including a code showing SVC type, and the main storage or disk address of the SVC routine.

Whether an SVRB need be constructed depends on the type of SVC routine. If the routine is a type-1, as indicated by a test of the SVC table entry, the SVC FLIH branches to the routine whose address is contained in the table entry. But if the routine is not a type-1, the SVC FLIH branches to the SVC SLIH to create a supervisor request block to hold the caller's register contents and to contain restart information for the SVC routine.

An SVC routine may be restarted after any one of the following conditions has stopped or delayed its execution:

- The routine issues an SVC instruction, thus causing an SVC interruption.
- The routine is not resident and cannot be loaded; its request must therefore be deferred.
- The routine is overlaid in a transient area block of main storage before it can be executed.
- The routine may request a resource that is not immediately available, and is therefore placed in a wait condition pending the availability of the resource.

In any of these cases, restart information -- old PSW, wait count, etc. -- is stored in the supervisor request block (SVRB) created for the routine by the SVC Second-Level Interruption Handler.

Constructing, Initializing, and Queuing the SVRB

If the test of the SVC table entry (see Section 12, "Control Blocks and Tables") indicates that the needed routine is not type 1, and therefore that additional processing is needed, the SVC SLIH constructs an SVRB from space it obtained the last time it was entered. The SVRB, when initialized with status and descriptive information about the SVC routine, will be queued to the request block queue belonging to the caller's TCB. If the SVC routine cannot be executed, or is interrupted by its issuance of an SVC, or cannot continue its execution, its entry point address or restart address, called the RB old PSW,

[1] Either an SVC instruction or an EXECUTE instruction that executes an SVC instruction.

will be stored in the SVRB by a supervisor routine.

It is necessary for the caller's register contents and SVC old PSW, stored in lower main storage by the SVC FLIH, to be moved to safer areas in two request blocks. Since a non-type-1 routine may issue an SVC, the register contents and old PSW previously saved may be overlaid by the register contents and old PSW of the calling SVC routine. To protect the original caller's status information, the SVC SLIH moves it from lower main storage to two request blocks, the caller's RB and the SVRB that is being constructed. As shown in Figure 2-1, the SVC old PSW is moved from lower main storage to the PSW save area in the caller's RB. Since the caller's RB may lack a register save area, the caller's register contents are moved to the save area of the SVRB. Hereafter, if the currently requested SVC routine itself causes a new SVC interruption, the SVC routine's register contents and old PSW may be saved in lower main storage with no harm to the original caller's status information.

The SVC SLIH queues the new SVRB to the head of the RB queue belonging to the current or caller's TCB. The order of request blocks on the RB queue determines the order in which the supervisor places into execution routines started or requested for the given task. The request block at the head of the RB queue represents the routine that is next to be executed for its task. The SVC routine represented by the SVRB, will be executed next for the current task. Then, when the supervisor's Exit routine has removed this SVRB from its RB queue, the new head or "current" RB will represent the interrupted routine or caller. The Dispatcher will then restore the caller to execution, providing that there is no other ready task of higher priority.

The SVC SLIH queues the SVRB to its RB queue by setting two pointers, one in the TCB, the other in the SVRB. The pointer[1] in the TCB points to the SVRB which is the "current" or head RB on the queue. The other pointer[2] in the SVRB points to the previously current RB, which represents the caller of the SVC routine. (Refer to Figure 2-2.)

The SLIH issues a GETMAIN macro instruction to obtain storage to be occupied by another SVRB the next time the SLIH is entered. The resulting SVC interruption produces linkage to the GETMAIN routine of main storage supervision, which allocates the requested space. Since the GETMAIN routine is a type-1, no SVRB is added to the current task's RB queue, but status information belonging to the SVC SLIH is stored in lower main storage, as explained previously.

So far the SLIH has saved registers and the SVC old PSW in the appropriate RBs, and queued the SVRB to the TCB and to the previously current RB. The SLIH next sets certain status bits in the SVRB. These status bits[3] flag the request block as an SVRB and indicate that the associated SVC routine is nonresident[4], even though a later test may prove that the routine is really resident. (The initial assumption is that the routine is nonresident.) The type of request block, as indicated by the status bits, determines the processing to be performed during the exiting procedure after the SVC routine has been executed.

Determining if the SVC Routine is Normally Resident in Main Storage

The SVC SLIH must determine whether the needed SVC routine is normally in main storage and may be reached via a branch, or whether the routine is normally nonresident

[1] In this chapter certain terms are accompanied by superscripts. The terms represent fields of control blocks that are defined in Table 2-1 at the end of "Interruption Handling."

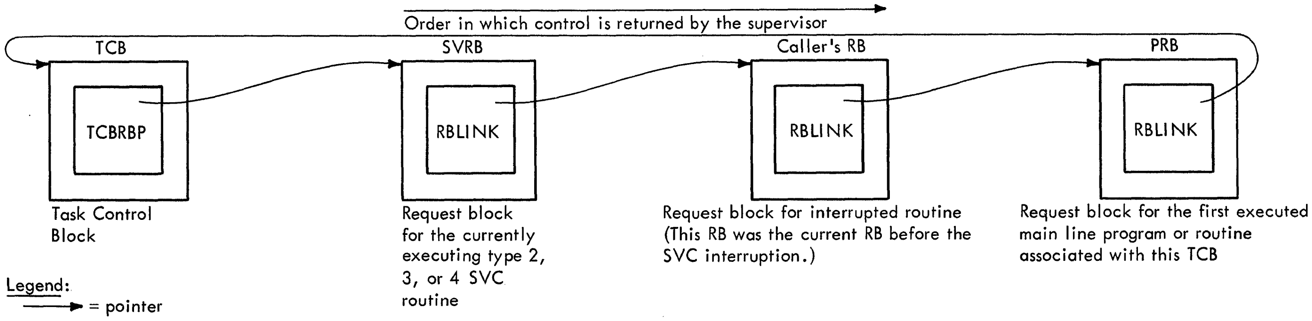

Order in which control is returned by the supervisor
TCB
SVRB
Caller's RB
PRB
TCBRBP
RBLINK
RBLINK
RBLINK
Task Control Block
Request block for the currently executing type 2, 3, or 4 SVC routine
Request block for interrupted routine (This RB was the current RB before the SVC interruption.)
Request block for the first executed main line program or routine associated with this TCB
Legend:
= pointer

Figure 2-2. A Request Block Queue

and should be fetched from the SVC library. The SLIH makes the determination by testing the "type" bits of the SVC table entry that was passed by the SVC FLIH. A type-2 routine is resident in the nucleus; a type-3 or 4 routine is nonresident and is located in the SVC library, unless it has been preloaded into main storage by the Nucleus Initialization Program at IPL time. If the test of the SVC table entry reveals that the needed routine is resident in the nucleus, the SVC SLIH prepares for entry to the routine. The preparation consists of: placing in the SVRB a "resident routine" flag[4] to later inform the supervisor Exit routine that it need not perform exit processing for a nonresident routine, the restoring of standard input registers that the SLIH has altered, and the loading of a return address for use by the SVC routine when its execution is complete. The SLIH then branches to the routine address contained in the SVC table entry. But if the test of the SVC table entry indicates a routine not resident in the nucleus, the SLIH branches to a part of its coding loosely termed the "transient area handler." The transient area handler's purpose is to determine the location of the routine and, if necessary, attempt to fetch the routine to a transient area block of main storage.

Fetching a Nonresident Routine From Auxiliary Storage

Introduction: The purpose of the transient area handler of the SVC SLIH is to monitor the transient areas of the nucleus and to fetch nonresident SVC routines from auxiliary storage. If the desired routine is not in one of the transient areas, the transient area handler fetches the routine from the SVC library (SYSRES volume) and gives the routine control. If there is no available space in one of the transient areas for the desired routine, the transient area handler makes the associated SVRB non-ready until space becomes available.

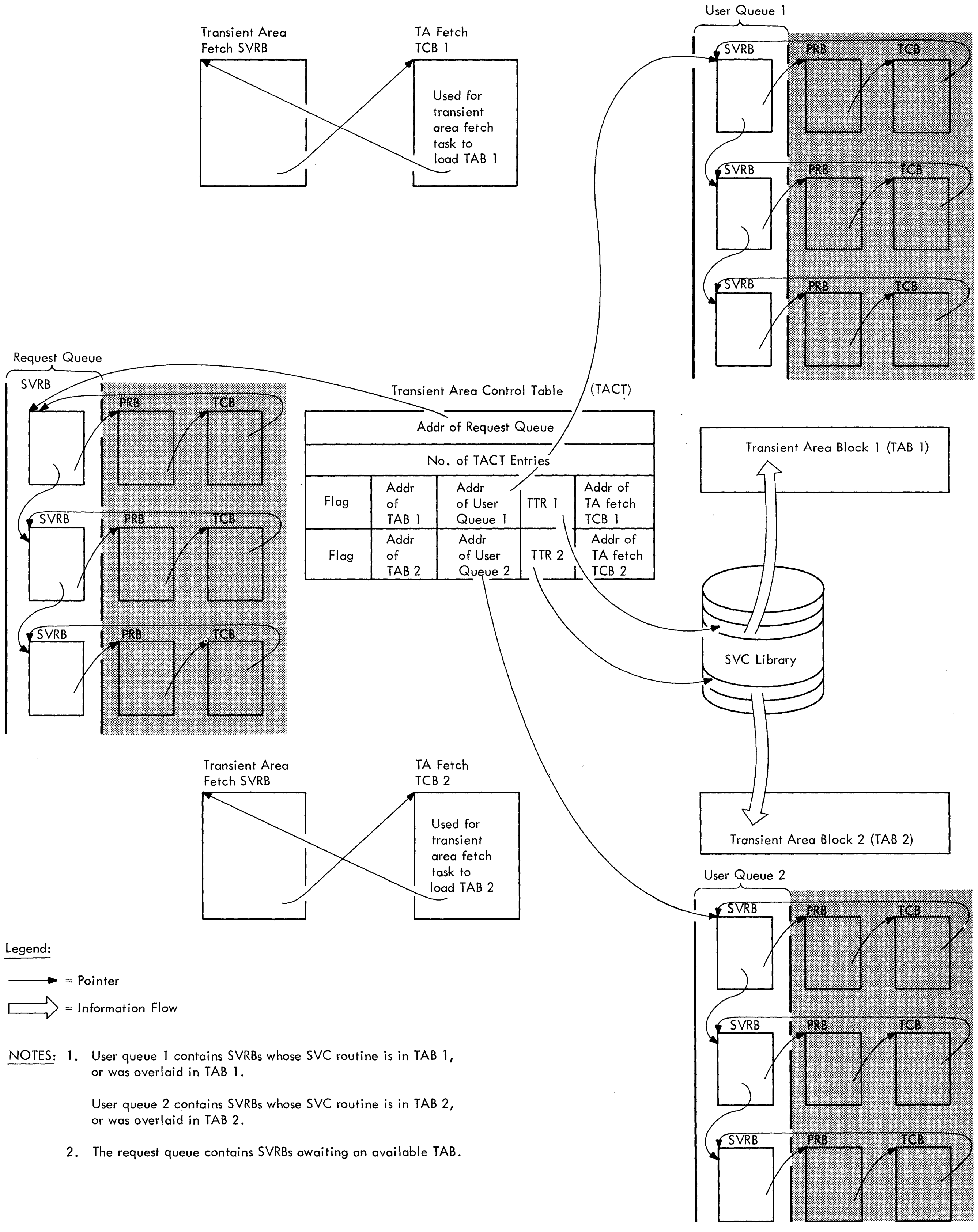
Determining if the Desired Routine is in the Link Pack Area of Main Storage: The transient area handler first determines if the desired SVC routine has been preloaded into the link pack area (LPA). The LPA contains reenterable modules from the link library and the SVC library that were preloaded at IPL time by the Nucleus Initialization Program (NIP). These modules remain resident until the next IPL.

The transient area handler determines if the requested SVC routine is in the link pack area by searching the contents directory entries[5] of the link pack area control queue for an entry which contains the name of the requested routine. The link pack area control queue contains entries describing all programs that reside in the link pack area of main storage. (See Section 12, "Control Blocks and Tables," for the format and content of a contents directory entry. For further information on the use of the contents directory, refer to Section 4, "Contents Supervision.") The name of the SVC routine used in the search is obtained from the interruption code that the machine stored in the SVC old PSW when the SVC instruction was executed. The old PSW now is in the request block belonging to the caller. The interruption code is converted to decimal and unpacked to provide a four-character value to be used as the right half of the name. The left half of the name is a constant value, IGC0.

Processing if the Routine is in the Link Pack Area: If the requested SVC routine is in the link pack area, the transient area handler finds a contents directory entry in the link pack area control queue that contains the routine's name[5]. The transient area handler then extracts the routine's entry point address[7] from the contents directory entry and branches to the routine in the same way as with a resident SVC routine.

Determining if the Routine is Already in a Transient Area Block of Main Storage: If the SVC routine is not in the link pack area, the transient area handler (location TARESTRT) first determines if the routine is already in a transient area block. If the routine is in a transient area block, it need not be fetched. There are at least two transient area blocks, each capable of containing one SVC routine. The number of transient area blocks, and thus the number of nonresident SVC routines that may be contained in the nucleus at one time, is specified during system generation. The transient area handler checks if the desired routine is in any of the transient area blocks by searching the entries of the transient area control table (TACT) for the routine name. (See Figure 2-3.) During its search, the transient area handler bypasses any TACT entry that indicates its transient area block is being loaded.

The TACT (see Section 12, "Control Blocks and Tables") contains one entry for each transient area block. Each entry contains four words: the address of the transient area block, the address of a "user queue" of SVRBs representing nonresident routines currently sharing a transient area, the relative track and record address (TTR) for the routine currently residing in the transient area block, and lastly the address of a TCB for a transient area fetch task (to be described later).



Addr of Request Queue				
No. of TACT Entries				
Flag	Addr of TAB 1	Addr of User Queue 1	TTR 1	Addr of TA fetch TCB 1
Flag	Addr of TAB 2	Addr of User Queue 2	TTR 2	Addr of TA fetch TCB 2

NOTES: 1. User queue 1 contains SVRBs whose SVC routine is in TAB 1, or was overlaid in TAB 1.

User queue 2 contains SVRBs whose SVC routine is in TAB 2, or was overlaid in TAB 2.

2. The request queue contains SVRBs awaiting an available TAB.

Figure 2-3. The Transient Area Queues

Processing if the Routine is Already in a Transient Area Block: If the transient area handler determines from its search of the TACT entries[8] and the user queues, that the desired routine is already in a transient area block, it performs as follows in order to bring the routine eventually under the control of the caller's TCB. The transient area handler queues the SVRB for the requested routine to the user queue for the transient area block that contains the routine. The SVRB is now a part of two queues, the RB queue belonging to the caller's TCB and the user queue (also called the transient queue) for a particular transient area block. Within the SVRB two different pointer fields are used for the two queues. The RBLINK field points to the next RB on the TCBs RB queue; the RBSVTQN field points to the next SVRB on the user queue.

Each user queue contains SVRBs whose routines are or have been in a particular transient area block. There is one user queue for each transient area block. Thus, if there are two transient area blocks, there are two user queues. The queues are built in the order in which routine requests are received. The requests are then serviced on a task priority basis. The purpose of each user queue is to permit the transient area handler to keep track of SVRBs whose routines are in a transient area block or have been overlaid in that block.

After queuing the SVRB to a user queue, the transient area handler sets up the address of the transient area block as an entry point for a branch to the SVC routine. It then loads the input registers, and branches to the transient area block to begin execution of the routine.

Processing if the Routine is not Already in a Transient Area Block: If the search of the TACT entries and their associated user queues indicates that the desired routine is not already in a transient area block, the transient area handler performs as follows. It rechecks the TACT entries[9] to find a transient area block that can be "used" (overlaid) by the requested routine. A transient area block can be "used" or overlaid in any of three cases: if it is "free," if all of the user SVRBs for the transient area block are not "ready," or if the caller's task is of higher priority than that of the tasks whose SVRBs are "using" the transient area.

A transient area is "free" if the routine residing therein is not being executed for any task. A "user" SVRB for a transient area is not "ready" if one or more nondispatchability bits[10] are set in its TCB, thus preventing the dispatching of the routine for this task. The user SVRB is also not ready if it is not the top RB in the RB queue of its task. The top RB is always pointed to directly by its TCB.

Preparation for the Overlaying of a Transient Area: If the transient area handler finds a transient area block that can be "used" or overlaid, it prepares for overlaying the area. It first places into a wait condition all using SVRBs whose routine is in the transient area block. It does this for each SVRB by saving the current wait count and setting the number 'FF' in the wait count field of the SVRB. This is done for each SVRB on the user queue whose TTR field[11] equals the TTR field[8] of the associated entry in the transient area control table. The saved wait counts, corrected for any intervening POST macro instructions, are later restored by the Transient Area Fetch routine during exiting procedures. (See "Loading the Routine.") If the routine in the located transient area block is not being executed, there are no using SVRBs. There is therefore no need to place them into a wait condition.

The transient area handler next prepares for the loading of the requested routine. It stores in the newly created SVRB the displacement[12] of the TACT entry, thus avoiding a new search of the TACT. It also stores in this SVRB an RB old PSW pointing to the transient area block. This PSW will later be used by the Dispatcher to begin execution of the routine. The transient area handler then sets up the input registers for the SVC routine and stores them in the caller's TCB[13], in preparation for later restoration by the Dispatcher when it causes entry to the desired routine. Pending the loading of the routine into the available transient area block, the transient area handler places the new SVRB into a wait condition (sets a wait count into its wait count field[14]). The wait condition prevents the Dispatcher from starting execution of the routine supposedly in the transient area block but not yet loaded. The transient area handler also queues the new SVRB to the user queue for the transient area block in order to keep track of the request for use of the block.

The transient area handler determines the address of the next TACT entry after the one for the transient area block (TAB) to be loaded. It saves the address in a word, called TACTNEXT, in the transient-area-handler module. The TACTNEXT location will be used to start the search for a TAB for the next-requested transient routine. TACTNEXT originally contained the address of the first TACT entry. Its contents are modified each time a TAB is loaded for a new SVC request or for an XCTL request issued by a type-4 SVC routine. A type-4 SVC routine is a nonresident routine that has more than one load module. It uses an XCTL macro instruction to cause linkage from one module to the next.

Next, the transient area handler indicates to the Dispatcher that a task switch must occur. This is necessary because the loading of the SVC routine will be performed under the control of a transient area fetch TCB to load the desired routine from the SVC library. Although there is only one Transient Area Fetch routine, it may operate under the control of any of several high-priority transient area fetch TCBs. There is one such permanent TCB for each transient area defined during system generation. The minimum number is two.

The task-switch indication to the Dispatcher is necessary because the Dispatcher cannot otherwise dispatch the routine for a task of higher priority than the current task. The transient area handler indicates the need for a task switch by placing the address of the transient area fetch TCB in a "new" TCB pointer[15] in the nucleus. The transient area handler then branches to the Dispatcher, which restores registers and gives control to the Transient Area Fetch routine (TAHFETCH) to load the desired SVC routine. During the loading process, although the transient area fetch task is of extremely high priority, other lower priority tasks can be performed while the Transient Area Fetch routine is waiting for the completion of an I/O operation.

Loading the Routine: When the Transient Area Fetch routine is given control, a transient area block is now "free" or able to be overlaid, as determined by previous processing. All SVRBs in the user queue for the transient area block are in a wait condition, including the SVRB which will soon control the awaited routine. The Transient Area Fetch routine, hereafter called the TA Fetch routine, extracts the relative disk address[11] and length[16] of the SVC routine from the caller's SVRB, and places them in the control area for use by the supervisor's Program Fetch routine. The control area consists of a work area, an input-output block (IOB), and a channel program. The Program Fetch routine converts the relative track and record address to an absolute disk address from which the SVC routine may be fetched. By use of the channel program, the Program Fetch routine transfers the desired routine from the SVC library to the available transient area block.

If no I/O error has occurred during the fetch process, the TA Fetch routine makes ready all user SVRBs representing requests for the loaded SVC routine. The purpose is to allow the Dispatcher to eventually place the routine into execution under the control of the user SVRB belonging to the highest priority ready task. In order to find all using SVRBs for the newly loaded routine, the TA Fetch routine searches the user queue belonging to the transient area block just loaded. The user SVRBs include both the SVRB associated with the current caller's task and SVRBs for other tasks. The copies of the SVC routine represented by the older SVRBs had previously been in execution and had been overlaid before their execution was complete. When the TA Fetch routine locates the using SVRBs for the currently loaded routine, it tries to remove the SVRBs from the wait condition. It does this by restoring the saved wait count, corrected for any POST macro instructions that have been executed while the SVRBs were waiting for the transient routine to be reloaded.

The TA Fetch routine next dequeues and makes ready each SVRB on the request queue. This action later permits the transient area handler to determine if these SVRBs, whose requests had previously been deferred, may now be serviced. That is, the SVC routine just loaded may be the routine needed for one or more of the deferred requests.

To prevent redispatching of the TA Fetch routine, the TA Fetch routine places its own SVRB in a wait condition. This action is necessary, since the transient area fetch tasks have the highest priority in the system. The Dispatcher is thus prevented at its next execution from redispatching the TA Fetch routine.

The TA Fetch routine then branches to the Dispatcher, which passes control to the current routine of the highest priority ready task. This routine is represented and controlled by the RB to which the TCB points, called the "current" RB. The SVC routine just loaded will receive control when one of its user SVRBs or a deferred-request SVRB, is the current RB for the highest priority ready task. The reader should note that the SVRB that controls the next execution of the loaded routine is not necessarily the SVRB most recently created. Task priority and readiness are the criteria that determine the order in which requests are serviced.

Deferring the Request

During its search of the TACT and the user queues, the transient area handler routine may find that there is no available transient area block. That is, all transient area blocks contain routines that are being executed; at least one user SVRB for each transient area block is ready; and the caller's task is not of higher priority than that of the tasks whose SVRBs are "using" the transient area blocks. With no transient area block available, the transient area handler defers the current request for the SVC routine. It does this by enqueuing the new SVRB to a special waiting queue called the request queue. The request queue is a queue of SVRBs that are waiting for a transient area block to become available. The request queue has a preassembled address called IEAQTAQ.

The reader should note that an SVRB in a user queue or in the request queue is also in the RB queue belonging to the TCB of the calling or interrupted program. However, the pointer field of the SVRB is different in the two cases.

The transient area handler defers the current request by queuing the new SVRB to the request queue* and by placing the SVRB into a wait condition (setting its RB wait count field greater than zero). It places in the SVRB an RB old PSW that points to a deferred-request restart point (TARESTRT) within the transient area handler. A branch is then made to the Dispatcher to give control to the current routine of the next highest priority ready task.

Restarting Deferred Requests

Deferred requests are restarted when the loading of an SVC routine is complete, or when the routine in a transient area block is no longer executed (becomes "free"). When either condition occurs, the SVRBs for deferred requests are dequeued from the request queue and their wait condition reset (each RB wait count field is cleared to zero). When one of the TCBs associated with the restarted SVRBs has the highest priority among the ready TCBs, the Dispatcher returns control to the deferred-request restart point to begin the search of the TACT entries. The first check determines if the requested routine is already in a transient area block. Thus, a restarted deferred request is processed exactly like an original request.

Minor Functions of SVC Interruption Handling

Besides the major functions already described, the SVC interruption handlers perform two minor functions. One function consists of making available the addresses of three important control blocks for use by other supervisor routines during later processing of the SVC interruption. The other function is the loading of the return register with the address of the appropriate exit routine so that the SVC routine, when complete, can begin the return of control to the caller by means of a simple branch, without the need for tests.

Making Available Control Block Addresses: The SVC First-Level Interruption Handler makes available to other supervisor routines the addresses of three important control blocks. It does this by placing in general registers 3, 4, and 5, respectively, the addresses of the communications vector table (CVT), the caller's TCB, and the current RB. The communications vector table contains addresses of resident control routines and addresses of certain tables. These addresses are used by nonresident SVC routines.

Preparing the Return Address for the SVC Routine: The return address is placed in the return register, general register 14, and depends on the type of SVC routine that will be executed. Type-1 routines, which are a commonly used non-SVC-issuing type, return control to the caller via the Type-1 Exit routine. Other types of routines return control via the Exit routine. Since most SVC routines are type-1, the SVC FLIH initially assumes that the needed routine is type-1, and places in the return register the address of the Type-1 Exit routine. If the FLIH later determines from the SVC table that the needed routine is not type-1, it branches to the SVC SLIH, which reloads the return register with a different return address appropriate for all other routines. In this case, the return address is the location of an SVC 3 instruction in the communication vector table. The SVC 3 instruction, when executed, causes a new SVC interruption and resultant linkage to the supervisor Exit routine. (For further information on the two supervisor exit routines, refer to Section 8, "Exiting Procedures.")

PROGRAM INTERRUPTIONS

If the program being executed attempts an invalid action, a program interruption occurs and a code describing the attempt is stored in the program OPSW. Invalid actions causing program interruptions include using incorrect addresses, issuing invalid operation codes, and attempting to execute privileged instructions. Additional causes of program interruptions are fixed-point overflow, decimal overflow, exponent underflow, and loss of significance; these events may be masked out.

The program interruption handler is automatically given control after any program interruption. It tests whether the interruption occurred in the supervisor or in user code, by examining the program

----

*The queuing field is RBSVTQN.

OPSW. If the interruption occurred in the supervisor, control is passed to the ABTERM Prologue routine, which schedules abnormal termination of the task being performed at the time of the interruption.

If the interruption occurred in user code, the supervisor tests the TCB for a program interruption element (PIE) address. A PIE is a control block associated with a user's error handling routine. If the user anticipates a program interruption and wishes to perform his own error handling, he issues a SPIE (set program interruption element) macro instruction. The SPIE service routine constructs a PIE and inserts its address in the TCB.

If the supervisor finds no PIE address, that means that the user does not wish to perform error handling; the ABTERM Prologue routine is entered, as above. If the supervisor finds a PIE address in the TCB, it tests the high-order bit in the address. This bit is set to one whenever control is given to the user's error handling routine; if the supervisor finds it on when handling a program interruption, then a second program interruption has occurred in the error routine, and the task must be terminated.

If a PIE exists and its high-order bit is zero, the supervisor tests whether the particular kind of program interruption that has occurred was specified by the user in the SPIE macro instruction. If it was, control is passed to the user's error handling routine. In a multiprocessing system, control is passed to the error routine via the Dispatcher, so that the error routine can be bypassed if the task has been set nondispatchable by the second CPU. If it was not, control is passed to the ABTERM Prologue routine which, with the ABTERM routine, schedules the abnormal termination of the task. The supervisor Exit routine is entered when a user error routine completes its processing, and the interrupted program, via the Dispatcher, regains control.

MULTIPROCESSING PROGRAM INTERRUPTION HANDLER

If the multiprocessing feature has been selected, and the system is in multisystem mode, the SSM instruction also causes a program interruption. The multiprocessing program interruption handler first determines if the interruption was caused by an SSM instruction. If it was, the system mask to be set is examined. If complete enablement is indicated, the supervisor lock and CPU identity bytes in the multiprocessing CVT are reset to zero only if originally set by the executing CPU; otherwise, they are allowed to remain set. Control is then returned to the interrupted program with the system mask enabled. If complete enablement is not indicated, the supervisor lock and CPU identity bytes are tested to determine if the second CPU is performing a disabled routine.

If the second CPU is not performing a disabled routine, the supervisor lock byte is set, the CPUID is placed in the CPU identity byte, and control is passed to the interrupted routine with the system mask set as indicated. If the second CPU is performing a disabled routine (supervisor lock byte set by second CPU), the program interruption old PSW is set (backed up) for reexecution of the SSM[1] instruction, and the system mask of the program interruption old PSW is examined.

If the program interruption old PSW is enabled for external interruptions, registers are restored, and the program interruption old PSW is loaded. The SSM[1] instruction is thus reexecuted until the second CPU releases the supervisor lock byte.

If the program interruption old PSW is disabled for external interruptions, the executing CPU branches to the External FLIH routine so that external signals from the second CPU (such as a Malfunction Alert) can be received. The SSM[1] instruction will be reissued when the calling task is next dispatched. Before the External FLIH routine is entered, the status of the task that issued the SSM instruction is saved. The registers are stored in the TCB, the current RB is set from the program interruption old PSW to reissue the SSM[1] instruction, the External FLIH bit is set in FLRETFLG to indicate that the registers have been saved and the External old PSW is set equal to the program interruption old PSW. The External FLIH routine tests the supervisor lock byte until it is reset by the second CPU. Between repeated tests of the lock byte the CPU is enabled for external interruptions. After the supervisor lock byte has been reset and any external interruptions which may have occurred have been processed, the External FLIH routine branches to the Dispatcher. When the calling task is dispatched, it reissues the SSM[1] instruction.

If the interruption was not caused by an SSM instruction, the program FLIH routine determines if the second CPU is performing a disabled supervisor routine by testing the lock and CPU identity bytes in the multiprocessing CVT. If the lock byte is

----

[1] Either on SSM instruction or an EXECUTE instruction that executes an SSM instruction.

not set, the program FLIH routine sets the lock byte, places the CPUID in the CPU identity byte, and the program FLIH routine proceeds. If the lock byte was set by the executing CPU, the program FLIH routine proceeds. If the lock byte was set by the second CPU, it is tested until reset. Between repeated tests of the lock byte, the executing CPU is enabled only for external interruptions by loading an enabled PSW so that the CPU can respond if the second CPU experiences a machine interruption and cannot reset the lock. Before the enabled PSW is loaded, a bit is set in a one-byte entry (FLRETFLG) in the Prefixed Storage Area (PSA) to indicate that, if an external interruption occurs, the External FLIH routine is to return control to the program FLIH routine. This bit is reset when the program FLIH routine is able to set the supervisor lock byte and proceed.

MODEL 91 PROGRAM INTERRUPTION HANDLER

For the Model 91, the program first-level interruption handler (PFLIH) routine has been expanded to recognize decimal instructions, the TESTRAN interpreter, and imprecise interruptions. Depending on options selected at system generation time, the PFLIH routine may also include one or more of the following sections:

- A section to handle interruptions due to the presence of a decimal instruction.

- A section to provide for return to the TESTRAN interpreter if necessary. In the case of multiple-imprecise interruptions, it is necessary that the SPIE macro instruction specify all possible types of interruption conditions.

Handling Decimal Instructions

On the Model 91, an operation-exception program interruption occurs when a decimal instruction is encountered in the execution of either a problem program or the TESTRAN interpreter. If the Decimal Simulator (IEAXDS00) routine has been included in the operating system at system generation time, the PFLIH routine gives control to the simulator to carry out the indicated operation.[1]

If an error condition arises during the instruction-processing operations of the Decimal Simulator routine, control is returned to the PFLIH routine for determination of how the condition is to be handled.

If the Decimal Simulator routine is not in the operating system when a decimal instruction interruption occurs, the PFLIH routine considers this to be an error condition and passes control to an appropriate error-handling routine (e.g., to a user exit, to the TESTRAN interpreter, or to a system task-terminating routine).

Entry From the Testran Interpreter

On the Model 91, when the TESTRAN interpreter is operating in either the 'trace' or the 'go-back' mode, it gives control to the PFLIH routine whenever a program interruption for a decimal instruction is encountered. Prior to giving control to the PFLIH routine, the TESTRAN interpreter sets a flag bit (the 'return-to-TESTRAN' flag) to indicate that the interpreter is in use. The action that is taken by the PFLIH routine if the Decimal Simulator routine is not in the system has been described in the section, "Handling Decimal Instructions."

If, because of an error, the Decimal Simulator routine returns control to the PFLIH routine (and the TESTRAN interprete had caused the PFLIH routine to be entered), the 'return-to-TESTRAN' flag is checked to verify the presence of the interpreter in the system, and control is returned to the TESTRAN interpreter for the handling of the error condition.

EXTERNAL INTERRUPTIONS

External interruptions are handled differently in uniprocessing and multiprocessing systems. In a uniprocessing system, the External First-Level Interruption Handler (FLIH) receives control after an external interruption. In a multiprocessing system, if the two CPUs are operating in multisystem mode, the Second CPU Interruption Analysis routine receives control. Otherwise, the Second CPU Interruption Analysis routine is bypassed, and the External FLIH routine receives control directly.

UNIPROCESSING SYSTEM

In a uniprocessing system, the External FLIH routine saves the registers in the current TCB, saves the external old PSW in the current RB, and determines the cause of the interruption from the old PSW. Control

----

[1] A program interruption may be caused by an EXECUTE instruction that makes reference to a decimal instruction. If this is the case, the PFLIH routine constructs, in a work area, a decimal instruction that is equivalent to the original instruction as it would be seen by the hardware.

is passed to the Timer Second-Level Interruption Handler if it is a timer interruption, or to the resident External routine if it is an operator key interruption.

MULTIPROCESSING SYSTEM

The Second CPU Interruption Analysis routine determines if the interruption was caused by one of the following conditions in the second CPU which requires immediate processing:

- A machine check interruption
- An unrecoverable channel failure
- A request to halt I/O that was started on the first CPU

If a malfunction alert signal (issued to the other CPU when a machine check occurs) has caused the external interruption (determined from the external old PSW), the Second CPU Recovery Management System Interface routine is entered. This routine tests a recovery management "time-out" flag in the prefixed storage area (PSA) of the second CPU to determine whether both CPUs are malfunctioning. If so, this CPU enters the wait state. Otherwise, the recovery management "time-out" flag is set in the PSA of this CPU, and an External Start (via a write direct instruction) is issued to the second CPU, causing it to enter the Recovery Management Support (RMS) routine. Before the External Start is issued, the supervisor lock byte is set for the second CPU, so that supervisor routines may be performed on that CPU. While the second CPU performs the RMS routine, a completion flag (set when the second CPU completes RMS) is tested. If the completion flag has not been set after a period of testing, this CPU enters the wait state. If the completion flag has been set, but the second CPU is in the wait state, this CPU also enters the wait state with the same error code. If the second CPU is not in the wait state (that is, RMS was successfully completed) control is returned to the Second CPU Interruption Analysis routine.

If the external interruption was initiated by an RMS routine because of an unrecoverable channel failure on the second CPU (bit 3 of STMASK=1), the Second CPU Recovery Management System Interface routine is entered. This routine operates as described above except that (1) an External Start is not issued since the RMS routine is already in process, and (2) the supervisor lock byte is not set since the RMS routine has already set it.

If a routine on the second CPU requests a Halt I/O for I/O that was started on the first CPU, an external interruption is issued to the first CPU (via the First CPU Signal routine) with an indication in STMASK (BIT 7=1) that the Second CPU Halt I/O routine should be entered. The Second CPU Halt I/O routine scans the UCB table to find each device which has been flagged for this CPU to perform Halt I/O and branches to the resident IOS routine. When Halt I/O has been completed, the UCB flag is turned off and also the Halt I/O request flag in the second CPU's STMASK. Control is then returned to the Second CPU Analysis routine.

If there are any other external interruptions, the External FLIH routine receives control via a LOAD PSW instruction. Otherwise, control is returned to the interrupted program.

In addition to testing for operator key and timer interruptions, the External FLIH routine in a multiprocessing system processes external interruptions which (1) occur during FLIH supervisor lock-testing routines when the CPU is enabled for external interruptions and (2) are caused by the second CPU (via a WRITE DIRECT instruction).

The External FLIH routine first determines if a FLIH routine was interrupted by examining the PSA byte FLRETFLG. If a FLIH routine, other than External FLIH, was interrupted, registers are saved, and the interruption code is saved in a PSA byte RNEXCODE. Control is then returned to the interrupted FLIH routine. The I/O and Program Check FLIH routines exit to the Dispatcher which tests FLRETFLG for unprocessed external interruptions and, if there are any, gives control to the External FLIH routine. If (1) the External FLIH routine is entered because of an unprocessed external interruption or (2) if the External FLIH routine was in process at the time of interruption, the registers are not saved (having already been saved by External FLIH), and the supervisor lock byte is tested and set. If a FLIH routine was not interrupted by the external interruption, registers are saved before the supervisor lock byte is tested.

Prior to testing for timer, key or second CPU interruptions, the External FLIH routine tests the supervisor lock byte. If the lock byte is not already set, it is set, the CPUID is placed in the identity byte, and the FLIH routine continues. If the lock has been set by the executing CPU, the FLIH routine continues. If set by the second CPU, the lock byte is tested until it is reset by the second CPU. Before each test, the CPU is enabled for external interruptions, and a bit in FLRETFLG is set to indicate that the interruption occurred during the External FLIH routine.

The multiprocessing External FLIH routine also tests for external interruptions caused by the second CPU. The word STMASK in the PSA of the second CPU is examined, and control is given to the appropriate routine as follows:

Bit set to 1	Indication
1	Enter Dispatcher
16	QUIESCE
17	VARY CPU OFFLINE
24	Start I/O on Channel 0
25	Start I/O on Channel 1
26	Start I/O on Channel 2
27	Start I/O on Channel 3
28	Start I/O on Channel 4
29	Start I/O on Channel 5
30	Start I/O on Channel 6

In each case except VARY CPU OFFLINE, the STMASK bit is reset after execution of the appropriate routine, and the external FLIH is resumed. When all bits have been accounted for, control is passed to the Dispatcher.

INPUT/OUTPUT INTERRUPTIONS

The basic function of the supervisor in handling input/output interruptions is to branch to the Input/Output Supervisor. All input/output services and error handling are performed within the Input/Output Supervisor.

When an input/output interruption occurs, the Input/Output First-Level Interruption Handler is automatically entered. Because the system may become enabled for input/output interruptions during the interruption handling, the Input/Output First-Level Interruption Handler may be entered again before the completion of the original interruption. To identify such a second entry, the original entry sets the I/O switch (IORGSW), which is tested whenever the interruption handler is entered. Only the first entry causes register saving and other initializing instructions; subsequent entries bypass these functions.

In a multiprocessing system, the supervisor lock and CPU identity bytes are tested before the interruption is processed. If the lock byte is not already set, it is set by the executing CPU which also sets the CPU identity byte, and interruption processing continues. If it has already been set by the second CPU, it is tested until reset. Between repeated tests of the lock byte, the system is enabled for external interruptions by loading an enabled PSW. Before loading of the enabled PSW, a bit is set in FLRETFLG to indicate to the External FLIH that control is to be returned to the Input/Output FLIH routine. This bit is reset after the lock byte has been set.

Upon return from the Input/Output Supervisor, the pseudo-disable switch is tested. If off, control is passed to the Dispatcher. If on, registers are restored and control is returned to the interrupted routine by loading the input/output old PSW. In the multiprocessing system, zeros are placed in the lock and CPU identity bytes if the system mask of the input/output old PSW is completely enabled.

MACHINE INTERRUPTIONS

Any one of three optional machine check recovery programs and one optional channel error recovery program may be selected by the user, depending on the model of System/360, as follows:

- SER0 and SER1 routines are available for Models 40, 50, 65, and 75. The SER1 routine is available for Models 91 and 95.
- The Machine-Check Handler for Model 65 (MCH/65).
- The Channel-Check Handler for models using 2860/2870 channels (CCH). This program can be selected to accompany either of the SER routines or the Machine-Check Handler, or it may be selected alone.

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

- If no recovery program has been selected, the machine loads the machine check new PSW, and the CPU enters the wait state.[1]
- 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 environmental data. When its function is complete, the SER0 routine places the CPU in the wait state.[1]

---

[1] The operator may then load the System Environment Recording, Editing, and Printing (SEREP) program in order to format and print the CPU logout area. The SEREP programs are model-dependent stand-alone diagnostic programs available for the Model 30 and for each higher numbered model.

- 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 the resumption of processing, or places the CPU in the wait state.[1]

- If the Machine-Check Handler for Model 65 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.[1]

If the multiprocessing feature has been selected, and the system is operating in multisystem mode, a machine interruption causes a malfunction alert signal to be sent to the second CPU. This CPU enters the wait state via the machine check new PSA. When the malfunction alert signal is received by the second CPU, the Second CPU Recovery Management System Interface routine (See "External Interruptions") receives control on the second CPU and issues an External Start to this CPU. The External Start causes this CPU to enter the Recovery Management Support routine.

When a channel failure occurs, the I/O Supervisor passes control to the selected recovery program. Processing varies, depending on the recovery option that the user has selected, as follows:

- If no recovery program has been selected, the I/O Supervisor loads the machine check new PSW, and the CPU enters the wait state.[1]

- If a SER routine or the Machine-Check Handler has been selected, the I/O Supervisor loads the machine check new PSW and control is given to the selected recovery program. The selected program records environmental data and then places the CPU in the wait state.[1]

- If the Channel-Check Handler for models using the 2860/2870 channels has been selected, the I/O Supervisor branches to it for possible recovery from the channel error condition. This program performs two main functions. It places information in the error routine interface bytes so that the appropriate device error routine may retry the channel operation that was being performed when the channel check occurred. It also records environmental data regarding the channel failure. This data is later written to the SYS1. LOGREC data set by a routine of the I/O Supervisor. (For a full description of the Channel-Check Handler (CCH), refer to the publication IBM System/360 Operating System: Input/Output Supervisor, Program Logic Manual, Form Y28-6616.)

SYSTEM ENVIRONMENT RECORDING

System Environment Recording (SER) is a set of control program routines which record hardware malfunctions of the Central Processing Unit and channels. There are two versions of SER, called SER0 and SER1. At system generation the user may select one of these two versions. If he selects neither, the default option is used. The version which is used as the default option depends on the model (or models) specified and the size of the system (see IBM System/360 Operating System: System Generation, Form C28-6554).

When a machine check interruption occurs control is given to SER if that is the recovery option selected. SER may also be entered by the SER interface of the I/O supervisor if a channel error occurs.

The less complex version of system environment recording, SER0, determines the type of malfunction and, if possible, writes out a record describing the error on a data set called SYS1.LOGREC. This data set resides on the primary system residence volume. If SER0 cannot write the record, the CPU is placed in a wait state and a message is printed to the operator to use SEREP. If the recording is partially or fully completed, the CPU is placed in a wait state and a message is printed to the operator requesting him to reload the Operating System.

The more complex version of System Environment Recording, SER1, also collects and writes out hardware environment data, but in addition, it performs selective termination analysis which attempts to associate the error with a specific task. If the error can be associated with a specific task and if the control program has not been damaged by the error, the task is terminated abnormally; if not, the CPU is placed in the wait state.

When the SYS1.LOGREC data set has been filled, the operator runs the environment recording edit and print (EREP) routine. This routine formats and writes the records placed on SYS1.LOGREC by SER onto printer, tape, or disk according to user specifications. EREP is described in the IBM System/360 Operating System: Utilities, Program Logic Manual, Form Y28-6614.

SERO

SERO collects, formats, and writes error information resulting from a machine check or from a channel error. The program is divided into two modules: the load nucleus resident module IFBSR000, and the link library resident module, IFBSR0XX (where XX is the model number -- 40, 50, 65, or 75).

LOAD NUCLEUS RESIDENT MODULE -- IFBSR000: The resident portion of SERO is nonreusable and does not require Operating System/360 facilities. The primary functions of this module are to halt all I/O activity and to read the first text record of the nonresident portion of SERO into an area which begins 32 bytes past the nucleus.

If a machine check occurs, the resident module gains control directly from the machine-check new PSW. If a channel error is detected, the module is entered from the I/O supervisor which loads the machine-check new PSW.

This module saves information to be used later by the non-resident portion of SERO in a 22-byte field in lower storage. After it has halted I/O on all devices, the module reads the first 1024 bytes of IFBSR0XX into storage. If after ten retries, the resident module is not able to read IFBSR0XX into main storage, it sets up the IOS wait state code 000F0A and branches to the Bell Ring/Wait State module which sounds the console alarm and places the CPU in the wait state. The code 000F0A is displayed in the instruction counter.

LINK LIBRARY RESIDENT MODULE -- IFBSR0XX: Like IFBSR000, the IFBSR0XX module does not require any operating system facilities. There is an IFBSR0XX module for each System/360 Model; the appropriate module is selected at SYSGEN time.

After the module loads the remainder of itself into main storage, it checks location 50 to determine which type of error has occurred. This location is preassembled to X'FF'. If the error is a machine check, location 50 is overlaid by the machine-check old PSW; a channel error does not change location 50. Once the type of error is established, the routine sets up the appropriate kind of record entry in which to place information about the error.

The routine enables itself for machine check interruptions. If it is already collecting error data and receives a machine check interruption, the routine stops all data collection and writes out what it has accumulated up to that point. If a third error occurs, the routine cannot continue; it prints out an error message.

If IFBSR0XX was entered because of a machine check interruption, the general purpose 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.

The routine 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 up to ten busy I/O devices are collected. The routine then fills in a record with the program identification, day, and time. After examining the seek address obtained from the header record of the SYS1.LOGREC data set, the routine writes on that data set the record it has just created and an end-of-file record.

If the routine records a partial or complete error record, it informs the operator by printing a message or displaying a code in the instruction counter.

1. IFBF05W: No machine check interruptions occurred during the data collection phase of the routine and a complete record entry describing the error was placed on SYS1.LOGREC.

2. IFBF06W: A machine check interruption occurred during the data collection phase of the routine, but the attempt to place a partial data record on SYS1.LOGREC was successful.

3. The IOS display code 000F05 or 000F06 is set up and the routine branches to the Bell Ring/Wait State module. This indicates that the routine has completed its functions as described in either 1 or 2 above but was unable to print a message to the operator.

If the routine does not write an error record it issues one of the following messages identifying the error.

1. IFBF07S: Successive machine check errors have occurred during the data collection phase of the routine and the attempt to place a partial record on SYS1.LOGREC was not successful.

2. IFBF08S: Because of I/O errors, the data collected on the original error was not entered on SYS1.LOGREC.

3. IFBF09S: The SYS1.LOGREC data set was full or the safety byte in its header record was off.

4. IFBF0AS: The link library resident module, IFBSER00, could not be read into main storage.

SER1

Like SER0, SER1 collects, formats, and writes error information resulting from a machine check or a channel failure. SER1, unlike SER0, is a single, serially reusable module that resides in the nucleus. In addition to writing error records, it attempts to identify the error with a specific task. If a task/error relationship can be established, and if the control program is in no way damaged by the error, the task is terminated abnormally, but system operation continues. If, however, the error cannot be associated with a task, or if the control program is affected by the error, the system must be reloaded.

SER1 is entered in the same manner as the resident portion of SER0. It is entered as the result of either of the following errors:

1. A machine check interruption. (The machine-check new PSW points to SER1.)

2. A channel check (inboard). (IOS loads the machine-check new PSW.)

3. An external machine check interruption on the Model 91.

SER1 checks location 50 to determine which type of error occurred. Location 50 initially contains X'FF', which is overlaid by the machine-check old PSW if the error is a machine check. Location 50 is not changed if SER1 is entered because of a channel error.

SER1 gathers error data into either a machine-check record entry or a channel-check record entry and writes the record on SYS1.LOGREC. SER1 functions within the framework of the operating system; all I/O communication with the SYS1.LOGREC data set is via the EXCP macro instruction unless the control program was affected by the error. If the control program is damaged, SER1 uses its own I/O routines. The DEB and DCB required when EXCP is used reside in the nucleus and are opened at IPL time by the nucleus initialization program (NIP).

If SER1 is able to associate the error with a task and the control program has not been damaged, SER1 terminates that task by branching to the abnormal termination service routine, ABTERM. When control returns from ABTERM, SER1 re-initializes itself and branches to the dispatcher so that the system can continue.

If the SER1 routine determines that only a job step need be terminated, it performs the following processing: SER1 sets all TCBs in the system nondispatchable, except certain system TCBs, and sets the "system must complete" flag in the current TCB. The system tasks that remain dispatchable are: the communications task, the rollout/rollin task (if that feature is present), the system error task, and the transient area fetch task. The SER1 routine then halts all input/output activity associated with the current TCB. It writes the error environment data on the SYS1.LOGREC data set and writes an error message to the operator. The TCBs are then made dispatchable and the ABTERM routine is entered for the job step that was affected by the failure. When control returns from the ABTERM routine, the SER1 routine branches to the Dispatcher.

Thus, the requirements for system continuation are task/error relationship, a complete record of the error, and successful termination of the task. In the following cases, these requirements are not met, so the system must be reloaded.

1. Additional failures occur while SER1 is handling an error. Data collection on the original error stops, and SER1 attempts to write a partial record on SYS1.LOGREC. The partial record contains the information gathered up to the time the second error occurred.

2. A complete record was written, but the error could not be associated with a specific task.

3. A complete record was written, but the control program was affected by the error.

4. The control program was damaged by the error and a complete record could not be written.

In any of these cases, a message is printed on the primary output device instructing the operator to reload the operating system, and SER1 places the system in the wait state.

The Model 91 can be interrupted by a special machine check called an external machine check. The SER1 routine for the Model 91 is given control when an external machine check occurs. If the system mask in the machine-check old PSW is not all ones, the data (record) associated with the machine failure is saved in the SER1 buffer area, and an internal indicator that a record has been saved is set. Control is then returned to the point of interruption. The record that SER1 saves is recorded on the SYS1.LOGREC data set if the next SER1 entry is caused by either a channel failure or a CPU (normal machine check) failure.

- If this next entry is due to a channel failure, the channel-failure record and the saved external machine-failure record are processed as one record on the SYS1.LOGREC data set, and the operating system causes a termination of the problem program as for a channel failure.
- If, instead, the next entry is due to a CPU failure, the CPU-failure record is recorded first on the SYS1.LOGREC data set, and then the external machine-check record is recorded on the same data set. The normal SER1 techniques of handling a CPU failure are then performed. That is, either the task or the system is terminated.

However, if the next entry to SER1 is due to an external machine check, the data associated with the first failure is lost, and an entry is made in the count of the number of consecutive external machine checks experienced. If the count reaches a value of ten, a record with the count value is written on SYS1.LOGREC, and the system is terminated.

If the initial check of the system mask in the old PSW (see preceding) showed that the mask was all ones, the SER1 routine is placed in a waiting loop until all input/output interruptions in the system have been serviced. If a channel failure occurs, the SER1 routine processes both the channel failure and the external (I/O) failure as a single record and terminates the system in the manner normally done for channel failures. If there are no channel failures and all the I/O interruptions are taken care of, the SER1 routine processes the external machine-check record and the system continues from the point of interruption.

MACHINE-CHECK HANDLER FOR MODEL 65 (MCH/65)

This program consists of a resident routine, and transient modules which reside on the SYS1.SVCLIB data set. It attempts to recover from a machine check interruption. It first determines if the instruction that was being executed when the machine check occurred can be retried.

If instruction retry is possible, the Machine-Check Handler attempts reexecution of the interrupted instruction. If, however, instruction retry is not possible, it tries to repair program damage. The program damage may be associated with either a defective storage protection feature (SPF) key or a defective main storage location. The Machine-Check Handler may correct a defective SPF key by issuing a Set Storage Key (SSK) instruction. A main storage location can sometimes be corrected by reloading (refreshing) the module that was being executed when the machine check occurred.

If program damage can be repaired, the Machine-Check Handler attempts to retry the interrupted instruction. If the retry is successful, the Machine-Check Handler has recovered completely from the machine check interruption.

If program damage cannot be repaired or instruction retry is unsuccessful, the Machine-Check Handler can either continue partial system operation or place the CPU in the wait state. The choice depends on the type of task that was current at the time of the machine interruption, the number of tasks that are affected, and the extent of the program damage. If limited system operation is possible, it either abnormally terminates the current job step or sets the current task nondispatchable. If even limited system operation is not possible, because a critical system task is permanently damaged, the Machine-Check Handler issues an error message and places the CPU in the wait state. The operator may then load the SEREP program in order to format and print diagnostic information from the CPU logout area.

For a full description of the Machine-Check Handler for Model 65 (MCH/65) refer to the publication IBM System/360 Operating System: Machine-Check Handler, Program Logic Manual, Form Y27-7155.

Table 2-1. Control Block Fields Used by the SVC Second-Level Interruption Handler

Superscript	Name of Control Block	Formal Name of Field	Common Name of Field (if any)	Purpose of Field
1	TCB	TCBRBP		Points to current or top RB of RB queue.
2	any RB	RBLINK		Points to next RB on RB queue
3	any RB	RBFTP		Indicates type of RB.
4	SVRB	RBFNSVRB		Indicates whether SVRB is for a nonresident routine.
5	CDE	CDEPRGNM	Program name	Contains the 8-character name of program represented by the CDE.
6	any RB	RBOPSW	RB old PSW	The entry point or restart address for the routine controlled by the RB.
7	CDE	CDEEPADR		Entry point address of module represented by the CDE.
8	TACT	TTR		Relative track and record address of routine in a TAB. Used to identify the routine in the TAB.
9	TACT	FLAG		Indicates if TAB is being loaded or is "free".
10	TCB	TCBFLGS		Flags that indicate to the Dispatcher that it may not place into execution any routine for the task.
11	SVRB	RBSVTTR		Contains relative track and record address of routine.
12	SVRB	RBTABNO		Displacement of TACT entry within the TACT.
13	TCB	TCBGRS	general register save area	Save area for general registers in a TCB.
14	any RB	RBWCF	wait count field	When greater than zero, signals Dispatcher not to place into execution the routine controlled by the RB.
15	none	IEATCBP	"new" TCB pointer	Indicates to the Dispatcher the TCB whose current routine should next be dispatched.
16	any RB	RBSIZE	length field	Contains the size of the RB in double words.

Task Supervision consists of allocating a requested service for a particular task. Task Supervision may be divided into three categories: services directly related to a task control block, services indirectly related to a task control block, and services internal to the supervisor.

Services directly related to a task control block (TCB) involve the creation, manipulation, or elimination of a TCB. These services consist of:

- Attaching a subtask.
- Changing the dispatching priority of a task.
- Extracting information from a task control block.
- Detaching a subtask.

Services indirectly related to a task control block consist of:

- Specifying a program interruption exit routine.
- Synchronizing a program with one or more events.
- Serializing the use of a resource.
- Scheduling an asynchronous exit routine.

Services internal to the supervisor consist of:

- Testing and indicating the need for a task switch.
- Testing the validity of user-supplied addresses.

SERVICES DIRECTLY RELATED TO A TASK CONTROL BLOCK

The attaching of a subtask, requested via the ATTACH macro instruction, consists of the creation of a TCB to represent the subtask, the placing of control information in the new TCB, the allocation of main storage to the subtask, the placing of the new TCB on two TCB queues, and the scheduling of linkage to contents supervision to obtain the program to be first executed for the new task. When the new TCB is highest priority among the ready TCBs, the specified program is given control.

The caller may request, via the CHAP macro instruction, that the dispatching priority of its own TCB or of one of its subtask TCBs be changed. The dispatching priority determines the order in which the supervisor's Dispatcher places into execution routines for competing tasks. The CHAP routine computes a new dispatching priority, tests the legality of the result, places a dispatching priority value in the specified TCB, queues the TCB to a new position in the TCB queue, and tests whether the current routine of the priority-altered TCB should receive control in place of the caller.

Specified information may be extracted from a particular TCB. The TCB is the caller's or one of its subtask TCBs. The control information, obtained via the Extract routine, is placed in a table whose address is provided as an operand of the EXTRACT macro instruction.

The detaching of a subtask TCB from its parent TCB's subtask queue is the final step of normal or abnormal termination of the subtask. The Detach routine also frees storage areas belonging to the subtask. The storage areas include the subtask TCB itself and any associated problem-program register save area.

ATTACHING A SUBTASK

A user or system routine issues an ATTACH macro instruction to cause the supervisor to begin the execution of a specified program as a subtask of the caller's task. As a subtask, the specified program, and other programs later invoked via LINK and XCTL macro instructions, can compete for CPU time and can use resources already allocated to the caller's task. The ATTACH macro instruction, when executed as a macro-expansion, causes an SVC interruption. The interruption handling routines branch to the Attach SVC routine to perform the requested service.

The Attach routine performs the following main functions:

- Obtains storage space for a new TCB.
- Places in the new TCB information needed to control the subtask.
- Allocates to the subtask subpools of main storage belonging to its parent task.

- Places the address of the new TCB on two lists:

  - the subtask queue of its parent task
  - the TCB queue used by the Dispatcher.

- Schedules linkage to contents supervision to locate the first program to be executed for the new subtask, fetch the program if necessary, and schedule its execution.

While performing the main functions, the Attach routine also performs certain minor functions:

- If the ATTACH macro instruction contains the ETXR operand, storage space is obtained for one or two control blocks (IQE, IRB) to be used for the scheduling and controlling of an end-of-task exit routine.

- Places control information in these control blocks.

- Decides the task whose current program will next receive control from the Dispatcher: the parent task or the new subtask. In a multiprocessing system, the new subtask may receive control on the second CPU.

The TCB that is created will be initialized by the Attach routine to contain status information and list origins for queues needed by program being executed for the subtask. For example, the TCB will contain a pointer to the top RB on the RB queue, representing the currently executing program for this TCB. (See Section 12, "Control Blocks and Tables," for a detailed description of the TCB fields.)

Obtaining Storage Space

The Attach routine first tests the recursion bit in the TCBNSTAE field. If the recursion bit is on, an ATTACH macro instruction has been issued in the STAE exit routine. This is an invalid action, and a four is placed in register 15 to indicate that the ATTACH request has not been serviced. The Attach routine exits by returning control to the Dispatcher.

If the ATTACH was not issued by the STAE exit routine, the Attach routine next determines the amount of storage needed for the new task.

The storage must include space for the new TCB, and optionally space for one or two control blocks used to schedule and control an end-of-task exit routine for the subtask. The control blocks are an interruption request block (IRB), used to control the execution of the exit routine, and an interruption queue element (IQE), which helps schedule the execution of the routine. (See "Scheduling User Exit Routines.")

The amount of storage space that the Attach routine must allocate depends on two factors: whether the ETXR (end-of-task exit request) operand has been specified in the ATTACH macro instruction, and whether an interruption request block (IRB) already exists for the specified exit routine. If the ETXR operand is not specified, the Attach routine needs space only for a new TCB. For this purpose it issues a GETMAIN macro instruction for 196-bytes from subpool 253, supervisor queue space. If the ETXR operand has been specified and an interruption request block (IRB) already exits for the exit routine, space for a TCB and for an interruption queue element (IQE) -- a total of 208 bytes -- is similarly obtained from subpool 253. But if the ETXR operand has been specified, and an IRB does not already exist for the desired exit routine, the Attach routine obtains space for an IRB, a TCB, and an IQE. It does this by the issuance of a CIRB (construct IRB) macro instruction, which causes a branch to the supervisor's CIRB routine. The CIRB routine is also called stage one of the exit effector (refer to "Scheduling User Exit Routines.")

The Attach routine determines in the following manner whether an interruption request block (IRB) already exists, and therefore whether it should, via the CIRB routine, create a new IRB. (Refer to Figure 3-1.) The Attach routine searches the subtask queue belonging to the caller's TCB, looking for a subtask TCB which 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 zero, this subtask has no IQE, and thus no end-of-task exit routine has been requested for the subtask. The next subtask TCB on the subtask queue is then examined. If a subtask's TCBIQE field is not zero, 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 address specified in the caller's ATTACH macro instruction, an IRB for the desired exit routine already exists. In this case the Attach routine need not create a new IRB.

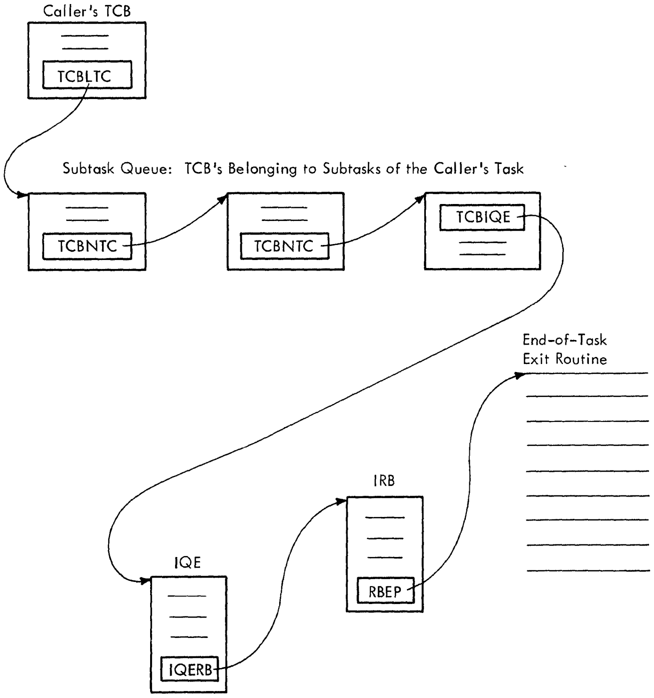


Figure 3-1. Queue Relationships Among a TCB, IQE, IRB, and End-of-Task Exit Routine

If the Attach routine finds that an IRB does not already exist for the specified exit routine, it issues a CIRB macro instruction to cause a branch to the CIRB (Construct IRB) routine. This routine obtains space for an IRB, initializes the IRB, and obtains a register save area for the end-of-task exit routine. Space for the new subtask's TCB and for the interruption queue element (IQE) is obtained as an extended save area belonging to the IRB. After control is returned to the Attach routine, it reduces the size of the IRB and uses the extended save area to build the TCB and the IQE.

Initializing the IQE, IRB, and TCB

If storage for an IQE was obtained, the Attach routine initializes the fields of the IQE as shown in Table 3-1.

Besides initializing the IQE, the Attach routine increases by a count of one a "use" count (RBUSE). This use count indicates the number of subtasks that use the same IRB to schedule and control an end-of-task exit routine. The supervisor Exit routine decreases the use count by a count of one each time that the end-of-task exit routine completes its execution. When the use count becomes zero, the supervisor Exit routine frees the storage space occupied by the IRB.

The Attach routine initializes the newly created subtask TCB by first clearing all areas except the register save area, then placing needed information in the TCB. If the ETXR operand was included in the caller's ATTACH macro instruction, 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.

Propagating Fields From the TCB of the Attaching Program

After initializing the newly created subtask TCB, the Attach routine transfers (propagates) from the caller's TCB to the new TCB certain fields that are the same in all TCBs within a job step. These fields include the pointer to the highest level TCB of the job step (TCBJSTCB), the storage protection key (TCBPKF), a pointer to the partition queue element (TCBPQE) (see Section 5, "Main Storage Supervision"), a pointer to the task I/O table (TCBTIO), and a pointer to the DCB for the job library (TCBJLB). There is, however, a limitation on the transferring of these fields. If the calling, or attaching, program is the Master Scheduler or an initiator, the

Table 3-1. Initialization of the Interruption Queue Element

Field Name	Type of Information in Field	Initialization of Field
IQELINK	Address of next IQE in a queue of IQE's	Zero
IQEPARAM	Parameter to be passed to the end-of-task exit routine	Address of newly created subtask TCB
IQEIRB	Address of the IRB	Address of the IRB just created, or the address of the IRB found during the search of the subtask queue
IQETCB	Address of TCB to which the IRB is to be queued	Address of caller's TCB

Attach routine places the address of the new TCB itself in the job-step TCB pointer, TCBJSTCB. (In this case, the "supervisor state" bit was set in the RB old PSW of the current RB.) The TCBJSTCB field in a job step TCB and in all higher level TCBs points to the TCB itself.

Note: For information on the assignment of a nonzero protection key for use by a job step, refer to "Task Creating Commands" in the MVT Job Management PLM.

Placing Parameter Information in the Fields of the Subtask TCB

After unchanged information from the caller's (attaching) TCB has been transferred to the new subtask TCB, information from the input parameters of the ATTACH macro instruction is placed in the subtask TCB. This information includes the supervisor mode bit, if needed; the address of an event control block (ECB), if specified; the limit and dispatching priorities for the new subtask; the "nonrolloutable count" field (TCBNROC); and the TCBFRA flag. (These last two items are initialized only if an initiator is attaching a job step task.)

The supervisor mode bit (TCBFSM) is set in the subtask TCB if two conditions exist: the caller's TCB is in supervisor mode, and a special input parameter has been provided by the caller. The supervisor mode bit, if set, later indicates to contents supervision that the RB old PSW for all programs operating under control of this TCB should be set in supervisor state.

If the "event control block" (ECB) parameter has been specified, the Attach routine checks the validity of the ECB address, and if valid, 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 abnormally terminates the caller's task by issuing an ABEND macro instruction with an error code of "42A". The ABEND macro instruction causes, via an SVC interruption, linkage to the ABEND SVC routine to abnormally terminate the task.

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 the input parameter 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 equal to the limit and dispatching priorities of the caller's task.

If the Attach routine determines that an initiator is attaching a job step, it indicates in the job step TCB whether the job step is eligible to be rolled out and whether it can cause rollout. If the ROLL parameter is ROLL=(NO,X), the Attach routine initializes to '01' the TCBNROC field. This marks the job step not eligible to be rolled out. If, however, the parameter is ROLL=(YES,X), the Attach routine initializes the TCBNROC field to '00'. This marks the job step eligible to be rolled out. (The TCBNROC field is later altered by the ENQ and DEQ routines to make the job step ineligible to be rolled out while one of its tasks is enqueued for a system resource.) If the parameter is ROLL=(X, YES), the Attach routine sets the TCBFRA flag in the new TCB to indicate that the job step is able to cause rollout. If, however, the parameter is ROLL=(X,NO), the TCBFRA flag is cleared to indicate that the job step cannot cause rollout. If the ROLL parameter is not specified, both the TCBNROC field and the TCBFRA flag are cleared to indicate that the new job step can be rolled out but cannot cause rollout. (The TCBFRA flag is later tested by the GETMAIN routine, if the new job step requests more storage space than can be allocated from its region and if the rollout feature is in the system. The TCBNROC field is later tested by the rollout/rollin module, during an attempted rollout, to determine if the new job step is eligible to be rolled out.)

Special Processing for Time Slicing

When the time-slicing feature is included in the system, the ATTACH routine tests whether the new TCB represents a time-sliced task. ATTACH locates the first time-slice control element (TSCE) through a pointer in the CVT, then compares the dispatching priority of the new task with that of each TSCE until a match is found or the last TSCE is checked. If no match is found, the new task is not a member of a time-sliced group and further time-slice processing is bypassed.

If a match is found, the new task is a member of a time-sliced group. The ATTACH routine sets the time-slice bit (TCBFTS) in the TCB and updates the TSCE pointers in the matched TSCE. If the new TCB represents the only task in the group, its address will be placed in the "First", "Last", and "Next to be Dispatched" fields of the TSCE. If the new task is not the only one in the group, its TCB is lowest on the TCB queue; the ATTACH routine places the address of the TCB in the Last field of the TSCE.

Allocating Subpools of Main Storage to the Subtask

The ATTACH routine allocates subpools of main storage to the attached TCB's programs according to parameters passed in the supervisor parameter list. These parameters were specified in the ATTACH macro instruction. The "give" parameter causes the allocation of specified subpools of main storage to the programs of the attached subtask for their exclusive use. The "share" parameter permits the programs of the subtask and the programs of the parent task to share access to the same subpools of main storage. The ATTACH routine manipulates ownership of the subpools by manipulating special Main Storage Supervisor queue elements, each representing a subpool of main storage. Each subpool queue element, originally queued to the parent TCB, may be either dequeued and placed on the subtask TCB's subpool queue ("give" parameter specified), or placed on the subtask TCB's subpool queue as duplicate queue elements ("share" parameter specified).

For each subpool specified in a "give" parameter, the Attach routine searches the subpool queue belonging to the caller's task. The queue starts at the address contained in the TCBMSS field of the caller's TCB. If a subpool queue element (SPQE) for the specified subpool is found on the queue, it is dequeued and placed on the new subtask's subpool queue. If the subpool queue element is not found, a new element for the subpool is created, placed on the subpool queue belonging to the subtask, and flagged as an "owned" subpool.

For each subpool specified in a "share" parameter, the Attach routine similarly searches the subpool queue chained from the parent, or caller's, TCB. In this case, however, if the subpool queue element is found, a new subpool queue element for the same subpool is created and placed on the subtask's subpool queue. The elements representing the same subpool (i.e., the original queue element and its duplicate) are both flagged as "shared" subpools. But if an original subpool queue element is not found on the parent task's subpool queue, two queue elements are created. One is flagged "owned" and "shared" and is queued to the parent task's subpool queue. The other element is flagged "shared" and is queued to the subtask's subpool queue.

There are two errors associated with the allocation of main storage to an attached subtask. Either error, when detected, causes the Attach routine to abnormally terminate the caller's task by issuing an ABEND macro instruction and specifying an error code. One error consists of the specification of the "give" or "share" parameter with a subpool number greater than 127, the maximum number for a subpool belonging to a user program. Such an error produces an abnormal termination of the caller's task and an error code of 22A. The other error occurs if the "give" parameter specifies a subpool whose queue element, when found, contains both the "owned" and "shared" attributes. In this case, the subpool cannot be "given" to the subtask. The resulting abnormal termination of the caller's task includes the error code 12A.

A special subpool of main storage, subpool zero, is processed separately. Subpool zero is always shared by all the tasks in a job step. Therefore, if subpool zero is specified with the "give" or "share" parameters, the specification is ignored.

The Attach routine begins the allocation of subpool zero to the subtask by testing whether the caller is the Master Scheduler. It checks the "supervisor mode" bit (TCBFSM) in the caller's TCB. The Master Scheduler cannot share subpool zero with its subtasks, because it has its own region of main storage, distinct from that of its subtasks, the Reader, Writer, Initiator, etc. If subpool zero were shared among the Master Scheduler and its subtasks, the subtasks could obtain storage from regions other than their own. If the "supervisor mode" bit is set in the caller's TCB, the Attach routine bypasses the sharing of subpool zero.

But if the caller is not the Master Scheduler or in supervisor mode, the Attach routine processes the allocation of subpool zero in the same manner as with any other subpool. That is, the subpool queue of the parent task is searched and the needed subpool queue elements are created and chained from the parent and/or the subtask TCB.

If subpool zero is specified, the Attach routine obtains, via the issuance of a GETMAIN macro instruction and resultant supervisor linkage to the GETMAIN routine, space for a register save area for use by the program specified in the ATTACH macro instruction. The address of the save area is stored in the subtask TCB. When the specified program has been fetched to main storage, the save area address will be passed to the program in general register 13.

Placing the Subtask TCB on Its Queues

The new TCB for the attached subtask is placed by the ATTACH routine on two TCB queues: the subtask queue for the parent TCB, and the main TCB queue. The subtask queue indicates the order in which TCBs were created, and is used by the supervisor ABEND routine during abnormal termination to establish the order in which a job step's resources are freed. The main TCB queue, or simply the TCB queue, is the queue of TCBs arranged in order of priority. This queue is manipulated by the CHAP SVC routine when it changes the dispatching priority of a TCB. The TCB queue is also used by the supervisor's Dispatcher routine when it tests which program should next be dispatched. The Dispatcher sometimes scans down this queue to determine the highest priority ready TCB. Both queues, the subtask queue for the parent TCB, and the TCB queue, consist of the same physical TCBs. The queues are created and manipulated by means of different sets of pointers within each TCB. (Refer to Section 12, "Control Blocks and Tables," to note the meaning of each pointer within the TCB).

Indicating to the Dispatcher the Need for a Task Switch

The Attach routine passes control to the Task Switch routine to determine if the parent TCB's current program or the subtask TCB's current program will receive control when the Dispatcher gives control to a main line program. The Task Switch routine examines the dispatching priorities of the two TCBs, parent and subtask, and stores the address of the higher priority TCB in a "new" TCB pointer (IEATCBP) to be later tested by the Dispatcher, when it receives control during the exiting procedure.

In a multiprocessing system, the parent and subtask TCBs may both have higher dispatching priorities than the current TCB on the second CPU so the Task Switching routine examines the dispatching priorities of three TCBs: the subtask TCB and the two "new" TCBs. If the "new" TCB pointer of either CPU contains zero, the Task Switch routine sets the "new" pointer of the other CPU to zero so the Dispatcher will search the TCB queue to determine the highest priority tasks.

Preparation for the Dispatching of the Caller and the Fetching of the Specified Program

To prepare for return of control to the caller, the Attach routine moves the caller's register contents, saved in the SVRB by the SVC Second Level Interruption Handler, to the save area of the caller's TCB. It also stores the address of the new subtask TCB in the register-1 save area location (TCBGRS) in the caller's TCB. These values will be loaded into the general registers by the Dispatcher when it next returns control to the caller, or attaching program. When the attaching program is redispatched, it regains control at the instruction immediately after the ATTACH macro instruction. The address of the new subtask TCB in register 1 is the return parameter from the Attach routine.

In order to obtain execution of the program specified in the ATTACH macro instruction, the Attach routine must obtain the assistance of one of the functions of contents supervision, called the Link function. The Link function will locate the desired program in main storage or in one of the auxiliary storage libraries, fetch the program to main storage, and cause linkage to the program for the newly created subtask.

The Attach routine determines the input register values for Contents Supervision and stores them in the new TCB. It also moves the entry point parameter -- either an entry point name or a partitioned data set directory entry -- from the input parameter list to the SVRB. The purpose is to prepare the SVRB for the control of Contents Supervision when it has been dispatched under the control of the new TCB. Another purpose of moving the entry point parameter is to permit the attaching program to reuse the parameter-list area if the program is redispatched before the Link function is complete.

The Attach routine schedules linkage to the Link function of contents supervision by dequeuing its SVRB from the caller's TCB and queuing it as the current RB for the new subtask. The Attach routine alters the old PSW in the SVRB so that it points to a special entry point in the Link function (IEAQCS01). The Link function is thus made the first routine to be executed for the newly created subtask when it becomes active.

By manipulating the register save areas and the TCB and RB queues, the Attach routine has effectively modified the two TCBs so that both are ready to be dispatched. The Attach routine branches directly to the Dispatcher to give control to the current routine of the higher priority of the two tasks, the attaching task or its newly created subtask. Note that the Attach routine branches directly to the Dispatcher, without the typical intermediate step of the supervisor Exit routine. The reason is that the Attach routine has already performed or made unnecessary the functions which the supervisor Exit routine normally performs. For example, the Exit routine normally removes the current RB from the TCB of the exiting program. Since the Attach routine has already removed its SVRB from the caller's TCB, it cannot branch to the Exit routine.

CHANGING THE PRIORITY OF A TASK

The CHAP SVC routine permits a problem or system program to alter the dispatching priority of its own TCB or the dispatching priority of one of its subtask TCBs. The subtask TCB must belong to the issuer's TCB; that is, be attached by a routine belonging to the caller's task, and therefore reside on its subtask queue. A program issuing the CHAP macro instruction may change the dispatching priority of a specified TCB to any value between zero and the limit priority of the issuer's TCB. The distinction between dispatching and limit priorities is as follows. Although both priorities are specified as parameters of the ATTACH macro instruction, they serve different functions. The dispatching priority determines the appropriate position of a TCB in the TCB queue, and indirectly the routine to be next placed in execution after an interruption. The Dispatcher places in execution the current program belonging to the ready TCB of highest dispatching priority. In contrast, the limit priority of a TCB is used by the CHAP SVC routine to determine the maximum value to which it may increase the dispatching priority of the TCB.

The CHAP routine can receive control as a type-1 SVC routine from the SVC FLIH, or serve as a subroutine via a branch entry from a supervisor routine. If it receives control through the branch entry, or if the caller is a system routine (protection key=0), the CHAP routine bypasses the usual validity checking of the input parameters. The assumption in this case is that the input parameters are valid and will not cause a program check when they are used by the CHAP routine.

After the CHAP routine has determined the type of requestor, it checks the input parameter for zero. If it is zero, the caller's TCB is the TCB whose dispatching priority is to be changed, and there is no parameter whose validity need be checked. But if the input parameter is not zero, it is the address of a word in main storage pointing to the TCB whose priority should be changed. In this case, the CHAP routine branches to a validity check subroutine used by many SVC routines to test an input parameter. The test determines if the parameter is a valid address and does not violate storage protection. If the address is not valid, the CHAP routine sets up an error code (22C), and branches to the ABTERM routine of the supervisor to schedule the abnormal termination of the caller's task. But if the address is valid, the CHAP routine determines if the specified TCB is a valid subtask TCB of the caller's task. It does this by searching the subtask queue of the caller's task, looking for the TCB address pointed to by the input parameter. The search is for a match between the address of the specified TCB and the address of one of the subtask TCBs. If the search does not produce a match, the assumption is that the input TCB was incorrectly specified. The CHAP routine in this case branches to the ABTERM routine with an error code (12C) to schedule an abnormal termination of the caller's task, since the requested service cannot be provided. But if the address of the specified TCB matches that of one of the TCBs on the subtask queue, or represents the issuer's TCB, validity checking is complete and normal processing continues.

If the time-slicing feature is included in the system, the CHAP routine tests whether the specified TCB represents a time-sliced task. CHAP does this by testing the time-slice bit (TCBFTS) in the TCB. If the bit is set, the task is time-sliced; the CHAP routine then resets the bit, and finds the time-slice control element (TSCE) that corresponds to the task's dispatching priority. If the address of the TCB is not the same as the First, Last, or Next fields in the TSCE, the new dispatching priority is determined; no change is required in the TSCE. (See Figure 3-2 for TSCE pointers.) If the address of the specified TCB appears as one of the above fields, the CHAP routine modifies the pointers as follows:

Field Containing TCB Address	Meaning and CHAP Processing
First, Last, and Next	Specified task is the only one in the group. CHAP sets all fields to zero to indicate the group is now empty.
First but not Last	Specified task is not the only one in the group. CHAP places address of next lower task on TCB queue into First.
Last and Next (not First)	Specified task is last in group and next to be dispatched. CHAP places address from First into Next and address of next higher TCB on TCB queue into Last.
Last but not Next	CHAP places address of next higher TCB on TCB queue into Last.

The remainder of the CHAP routine contains 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 zero or negative. In either case the

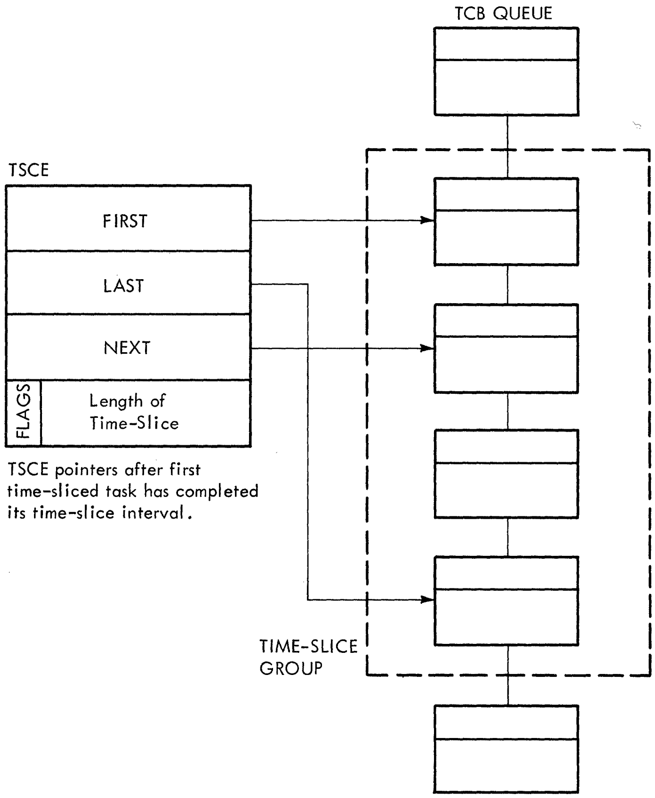

TCB QUEUE
TSCE
FIRST
LAST
NEXT
FLAGS
Length of Time-Slice
TSCE pointers after first time-sliced task has completed its time-slice interval.
TIME-SLICE GROUP

Figure 3-2. TSCE Pointers

CHAP routine sets the dispatching priority field (TCBDSP) of the specified TCB to zero. (A negative dispatching priority is meaningless and is treated as a request for a change to zero priority.)

The remaining tests will be discussed as separate cases, as follows:

Case 1: The result of the requested change would be a dispatching priority greater than zero, but equal to or less than the TCB. The CHAP routine algebraically adds the desired change to the original dispatching priority of the specified TCB and places the result in the dispatching priority field (TCBDSP) of the TCB. The request is thus satisfied.

Note: The remaining cases consider conditions in which the result of the change is greater than the limit priority (TCBLMP) of the specified TCB.

Case 2: The specified TCB represents a subtask of the issuer's TCB, and the desired change would make the dispatching priority of the subtask TCB greater than the limit priority of its parent (issuer's) TCB. In this case, the CHAP routine cannot quite satisfy the request. It sets both the dispatching priority (TCBDSP) and the limit priority (TCBLMP) of the specified TCB equal to the limit priority of the parent TCB. The request is thus satisfied within the limits of the system.

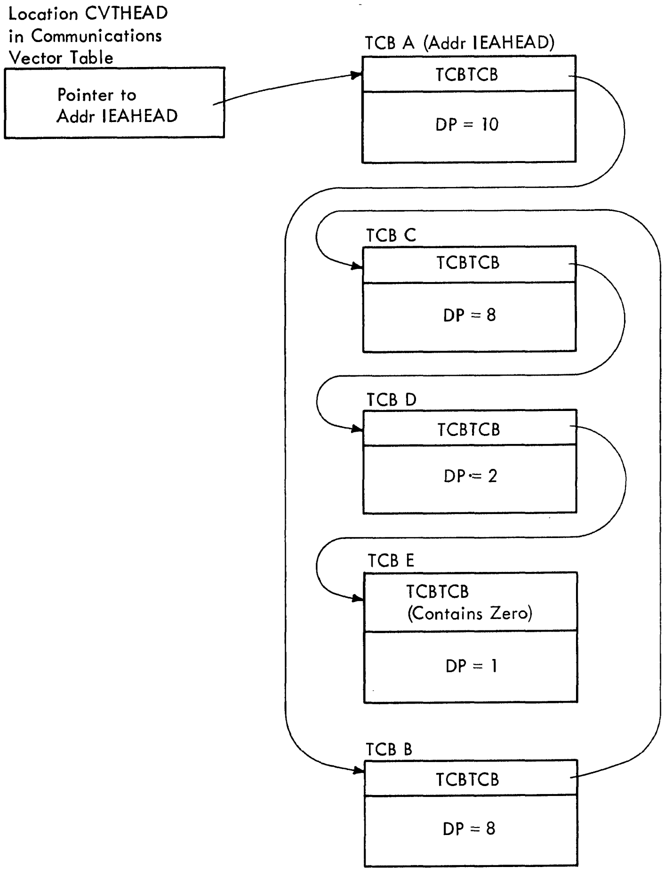
Case 3: The specified TCB represents a subtask of the issuer's TCB, and the desired change would not make the dispatching priority of the subtask TCB greater than the limit priority of its parent TCB. In this case the CHAP routine sets both the dispatching priority and limit priority of the specified TCB to the value produced by the change. This time the request can be completely satisfied without any compromises.

Case 4: The specified TCB is the issuer's TCB. In this case, since the result of the desired change would be a dispatching priority that exceeds the limit priority of the issuer's TCB, the request cannot be completely satisfied. The CHAP routine sets the dispatching priority of the issuer's TCB equal to its limit priority.

If the time-slicing feature is included in the system, the CHAP routine tests whether the new dispatching priority is time-sliced. If there is a TSCE for the priority, the CHAP routine sets the time-slice bit (TCBFTS) in the TCB. The CHAP routine tests the Next field in the TSCE; if it contains zero, the specified task is the only member of the time-sliced group, and the CHAP routine places its TCB address in the First, Next, and Last fields in the TSCE. Otherwise, the new TCB address is stored in the Last field.

Having changed the dispatching priority of the TCB, the CHAP routine must next realign the TCB queue so that it is ordered from high to low dispatching priority. This queue is sometimes used by the Dispatcher during the exiting procedure to determine the highest priority ready task whose current routine it should dispatch.

In order to reorder the TCB queue, the CHAP routine searches the TCB queue for two TCBs. One is the specified TCB whose dispatching priority it has just changed; the other is the first TCB that has a lower dispatching priority than the new priority of the specified TCB. The CHAP routine begins its search at the highest priority TCB, located at address IEAHEAD. (See Figure 3-3.) The address IEAHEAD is contained in a field (CVTHEAD) of the communication vector table. This table, also called the CVT, contains pointers to major control blocks used by the control program. Note on Figure 3-3 that the pointer (TCBTCB) in each TCB points to the next lower priority TCB on the queue. (Refer to the TCB description in Section 12, "Control Blocks and Tables," for the positions of the permanent system TCBs on the TCB queue.)



Legend: DP = dispatching priority value

Note: Each TCBTCB field points to the TCB of next lower dispatching priority.

Figure 3-3. The Task Control Block Queue

When the CHAP routine finds the two TCBs (the specified TCB and the next lower priority TCB) it rearranges pointers so that the specified TCB is removed from its current position on the queue and reinserted just above the next lower priority TCB. If other TCBs on the queue have a priority equal to the new dispatching priority of the specified TCB, it is placed below them on the queue.

During its search of the TCB queue, the CHAP routine branches to the Task Switching routine to determine if there is a ready TCB whose dispatching priority is now higher than that of the caller's TCB. This situation can occur in two different ways. The caller may have changed one of its subtasks to a priority higher than that of its own tasks, or the caller may have changed its own tasks to a lower priority. When the TCB queue is reordered, a TCB previously of lower priority than the caller's, now exceeds the caller's priority. In a multiprocessing system, there also may be a ready TCB whose dispatching priority is higher than that of the current task on the second CPU.

If the Task Switching routine finds a ready TCB with a higher dispatching priority, it indicates to the Dispatcher the need for a task switch. The indication, also performed by other supervisor routines, consists of storing the address of the higher priority TCB in a one-word "new" TCB pointer at address IEATCBP. During the exiting procedure that follows the execution of an SVC routine, the Dispatcher inspects the "new" TCB pointer to determine if it should redispatch the interrupted routine or the current routine belonging to another ready task.

After the CHAP routine has realigned the position of the specified TCB in the TCB queue, it returns control either to the caller or the current routine of another ready task. If the CHAP routine was entered via a branch from a supervisor routine, it returns control directly to the caller, deferring any indicated task switch to the next time the Dispatcher is entered. But if the CHAP routine was entered from the SVC FLIH, via an SVC interruption, it branches to the Type-1 Exit routine. The Type-1 Exit routine tests whether the CHAP routine has indicated the need for a task switch. If it has, the Type-1 Exit routine branches to the Dispatcher to give control to the current routine of the higher priority task. But if the need for a task switch has not been indicated, the Type-1 Exit routine returns control directly to the caller.

EXTRACTING INFORMATION FROM A TASK CONTROL BLOCK

The purpose of the Extract SVC routine is to permit the macro-issuing or calling program to obtain from a specified TCB 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 and Data Management Macro Instructions, under the heading of "Extract."

Besides extracting information from a specified TCB, the Extract routine 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. Input parameters, if incorrectly specified by the using program, cause the routine to generate an error code and cause an abnormal termination of the offending task.

Like certain other type-1 SVC routines, the Extract routine may be entered either from the SVC FLIH during an SVC interruption, or via a branch from a Supervisor routine. The routine first tests the 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 and precautionary measure for a subtask is bypassed.

If the specified TCB represents a subtask of the caller's TCB, the Extract routine 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. Its purpose is to determine if the specified TCB address truly 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, the Extract routine schedules an abnormal termination similar to that previously discussed. An error code (328) defining the incorrect address specification is passed to the ABTERM routine.

The Extract routine next determines whether it should check the validity of the input parameters supplied by the calling program. If the caller is a system routine, as indicated by a protection key of zero in the SVC old PSW, the assumption is that the caller has checked its parameters before passing them to the Extract routine. In this case no linkage to the Validity Check routine occurs, and the Extract routine immediately obtains the desired information. The Validity Check routine is used by SVC routines to check the validity of input parameters passed to the routines by a user program. If the caller is not a system routine, its input parameters must be checked. Accordingly, the Extract routine 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 listis 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 fail, indicating that the calling program has incorrectly specified the extract list address, the Extract routine branches to the ABTERM routine, passing to it an error code (128) indicating the type of incorrect specification. The ABTERM routine will schedule linkage to the ABEND routine, which will abnormally terminate the caller's task.

The validity checking detects an invalid list address that could cause a program check if it were used by the Extract routine. More important, the validity checking detects whether the caller has passed a list address pointing to a storage area which is not owned by the caller. Therefore, Extract can avoid storing into locations specified by the list. If the Extract routine used the list address without validity checking, it could store anywhere in storage, destroying data or programs belonging to another job step or to the supervisor. It could do this, since it operates with a storage protection key of zero. Note that validity checking does not prevent the caller from passing an invalid list address which causes the Extract routine to destroy the caller's data or program or the data or programs of another task in the caller's job step.

If input 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. The Extract routine tests each bit of an extraction byte, a part of the parameter list, which represents the FIELDS parameter of the EXTRACT macro instruction (see "Extract" in <u>Supervisor and Data Management Macro Instructions</u>). For each bit that is set, the Extract routine 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 <u>Supervisor and Data Management Macro Instructions</u>.)

Note that some of the fields that may be requested are not directly contained in the specified TCB. These fields are those requested by the following parameters: GRS (general register save area), FPS (floating-point register save area), and the ETXR (end-of-task exit routine). 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, ETXR, 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 two other control blocks, 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. (See Figure 3-1.)

When the Extract routine has placed all the requested information in the user-specified list, it either returns control directly to the caller, or prepares for a return to a program by branching to the Type-1 Exit routine. The Type-1 Exit routine, after making certain tests, will either return control directly to the caller, or branch to the Dispatcher to return control to the current routine belonging to another TCB. If, however, entry to the Extract routine occurred from a supervisor routine via a branch, the Extract routine returns control directly to the caller.

DETACHING A SUBTASK

The Detach SVC routine permits a program being executed for a "parent" task to detach its subtask if the subtask has been normally or abnormally terminated. The Detach routine checks that the address of the subtask's TCB passed to the Detach routine is valid, and that the subtask has been terminated. It dequeues the subtask TCB from the subtask queue of its parent TCB and frees storage areas belonging to the subtask, including the subtask TCB itself. If the caller specifies an invalid subtask TCB address, the Detach routine abnormally terminates the caller's task. But, if the subtask has not been normally or abnormally terminated previously, it is now abnormally terminated.

The Detach routine is entered from the SVC SLIH. To determine if the caller has passed a valid TCB pointer, the Detach routine first branches to the supervisor's Validity Check routine to test the supposed subtask TCB address. The Validity Check routine does not determine if the address belongs to a TCB but only that it will not later cause a program check. If any validity check fails, the check routine informs the Detach routine by supplying a return code. In this case, the Detach routine sets up an error code (0023E000) and issues an ABEND macro instruction to obtain supervisor linkage to the ABEND routine to abnormally terminate the caller's task. If the input address is valid, the Detach routine proceeds as follows.

The routine next determines if the caller belongs to the parent task of the specified subtask. It does this by searching the subtask queue of the caller's TCB for the specified TCB address. The list origin for the parent task's subtask queue is the TCBLTC field of the parent's TCB. If the subtask TCB address is not found,[1] the Detach routine sets up the same error code (0023E000) as that for an invalid TCB address, and obtains linkage, via an ABEND macro instruction, to the ABEND routine to abnormally terminate the caller's task. But 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 routine or the ABEND routine. The Detach routine makes this determination by testing the "completion" indicator TCBFC 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 not terminated, normally or abnormally, detaching cannot occur. In this case, the Detach routine performs processing to abnormally terminate the subtask. (This processing will be described later in this discussion.) But if the subtask has been terminated, normally or abnormally, the Detach routine proceeds with its processing, as follows.

Since the subtask is terminated, the routine must remove the subtask TCB from the subtask queue of the caller's TCB. This is necessary since the ABEND or EOT routines during a later termination of the caller's task will try to release resources supposedly belonging to its subtasks.

After removing the subtask TCB from its subtask queue, the Detach routine frees storage areas belonging to the subtask that were not freed during the termination process. These storage areas include a problem-program register save area, if the subtask has such an area, and the space occupied by the subtask's TCB. The save area consists of 72 bytes in subpool 250; the TCB contains 192 bytes in subpool 253, supervisor queue space. If the subtask has a problem-program register save area, its

----

[1] The subtask TCB is not found if neither an ECB nor an ETXR was specified when the subtask was attached, and the subtask has been terminated, normally or abnormally. In this case, the subtask TCB has been purged.

address is contained in the TCBFSA field of its TCB. After freeing the subtask's storage areas for reuse, the Detach routine returns control to the caller, via the Exit routine and the Dispatcher.

If the Detach routine discovers that the subtask was not normally or abnormally terminated, it initiates abnormal termination of the subtask, unless it is already being terminated. (The TCBFC flag, if set, indicates that the subtask has been normally or abnormally terminated.) If the subtask is not already being terminated, the Detach routine sets up an error code (13E), and branches to the ABTERM routine 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.

The Detach routine next performs processing whose purpose is to inform the Detach routine, and if possible a routine of the parent task, that the subtask has been abnormally terminated. The Detach routine first 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). The Detach routine then obtains four bytes of space (subpool 250) for a new ECB in which the ABEND routine[1] can post the subtask's termination. The Detach routine initializes the new ECB to zero and places its address in the subtask TCB (TCBECB field). The routine 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 (ETXR), if the caller specified the ETXR operand when it attached the subtask.)

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

If the subtask does not have an ECB which the Detach routine can post to inform the caller of the subtask termination, the Detach routine returns control to the caller. The return path includes the Exit routine and the Dispatcher.

If, however, the subtask has an ECB, the Detach routine checks the ECB in preparation for posting. It branches to a validity check subroutine belonging to the EOT routine. The subroutine checks whether 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 will cause a program check.

To avoid the possibility of a program check, the validity check subroutine examines the RB address supposedly contained in the ECB. If any of the tests fail, indicating that the ECB does not contain a valid RB address, the Detach routine sets up an error code (00202000) and invokes the ABEND routine to abnormally terminate the caller's task. A serious user error has been detected. If, however, the validity check suggests a valid ECB, the Detach routine posts the ECB, as an indication to the caller that the subtask has been abnormally terminated. Then, via the Exit routine and the Dispatcher, it returns control to the caller.

SERVICES INDIRECTLY RELATED TO A TASK CONTROL BLOCK

These varied services consist of:

- Specifying a program interruption exit routine.
- Sychronizing a program with one or more events.
- Serializing the use of a resource.
- Scheduling an asychronous exit routine.
- Specifying a task asynchronous exit routine.

A user program may specify a program interruption exit routine which will handle program interruptions occurring during any program executed for the user's task. The supervisor must be able to test for the existence of a user routine. The SPIE routine therefore places in the TCB of the macro-issuing program an indirect pointer to the user routine. If after a program interruption has occurred, the Program Interruption First-Level Interruption Han-

----

[1] The actual posting is performed for the EOT routine, which is invoked by the ABEND routine when the termination is complete.

dler finds an address in the pointer field, it passes control to the user routine to handle the interruption. Otherwise, the FLIH uses the ABTERM routine to schedule an abnormal termination of the task whose error caused the interruption.

By use of the Wait and Post routines, a user or system program may synchronize its execution with the occurrence of one or more events, such as the completion of an I/O operation. The Wait routine stops the execution of the requestor until the specified events have occurred. When they have occurred, the Post routine indicates their occurrence by altering bits in one or more event control blocks. It then makes ready the waiting requestor so that it may be placed into execution by the Dispatcher.

By serializing the use of resources, the ENQ and DEQ routines permit requestors representing different tasks to gain one-at-a-time access to a resource or set of resources. The resources may include one or more data sets, records within a data set, programs, or work areas within main storage. If the resource is available, control is returned to the requestor, optionally with a return code indicating the availability of the resource. If the resource is not available, either of two functions are performed, depending on the RET parameter that is supplied by the requestor. The requestor is placed in a wait condition, pending the availability of the resource, or control is returned to the requestor with a code indicating that the resource is not available. When a routine has issued a DEQ macro instruction to signal that it is no longer using the resource, the DEQ routine reduces the wait count of a waiting requestor and tests it for readiness. If the requestor is now ready, the DEQ routine determines if the requestor can be executed in place of the DEQ-issuing routine.

An asynchronous exit routine is scheduled by the supervisor to provide special handling of an unpredictable event, such as an end-of-task condition or the expiration of a timer interval. The scheduling of the exit routine, begun when the event actually occurs, is a multipart procedure interwoven with the performance of different tasks. Preparation for the event takes place when a system routine issues a CIRB macro instruction to cause the Exit Effector, stage 1, to construct an interruption request block or IRB. The IRB will control the future execution of the asynchronous exit routine when it is scheduled. When the unpredictable event occurs, the supervisor invokes stage 2 of the Exit Effector to begin the scheduling by placing an interruption queue element on a push-down exit queue. Final scheduling, performed by stage 3 of the Exit Effector, moves the interruption queue element to a queue belonging to the IRB. The IRB is then queued to the "head" position on the RB queue belonging to the requestor's TCB. When the TCB to which the IRB is queued is the highest priority ready TCB, the Dispatcher places the asynchronous exit routine in execution for its assigned task. When the asynchronous exit routine is finished, the supervisor's Exit routine removes the old scheduling and prepares for new scheduling. That is, it updates queue elements and prepares to queue the IRB to the RB queue of another TCB, if there are other requests for the exit routine. If there are no other requests, the supervisor's Exit routine dequeues the IRB from its TCB, and if the IRB was dynamically acquired, frees the storage space it occupies.

An asynchronous exit routine can also be specified to receive control when a task is scheduled for ABEND processing. The STAE macro instruction prepares the task to intercept abnormal termination processing through the STAE service routine, which receives control via an SVC 60 when the STAE macro instruction is issued. When the task has entered ABEND processing, the ABEND/STAE interface routine is invoked, which schedules a user-written STAE exit routine via the SYNCH macro instruction. If the STAE exit routine indicates that a retry routine should be scheduled, the ABEND/STAE interface routine sets the resume PSW to point to the address of the STAE retry routine. The ABEND/STAE interface routine then exits, giving control to the Dispatcher.

SPECIFYING A PROGRAM INTERRUPTION EXIT ROUTINE

Before reading the following discussion, the reader should carefully study "Program Interruption Processing" in Supervisor and Data Management Services.

The SPIE routine completes the processing needed for a user to specify a program interruption exit routine. The initial processing -- creating and initializing the fields of a program interruption control area (PICA) -- is performed by executable code produced by the expansion of the SPIE macro during an assembly of the source program. The execution of the instructions of the macro expansion places in the fields of the PICA a program mask, the address of the user program-interruption exit routine, and an interruption mask. If after the execution of the SPIE routine a program check occurs in a program being executed for the issuer's task, the information contained in the PICA will determine the resultant processing of the program interruption. In order for the supervisor to pass control to the correct error handling routine, the supervisor must be able to test for the existence of a user routine. The main function of the SPIE routine is to place in the TCB of the macro-issuing program an indirect pointer to the user routine. If after a program interruption has occurred, the supervisor finds an address in the pointer field, it will pass control to the user routine to handle the interruption. Otherwise, the supervisor's Program FLIH will schedule an abnormal termination of the task whose error caused the program interruption.

After the user program has issued a SPIE macro instruction, and the resulting macro expansion has constructed and initializes a PICA, an SVC interruption gives control to the supervisor. The First and Second-Level SVC Interruption Handlers pass control to the SPIE routine to complete the preparation for user processing of a possible program interruption. The SPIE routine first determines whether to create a program interruption element (PIE). The supervisor will store in the PIE, when a program interruption occurs, the information needed by a user-specified exit routine to handle the interruption. This information consists of the program check old PSW, general registers 14 through 2, and the address of the current PICA. The question of whether to construct a new PIE hinges on whether the current PICA is the first issued for the current task. Although there can be several PICAs, one for each issuance of the SPIE macro instruction for a given task, only the last specified PICA is active. The SPIE routine places the address of the newly created PICA in the PIE for the task. But the problem is first to determine if a PIE already exists for the current task.

The SPIE routine tests for the existence of a PIE by examining the PIE pointer (TCBPIE) in the current TCB. If there is no PIE for the task, the current SPIE macro instruction must be the first issued for this task. In this case, the routine issues a GETMAIN macro instruction for the needed storage[1] and places the address of the new PIE into the current TCB. The GETMAIN routine assigns to the storage area the task's storage protection key so that the user-specified program check routine, when given control, can modify the data stored in the PIE.

After locating or creating the PIE, the SPIE routine obtains the address of the previous PICA from the PIE. If the PIE is newly created this address is zero. The previous PICA address is returned to the caller in general register 1. If this register contains zero, no previous SPIE macro instruction was issued for the current task. The caller may use the old PICA address in a later SPIE macro instruction to restore to use the previous PICA.

The SPIE routine places in the PIE, whether newly created or old, the address of the new PICA that the macro expansion provided as input. The PICA with its user program-check routine address will then be available to the supervisor in the event of a program interruption. The PIE may already contain the address of a PICA, the one created by the last issuance of the SPIE macro instruction for the current task.

As a last major function, the routine moves the program mask field of the PICA to the RB old PSW. If the PICA address in the PIE is zero, the current program mask field of the RB old PSW is saved in the first byte of the TCBPIE field of the current TCB. The new program mask, supplied as an input parameter, is then placed in the RB old PSW. By placing the program mask in the RB old PSW, which the Dispatcher will use to return control to the caller, the SPIE routine is effectively issuing a Set Program Mask instruction for the caller.

Finally, to begin the exiting procedure that will complete the processing of the SVC interruption, the SPIE routine requests a supervisor-assisted linkage to the supervisor Exit routine. It obtains the linkage to the Exit routine by branching to an SVC 3 instruction in the communications vector table. The SVC-3 instruction causes an SVC interruption which ultimately passes control to the Exit routine.

If the PICA address provided as input is zero, the SPIE routine performs the previously described functions. However, since the PICA address stored in the PIE is zero, if a program interruption occurs, the Program-Check First-Level Interruption Handler recognizes that a user program-check routine has not been requested. It therefore branches to the ABTERM routine to schedule an abnormal termination of the task in which the program check occurred.

SYNCHRONIZING A PROGRAM WITH ONE OR MORE EVENTS

Synchronizing a program with external events consists of two actions:

1. Causing a program or routine to wait for one or more events.

----

[1] Space is allocated in subpool zero.

2. Indicating the occurrence of an event and restarting the waiting program or routine.

Causing a Program to Wait for One or More Events

The purpose of the Wait SVC routine is to permit a user or system program to stop its execution until a specified number of events have occurred, such as the completion of one or more I/O operations. When the specified events have occurred, the use of the Post SVC routine will indicate the occurrence of the awaited event or events and make the program ready (no longer waiting), so that its execution may continue.

The Wait routine performs the following main functions:

- Places the program that issued the WAIT macro instruction into a wait condition so that it cannot be executed until the awaited event or events have occurred.
- Recognizes those events that have already occurred and reduces the number of awaited events accordingly.
- Places in one or more special communications areas, called event control blocks (ECBs), an indication that one or more events are awaited by the issuing program. Each ECB represents a unique event that is awaited.
- Performs job step wait limit timing for the step under examination.

Like other type-1 (resident and nonreentrant) SVC routines, the Wait routine is entered from the SVC FLIH after an SVC interruption. The Wait routine first sets the system mask field of the SVC old PSW to all ones. It does this so that when the SVC old PSW is loaded to redispatch the caller, the caller will be enabled for I/O and external interruptions. This is done to prevent those supervisor routines that operate disabled and use the Wait routine from placing the caller into a disabled wait state.

The Wait routine then checks whether a wait count has been specified as an input parameter. The wait count, or number of awaited events, must be specified as an operand of the WAIT macro instruction. If no wait count has been specified, as indicated by a test of register 1, the Wait routine ignores the request represented by the macro instruction and branches to the Type-1 Exit routine to return control to the caller or macro-issuing program. If a wait count has been specified, the Wait routine continues normal processing.

The Wait routine next compares the number of awaited events, represented by the wait count, with the number of event control blocks (ECBs) that the caller has specified. The caller has passed to the Wait routine, via the coding of the macro expansion, the address of either a single event control block (for a single awaited event) or the address of a list of event control blocks if it awaits more than one event. The Wait routine checks the validity of the list address and then counts the number of specified ECBs. If the caller has specified a larger wait count than the number of ECBs, the WAIT request cannot be processed. The caller has made a serious error. In this case, the routine sets up an error code of 101 and branches to the ABTERM routine in order to schedule an abnormal termination of the calling task. If the number of awaited events, as indicated by the wait count, is equal to the number of specified ECBs, the Wait routine can perform the next main step of its processing -- determining whether to test the validity of the input parameters passed by the caller. But if the wait count is less than the number of specified ECBs, the routine sets a "search" flag in the request block (RB) of the caller.

The reason for the setting of the search flag (RBECBWT bit in RBSTAB field) in the RB of the caller is as follows. The calling program has specified a smaller wait count than the number of ECBs. This means that the caller awaits fewer events than the maximum number that can occur. For example, the caller may await the completion of any one of three possible I/O operations. In this case, the wait count would be one, and the number of ECBs would be three. When an awaited event (in this example, a single I/O completion) has occurred, the Post SVC routine will post the event in the ECB specified by the caller of the Post routine. Part of the posting action consists of clearing the wait bit that was set previously by the Wait routine. Since the WAIT request has now been fulfilled, that is, the single awaited completion of three possible I/O operations has occurred, the wait bit remaining set in each of the two ECBs not yet posted is now misleading, and may cause later incorrect processing by the Post routine. The Post routine will examine the search bit in the RB of the waiting program. If the search bit (RBECBWT) is set, the Post routine will clear the wait bit in each of the ECBs not yet posted and will also clear the search bit. The misleading indication is thus removed.

After the Wait routine has set the search bit (RBECBWT) in the caller's RB (or if this step was bypassed because the wait count equals the number of ECBs), the routine decides whether to check the validity of an input parameter passed to the Wait routine by the caller. This parameter is either the address of a single ECB, or of more than one ECB if a list of ECBs had been passed. (Refer to the WAIT macro instruction in Supervisor and Data Management Macro Instructions.) If a system routine is the caller, as determined by a zero in the protection key field of the SVC old PSW, the assumption is that the ECB addresses are correct and need no validity checking. But if the nonzero protection key indicates that a user program is the caller, the Wait routine decides to branch to the supervisor's Validity Check routine to test each ECB address that the user has specified.

The Validity Check routine, as indicated previously, performs three checks of each input address. It determines if the address lies on a fullword boundary, exists within the boundaries of main storage, and designates a storage area whose storage protection key matches the protection key in the caller's TCB. If any of these tests fail, indicating that the caller has incorrectly specified an ECB address, the Wait routine sets up an error code and exits to the ABTERM routine to schedule an abnormal termination of the caller's task. If all ECB addresses passed to the Wait routine are valid, or if validity checking is bypassed (caller is a system routine), the Wait routine continues processing.

The routine next tests bits in each specified ECB. (The reader should refer to Section 12 of this manual to observe the format of an ECB.) Wait tests the wait bit and completion bit in each ECB to determine the status of the event represented by the ECB. For each status the processing is different. If the wait bit is already set in any specified ECB, an error condition exists. One possible cause of such an error condition is that two programs being executed under the control of two different TCBs have specified the same ECB as an operand (i.e., the two programs are awaiting the identical event). If a wait bit is already set in one of the ECBs, the Wait routine sets up an error code (301) and branches to the ABTERM routine to schedule an abnormal termination of the caller's task.

If the wait bit is not already set in an ECB, Wait examines the completion bit to determine if the event that the caller is now awaiting has already occurred. A completion bit that is set indicates that the awaited event represented by the ECB has already occurred and has been posted by the Post routine. In this case, the Wait routine reduces by a count of one the specified wait count. This is necessary because the caller should wait only for those events that have not yet occurred. When the routine has subtracted one from the wait count, it tests the remainder to determine if the wait count has been reduced to zero. If the wait count is now zero, all awaited events have occurred (such as one I/O completion out of a possible three completions), and the Wait routine must perform special processing. If a completion bit is not set (meaning that the Post SVC routine has not posted this event's occurrence), the Wait routine sets the wait bit in the ECB (indicating that the awaited event has not yet occurred) and places in the ECB the address of the call- er's RB. This RB address is needed by the Post routine after an awaited event has occurred, when it wishes to adjust the wait count stored in the RB of the waiting program.

It was previously stated that for each completion bit that the Wait routine finds set in an ECB, it subtracts one from the specified wait count parameter. If the resultant wait count is zero, the required number of awaited events has occurred. If the number of needed events is less than the number of specified ECBs (as indicated by the "search" flag in the caller's RB), the Wait routine must clear the wait bit in each ECB that has not been posted. The purpose of clearing the wait bit in each unposted ECB is to prevent the Post routine from later confusing an uncleared wait bit with a new WAIT request. This is the same function that the Post routine performs when it decreases a wait count to zero and finds the search flag (RBECBWT) set in the waiting RB.

When the Wait routine has processed all ECBs specified in the input parameter list, it inserts the final wait count in the wait count field (RBWCF) of the caller's RB. If the wait count is greater than zero, the caller is now in the wait condition, or just "waiting", and cannot be dispatched. In this case, the Wait routine must indicate to the Type-1 Exit routine and to the Dispatcher that the Dispatcher must perform a task switch; i.e., the Dispatcher must search the TCB queue for the next highest priority ready TCB, and dispatch the current program associated with that TCB. The Wait routine indicates the need for a task switch by clearing the "new" TCB pointer at location IEATCBP. If the final wait count, placed in the wait count field of the caller's RB is zero, the awaited events have already occurred and the caller must not wait. Therefore the Wait routine does not indicate to the Dispatcher the need for a task switch.

After the routine has placed the wait count in the caller's RB, and has or has not indicated the need for a task switch, it must determine if the step under inspection is being job-step timed. It does this by determining if there is a job step TQE, and by testing the TCBTME field of the initiator TCB for a non-zero value. If the field is zero, Wait branches to the Type-1 Exit routine, because the 'step under inspection has not requested job step timing. If the field is non-zero, indicating that the step has requested job step timing, the entire tree of tasks must be examined to determine if the entire step is in an SVC wait. Wait uses the task select routine to examine all the TCB's in the tree of tasks, beginning with the job step TCB. When a TCB is found by the task select routine, Wait determines if the TCB which was just located is the TCB which originated the wait. If this is the case, the SVC old PSW is examined to determine if the Wait routine was entered because of the issuance of an SVC Wait (as opposed to a branch entry to Wait). If an SVC Wait was issued, the task select routine is entered again to find another TCB. If an SVC Wait was not issued, and the TCB is that which originated the wait, the Wait Routine branches to the Type-1 Exit routine. If the TCB located by the task select routine is not the TCB which originated the wait, the Wait routine tests the task ended bit in the TCBFLGS bytes of the TCB. If the ended bit is on, the task select routine is entered once again to find another TCB. If the ended bit is not on, the Wait routine selects the top RB on the TCB's RB chain, and examines the wait count field (RBWCF). If this field is zero (indicating the task is not waiting on any events), the Wait routine branches to the type 1 Exit routine. If the RBWTCF field is not zero, the Wait routine examines the RB old PSW field in the TCB's top RB. If the last instruction executed by the task currently under inspection (as indicated by the address contained in the right half of the RB old PSW, minus two) is an SVC Wait, the task select routine is entered to locate another TCB. If the last instruction executed was not an SVC Wait, the Wait routine branches to the Type 1 Exit routine.

When the task select routine can find no more TCBs in the tree of tasks (indicating that the entire tree of tasks is in an SVC Wait), the Wait routine uses the Dequeue TQE (entry point IEAQTD01) routine in the Timer Second Level Interruption Handler to remove the job step TQE from the timer queue. The Wait routine next converts the job step TQE from a task TQE to a 30-minute wait limit TQE while saving the CPU remaining time in the reserved slot of the TQE. The TQE is then enqueued on the timer queue by the Enqueue TQE routine (entry point IEAQTE00). The reason for this manipulation of the job step TQE lies within the job step timing algorithm.

When a tree of tasks is in an SVC wait, the step is not CPU timed. But because of the possibility of a Wait on an ECB which will never be posted, job step timing requires that a wait limit TQE be imposed on a step. The effect of the wait limit TQE would be to abnormally terminate a step which has waited on an event(s) for more than a specified amount of time (30 minutes), without having the event(s) occur.

After the routine has or has not converted the job step TQE, it branches to the Type-1 Exit routine to start the return to a main-line program. The Type-1 Exit routine tests the TCB pointers, IEATCBP and IEATCBP+4. If the need for a task switch has been indicated by the inequality of the two TCB pointers, the Type-1 Exit routine branches to the Dispatcher to perform a search of the TCB queue, and to return control to the current program of another task. If a task switch has not been indicated, the Type-1 Exit routine loads the SVC old PSW to give control directly to the caller. Since in this case all specified events have already occurred, the caller does not wait, except for supervisor processing.

Indicating the Occurrence of an Event and Restarting a Waiting Program

The Post SVC routine permits a program (the "posting" program or caller) to signal the occurrence of an event, such as the completion of an I/O operation, awaited by a waiting program. The routine signals (posts) the event's occurrence by altering one of two bits in a specified event control block (ECB) shared by both waiting and posting programs. The Post routine places in the event control block a "post code" supplied by the posting program. The post code may later be inspected by the waiting program, after it resumes execution, in order to determine the type of event that occurred. The Post routine determines if the program that is awaiting the posted event can be made ready (i.e., whether all awaited events have occurred).

If the waiting program can be made ready, and belongs to a task of higher dispatching priority than that of the posting program, the Post routine indicates to the Dispatcher that a task switch is needed (i.e., a ready program whose TCB is of higher priority than that of the caller should be dispatched). The Post routine determines if the initiator TCB of the TCB being posted has a TQE (the job step TQE) which indicates the step is being job step timed. If a job step TQE does exist, and it is a 30 minute wait limit TQE, the Post routine dequeues the TQE from the timer queue and converts the element to a task TQE.

There are three branch entry points to the Post routine. One (IGC002+6) is used exclusively by supervisor routines. A second (IEAOPT01) is used exclusively by the I/O Supervisor. The third (IEAOPT02) is used by both the I/O Supervisor and supervisor routines when they need to check the validity of user-specified ECBs. The I/O Supervisor's branch entry permits the I/O Supervisor to pass parameters in registers different from the standard registers, and also permits the saving of registers across the Post routine.

On branch entry from the I/O Supervisor, the Post routine saves the input registers, places the input parameters in the standard registers, and branches to the main-line part of the Post routine. On return from the main-line part of the Post routine, the saved registers are restored and control is returned to the I/O Supervisor. In this case, any task switch whose need is indicated by the Post routine will not occur until the I/O Supervisor branches to the Dispatcher, via the I/O FLIH.

On branch entry from a supervisor routine, the Post routine assumes that the input parameters are in the standard registers. This entry allows a supervisor routine to post an event without causing a task switch until the caller of the Post routine exits, instead of occurring when the Post routine exits.

With any branch entry, the Post routine returns control to the calling routine. But if the Post routine is entered via an SVC interruption, it exits via the Type-1 Exit routine.

The main-line part of the Post routine first determines if validity checking is necessary. Validity checking is bypassed if either of the exclusive branch entries is used, or if the entry is from the SVC FLIH and the calling program is a system routine. (A system routine operates with protection key of zero.) In these cases, the assumption is that the calling routine has passed a valid ECB address.

If validity checking is necessary, the Post routine determines that the ECB address passed by the caller is valid. Then, if the wait bit is set in the specified ECB, it checks the validity of the RB address contained in the ECB. The Post routine branches to the supervisor's Validity Check routine to perform the needed address checking.

The Validity Check routine, as indicated in the discussion of the Wait routine, performs three checks of an ECB address. It determines if the address lies on a fullword boundary, exists within the boundaries of main storage, and designates a storage area whose storage protection key matches the protection key in the caller's TCB. If any of these tests fails, indicating that the caller has incorrectly specified an ECB address, the Post routine sets up an error code (102) and exits to the ABTERM routine to schedule an abnormal termination of the caller's task. However, if the ECB address passed to the Post routine is valid, or if validity checking is bypassed, the Post routine continues processing.

The Post routine next tests the wait bit in the specified ECB. If the wait bit is set, the Post routine must check the validity of the RB address contained in the ECB. This is the address of the RB for the program that awaits the event now being posted. The RB address was placed in the ECB by the Wait routine when it serviced the WAIT macro instruction issued by the now-waiting program. Since the ECB is part of user-specified storage, and may have been modified by a user program after the Wait routine stored the RB address of the waiting program, the Post routine must now check the RB address.

The Post routine performs the check by making four tests. The first test determines whether the RB address is on a fullword boundary and is within machine-specified storage. The second test checks whether the old PSW field (RBOPSW) of the RB specified by the address is enabled for system interruptions. The third test compares the protection key in the RB old PSW of the specified RB with the protection key in the RB old PSW of the waiting program's RB. The fourth test determines whether the last-executed instruction of the waiting program, located via its RB old PSW field, was a WAIT macro instruction (SVC-1). If any of these tests fail, indicating that the RB address has been altered, the Post routine sets up an error code (202) and branches to the ABTERM routine to schedule an abnormal termination of the caller's task. If, however, the RB address appears valid, or the wait bit had not been set, indicating that the now posted event is not yet awaited, the Post routine continues processing.

The next step is to check the completion bit in the specified ECB. If the completion bit is set, indicating that the event now being posted has already been posted, there is no need for further processing. The Post routine treats this condition as a no-operation, and branches to the Type-1 Exit routine or to the caller.

If the Post routine was entered at a branch entry, it branches to the calling routine instead of to the Type-1 Exit routine. This is done without special tests. The Post routine branches to the address in the return register, general register 14. If the routine was entered from the SVC FLIH, general register 14 contains the address of the Type-1 Exit routine. But if the routine was entered at a branch entry point, general register 14 contains the return address of the caller.

If the completion bit is not set, the event represented by the ECB has not previously been posted, and processing can continue. The Post routine places in the specified ECB information useful to the waiting program and to the Wait and Post routines. The routine stores in the ECB a Post code specified as an operand of the POST macro instruction. The post code can supply to the waiting program, when it resumes execution, information about the event's occurrence. Besides storing the post code in the ECB, the Post routine sets the completion bit and clears the wait bit. These bits now indicate to both the Wait and Post routines, and also to a user program if it inspects the ECB, that the event represented by the ECB has occurred and is not now awaited.

The Post routine must next determine whether to decrease the wait count stored in a waiting program's RB. The wait count, stored in the RBWCF field of a waiting program's RB, indicates the number of awaited events that must occur before the program can resume execution. As long as the wait count stored in an RB is greater than zero, the program represented by the RB may not be dispatched.

The Post routine tests if the wait count in the waiting program's RB is already zero. This can occur in the special case in which the waiting program's task was abnormally terminated, via ABTERM, because of an event asychronous to the waiting program. The ABTERM routine resets to zero the wait count in the top RB on the RB queue of the TCB for which it is scheduling an abnormal termination. In this case, the Post routine returns control to the caller without changing the wait count in the waiting program's RB.

If the event is awaited, as indicated by a nonzero wait count, the routine subtracts one from the wait count field (RBWCF) of the waiting program's RB. It then tests the remaining wait count to determine if the waiting program can be made ready (i.e., whether the new wait count is now zero). If the new wait count is not zero, all events awaited by the program have not yet occurred, and further processing is not possible. In this case the Post routine returns control to the caller, or posting program, either directly if the caller is the I/O Supervisor, or via the Type-1 Exit routine. If, however, the new wait count is zero, indicating that the posted event is the last needed by the waiting program, further processing occurs.

The Post routine next determines if the posted ECB is part of a list of ECBs. In other words, is the minimum number of awaited events (the wait count) less than the number of specified ECBs (e.g., one needed I/O completion among three possible I/O completions)? If the answer is yes, one or more unposted ECBs exist whose wait bits remain set. These ECBs will cause error in future processing by the Post routine. The wait bits must be cleared. To determine if there are remaining unposted ECBs associated with the program whose wait count is now zero, the Post routine tests the "search" bit (RBECBWT) in the RB of the waiting program. If the bit is set (see discussion of Wait), the Post routine assumes that the number of awaited events is less than the number of specified ECBs. It obtains the address of the ECB list belonging to the waiting program, checks the validity of the list, and clears the wait bit in each outstanding ECB of the list.

After all unposted wait bits have been cleared, or if no unposted wait bits remained, the Post routine tests the TCBTME field of the initiator TCB of the task which is being posted. If the field is zero, it indicates that job step timing is not being performed for this step, and the Post routine would test if the program may be dispatched. If the field is non-zero, the Post routine examines the TQE type--REAL or TASK. If the TQE is TASK type, it indicates that the entire tree of tasks was not in an SVC wait, and the Post routine would then test if the program may be dispatched. If the TQE is REAL and on the timer queue, it indicates that a 30-minute wait limit TQE had been placed on the timer queue. If such is the case, the Post routine would branch to the Dequeue TQE routine (entry point IEAQTD01) in the Timer Second-Level Interruption Handler to remove the element from the queue. The Post routine would reinstate the actual CPU time remaining value in the TQEVAL field of the TQE. It would then mark the TQE as TASK type. This processing would allow the Dispatcher to once again calculate the CPU time used by this job step.

After the Post routine had or had not manipulated the job step TQE, it tests if the program whose wait count is now zero may be dispatched. The Post routine branches to the Task Switching routine to perform three tests. One test determines that the RB of the waiting program is at the top of its task's RB queue, and is therefore the current program for its task. An RB is at the top of its RB queue if it is pointed to directly by its TCB. The second test determines that special non-dispatchability bits (TCBFLGS field) have not been set in the TCB for the waiting program's task. If both tests are successful, a third test determines if the TCB of the waiting program has a higher dispatching priority than the TCB of the posting program.

If any of the tests fails, the Post routine returns control to the caller, or posting program, either directly (if the caller is the I/O Supervisor) or indirectly, via the Type-1 Exit routine. If the tests show that control should be returned to the waiting program, the Post routine indicates the need for a task switch by setting the "new" TCB pointer at IEATCBP to the address of the waiting program's TCB. The Post routine then returns control to the caller. If the return is indirect through the Type-1 Exit routine, the Type-1 Exit routine branches to the Dispatcher to perform the task switch. If the return, after the branch to the Post routine, is directly to the caller, the task switch does not occur until the caller itself branches to the Type-1 Exit routine or to the Dispatcher.

SERIALIZING THE USE OF A RESOURCE

The ENQ routine, working with the DEQ routine, permits programs issuing the ENQ macro instruction (or, in systems that include the shared DASD feature, the RESERVE 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 ENQ-issuing program is using any of the requested resources, the ENQ routine, via the Exit routine and the Dispatcher, returns control to the caller, which then gains access to its resource(s). But if any of the caller's resources are already in use by another ENQ-issuing program, being executed for another task, the ENQ routine places the caller in a wait condition until the resource becomes available. When the program that is using the resource(s) completes its use, it issues a DEQ macro instruction that causes the DEQ routine to remove one or more elements from the request queue, and reduce the wait count for the waiting program. If the wait count is now zero, the DEQ routine, via the Exit routine and the Dispatcher, may return control to the previously waiting (now ready) program. The program then gains access to its resource(s).

Separate although related functions are needed when a resource is requested and when the use of the resource is signaled complete. The functions may be listed under the headings of major and minor functions. Major functions are those which satisfy the principal purpose of the ENQ and DEQ macro instruction. Minor functions, although also important, are not related to the central purpose of the macro instructions. For example, the validity checking of input addresses may be considered a minor function.

Types of Resource Requests

There are two types of resource requests which may be specified by the ENQ-issuing program: an "exclusive" (E) request or a "shared" (S) request. The ENQ routine handles these two types of requests differently. An exclusive request is treated strictly 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. A "group" of shared requests, however, if 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 honored strictly on a task-priority basis. Figure 3-4 illustrates the handling of typical combinations of shared and exclusive resource requests.

Description of the Resource Queues

Before the discussion can proceed, the reader must become familiar with the construction of the resource queues and the nature of the search for already existing resource requests. Each resource request contained 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, specifying a set of resources, is represented on the resource queues by a major queue control block, or major QCB. Each major QCB contains, besides pointers to other control blocks, the Qname for a set of resources, e.g., the name of a data set. A major QCB thus represents a set of resources.

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

Figure 3-4. The Handling of Shared and Exclusive Requests

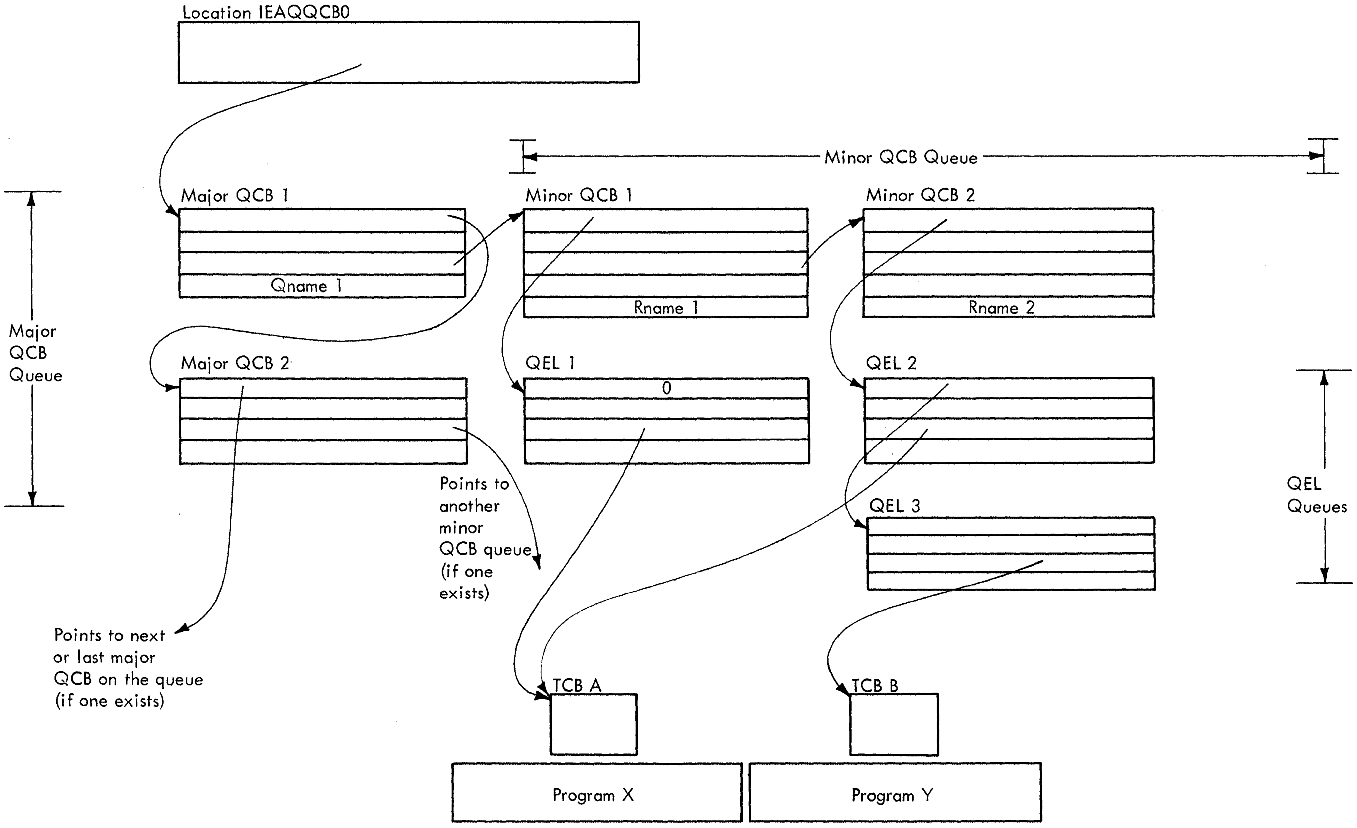
Each major QCB points to a minor QCB, which represents a particular resource within the set of resources, e.g., a specific record within a data set. As the reader may expect, a minor QCB contains, besides pointers, an Rname which is the name of the particular resource that has been requested. Each minor 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, or 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 two or more QELs, each representing 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 using program must complete its use and issue a DEQ macro instruction. The DEQ routine then moves the needed QEL or QELs to the top of the queues.

Figure 3-5 illustrates the resource queues.

In Figure 3-5, program X 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 Y has requested one of the resources being used by program X, that represented by major QCB 1 and minor QCB 2. Since the resource desired by program Y 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.



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 3-5. The Resource Queues

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

Requesting One or More Resources

The functions needed when a resource is requested may be listed under the headings of major and minor functions. Major functions are those which satisfy the principal purpose of the ENQ macro instruction. Minor functions, although also important; are not related to the central purpose of the macro instruction. For example, validity checking of input addresses may be considered a minor function.

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

- If necessary, creation of one or more queue control blocks (QCBs) to represent the requested resource, and the placing of these queue control blocks on the resource queues.
- Depending on the RET parameter, the creation of a queue element (QEL) to represent the request, and the placement of the QEL on a QEL queue.
- If the resource is available, the returning of control to the requestor, 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, either of two functions are performed, depending on the RET parameter:
  - The requestor is placed in a wait condition, pending the availability of the resource, or
  - Control is returned to the requestor with a code that indicates that the resource is unavailable.

The first major function, performed by the ENQ routine, is to search the resource queues to determine if the requested resource is already in use. The ENQ routine searches the major QCB queue for a major QCB that contains the specified Qname. If it finds the Qname, at least one resource in the set of resources is in use, and the routine then searches the associated minor QCB queue for the Rname.

PROCESSING IF THE REQUESTED RESOURCE IS NOT IN USE: If the requested resource is not in use, as indicated by the absence of QCBs with the specified Qname and Rname, control is returned to the caller. Depending on the RET code supplied by the caller, a return code may or may not be issued, and a QEL may or may not be constructed and placed on the resource queues. (Refer to Table 3-2 for the various results.)

PROCESSING IF THE REQUESTED RESOURCE IS IN USE: If another requestor has access to the resource, as indicated by a major and minor QCB containing the resource names, the resultant processing varies. It depends on the particular RET option that the caller has specified, on the type of request -- shared (S) or exclusive (E) -- and on the types of QEL's already on the queue. (The RET-parameter formats and the QEL formats appear in Section 12 of this manual.) Table 3-3 lists the different forms of resultant processing.

Note in Table 3-3 that a QEL is constructed and placed on a QEL queue if the requestor wants access to the resource and is willing to wait for it. The requestor's willingness to wait for the resource is indicated by a RET option of HAVE, NONE, or the omission of the RET operand. The RET option of TEST never causes creation of a QEL, only the generation of a return code (see Table 3-4) indicating whether the resource is available. If RET is USE, a QEL is created only if the requestor can have immediate access to the resource (Part 2 of Table 3-3).

Note that if all previous QELs on the queue and the present request are both for "shared" resources, there is no need for the caller to wait. The new requestor and those represented by the "shared" previous QELs on the queue may share the resource on a task-priority basis. Thus, a requestor need not have its QEL at the top of the "shared" group of QELs. Any requestor represented in the shared group may be executed if other requestors represented in the group are waiting for an event, such as an I/O completion, provided at least one member of the group is at the top of the queue.

RETURNING CONTROL: Control is returned to the caller if the requested resource or resources are available, or to the current routine of the next highest priority ready task if the caller must wait because the requested resource is in use. If the caller is to receive control, the return path is via the Exit routine and the

Table 3-2. Processing if a Requested Resource is not in Use

RET Parameter	Meaning of RET Parameter	QCB and/or QEL Constructed and Queued	Control is Returned to Caller With Code of:	Meaning of Return Code
TEST	Tests the queues to determine if the caller can have immediate use of the resource. Never constructs control blocks.	no	0	Resource is available
USE	Places QCB and/or QEL on queues only if caller can have immediate access to the resource.	yes	0	Resource is available
HAVE	Delay can be tolerated. Places QCB and/or QEL on queues.	yes	0	Resource is available
NONE or omitted	Same as HAVE but produces no return code.	yes	no code	

Table 3-3. Processing if a Requested Resource is in Use

Type of Previous QEL and Present Request	RET parameter is:	Resultant Processing
1. The previous QEL on the queue is "exclusive," or the present request is "exclusive."	USE or TEST	Sets return code equal to 4 and, via the Exit routine and the Dispatcher, returns control to the caller. A QEL is not constructed to represent the request.
	HAVE, NONE or omitted	Places requestor into wait condition by increasing SVRB wait count, constructs a QEL and places it on a QEL queue, indicates that a task switch is needed, branches to the Dispatcher to perform a task switch. If RET is HAVE, a return code of 0 is also produced.[1]
2. The previous QELs and the present request are both "shared," or There is no previous QEL for the resource (i.e., the QEL queue is empty).	TEST	Sets return code equal to 0 and, via the Exit routine and the Dispatcher, returns control to the caller. No QEL is constructed.
	NONE or omitted	Constructs a new QEL, places it on a QEL queue, and via the Exit routine and the Dispatcher, returns control to the caller.
	USE or HAVE	Sets return code equal to 0, constructs a new QEL, places it on a QEL queue, and via the Exit routine and the Dispatcher, returns control to the caller.

[1] This return code is passed to the requestor only after the resource becomes available.

Table 3-4. Return Codes for the ENQ Routine

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.

Dispatcher. But if the current routine of another task is to receive control, the return path is via the Dispatcher only. To determine the appropriate return path, the ENQ routine tests the RB wait count field in the current SVRB. If the RB wait count is zero, all requested resources are available and the caller can receive control. But if the RB wait count is greater than zero, the caller is effectively in a wait condition and cannot be given control.

If the caller can receive control, the ENQ routine branches to the Exit routine to remove the SVRB from its RB queue and free the storage area it occupies. The Dispatcher then returns control to the caller by loading the RB old PSW contained in the caller's RB.

If the caller cannot be given control, the ENQ routine prepares for the caller's future restart. It does this by changing the SVRB old PSW to point to the SVC 3 instruction in the communication vector table. When in the future the DEQ routine permits the caller's task to regain control, the first instruction to be executed will be the SVC 3, which causes supervisor linkage to the Exit routine to remove the SVRB.

After preparing for the caller's future restart, the ENQ routine indicates to the Dispatcher that it should search the TCB queue for the next highest priority ready TCB. The indication to the Dispatcher is the setting of the "new" TCB pointer (IEATCBP) to zero. Then the routine branches to the Dispatcher to search down the TCB queue to find the next highest priority ready TCB. When it finds the TCB, the Dispatcher places in execution the current routine of the associated task, by loading the RB old PSW contained in the current RB.

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

- Setting of the caller's task in "must complete" status, if specified, and if the caller is a system task.

- Detection of abnormal conditions that can cause the generation of an error code or the abnormal termination of the caller's task.

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

- Increasing of the "enqueue count" in the requestor's TCB.

- Increasing by a count of one the "nonrolloutable count" (TCBNROC) in the caller's job step TCB.

If the "set must complete" parameter is specified, the ENQ routine permits accelerated completion of the caller's task by setting nondispatchable all other tasks in the job step or system. To prevent scheduling of an abnormal termination of the caller's task, the ENQ routine places a special "must complete" flag in the TCB for the caller's task to serve as an indicator to the ABTERM and ABEND routines.

Two types of errors are checked. Invalid input-list addresses and duplicate resource requests for the same task are detected. A duplicate resource request is caused by two ENQ macro instructions for the same resource and task without an intervening DEQ macro instruction. The two types of error, when detected, result in either a return code and return of control to the caller, or an error code and the abnormal termination of the caller's task, via the supervisor linkage to the ABEND routine.

The AUTOPRG subroutine is used when the ABEND routine issues an ENQ macro instruction during an abnormal task termination. It consists of a purge of resource requests (QELs), and if necessary QCBs, belonging to tasks that are being abnormally terminated. Since the QELs cannot be removed by their original requestor, via the DEQ macro instruction, they are removed from the resource queues by the AUTOPRG subroutine, to make the requested resource available to the ABEND routine.

As an additional minor function, an "enqueue count" is maintained in the requestor's TCB. The enqueue count is stored in TCBQEL, the high-order byte of the TCBFSA field. The count is increased by the ENQ routine for each resource request and decreased by the DEQ routine when the use of the resource is signaled complete. The enqueue count is tested by the supervisor's EOT routine when the requestor's task is terminated normally. The test determines if all resource requests previously created for the task, via ENQ macro instructions, have been removed via corresponding DEQ macro instructions.

Placing the Caller's Task in "Must Complete" Status: The "set must complete" function is used by system programs[1] to allow the programs of one task to be executed while the programs of other tasks in the job step (STEP option) or other tasks in the system (SYSTEM option) are held nondispatchable, unable to be executed. The purpose is to prevent the abnormal termination of the "must complete" task by a routine belonging to another task in the job step or in the system. If a routine being executed for the "must complete" task produces a program check, the ABEND routine, via the system quiesce routine, terminates the task by setting it and its related tasks nondispatchable. The system quiesce routine also issues a message to the operator indicating that a CPU wait state has been averted and that no more jobs should be scheduled. Jobs that are already scheduled are allowed to reach normal termination. (See the description of ABEND2 and the System Quiesce routine in "Termination Procedures.")

The ENQ routine makes several checks to determine if the requestor's task should be set in "must complete" status. The ENQ routine tests whether the following requirements have been met:

- The requestor is a system routine, as indicated by a zero protection key in the requestor's RB old PSW.
- The RET operand of the ENQ macro instruction is not TEST.
- The SMC ("set must complete") operand of the ENQ macro instruction has been specified.
- The current SVRB is in a ready condition. (A ready condition is indicated by a RBWCF field of zero.)

The processing varies, depending on the outcome of the tests. If all requirements have been m@t, the ENQ routine performs "set must complete" processing. To perform "set must complete" processing the ENQ routine invokes the Set Status routine (IGC079) via the STATUS macro instruction. If the request is for step "must complete" status, it sets the "step must complete" nondispatchability flag (TCBSTP) in all TCBs of the job step except the requestor's TCB. (The job step's Initiator is also set nondispatchable.) If the request is for system "must complete" status, the ENQ routine sets the "system must complete" nondispatchability flag (TCBSYS) in all TCBs of the system, except the requestor's TCB and the TCBs of certain system tasks. The system tasks that remain dispatchable are the communications task, the rollout/rollin task (if the rollout feature is included), the system error task, and the transient area fetch tasks. The "must complete" nondispatchability flags indicate to the Dispatcher that it may not place in execution the routines controlled by these TCBs.

As part of "set must complete" processing, the ENQ routine also sets two flags in the requestor's TCB. One flag, when set, prevents the Stage 3 Exit Effector from scheduling user exit routines for the requestor's task. This precaution prevents the initiation of an abnormal termination in a user exit routine while the task is in "must complete" status. The other flag, when set, causes the ABEND routine to branch to the system quiesce routine if an abnormal termination is initiated during performance of the "must complete" task. The system quiesce routine terminates the "must complete" task and its subtasks and issues a message to the operator indicating that a CPU wait state has been averted and that the system should be allowed to quiesce (i.e., no more jobs should be scheduled and the jobs that are already scheduled should be allowed to reach normal termination).

If all requirements have not been met, the ENQ routine processes as follows. If the requestor is not a system routine, it sets up an error code (338) and invokes the ABEND routine to abnormally terminate the requestor's task. If the RET operand is TEST, or if the SMC operand has not been specified, the ENQ routine bypasses "set must complete" processing. If the current SVRB is in a wait condition, meaning that the requested resource is not available, the ENQ routine temporarily bypasses "set must complete" processing. Later, however, before exiting, it will point the SVRB old PSW to a restart point in the "set must complete" coding. When the requestor's task is redispatched, after the resource becomes available, the restarted ENQ routine will set the requestor's task in "must complete" status.

Detecting Abnormal Conditions: The detection of abnormal conditions consists of checking the validity of input addresses, and checking for duplicate requests issued for the same task.

[1] The test is for zero protection key in the requestor's RB old PSW.

• Table 3-5. TCB Flags That are Set if a Task is in "Must Complete" Status

Common Name	Symbolic Name	Displacement in TCB	TCB(s) in Which Flag Is Set	Purpose of Flag When Set
"Must Complete" nondispatchability flag (system or job step)	TCBSYS	33.4	All TCBs in system, except "must complete" TCB and certain system TCBs[1]	Indicates to the Dispatcher that it may not place into execution any routine associated with this TCB.
	TCBSTP	33.5	All TCBs in job step except "must complete" TCB	
"Must Complete" flag (system or job step)	TCBFSMC	30.3	"Must complete" TCB	If this task is in error, indicates to the ABEND routine that the task should be terminated and the system be allowed to quiesce via the System Quiesce routine.
	TCBFJMC	30.4		
Prohibit asynchronous exits flag	TCBFX	29.7	"Must complete" TCB	Indicates to the Stage 3 Exit Effector that it should not schedule a user exit routine for this task.

[1] The system TCBs that are not flagged nondispatchable are the communications TCB, the rollout/rollin TCB, the system error TCB, and the transient area fetch TCBs.

Checking the Validity of Input Addresses: The ENQ routine checks the validity of input parameters supplied by the caller. The parameters are a list of main storage addresses that point to names of resources or sets of resources. The check is designed to prevent a program check during later processing when the resource queues are being updated. The queues might be seriously disrupted, thus interfering with the performance of other tasks.

The ENQ routine must first determine if it is necessary to check the validity of the input parameters. If the caller is a system routine, as indicated by a zero protection key in the caller's RB old PSW, the assumption is that the input parameters are valid. In this case, the ENQ routine bypasses a validity check of the input parameter list. But if entry is from a user program, indicated by a nonzero protection key in the RB old PSW, the ENQ routine uses the supervisor's Validity Check routine to test the attributes of each input address. (For details see "Testing the Validity of User-Supplied Addresses.") If any one of the tests fails, indicating that the caller has incorrectly specified the address of a resource, the ENQ routine sets an error code (438) and issues an ABEND macro instruction. The ABEND macro instruction causes supervisor linkage to the ABEND routine to abnormally terminate the caller's task. Thus, the cause of the abnormal termination is pinpointed, avoiding the chance of a later program check during queue manipulation.

Checking for Duplicate Requests Issued for the Same Task: The ENQ routine determines if the caller, or another routine within the same task, has previously requested the same resource and has not dequeued the request from the queue. If the ENQ routine finds QCBs on the queues containing the same resource names as those requested by the caller, and an associated QEL containing the caller's TCB address, the caller has made a program error. According to the RET option that the caller has specified, the caller's task is abnormally terminated, or the caller is given control with a return code indicating that the requested resource is already enqueued for the caller's task. If the RET option is NONE, or has been omitted, the ENQ routine sets up an error code (138), and by issuance of the ABEND macro instruction, causes supervisor linkage to the ABEND routine to abnormally terminate the caller's task. But if the RET option has been specified and is not NONE, the ENQ routine sets up a return code (8) that indicates that the desired resource is already enqueued for the caller's task, and after processing other parameter-list elements, returns control to the caller. The caller can then optionally gain access to its requested resource.

Purging Requests Previously Enqueued for an Abnormally Terminating Task: If the requested resource is already enqueued, and if the caller's task is being abnormally terminated, the ENQ routine performs a special service for the third load module of the ABEND routine. It allows the ABEND routine to gain access to the abnormal dump data set, SYSABEND (or SYSUDUMP).

If the caller is the ABEND routine, it has issued an ENQ macro instruction to gain exclusive use of the dump data set, on which the terminated task's resources will be dumped. But the data set may already be enqueued for a subtask of the caller's task. The subtask may have been set in abnormal wait state (nondispatchable), as part of a higher level termination, before a DEQ macro instruction could be issued and the data set dequeued. In this case, the data set may be needlessly unavailable.

The ENQ routine, via its Autopurge subroutine, makes available the dump data set by releasing the QELs that represent previous requests for its use. It does this by removing from the resource queues all QELs belonging to the current task and its subtasks. The current ENQ request for the dump data set can then be serviced. (For information on the need for enqueuing the dump data set, refer to "Processing During ABEND3" in the chapter entitled "Termination Procedures.")

Increasing the Nonrolloutable Count: The ENQ routine increases by a count of one the "nonrolloutable count" (TCBNROC) in the caller's job step TCB. It does this for each resource for which the ENQ macro instruction is issued. To increase the count, the ENQ routine invokes the Set Status routine (IGC079), via the STATUS macro instruction. The "nonrolloutable count," when greater than zero, makes the job step ineligible to be rolled out.

PROCESSING IN SYSTEMS WITH SHARED DASD: When the shared DASD feature is included in the system, the ENQ routine performs device reservation functions in addition to its normal functions. This section describes the device reservation functions.

The RESERVE macro instruction must specify a valid UCB address for a shared direct access device. The ENQ routine checks the UCB address, and if it is not valid issues an ABEND macro instruction. This test follows the verification of input addresses that is a normal part of ENQ processing.

The QEL initialization function of the ENQ routine is expanded for a reserve request. When control of the requested resource can be assigned to a task, the ENQ routine places the UCB address in the QEL and sets the QEL reserve flag. The reserve count in the specified UCB is then incremented by one.

The requested resource cannot be assigned to perform a new task when the following conditions occur:

- Resource is in use.

- Previous QEL on the queue is exclusive, or the present request is exclusive.

- RET operand of the RESERVE macro instruction specifies HAVE, NONE, or is omitted.

Under these conditions, the ENQ routine prepares for a task switch. It increments the SRVB wait count by one, thus placing the task for which the resource was requested in a wait condition. The ENQ routine places the address of the SVRB in the QEL for subsequent use by the DEQ routine. The need for a task switch is indicated and control is given to the Dispatcher.

When the requested resource becomes available because it is no longer needed in the performance of another task, the DEQ routine will remove the top QEL and determine whether the task associated with the new top QEL should be made ready. If it should, the DEQ routine decrements the wait count in the SVRB whose address is in the new top QEL. When the wait count becomes zero, the waiting task can be dispatched; the ENQ routine then regains control. The "reserve restart" subroutine of the ENQ routine inserts the UCB address in the QEL, sets the reserve flag, and increments the reserve count in the UCB by one.

Signaling That the Use of One or More Resources is Complete

When a program that previously issued an ENQ macro instruction (or, in systems which include the shared DASD feature, a program which issued a RESERVE macro instruction) and has been using an enqueued resource completes its use, it issues a DEQ macro instruction. The DEQ macro instruction, via an SVC interruption (SVC 48), 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 one or more QEL queues. If the RB wait count becomes zero, thus making ready the waiting program, the DEQ routine invokes the Task Switching routine. The Task Switching routine tests the need for a possible task switch, and branches to the Exit routine and the Dispatcher to return control. The program that receives control is either the caller or the previously waiting program, depending on relative task priority. An additional function of the DEQ routine, used only by a supervisor routine, is to reset a task in "must complete" status, 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 whose use is now complete. 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 requestor represented on the QEL queue, reduction of the wait count in its SVRB, and testing if the requestor is ready to resume execution.

- Determining if a readied requestor can replace the caller as the next-to-be-executed routine. This involves a comparison of TCB dispatching priorities by the Task Switching routine.

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

Updating the Resource Queues: In order to update the resource queues, the DEQ routine searches for the QEL that represents a request that should now be dequeued. It first finds both a major QCB and a minor QCB containing the specified resource names. The routine then examines the QEL queue associated with the specified resource. If the caller's TCB address matches that stored in one of the QELs logically at the top of its queue, the DEQ routine dequeues the QEL and, via supervisor linkage to the FREEMAIN routine, frees the space that the QEL occupies.

The DEQ routine examines the QCB queues to determine if any QCB may be released. If there are no more QELs queued to the minor QCB for the resource, there are no further requests for the resource, and the minor QCB can be released. In this case, the routine dequeues the minor QCB from its queue and frees the space that it occupies. It then examines the minor QCB queue to decide whether the major QCB is no longer needed and can be similarly eliminated. If there are no minor QCBs queued to the major QCB, there are no outstanding requests for the entire set of resources. In this case, the DEQ routine removes the major QCB from its queue and frees its space. The routine then processes in a similar manner any other input parameters which represent QELs to be dequeued.

Determining if the Next Waiting Requestor Should be Readied: After the old top QEL is dequeued, the DEQ routine determines if the next waiting requestor, 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. According to the result of the decision, the SVRB wait count for the waiting requestor may or may not be reduced and tested for readiness (zero wait count). The criteria and results for three different situations are described in Figure 3-6.

Determining if a Readied Requestor Should Be Dispatched: For each SVRB whose awaited resources are available, as indicated by a zero RB wait count, the DEQ routine tests whether the associated requestor can be dispatched. If the requestor's task is of higher dispatching priority than the caller's, the requestor may be dispatched in place of the caller. For each SVRB that has a zero wait count, the DEQ routine invokes the supervisor's Task Switching routine to compare dispatching priorities. If the readied requestor's TCB has a higher priority than the caller's, the Task Switching routine indicates this fact to the Dispatcher by placing the requestor's TCB address in the "new" TCB pointer, IEATCBP.

Returning Control: The DEQ routine returns control to the caller or a readied requestor, via the Exit routine and the Dispatcher. The Exit routine dequeues the SVRB from its RB queue and frees the space that the SVRB occupies. The Dispatcher decides whether to return control to the caller or to a readied requestor, depending on the contents of the "new" TCB pointer, IEATCBP. If the "new" TCB pointer contains the address of the current TCB, the Dispatcher returns control to the caller. Otherwise, the Dispatcher returns control to the requestor whose TCB address is in the pointer. In this case a task switch has occurred.

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

Conditions	Status of QEL Queue	Resultant Processing
Condition A		
dequeued QEL	"shared" QEL	Routine does not reduce wait count in requestor's RB. (Requestor already has access to the resource.)
new top QEL	"shared" QEL	
Condition B		
dequeued QEL	QEL of either type	Routine reduces wait count in requestor's RB and if new wait count is zero, it invokes the Task Switching routine to test whether the requestor may be dispatched instead of the caller.
new top QEL	"exclusive" QEL	
Condition C		
dequeued QEL	"exclusive" QEL	Routine reduces wait count in requestor'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.
new top QEL	"shared" QEL	

Figure 3-6. Determining if the Next Waiting Requestor Should be Readied

- If the "reset must complete" parameter is present, the clearing of the "must complete" status of the caller's task.
- Checking the validity of input addresses.
- Checking if a specified resource was originally requested for any task.
- Checking if the caller has access to a specified resource.
- Reducing the "enqueue count" in the caller's TCB. The enqueue 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.")
- Decreasing by a count of one the "non-rolloutable count" (TCBNROC) in the caller's job step TCB.

The Clearing of "Must Complete" Task Status: If the "reset must complete" parameter has been specified, the DEQ routine restores multitask operation to the job step or system, which temporarily had been performing only the caller's task. This restoration is done only if the caller is a system routine (uses zero protection key). If the caller is not a system routine, the DEQ routine sets up an error code (330) and invokes the ABEND routine to abnormally terminate the caller's task.

The DEQ routine clears the "must complete" nondispatchability flag (see Table 3-5) in each TCB of the job step or system, depending on the scope. This action allows the Dispatcher to restart routines for previously nondispatchable tasks. The DEQ routine also clears two flags in the caller's TCB. One flag, when cleared, allows the Stage 3 Exit Effector to resume the scheduling of user exit routines for the caller's task. The other flag, when cleared, permits the ABEND routine to abnormally terminate the caller's task, if the need arises, instead of placing the CPU in a disabled wait state (see Table 3-5). To clear the "must complete" status, the DEQ routine invokes the Set Status routine (IGC079), via the STATUS macro instruction.

Checking the Validity of User-Supplied Addresses: The DEQ routine must check the validity of a list of main storage addresses supplied by a user program. The addresses point to names of resources or sets of resources. But if entry is from a system routine, the assumption is that the input parameters are valid, and no validity check is made.

If entry is from a user program, as indicated by a nonzero protection key in the caller's RB old PSW, the DEQ routine checks input parameters via the supervisor's Validity Check routine. The Validity Check routine tests the typical three attributes of each input address. (For details, see "Testing the Validity of User-Supplied Addresses.") If any of the validity checks fails, indicating that the caller has incorrectly specified the address of a resource, the DEQ routine sets an error code (430) and issues an ABEND macro instruction. The ABEND macro instruction causes supervisor-assisted linkage to the ABEND routine to abnormally terminate the caller's task.

Determining if a Specified Resource Was Originally Requested for Any Task: If the input parameters are valid, the DEQ routine searches the QCB queues to determine if the specified resource was originally requested for any task. The resource may be dequeued only if it was previously enqueued. If the resource was enqueued, the resource names are represented on the QCB queues, contained in a major and a minor QCB. But if the two QCBs representing the resource cannot be found, a DEQ macro instruction has been issued for a resource that was not enqueued, or which has already been dequeued. The DEQ routine recognizes an error condition and reacts according to the RET option, as shown in Table 3-6.

Determining if the Caller Has Access to a Specified Resource: The caller can rightfully dequeue a resource only if it has access to it. To determine if the caller has such access, the DEQ routine examines the QEL queue associated with the resource. The QEL containing the caller's TCB address should be at the logical top of the queue, or be one of a "shared" group of QELs at the logical top of the queue. If neither condition exists, the DEQ routine recognizes an error condition, as shown in Table 3-6.

Decreasing the "Nonrolloutable Count": The DEQ routine decreases by a count of one the "nonrolloutable count" (TCBNROC) in the caller's job step TCB. It does this for each resource for which a DEQ macro instruction is issued by a routine of the job step. To decrease the count, the DEQ routine invokes the Set Status routine (IGC079), via the STATUS macro instruction. When the "nonrolloutable count" is zero, the job step is eligible to be rolled out to satisfy an unconditional storage request from a job step of another job.

Table 3-6. Error Conditions When use of a Resource is Signaled Complete

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.

PROCESSING IN SYSTEMS WITH SHARED DASD: When the shared DASD feature is included in the system, the DEQ routine performs device release functions in addition to its normal functions. This section describes the device release functions.

A release request (a DEQ macro instruction associated with a RESERVE macro instruction) is indicated when the reserve flag in the QEL is set. The DEQ routine decrements by one the reserve count in the UCB whose address is in the QEL; if this reduces the count to zero, the associated direct-access device must be released. The EXCP Interface subroutine of the DEQ routine issues a GETMAIN macro instruction to obtain space for the control blocks required for the EXCP macro. When all control blocks (IOB, DCB, ECB, DEB, CCW, AVT) have been initialized, the EXCP Interface subroutine issues an EXCP macro, followed by a WAIT macro instruction.

The effect of the execution of the EXCP Interface subroutine is that I/O activity is initiated at the specified direct-access device. Because the reserve count was reduced to zero before the I/O activity started, IOS will physically release the device.

When the WAIT macro instruction has been satisfied, the EXCP Interface subroutine regains control to remove the control blocks it initialized. Normal DEQ processing then resumes.

The ABEND6 routine must also terminate device reservations acquired through the RESERVE macro instruction and not released through a subsequent DEQ macro instruction. These device reservations occur only in systems with the shared DASD option.

Outstanding reservations are reflected in the TCB enqueue count (offset 112 in the TCB); the enqueue count indicates the number of outstanding ENQ requests (that is, it is not directly related to outstanding reservations). When the enqueue count is not zero, the ABEND4 routine branches to the ENQ/DEQ Purge subroutine in the ENQ/DEQ module. If shared DASD is included in the system, this routine determines whether the terminating task has outstanding device reservations. The QEL indicates whether it was created as a result of a RESERVE or an ENQ macro instruction. If the result of RESERVE, the device is released.

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 (e.g., 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. Note that these routines do not schedule the execution of user program check routines. ABEND processing that results from a program check can be intercepted by a STAE macro instruction which specifies a STAE exit routine address. See "Specifying a Task Asynchronous Exit Routine" for a description of the STAE routine.

As shown in Figure 3-7, the handling of a request for the future execution of a user exit routine is a multipart procedure, interwoven with the execution of programs executed for other tasks. The procedure begins when the user program originally issues a request for an exit routine. The user program makes the request via operands in such macro instructions as ATTACH (ETXR operand), STIMER, and DCB. A system routine (e.g., the Attach routine) then issues a special system macro instruction (CIRB). The CIRB macro instruction causes the Stage 1 Exit Effector to construct 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 the 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. There are two such queues whose elements represent requests to use a particular exit routine. One queue contains IQEs and represents 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 of each program requesting the same exit routine. When an element is placed on either exit queue, the

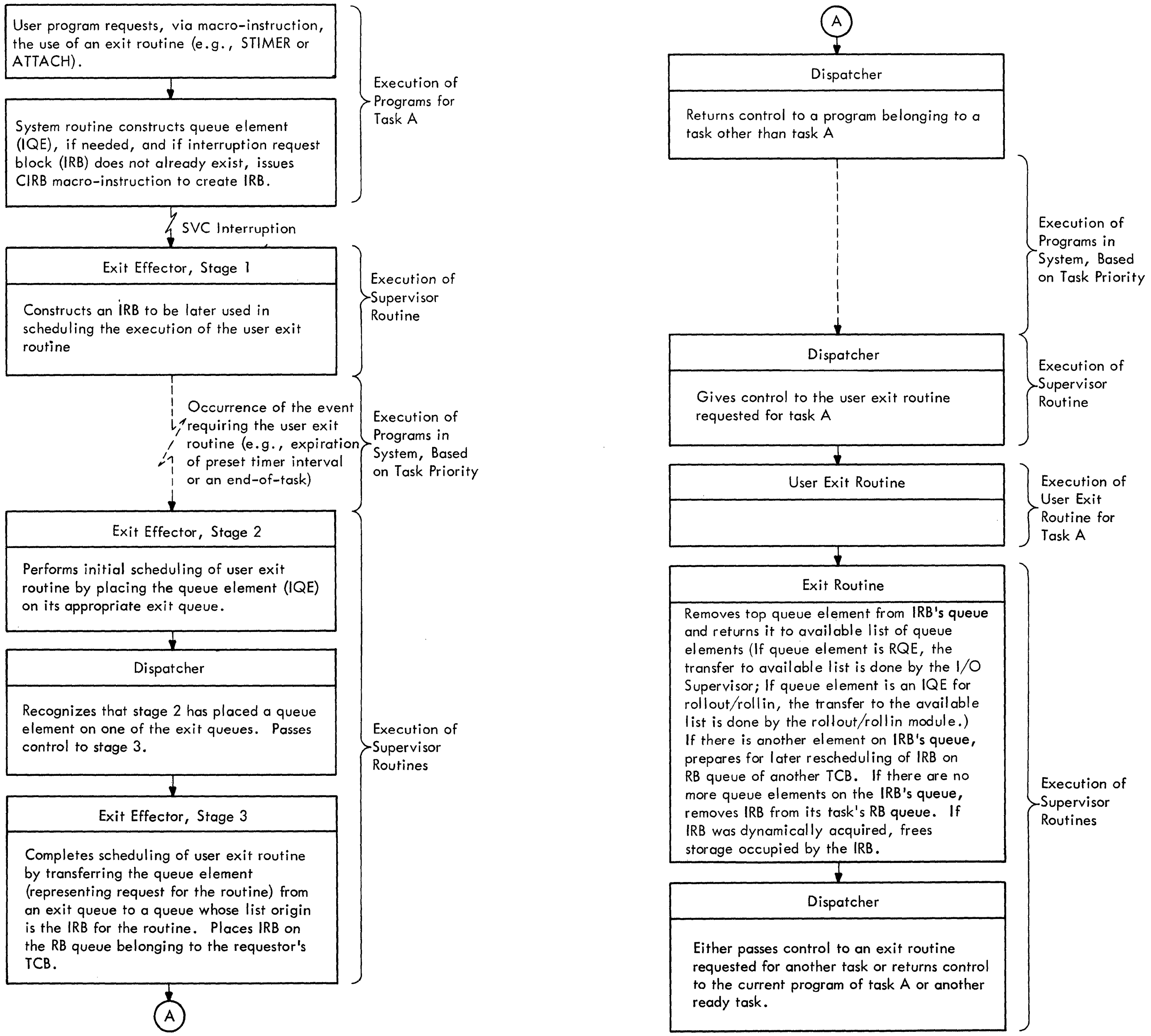

User program requests, via macro-instruction, the use of an exit routine (e.g., STIMER or ATTACH).
System routine constructs queue element (IQE), if needed, and if interruption request block (IRB) does not already exist, issues CIRB macro-instruction to create IRB.
Execution of Programs for Task A
SVC Interruption
Exit Effector, Stage 1
Constructs an IRB to be later used in scheduling the execution of the user exit routine
Execution of Supervisor Routine
Occurrence of the event requiring the user exit routine (e.g., expiration of preset timer interval or an end-of-task)
Execution of Programs in System, Based on Task Priority
Exit Effector, Stage 2
Performs initial scheduling of user exit routine by placing the queue element (IQE) on its appropriate exit queue.
Dispatcher
Recognizes that stage 2 has placed a queue element on one of the exit queues. Passes control to stage 3.
Exit Effector, Stage 3
Completes scheduling of user exit routine by transferring the queue element (representing request for the routine) from an exit queue to a queue whose list origin is the IRB for the routine. Places IRB on the RB queue belonging to the requestor's TCB.
Execution of Supervisor Routines
A
A
Dispatcher
Returns control to a program belonging to a task other than task A
Execution of Programs in System, Based on Task Priority
Dispatcher
Gives control to the user exit routine requested for task A
Execution of Supervisor Routine
User Exit Routine
Execution of User Exit Routine for Task A
Exit Routine
Removes top queue element from IRB's queue and returns it to available list of queue elements (If queue element is RQE, the transfer to available list is done by the I/O Supervisor; If queue element is an IQE for rollout/rollin, the transfer to the available list is done by the rollout/rollin module.) If there is another element on IRB's queue, prepares for later rescheduling of IRB on RB queue of another TCB. If there are no more queue elements on the IRB's queue, removes IRB from its task's RB queue. If IRB was dynamically acquired, frees storage occupied by the IRB.
Dispatcher
Either passes control to an exit routine requested for another task or returns control to the current program of task A or another ready task.
Execution of Supervisor Routines

Figure 3-7. Scheduling of Asynchronous Exit Routines

IRB that represents an exit routine is not yet on a task's RB queue and cannot yet be executed. The placing of 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, can compete for CPU time with programs being executed for other tasks. When the requesting program's TCB, which points to the IRB, has highest priority among the ready TCBs, the Dispatcher loads the IRB's old PSW to place the user exit routine in execution.

When the user exit routine is complete, it invokes supervisor-assisted linkage to the supervisor Exit routine. The supervisor Exit routine removes the top queue element from the IRB's 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 dynamic and 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.

The Stage 1 Exit Effector (CIRB Routine)

The Stage 1 Exit Effector is a resident, disabled, reenterable SVC routine that may be called by a supervisor routine, such as the Attach routine, or by a data management routine. Its purpose is to create and initialize, according to input parameters, an interruption request block or IRB to control a user exit routine whose future use is requested by the caller. The routine obtains 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 SVC routine, when it is executed, uses the work area to construct both the new TCB for the subtask and the IQE for the ETXR or end-of-task exit routine. The Stage 1 Exit Effector obtains space for the IRB and the work area, if requested, from supervisor queue space, subpool 253. The work area follows and is immediately contiguous to the IRB. The register save area, if requested, is obtained from subpool zero of the user program's region of storage, and is therefore not contiguous to the IRB and its work area. After obtaining the needed storage for the IRB and optional work and save areas, the Stage 1 Exit Effector initializes the IRB, as shown in Section 12. The initialization is done according to flag bits passed to the routine in register 1. The information placed in the IRB during the initialization includes the save area address, the size of the IRB, the entry-point address of the user exit routine, and the PSW to be loaded to start execution of the user exit routine. When the Stage 1 Exit Effector completes the initialization of the IRB, it returns control to the calling program via the supervisor Exit routine and the Dispatcher.

The Stage 2 Exit Effector

When Stage 2 is entered, Stage 1 has already created and initialized the IRB, and the requesting system routine has created and initialized the IQE. Stage 2 is entered as a subroutine by any supervisor routine wishing to schedule a user exit routine. Two typical callers are the supervisor EOT routine, during end-of-task processing, and the Timer Second-Level Interruption Handler, when a preset timer interval has expired. Stage 2 places the input queue element, whose address is passed in register 1, onto either of two exit queues. The queue element is queued at the bottom of the appropriate queue. If the input address appears in true form (a positive address), Stage 2 places the queue element on the queue of RQEs, (called AEQA) 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 (AEQJ), whose elements are used to schedule non-data-management exits. (See Section 12, "Control Blocks and Tables" 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

The Stage 3 Exit Effector operates as a subroutine of the Dispatcher. Its purpose is twofold: to transfer IQEs and RQEs from their exit queues to queues belonging to particular IRBs (and thus specific to a particular user exit routine), 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 a special high-priority system TCB.

Stage 3 is entered from the Dispatcher if the Stage-3 switch (IEA0DS01) has been set by Stage 2, indicating that at least one IQE or RQE is on an exit queue. (There are two exit queues, one for IQEs, the other for RQEs.) Stage 3 begins by processing the IQE queue. If there are no elements on the IQE queue, the routine then processes the RQE queue.

If there is at least one IQE, stage 3 performs some tests to determine if each

IQE on the queue may be removed from the exit queue and placed on a queue belonging to an IRB (which represents a particular user exit routine). If any of the tests indicates that an IQE should not be transferred to an IRB queue, Stage 3 obtains the next IQE on the list and repeats the tests. One of the tests asks whether the IQE's intended IRB is "active." (The IQE contains a pointer to its "intended" IRB. (See Section 12, "Control Blocks and Tables.") An IRB is considered "active" if the IRB is already queued to a TCB. This condition is indicated by the RBFACTV bit in the RBSTAB field of the IRB. If the IQE's intended IRB is already queued to a TCB, i.e., is "active", the routine then tests if the same TCB is specified by this IQE. (The IQE contains a pointer to the TCB for the task that needs the user exit routine.) In other words, this test asks whether the IRB that is already scheduled (queued) is the intended IRB for this IQE. If the IRB is queued to the correct TCB, or if the IRB is inactive (not queued to any TCB), Stage 3 removes the IQE from its exit queue and queues it to the bottom of the IRB's queue. (The list origin for the IRB's IQE queue is in the IRB and is called the RBIQE field. See Section 12.) After placing the IQE on the IRB's list of queue elements, Stage 3 proceeds to initialize the IRB as follows, in preparation for entry to the user (asynchronous) exit routine.

If the IRB is not already on the RB queue of a TCB (as indicated by the previous test of the RBFACTV bit in the IRB status field), the routine places the IRB on the appropriate task's RB queue. The TCB then points to this IRB as the current RB representing the routine next to be executed for this task. Stage 3 then saves register contents stored in the TCB (that could later be overlaid and thus lost) by moving the contents to the register save area of the IRB. It initializes the PSW and standard register settings for the user exit routine. Then, in order to determine if the newly queued IRB's task is of higher priority than the current task, Stage 3 invokes the Task Switching routine to test if a task switch is needed. If a task switch is needed, the Task Switching routine indicates this need to the Dispatcher. It does this by placing the address of the higher priority TCB in the "new" TCB pointer (IEATCBP). The address of the "new" TCB pointer is contained in the CVT at location CVTTCBP.

If there are one or more IQEs remaining unprocessed on the IQE exit queue, Stage 3 processes these IQEs in a manner similar to that just described.

When all elements on the IQE queue have been processed, Stage 3 processes the queue of RQEs in a similar fashion. The reader may recall that the RQEs, supplied by the I/O Supervisor, are used to schedule I/O exit routines. One feature of RQE processing is different from IQE processing, and deserves special mention.

One or more of the RQEs may represent a request by an I/O routine for the use of a system error handling routine. When Stage 3 examines each element on the RQE queue, it tests if the queue element represents a request to use an error routine. The "F" bit in each RQE indicates whether the RQE represents such a request. (See format of an RQE in Section 12.) If one or more RQEs represent requests for the use of an error routine, Stage 3 performs special processing for these RQEs. Other RQE's, not representing requests for error routines, are processed in a manner very similar to IQE processing.

For each RQE representing a request to use an error routine, Stage 3 tests first whether a special system request block is "active", i.e., already queued to its system error TCB. The system error TCB is a permanent TCB of high priority whose current "dummy" RB is normally in a wait condition. The request block that represents a system error handling routine is called a system interruption request block, or SIRB.

If the SIRB is already queued to the system error TCB, the "active" bit (RBFACTV) is set in the SIRB's status field. In this case, the error routine has already been scheduled for another request. The new request must then be deferred and await the next execution of Stage 3, when the Dispatcher is next entered.

If the SIRB is not "active", that is, not queued to the system error TCB, Stage 3 clears the I/O error flag or 'F' bit in the RQE. It removes the RQE from its asynchronous exit queue and queues it to the SIRB. Stage 3 then initializes the SIRB. As part of this initialization, it sets the RB old PSW to provide reentry to the "error fetch sequence" (ERFETCH) of Stage 3.

Besides altering the RB old PSW, Stage 3 queues the SIRB to the system error TCB. The error TCB now points directly to the SIRB, instead of to its permanent dummy RB. When all RQEs on the asynchronous exit queue have been processed, the Dispatcher will cause reentry to Stage 3 at entry point ERFETCH, under control of the system error TCB.

If Stage 3 did not complete the processing of RQEs at the time it discovered a request for a system error routine, it now completes the processing of other RQEs. (If an RQE represents a request for an I/O error routine, the 'F' flag appears set in the RQE.) After Stage 3 queues the SIRB to the system error TCB, it defers subsequent error requests on the RQE queue. These requests are not processed until the SIRB becomes "inactive", removed from its TCB during Exit routine processing.

In a multiprocessing system, the TCB indicated by the IQE or RQE may be the current TCB on the second CPU. If it is, control is passed to the SHOLDTAP routine which interrupts the second CPU with an indication that the Dispatcher is to gain control on the second CPU and place the IRB on the appropriate RB Queue.

After processing the two lists, IQEs and RQEs, Stage 3 returns control to the Dispatcher. The Dispatcher passes control to the interrupted program belonging to the current task, or to a user exit routine belonging to another task, or to the Stage 3 Exit Effector at entry point ERFETCH to begin the loading of an error routine.

FETCHING AN ERROR ROUTINE: ERFETCH is the entry point to a so-called "error fetch sequence" that performs for error routines the function that the Transient Area Fetch routine performs for transient SVC routines. That is, the "error fetch sequence" searches for the desired error routine and, if necessary, fetches it to the I/O Supervisor transient area.

The "error fetch sequence" first checks to see if the I/O Supervisor transient area of main storage contains the needed error routine. If the error routine is in the transient area, its entry point address is placed in the old PSW field of the SIRB. When the Dispatcher next regains control, it will load this PSW to begin execution of the error routine. If the needed error routine is not in the I/O Supervisor transient area, the "error fetch sequence" invokes the BLDL routine to get data set directory information, in preparation for fetching the I/O error module.

If there is an error during execution of the BLDL routine, the error fetch sequence sets up an error code (806) and branches to the ABTERM routine. The ABTERM routine schedules abnormal termination of the task for which the error routine was requested. The error fetch sequence then returns control to the current routine of the highest priority ready task, via the Exit routine and the Dispatcher.

If there is no error during execution of the BLDL routine, the error fetch sequence branches to the supervisor's Program Fetch routine to load the error routine. If no error is detected during the fetch process, the error fetch sequence places in the RB old PSW of the SIRB the entry point address of the I/O Supervisor transient area. The error fetch sequence then branches to the Dispatcher which will load the SIRB old PSW to start execution of the error routine. If, however, an error is detected during the fetch process, the error fetch sequence loads a special PSW to place the CPU in the wait state. This is necessary since a critical error has occurred. The operator can cause reentry to the error fetch sequence to retry the fetch by pressing the Reset and Start keys on the Operator Control Panel.

The Exit Routine

This discussion of the Exit routine will include only that part of its processing that affects the scheduling of user (asynchronous) exit routines. Other aspects of its processing will be described later in the topic entitled "Exiting Procedures."

The Exit routine, among its other functions, deletes the scheduling performed by the three Exit Effectors. The Exit routine is given control by the SVC FLIH after a user exit routine has issued a RETURN macro instruction. For both types of RBs -- SIRB and IRB -- if there are no other requests (queue elements) for the user exit routine, the Exit routine removes the RB from its TCB. If the RB is an IRB, and therefore was dynamically acquired by a GETMAIN macro instruction, the Exit routine 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 and is no longer needed. 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 to use the asynchronous exit routine for another task, the Exit routine prepares for future reentry to the 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.

The above discussion is an overview of the Exit routine's role in the scheduling of user (asynchronous) exit routines. A more detailed description of the same processing will now follow.

The Exit routine first determines the type of RB under which the caller (RETURN-issuing program) is operating. If the type is either SIRB or IRB (as indicated by the RBFTP bits in the RBSTAB field), the Exit routine assumes that the RETURN-issuing program, or caller, is a user exit routine. If the caller's RB is an SIRB, which means that the RETURN-issuing program is a system error routine, the Exit routine branches to the routine that removes an RB from its TCB and tests whether to free the RB's storage space. Since the SIRB is a permanent RB, its space is not freed. But the SIRB is removed from the system error TCB. Since there are no further requests for the error routine (no RQEs queued to the SIRB), the Exit routine branches to the Dispatcher to return control to the current routine of another task.

If the caller's RB is an IRB, the returning program is a user exit routine and not a system error handling routine. In this case the Exit routine performs more elaborate processing. It first checks the type of queue element at the top of the IRB's queue to determine if the element is an IQE or an RQE. (A queue can contain only one type, not both.) The Exit routine makes this check by testing the RBSTAB subfield called RBFIQETP, which indicates the type of elements queued to the IRB: RQEs or IQEs. (Refer to Section 12 for the formats of this RB field.)

If the IRB's queue contains one or more RQEs, the Exit routine removes the top RQE (no longer needed) and places the RQE on a list of available RQEs for use by the I/O supervisor. The Exit routine obtains this requeuing by branching to an entry point (INT025) to a section of the I/O supervisor that returns RQEs to an available list for future use. The Exit routine then tests whether another RQE is on the IRB's queue, representing an outstanding request for use of the I/O exit routine by a program belonging to the same task. If another RQE is on the queue, the Exit routine initializes registers to prepare for future reentry to the user exit routine and branches to the Dispatcher.

If in its test of the RQE queue, the Exit routine finds that there are no other RQEs on the IRB's queue, it performs processing somewhat similar to that performed for an SIRB. The routine transfers the contents of the caller's registers from the IRB to the TCB's save area and removes the IRB from the RB queue belonging to its TCB, since there are no further requests for the Exit routine and the IRB is no longer needed. The Exit routine then tests whether the space occupied by the IRB may be freed. The test consists of checking the RBFDYN bit in the IRB. If the bit shows that the IRB was dynamically acquired and is not a permanent RB (such as the SIRB) the block may be freed. If the IRB was dynamically acquired, the Exit routine frees its storage area by invoking the FREEMAIN routine, and branches to the Transient Area Refresh routine. The Dispatcher returns control to the current program belonging to the user exit routine's task, when that task next becomes the highest priority ready task. The PSW for this program is contained in the next RB on the TCB's RB queue, after the IRB has been removed from the RB queue.

If the previous test of the type of queue element on the IRB's queue indicated an IQE, a request for a non-I/O exit routine, the Exit routine tests for a zero "use count." The use count (stored in the 25th byte of the IRB) indicates to the Exit routine the number of outstanding requests to execute the same user exit routine. For example, successive ATTACH macro instructions issued for a parent task may have specified in the ETXR operand the use of the same end-of-task routine for different subtasks. If the IRB use count is not zero, the Exit routine decreases by one the use count to indicate the remaining number of requests, not yet scheduled, for the user exit routine.

After decreasing the use count, or if the use count is already zero (indicating no outstanding requests for the user exit routine), the Exit routine removes the top IQE from the IRB. This is done because the represented request has been serviced. The Exit routine then tests whether the user program has provided a work area at the end of the IRB. It also tests whether the user program wants the IQE to be queued on a "next available" list. The IQE may be queued in the work area as an "available" element for use by the Stage 2 Exit Effector in scheduling a new request.

The Exit routine tests for the existence of the work area by determining the size of the IRB. A work area exists if the size exceeds 93 bytes (12 doublewords, as indicated by the RBSIZE field of the IRB). The exit routine then determines whether to queue the IQE to the "next available" list (RBNEXAV). If the RBFIQETP field is '11', it queues the IQE to the "next available" list. Otherwise, the routine continues processing.

The Exit routine next tests for another IQE on the IRB's queue of IQEs. The processing from this point is similar to that for RQEs, previously discussed.

Finally, after either initializing registers for reentry to the user exit routine, or removing the IRB and freeing its storage, the Exit routine branches to the Transient Area Refresh routine. The Transient Area Refresh routine determines that the exiting routine is not a transient SVC routine. It then returns control to the Dispatcher. The Dispatcher returns control to either the usér exit routine or another routine. The other routine may be the routine that was interrupted by the timer, if a timer interruption had occurred; or it may be the current routine belonging to another task.

SPECIFYING A TASK ASYNCHRONOUS EXIT ROUTINE

The STAE macro instruction enables the user to specify a STAE exit routine that is entered asynchronously if the task enters abnormal termination processing. The functions of the STAE service routine module and the four ABEND/STAE interface routine (ASIR) modules are:

- The STAE Service Routine (IGC00060) -- Receives control via an SVC 60 when the STAE macro instruction is issued. Checks the validity of the STAE request, and creates or modifies a STAE control block.
- ASIR1 (IGC0B01C) -- Receives control from ABEND1. Quiesces I/O operations that are in progress for the ABENDing task, establishes a work area, and schedules the user-written STAE exit routine. If the STAE routine does not request that a STAE retry routine be executed, ASIR1 returns to ABEND processing when the STAE exit routine processing is finished. If a STAE retry routine is requested, ASIR1 invokes ASIR2. If the program using STAE is in supervisor mode and requests a STAE retry routine without a purge of the RB chain, ASIR1 invokes ASIR3.
- ASIR2 (IGC0C01C) -- Receives control from ASIR1. Closes the data sets allocated to RB of the RBs positioned between the STAE issuer up to and including the RB of the task scheduled for ABEND. Invokes the WTOR Purge routine. If any of the DCBs examined by ASIR2 are using BTAM QTAM for a line group, BISAM, or QISAM, ASIR2 invokes ASIR4. If none of the above access methods are indicated, ASIR2 invokes ASIR3.
- ASIR3 (IGC0D01C) -- Receives control from ASIR1, ASIR2, or ASIR4. Sets dispatchable, the subtasks related to the task using STAE, frees the storage occupied by the STAE control block, and schedules the STAE retry routine so that it is the next program executed.
- ASIR4 (IGC0E01C) -- Receives control from ASIR2. Repeats the search for open data sets represented by DCBs using BTAM, QTAM for a line group, BISAM, or QISAM, closes these data sets, and invokes ASIR3.

When the STAE macro instruction is issued, the resulting macro expansion places in register 0 a code indicating the desired option (create, cancel, or overlay), and, in register 1, the address of a two-word parameter list containing the STAE exit routine address and the STAE exit routine parameter list address. If the STAE request specifies that the user wants the STAE environment to remain in effect if he XCTLs to another routine, the high order bit of register 1 is set to one. The last instruction of the macro expansion is an SVC 60, which invokes the STAE service routine.

The STAE Service Routine

The STAE service routine (IGC00060) first examines the contents of the TCBNSTAE field of the TCB (displacement dec. 160). If the STAE has been issued in the STAE exit routine (the high order bit in the first byte of TCBNSTAE is on), an error code of 8 is placed in register 15, and control is returned to the user. The STAE request is not serviced in this situation, since STAE exit routine processing is already attempting to deal with an error situation.

The STAE service routine next tests the contents of register 0 to determine the option of the STAE request -- to create, cancel, or overlay a STAE control block (SCB). If register 0 contains a zero (the create option), the STAE exit routine address and the parameter list address specified in the STAE macro instruction are checked for validity. If either address is invalid, an error code of 12 is placed in register 15, and control is returned to the user.

The STAE service routine issues a conditional GETMAIN to obtain 16 bytes of storage for the SCB. The first word of the Extended Save Area of the STAE service routine SVRB is passed to GETMAIN to be used for the address of the storage that is obtained. If storage is not available, control is returned to the user with the return code of 4 in register 15. If storage is available, the SCB is created by placing the previous SCB address, or zero in the first word, the address of the STAE exit routine in the second word, the address of the STAE 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 TCBNSTAE field of the TCB.

If the XCTL option is requested in the STAE macro instruction (the high order bit of register 1 is on), the STAE service routine turns on the XCTL flag in the TCBNSTAE field.

If the contents of register 0 is not zero, or if register 0 contains a zero but the STAE exit routine address is zero, either the cancel or the overlay option is being specified. The STAE service routine tests the TCBNSTAE field to determine if an SCB already exists. If it is zero, 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 cancelled or overlayed.

The STAE service routine next compares the RB address of the current SCB with the RB address of the program that is requesting that the SCB be cancelled or overlayed. If the RB 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 unintentional destruction of another program's SCB.

The STAE service routine now determines whether the STAE request is the cancel or overlay option. If register 0 contains a 4 or a zero with a STAE exit routine address of zero, (the cancel option), the address of the previous SCB, which is contained in the first word of the current SCB, is moved into the TCBNSTAE field. A FREEMAIN is then issued to free the storage occupied by the cancelled SCB.

If register 0 contains an 8 (the overlay option), the STAE exit routine address and the STAE parameter list address are obtained and checked for validity. If either address is invalid, an error code of 12 is placed in register 15, and control is returned to the user. If the STAE exit routine address and the parameter list address are valid, they are moved into the second and third words respectively of the current SCB. If the XCTL option is specified, the XCTL option flag in the TCBNSTAE field is turned on.

When the SCB has been successfully created, cancelled, or overlayed, the STAE service routine returns control to the user with a return code of zero in register 15.

ABEND/STAE Interface 1 Routine (ASIR1)

The ABEND/STAE interface routine, load 1, (IGC0B01C) receives control from ABEND1 (IGC0001C) when ABEND1 determines that an SCB exists. ASIR1 first turns on the STAE recursion flag in the TCBNSTAE field of the TCB. This bit is tested by ABEND1 to prevent STAE processing from being invoked twice for the same error.

ASIR1 tests for several conditions before establishing a work area and scheduling the STAE exit routine. If the STAE user is in "must complete" status but is not a supervisor program, control is returned to ABEND1 and abnormal termination processing continues. If the RB address in the current SCB cannot be found on the RB chain, that SCB is cancelled and the next SCB on the chain is tested. If none of the RB addresses in any of the SCBs are on the RB chain, ASIR1 returns control to ABEND1. If an SCB is found that contains an RB address on the RB chain, that SCB is used for further processing.

If the STAE user is a supervisor routine, ASIR1 sets a bit in the TCBNSTAE field. This bit is later referenced by SYNCH to ensure that the STAE exit routine will be scheduled in the same mode as that of the user.

ASIR1 next determines if any I/O operations are in progress for the task that was scheduled for ABEND processing. If the TCBDEB field in the TCB contains a zero, no I/O operations are in progress. If the TCBDEB field contains a non-zero value, one or more data sets are open. Since I/O operations may be in progress for the task, ASIR1 sets the purge-quiesce bit in the TCBNSTAE field and invokes the Purge I/O routine with the quiesce I/O option. If the Purge I/O routine encounters an ABEND situation while attempting to quiesce I/O, ABEND1 is reentered. ABEND1 tests the purge-quiesce bit and returns control to ASIR1. If the Purge I/O routine did not successfully quiesce I/O, the halt I/O bit in the TCBNSTAE field is set to indicate that I/O is not restorable, and the Purge I/O routine is reinvoked with the halt I/O option. Upon return from the Purge I/O routine, if I/O operations were halted, or if the Purge I/O routine was not called, the first word in the extended save area (ESA) of the SVRB is set to zero. If the Purge I/O routine has successfully quiesced I/O, the address of the first I/O block (IOB) on the IOB restore chain is placed in the first word of the ESA by the Purge I/O routine for later restoration of I/O by the user.

ASIR1 next attempts to get 176 bytes of storage for a register save and work area by issuing a conditional GETMAIN macro instruction. The request is conditional because the STAE processing can continue if storage cannot be obtained. If storage is not available, registers 0, 1, and 2 are initialized as parameter registers. A 12 is placed in register 0, the ABEND completion code that appears in the TCBCMP field in register 1, and the address of the STAE exit routine parameter list in register 2.

If storage for the work area is obtained, the starting address of this area is placed in the ESA of the SVRB and in register 1 to be passed to the STAE exit routine. ASIR1 initializes the work area with system status information at the time the ABEND was scheduled. The address of the STAE exit routine parameter list is placed in word 1; the ABEND completion code found in the TCBCMP field in word 2; the PSW at the time of the ABEND in words 3 and 4; and the problem program PSW before the ABEND occurred, or 0 if the task is a supervisor task, in words 5 and 6. Words 7 through 22 contain the user's registers at the time of the ABEND. If the STAE user is a supervisor task, the RB address of the ABENDing program is placed in word 23, and zeros in words 24 through 26. If the STAE user is a problem program, the program name, or zero if the name cannot be found, is moved into words 23 and 24, the address of the entry point of the ABENDing program into word 25, and zero into word 26. The starting address of the remaining 72 bytes of the work area is placed in register 13, to be passed to the STAE exit routine and used as a register save area.

Based on the results of the Purge I/O routine, ASIR1 places in register 0 a zero if I/O operations have been quiesced, a four if active I/O operations were halted, or an eight if no I/O operations were in progress at the time of the ABEND. The ABTERM bit in the TCBFLGS field is set to zero so that if a subsequent ABEND situation occurs, the associated completion code can be stored in the TCBCMP field.

ASIR1 effects the scheduling of the STAE exit routine by issuing a SYNCH macro instruction, which creates an RB for the STAE exit routine.

When STAE exit routine processing has completed, control is returned to ASIR1 unless the STAE exit routine has requested, or encounters, an ABEND situation. ASIR1 first frees the last 76 bytes of the work area (the user's register save area) via a FREEMAIN macro instruction. Register 15 is examined to determine if the STAE user indicated that a STAE retry routine be scheduled. If register 15 contains a 0, the STAE user has not provided a STAE retry routine. ASIR1 returns control to ABEND1 via the EXIT SVC instruction and abnormal termination processing continues as originally scheduled.

If register 15 contains a four, the STAE user has requested that a retry routine be scheduled and that the RB chain be purged. The address of the STAE retry routine, passed to ASIR1 in register 0, and the address of the STAE retry routine parameter list, placed in the first word of the work area by the STAE exit routine, are checked for validity. If either is invalid, control is returned to ABEND1 as previously described. If both addresses are valid, ASIR1 passes information contained in the ESA to the other ASIR modules by placing the address of the first IOB on the restore chain or zero in register 7, the address of the work area or zero in register 8, the address of the STAE retry routine in register 10, and, if the STAE user is a problem program, the name of the program scheduled for ABEND in registers 11 and 12, and the entry point address of that program in register 13. ASIR1 then invokes ASIR2 via the XCTL macro instruction.

If register 15 contains an eight, the STAE user has requested that a retry routine be scheduled and that the RB chain not be purged. If the STAE user is not a supervisor program, as indicated by the supervisor bit in the TCBNSTAE field, control is returned to ABEND1, since the option of not purging the RB chain is reserved only for supervisor program. If the STAE user is in supervisor mode, information is stored in the parameter registers, as described above. ASIR1 then sets the NORBPG flag in the TCBNSTAE field and invokes ASIR3 via the XCTL macro instruction.

ABEND/STAE Interface 2 Routine (ASIR2)

The ABEND/STAE interface routine, load 2, (IGC0C01C) receives control from ASIR1 (IGC0B01C) when ASIR1 determines that the RB chain must be purged and the STAE retry routine scheduled. Upon entry, ASIR2 stores the contents of the parameter registers 7, 8, 10, 11, 12, and 13 in the ESA.

To determine if any I/O operations are in progress for tasks represented by RBs that are between the RB of the program issuing STAE and the RB of the program scheduled for ABEND, the TCBDEB field is tested. If the TCBDEB field is zero, no I/O is in progress, and the WTOR Purge routine can be called immediately. If the contents of the TCBDEB field is not zero, ASIR2 determines if the RB address of the ABENDing program is the same as the RB address of the program that issued the STAE macro instruction. If the RB addresses are equal, the ABENDing program is the program that issued STAE, and no intervening RBs exist. In this case also, thw WTOR Purge routine can be invoked immediately.

If the TCBDEB field is not zero and the RB addresses of the STAE issuer and the ABENDing task are not equal, ASIR2 must determine if any open data sets are associated with any of the intervening RBs. The search is accomplished by determining if the addresses of any of the DCBs on the related DEB chains are contained in the boundaries of a program represented by one of the intervening RBs. The first intervening RB tested is that of the program scheduled for ABEND. If an open DCB associated with one of the intervening RBs is found, ASIR2 determines from the DCBDSORG field if the access method being used is BTAM, QTAM, for a line group, BISAM, or QISAM. If the DCB is using one of these access methods, the ISAM/TAM switch is set to indicate that ASIR4 must be the next module invoked to complete the close DCB processing. ASIR2 continues the search by examining the next DCB.

If an access method other than BTAM, QTAM for a line group, BISAM, or QISAM is used for the DCB to be closed, ASIR2 must ensure that the user will not attempt to restore I/O events associated with this DCB. If no I/O operations were in progress when the Purge I/O routine was called in ASIR1, or if I/O operations were halted by the Purge I/O routine, I/O is not restorable, and the DCB in question is closed without further processing. If the I/O operations are restorable, an I/O event related to the DCB to be closed may be queued on the IOB restore chain that was created by the Purge I/O routine. Depending on the access method used, ASIR2 determines the addresses of the IOBs related to the DCB and compares them with the addresses of the IOBs on the restore chain. If they are equal, the IOBs on the restore chain are dequeued. The DCB is then closed via the CLOSE macro instruction. The search continues until all DCBs associated with intervening RBs have been closed and all IOBs related to these DCBs have been removed from the IOB restore chain.

When the DCB search reaches the RB of the STAE issuer, or if this search was not necessary, ASIR2 invokes the WTOR Purge routine. The address of this routine is obtained from the secondary CVT. Upon return from the WTOR Purge routine, parameter registers 7, 8, 10, 11, 12, and 13 are initialized as previously described for ASIR1. If the ISAM/TAM switch is on, indicating that the ASIR2 found one or more DCBs using BTAM, QTAM for a line group, BISAM, or QISAM during the DCB search, ASIR2 invokes ASIR4 which repeats the DCB search. If the switch has not been set, ASIR2 invokes ASIR3 via the XCTL macro instruction.

ABEND/STAE Interface 3 Routine (ASIR3)

The ABEND/STAE interface routine, load 3, (IGC0D01C) receives control from one of the following ASIR modules:

- ASIR1 (IGC0B01C) when the STAE exit routine has requested that a STAE retry routine be scheduled but that the RB chain not be purged. (The user of STAE must be a supervisor program.)

- ASIR2 (IGC0C01C) when all DCBs associated with RBs that exist between the RB of the STAE issuer and the RB of the task scheduled for ABEND have been closed.

- ASIR4 (IGC0E01C) when all DCBs (using BTAM, QTAM for a line group, BISAM, or QISAM) associated with RBs that exist between the RB of the STAE issuer and the RB of the task scheduled for ABEND have been closed.

ASIR3 stores the contents of the parameter registers in the ESA. Since this task is to be reestablished when the STAE retry routine is given control, all the subtasks associated with the task scheduled for ABEND must be set dispatchable. (The associated subtasks were previously set nondispatchable if abnormal termination processing was entered from ABATERM.) The TCBLTC field, which contains the address of the last TCB on the subtask queue, is referenced to identify the associated subtasks. If the TCBLTC field is zero, no associated subtasks exist. If the field contains the address of a TCB, the nondispatchability flags are turned off for that TCB. The addresses of the TCBs of other associated subtasks are obtained from the TCBNTC field. These TCBs are also set dispatchable.

If the NORBPG flag (set by ASIR1) in the TCBNSTAE field is not on, indicating that the user of STAE requested a purge of the RB chain, ASIR3 must purge the RBs on the chain that exist between the RB of the STAE issuer and the RB of the program scheduled for ABEND. The purge is accomplished by setting the RBOPSW of these RBs to point to an SVC 3 instruction located in the CVT. Since the STAE retry routine will run under the RB of the STAE user, a new RB need not be created.

If the NORBPG flag is on, the user has not requested a purge of the RB chain. An RB must be built for the STAE retry routine. ASIR3 issues an unconditional request for 32 bytes of storage via the GETMAIN macro instruction. The PSW is set to reflect the same mode as that of the STAE user, and the pointer fields in the RB and the TCB are set so that the STAE retry routine is the next program executed after ASIR3 exits. The user's register 14 is set to point to an SVC 3 instruction in the CVT.

Before scheduling the STAE retry routine, the TCBNSTAE field is updated to point to the previous SCB. The SCB that is currently being processed is freed via the

To determine if a work area was obtained by ASIR1, the work area address in the ESA of the SVRB is tested. If the address is zero, a work area was not obtained, and ASIR3 must initialize parameter registers to be passed to the STAE retry routine. A 12 is placed in register 0, the ABEND completion code in register 1, and the address of the first IOB on the restore chain, or 0, in register 2. If a work area was established by ASIR1, it is reinitialized with system status information as it was in ASIR1 except that the second word of the work area now contains the address of the first IOB on the restore chain, and the last word now contains an address to be passed to the Restore routine for restoring purged I/O.

The RBOPSW of the STAE user is set to point to the address of the STAE retry routine, and the ABTERM and prevent asynchronous exit flags in the TCB are cleared. ASIR3 issues an EXIT macro instruction and gives control to the dispatcher which will schedule the STAE retry routine.

ABEND/STAE Interface 4 Routine (ASIR4)

The ABEND/STAE interface routine, load 4 (IGC0E01C) receives control from ASIR2 (IGC0C01C) when ASIR2 has set the ISAM/TAM switch, indicating that, during the DCB search, one or more DCBs using BTAM, QTAM for a line group, BISAM, or QISAM were found. ASIR4 repeats the search made by ASIR2 and closes the DCBs, using BTAM, QTAM, BISAM, or QISAM, that are related to RBs that exist between the RB of the STAE issuer and the RB of the program scheduled for ABEND processing. As in ASIR2, the RB of the program scheduled for ABEND is examined first. The access method of all DCBs on the DEB chains related to that RB are tested. If the related DCBDSORG field indicates that the DCB is using BTAM, QTAM for a line group, BISAM, or QISAM, ASIR4 determines if that DCB is related to the intervening RB. If it is, any I/O events related to that DCB are removed from the IOB restore chain. The DCB is then closed. The search continues until all DCBs associated with intervening RBs have been tested, the related IOBs have been removed from the IOB restore chain, and the associated DCBs have been closed. The search ends when the RB of the STAE issuer is reached. ASIR4 initializes the parameter registers and invokes ASIR3 via the XCTL macro instruction.

SERVICES INTERNAL TO THE SUPERVISOR

Supervisor internal services consist of testing and indicating the need for a task switch, testing the validity of user-supplied addresses, and changing the status of tasks. In a multiprocessing system, additional supervisor internal services include determining the relative priority of tasks, testing the dispatchability of tasks, and initiating an external interruption in a second CPU.

TESTING AND INDICATING THE NEED FOR A TASK SWITCH

The Task Switching routine is one of the subroutines used by a number of supervisor routines. The routine determines whether a newly readied task, which may be of higher priority than that of the caller's, should be dispatched in place of the caller's task.

The routine is entered if a supervisor routine has reduced to zero a program's RB wait count, or has cleared a non-dispatchability flag in a TCB. For example, the Post routine may make ready a program that was awaiting the completion of an I/O operation, or the DEQ routine may make ready a program that was awaiting a serially reusable resource. In either case, the supervisor routine does not know if the readied routine belongs to a task ofhigher priority than that of the caller, and whether it should replace the caller as the currently dispatchable program.

To answer this question, the supervisor routine branches to the Task Switching routine. The Task Switching routine compares the dispatching priority of the readied routine's TCB with that of another TCB. The other TCB is either the caller's TCB or the TCB for another readied routine, if more than one routine has just been readied (e.g., during DEQ processing). According to the result of the comparison, the Task Switching routine places the address of the higher priority TCB in the "new" TCB pointer IEATCBP.[1] Later the Dispatcher will consult this TCB pointer to determine the task and routine it should dispatch.

The operation of the Task Switching routine, just broadly discussed, will now be described in greater detail.

Upon branch entry from the calling supervisor routine, the Task Switching rou-

[1] The address of the "new" TCB pointer is in the communications vector table (CVT) at location CVTTCBP.

tine compares the dispatching priority of the readied routine's TCB, passed as an input parameter, with the dispatching priority of another TCB. The address of the other TCB -- either the current TCB or the TCB of a recently readied routine -- is stored in either half of the doubleword TCB pointer at location IEATCBP. The address stored in the first word, the "new" TCB pointer, has two possible values: zero, or the address of a TCB for a previously readied task.

If the first word of the TCB pointer contains zero, the Task Switching routine compares the dispatching priority of the current TCB with that of the TCB for the newly readied routine. But if the first word of the TCB pointer is not zero, the routine compares the dispatching priority of a previously readied task, whose TCB address is in the "new" TCB pointer, with the dispatching priority of the TCB for the newly readied routine. (In this case, the Task Switching routine has been invoked more than once after the same interruption.) If the priority of the newly readied task is higher than that of the other, the Task Switching routine stores the address of the readied TCB in the "new" TCB pointer (IEATCBP) and returns control to the invoking routine. Later the Dispatcher will dispatch the current routine whose TCB address has been placed in the "new" TCB pointer. But if the priority of the readied (input) TCB is lower than that of the other TCB, the Task Switching routine does not change the TCB pointer. It merely returns control to the calling supervisor routine. In this case, the Dispatcher, when it gains control, will dispatch the current routine for any of three possible ready tasks: the current task, if the "new" TCB pointer (IEATCBP) points to the current TCB; a previously readied task, if a previous use of the Task Switching routine has placed the TCB address in IEATCBP; or another task found by a scan of the TCB queue, if IEATCBP contains zero.

A special case exists in which the Task Switching routine cannot make a comparison between TCB priorities. This is the case if the two TCBs have the same dispatching priority. If the time-slicing feature is included in the system, the Task Switching routine tests the time-slice bit (TCBFTS) in the TCB. If the bit is set, the Task Switching routine returns control to the calling supervisor routine without changing the TCB pointer. In this case, the Task Switching routine must search down the TCB queue to discover which TCB is at a higher relative position on the queue. It begins its search with the TCB pointed to by IEATCBP or, if this location contains zero, with the current TCB. The address of the input or newly readied TCB is stored in the "new" TCB pointer only if the input TCB is not found below the other TCB on the queue. Otherwise, the routine does not change the TCB pointer.

In a multiprocessing system, the Task Switching routine also determines if the newly readied task should be dispatched in place of the current task on the second CPU.

If the first word of the TCB pointer of either CPU contains zeros, zeros are also placed in the first word of the second CPU's TCB pointer. Later, the Dispatcher will search from the top of the TCB queue to find the two highest priority ready tasks.

If the first word of the TCB pointer of neither CPU contains zeros, control is passed to the Relative Priority routine (RELPRIOR) TO compare the dispatching priorities of the two TCBs whose addresses are in the first words of the TCB pointers. The TCB with the lower dispatching priority is compared with the newly readied TCB. If the priority of the newly readied task is higher, the address of the readied TCB replaces the address of the other TCB in the first word of the TCB pointer.

The Task Switching routine then determines if the TCB to be dispatched on the executing CPU is the current TCB on the second CPU or vice versa. If so, the addresses in the first word of each TCB pointer are interchanged.

The Task Switching routine then returns control to the calling supervisor routine. Later the Dispatcher will decide the program to be executed, as described above.

TESTING THE VALIDITY OF USER-SUPPLIED ADDRESSES

Supervisor routines use the Validity Check routine as a subroutine to check main storage addresses passed as input parameters by user programs. The Validity Check routine tests the following attributes of each input address: fullword boundary alignment (optional), whether the address lies within the boundaries of main storage, and if the address specified a storage area whose storage protection key matches the protection key in the TCB of the calling main line program. If any of these tests fails, the routine informs the invoking supervisor routine by altering the condition code of the current PSW. Since the calling main line program has made a serious error, the invoking supervisor routine 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.

CHANGING THE STATUS OF TASKS

Supervisor-mode routines can use the Set Status routine (IGC079) to set or reset the status of particular tasks. The affected task status can be either the "nonrolloutable" status, the "must complete" status, or the "nondispatchability" status. (A supervisor-mode routine operates under control of a TCB whose supervisor-mode flag (TCBFSM) is set. Such a routine belongs to the supervisor or the Master Scheduler.)

The Set Status routine is invoked, via supervisor linkage (SVC 79), through use of the STATUS macro instruction. The routine is entered either from the SVC First-Level Interruption Handler, or via a branch from a Type-1 SVC routine. (A type-1 SVC routine may not cause an SVC interruption.) Control is returned to the caller via the Type-1 Exit routine.

The Set Status routine sets (or resets) the following conditions for a task or a group of tasks:

- "Nonrolloutable" status, so that the tasks of the job step are ineligible (or eligible) to be rolled out.
- "Must complete" status, so that other tasks of the job step or system are made nondispatchable (or dispatchable) while the current task is being performed.
- "Nondispatchability" status, so that the routines of the tasks cannot (or can) be restarted by the Dispatcher.

Setting or Resetting the Nonrolloutable Status

When entered via the macro instruction STATUS SET, NR, the Set Status routine adds 'one' to the "nonrolloutable count" (TCBNROC) in the job step TCB. The job step TCB is either that associated with the specified TCB, or that associated with the caller's TCB (if 'S' or no TCB address is specified). The "nonrolloutable count" is later tested by the rollout/rollin module to determine if the job step is eligible to be rolled out. A job step is eligible to be rolled out if its "nonrolloutable count" is zero.

When entered via the macro instruction STATUS RESET, NR, the routine subtracts 'one' from the "nonrolloutable count" in the job step TCB. (The particular job step is defined in the previous paragraph.) The Set Status routine schedules linkage to the rollout/rollin module to restart deferred rollout requests, if the "nonrolloutable count" becomes zero while there is at least one element on the rollout request queue (IEAROQUE). If such scheduling is needed, the routine obtains an interruption queue element (IQE), via the GETIQE routine in module IEAQPRT0, and branches to the Stage-2 Exit Effector to place the IQE on the asynchronous exit queue (AEQJ).

Setting or Resetting the "Must Complete" Status

When entered via the macro instruction STATUS SET, MC, [STEP][SYSTEM], the Set Status routine sets the caller's task in "step" or "system" must complete status. (If the RESET operand is specified, the "must complete" status that was prevously set is cleared.) The routine sets the "must complete" flag in the current TCB, the "prohibit asynchronous exits" flag in the current TCB, and the step or system "must complete" nondispatchability flag in other TCBs of the job step or system. (For the names and meanings of these flags, see Table 3-5 in "Serializing the Use of a Resource.")

In a multiprocessing system, after the caller's task has been set in "must complete" status, control is passed to the Task Removal subroutine. The Task Removal routine (TESTDSP) determines whether the current task on the second CPU has been set nondispatchable, and, if it has, interrupts the second CPU with an indication (in STMASK) that the Dispatcher routine must gain control. If the RESET operand is specified, after the "must complete" status is cleared, the Set Status routine indicates to the Dispatcher that the TCB queue must be searched from the top to find the two highest priority ready TCBs. This is done by setting the "new" TCB pointer (IEATCBP) of both CPUs to zero.

Setting or Resetting Nondispatchability

When entered via the macro instruction STATUS SET, ND, [STEP][SYSTEM][tcbloc-addrx],(nn), the Set Status routine sets the specified nondispatchability flag or flags in the specified set of TCBs. (If RESET is specified, the specified nondispatchability flag or flags are cleared in the specified set of TCBs.) Three sets of tasks can be specified: the system, the job step, or a specified task and its descendants. If SYSTEM is specified, all tasks of the system are set nondispatchable

• Table 3-7. Mask Bit Numbers Used in the STATUS Macro Instruction

Mask Bit Number	Flag Name	Offset of Flag in TCB	Meaning of Flag
1	TCBNDUMP	32.0	This task is nondispatchable while the resources of a task in this job step are being dumped.
2	TCBSER	32.1	This task is nondispatchable while the SER1 routine is being executed for this task.
3-5			reserved.
6		32.5	This task is nondispatchable while VARY or QUIESCE processing is being performed in a multiprocessing system.
7-8			reserved
9	TCBFC	33.0	This task is nondispatchable because it has been normally or abnormally terminated.
10	TCBABWF	33.1	This task is nondispatchable as part of a tree of tasks that is being abnormally terminated.
11	TCBWFC	33.2	This task is nondispatchable because it is waiting for requested storage space.
12	TCBFRO	33.3	This task is nondispatchable because it is part of a rolled out job step.
13	TCBSYS	33.4	This task is nondispatchable while another task in the system is in "system must complete" status.
14	TCBSTP	33.5	This task is nondispatchable while another task in the same job step is in "step must complete" status.
15	TCBFCD1	33.6	This task is nondispatchable because it is an initiator that is waiting for a requested region of main storage.
16			reserved

except the current task and the permanent system task.[1] If STEP is specified, all tasks of the job step are set nondispatchable except the current task and the job step's initiator. If a TCB address (tcbloc-addrx) is specified, the task and its descendants are set nondispatchable.

The particular nondispatchability flag or flags that are set (or cleared) in each TCB depend on the mask bit number (nn) specified in the STATUS macro instruction. (See Table 3-7.)

In a multiprocessing system, after the nondispatchability flags have been set, control is passed to the Task Removal (TESTDSP) subroutine which determines whether the current task on the second CPU has been set nondispatchable. If it has, the second CPU is interrupted with an indication (in STMASK) that the Dispatcher routine should gain control.

DETERMINING THE RELATIVE DISPATCHING PRIORITIES OF TASKS

The Relative Priority subroutine (entry point RELPRIOR) is used by the Task Switching and Dispatcher routines in a multiprocessing system to determine which of two TCBs has the higher dispatching priority. Condition codes indicate the results as follows:

Code	Indication
0	They are the same TCB.
1	Second TCB has higher priority.
2	First TCB has higher priority.

[1] The permanent system tasks are: the transient area fetch tasks, the system error task, the rollout/rollin task (if the rollout feature is present), the communications task, and the master scheduler task.

If the dispatching priority of both TCBs is equal, the RELPRIOR routine searches down the TCB queue, starting with the first TCB being compared, to determine which TCB is at a higher position on the queue. The TCB higher on the queue will have the higher dispatching priority.

TESTING THE DISPATCHABILITY OF TASKS

The Task Removal subroutine (entry point TESTDSP) ensures, in a multiprocessing system, that a task which has been set nondispatchable by a routine on one CPU does not continue to run on the second CPU. The address of TESTDSP is contained in the multiprocessing CVT.

The Task Removal routine first determines if the TCB whose address is in the "old" TCB pointer (IEATCBP+4) for the second CPU has been set nondispatchable. If it has, the First CPU Signal routine is invoked to cause the Dispatcher routine to gain control on the second CPU. After the Dispatcher on the second CPU has stored the status of the "old" TCB, the Task Removal routine determines if the "old" TCB for this CPU or the "new" TCB for either CPU has been set nondispatchable. If so, the "new" TCB pointers (IEATCBP) for both CPUs are set to zero. This causes the Dispatcher to search from the top of the TCB queue to find the two highest ready tasks.

INITIATING AN EXTERNAL INTERRUPTION IN A SECOND CPU

The First CPU Signal and SHOLDTAP subroutines are used by supervisor routines in a multiprocessing system to cause an external interruption in the second CPU. As a result of the external interruption, a routine specified in the word STMASK receives control on the second CPU (see description of External FLIH routine). The word STMASK is located in the prefixed storage area, and the bit designating the routine to receive control on the second CPU is set in STMASK by the calling routine. The address of the SHOLDTAP routine is contained in the multiprocessing CVT.

The SHOLDTAP routine first tests the pending bit, bit 0 in the STMASK byte, to determine if the previous external interruption has been processed and the bit reset to zero by the External FLIH routine. If it is set to 1, the interruption has not been processed, and control is returned to the calling routine. Otherwise, bit 0 is set to 1, and a WRITE DIRECT instruction is issued. This instruction causes an external interruption in the second CPU.

The First CPU Signal routine (entry point FLASH) is used when the second CPU must perform an immediate service for the first CPU. After issuing a WRITE DIRECT instruction, the First CPU Signal routine tests the word STMASK to determine if the external interruption has been processed and the immediate service performed. (The appropriate bit in STMASK is cleared by the External FLIH routine after the service has been performed.) Control is returned to the calling routine only after the immediate service has been performed.

The contents supervision feature of the supervisor determines the location of requested programs, fetches the programs to main storage if necessary, and schedules the execution of these programs for their tasks. As a byproduct of these functions, records are kept of all programs in main storage.

Contents Supervision consists of two types of functions: common functions and special functions. The common functions satisfy requests for linkage to a module or requests to fetch a module to main storage for future use. These common functions are requested by the LINK, LOAD, XCTL, SYNCH, and ATTACH macro instructions. These functions are performed by a group of subroutines that will be called the "common subroutines." The special functions satisfy a particular request from a system or user routine, or assist one of the common functions. Examples of special functions are the identification of an embedded module entry point, or the loading of a segment of a module in overlay mode.

The common functions consist of:

- Searching for the requested module in the contents directory.
- Creating, if necessary, a contents directory entry (CDE) to describe the requested module, placing descriptive information in the CDE from the input parameters of the request, and queuing the CDE on the appropriate contents directory queue.
- Testing the module's status to determine if it is available for use. The module's status is tested if a CDE is found in one of the contents directory queues or if a BLDL procedure is performed for the module.
- Causing the fetching of a module that is not in main storage or that is not reusable.
- Determining the relocated alias entry point and updating the appropriate contents directory queue if the module request specifies an alias entry point.
- Deferring the request if the module is not available.
- Restarting a deferred request when the module becomes available.
- Scheduling execution of the module by creating a program request block (PRB), and placing it behind the current SVRB on the caller's RB queue.

The special functions are used to assist one of the common functions or to perform a specialized service. Special processing is performed for a LOAD request, and for an XCTL request issued by an SVC routine. Through the servicing of an IDENTIFY request, the supervisor is informed of an embedded entry point within a specified module. Through the servicing of a DELETE request, the supervisor is informed that a module fetched because of a LOAD request is no longer needed in main storage. If a module must be loaded in overlay mode, the Overlay Supervisor is invoked to prepare for and control the loading of the appropriate segments. Lastly, the actual loading of a module, although requested by other contents supervision components, is performed by the Program Fetch routine. This routine acts as a loader for Contents Supervision, the Transient Area Fetch routine, the Overlay Supervisor, and the Stage 3 Exit Effector.

THE COMMON FUNCTIONS OF CONTENTS SUPERVISION

The first part of this section describes the common functions in the sequence in which they are performed by the supervisor. The second part of the section describes each major function in greater detail in the logical order previously listed.

GENERAL DESCRIPTION OF THE COMMON FUNCTIONS

The common subroutines are entered from the SVC SLIH because of a LINK, LOAD, XCTL, or ATTACH request. If the entry is because of an ATTACH request, the program for which linkage is desired is the first program to be executed for the new subtask, as specified by the ATTACH macro instruction. (See "Attaching a Subtask" in Section 3, "Task Supervision.")

Contents Supervision performs initialization and input processing peculiar to the type of module request. Then the request is serviced by a group of common subroutines which locate the requested module, determine its status, and test whether it is available. A module is available if it is in main storage and is either reenterable, or serially reusable but not in use, or is nonreusable but not yet used. If the module is available, its execution is scheduled. If it is not available, it is fetched from auxiliary storage and then scheduled. If, however, the module cannot be fetched, the request is deferred.

In systems that include Main Storage Hierarchy Support, contents supervision service routines for LINK, LOAD, XCTL, and ATTACH requests direct program loading into the appropriate hierarchy in main storage. These service routines, upon entry from the SVC SLIH, extract the hierarchy number from the parameter list and, if a copy of the requested program is to be loaded, pass the number to the Program Fetch routine. The GETMAIN request later uses the number when it allocates storage for program loading.

If hierarchy is not specified in the LINK, LOAD, XCTL, or ATTACH request, the Program Fetch routine loads the program into the hierarchy or hierarchies as stated in the scatter table. The hierarchy number (0 or 1) is included in the GETMAIN request issued for each CSECT of the requested module.

Allocation of an Available Module

If a module is available for immediate allocation, the "use/responsibility" count, which records the number of outstanding requests for the module, is increased and the module is "allocated" to the requestor. The expression "allocated" means different things, depending on the type of request. For a LOAD request, allocation means ensuring that a load-list element exists for the request. A load-list element represents one or more LOAD requests for the module. It contains a "responsibility count" of the number of outstanding LOAD requests for the module, and a pointer to the contents directory entry which describes the module. For other types of requests, allocation (in this case scheduling) means creating a program request block (PRB) which will control the module's execution, then placing the PRB on the caller's RB queue, and initializing the PRB's fields. After either type of allocation is complete, the appropriate subroutine, via the Exit routine and the Dispatcher, passes control to either the requested module or the caller.

Deferring the Request for an Unavailable Module

If a module is unavailable, it cannot be immediately allocated to its requestor. A module is unavailable if it is being fetched because of a previous request, or if it is a serially reusable module that is in use. In either case, the SVRB under whose control the supervisor is operating is placed on a list of waiting SVRBs. This list represents requests for the module which cannot yet be serviced. Subroutine CDQUECTL places the SVRB in a wait condition, ensures a task switch (since processing for the current task cannot proceed), and branches to the Dispatcher to give control to the current routine of another task.

Preparing to Fetch a Module

If a module is not in the link pack area or in the job pack area for the requestor's job step, or is nonreusable and has already been used, a new copy must be fetched from auxiliary storage. The search of the appropriate library requires the retrieval of the data set directory entry, whose location may be indicated by parameters in the caller's macro instruction. The data set directory is obtained via the BLDL routine of data management. When the module is located, its attributes are recorded in a contents directory entry (CDE) that was built and initialized before the execution of the BLDL routine.

If the data set directory entry indicates that the caller has specified an alias entry point, special processing is performed. This processing includes:

- Determining if the module is already in main storage.
- Calculating a relocated entry point address.
- Ensuring that there are two CDEs for the module, one containing the main entry point name, the other containing the alias entry point name.

In systems generated with storage hierarchies, the expansions of the LINK, LOAD, XCTL, and ATTACH macro instructions include a one-byte "hierarchy ID" value. This value is derived as follows:

Value	Derivation
00	No hierarchy specified
01	Hierarchy 0 specified (HIARCHY=0)
02	Hierarchy 1 specified (HIARCHY=1)

For LINK, XCTL, and ATTACH, the hierarchy identification appears as the high-order byte of the second full word of the parameter list pointed to by register 15. For LOAD, the hierarchy ID is passed in the high-order byte of register 1. When the Program Fetch routine is entered, the ID is placed into the high-order byte of register 5, which points to the address of BLDL.

Fetching the Module

After preparation for fetching the module is complete, control is passed to the Program Fetch routine to load the module into main storage. The hierarchy identification is checked. The Program Fetch routine then computes the module's relocated entry point address. A common subroutine stores the address in the module's CDE for use in future linkage to the module. If the data set directory information obtained from the BLDL procedure indicates that the module is in overlay mode or contains TESTRAN symbol records, other routines are invoked. If the module is in overlay mode, contents supervision issues a LOAD macro instruction to load nonresident routines of the Overlay Supervisor (IEWSZOVR), in preparation for later linkage to these routines. If the module contains TESTRAN symbol records, the TESTRAN routines are invoked via an SVC 61 instruction.

Updating the Contents Directory

Next, a check is made to determine if the relocated entry point returned by the Program Fetch routine is an alias entry point, and therefore has been stored in a "minor" contents directory entry (CDE). If this is so, the relocated main entry point is calculated and stored in the "major" CDE. In addition, if there are other minor CDEs for the module (meaning that there are other alias entry points), relocated entry points are calculated for all minor CDEs pertaining to the module. Thus, all pertinent CDEs pertaining to the module are updated to contain relocated entry points.

Restarting Deferred Requests

After calculating relocated entry points for the contents directory, the subroutines prepare deferred requests to compete for a new search for their desired module. Any other request blocks (RBs) queued to the module's CDE are removed and made ready. The RB belonging to the highest priority ready TCB will control the resumed search for the module, beginning at entry point CDCONTRL.

After the RBs are made ready, a branch to the Task Switching routine occurs in order to test if any of the readied RBs may, the next time the Dispatcher is entered, replace the current RB as the controller of the module. The common subroutines later test whether the Task Switching routine has indicated the need for a task switch. This test occurs just before a PRB is constructed to schedule the execution of the module for the current task. For a LOAD request, the test occurs in the Dispatcher.

DETAILED DESCRIPTIONS OF THE COMMON FUNCTIONS

Thus far, the general discussion of "Contents Supervision" has followed the sequence of processing used by the supervisor. Each major function will now be described in greater detail, but not necessarily in the exact sequence in which it occurs. For an aid in visualizing the time relationships between the major functions, the reader may refer to the flowcharts for LINK, LOAD, XCTL, and SYNCH processing in Section 11.

Searching for the Module

The first function of Contents Supervision is to search for the desired module. The module may be in any of several locations: the job step's region of main storage, one of the libraries of auxiliary storage, or the link pack area of main storage. Contents Supervision first searches the job step's region, then (if appropriate) the libraries of auxiliary storage, and lastly the link pack area of main storage.

Initially, for all module requests except SYNCH,[1] subroutine CDSEARCH searches the job step's region. In the region, modules are assigned to subpools loosely called a "job pack area." Subroutine CDSEARCH searches for the module in the job pack area by examining a contents directory queue called a job pack area control queue. Each job step in the system has its own job pack area control queue (JPACQ).[2] Each JPACQ contains contents directory entries (CDEs) that represent user modules in the region's job pack area. These modules may be used only by the job step in whose region they are stored. Subroutine CDSEARCH examines each CDE in the job step's JPACQ, seeking a match between the module name supplied as an input parameter and the module name contained in the CDE.

For a LOAD request, before the examination of the JPACQ, another subroutine (CDLLSRCH) searches the load list for the caller's task. The load list contains elements, each of which points to a CDE for a module that was loaded for the task, via a LOAD macro instruction. The subroutine examines each CDE pointed to by a load list element, looking for a name match, as described above. (See Figure 4-1.) There are thus initially two ways to find a module's CDE: a search of the job pack

----

[1] For a SYNCH request the supervisor assumes that the module is in main storage, and does not search for the module.

[2] The list origin for the JPACQ is the TCBJPQ field of the job step TCB.

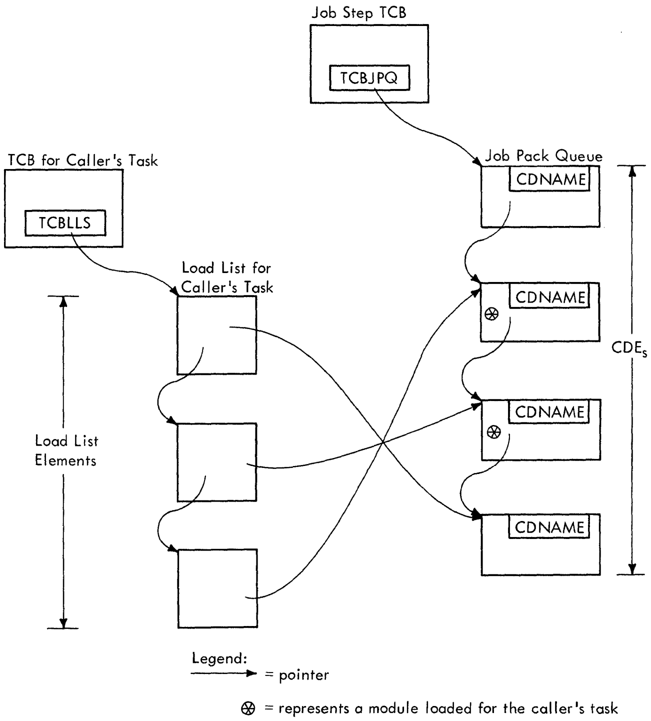


Figure 4-1. Subroutine CDSEARCH Uses the Load List and the Job Pack Queue in its Search for the Module's Name

area control queue, or a search of the task's load list.

If a CDE is found whose entry point name matches that supplied as an input parameter, the module must be in the job step's region of main storage. Control is then given subroutine CDALLOC to test the status of the "found" module. The module is either immediately available, not immediately available, or is not available at all (meaning that a new copy must be fetched).

If subroutine CDSEARCH cannot find the required CDE in the JPACQ, it recognizes that the desired module is not in the job pack area, and branches to subroutine CDSETUP to continue the search for the module. (See Figure 4-2.)

According to the contents of the DCB parameter, an operand of the requesting macro instruction, the directory of the appropriate library is searched. Supervisor linkage to the BLDL routine of data management causes the loading of a directory entry from the specified data set for examination by a subroutine of Contents Supervision. If the DCB parameter is zero, meaning that a library is not specified, the directory of the job library, if one is present, is searched. Otherwise, the directory of the library specified by the DCB parameter is searched. If the desired module (actually the entry point name of the module) is still not found, subroutine CDSEARCH examines the other contents directory queue, called the link pack area control queue (LPACQ).

The LPACQ contains CDEs describing the modules normally resident in the link pack area of main storage. Modules in this area are loaded by the nucleus initialization program (NIP). They may be shared by various job steps in the system. If the search of the LPACQ does not locate the module's name, the next step is to search, via the BLDL procedure, the directory of the link library. If the entry point name is not found in this directory, the assumption is that the caller has made an incorrect request. Accordingly, one of the common subroutines sets up an error code (806) and issues an ABEND macro instruction to obtain linkage to the ABEND SVC routine to abnormally terminate the caller's task.

Creating a Contents Directory Entry

During the search for the module, just before the preparation for the BLDL procedure, subroutine CDSETUP determines if a CDE exists. It does this by testing whether a BLDL work area was created during a previous request for the module. If the module's CDE does not exist, space is obtained by the GETMAIN SVC routine. The CDE is then initialized and placed on the JPACQ. The "attributes" field (CDATTR) is initialized so that all bits are set. Later, after the BLDL routine has obtained the module's data set directory entry, the common subroutines will clear the bits that are not applicable to the module's status. The entry point address field and the extent list address field are initialized to zero. (The extent list describes the entry point and size of each loadable section.) See Section 12, "Controls Blocks and Tables" for a description of the CDE fields.

Testing Module Status

There are two distinct times that the attributes of a module may be checked in order to determine its status. One time is after a CDE is found in the JPACQ or the LPACQ. In this case, the status-checking subroutine (CDALLOC) checks the bits in the attributes field of the CDE. Its purpose is to determine if the module can be immediately allocated; whether the request must be deferred and placed on a queue of waiting requestors; or whether a new copy of the module should be fetched to main storage. The other time that the module's attributes may be tested is after the execution of the BLDL routine has found a data-set directory entry for the module.

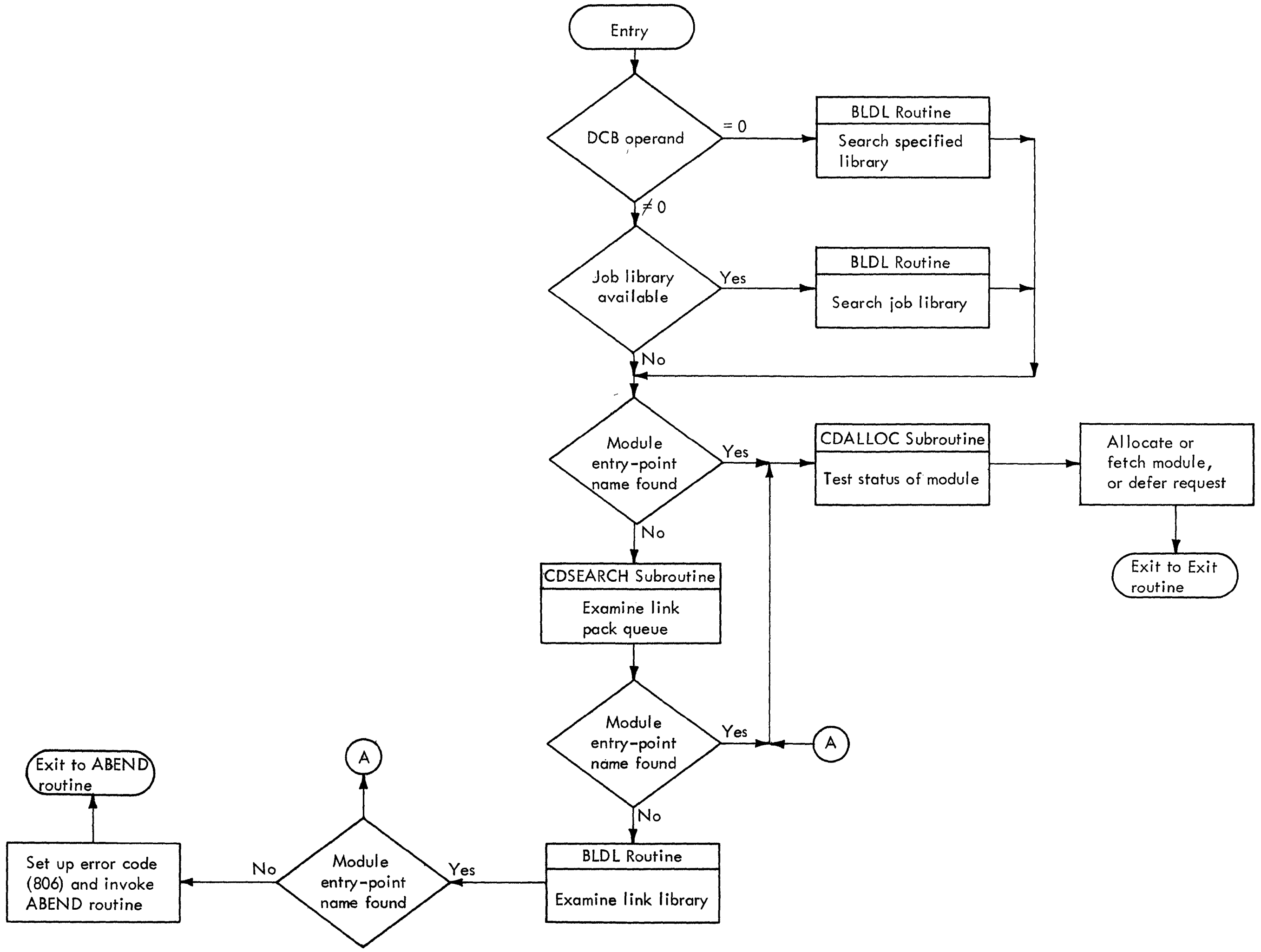


Figure 4-2. Further Search by the Common Subroutines of Contents Supervision if the Module's CDE is not in the Job Pack Queue

The attributes in the directory entry are tested to determine if the module is in main storage, recorded under another entry point name, or whether the module must be fetched.

Fetching the Module

After the BLDL routine has located the module on auxiliary storage, and if no abnormal condition has been detected, the appropriate subpool of main storage into which the module should be loaded is determined. The module's attributes, indicated in the data-set directory entry, are tested to decide the appropriate subpool. Subpool 252 is selected if the module is reenterable, and is in either the link library or the SVC library. Subpool 252 is a supervisor-protected area within the caller's region of main storage. Otherwise, subpool 251 is chosen. This subpool belongs to the job pack area for the caller's job step.

Interruptions are enabled, and the module is loaded into the chosen subpool by the Program Fetch routine. If there is no I/O error, interruptions are again disabled, and the relocated module entry point, returned by the Program Fetch routine, is stored in the CDENTPT field of the previously created CDE. The module's attributes, as indicated in the partitioned data set directory entry, are next tested. If the module is in overlay mode, the Overlay Supervisor (IEWSZOVR) is loaded from the link library, via a LOAD macro instruction. If the module contains TESTRAN symbol records, the TESTRAN routines are invoked via an SVC 61 instruction. To indicate that the module is not being loaded, the "not in storage" bit (NIC) is cleared. If the module is refreshable (eligible to be reloaded by the

Machine-Check Handler for Model 65[1]), the "refreshable" indicator REFR is set. This bit is tested by the Machine-Check Handler, if this recovery program is included in the system, when a machine check occurs. To indicate that the module is in use, the common subroutines set the "non-functional" flag (NFN). These three bits belong to the attributes fields (CDATTR and CDATTR2) of the CDE representing the module.

Performing Alias Processing

If the module request specifies an alias entry point, two types of special processing occur: one after the BLDL routine has obtained the data-set directory entry, the other after the Program Fetch routine has loaded the module.

If the module request specifies an alias entry point name, two types of alias processing can occur, depending on whether the module is already in main storage. The first type determines if the requested module is in main storage, recorded under its major entry point name. This determination is now possible, since the data-set directory entry is available for examination. The second type of processing ensures that all relocated module entry point addresses have been recorded, both main and alias, even though the current request may specify only one alias entry point name.

In the first type of processing, the common subroutines determine if the module is in main storage, recorded under its major entry point name. A new search is now necessary, since the original search was made under the assumption that the specified entry point name was a major name. The major entry point name is obtained from the data-set directory entry, and the JPACQ is searched for this name. If the name is found, the JPACQ is updated to include the alias entry point name, and the Program Fetch routine is not invoked. If, however, the major entry point name is not found, the Program Fetch routine is invoked to load the module. If the module is already being fetched, the current request is deferred.

In the second type of processing, the subroutines ensure that all relocated entry point addresses are recorded in the contents directory. The relocated major entry point address is calculated and placed in the module's major CDE. If there is at least one minor CDE queued to the major CDE (meaning that an alias or identified entry point was previously requested), the relocated entry point address for each alias is calculated and stored in its related minor CDE. (There is one minor CDE for each alias or identified entry point.)

The calculation of the relocated entry points is performed by the Relocate subroutine, which is provided with certain inputs. The inputs include the relative alias entry point address, obtained from the data-set directory entry and currently in the minor CDE, and the address of the extent list for the module, contained in the major CDE. The extent list, created by the Program Fetch routine when it loaded the module, contains the starting address of each block of main storage occupied by the module and the length of each block.

Deferring a Request

A request for a module may be deferred if the module is in main storage and is serially reusable and in use, or if the module is in the process of being fetched to main storage. In either case, the common subroutines (entered at CDQUECTL) place the SVRB for the current request on a list of waiting SVRBs, whose list origin is the CDRBP field of the major CDE for the module. Each SVRB on the list is queued to the next waiting RB via its RBPGMQ field. A new SVRB is placed on the list according to the dispatching priority of its TCB. After joining the list of waiting SVRBs, the SVRB for the current request is placed in a wait condition, its RBWCF field set greater than zero. Since processing of the current request cannot proceed, the need for a task switch is indicated to the Dispatcher. The indication is the setting of the first word of the TCB pointer (IEATCBP) to zero.

Just before the current SVRB is placed on the list of waiting SVRBs, the list is searched for an SVRB representing a previous request from the caller's task for the same module. If such an SVRB is found, the module is permanently unavailable, and an error code (A06) is set up. The requestor's task is then abnormally terminated by the ABEND routine.

Restarting Deferred Requests

Periodically a deferred request may be restarted. The purpose of such restart is to give control of the module to the requestor representing the highest priority ready task. When a module is available, an SVRB that was previously waiting for the module may compete with the current SVRB for access to the module. According to the relative task dispatching priorities, the

[1] The Machine-Check Handler for Model 65 (MCH/65) is a system generation option available with the System/360 Model 65. Refer to Section 2, "Interruption Handling."

current request is serviced, or a deferred request is restarted.

The restart procedure consists of two parts: preparation for restart, and the performance of a task switch. Preparation for restart can occur at two different times during the execution of the common subroutines. It can occur after the BLDL routine has found a data-set directory entry for the module. It can also occur after the Program Fetch routine has loaded the module into main storage. The task switch, if needed, is performed by the Dispatcher after the scheduling subroutine of Contents Supervision (CDEPILOG) has been entered.

PREPARATION FOR RESTART: During the preparation for restart, the DQLOAD subroutine makes ready the SVRBs on the waiting list, and determines if one of these SVRBs may replace the current SVRB as the controller of Contents Supervision.

The DQLOAD subroutine removes from the waiting list any SVRBs queued to the current SVRB. (The current SVRB is the one currently controlling the execution of the subroutines of Contents Supervision.) For each SVRB on the list, the subroutine clears the wait bit (RBWCF), and sets the RB old PSW to restart future execution at the beginning of the search phase of Contents Supervision (location CDCONTRL). This is the point at which restart will occur, if a task switch is performed.

Subroutine DQLOAD determines if any deferred-request SVRB can replace the current SVRB by comparing task dispatching priorities. The subroutine invokes the supervisor's Task Switching routine to compare the dispatching priority of each deferred-request TCB with that of the current TCB. The result of the series of invocations of the Task Switching routine is that the TCB pointer (IEATCBP) contains the address of the TCB whose current routine will next be dispatched (see "Testing and Indicating the Need for a Task Switch"). Depending on the relative task priorities, the current TCB may remain, or may be displaced, as the next-to-be dispatched TCB.

PERFORMANCE OF A TASK SWITCH: If the preparation for restart has altered the TCB pointer, a future branch to the Dispatcher will cause the restart of Contents Supervision at its search phase, under the control of one of its deferred-request SVRBs. The branch to the Dispatcher, if warranted, will occur during the execution of the scheduling subroutine CDEPILOG.

CDEPILOG (entry point IEAQCS03) tests if an available module should be allocated to the current requestor, or whether the Dispatcher should be entered to perform a task switch. If the two words of the TCB pointer, IEATCBP and IEATCBP + 4, are unequal, the need for a task switch has been indicated by the Task Switching routine. CDEPILOG prepares the current requestor for restart by pointing the RB old PSW in the current SVRB to the beginning of CDEPILOG. The subroutine then branches to the Dispatcher to perform the task switch. The 'Dispatcher restarts Contents Supervision at entry point CDCONTRL, under control of the selected TCB and its deferred-request SVRB. A new search for the desired module then begins, as if the restarted request had just been issued.

Scheduling Execution of the Module

When the desired module is in main storage and is immediately usable, as indicated by the test of the CDE attributes, the allocation subroutine CDALLOC recognizes the need for the immediate allocation of the module to a requestor. CDALLOC clears the "release" flag in the CDATTR2 field of the major CDE for the module. The major CDE contains the main entry point of the module and a field (CDATTR) describing its attributes, e.g., reentrant, "load only," etc. The "release" flag (CDATTR2), when cleared, indicates to the GETMAIN SVC routine that the space reserved for the module may not be reused to satisfy a later request for space.

Subroutine CDALLOC branches to two other subroutines, CDMOPUP and CDEPILOG, to perform the allocation or scheduling of the linkage to the module. The first step, performed by CDMOPUP, is to increase the "use/responsibility" count in the major CDE. The use/responsibility count is a record of the number of outstanding requests for the module issued by LINK, LOAD, XCTL, or ATTACH macro instructions. The count is decreased by the Delete SVC routine or by the Exit routine when each execution of the module has been completed.

Subroutine CDEPILOG tests whether there is a need for a task switch, to give deferred requests a new chance to search for their desired module. (See "Restarting a Deferred Request.") If a task switch is not needed, CDEPILOG gets space for and initializes a program request block (PRB) to schedule and control the execution of the requested module.

CDEPILOG obtains the information for initializing the PRB from information contained in the fields of the current SVRB. It places in the RB old PSW (RBOPSW field) of the PRB the relocated module entry point that was stored in the CDE. For an ATTACH or SYNCH request, the mode bit and protection key of the RB old PSW are duplicated from the requestor's TCB. But for an XCTL or LINK request, the first word of the old PSW is obtained from the caller's RB. For an XCTL request, the first word of the caller's old PSW was saved in the register-zero save location of the caller's SVRB, before Contents Supervision was entered.

CDEPILOG places the newly created PRB on the current task's RB queue behind the SVRB used by Contents Supervision. Later, when the Exit routine is entered, the SVRB will be removed and freed, leaving the PRB as the current RB for the requestor's task. After queuing and initializing the newly created PRB, which will control the module's execution, CDEPILOG passes control to the module or to a restarted requestor, via the Exit routine and the Dispatcher.

SPECIAL FUNCTIONS OF CONTENTS SUPERVISION

The special functions are used to assist one of the common functions or to perform a specialized service for a requestor. These functions consist of:

- Final processing for a LOAD request.
- Special processing for an XCTL request.
- Informing the supervisor of an embedded module entry point (IDENTIFY).
- Informing the supervisor that a module fetched via a LOAD macro instruction is no longer needed in main storage (DELETE).
- Supervising the loading of segments of an overlay module.
- Fetching a module to main storage.

FINAL LOAD PROCESSING

Final processing for a LOAD request is performed after the desired module is in main storage and is available. It consists of checking the load list for the caller's task to determine if a load-list element exists for the requested module.

The load list indirectly points to modules requested for a task via the LOAD macro instruction. If a module was loaded by an alias entry point name, the load-list element points to a minor CDE; otherwise the load-list element contains a pointer to a module's major CDE. It also contains a "responsibility" count (LLCOUNT) of the number of LOAD requests for the module.

If a load-list element does not exist for the module, a new element is constructed, initialized, and placed on the load list for the caller's task. It is queued from the load-list pointer (TCBLLS) in the caller's TCB.

After the load-list element is created, or if a determination is made that it already exists, the responsibility count is increased to include the current request. Control is then returned to the requestor or to the current program of the highest priority ready task, via the Exit routine and the Dispatcher.

SPECIAL XCTL PROCESSING

Special processing is performed when a caller has issued an XCTL macro instruction. If the macro instruction is issued by a user program or a user exit routine, processing is performed before control is passed to the common subroutines. If, however, an XCTL macro instruction is issued by an SVC routine, special processing is done by the transient area handler. The transient area handler schedules linkage to the desired SVC routine, and does not use the common subroutines of Contents Supervision. The simpler XCTL processing will be discussed first.

Processing if the Requestor is a User Program or a User Exit Routine

For both types of requestors (a user program or a user exit routine) the requestor's RB must be eliminated, since an XCTL request does not permit return of control to the requestor. The Exit routine is used to dequeue and free the RB of the exiting program or routine. Depending on the type of requestor, the RB is removed immediately or is removed after the requested module has been executed.

If the requestor is a user program, operating under control of a program request block (PRB), the requestor's PRB is removed immediately, before the requested module is obtained. This arrangement allows the requested program to overlay the requesting program, if necessary. To prepare for PRB removal, the positions of the caller's PRB and the SVRB for Contents Supervision are interchanged on the RB queue, so that the PRB is at the "head" of the queue (see Figure 4-3, part B). Restart of Contents Supervision is scheduled by pointing the RB old PSW in the SVRB to the search phase of the common subroutines (location CDADVANS). The Exit routine is then invoked to remove and free the caller's PRB (see Figure 4-3, part C). After eliminating the PRB, the Exit routine branches to the Dispatcher to restart Contents Supervision at CDADVANS, to begin the search for the requested module.

If the requestor is a user exit routine, operating under the control of an IRB, the requestor's IRB is removed from its RB queue only after the requested module has been obtained and executed. This delay is necessary because the IRB contains register contents belonging to the program that was interrupted by the asynchronous event. The register contents remain in the IRB until the Exit routine is entered after the requested module has been executed.

The Exit routine is scheduled (but not invoked) by placing in the RB old PSW of the requestor's IRB the address of an SVC 3 instruction. A branch is then made to the common subroutines (location CDADVANS) to search for the requested module. When the module has been obtained and executed, the Dispatcher gives control to the SVC 3 instruction. The instruction causes supervisor linkage to the Exit routine to dequeue and free the requestor's IRB (see Figure 4-3, part E1).

Processing if the Requestor is an SVC Routine

If the requestor is an SVC routine operating under the control of an SVRB, the transient area handler's XCTL routine (entry point IEAQTR03) performs special processing. The request is handled very similarly to any SVC request that reaches the SVC Second-Level Interruption Handler. The following discussion will first provide an overview of the transient area XCTL function, then a more detailed coverage.

After initial housekeeping, the Transient Area XCTL routine performs the following functions:

- Updates the transient area queue by removing the requestor's SVRB.
- Tests for and passes control to a requested routine in the link pack area of main storage.
- Determines if the requested routine is in a transient area block.
- Preparing for linkage to the routine if it is in a transient area block.
- Performs special processing to locate an available transient area block (TAB), if the routine is not already in a TAB.
- Defers the request if a TAB is not available.
- Prepares for overlaying a transient area block, if one is available.
- Loads the routine into an available TAB.

The Transient Area XCTL routine tests if the requestor is a resident or nonresident SVC routine. It tests the status bit (RBFNSVRB) in the requestor's SVRB.

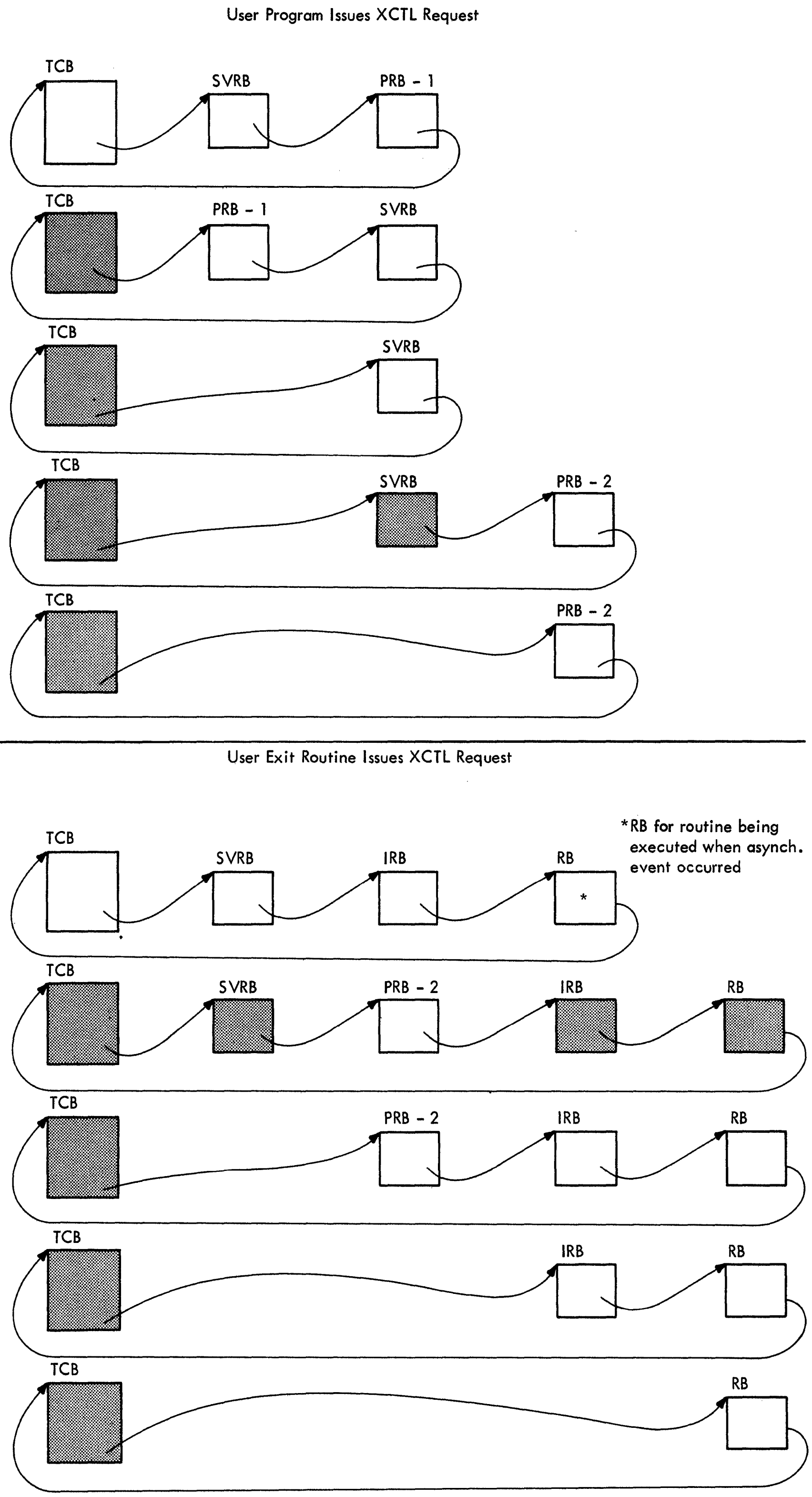
UPDATING THE TRANSIENT AREA QUEUE: If the requestor is nonresident, the TAXEXIT subroutine removes the requestor's SVRB from the user queue for the TAB that contains the requesting routine. (The user queues are described in "Fetching a Nonresident Routine from Auxiliary Storage" in Section 2, "Interruption Handling." (See Figure 4-4.)

The removing of the requestor's SVRB from the user queue is necessary because control will not be returned to the requestor. The requesting routine is no longer a "user" of a transient area block. If the requesting routine is resident in the link pack area, the TAXEXIT subroutine is bypassed, since the requestor's SVRB is not on a user queue.

TESTING FOR AND PASSING CONTROL TO A ROUTINE IN THE LINK PACK AREA: The Transient Area XCTL routine (hereafter called the TA XCTL routine) next tests if the requested routine is in the link pack area. The test consists of a search of the contents directory entries on the LPACQ. If the desired routine is in the link pack area, the requestor's SVRB is flagged as "resident" (the RBFNSVRB bit in the RBSTAB field is cleared). The requestor's SVRB, rather than the SVRB created by the SLIH after the current SVC interruption, will control the execution of the requested routine when it is finally dispatched.

The TA XCTL routine prepares for the passing of control to the routine as follows. It sets up registers, and points the RB old PSW in the requestor's SVRB to the address of the desired routine. This is the PSW that is loaded by the Dispatcher to give control to the routine. The TA XCTL routine then uses an SVC-3 instruction to gain linkage to the Exit routine. The Exit routine will remove from the RB queue and free the SVRB created for the current XCTL request, since it is no longer needed. Control is then passed to the desired routine, via the Dispatcher.

DETERMINING IF THE ROUTINE IS IN A TRANSIENT AREA BLOCK: If the requested routine is not in the link pack area, as indicated by the search of the LPACQ, the assumption is that the routine is nonresident. The TAXCTL routine then determines if the SVC routine is in one of the transient area blocks (TABs) of main storage into which nonresident routines are loaded. If the



User Program Issues XCTL Request

A. Condition of RB queue before XCTL processing.

B. Caller's PRB and SVRB are switched during XCTL processing.

C. Caller's PRB is removed by first execution of Exit routine.

D. New PRB for requested module is created by CDEPILOG subroutine.

E. SVRB is removed by next execution of Exit routine.

User Exit Routine Issues XCTL Request

*RB for routine being executed when asynch. event occurred

A1. Condition of RB queue before XCTL processing.

B1. New PRB for requested module is created by CDEPILOG subroutine.

C1. SVRB if removed by Exit routine when execution of Contents Supervision is complete.

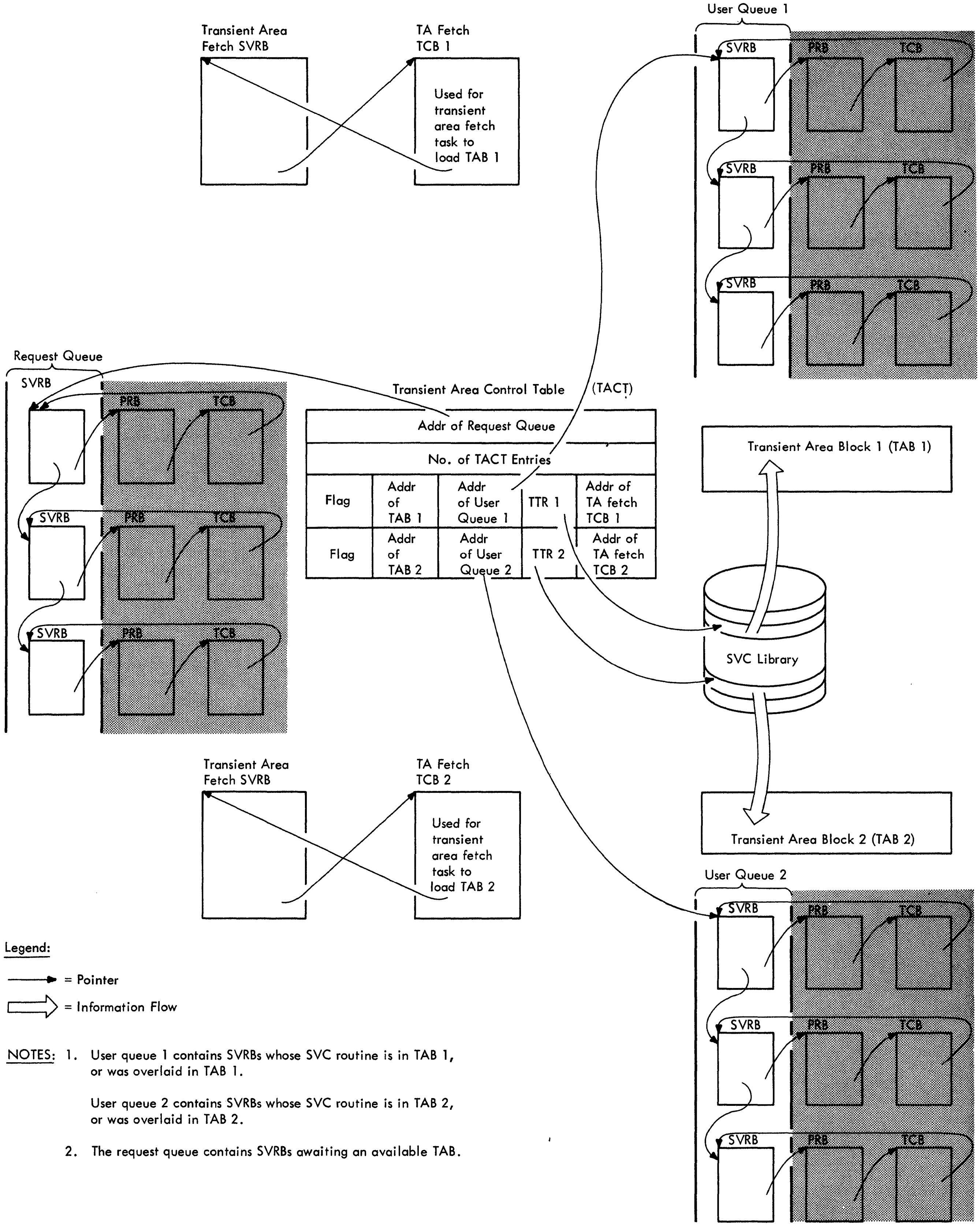
D1. PRB for requested module is removed by the Exit routine after the requested module is executed.

E1. Caller's IRB is removed by the Exit routine after the Dispatcher tries to restart the user exit routine.

Legend:

SVRB is for Contents Supervision.
PRB - 1 is for calling user program.
PRB - 2 is new PRB for requested module.
IRB is for calling user exit routine.

Figure 4-3. Manipulation of the Caller's RB Queue During Servicing of an XCTL Request


Transient Area Fetch SVRB
TA Fetch TCB 1
Used for transient area fetch task to load TAB 1
User Queue 1
SVRB
PRB
TCB
SVRB
PRB
TCB
SVRB
PRB
TCB
Request Queue
SVRB
PRB
TCB
SVRB
PRB
TCB
SVRB
PRB
TCB
Transient Area Control Table (TACT)
Transient Area Block 1 (TAB 1)
SVC Library
Transient Area Fetch SVRB
TA Fetch TCB 2
Used for transient area fetch task to load TAB 2
Transient Area Block 2 (TAB 2)
User Queue 2
SVRB
PRB
TCB
SVRB
PRB
TCB
SVRB
PRB
TCB
Legend:
= Pointer
= Information Flow

Addr of Request Queue				
No. of TACT Entries				
Flag	Addr of TAB 1	Addr of User Queue 1	TTR 1	Addr of TA fetch TCB 1
Flag	Addr of TAB 2	Addr of User Queue 2	TTR 2	Addr of TA fetch TCB 2

NOTES: 1. User queue 1 contains SVRBs whose SVC routine is in TAB 1, or was overlaid in TAB 1.

User queue 2 contains SVRBs whose SVC routine is in TAB 2, or was overlaid in TAB 2.

2. The request queue contains SVRBs awaiting an available TAB.

Figure 4-4. The Transient Area Queues

routine's name is in the permanent SVRB for a TA fetch task, the routine is currently in a TAB. (See "Fetching a Nonresident Routine From Auxiliary Storage" in Section 2, "Interruption Handling.") Accordingly, the TA XCTL routine prepares for linkage to the SVC routine.

PROCESSING IF THE ROUTINE IS IN A TRANSIENT AREA BLOCK: The TA XCTL routine then stores data in the requestor's SVRB that is needed to "refresh" the routine in case it is overlaid in the TAB before its execution is complete. This data includes the relative track and record address of the routine on auxiliary storage (obtained from the TACT entry), the routine length, the right half of the routine name, and the displacement of the TACT entry for the TAB in which the routine currently resides. The routine's name and length are obtained from the permanent SVRB belonging to the transient area fetch task associated with the TAB. The displacement of the TACT entry is calculated from the TACT address.

The TA XCTL routine then increases the "user" count for the transient area blocks. This count of the total number of user SVRBs of all TABs is examined during the execution of the TA Refresh routine when the Dispatcher is next entered. After increasing the user count, the TA XCTL routine places the requestor's SVRB on TAB's user queue, in order to keep track of the users of the TAB (see Figure 4-4).

The preparation for the passing of control to the nonresident routine is identical to that previously described for a routine resident in the link pack area.

PROCESSING IF THE ROUTINE IS NOT IN A TRANSIENT AREA BLOCK: If the name of the SVC routine is not in the permanent SVRB for a TA fetch task, the routine is not already in a TAB. The TA XCTL routine then tries to obtain a TAB into which it may place the routine. It examines the transient area control table and the user queues to find a TAB that is available. (See Figure 4-4 and Section 12, "Control Blocks and Tables.") A TAB is available in any of the following cases:

- The TAB is not being used.
- The TAB has no using SVRBs that are ready.
- The TAB may be overlaid by the requested routine. It may be overlaid if the task dispatching priority of its current user is lower than that of the requestor.

If a TAB is not available, the request is deferred. If, however, a TAB is available, the requested routine is loaded into it. When the fetch process is complete, control is passed to the routine, via the Dispatcher.

DEFERRING THE REQUEST: This discussion will first consider the case in which an available TAB cannot be found. If a TAB is not available, the TA XCTL routine defers the current request by placing the current SVRB on the transient area request queue (see Figure 4-4). The request queue is a list of SVRBs whose routines cannot be immediately scheduled for execution. The TA XCTL routine places the current SVRB into a wait condition, since execution of the current request cannot continue. To schedule the restart of this request, the TA XCTL routine points the RB old PSW in the SVRB to the "retry" entry point, called TAXRETRY. Then, to permit the Dispatcher to pass control to the current routine of another task, the TA XCTL routine indicates the need for a task switch (sets location IEATCBP equal to zero) and branches to the Dispatcher.

PREPARATION FOR THE OVERLAYING OF A TRANSIENT AREA BLOCK: If an available TAB is found, the TA XCTL routine prepares to fetch the SVC routine to the TAB. It sets into a wait condition the SVRBs on the TAB's user queue, which represent active requests for the routine currently in the TAB. Then, to delay attempted execution of the requested routine until it is fetched, the TA XCTL routine sets the requestor's SVRB in a wait condition. Then, to permit entry to the routine after it has been fetched, the TA XCTL routine points the RB old PSW in the SVRB to the address of the TAB. This address will be the entry point of the routine when the fetch is complete. To prevent accidental overlay of the TAB during the fetch process, the transient area control table (TACT) entry is flagged to indicate that the TAB is being loaded.

The extent of fetch processing (i.e., whether a BLDL macro instruction must be issued) is determined by whether the DE operand was specified in the XCTL macro instruction. If the DE operand was specified, the TA XCTL routine sets the RB old PSW in the transient area fetch SVRB (queued to a transient area fetch TCB) to bypass the BLDL procedure. (See Figure 4-4.) But if the DE operand was not specified, the RB old PSW in the transient area fetch SVRB is set to enter the BLDL procedure.

In either case, the TA XCTL routine invokes the supervisor's Task Switching routine to prepare for a switch to a transient area fetch task, under which the fetch will be performed. Then, to schedule removal of the SVRB for Contents Supervision, the RB old PSW in that SVRB is set for future entry to the Exit routine. The Exit routine will be entered when the Transient Area Fetch routine waits for I/O completion and the requestor's task again receives control. A branch is made to the Dispatcher, which passes control to the Transient Area Fetch routine to load the requested SVC routine.

INFORMING THE SUPERVISOR OF AN EMBEDDED MODULE ENTRY POINT

The Identify SVC routine informs the supervisor of a module's embedded entry-point name that was not established by the Linkage Editor. The routine informs the supervisor by creating a CDE to represent the embedded entry-point name. The Identify routine is a type-2 SVC routine (resident, SVC-issuing, disabled). It is entered from the SVC Second-Level Interruption Handler after an SVC (41) instruction has been issued.

The Identify routine searches the contents directory queues (JPACQ and LPACQ) for the specified entry-point name. The name can be the major name of a module, an alias name of a module, or a name specified in a previous IDENTIFY macro instruction. If the specified entry-point name cannot be found, the routine then determines if the specified entry-point address is valid. The entry-point address is valid if it exists in either the caller's module, or in a module which was loaded for the caller's task.

If the entry-point name cannot be found in the contents directory, and if the entry-point address is valid, the routine creates a minor CDE, which defines the identified entry point, and queues it to the module's major CDE. The Identify routine then sets up a return code indicating the result of its search, and returns control to the caller, via the Exit routine and the Dispatcher.

Upon entry (at location IGC041) the Identify routine first tests if the caller is a valid user program. The routine determines if the caller is valid by testing the type of RB under which the caller is operating. (The test is of the RBFTP subfield in the RBSTAB field.) If the RB is not a PRB, the caller is invalid. Accordingly, the Identify routine sets up a return code (hexadecimal 10), and via the Exit routine and the Dispatcher, returns control to the caller. If the test of RB type indicates that the caller is valid, the Identify routine begins its search for a contents directory entry (CDE) that may contain the desired entry-point name.

In order to perform its search, the Identify routine must first determine which contents directory queue may contain the needed CDE: the link pack area control queue (LPACQ), or the job pack area control queue (JPACQ) for the caller's job step. The initial assumption by the Identify routine is that the requested entry point lies within the caller's module. The routine therefore determines which queue it should search by testing whether the current module was loaded by the Nucleus Initialization Program (NIP). It tests the NIP bit in the CDATTR field of the caller's CDE (the CDE pointed to by the caller's RB). The NIP bit, if set by the Nucleus Initialization Program, indicates that the desired module is in the link pack area.

According to the result of the foregoing test, the Identify routine prepares to search either the link pack area queue, or the job pack area queue for the caller's job step. (Each job step has its job pack area within its own region of main storage.) The routine then searches the CDEs of the selected queue for a match between the input entry name, supplied as an operand of the IDENTIFY macro instruction, and the entry name in a CDE.

If a CDE is found whose entry-point name agrees with the requested name, the Identify routine determines if the CDE is a minor CDE by testing the MIN flag of its CDATTR field. A minor CDE contains either an alias entry-point name (established by the Linkage Editor), or an entry-point name provided by a previous execution of the Identify routine.

If the CDE is not a minor CDE, it represents a major entry-point name for the module. Since the located entry point is not an alias, the Identify routine sets up an error code (8), indicating that the specified entry-point name is the same as the major name of a module currently in storage. The routine then returns control to the caller, via the Exit routine and the Dispatcher.

If, however, the CDE is a minor CDE, the Identify routine compares the requested entry-point address with the address contained in the CDE. If these addresses are the same, a previous IDENTIFY macro instruction specifying the same entry-point address was issued. A return code (4) is used to inform the caller. But if the two entry-point addresses are unequal, a previously issued IDENTIFY macro instruction specified the same entry-point name but a different address. In this case, the routine informs the caller with a return code (hexadecimal 14), and returns control, via the Exit routine and the Dispatcher.

If in its search of either of the CDE queues, the Identify routine does not find a CDE containing the specified entry-point name, it makes an initial assumption that the entry point lies within the caller's module. The routine then examines the extent list for the caller's module to determine if the desired entry-point address is in the module. The extent list for a module contains the starting address and length in bytes for each control section of the module. The Identify routine obtains the address of the extent list for the caller's module from the module's CDE CDXLMJP field). See Figure 4-5. The extent-list pointer was placed in the CDE by the Program Fetch routine. The address of the CDE for the current module, in turn, is contained in the caller's PRB (RBCDE field).

If the entry-point address is found, the Identify routine creates and initializes a minor CDE. If, however, the entry-point address is not found, the routine continues its search for a module that contains the address. The continued search is made via the load list for the caller's task. This list represents the LOAD requests for modules made for this task. (See "Load List" in Section 12, "Control Blocks and Tables.")

For each load-list element, the routine first obtains the CDE pointer in that element (LLCDPTR) to gain access to the related CDE (see Figure 4-5). Each CDE, as stated before, contains a pointer to an associated extent list. The Identify routine then examines the extent list in the same way it had examined the extent list for the caller's module. The routine examines the extent list indirectly pointed to by all elements in the load list belonging to the caller's task. If a module containing the specified entry-point

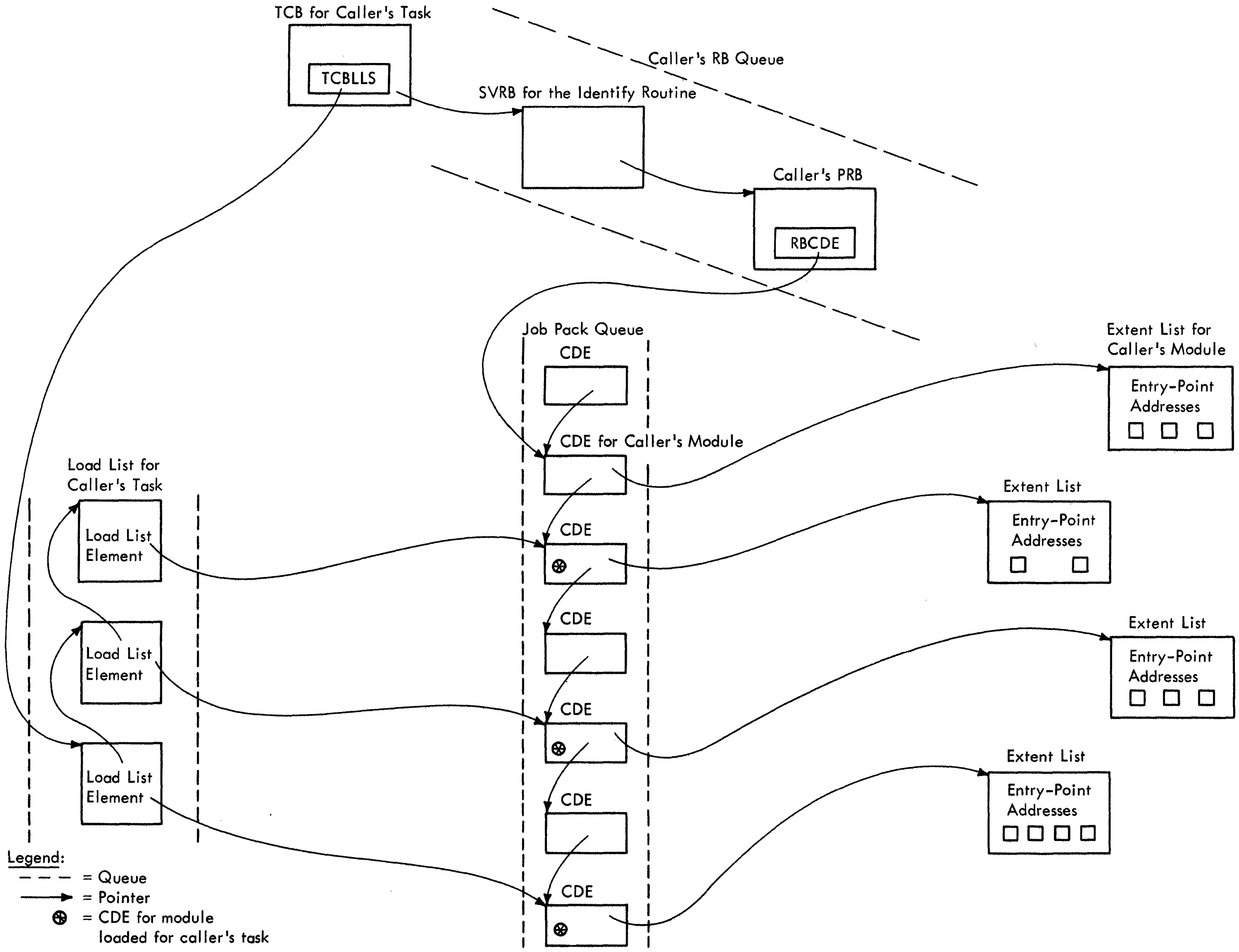


Figure 4-5. Finding an Extent List by Searching the Job Pack Queue or the Load List

address is not found, the Identify routine indicates this result by a return code (hexadecimal "C"). It then returns control to the caller, via the Exit routine and the Dispatcher.

If the desired entry-point address is found, the Identify routine next decides whether to create a minor CDE to represent the desired entry-point name. Since the routine should not create a duplicate CDE, it must determine if the needed CDE already exists on the CDE queue that it did not search. Accordingly, the Identify routine searches the remaining CDE queue.

If an entry-point name match is obtained, the routine does not create a new CDE. Instead, it sets up a return code (8), indicating that the desired entry point is the same as that of a module already in main storage. Then the Identify routine returns control to the caller, via the Exit routine and the Dispatcher.

If an entry-point name match is not obtained, the Identify routine creates a minor CDE to represent the desired entry-point name. It issues a GETMAIN macro instruction to obtain space for the new CDE (24 bytes from subpool 255, supervisor queue area). The routine then initializes the subfields of the CDE (MIN, REN, SER, and NLR) to indicate that the CDE represents a minor entry point and to indicate the module's attributes. (See Section 12, "Control Blocks and Tables" for a description of these subfields.) After initializing the new CDE, the routine queues it to the appropriate CDE queue.

Then, setting up a return code (0) to indicate successful completion of the IDENTIFY request, the routine returns control to the caller, via the Exit routine and the Dispatcher.

INFORMING THE SUPERVISOR THAT A LOADED MODULE IS NO LONGER NEEDED IN MAIN STORAGE

The Delete SVC routine is used by a system or user program to indicate to the supervisor that a module previously fetched via a LOAD macro instruction is no longer needed in main storage. The routine searches the current task's load list in order to find the contents directory entry (CDE) representing the module to be deleted. If the routine does not find the CDE, it returns control to the caller, via the Exit routine and the Dispatcher, with a return code indicating that no record of the module can be found. If the routine finds a record of the specified module, it reduces a "responsibility" count of the number of LOAD requests. In addition, if the module is not in use and there are no outstanding requests for its use, the Delete routine, via subroutine CDHKEEP, frees the space occupied by the module, its extent list, and its CDEs, thus removing all traces of the module from main storage. The Delete routine then returns control to the caller, via the Exit routine and the Dispatcher.

Upon entry (at address IGC009) the Delete routine first searches the load list for the caller's task in order to find a contents directory entry (CDE) containing the specified entry-point name. If such a CDE can be found, processing of the request can continue. Otherwise, the routine sets up a return code (4) and returns control to the caller, via the Exit routine and the Dispatcher. The Delete routine obtains the load-list origin from the TCBLLS field of the current TCB (see Section 12, "Control Blocks and Tables"). It searches the elements of the load list, examining each CDE pointed to by each load list element. If it does not find a match between the specified entry-point name, supplied as an input parameter of the macro instruction, and the name in any of the CDEs indicated by the load list, the return code is set up and control is returned to the caller, as stated previously. If the routine finds a match, processing continues as follows.

The Delete routine subtracts one from the "responsibility" count (LLCOUNT) in the load list element for the specified module. This count is a record of the number of outstanding LOAD requests for the module. (See Section 12, "Control Blocks and Tables.") Each execution of the Delete routine will similarly decrease the responsibility count until the count reaches zero. The routine next checks whether this count has reached zero. A responsibility count of zero 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 responsibility count in the load list element is zero, the routine removes the element from the load list, and issues a FREEMAIN macro instruction to free its space. This action is appropriate, since a load list element merely indicates an outstanding LOAD request for a module, not whether the module has been fetched via another type of macro instruction, or whether the module is still being used.

The Delete routine next subtracts one from the "use/responsibility" count in the major CDE that it has found. This count, unlike the responsibility count in a load list element, records the total number of requests for a module, via ATTACH, LINK, LOAD, or XCTL macro instructions. The count is increased for each such request and decreased for each DELETE or SVC 3 instruction.

The routine tests the use/responsibility count in the major CDE to determine if the module's storage areas may be freed. These areas include the space occupied by the module, its CDEs, and its extent list. If the count is not zero, at least one requesting program within the current task has not completed its use of the module. That is, the module has not yet issued a RETURN macro instruction, nor has a DELETE macro instruction been issued for it. Since the module's storage areas cannot be freed, the routine returns control to the caller, via the Exit routine and the Dispatcher.

If, however, the use/responsibility count is zero, the Delete routine acknowledges the lack of outstanding requests for the module by branching to subroutine CDHKEEP to free the storage space occupied by the module, its extent list, and its CDEs (both major and minor CDEs, if both types exist). The address of the extent list for the module is obtained from its major CDE. After freeing the module's storage space, the Delete routine returns control to the caller, via the Exit routine and the Dispatcher, with a return code of zero.

SUPERVISING THE LOADING OF SEGMENTS OF AN OVERLAY MODULE

The Overlay Supervisor directs the loading of segments of an overlay module. Before the execution of an overlay module, the Linkage Editor builds two sets of tables, the segment table and the entry tables, which it places in the overlay module. Later, during execution of the module, the Overlay Supervisor uses and alters information in the tables to perform its functions.

Preparatory Linkage Editor Functions

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

There is only one segment table (SEGTAB) in an overlay module, as shown in Figure 4-6. The segment table is used to keep track of the relationship of the segments in the module, and to determine which segments are in main storage or are being loaded.

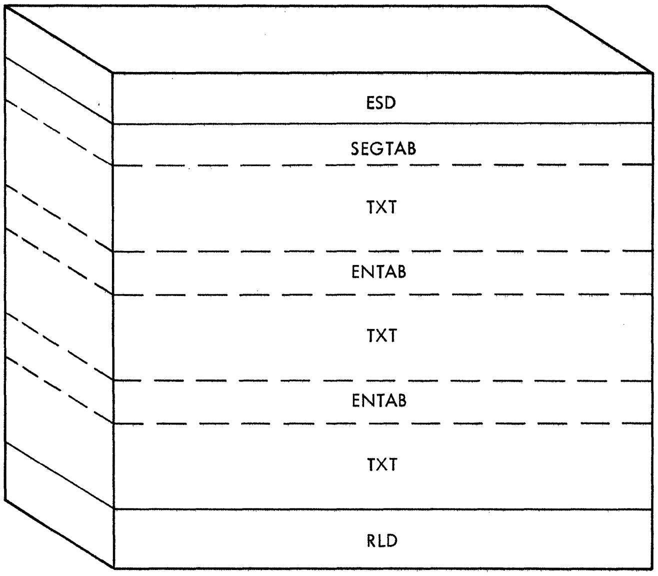


Figure 4-6. Organization of an Overlay Module

The Linkage Editor builds an entry table for each segment that contains V-type address constants. (See Figure 4-6.) A table entry is made for each constant that refers to a symbol whose segment must be fetched via a CALL or branch instruction. The Linkage Editor saves in each entry the value it assigns to the constant. It places in the value field of the constant the address of the ENTAB entry.

During module execution, when the branch instruction that uses the address constant is executed, the branch will give control to an instruction in the associated ENTAB entry. Instructions in the ENTAB will provide supervisor linkage to the Overlay Supervisor if the desired segment is not in main storage. If the segment has been fetched by the Overlay Supervisor, instructions in the ENTAB will provide a branch to the segment.

If Main Storage Hierarchy Support is included in the system, the loading of overlay structure programs can be directed into hierarchy 0 or hierarchy 1 by the parameter HIARCHY=, but segments of a program written in overlay mode cannot be loaded into different hierarchies. When hierarchy is not specified, the overlay structure exists in hierarchy 0.

Functions of the Overlay Supervisor

The Overlay Supervisor receives control either when an overlay segment issues a SEGLD or SEGWT request for another segment, or when a segment issues a CALL or branch instruction to an external address in another segment not in main storage. In both cases, the Overlay Supervisor 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. It then causes the loading of the requested segment, if not already in main storage, and any needed segments in its path. The actual loading is performed by the Program Fetch routine.

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.

Lastly, depending on the type of invoking macro instruction, control is given to the:

- Caller before loading is complete (SEGLD).
- Caller after loading is complete (SEGWT).
- Branch address in the requested segment after it is loaded (CALL or branch instruction).

<u>Linkage to the Overlay Supervisor</u>

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.

<u>DIRECT SUPERVISOR LINKAGE</u>: When the expansion of a SEGLD or SEGWT macro instruction is issued, an SVC (37) interruption occurs and control is given, in turn, to the SVC First-Level Interruption Handler, the SVC Second-Level Interruption Handler, and to resident module IGC037 of the Overlay Supervisor. If direct branch entry to the requested segment, via the caller's ENTAB, has been prepared through a previous branch or CALL, control is returned to the caller (see Figure 4-7). In this case, further processing of the current request is not needed. But if a direct branch entry has not been prepared, module IGC037, after performing initialization, issues a LINK macro instruction to obtain supervisor linkage to the nonresident module IEWSZOVR. This module processes the request, as described in "Types of Processing."

<u>SUPERVISOR LINKAGE VIA THE CALLER'S ENTRY TABLE</u>: 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. The SVC 45 instruction causes supervisor linkage, via the SVC First-Level and Second-Level Interruption Handlers, to resident module IGC037 of the Overlay Supervisor (see Figure 4-8, A, B, and C). After performing initialization, module IGC037 issues a LINK macro instruction to obtain supervisor linkage to the nonresident module IEWSZOVR. This module processes the branch request, as described in "Types of Processing."

<u>Types of Processing</u>

During execution of an overlay module, the loading of a requested segment and the passing of control depend on the type of instruction that the caller has issued and whether:

- The requested segment is in main storage.
- A SEGLD request is being processed.
- A CALL or branch instruction was previously issued specifying the same external address.

The type of processing for each set of conditions is summarized in Table 4-1.

<u>Determining the Segments That Must Be Loaded</u>

The nonresident module (IEWSZOVR) of the Overlay Supervisor determines which segments should be loaded. It does this by scanning the segment table of the overlay module, which was loaded with the root segment. It examines status indicators in the segment table, previously set by the Linkage Editor or the Overlay Supervisor, to determine which, if any, segments in the path of the requested segment must be loaded. For each segment that must be loaded, IEWSZOVR sets indicators to control a subsequent fetch process.

The segment table, a part of the root segment, was built by the Linkage Editor. It contains one entry for each segment of the overlay module. The entries 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 tree structure. Figure 4-9 illustrates a typical segment table for a "single-region" overlay structure. (An overlay program can be designed in single or multiple regions of main storage -- not to be confused with job-step regions. (See the <u>Linkage Editor</u> SRL publication for further information.)

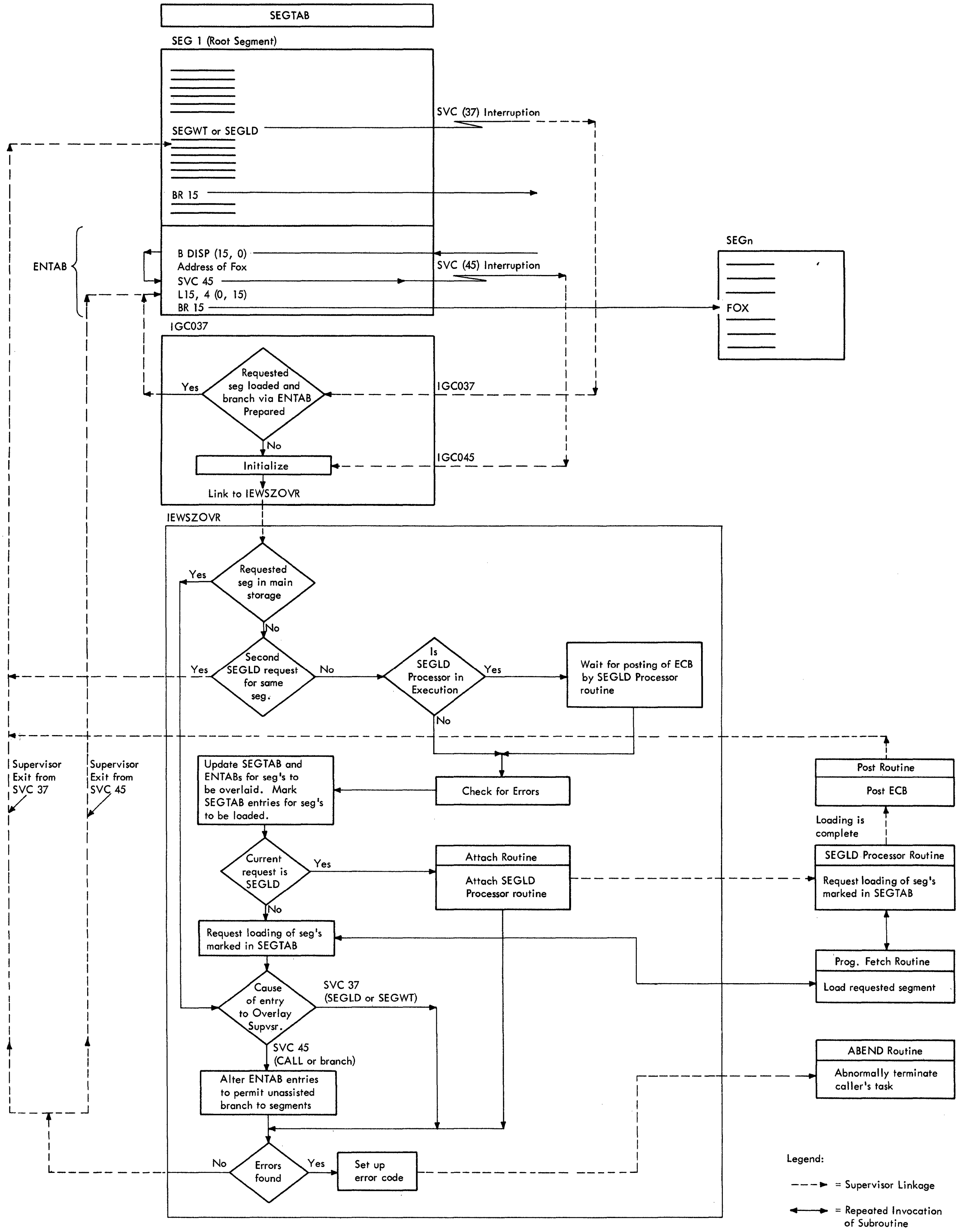

SEGTAB
SEG 1 (Root Segment)
SEGWT or SEGLD
SVC (37) Interruption
BR 15
ENTAB
B DISP (15, 0)
Address of Fox
SVC 45
SVC (45) Interruption
L15, 4 (0, 15)
BR 15
SEGn
FOX
IGC037
Requested seg loaded and branch via ENTAB Prepared
Yes
No
IGC037
Initialize
IGC045
Link to IEWSZOVR
IEWSZOVR
Requested seg in main storage
Yes
No
Second SEGLD request for same seg.
Yes
No
Is SEGLD Processor in Execution
Yes
No
Wait for posting of ECB by SEGLD Processor routine
Supervisor Exit from SVC 37
Supervisor Exit from SVC 45
Update SEGTAB and ENTABs for seg's to be overlaid. Mark SEGTAB entries for seg's to be loaded.
Check for Errors
Post Routine
Post ECB
Loading is complete
Current request is SEGLD
Yes
No
Attach Routine
Attach SEGLD Processor routine
SEGLD Processor Routine
Request loading of seg's marked in SEGTAB
Request loading of seg's marked in SEGTAB
Prog. Fetch Routine
Load requested segment
Cause of entry to Overlay Supvsr.
SVC 37 (SEGLD or SEGWT)
SVC 45 (CALL or branch)
Alter ENTAB entries to permit unassisted branch to segments
ABEND Routine
Abnormally terminate caller's task
Errors found
No
Yes
Set up error code
Legend:
= Supervisor Linkage
= Repeated Invocation of Subroutine

Figure 4-7. Functional Flow of Overlay Supervision

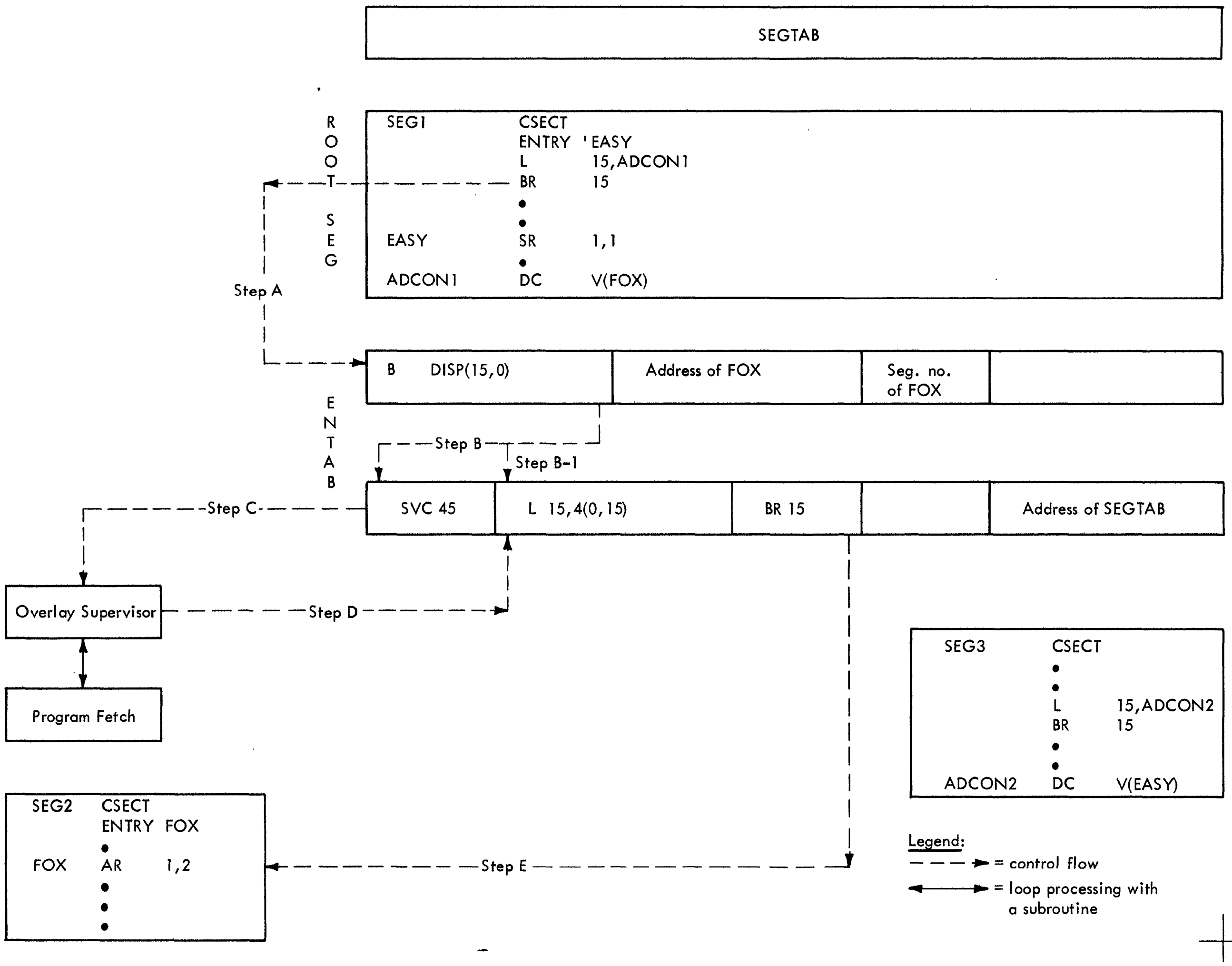


Figure 4-8. Use of the Caller's ENTAB to Branch to a Segment

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 Table 4-2.

Controlling the Loading of Needed Segments

The loading of needed segments is performed in two different ways, depending on whether the current request is made via a SEGWT or a SEGLD macro instruction.

For a SEGWT request, IEWSZOVR, as part of the caller's task, directly invokes the Program Fetch routine to load each segment whose SEGTAB entry is marked 01 ("loading scheduled"). The caller is given control only after all such segments have been loaded.

For a SEGLD request, IEWSZOVR attaches as a subtask the SEGLD Processor routine (OVLALD02) which, under control of the subtask TCB, invokes the Program Fetch routine to load each segment. As with a SEGWT request, each segment is loaded whose SEGTAB entry is marked 01 ("loading scheduled"). However, at the first I/O wait interval, control is returned to the issuer of the SEGLD macro instruction, although the needed segments have not yet been loaded. Later, if the caller tries to branch to the requested segment before loading is complete, its task is forced to wait. While the caller's task waits, the SEGLD Processor routine completes the loading of the needed segments, and then posts an event control block to ready the waiting task.

Preparation for an Unassisted Branch to the Loaded Segment

When the requested segment and any needed segments in its path have been loaded, it is desirable to permit the caller to branch to the requested segment via its ENTAB, without help from the Overlay Supervisor. Such an unassisted branch

Table 4-1. Types of Processing During Overlay Supervision

Instruction	Conditions	Major Processing
SEGLD (SVC 37)	1. Requested segment and/or segments in its path are not in main storage, and are not in process of being loaded.	1. Loading of needed segments is started. The caller's entry table is not altered to prepare for a branch to the requested segment. Control is returned to the caller while the segment or segments are being loaded. The requested segment is not entered.
	2. Requested segment is in main storage or is being loaded.	2. Control is returned to the caller.
SEGWT (SVC 37)	3. Same conditions as in (1).	3. 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 only after the requested segment and any needed segments in its path have been loaded. The requested segment is not entered.
	4. Requested segment is being loaded for a SEGLD request.	4. Processing of the SEGWT request waits until loading is complete. No new loading occurs. Remaining processing is as in (3).
	5. Requested segment is in main storage.	5. Control is returned to the caller.
CALL or branch (SVC 45)	6. Segment was requested via SEGLD or SEGWT and is in main storage.	6. The caller's entry table is altered to prepare for a future branch to the same external address without entry to the Overlay Supervisor. Control is then given to the requested segment at the specified address.
	7. Segment was requested via a SEGLD and loading is not complete	7. Processing of the CALL or branch request waits until loading is complete. No new loading occurs. Remaining processing is as in (6).
	8. Requested segment is not in main storage, nor is it being loaded.	8. Needed segments are loaded. When loading is complete, the remaining processing is the same as in (6).
	9. Caller previously issued a CALL or branch instruction specifying same external address.	9. Overlay Supervisor is not entered. The caller's entry table, previously altered as in (6), provides a direct branch to the requested segment.

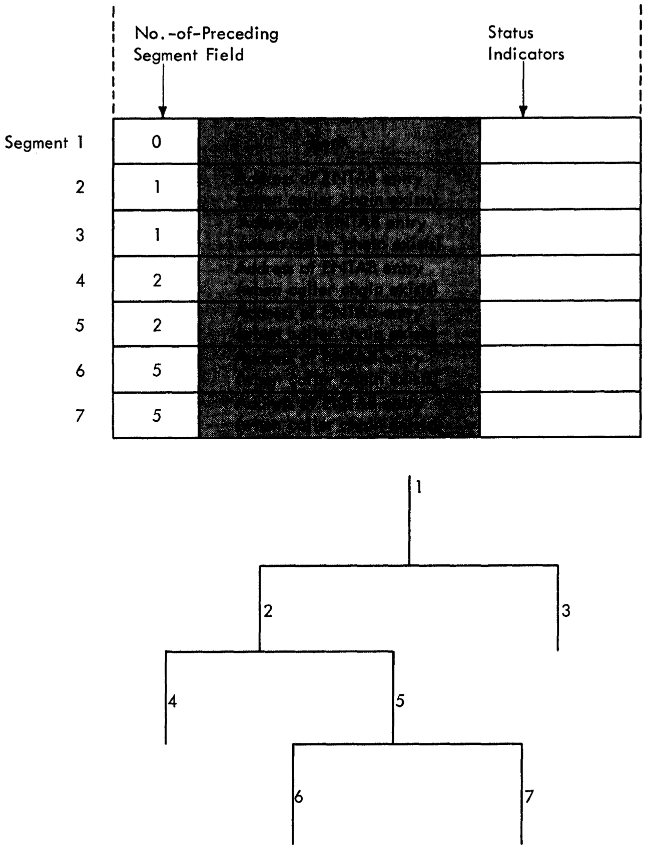

No.-of-Preceding Segment Field
Status Indicators
Segment 1
2
3
4
5
6
7
0
1
1
2
2
5
5
Zero
Address of ENTAB entry (when caller chain exists)
Address of ENTAB entry (when caller chain exists)
Address of ENTAB entry (when caller chain exists)
Address of ENTAB entry (when caller chain exists)
Address of ENTAB entry (when caller chain exists)
Address of ENTAB entry (when caller chain exists)
1
2
3
4
5
6
7

Figure 4-9. Organization of SEGTAB Entries for a Single-Region Overlay Structure

would bypass the SVC 45 instruction in the caller's ENTAB (see steps A, B-1, and E of Figure 4-7).

The alteration of the caller's ENTAB occurs after the caller has issued its first CALL or branch instruction to obtain linkage to the requested segment. The CALL or branch instruction may itself cause the loading of the segment (see Table 4-1 and Figure 4-7).

When module IEWSZOVR is entered after an SVC (45) interruption, 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 4-8, 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 4-8, Step B1). 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 no longer in main storage. The resetting of ENTAB entries in a caller chain accompanies the processing shown for condition 4 of Table 4-2. shown for condition 4 of Table 4-2.

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 Table 4-1 and Figure 4-7.

FETCHING ROUTINES AND MODULES TO MAIN STORAGE

The Program Fetch routine loads SVC routines, I/O error-handling routines, and other modules. As part of the loading process, the Program Fetch routine obtains needed storage space, performs I/O operations, and relocates address constants when necessary.

The Program Fetch routine is invoked, via a branch instruction, by any of several supervisor routines, depending on the type of module or routine that is requested, as follows:

Type of Requested Module or Routine	Routine That Invokes Program Fetch
Nonresident SVC routine	Transient Area Fetch routine
I/O error handling routine	Stage 3 Exit Effector
Nonoverlay module that is not available in main storage, or the root segment of an overlay module that is not available in main storage	Common subroutines of Contents Supervision
A segment of an overlay module (except the root segment)	Overlay Supervisor

Table 4-2. Processing of Segment Table Entries

Conditions	Resultant Processing by IEWSZOVR
1. Requested segment is in main storage. (indicator 10)	If entry is for a SEGWT or SEGLD request, control is returned to caller. If entry is for CALL or branch, ENTAB entries are altered to provide future branch 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.

Fetching SVC and I/O Error-Handling Routines

Either the SVC Second-Level Interruption Handler or the Stage 3 Exit Effector determines if a usable copy of the desired routine is in a transient area block (TAB) of main storage. If a usable copy is in a TAB, control is given to the routine. Otherwise, the Program Fetch routine is invoked to load the requested routine into a TAB. A nonresident SVC routine is placed in an SVC transient area block; an I/O error-handling routine is placed in the I/O Supervisor transient area block (see Figure 4-10).

If the Program Fetch routine must be invoked, the caller places in a fetch work area the relative disk address and the size of the routine to be loaded. The caller obtains this information from the data-set directory entry belonging to the SYS1. SVCLIB data set.

Note: A separate fetch work area precedes each transient area block. Each work area contains 68 bytes of space and is constructed during system generation. (See "Program Fetch Work Area in Section 12.) The work area contains an input/output block (IOB), an event control block (ECB), and a channel program. (See Figure 4-11.)

The Program Fetch routine determines the absolute disk address of the requested routine and causes the loading of the routine. It converts the relative disk address of the routine to an absolute address by means of a resident "convert" routine. It then issues an EXCP macro instruction and a WAIT macro instruction. The EXCP macro instruction causes the I/O Supervisor to be invoked to fetch the desired routine from the SYS1.SVCLIB data set to the appropriate TAB. The routine's entry point address is the same as the address of the TAB. No relocation is needed, since a transient SVC routine contains no relocatable address constants.

When the requested routine has been loaded, the Program Fetch routine checks for I/O errors, places a return code in register 15 to indicate that the fetch has been successful or that I/O error or invalid information has been detected, and returns control to the calling routine.

Fetching Nonresident Modules

The Program Fetch routine is invoked either by the common subroutines of Con-

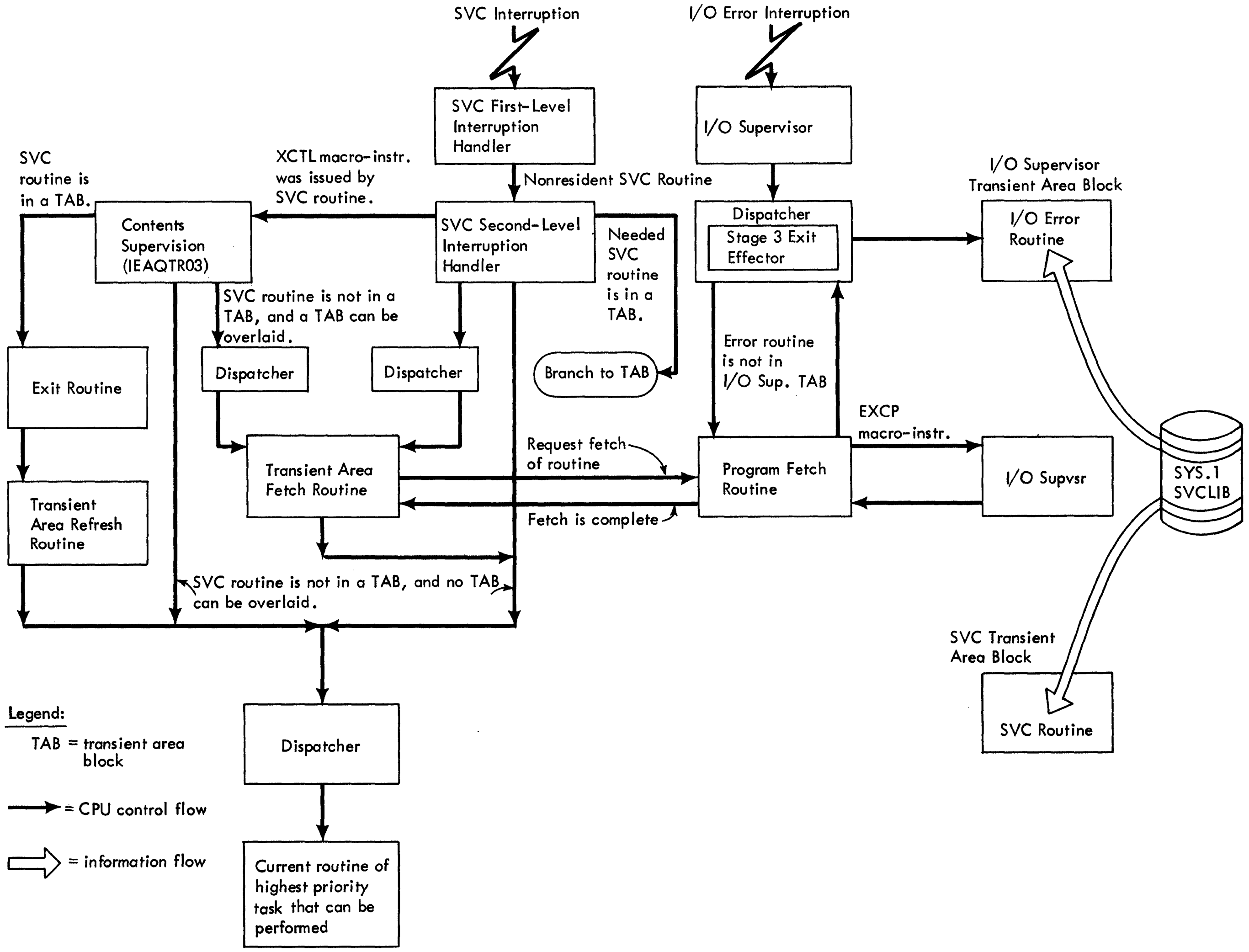

SVC Interruption
I/O Error Interruption
SVC First-Level Interruption Handler
I/O Supervisor
SVC routine is in a TAB.
XCTL macro-instr. was issued by SVC routine.
Nonresident SVC Routine
I/O Supervisor Transient Area Block
Contents Supervision (IEAQTR03)
SVC Second-Level Interruption Handler
Needed SVC routine is in a TAB.
Dispatcher
Stage 3 Exit Effector
I/O Error Routine
SVC routine is not in a TAB, and a TAB can be overlaid.
Error routine is not in I/O Sup. TAB
Exit Routine
Dispatcher
Dispatcher
Branch to TAB
EXCP macro-instr.
Transient Area Fetch Routine
Request fetch of routine
Program Fetch Routine
I/O Supvsr
SYS.1 SVCLIB
Transient Area Refresh Routine
Fetch is complete
SVC routine is not in a TAB, and no TAB can be overlaid.
SVC Transient Area Block
Legend:
TAB = transient area block
Dispatcher
SVC Routine
= CPU control flow
= information flow
Current routine of highest priority task that can be performed

Figure 4-10. Relationships of Program Fetch Routine to Other Routines for the Fetch of an SVC Routine or an I/O Error Routine

tents Supervision or by the Overlay Supervisor.

It is invoked by the common subroutines of Contents Supervision after a LINK, ATTACH, LOAD, or XCTL macro instruction has been issued, if a usable copy of the needed module is not in main storage. It is invoked by the Overlay Supervisor after a SEGWT, SEGLD, or CALL macro instruction, or a branch instruction has been issued, if the needed segment of an overlay module is not in main storage. The relationship of the Program Fetch routine to other routines for the fetch of a module or overlay segment is depicted in Figure 4-12.

The major functions of the Program Fetch routine for the loading of a nonresident module or an overlay segment are:

Initialization
: Initializes a fetch work area, builds an extent list, and (if the module is in overlay mode) fetches the module's note list. If the module is to be scatter-loaded, the routine fetches the scatter/translation table.

Loading
: Transfers text records and relocation list dictionary (RLD) records from auxiliary storage to main storage. The text records constitute the program that is loaded. The RLD records are used for relocation.

Relocation
: Changes the values of address constants in the loaded program from relative addresses to absolute addresses.

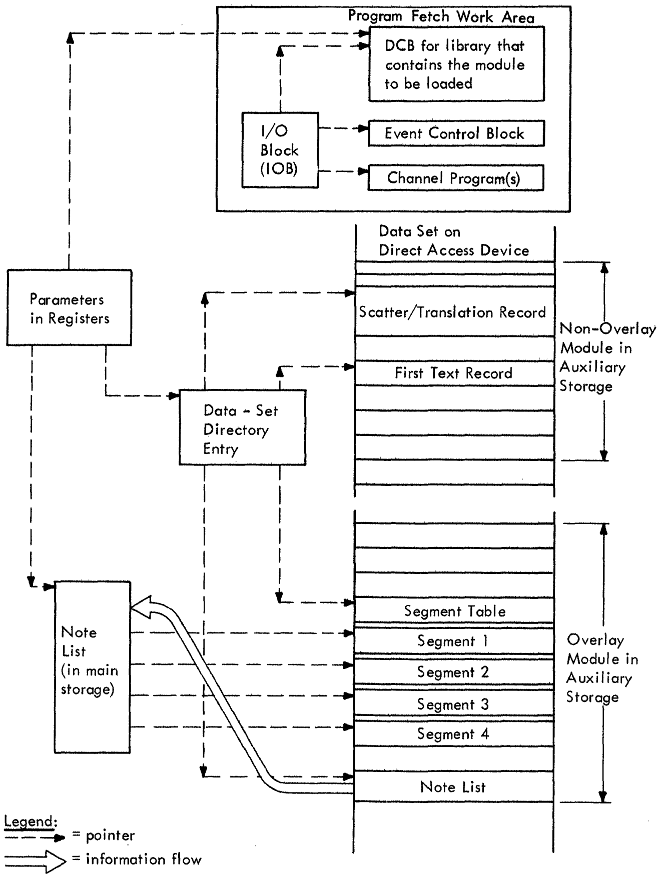

Program Fetch Work Area
DCB for library that contains the module to be loaded
I/O Block (IOB)
Event Control Block
Channel Program(s)
Data Set on Direct Access Device
Parameters in Registers
Scatter/Translation Record
First Text Record
Non-Overlay Module in Auxiliary Storage
Data - Set Directory Entry
Note List (in main storage)
Segment Table
Segment 1
Segment 2
Segment 3
Segment 4
Note List
Overlay Module in Auxiliary Storage
Legend:
= pointer
= information flow

Figure 4-11. Control Blocks and Tables Used by the Program Fetch Routine

Termination

Checks the completion of I/O operations, calculates the relocated module entry-point address, initializes the segment table (if the module is in overlay mode), sets up a return code, and returns control to the caller.

INITIALIZATION: The Program Fetch routine can make available three areas or tables for later use. They are the program fetch work area, the extent list, and the note list. The fetch work area is used by the Program Fetch routine to load module records. The extent list is used by the common subroutines of Contents Supervision to prepare linkage to the module; it is used by the CDEXIT routine to free the module's storage areas during end-of-task and abnormal termination procedures. The note list is part of an overlay module; it contains the relative disk address of each segment and, after main storage has been obtained, contains the module's relocation factor.

Initializing the Fetch Work Area: The Program Fetch routine initializes a work area whose address is furnished by the caller. It places in the work area information that it will use to load the requested module. This information consists of:

- An input/output block (IOB). The IOB provides information that is needed by the I/O Supervisor.

- Two event control blocks (ECBs). One ECB is posted by the I/O Supervisor when a channel-end condition occurs. The other is posted by a PCI Appendage routine when a program-controlled interruption occurs in a channel program. The posting of either ECB permits the restarting of the Program Fetch routine after an I/O wait interval.

- Three channel programs. The channel programs are similar. They are used to overlap the reading of one or more module records with the relocation of address constants pointed to by a previously loaded RLD record.

- Three RLD buffers. Each buffer is 260 bytes in length, and is capable of holding an RLD record, a control record, or a composite control and RLD record. (For record formats, see Section 12, "Control Blocks and Tables.")

- A buffer table. This table contains a 12-byte entry for each RLD buffer. Each entry contains:

  - A pointer to the next entry.

  - The address of an RLD buffer.

  - The address of a channel program.

Building an Extent List: The extent list, when completed, contains the main storage address and length of each loadable section of a module (see Figure 4-13). The size of the extent list and the procedures for constructing it depend on whether the module is to be block-loaded or scatter-loaded. During the construction of the extent list, main storage is obtained in preparation for loading the module.

If the module is to be block-loaded, the Program Fetch routine obtains space for an extent list, and if necessary, a note list. The routine places in the "length" field of the extent list the total size of the module, as shown in the data-set directory entry. Next, the Program Fetch routine issues a GETMAIN macro instruction to obtain space for the module. The assigned

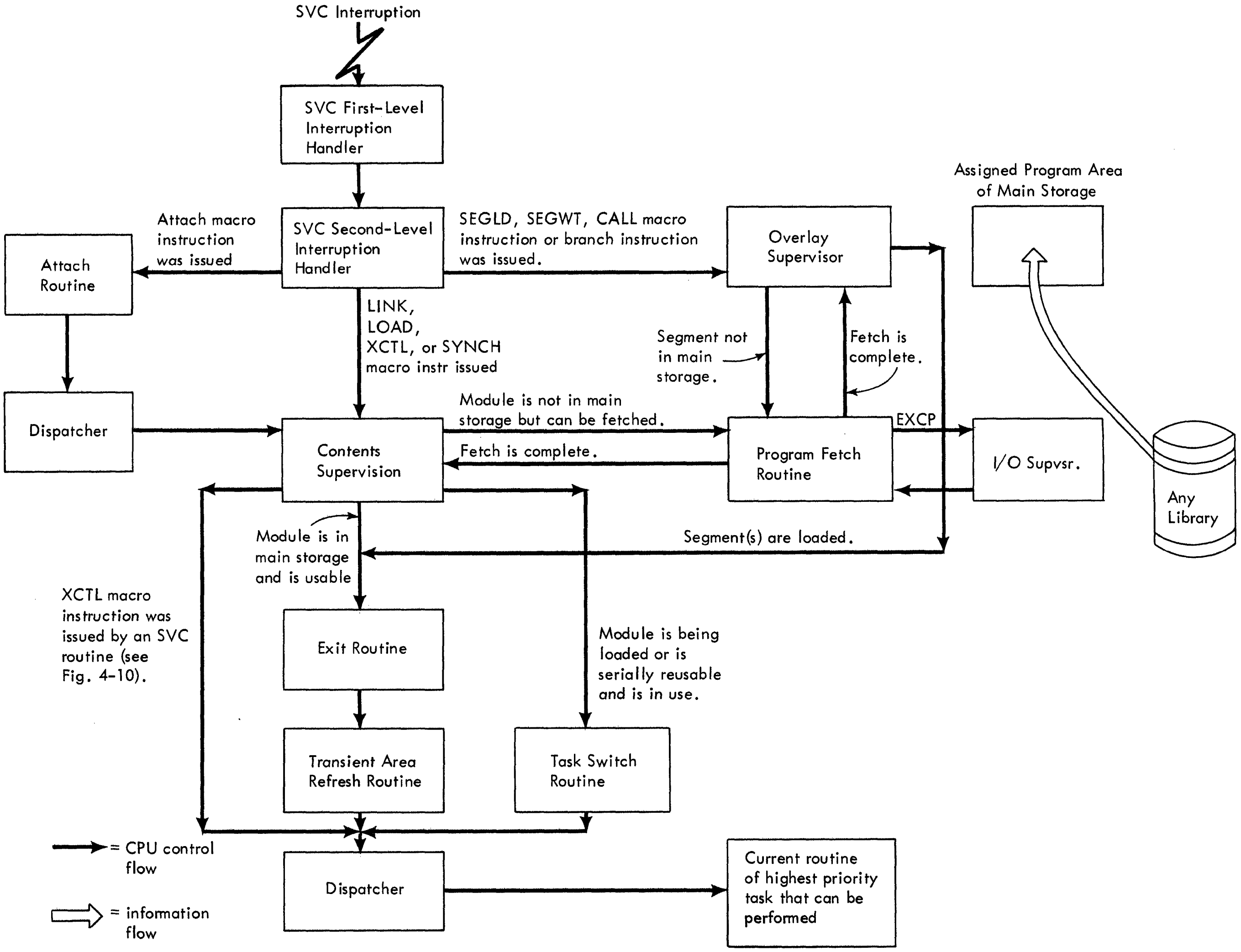

SVC Interruption
SVC First-Level Interruption Handler
Attach macro instruction was issued
SVC Second-Level Interruption Handler
SEGLD, SEGWT, CALL macro instruction or branch instruction was issued.
Overlay Supervisor
Assigned Program Area of Main Storage
Attach Routine
LINK, LOAD, XCTL, or SYNCH macro instr issued
Segment not in main storage.
Fetch is complete.
Dispatcher
Module is not in main storage but can be fetched.
EXCP
Contents Supervision
Fetch is complete.
Program Fetch Routine
I/O Supvsr.
Any Library
Module is in main storage and is usable
Segment(s) are loaded.
XCTL macro instruction was issued by an SVC routine (see Fig. 4-10).
Exit Routine
Module is being loaded or is serially reusable and is in use.
Transient Area Refresh Routine
Task Switch Routine
= CPU control flow
= information flow
Dispatcher
Current routine of highest priority task that can be performed

Figure 4-12. Relationship of Program Fetch Routine to Other Routines for the Fetch of a Module or Overlay Segment Module or Overlay Segment

No. of Bytes in Extent List
No. of Relocation Factors
Length of First Storage Block
.
.
Length of Last Storage Block
Address of First Storage Block
.
.
Address of Last Storage Block

Figure 4-13. Extent List

main storage address returned by the GETMAIN routine is then placed in the address field of the extent list.

In systems generated with storage hierarchies, a GETMAIN request is issued for the creation of the block extent list, followed by an unconditional GETMAIN request using the specified hierarchy. If no hierarchy is specified, the request is satisfied from hierarchy 0. If the unconditional request made by Program Fetch cannot be fulfilled, the GETMAIN routine determines whether to invoke ABEND or Rollout/Rollin functions.

If the module is to be scatter-loaded, the Program Fetch routine builds an extent list and obtains space for the module, as follows:

1. Determines the needed space for the extent list. It does this by calculating the size of the scatter list/translation table from information contained in the data set directory entry. The scatter list and translation table are placed by the Linkage Editor in a module that can be scatter-loaded (see Linkage Editor PLM).

2. Issues a GETMAIN macro instruction for space for the combined extent list and scatter list/translation table.

3. Obtains the relative disk address of the first scatter list/translation table record from the data-set directory entry and converts it to an absolute disk address. The routine obtains the size of the scatter list/translation table from the data set directory entry. It then issues an EXCP macro instruction to read the record(s). The scatter list/translation record(s) are read from auxiliary storage to the lower part of the space allocated to the extent list.

4. Initializes the extent list with the length of the extent list itself, the number of scatterable control sections, and the length of each control section of the module. The routine determines the length of the extent list from the number of entries in the scatter list. It calculates the length of each control section from the relative addresses of the control sections, recorded in the scatter list/translation table.

5. Obtains space for each control section by the issuance of a GETMAIN macro instruction that specifies the list of control-section lengths just calculated (step 4). The GETMAIN routine returns to the Program Fetch routine the allocated address for each control section.

6. Calculates the relocated address for each control section from its allocated address (obtained from the GETMAIN routine) and its relative address (obtained from the scatter list/translation table).

When a request is made for a specific hierarchy, a conditional GETMAIN request is issued for the specified hierarchy. If sufficient contiguous storage is not available, Program Fetch builds a list of lengths in preparation for the scatter attempt for each CSECT. The GETMAIN request is then issued for the specified hierarchy.

If the request is made without specifying a hierarchy in a system generated with storage hierarchies, initiation for hierarchy loading is performed. The size of the extent list for scatter and the size of the scatter list/translation table record are determined before the GETMAIN request is issued. The scatter list/translation table record is processed to determine the linkage editor hierarchy designator. If all designators reference the same hierarchy, an attempt is made to block load the module. If this is unsuccessful, Program Fetch builds a list of lengths for each CSECT and an unconditional GETMAIN request is issued for the proper hierarchy.

When the scatter list/translation table record indicates that the module had been link edited to utilize multiple hierarchies, Program Fetch builds a list of lengths for each CSECT and appends the appropriate hierarchy designator to each CSECT. An unconditional GETMAIN request is then issued and space is obtained from both hierarchies 0 and 1.

Obtaining the Note List: If the module to be loaded is in overlay mode, the Program Fetch routine must load the note list before it fetches the root segment of the module. The note list, placed in an overlay module, by the Linkage Editor, contains the relative disk address (TTR) of each segment of the module. When the root segment has been loaded, the Program Fetch routine will store in the note list the address of the segment table (SEGTAB), and the relocation factor for the module. The note list will remain in main storage throughout the module's execution. (See Figure 4-14.)

```
┌──────────┬─────────────────────────────┐
│          │Relocation factor for module │
├──────────┴──────────────┬──────────────┤
│                         │ Concatenation│
│                         │ Number       │
├─────────────────────────┴──────────────┤
│TTR - relative (to beginning of data    │
│set) disk address of segment 1          │
├────────────────────────────────────────┤
│TRR - relative (to beginning of data    │
│set) disk address of segment 2          │
└────────────────────────────────────────┘
.                                        .
.                                        .
┌────────────────────────────────────────┐
│TTR - relative (to beginning of data    │
│set) disk address of segment N          │
└────────────────────────────────────────┘
```

Note: Concatentation Number - This is a value specifying this data set's sequential position within a group of concatenated data sets.

Figure 4-14. Note List as It Exists in Main Storage

To load the note list, the Program Fetch routine follows a procedure similar to that just described in steps 1, 2, and 3 in "Building an Extent List."

LOADING OF MODULE RECORDS: The Program Fetch routine loads module records of several types: control records, text records, RLD records, and composite control/RLD records. A typical logical sequence is shown in Figure 4-15. Their formats are described in Section 12, "Control Blocks and Tables." (For a discussion of each type, see the Linkage Editor PLM.)

The loading of module records consists broadly of four functions:

- Preparing for the execution of a channel program. An absolute disk seek address is computed and made available to the I/O Supervisor.

- Starting a channel program. The I/O Supervisor is invoked to start the I/O operation at the specified disk address.

- Reading of module records. Text and RLD or control records are read to main storage blocks or to buffers.

- Switching of channel programs. Three channel programs are switched to follow the sequence of module records on the direct-access device.

Preparing for Execution of a Channel Program: The Program Fetch routine, in order to obtain the execution of a channel program, must furnish to the I/O Supervisor an absolute disk address at which the first I/O operation will begin. The routine accomplishes this objective by:

- Obtaining the relative track and record address (TTR) of the first text record from the data set directory entry, or obtaining the TTR of the needed segment from the note list.

- Converting the relative address to an absolute address, via a branch to a "convert" routine that is resident in the nucleus.

- Placing the absolute disk seek address in the program fetch input/output block (IOB), for later use by the I/O Supervisor.

Starting a Channel Program: The Program Fetch routine starts a channel program by issuing an EXCP macro instruction to obtain supervisor linkage to the I/O Supervisor. The IOB address is provided as an operand of the macro instruction.

The EXCP Supervisor, part of the I/O Supervisor, obtains control from the I/O First-Level Interruption Handler (I/O FLIH). The EXCP Supervisor issues a Start I/O instruction, followed by a Stand-Alone Seek command. The Stand-Alone Seek command moves the access arm of the direct-access device to the seek address contained in the IOB. The I/O Supervisor, via a Transfer in Channel command, then passes control to a fetch channel program, whose address the Program Fetch routine placed in its IOB. The fetch channel program causes the first text record to be read to main storage, beginning at the first assigned main storage address contained in the extent list.

After the channel program has been started, the I/O Supervisor returns control to the Program Fetch routine to await posting of an event control block by the I/O Supervisor or an appendage routine. Such posting indicates that one or two records have been read and that further processing can occur in the Program Fetch routine.

Reading of Module Records: The channel program causes the reading of two records, a text record and an RLD or control record, if the RLD or control record follows the text record. The text record is placed in its appropriate block of main storage. The RLD or control record is placed in an RLD buffer.

Switching of Channel Programs: If an RLD and control record, or a control record alone, does not follow a text record, control must be passed to another channel program to read a single record. The record must then be tested for control

Record 1 Control 20 bytes	Record 2 Text 500 bytes	Record 3 Control 20 bytes	Record 4 Text 1024 bytes	Record 5 RLD 260 bytes	Record 6 Control-RLD- End-of-Seg. 200 bytes	Record 7 Text 15 bytes

Figure 4-15. Typical Load-Module Logical Format on Direct-Access Device

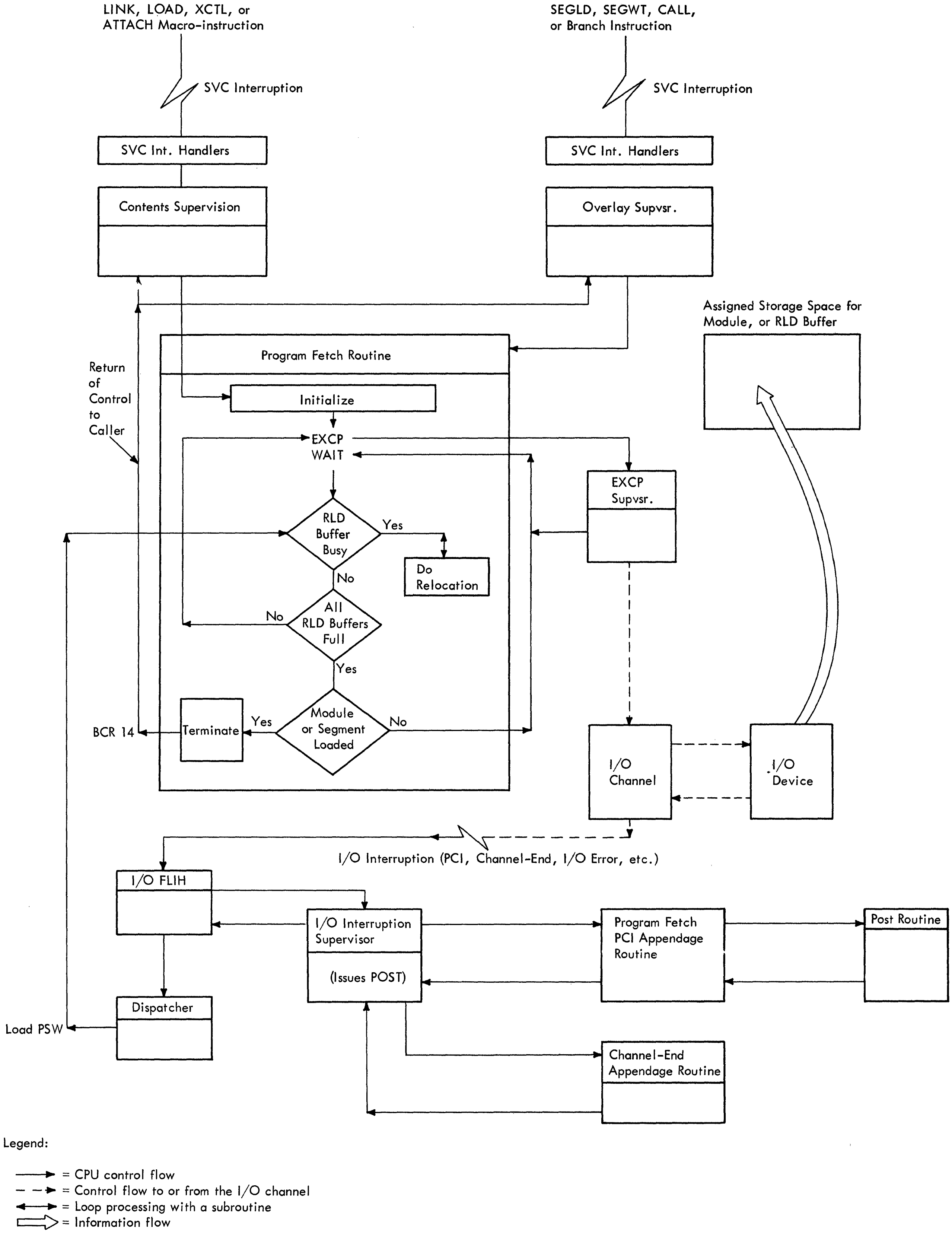

LINK, LOAD, XCTL, or ATTACH Macro-instruction
SVC Interruption
SVC Int. Handlers
Contents Supervision
SEGLD, SEGWT, CALL, or Branch Instruction
SVC Interruption
SVC Int. Handlers
Overlay Supvsr.
Assigned Storage Space for Module, or RLD Buffer
Return of Control to Caller
Program Fetch Routine
Initialize
EXCP
WAIT
EXCP Supvsr.
RLD Buffer Busy
Yes
Do Relocation
No
All RLD Buffers Full
No
Yes
Module or Segment Loaded
Yes
No
Terminate
BCR 14
I/O Channel
I/O Device
I/O Interruption (PCI, Channel-End, I/O Error, etc.)
I/O FLIH
I/O Interruption Supervisor
(Issues POST)
Program Fetch PCI Appendage Routine
Post Routine
Dispatcher
Load PSW
Channel-End Appendage Routine
Legend:
= CPU control flow
= Control flow to or from the I/O channel
= Loop processing with a subroutine
= Information flow

Figure 4-16. Overall Control Flow During the Loading of a Module or Segment

information. The Program Fetch PCI Appendage routine tests a record in the current RLD buffer and, when necessary, causes a channel-program switch between two-record mode and single-record mode. The PCI Appendage routine obtains control from the I/O Supervisor during the execution of any of the three fetch channel programs. (For overall control flow, see Figure 4-16.)

A channel command word in each channel program causes a program-controlled interruption (PCI). The PCI (a type of I/O interruption) causes supervisor linkage to the I/O Supervisor, which determines the cause of the interruption, and branches to the PCI Appendage routine. The PCI Appendage routine tests the buffer table and the current RLD buffer to determine the channel-program switching that is required. The processing that results from these tests is described in Table 4-3.

The I/O Supervisor processes a channel-end interruption, if the No-Operation command in a channel program is not altered before the channel program finishes. The I/O Supervisor gives control to the Program Fetch Channel-End Appendage routine. This routine tests if the entire module or segment has been loaded.

If the entire module or segment has been loaded, the Channel-End Appendage routine

Table 4-3. Channel-Program Switching After a Program-Controlled Interruption

Conditions	Resultant Processing by PCI Appendage Routine
1. The next RLD buffer is filled (busy).	1. Indicates in buffer table that all buffers are filled ("busy"). Does not alter current channel program, which continues in execution. Performs Step 7.
2. The last record (in current buffer) was either an RLD and control record, or a control record alone.	2. Initializes the next channel program to read a pair of records, starting with a text record. Alters the No-Operation (NOP) command in the current channel program to transfer-in-channel (TIC) to the next channel program to read a pair of records. Tests the last record (control information) to determine if the next text record is the last text record of the module or segment. (See Step 6.)
3. The last record was not an RLD record.	3. If the entire module or segment has not been loaded (see Step 5), alters the NOP command in the current channel program to transfer-in-channel (TIC) to the next channel program to read a single RLD or control record. Performs Step 7.
4. An extent boundary was crossed on the direct access device.	4. Obtains from the data extent block for the library the initial extent boundary for the next part of the module. Places the extent boundary into the appropriate unit control block. Computes new absolute seek address and places it in the IOBSEEK field of the IOB. These actions are in preparation for the issuance of another EXCP macro instruction.
5. The entire module or segment has been loaded.	5. Sets appropriate "end" flag and performs Step 7.
6. The next text record is the last text record of the module or segment (as indicated by the end-of-segment (EOS) or end-of-module (EOM) flag in the previous control record).	6. Prepares for the reading of a single text record by clearing the command chaining flag in the First Read Channel command word of the next channel program.
7. Processing described in Step 1, 3, or 5 has been performed.	7. Posts the fetch event control block (ECB) to prepare the Program Fetch routine for restart by the Dispatcher. Restart occurs at the instruction after the WAIT macro instruction.

returns control to the I/O Supervisor to post the I/O event control block (ECB), in preparation for the restarting of the Program Fetch routine. Control is passed from the I/O Supervisor to the Program Fetch routine, via the I/O First-Level Interruption Handler and the Dispatcher (see Figure 4-11). The Program Fetch routine then performs termination procedures.

If, however, the entire module or segment has not been loaded, the Channel-End Appendage routine returns control to the I/O Supervisor to restart the channel program.

RELOCATING ADDRESS CONSTANTS IN RELOCATION LIST DICTIONARY (RLD) RECORDS: The Program Fetch routine is restarted after the PCI Appendage routine or the I/O Supervisor has posted an ECB. The Relocation subroutine of the Program Fetch routine then examines the buffer table to determine whether an RLD record, containing relocatable address constants, is in an RLD buffer. The subroutine searches for a buffer table entry whose "busy" indicator is set. The indication means that the associated buffer contains an RLD record. When such a buffer is found, the Relocation subroutine relocates each address constant specified in the record. When RLD records in all "busy" buffers have been processed, the Program Fetch routine either restarts a channel program, if a buffer is empty, or issues a WAIT macro instruction to await the loading of another record.

The Relocation subroutine adjusts the value of an address constant by combining (adding or subtracting) a relocation factor with the value of the constant. Each RLD record contains the Linkage-Editor assigned address of the constant and a flag that indicates addition or subtraction of the relocation factor. (See "Relocation List Dictionary Record" in Section 12, "Control Blocks and Tables.")

For a block-loaded module, the relocation factor is the difference between its Linkage-Editor assigned address (usually zero) and the first byte of main storage into which the module has been loaded. The relocation factor is either added to or subtracted from the value field of each relocatable address constant. As an example, assume that a module is block-loaded into main storage, beginning at address 4000. If the flag bit in the RLD record is positive, a relocation factor of 4000 is added to the value field of each address constant. If, however, the flag bit in the RLD record is negative, 4000 is subtracted from the value field of the constant.

For an overlay module, relocation is similar to that just described, since an overlay module is effectively block-loaded. The root segment's relocation factor is used to adjust the address constants of all segments of the module. The Program Fetch routine stores the relocation factor in the note list, so that it is available in main storage throughout the module's execution (see Figure 4-14).

For a scatter-loaded module, each entry of an RLD record contains the Linkage-Editor assigned address of an address constant, a relocation pointer, and a position pointer. The position pointer is used to locate the address constant. The relocation pointer is used to find the relocation factor by which the address constant will be adjusted.

The position pointer is used to index the translation table to obtain a value that indicates the control section in which the address constant is located. The translation table value is then used to obtain a relocation factor from the scatter list. The relocation factor, when combined with the Linkage-Editor assigned address of the constant, yields the location of the address constant. (For more information on the translation table and scatter list, see the Linkage Editor PLM.)

The relocation pointer is similarly used as an index to obtain the relocation factor for the control section to which the address constant refers. This relocation factor is combined with the Linkage-Editor assigned value of the constant. The resultant relocated value is then placed in the value field of the constant.

TERMINATION: If the control record before the last text record contains an "end" indicator, the PCI Appendage routine sets an "end" flag to inform the Termination subroutine. After relocation has been performed, a test of the "end" flag causes the subroutine to be entered.

Table 4-4. Program Fetch Return Codes

Code	Meaning
X'00'	Successful Load
X'0C'	Invalid Scatter Information
X'0D'	Invalid Record Type
X'0E'	Invalid Address Encountered
X'0F'	Permanent I/O Error

The Termination subroutine performs its processing or waits, according to whether all I/O operations have been completed. When all I/O operations have been completed, the subroutine places in the return register a completion code to inform the caller of the result of the attempted loading (see Table 4-4).

The rest of the termination procedure depends on the type of module that has been loaded (see Table 4-5). When termination is complete, the Program Fetch routine returns control to the caller.

Table 4-5. Termination Processing According to Module Type

Type of Module	Processing by the Program Fetch Routine
Block-loaded module	Computes relocated entry-point address for the module, and places it in the fetch parameter list for use by the caller.
Scatter-loaded module	Computes the relocation factor for the entry-point address and places it in the fetch parameter list. The subroutines of Contents Supervision use this relocation factor to compute relocated entry-point addresses. Frees the space occupied by the scatter list/translation table.
Root segment of overlay module	Places in the segment table the main storage address of the data control block (DCB) and of the note list for use by the Overlay Supervisor.

Main storage space is a resource and, like other resources, is shared by many users. Allocation of space must be controlled, and space must be requested when it is needed and be freed when it is no longer needed. Control over space allocation is excercised by the routines of Main Storage Supervision and by the routines of the optional rollin/rollout module. The Main Storage Supervision routines service two macro instructions: GETMAIN, which is used to allocate space; and FREEMAIN, which is used to free space that was previously allocated. Each macro instruction results in an SVC interruption and entry to a corresponding service routine.

Requests for allocation of main storage space are serviced by Main Storage Supervision elements collectively called the GETMAIN routine. This routine services all requests for space, including requests for a region, space within an existing region, or space in the system queue area. By keeping and continually updating control blocks that record where space is available, the GETMAIN routine can determine where and how a request may be satisfied.

Requests to free main storage space are serviced by Main Storage Supervision elements collectively called the FREEMAIN routine. This routine updates control blocks to reflect the change of status of the freed space, thereby making the space available for reallocation by the GETMAIN routine.

An unconditional request for the allocation of main storage space in an existing region, if unsatisfied by the GETMAIN routine, can cause the GETMAIN routine to schedule linkage to the rollout/rollin module. This extra effort to obtain the requested space is possible if the rollout feature is included in the system and if the requestor belongs to a job step eligible to cause rollout. The rollout/rollin module is not scheduled if the requestor is a system routine, if the request is for space in the system queue area, or if the request is for a region in which to start a new job step.

The rollout/rollin module, when executed for the GETMAIN routine, tries to obtain a temporary additional region for use by the requestor's task and other tasks of its job step. This is necessary since the requesting job step needs more space than is available in its existing region. The rollout/rollin module first tries to allocate the temporary region from unassigned space in the dynamic area. If sufficient unassigned space is not available, the rollout/rollin module then searches for a suitable job step of another job that it may roll out. A job step is suitable to be rolled out if its dispatching priority is lower than that of the requestor's job step, its job step TCB is flagged eligible to be rolled out, and if it is not using or waiting for a system resource for which it has issued an ENQ macro instruction.

If the rollout/rollin module finds a suitable job step whose region is large enough to satisfy the current request, it waits for completion of active I/O commands, suspends pending I/O commands, defers pending operator replies, and transfers (rolls out) to auxiliary storage the contents of the selected job step's region. It then builds and initializes control blocks to allocate the rolled out region to the requestor's job step. The rollout/rollin module returns control to the requestor, which reissues its original GETMAIN macro instruction, causing supervisor linkage to the GETMAIN routine. The GETMAIN routine then services the request from the region just obtained through rollout.

At key decision points in the rollout processing there are dummy user routines which the user may replace with his own optional appendages. The user-written appendages may do the following:

- Determine whether more than one job step can concurrently obtain space through rollout of other job steps' regions. Such an option is called "multiple rollouts."

- Decide whether a region belonging to a job step of <u>higher</u> dispatching priority than the requestor's job step should be rolled out.

- Decide if a job step should be abnormally terminated, if there is no job step suitable to be rolled out. Abnormal termination could be selected in place of the standard alternative of placing the requestor's job step on a wait queue, pending a new attempt at rollout.

- Specify additional criteria that must be met by a job step before it can be rolled out.

After the requestor's job step has completed its use of the borrowed region (signalled by issuance of a FREEMAIN macro instruction), the FREEMAIN routine schedules linkage to the rollout/rollin module. This time the module transfers (rolls in) the contents of the rolled out job step's region from auxiliary storage to its originally assigned location in main storage. Deferred I/O commands and deferred operator replies are then restored to the job step. The rollout/rollin module returns control to the current routine of the highest priroity ready task, via the Exit routine and the Dispatcher.

INTERRUPTION HANDLING FOR MAIN STORAGE SUPERVISION

Both the GETMAIN and FREEMAIN macro instruction may be expressed by programmers in two forms. S (storage) type macro instructions are used when parameters are supplied in a parameter list, and R (register) type macro instructions are used when parameters are supplied in general registers. Figure 5-1 shows the SVC instructions contained in expansions for each type.

When any SVC instruction is executed, an SVC interruption occurs and control is given to the SVC First-Level Interruption Handler, which saves a record of the interrupted environment and routes control to an appropriate SVC service routine. A description of SVC first-level interruption handling is contained in the section "SVC Interruption Handling". Figure 5-2 shows the handling of interruptions resulting from issuance of GETMAIN and FREEMAIN macro instructions.

For SVC 4 and SVC 5 instructions, the SVC First-Level Interruption Handler gives control to the GETMAIN and FREEMAIN routines, respectively. For SVC 10 instructions, it gives control to the REGMAIN routine, which examines register 1 to determine whether a GETMAIN or FREEMAIN macro instruction was given, and routes control accordingly.

Macro Instruction	Type	SVC Instruction
GETMAIN	S	SVC 4
	R	SVC 10*
FREEMAIN	S	SVC 5
	R	SVC 10*
*High-order bit of register 1 will contain 1 for GETMAIN; 0 for FREEMAIN.		

Figure 5-1. GETMAIN/FREEMAIN SVC Instructions

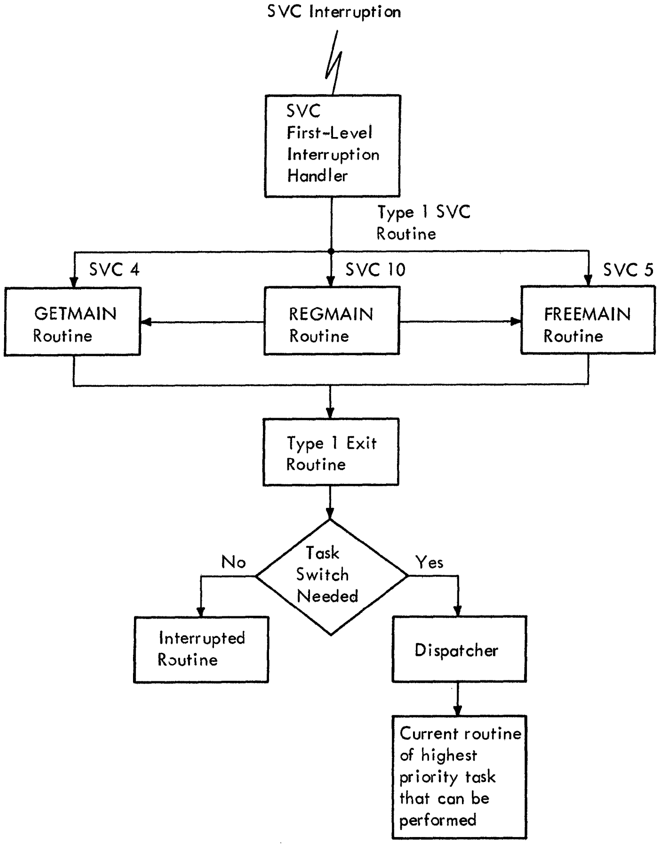


Figure 5-2. Main Storage Supervision Interruption Handling

When any SVC instruction is executed, an SVC interruption occurs and control is given to the SVC First-Level Interruption Handler, which saves a record of the interrupted environment and routes control to an appropriate SVC service routine. A description of SVC first-level interruption handling is contained in the section "SVC Interruption Handling". Figure 5-2 shows the handling of interruptions resulting from issuance of GETMAIN and FREEMAIN macro instructions.

For SVC 4 and SVC 5 instructions, the SVC First-Level Interruption Handler gives control to the GETMAIN and FREEMAIN routines, respectively. For SVC 10 instructions, it gives control to the REGMAIN routine, which examines register 1 to determine whether a GETMAIN or FREEMAIN macro instruction was given, and routes control accordingly.

The GETMAIN, FREEMAIN, and REGMAIN routines are type 1 SVC routines. After the GETMAIN and FREEMAIN routines have completed their processing, they give control to the Type-1 Exit Routine. The Type 1

Exit routine determines whether the task for which the SVC instruction was executed is to be reinstated. If so, it restores the saved contents of registers and returns control to the routine in which the SVC instruction was encountered. If, however, a different task is to gain control, the Type-1 Exit routine saves register contents in the current TCB, saves the SVC old PSW in the current request block, and branches to the Dispatcher. The Dispatcher routes control to the current routine of the highest priority ready task.

ALLOCATING MAIN STORAGE

All requests for space are handled by the GETMAIN routine. These include requests for regions, space within regions, and space in the supervisor queue area of main storage. Basically, the GETMAIN routine scans queues of elements that represent available space to locate the amount of space of the type requested. When the space is found, the GETMAIN routine updates the affected queues to reflect its subsequent unavailability and returns the address of the space to the requestor. If the requested space is not available, the GETMAIN routine responds according to the type of storage that is requested: a new region, space within an existing region, or space in the system queue area.

If requested space for a new region is not available, and the request is conditional, the GETMAIN routine sets up a return code and returns control to the requestor, via the Type-1 Exit routine. If, however, the request is unconditional, the GETMAIN routine makes the requestor's task nondispatchable, pending the availability of sufficient free space in the dynamic area, and causes control to be given to the current routine of the highest priority ready task.

If requested space within an existing region is not available, and the request is conditional, the GETMAIN routine sets up a return code and returns control to the requestor, via the Type-1 Exit routine. If, however, the request is unconditional, the GETMAIN routine tries to find space that may be freed and allocated to the requestor's task. It first searches for unused modules in the requestor's region that may be purged. If sufficient space cannot be made available by the module purge, and if the rollout feature cannot be used, the GETMAIN routine causes the abnormal termination of the requestor's task. If, however, the rollout feature is part of the system and the requestor's task is eligible to cause rollout, the GETMAIN routine schedules linkage to the rollout/rollin module. The rollout/rollin module tries to obtain temporary allocation of an additional region for use by the requestor's job step. The additional region may be obtained either from free space in the dynamic area or by temporary reallocation of a region previously allocated to a job step of another job. If the rollout/rollin module cannot find the needed region, it either causes the abnormal termination of the requestor's job step or another job step, or makes the requestor's job step temporarily nondispatchable pending the availability of the needed region. The choice depends on the option specified in a user-written appendage.

If requested space in the system queue area is not available, the GETMAIN routine tries to expand that area. It does this, if possible, by adding to the system queue area the space that lies adjacent to it in the dynamic area. If the request can now be serviced, space is allocated to the requestor. Otherwise, the GETMAIN routine causes the CPU to be placed in the wait state. In a multiprocessing system, if the system queue area is expanded, the new size and origin of the dynamic area is placed in the PQE.

Following entry to the GETMAIN routine, the Subpool Check (CSPCHK) subroutine is entered to determine what type of space is requested. Table 5-1 shows the subpool numbers associated with each type of request.

ALLOCATING A REGION

Space for regions is obtained from the dynamic area of main storage (see Figure 5-3). The PQEPTR field at offset 8 in location GOVRFLB contains the address of a two-word dummy partition queue element (DPQE). Word one of the DPQE contains the address of a partition queue element (PQE) that describes unassigned processor storage not belonging to any region. Word two of the DPQE contains the address of the last PQE constructed by NIP. Word three of the PQE for hierarchy 0 contains the address of the PQE that describes unassigned LCS not belonging to any region. A free block queue element (FBQE) is located in the first three words of each type of storage. The first two words of the corresponding PQE contain the address of its FBQE. If Main Storage Hierarchy Support is not included in the system, only the PQE for processor storage is constructed and the last PQE's pointer to the next PQE (PQEPTR) is set to zero.

• Table 5-1. Subpool Numbers Used for Requesting Space

Subpool No.	Signifies Request for:	Storage Key Assignment	Notes
246	Region		Signifies request to free existing region and assign new region.
247	Region		Signifies request to assign new region or free existing region.
248	Region		Signifies request from Rollout/Rollin routine to assign a region
0-127	Space within region	Job step's storage protection key (reset to 0 when space is freed)	When subpool 0 is requested by programs executing in supervisor state, subpool 252 is assigned.
250	Space within region	Job step's storage protection key (reset to 0 when space is freed)	When requested by programs executing in supervisor state, subpool 0 is assigned.
251	Space within region	Job Step's Storage protection key (reset to 0 when space is freed)	
252	Space within region	0 storage protection key	
253	Space within system queue area	0 storage protection key	Assigned space will be freed when task terminates.
254	Space within system queue area	0 storage protection key	Assigned space will be freed when job step terminates.
255	Space within system queue area	0 storage protection key	Assigned space must be explicitly freed.

To assign a region, the GETMAIN routine first determines the beginning address of the region:

```
Beginning  =  Size of Dynamic Area + Begin-
Address       ning Address of Dynamic Area
              - Size of Region Requested
```

The GETMAIN routine then subtracts the number of bytes to be occupied by the region from the number of bytes in the FBQE that represents the dynamic area.

For each region, the GETMAIN routine builds a free block queue element (FBQE) at the beginning of the region and a dummy partition queue element and a partition queue element (PQE) in the system queue area (see Figure 5-3). The GETMAIN routine places in the free block queue element a count of the number of contiguous free bytes that can be allocated in the region. The dummy partition queue element is made to point to the partition queue element, which in turn is given a pointer to the free block queue element. The GETMAIN routine places in the PQE the size of the region and the region address. It places the address of the dummy PQE in the TCBPQE field of the TCB of the job step task for which the region was requested. If Main Storage Hierarchy Support is included in the system, regions may be requested in either hierarchy, or a region segment may be requested in both hierarchies. A PQE is constructed for each region segment and both PQEs are chained (by way of a dummy PQE, as associated with GOVRFLB) to the TCB that represents the task for which the region was requested. (For the formats of the dummy PQE, PQE, and FBQE, see Section 12, "Control Blocks and Tables.")

The GETMAIN routines additionally support obtaining a region at a specific storage address and quiescing the system if

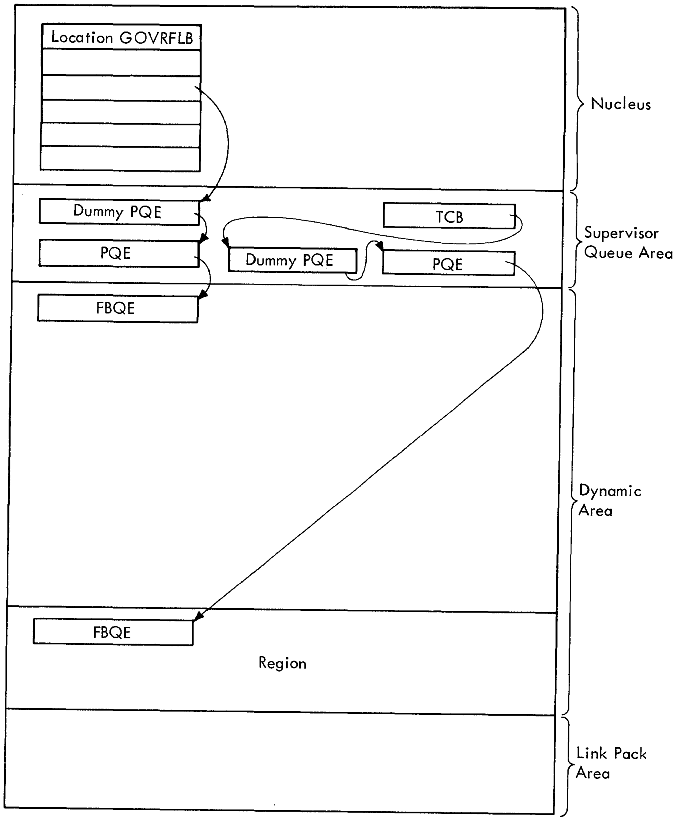


Figure 5-3. Element Relationships: Region Allocation

a valid request for a region at a specific address cannot be satisfied.

The function of obtaining a region is performed by the GETPART module, invoked by expansion of the GETMAIN macro instruction.

In order to obtain a region at a specific main storage address, the list form of the macro instruction must be used. The list contains an address pointer and a length pointer; the address pointer indicates the location of a list containing the addresses at which storage is to be obtained, the length pointer points to a corresponding list of lengths specifying the size of each of the requested regions. Figure 5-4 shows the lists and pointers; Table 5-2 shows the subpool use for list and register forms of GETMAIN requests for region allocation.

If a request contains a specific address which is not in either the dynamic area (between the system queue area and the link pack area) or within hierarchy one in systems with Main Storage Hierarchy Support, the GETPART module returns with a code of X'08' in register 15. If the address is valid, but not enough storage is available, the requester is placed in a wait condition and no further requests, except for subpool 248(from Rollout/ Rollin), are accepted until the first specific address request is satisfied.

In a multiprocessing system, if the requested storage area is not available, GETPART determines from FSSEMAP whether any of the storage has been logically removed from the system. (See Section 12 "Control Blocks and Tables" for a description of FSSEMAP.) A storage area may be marked offline in FSSEMAP if (1) a VARY STORAGE OFFLINE command has been issued, (2) the storage address range is set disabled (determined by the Multiprocessing NIP routine), or (3) the storage area is malfunctioning (determined by the Multiprocessing NIP routine). If any of the requested storage area is marked offline in FSSEMAP, GETPART returns with a code of X'08' in register 15, a message is issued that main storage is not available, and the job is abnormally terminated. If the storage is not marked offline, the requestor is placed in a wait condition until the request can be satisfied.

If a list request with more than one entry cannot be completely satisfied, all storage already obtained for the request is returned to the system.

A FREE/GET (EXCHANGE) request for a specific address must be issued using the list form and must specify subpool 246. GETPART frees the region and replaces it with one at the address specified. In systems with Main Storage Hierarchy Support, only the region in hierarchy 0 is freed. All FREE/GET requests for specific addresses are assumed to be within the boundaries of the original region; no provision is made to handle an invalid request. In the list form, the address entries must contain the hierarchy identification in the high order byte if the system includes Main Storage Hierarchy Support.

Table 5-2. Subpool Use for List and Register Forms of GETMAIN (GETPART Module)

Subpool No.	List Request	Register Request
246	Free, then get region Address = 0, get region anywhere Address ≠ 0, get region at specified address	Free, then get region
247	Address = 0, get region anywhere Address ≠ 0, get region at specified address	Register 1 negative, get region Register 1 zero or positive, free region
248	Request from Rollout/Rollin	Request from Rollout/Rollin

If the dynamic area does not contain sufficient free space for the requested region, the GETMAIN routine responds according to whether the GETMAIN request is conditional or unconditional. If the request is conditional, the GETMAIN routine places a return code (4) in register 15 to inform the requestor that space cannot be allocated. It then returns control to the requestor, via the Type-1 Exit routine. If, however, the request is unconditional, the GETMAIN routine makes the requestor's task nondispatchable, prepares for future reissuance of the request, and causes control to be routed to the current routine of the highest priority ready task. It does this by:

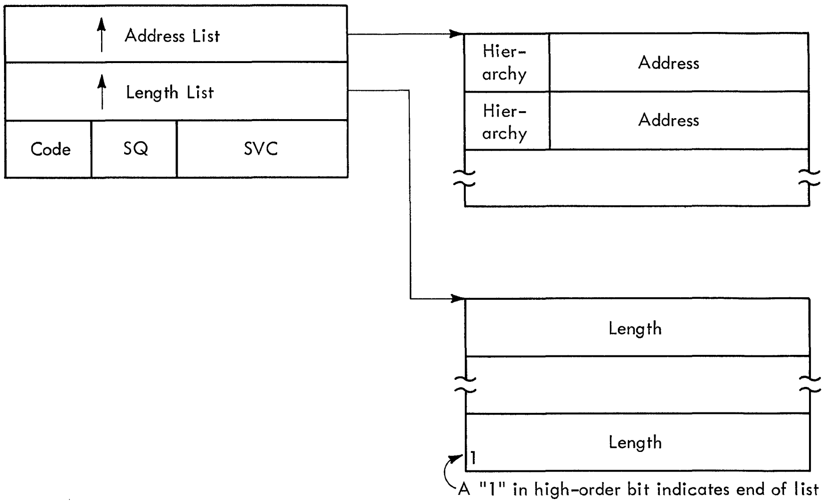
- Setting the TCBFCD1 nondispatchability flag in the requestor's TCB.
- Pointing the SVC old PSW to the invoking GETMAIN macro instruction, and storing this restart address in the requestor's RB old PSW.
- Indicating to the Dispatcher that a task switch is needed. (It does this by placing zero in the "new" TCB pointer IEATCBP.)
- Branching to the Type-1 Exit routine, which detects the task switch indication of the "new" TCB pointer. The Type-1 Exit routine then branches to the Dispatcher to locate the highest priority ready task whose current routine will be given control.

ALLOCATING SPACE WITHIN A REGION

Any GETMAIN macro instruction in which subpools 0-127, 250, 251, or 252 are specified indicates that space within an existing region is desired.

Processing If the Requested Space Is Available

When the initial request for a subpool is received, the GETMAIN routine builds a subpool queue element (SPQE) in the super-



A "0" Address indicates normal GETPART for region

A non-zero Address indicates specific region start address for GETPART

A "1" in high-order bit indicates end of list

Figure 5-4. List Structure for List Form of GETMAIN Macro Instruction

visor queue area (see Figure 5-5). The SPQE contains the subpool number and, if other subpools exist, a pointer to another SPQE. (Each time a request is received, the chain of SPQEs is scanned by the GETMAIN routine to determine whether the requested subpool exists.)

The GETMAIN routine also builds a descriptor queue element (DQE) in the supervisor queue area, and places the address of the DQE into the subpool queue element. The DQE contains a count of the number of bytes of main storage allocated to a block in the subpool (space within regions is assigned in 2048-byte blocks). For each subsequent request for space in the same subpool that cannot be satisfied with space defined by existing DQEs, the GETMAIN routine builds another DQE. All DQEs representing space in the same subpool are chained together. After each 2048-byte block is assigned, it is given a storage protection key (see Table 5-1). Then, when each block is freed, its storage protection key is reset to zero.

If any free space exists within the 2048-byte blocks defined by a DQE, the GETMAIN routine builds a free queue element (FQE) within the 2048-byte block that contains the free space, and places into it a count of the number of bytes available. All such FQEs within one contiguous area are chained together; the GETMAIN routine places the address of the first such FQE into the associated DQE. FQEs built in space assigned to subpools 0-127, 250, or 251 are exposed to accidental damage by job steps, as the space is assigned the storage protection keys of the steps. These FQEs are the only supervisor queue elements so

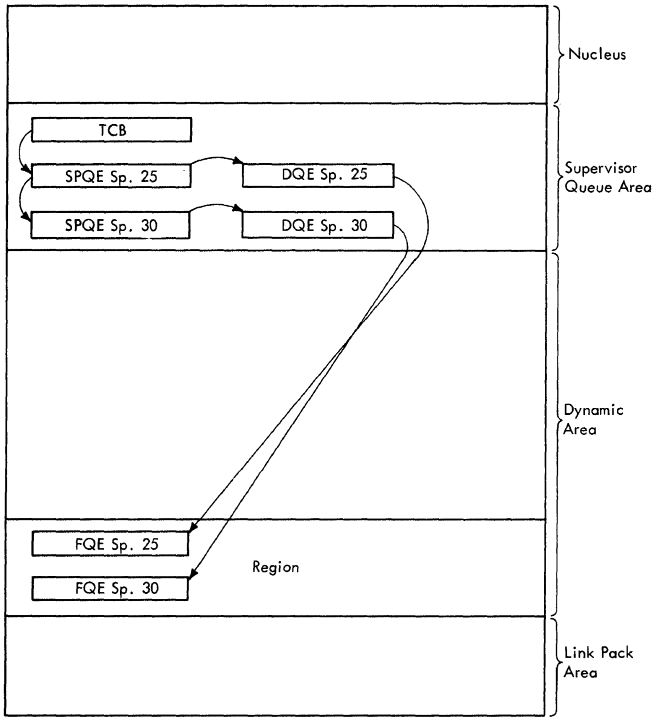


Figure 5-5. Element Relationships for Intra-Region Allocation

exposed. All others are built in areas that are assigned the supervisor storage protection key.

To locate free space in an existing subpool, the GETMAIN routine first locates the subpool by scanning the chain of SPQEs. It then determines the address of the first DQE and scans the chain of DQEs to locate an FQE containing sufficient space to satisfy the request. If sufficient space exists, the GETMAIN routine decrements the count of available bytes in the FQE. If sufficient free space to satisfy the request does not exist in the requested subpool, the GETMAIN routine locates space not yet assigned to any subpool, and adds the space to the requested subpool by building a DQE.

After space is assigned, the GETMAIN routine places the address of the assigned space into register 1 if an SVC 10 instruction caused entry, or places the address into the location specified by the programmer if an SVC 4 instruction caused entry.

Processing if the Requested Space is not Available

If there is not enough free space in the region to satisfy the request, the GETMAIN routine enlarges the scope of its search by:

- Purging unused modules in the region.
- Examining a region previously borrowed by the requestor's job step through rollout, if the rollout feature is part of the system.
- Testing whether to schedule linkage to the rollout/rollin module to "borrow" an additional region.

ATTEMPTING TO FREE SPACE BY PURGING UNUSED MODULES: The GETMAIN routine branches to its CDPURGE routine to attempt to purge one or more unused modules in the requestor's region. The space freed by this purge may be sufficient to satisfy the current storage request. If the purge flag is set (hex. '80') in the TCBJPQ field of the job step TCB, the CDPURGE routine examines all contents directory entries (CDEs) in the job pack queue. Each CDE that has its "release" flag (REL) set in its attributes field represents a module in the region that is no longer needed. That is, there are no outstanding requests for the module by any routine in the job step. For each such module the CDPURGE routine branches to the CDDESTRY routine (in CDEXIT) to dequeue the CDE and free the associated module and its extent list. After all CDEs in the region's job pack queue have been examined and all unused modules purged, the CDPURGE routine returns control to the main line of the GETMAIN routine.

If the module purge has freed enough space to satisfy the request, the GETMAIN routine allocates the needed space to the requestor's task. It then returns control to the requestor, via the Type-1 Exit routine.

EXAMINING A PREVIOUSLY BORROWED REGION: If sufficient space cannot be freed by the module purge, the GETMAIN routine determines if there is a possiblity of satisfying the storage request from space outside the requestor's region. The requestor's job step may previously have "borrowed" an additional region through the action of the rollout feature. If so, the borrowed region is searched, via a branch to the GMCOMMON routine. If the request is conditional and there is no borrowed region or the borrowed region is searched to no avail, the GETMAIN routine sets up a return code (4) and returns control to the requestor, via the Type-1 Exit routine. If, however, the request is unconditional and the rollout feature is not part of the system, the GETMAIN routine must cause the abnormal termination of the requestor's task. It sets up a condition code (hex. '804') and branches to the ABTERM routine to schedule the abnormal termination.

DETERMINING WHETHER TO SCHEDULE LINKAGE TO THE ROLLOUT/ROLLIN MODULE: If requested space in an owned or borrowed region is not available, the GETMAIN routine determines if it can schedule the rollout/rollin module to borrow, if possible, an additional region for use by the job step. The GETMAIN routine schedules linkage to the rollout/rollin module only if the following requirements are met:

- The request is unconditional.
- The rollout feature is part of the system.
- The request is made by a user routine.
- The requestor's task belongs to a job step that is eligible to cause rollout. (The eligibility is indicated by the 'set' condition of the TCBFRA flag in the job step TCB. Such eligibility was established by a JOB or EXEC statement parameter (ROLL) when the job entered the input stream. The eligibility was recorded in the job step TCB by the Attach routine when an initiator attached the job step.)

Unless all of the above requirements are met, the GETMAIN routine cannot make space available to satisfy the storage request. It therefore sets up a condition code (hex. '804') to indicate that storage is unavailable, and branches to the ABTERM routine to schedule the abnormal termination of the requestor's task.

SCHEDULING LINKAGE TO THE ROLLOUT/ROLLIN MODULE: The GETMAIN routine schedules linkage to the rollout/rollin module (hereafter called the RO/RI module) by means of the asynchronous exit mechanism. (This mechanism is described in "Scheduling a User Exit Routine" in Section 3, "Task Supervision.") Like the scheduling of other asynchronous exit routines, the scheduling of the RO/RI module involves the Stage 1 Exit Effector, the Stage 2 Exit Effector, and the Stage 3 Exit Effector. Only stages 2 and 3, however, are involved directly in the GETMAIN routine's attempt to schedule the RO/RI module. The Stage 1 Exit Effector is used by the Nucleus Initialization Program during system initialization.

If the rollout feature is to be part of the system, the Nucleus Initialization Program (NIP) uses the CIRB macro instruction to invoke the Stage 1 Exit Effector. Stage 1 then gets space for and initializes a special permanent system IRB and a 240-byte work area. The IRB is called the rollout/rollin IRB and is used by the supervisor to schedule and control the RO/RI module. The NIP formats the 240-byte work area into ten combined interruption queue elements (IQEs) and rollout/rollin parameter lists. Each IQE is used in scheduling linkage to the RO/RI module. Each associated parameter list provides input information, such as the requestor's TCB address, needed by the RO/RI module. (See the IQE format in Section 12, "Control Blocks and Tables" for the format of a rollout/rollin IQE parameter list.)

Execution of the RO/RI module occurs under control of a special permanent system TCB of high dispatching priority. This TCB, called the rollout/rollin TCB, is created during the nucleus initialization procedure, if the rollout feature is to be part of the system.. The position of the RO/RI TCB on the TCB queue, and therefore its dispatching priority relative to the other permanent system TCBs, is shown in Figure 5-6.

Rollout and rollin processing are performed as part of the rollout/rollin task (hereafter called the RO/RI task). This task is held nondispatchable when linkage to the RO/RI module is not needed. The task is nondispatchable because its TCB points directly to a permanent rollout/rollin PRB that is kept in a wait condition. While the task is nondispatchable, the rollout/rollin PRB, in turn, points to the rollout/rollin IRB. (See part 1 of Figure 5-7.) When linkage to the RO/RI

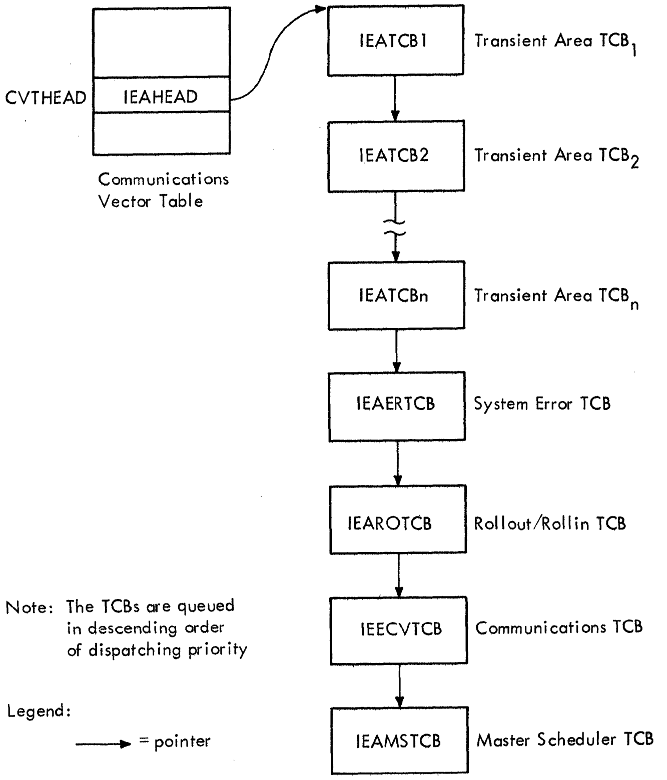


Figure 5-6. Position of Rollout/Rollin TCB on TCB Queue

module is needed, the scheduling process reverses the position of the PRB and the IRB on the RO/RI task's RB queue. (See part 2 of Figure 5-7.) Since the RO/RI IRB is usually in a ready condition (its wait count equal to zero), the reversal of the position of the two RBs make the RO/RI task dispatchable.

Scheduling of the RO/RI module occurs in two phases. Initial scheduling is done by the SHEDRO routine, a subroutine of the GETMAIN routine. Final scheduling is performed by the Stage 3 Exit Effector, after the GETMAIN routine has exited and the Dispatcher has been entered. The Stage 3 Exit Effector is a subroutine of the Dispatcher. The Stage 3 Exit Effector readies the RO/RI task, which is then given control by the Dispatcher. The processing is described in the next two topics. (See Figure 5-8 for the overall flow and Figure 5-9 for a pictorial summary of the processing.)

Initial Scheduling of the Rollout/Rollin Module: The GETMAIN routine uses its subroutine, the SHEDRO routine, to perform the following main functions:

- Obtains an interruption queue element (IQE) and rollout/rollin parameter list. Initializes both the IQE and the parameter list.

- Places the IQE on the asynchronous exit queue (AEQJ), via a branch to the Stage 2 Exit Effector.

- Prepares for a task switch and for eventual return of control to the requestor.

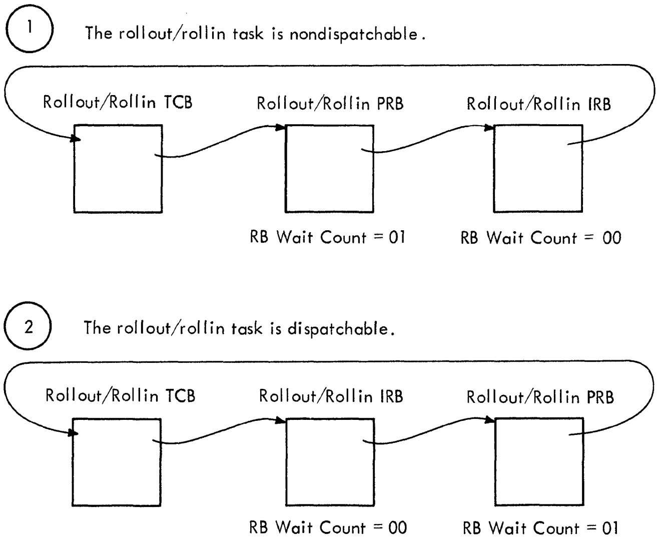


Figure 5-7. Relationship of the Rollout/Rollin TCB, PRB, and IRB During Scheduling of the Rollout/Rollin Task

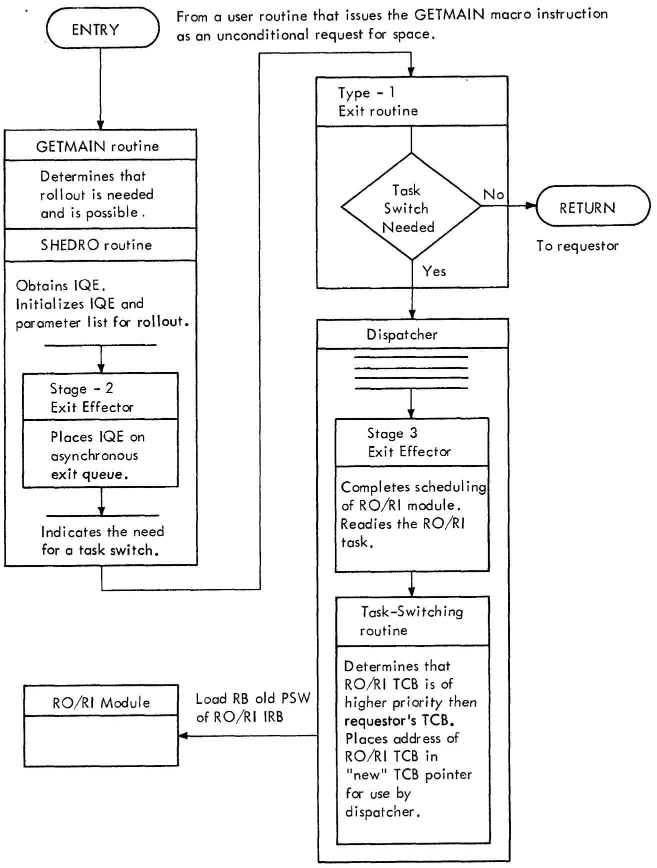
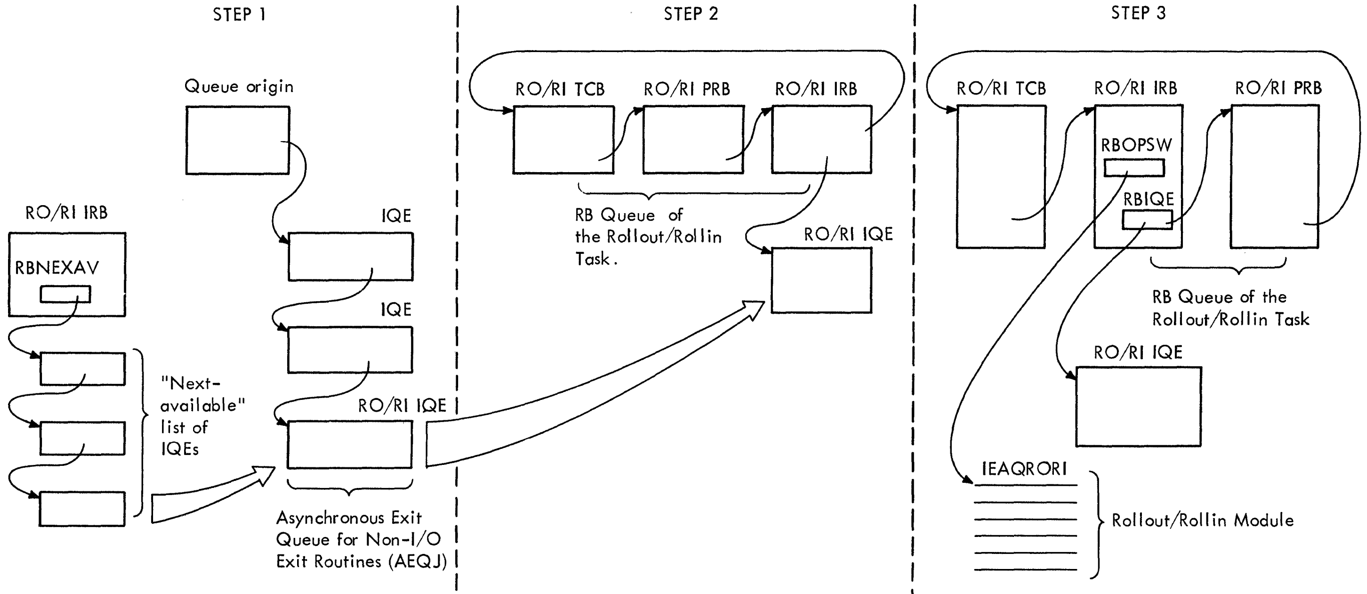

ENTRY
From a user routine that issues the GETMAIN macro instruction as an unconditional request for space.
GETMAIN routine
Determines that rollout is needed and is possible.
SHEDRO routine
Obtains IQE. Initializes IQE and parameter list for rollout.
Stage - 2 Exit Effector
Places IQE on asynchronous exit queue.
Indicates the need for a task switch.
Type - 1 Exit routine
Task Switch Needed
No
RETURN
To requestor
Yes
Dispatcher
Stage 3 Exit Effector
Completes scheduling of RO/RI module. Readies the RO/RI task.
Task-Switching routine
Determines that RO/RI TCB is of higher priority then requestor's TCB. Places address of RO/RI TCB in "new" TCB pointer for use by dispatcher.
RO/RI Module
Load RB old PSW of RO/RI IRB

Figure 5-8. Scheduling of Rollout: Overall Flow

Obtaining a Rollout IQE and Parameter List: The SHEDRO routine obtains an IQE and parameter list to keep track of the rollout request, and to schedule and control the execution of the RO/RI module. It obtains the IQE and parameter list by means of its GETIQE routine (invoked at location IQEROUT). The GETIQE routine obtains them, if possible, from a "next available" list (RBNEXAV) queued from the RO/RI IRB. If there are no more available IQEs, the GETIQE routine obtains the needed space (24 bytes, subpool 255), via a branch to the GETMAIN routine. If it obtains space, the routine initializes the IQE and parameter list. After the GETIQE routine has obtained the IQE and parameter list, it returns control to the SHEDRO routine. The SHEDRO routine then initializes the IQE to indicate a rollout request, and places in the parameter list the address of the requestor's TCB and the size of the requested space.



The SHEDRO routine obtains a rollout/rollin IQE either from an available list or by getting space and initializing a new IQE. The SHEDRO routine then invokes the Stage 2 Exit Effector to place the IQE on the queue.

IQE for rollout/rollin is removed from the asynchronous exit queue and is queued from the rollout/rollin IRB by the Stage 3 Exit Effector when the Dispatcher is next entered.

The rollout/rollin TCB is readied by the Stage 3 Exit Effector by its rearrangement of the task's RB queue. The rollout/rollin TCB now points to the rollout/rollin IRB, which is ready. Since the rollout/rollin task is now ready and of very high dispatching priority, the Dispatcher gives control to this task at location IEAQRORI in the rollout/rollin module.

Legend:

RO/RI = rollout/rollin
RBOPSW = RB old PSW
———→ = pointer

Figure 5-9. Steps in the Scheduling of the Rollout/Rollin Task

<u>Placing the IQE on the Asynchronous Exit Queue</u>: The SHEDRO routine uses its SCHEDIRB subroutine to invoke the Stage 2 Exit Effector. The Stage 2 Exit Effector then places the IQE representing the rollout request onto the asynchronous exit queue. (See Figure 5-9.) This is the same queue on which the Stage 2 Exit Effector places IQEs that represent requests for an end-of-task exit routine (ETXR) or a timer exit routine. The Stage 3 Exit Effector will, when the Dispatcher is next entered, complete the scheduling of the exit routines whose IRBs are represented on the queue. Although the IQEs are placed on the asynchronous exit queue in first-in, first-out order, the represented requests will be serviced by the Stage 3 Exit Effector on a task-priority basis.

<u>Preparing for a Task Switch and for Eventual Return of Control to the Requestor</u>: The SHEDRO routine does three things to prepare for a task switch and to provide for eventual return of control to the requestor:

- Indicates to the Type-1 Exit routine that a task switch is needed.
- Makes the requestor's task nondispatchable (sets the TCBWFC flag).
- Points the SVC old PSW to a restart address in the requestor's task.

The SHEDRO routine indicates the need for a task switch by storing zero in the "new" TCB pointer (IEATCBP). Without such an indication, the Type-1 Exit routine, when entered during the exiting procedure from GETMAIN, would return control to the routine that had issued the GETMAIN macro instruction. With the task switch indication, the Type-1 Exit routine will branch to the Dispatcher, which will then determine the task to which it will give control.

The SHEDRO routine makes the requestor's task nondispatchable to prevent accidental redispatching of the requestor's task before its needed storage space has been allocated.

The SHEDRO routine points the SVC old PSW to the GETMAIN macro instruction issued by the requestor. (This procedure is described in the program listing as "backing up the PSW," since it causes the restart address to be two bytes earlier in the requesting routine than the normal address in the SVC old PSW.) The old PSW is altered so that when rollout is successful, the requestor can be redispatched to reissue its GETMAIN macro instruction. The GETMAIN routine will then be entered, via supervisor linkage, to satisfy the request from the newly borrowed region.

Final Scheduling of the Rollout/Rollin Module: During the exiting procedure from the GETMAIN routine, the Type-1 Exit routine is entered, detects that a task switch is needed, and branches to the Dispatcher. The Dispatcher, finding that there is at least one IQE on the asynchronous exit queues, enters the Stage 3 Exit Effector to complete the scheduling of the appropriate asynchronous exit routine. In this case the appropriate exit routine is the RO/RI module. To complete the scheduling of the RO/RI module, the Stage 3 Exit Effector performs the following main functions:

- Removes the RO/RI IQE from the asynchronous exit queue and places it on the list of IQEs queued from the RO/RI IRB. (The IRB's list origin for IQEs is RBIQE. (See Figure 5-9.)

- Readies the RO/RI task.

- Indicates to the Dispatcher that it should next dispatch the RO/RI task.

- Moves the address of the RO/RI parameter list from the IQE to register 1 to serve as input information for the RO/RI module.

The queuing of the IQE to the RO/RI IRB is recognition by the Stage 3 Exit Effector that the IQE represents a request for execution of the RO/RI module under control of the RO/RI TCB. The IQE will remain queued from the IRB throughout rollout processing. When the RO/RI module completes its processing of the rollout request, it will dequeue the IQE from the IRB's active queue and return it to the IRB's "next available" list (RBNEXAV).

The Stage 3 Exit Effector readies the RO/RI task by reversing the order of the PRB and IRB on the RO/RI task's RB queue, as illustrated in Figures 5-7 and 5-9. Since the IRB is normally ready and the RO/RI TCB has no nondispatchability flag set, the task is dispatchable as soon as its RB queue is reordered.

The Stage 3 Exit Effector then indicates to the Dispatcher that it should next dispatch the RO/RI task. Stage 3 does this by invoking the supervisor's Task Switching routine and passing to it the address of the RO/RI TCB. The Task Switching routine compares the dispatching priority of the RO/RI TCB with that of the requestor's task, and determines that the RO/RI task is ready. Since the RO/RI task is of extremely high dispatching priority and is ready, the Task Switching routine selects the RO/RI TCB and places its address in the "new" TCB pointer as information for the Dispatcher. The invoking of the Task Switching routine is necessary, since otherwise the Dispatcher would remain unaware that a task is ready that is higher in priority than the current task. The Dispatcher can never discover a higher priority ready task by searching the TCB queue. When it searches the TCB queue, it searches in a downward-priority direction, beginning with the current TCB.

The address of the RO/RI parameter list, when moved from the IQE to register 1, serves an important purpose. It indicates to the RO/RI module the type of service that it should perform. If the address is positive, the request is for rollout. If, however, the address is negative, the request is for rollin. Lastly, if the address is zero, the request is to reschedule rollout processing for deferred rollout requests. These requests had earlier caused entry to the RO/RI module, but a job step suitable to be rolled out could not be found. (The handling of deferred rollout requests will be described later in "Processing If a Job Step Suitable for Rollout Cannot Be Found" and "Performing Final Common Processing.")

ALLOCATING A BORROWED REGION THROUGH ROLLOUT

Rollout is an attempt to allocate temporarily an extra region for a job step that needs more space than is available in its existing region or regions. The RO/RI module first tries to allocate the extra region from free space in the dynamic area. If, however, there is not enough contiguous free space, the RO/RI module writes the contents of another job step's region from main storage to auxiliary storage. The "borrowed" region is then allocated to the requestor's job step.

The RO/RI module consists of a central routine, called the Rollout/Rollin Criterion routine, and various subroutines. The RO/RI Criterion routine coordinates the rollout activities of the subroutines. These activities include deferring I/O requests for the job step to be rolled out, deferring its operator replies, setting its tasks nondispatchable, and causing the transfer of the contents of the selected region to the rollout data set.

The main functions performed during rollout are:

- Determining wnether rollout should be performed.
- Obtaining the needed space from unassigned storage.
- Finding a job step and region suitable to be rolled out.
- Processing if a suitable job step and region cannot be found.
- Processing if a suitable job step can be found. This processing includes allocating the selected region if its contents are already rolled out but the region is not in use. If the contents of the region are not already rolled out, the processing includes setting nondispatchable the tasks of the job step to be rolled out, deferring its I/O requests, and deferring its operator replies.
- Transferring the contents of the selected region to the rollout data set.
- Allocating the borrowed region to the requestor's job step.
- Processing if there was an unrecoverable I/O error during the rollout.
- Preparing for exit from the rollout/rollin module.

<u>Determining Whether Rollout Should Be Performed</u>

The RO/RI Criterion routine, when dispatched at entry point IEAQRORI, determines first whether rollout is being requested, then whether rollout should be performed. If rollout should not be performed, the RO/RI Criterion routine defers the current rollout request and branches to the Rollout/Rollin Exit subroutine to prepare for exit from the RO/RI module. If rollout should be performed, the RO/RI Criterion routine continues processing. In determining whether rollout should be performed, the routine does the following:

- Determines whether the current request is for rollout, rollin, or restart of deferred rollout requests. Routes control to the appropriate part of the RO/RI Criterion routine to service the request.
- Determines whether another job step has caused a rollout that is still in effect.
- Defers the current rollout request, if "multiple rollouts" are prohibited and if another job step has caused a rollout that is still in effect.
- Continues processing the current rollout request if no other job step has caused a rollout that is still in effect, or if another rollout is still in effect but a user-written appendage permits multiple rollouts.

<u>DETERMINING WHETHER THE CURRENT REQUEST IS FOR ROLLOUT</u>: The RO/RI Criterion routine determines the type of request by testing the parameter list address passed to the RO/RI module in register 1. If the address is positive, the request is for rollout. (The polarity of the parameter list address in the RO/RI IQE was set by the GETMAIN routine's SHEDRO or SCHEDRRI routine when it scheduled linkage to the RO/RI module. The parameter list address was placed in register 1 by the Stage 3 Exit Effector during the final phase of scheduling.)

<u>DETERMINING WHETHER ANOTHER JOB STEP HAS CAUSED A ROLLOUT THAT IS STILL IN EFFECT</u>: The RO/RI Criterion routine tests the "rollouts invoked" counter and, if necessary, examines the TCB queue to determine if a job step other than the requestor's has caused a rollout that is still in effect. These tests are made because concurrent rollouts for different requesting job steps are not allowed, unless permitted by the choice of a user-written Coincident Rollout appendage (IEAQAPG1). Such "multiple rollouts" are not normally permitted because concurrent requesting job steps could each attempt to roll out more than half of the main storage space available for rollout. In that case, the competing job steps would be placed on the deferred request queue, awaiting main storage space that would never be available. The system would thus be in an "interlock," unable to continue processing.

<u>DEFERRING THE CURRENT ROLLOUT REQUEST</u>: The RO/RI Criterion routine defers the current rollout request, if multiple rollouts are prohibited, and if another job step has caused a rollout that is still in effect. The routine defers the rollout request by transferring the requestor's IQE from the RO/RI IRB's queue of active IQEs to wait queue called the "rollout request queue." (The origin of the rollout request queue is defined in the secondary communications vector table as IEAROQUE.) The IQEs on the rollout queue are rescheduled for new linkage to the RO/RI module after either of two events has occurred: a region's contents have been rolled in, or the DEQ routine has marked a job step TCB as eligible to be rolled out (TCBNROC equals zero). Either event means that another region is avail-

able for possible rollout. (For further information on the restart of deferred rollout requests, see "Performing Final Common Processing.")

DETERMINING IF PROCESSING OF THE CURRENT ROLLOUT REQUEST SHOULD BE CONTINUED: The RO/RI Criterion routine continues processing the current rollout request if no other job step has caused a rollout that is still in effect, or if another competing rollout is still in effect but a user-written appendage permits such multiple rollouts. Without a user appendage, the RO/RI Criterion routine continues the processing of the current request only if no other job step has caused a rollout that is still in effect. A user-written appendage, if provided, can be substituted for the IBM-provided decision. Decisions made in the user appendage can provide flexible control of the number of job steps that can concurrently invoke rollout.

NOTE: If the user appendage allows more than one job step to invoke rollout concurrently, it is responsible for preventing interlocks.

Obtaining the Needed Space from Unassigned Storage

If the RO/RI Criterion routine decides that rollout should be performed, it tries to obtain a new region from unallocated space in the dynamic area via a conditional GETMAIN macro instruction that specifies subpool 246. The result is supervisor linkage to the GETMAIN routine. If there is insufficient space, the GETMAIN routine returns a code of '4', and the RO/RI Criterion routine then tries to find a job step and region suitable to be rolled out. If, however, the GETMAIN routine can allocate a new region, it builds a partition queue element (PQE) and a free block queue element (FBQE), and queues the PQE from the RO/RI TCB. The GETMAIN routine in this case supplies the RO/RI Criterion routine with a code of '0', indicating that the region has been allocated, and provides the address of the PQE representing the new region. (The PQE address is returned in a parameter list.)

When the RO/RI Criterion routine detects that a new region has been allocated, it does the following:

- Removes the newly created PQE from the RO/RI task's PQE queue and places it on the PQE queue of the requestor's job step TCB. The routine reorders the PQE queue, if necessary, so that the PQEs are queued according to ascending order of region addresses.

- Initializes the TCB address (PQETCB) in the new PQE to zero to indicate that the region was allocated from free space. This field will be tested during rollin to determine whether the region should be freed.

- Increases the "rollouts invoked" counter (IEAROICT) by a count of 'one', to indicate that a rollout has been invoked and is still in effect. This counter is tested each time that the RO/RI Criterion routine is entered for rollout, to determine whether rollout should be performed. (See "Determining Whether Rollout Should Be Performed.")

- Sets the "borrowed" flag (PQEBOR) in the rollout flags field of the new PQE. This flag, when set, indicates that the region described by the PQE is not "owned" by the job step to which it is allocated.

- Sets the "rollout invoked" flag (TCBFRI) in the requestor's job step TCB. This flag, when set, indicates that the job step has invoked one or more rollouts that are still in effect.

- Makes the requestor's task dispatchable by clearing the "core wait" nondispatchability flag (TCBWFC). This is done in preparation for the redispatching of the requestor's task.

- Branches to the RO/RI module's Retexit routine to prepare for exiting from the RO/RI module. (See "Preparation for Exit from the Rollout/Rollin Module.")

Obtaining a Job Step Suitable to Be Rolled Out

If a new region cannot be allocated from free space, the RO/RI Criterion routine tries to obtain a job step that is suitable to be rolled out. A job step is suitable if:

- It has not caused a rollout which is still in effect.

- Its TCB is marked eligible to be rolled out.

- It owns a region that is large enough to satisfy the current storage request and that is not already in use by a borrower.

The process of obtaining a job step suitable to be rolled out consists of two functional parts: finding a job step, and testing the selected job step to see that is meets the above requirements.

FINDING A JOB STEP: The RO/RI Criterion routine branches to the GETSTEP routine to find a job step whose suitability can be tested. The GETSTEP routine receives as input parameters the address of the requestor's job step TCB and the address of the rollout parameter list. The parameter list contains the size of the requested storage. The GETSTEP routine performs the following functions:

- Determines if the requestor's job step has previously caused a rollout that is still in effect. (The routine tests the TCBFRI flag in the requestor's job step TCB.) A requesting job step may invoke successive rollouts which are concurrently in effect.

- If so, invokes the TESTSTEP routine to test if one or more regions previously borrowed by the requestor's job step contain enough free space to satisfy the current request.

- Searches the TCB queue for a lower priority job step which may be tested for suitability, if the current request cannot be satisfied from a previously borrowed region. The TCB queue is searched in a downward priority direction, starting with the requestor's job step TCB and ending with the last TCB on the queue. The routine saves the address of the lowest priority job step TCB that it finds.

- Branches to the TESTSTEP routine to test the suitability of the selected job step. If the job step is not suitable, the GETSTEP routine repeats its search of the TCB queue. This time, however, the search ends with the previously selected TCB. The search is finished when a suitable job step has been found, or when all job steps lower in priority than the requestor's have been examined and none has proved suitable.

- Branches to an optional user-written appendage (IEAQAPG2), if it cannot find a job step which is suitable to be rolled out. The user (High Priority Pass) appendage, if present, dynamically determines whether the GETSTEP routine should make a new search of the TCB queue, this time examining job steps that are higher in priority than the requestor's.

- Searches the TCB queue for a higher priority job step which may be tested for suitability, if the High Priority Pass appendage so decides. The TCB queue is searched in a downward priority direction, starting with the master scheduler TCB and ending with the requestor's job step TCB. The search and examination of job steps is similar to the low priority search previously described.

- Returns control to either of two return points in the RO/RI Criterion routine, after completing its examination of job steps that were candidates for rollout. The particular return point depends on whether a job step suitable for rollout has been found. If the GETSTEP routine finds a suitable job step, it places in register 0 the address of the PQE belonging to the job step.

TESTING THE SELECTED JOB STEP: Each job step selected by the GETSTEP routine is further tested for suitability by the TESTSTEP routine. The TESTSTEP routine determines that a selected job step is suitable to be rolled out if:

- The job step has not invoked a rollout which is still in effect. (Although concurrent rollouts may be permitted by a user appendage (IEAQAPG1), nested rollouts are never permitted. A nested rollout is the rollout of a job step that has itself caused a rollout that is still in effect.)

- The job step is eligible to be rolled out. The step is eligible if the "nonrolloutable count" (TCBNROC) is zero in its TCB. A zero count means that the job step was initialized as eligible when it was attached and is not currently using or waiting to use a system resource that requires the ENQ macro instruction. The "nonrolloutable count" was initialized to either zero or one by the Attach routine when an initiator attached the job step. The initialization reflects the job step's eligibility to be rolled out, as specified by the ROLL operand of the JOB or EXEC statement when the job was placed in the input stream. The "nonrolloutable count," after initialization, is increased by one by the ENQ routine for each system resource for which an ENQ macro instruction is issued by the job step. The count is similarly decreased by the DEQ routine for each issuance of the DEQ macro instruction by the job step.

- The job step's region is large enough to satisfy the current storage request.

- The region is not being used by a job step that has invoked rollout. Such a borrower could be either the current requestor's job step, if it has previously invoked rollout, or another requesting job step if concurrent rollouts are permitted. If the region is not being used by a borrower, its "in use" flag (PQEUSE) in the PQERFLGS field is zero.

- The job step and its region are approved by a user-written appendage, if such an appendage has been provided. The Criterion Selection appendage (IEAQAPG4) can be provided by the installation to make further tests of a job step already approved by the TESTSTEP routine.

- Returns control to the caller (usually the GETSTEP routine), with the PQE address in register 0 if it has approved the job step and region.

Processing If a Job Step Suitable for Rollout Cannot Be Found

If the GETSTEP routine cannot find a job step suitable to be rolled out, the RO/RI Criterion routine can follow either of two possible courses of action. If can cause the abnormal termination of a job step, or it can defer the current rollout request by placing the requestor's IQE on a wait queue called the "rollout queue." The particular choice depends on the decision of a user-written ABEND appendage (IEAQAPG3), if the appendage is present. If the appendage is not present, the current rollout request is deferred.

CAUSING THE ABNORMAL TERMINATION OF A JOB STEP: The ABEND appendage, if present, can request the abnormal termination of either the requestor's job step or another job step in the system. The appendage provides the address of the selected job step TCB in a register. Termination of the requestor's job step removes it from the system if it cannot wait for storage to become available. Termination of another job step results in the freeing of a region. After such termination is complete, the RO/RI module is reentered twice: first to perform rollin, then to make a new attempt at rollout for the deferred request. (See "Scheduling Deferred Rollout Requests.")

If the requestor's job step task is to be terminated, the RO/RI Criterion routine branches to the ABTERM routine, providing the address of the requestor's job step TCB. The ABTERM routine schedules the abnormal termination of the job step, then returns control to the RO/RI Criterion routine. The RO/RI Criterion routine sets the requestor's task dispatchable (clears the TCBWFC flag), and branches to the RETEXIT routine. The RETEXIT routine prepares for exiting from the RO/RI module and eventual dispatching of a task of an another job step. (See "Exiting from the Rollout/Rollin Module.")

If a job step other then the requestor's is to be terminated, the RO/RI Criterion routine first determines that the TCB specified by the ABEND appendage is really a job step TCB. If the TCB is really a job step TCB, the routine branches to the ABTERM routine to schedule the abnormal termination of the specified job step. It then defers the current rollout request, by placing the requestor's IQE on the rollout queue. If, however, the TCB specified for abnormal termination is not really a job step TCB, the RO/RI Criterion routine defers the current rollout request without scheduling an abnormal termination.

DEFERRING THE CURRENT ROLLOUT REQUEST: The RO/RI Criterion routine defers the current rollout request if the ABEND appendage (IEAQAPG3) decides against a termination (or if there is no ABEND appendage). The rollout request is deferred until space is freed or until an ineligible job step is made eligible to be rolled out. (The method of deferring a rollout request is described in "Determining Whether Rollout Should Be Performed." The restart of deferred rollout requests is described in "Performing Final Common Processing.") After deferring the current rollout request, the RO/RI Criterion routine branches to the Rollout Exit routine to prepare for a task switch and for return of control to another task. (See "Exiting from the Rollout/Rollin Module.")

Processing If a Suitable Job Step Can Be Found

If the GETSTEP routine finds a suitable job step to be rolled out, it returns control to the main line of the RO/RI Criterion routine, providing the address of the selected PQE. This PQE describes the region that will be allocated to the requestor's job step. The region's contents can be in either of two conditions: already rolled out for a requestor but not in use, or not already rolled out.

If the contents of the selected region have already been rolled out, the RO/RI routine does not attempt a second rollout. In this case, the routine merely allocates the selected region to the requestor's job step.

If, however, the contents of the selected region have not already been rolled out, the RO/RI Criterion routine prepares to roll out the region's contents to the rollout data set. (See "Preparing to Roll Out the Contents of the Selected Region.")

If Main Storage Hierarchy Support is included in the system, and a task whose region is selected for rollout has another region in either hierarchy 0 or 1, this remaining region is not affected by rollout.

ALLOCATING THE SELECTED REGION: The selected region is allocated to the requestor's job step if two conditions are met: the region's contents have already been rolled out, and the region is not being used. The RO/RI Criterion routine tests only whether the region's contents have been rolled out. The TESTSTEP routine previously tested whether the region is in use.

If the conditions are met, the RO/RI Criterion routine allocates the selected region to the requestor's job step by performing the following functions:

- Sets the "rollout" flag (PQERO) and the "in use" flag (PQEUSE) in the owner's PQE to indicate that the contents of the region have been rolled out and that the region is being used by a borrowing job step.

- Branches to the BUILDPQE subroutine to obtain space for and initialize a new PQE to describe the borrowed region. The RO/RI Criterion routine will later place this PQE on the PQE queue of the requestor's job step. The new PQE is initialized to point to a free block queue element (FBQE) that describes as free the entire borrowed region. The last four words of the new PQE are copied from the corresponding fields of the owner's PQE. (These fields contain the owning job step's TCB address, the region size, the region address, and flags. See Section 12, "Control Blocks and Tables," for additional format information.) There are thus two PQEs describing the same region: the owner's PQE and the borrower's PQE, associated with different job step TCBs. The owner's PQE is flagged "owned," "rolled out," and "in use." The borrower's PQE is flagged "borrowed."

- Branches to the SETKEYS subroutine to set to zero the storage key of all 2K blocks in the region. This is done so that no user routine can store information in the region before the GETMAIN routine has been reentered to allocate the region's space to the current requester.

- Branches to location RR004 to: increase the "rollouts invoked" counter, set the "borrowed" flag[1] (PQEBOR) in the new PQE, place the new PQE on the PQE queue of the requestor's job step, set the "rollout invoked" flag (TCBFRI) in the requestor's job step TCB, and clear the "core wait" nondispatchability flag (TCBWFC) in the requestor's TCB. (See "Obtaining the Needed Space from Unassigned Storage" for a discussion of these actions.)

- Bṛanches to the RETEXIT routine to prepare for exit from the RO/RI module and return control to the requestor's task. (See "Exiting from the Rollout/ Rollin Module.")

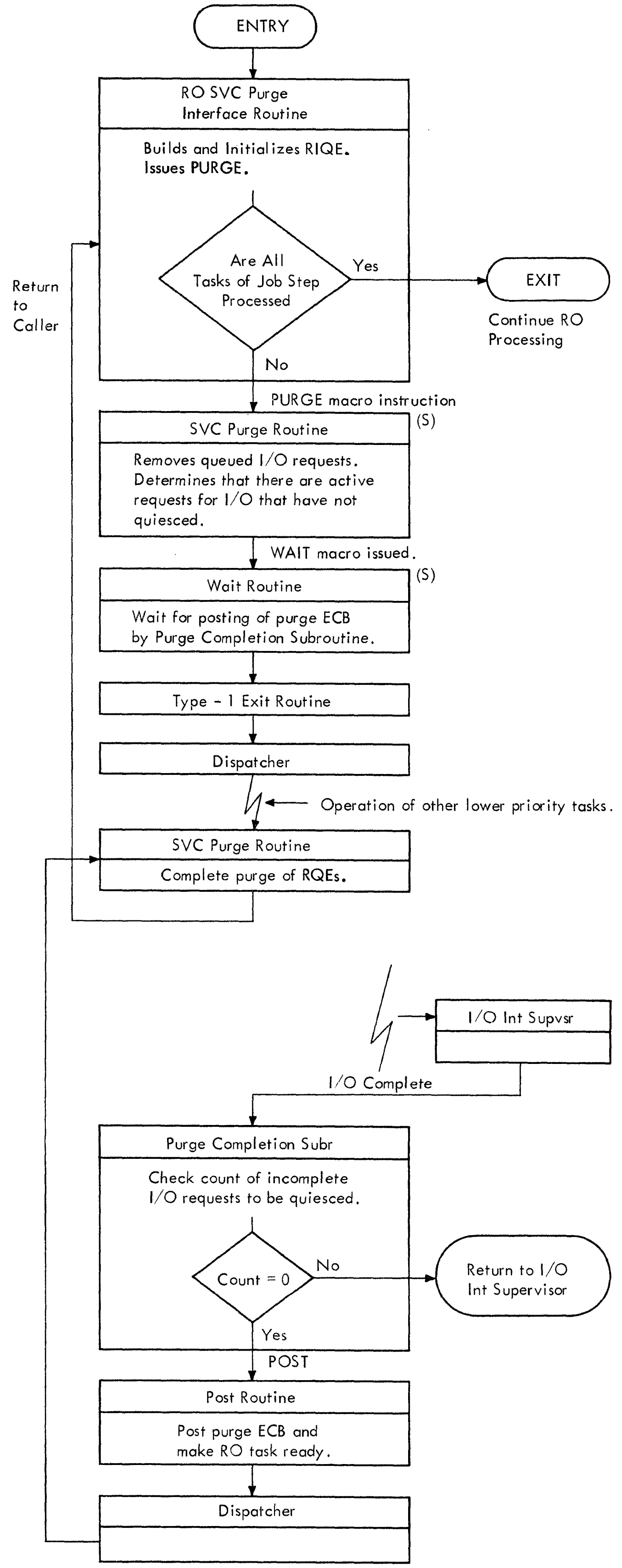
PREPARING TO ROLLOUT THE CONTENTS OF THE SELECTED REGION: The RO/RI Criterion routine prepares to roll out the contents of the selected region, if they have not already been rolled out. Preparation consists of the following functions, performed for the job step to be rolled out:

- Setting nondispatchable the tasks of the job step. This is done to prevent the restart of these tasks by the Dispatcher while the job step is not in main storage.

- Deferring the job step's I/O requests. I/O commands that are executed for the job step after it has been rolled out could cause information to be read into or written from main storage areas that no longer belong to the job step. To prevent this, queued I/O request elements, which represent channel programs not yet executed, are purged. Pointers to I/O blocks (IOBs) associated with these request elements however, are saved to permit restart. The purged request elements will be reinstated when the rolled out job step has been rolled in. Active I/O requests however, which represent channel programs being currently executed, are allowed to complete before the job step is rolled out. (Figure 5-10 illustrates the overall functional flow).

- Deferring the job step's operator replies. Replies received while the job step is rolled out must not be read into main storage areas that no longer belong to the job step for which they were issued. These replies are therefore saved in temporary buffers, and are later transferred to the appropriate user buffers when the rolled out step has been rolled in.

[1] The "borrowed" flag is set in the new PQE to indicate that the represented region is not owned by the job step to which it is allocated.


ENTRY
RO SVC Purge Interface Routine
Builds and Initializes RIQE. Issues PURGE.
Are All Tasks of Job Step Processed
Yes
EXIT
Continue RO Processing
No
Return to Caller
PURGE macro instruction
(S)
SVC Purge Routine
Removes queued I/O requests. Determines that there are active requests for I/O that have not quiesced.
WAIT macro issued.
(S)
Wait Routine
Wait for posting of purge ECB by Purge Completion Subroutine.
Type - 1 Exit Routine
Dispatcher
Operation of other lower priority tasks.
SVC Purge Routine
Complete purge of RQEs.
I/O Int Supvsr
I/O Complete
Purge Completion Subr
Check count of incomplete I/O requests to be quiesced.
Count = 0
No
Return to I/O Int Supervisor
Yes
POST
Post Routine
Post purge ECB and make RO task ready.
Dispatcher

• Figure 5-10. Interfaces Between Rollout Module and SVC Purge Routine

<u>Setting Nondispatchable the Tasks of the Job Step</u>: The RO/RI Criterion routine issues the STATUS macro instruction to cause supervisor linkage to the Set Status routine (IGC079). This routine sets nondispatchable all tasks of the specified job step by setting the TCBFRO flag in each TCB.

The operands of the STATUS macro instruction, as used above, have these meanings:

STATUS	SET ND,	(1)	(12)
	Causes setting of nondispatchability flag specified by mask operand (12).	Indicates that the TCB whose address is in register 1 and its descendants should be set as specified.	This mask number indicates that the "rolled out" nondispatchability flag (TCBFRO) should be set.

<u>Deferring the Job Step's I/O Requests</u>: The RO/RI Criterion routine branches to the SVC Purge Interface routine (PRGIO) to defer the job step's I/O requests. The SVC Purge Interface routine performs the following functions for each task of the job step:

- Obtains space for and initializes a rollout I/O queue element (RIQE)[1]. Each RIQE will serve as a list origin for a queue of I/O blocks (IOBs) that represent the task's deferred channel programs. The IOBs will be used to restart the channel programs after the job step has been rolled in.

- Stores in the SVC purge parameter list[1] the address of the TCB whose queued request elements will be purged. Also places in the purge parameter list a pointer to the IOB list origin in the RIQE. The I/O Supervisor's SVC Purge routine will use this parameter list during its purge of the task's request elements.

- Issues a PURGE macro instruction to gain supervisor linkage to the I/O Supervisor's SVC Purge routine (IGC016). Flags (hex. '02') in the purge parameter list specify the "purge by TCB" and "quiesce" options. The address of the purge parameter list is provided in register 1. (See the publication <u>I/O Supervisor PLM</u> for detailed information on the SVC Purge routine.)

[1] See Section 12, "Control Blocks and Tables."

The SVC Purge routine searches the system queues for I/O request elements belonging to the specified task. It removes from the logical channel queues and the seek queues the request elements that are not yet active. It returns these request elements to the free list in the IOS. It queues their associated IOBs from the list origin in the input RIQE, so that the IOBs would be available when I/O operations are resumed. (See Figure 5-11.)

The routine then waits for completion of active I/O requests. Such requests represent I/O operations in process. The routine waits by issuing a wait macro instruction specifying the purge ECB and a wait count equal to the number of I/O requests that must complete. (The address of the purge ECB is in the SVC purge parameter list.)

During the subsequent wait period, control is given to lower priority tasks in the system. When each active I/O request completes, the I/O Interruption Supervisor received control and branches to the Purge Completion subroutine. This subroutine, part of the SVC Purge routine, decreases and tests the count of I/O requests awaiting completion. (This count is kept at offset 8 in the SVC purge parameter list.) When the count reaches zero, the Purge Completion subroutine posts the purge ECB complete, and the Dispatcher returns control to the main line of the SVC Purge routine. The SVC Purge routine then completes the purge of queued request elements, and returns control to the RO/RI module's SVC Purge Interface routine.

- Returns control to the RO/RI Criterion routine to continue the preparation for rollout, after the SVC Purge routine has been invoked for all tasks of the job step.

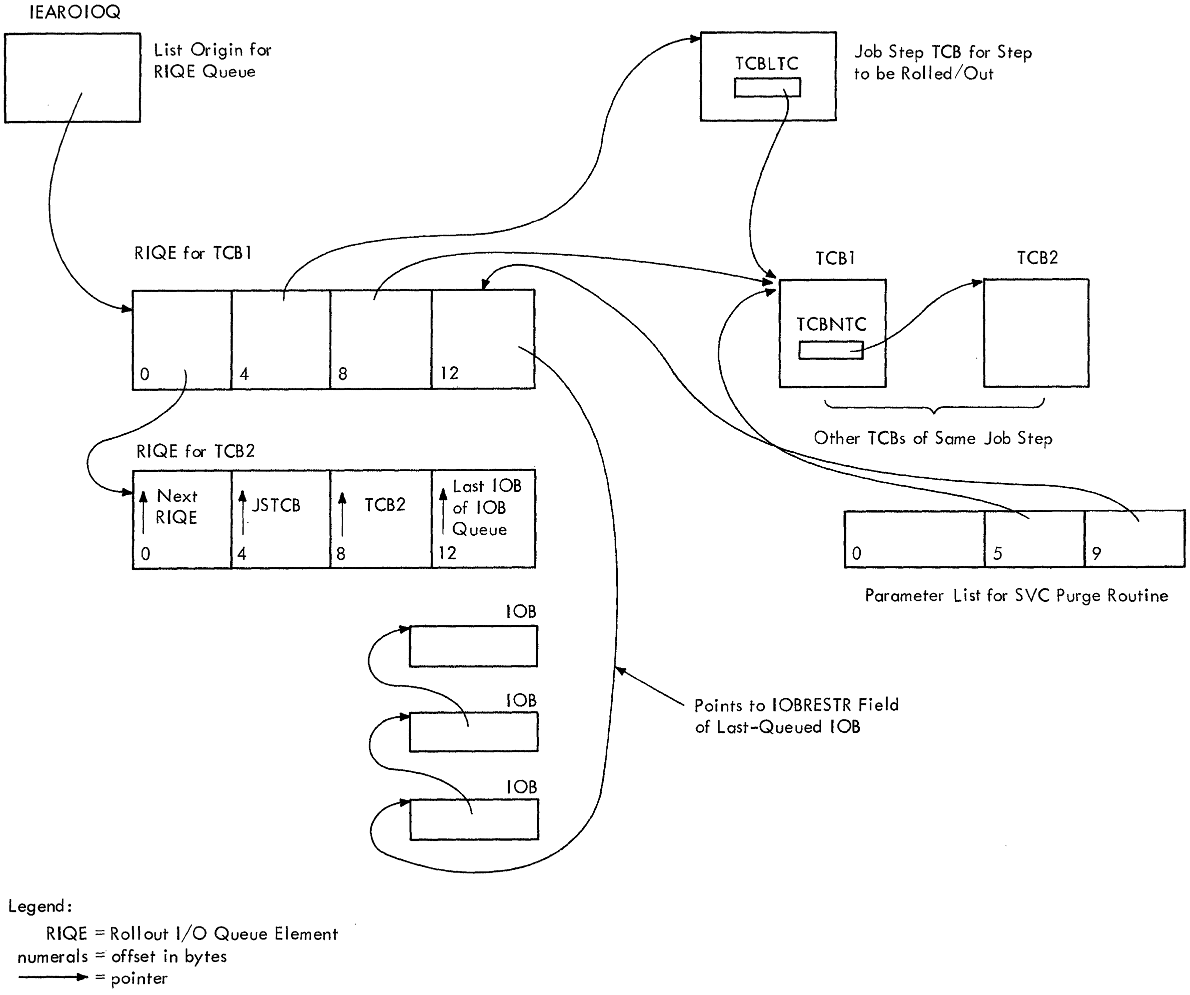

IEAROIOQ
List Origin for RIQE Queue
TCBLTC
Job Step TCB for Step to be Rolled/Out
RIQE for TCB1
0
4
8
12
TCB1
TCB2
TCBNTC
Other TCBs of Same Job Step
RIQE for TCB2
Next RIQE
0
JSTCB
4
TCB2
8
Last IOB of IOB Queue
12
0
5
9
Parameter List for SVC Purge Routine
IOB
IOB
IOB
Points to IOBRESTR Field of Last-Queued IOB
Legend:
RIQE = Rollout I/O Queue Element
numerals = offset in bytes
→ = pointer

Figure 5-11. How IOBs for Deferred I/O Requests are Queued

Deferring the Job Step's Operator Replies: The RO/RI Criterion routine branches to the Reply Purge routine (PRGRQE). This routine sets the "rollout" flag in reply queue elements belonging to the job step to be rolled out. The reply queue elements represent operator replies not yet received by the job step. If a reply is received while the job step is rolled out, the communications task Reply Processor routine (IGC1203D) will determine that the rollout flag is set in the reply queue element, and will save the reply in a temporary buffer until the job step is rolled in. (See "Reply Processing" in Section 7, "Console Communications and System Log.")

In order to flag outstanding replies, the Reply Purge routine:

- Finds each reply queue element belonging to the job step being rolled out. It recognizes the element by its TCB pointer (RQETCB) and the job step TCB pointer in the specified TCB. (See Section 12, "Control Blocks and Tables," for the format of a reply queue element.)

- Ignores reply queue elements belonging to other rolled out steps (meaningful only if concurrent rollouts are permitted). Also ignores reply queue elements flagged for purge. These latter elements were flagged by the WTOR Purge routine (IEECVPRG) because of a normal or abnormal task termination and will be purged by the Reply Processor routine (IGC1203D).

- Sets the rollout flag (RQERO) in the selected reply queue element as an indication for the Reply Processor routine.

- Returns control to the RO/RI Criterion routine when all reply queue elements on the queue have been examined, and elements belonging to the job step have been flagged.

Transferring the Contents of the Selected Region to the Rollout Data Set

When preparation for rollout is complete, the RO/RI Criterion routine branches to the Start Transfer routine (STARTIO), passing the address of the PQE for the selected region. This routine starts and controls the transfer of the selected region's contents to the rollout data set. It is also used during rollin to transfer the rolled out job step from the rollout data set to its region of main storage.

The Start Transfer routine does the following:

- Initializes the channel programs.

- Starts the channel programs.

- Reinitializes the channel programs.

- Handles a normal channel-end condition.

- Handles an end-of-cylinder condition.

- Responds to the type of completion, normal or abnormal.

INITIALIZING THE CHANNEL PROGRAMS: The Start Transfer routine first issues an SSM instruction. This instruction sets the system mask in the current PSW to permit I/O interruptions on all channels. This is necessary because the standard PSW under which the RO/RI module operates does not permit external and I/O interruptions. (The normally disabled mode of operation is typical of most supervisor routines.)

The Start Transfer routine next branches to the Channel Program Initialization subroutine (CPINIT). This subroutine initializes two channel programs and prepares for the starting of the I/O device by the I/O Supervisor. The subroutine's functions are as follows:

- Determines from the polarity of the input PQE address whether rollout or rollin is needed.

- Calculates and saves the address of the region's upper boundary, for use by the PCI appendage routine and the Channel End Appendage routine in determining when the last record has been transferred. (Both appendages are part of the Start Transfer routine.)

- Places the data address (the starting address of the region) in the Read/Write channel command word (CCW) of each channel program.

- Sets the command code to "Write" in the Read/Write channel command word (CCW) of each channel program. (If the Start Transfer routine had been entered for rollin, the command code would be set to "Read".)

- Stores in the IOBSTART field of the rollout input/output block (IOB) the address of the Search ID Equal command of the first channel program. (The I/O Supervisor will use this address in a Transfer in Channel (TIC) to the Search command to start the channel program.)

- Sets the NOP command code in the NOP/TIC command in both channel programs. (The PCI Appendage routine will later change one of these commands to a TIC.)

- Calculates the relative disk address (TTR) at which writing (or reading) will begin in the rollout data set. This address, when converted to an absolute address, will be used in the Seek command to be issued by the I/O Supervisor. The following formula is used to calculate the relative disk address:

  $TTR = ((R \ \ - K_1)/R \ )/N$

  where:

  R = the address of the region whose contents are to be rolled out (or rolled in).

  K_1 = the address of the last byte of the system queue area plus one. (This address was stored in the GOVRFLB table by the Nucleus Initialization Program (NIP).

  R = record size in bytes.

  N = number of records per track on the direct access device.

- Branches to a convert routine (IECPCNVT) whose address is in the communications vector table. This routine converts the relative disk address (TTR) to an absolute disk address (MBBCCHHR).
- Places the absolute disk address in the IOBSEEK field of the rollout IOB for use by the I/O Supervisor in its Seek command.

STARTING THE CHANNEL PROGRAMS: The Start Transfer routine starts execution of the channel programs by issuing an EXCP macro instruction which specifies the rollout IOB. The EXCP macro instruction causes supervisor linkage to the EXCP Supervisor, which starts the first channel program.

After the first channel program has been started, the I/O Supervisor returns control to the Start Transfer routine, via the I/O First-Level Interruption Handler and the Dispatcher. The Start Transfer routine then issues a WAIT macro instruction, specifying the rollout event control block (ECB). The macro instruction causes supervisor linkage to the Wait routine, which places the Start Transfer routine and the RO/RI IRB in a wait condition. They await the posting of the rollout ECB by the I/O Supervisor. The posting will indicate either that the channel programs have completed the data transfer or that an I/O error has occurred. Until the rollout ECB is posted, control is given to other lower priority tasks, via the Dispatcher.

REINITIALIZING THE CHANNEL PROGRAMS: When the channel fetches the Read/Write command, it detects that the program-controlled interruption (PCI) flag is set in the command. (The PCI flag is bit 36 in the 64-bit CCW.) The channel then interrupts the CPU, although continuing the execution of the channel command. The PCI Interruption causes supervisor linkage to the I/O Supervisor. The I/O Supervisor determines the cause of the interruption and branches to the RO/RI module's PCI Appendage routine (PCIAPG).

The PCI Appendage routine determines whether the last record is being transferred. If so, the routine returns control immediately to the I/O Supervisor to await the channel-end interruption, when the last CCW is fetched by the channel. If, however, the last record is not being transferred, the routine prepares for a TIC to the next channel program to continue the transfer. The PCI Appendage routine:

- Computes the data address for the Read/Write CCW of the next channel program. It does this by adding the record size (1024) to the address field of the CCW.
- If the sum is greater than the upper boundary of the region, the last record is being transferred. In this case, returns control to the I/O Supervisor. Control is then routed to a lower-priority ready task, via the I/O First-Level Interruption Handler and the Dispatcher.
- If the sum is not greater than the upper boundary of the region, stores the computed data address in the Read/Write CCW of the next channel program, and continues processing.
- Places a NOP command code (hex. '03') in the NOP/TIC CCW of the next channel program. This is necessary because the next record could be the last record. In that case, the channel's detection of no more CCW's would cause a needed channel-end interruption.
- Updates by 1024 bytes the address field of the Search ID Equal CCW in the next channel program. The search will identify the record to be transferred by the Read/Write command that follows the Search command.
- Places the TIC command code (hex. '08') in the NOP/TIC CCW of the current channel program. It does this to continue channel program execution, since the current record is not the last.
- Switches the contents of the "current" and "next" initialization pointers, so

that channel-program switching can continue.

- Returns control to the I/O Supervisor, which then gives control to a lower priority ready task, via the I/O First-Level Interruption Handler and the Dispatcher. Performance of the ready task continues, overlapping the data transfer, until it is interrupted by the next I/O interruption.

HANDLING A CHANNEL-END CONDITION: A channel-end interruption occurs after the channel executes a NOP/TIC command that has not been changed to a TIC by the PCI appendage. The interruption causes supervisor linkage to the I/O Supervisor, which determines the cause of the interruption, and branches to the RO/RI module's Channel End Appendage.

The Channel End Appendage determines if the last record has been transferred. If so, the appendage returns control to the I/O Supervisor. The I/O Supervisor then posts the rollout ECB, indicating in the completion code whether the transfer has completed normally or with error.

The POST macro instruction causes supervisor linkage to the Post routine (IGC002), which places the completion code in the ECB and readies the waiting RO/RI IRB. The Post routine also alters the "new" TCB pointer (IEATCBP), via the Task Switching routine, to indicate the need for a task switch. Then, the Post routine returns control to the RO/RI task's Start Transfer routine, via the Dispatcher.

If however, the last record has not been transferred, the Channel End Appendage resets flags and an error count in the rollout IOB, and returns control to the I/O Supervisor to restart the channel programs.

HANDLING AN END-OF-CYLINDER CONDITION: If an abnormal condition occurs at the direct-access device, the I/O Supervisor gains control via supervisor linkage, determines the cause, and- branches to the RO/RI module's Abnormal End Appendage routine. This routine (ABEAPG) determines if an end-of-cylinder condition exists. It does this by checking error indicators in the rollout IOB. If an end-of-cylinder condition does not exist, the routine returns control to the I/O Supervisor for further error handling. If, however, an end-of-cylinder condition does exist, the routine obtains the address of a previously executed CCW, stores the address in the IOBSTART field of the IOB, and returns control to the I/O Supervisor. The I/O Supervisor then restarts the channel programs, beginning with the specified CCW.

RESPONDING TO THE TYPE OF COMPLETION: The Start Transfer routine regains control from the Dispatcher when the I/O Supervisor has posted the rollout ECB. Control is returned to the instruction immediately following the WAIT macro instruction. The ECB is posted when any of the following conditions has occurred:

- The region's contents have been transferred without error.

- An error has occurred after a channel-end interruption. This type of error may be recoverable.

- An unrecoverable error has occurred.

The Start Transfer routine determines the type of completion by examining the completion code in the rollout ECB. (See Section 12, "Control Blocks and Tables," for the ECB completion codes.)

If the region's contents have been transferred without error, the routine returns control to the RO/RI Criterion routine. The RO/RI Criterion routine then allocates to the requestor's job step the region whose contents have been rolled out.

If an error has occurred after a channel-end interruption,[1] the Start Transfer routine branches to the Channel Program Initialization routine (CPINIT) to reinitialize the channel programs. The Start Transfer routine then reissues the EXCP macro instruction to restart the channel programs. It thus makes a new attempt to transfer the region's contents.

If an unrecoverable error has occurred, the Start Transfer routine issues an output message (IEA1001 jobname stepname). It then branches to do special processing that depends on whether rollout or rollin is being performed. If rollout is being performed, deferred I/O requests and deferred operator replies are restarted and the region is reallocated to its owning job step. A new attempt is then made to find a job step suitable to be rolled out. (See "Processing If I/O Error Occurred During Rollout.") If rollin is being performed, the job step that could not be rolled in is scheduled for abnormal termination, and queued rollout requests are restarted.

Allocating the Borrowed Region to the Requestor's Job Step

If the Start Transfer routine (STARTIO) determines that there was no permanent

[1] For this type of error the "IOB intercept" code appears in the completion code field of the ECB.

error during rollout, it returns control to the RO/RI Criterion routine. The RO/RI Criterion routine then does the following:

- Issues a message to the operator in the form "IEA1231, jobname, stepname R/O of jobname, stepname". The routine issues the message by means of a WTO macro instruction and resulting supervisor linkage to the Write-to-Operator routine (IGC0003E).

- Disables I/O interruptions to prevent delay in returning control to the requestor's task.

- Reallocates to the requestor's job step the region owned by the rolled-out job step. (The reallocation is done at symbolic location RR03.) The processing is similar to that previously described. (See "Processing If a Suitable Job Step Can Be Found.")

Processing If I/O Error Occurred During Rollout

If the Start Transfer routine (STARTIO) determines that a permanent I/O error occurred during the attempted rollout, it branches to the Rollout Retry routine (RETRY). The Rollout Retry routine restores to readiness the partially rolled out job step. It does this by:

- Restarting the job step's deferred I/O requests and operator replies, via the RSTRIO and RSTRQE routines. (See "Restarting Deferred I/O Requests for the Rolled-In Job Step" and "Restarting Deferred Operator Replies for the Rolled-In Job Step.")

- Sets the "nonrolloutable" count (TCBNROC) for the job step, so that a new attempt to roll out the step will not be made.

- Invokes the Set Status routine (IGC079) to reset the "rollout nondispatchability" flag (TCBRFO) in each TCB of the job step. The Set Status routine is invoked via the STATUS macro instruction.

- Branches to the TESTSTEP routine to resume the search for a job step suitable to be rolled out. The TESTSTEP routine gives control to the GETSTEP routine to search the TCB queue, as previously explained. (See "Obtaining a Job Step Suitable to Be Rolled Out.") If the GETSTEP routine can obtain a suitable step, the RO/RI Criterion routine rolls out the selected step. If, however, the GETSTEP routine cannot obtain a suitable step, the RO/RI Criterion routine either schedules a job step for abnormal termination or places the current request on the rollout request queue. (See "Processing If a Job Step Suitable for Rollout Cannot Be Found.")

Exiting From the Rollout/Rollin Module

The RETEXIT routine provides an exit from the RO/RI Module. It performs the following functions:

- Places on the "next available" list the interruption queue element (IQE) that represents the current RO/RI request. The IQE is queued from the RBNEXAV field of the RO/RI IRB. This is done after the RO/RI module has been executed for any of its major functions: rollout, rollin, or scheduling of deferred rollout requests (IQEs). This procedure is bypassed if rollout cannot be performed and the rollout request is deferred. (See "Deferring the Current Rollout Request" in "Processing If a Job Step Suitable for Rollout Cannot Be Found.")

- Ensures a task switch by placing zero in the "new" TCB pointer (IEATCBP). This indication will cause the Dispatcher to search the TCB queue for a ready task.

- Issues an SVC 3 instruction to invoke the supervisor Exit routine (IGC003). The Exit routine dequeues the RO/RI IRB from the RO/RI TCB. This action makes the RO/RI task nondispatchable. The supervisor Exit routine then gains linkage to the Dispatcher, via the Transient Area Refresh routine. The Dispatcher searches down the TCB queue, starting with the RO/RI TCB, and dispatches the current routine of the highest priority ready task.

ALLOCATING SPACE IN THE SYSTEM QUEUE AREA

The system queue area is restricted to control program routines. Only those routines that operate under a storage protection key of zero can use space in this area.

To obtain space in the system queue area, requestors must specify subpools 253, 254, 255 in the GETMAIN macro instruction.

- Space within subpool 253, unless explicitly freed, will automatically be released when the task for which it is being used is terminated.

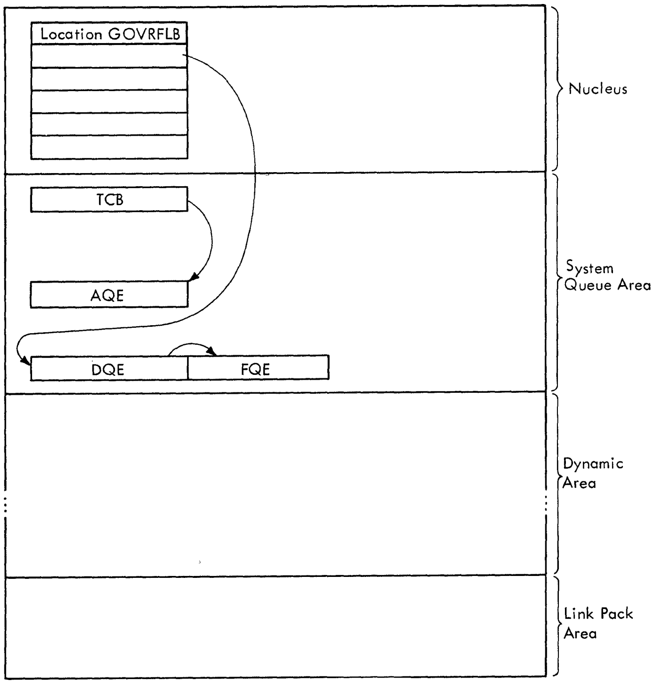
- Space within subpool 254, unless explicitly freed, will be released automatically when the job step for which it is being used is completed.

- Space within subpool 255 must be freed explicitly with a FREEMAIN macro instruction.

Before the system is generated, users of System/360 Operating System must specify the amount of space needed for a system queue area. During execution of the nucleus initialization program, a descriptor queue element (DQE) containing a record of the number of 2048-byte blocks assigned to the system queue area is built within the area (see Figure 5-12). Also built, adjacent to the DQE, is a free queue element (FQE) that contains the number of bytes of available space (initially, all space is available) in the system queue area. Location GOVRFLB in the nucleus contains a pointer to the descriptor queue element; the descriptor queue element contains a pointer to the free queue element.

Subpool 253 Allocation

When subpool 253 is specified in the GETMAIN macro instruction, 8 bytes are added to the size requested, and the location of the beginning of the available space is determined. In the first 8 bytes of the requested area, the GETMAIN routine builds an allocated queue element (AQE), into which it places the number of requested bytes, plus eight. It then chains the AQE to an AQE queue whose origin is in the TCB (TCBAQE field) of the task for which the space was requested. When that task is terminated, Supervisor termination routines will scan the AQE queue and give a FREEMAIN macro instruction to free all space associated with subpool 253.



- Figure 5-12. Element Relationships for System Queue Area Allocation

Subpool 254 Allocation

When subpool 254 is specified in a GETMAIN macro instruction, the GETMAIN routine builds an allocated queue element as it does for subpool 253, but chains the AQE to an AQE queue whose origin is in the TCB of the job step for which the space was requested. When that job step is completed, supervisor termination routines will scan the AQE queue and give a FREEMAIN macro instruction to free all space associated with subpool 254.

Subpool 255 Allocation

When subpool 255 is specified in a GETMAIN macro instruction, the GETMAIN routine passes the address of a free area to the requesting routine in general register 1 if an SVC 10 instruction caused entry, or in a prespecified location if an SVC 4 instruction caused entry. No allocated queue element (AQE) is built. An AQE is not required, as it is the responsibility of the requestor to ensure that the space is freed with a FREEMAIN macro instruction.

FREEMAIN ROUTINE

The FREEMAIN routine services the FREEMAIN macro instruction, which is used to free space when it is no longer needed. Space assigned to a region, space within a region, space assigned to one or more borrowed regions, or space in the system queue area may be freed. Basically, the FREEMAIN routine returns the allocated space to availability by adding queue elements representing the space to chains in which are recorded all free areas in main storage.

FREEING SPACE ASSIGNED TO A REGION

To free a region, the TCBPQE field of the TCB that represents the task for which the region is being used is checked to determine the address of the partition queue element of the appropriate region. The space (8 bytes) occupied by that partition queue element is then released (see the section "Freeing Space in Supervisor Queue Area"). Next, if the region to be freed is adjacent to an existing free area, it is combined with that area. This is done by adding the number of bytes in the region being freed to the size field of the free block queue element for the existing free area and, if necessary, relocating the FBQE to the beginning of the newly enlarged free area.

If a region being freed is not adjacent to a free area, the FREEMAIN routine builds an FBQE for the area and adds it to the chain of FBQEs that represents all space available for allocation as regions.

In a multiprocessing system, after a region has been freed, control is passed to the Vary Storage Offline routine (IFSVRYOF) which determines whether any of the freed region has been scheduled to be logically removed from the system because of a VARY STORAGE OFFLINE command. The Vary Storage Offline routine checks for Vary Queue Elements (VQEs) which are created when a VARY command is issued. If there are none, control is returned. Otherwise, the area of main storage specified by each VQE is compared with that specified by the freed PQE. For each VQE which applies to the freed area, the FBQE(s) and FSSEMAP are modified to indicate the area of main storage that has been made unavailable. (See Section 12, "Control Blocks and Tables" for a description of FSSEMAP). The VARY task ECB is POSTed in each applicable VQE to indicate that a partition within the range of that VQE has been processed.

If the region being freed is not owned by the requestor, it may be possible to roll in the job step that owns the region. In this case, the FREEMAIN routine branches to the FREBRF routine. This routine tests the region's attributes, and if possible, releases the region from the current job step and schedules linkage to the RO/RI module (IEAQRORI). (For a description of the FREBRF routine, see "Freeing Space Within a Region.")

FREEING SPACE WITHIN A REGION

To free space within a region, the CSPCHK subroutine is used by the FREEMAIN routine to locate the subpool queue element (SPQE) representing the subpool from which space is to be freed. The address of the SPQE queue is contained in the TCBMSS field of the task control block associated with the task to which the space is assigned. Then the descriptor queue element that represents the area in which the space is to be freed is located. Next, the two free queue elements between which the space exists are located and a new free queue element (FQE) to represent the newly freed space is constructed. This FQE is either added to the chain of FQEs or, if the space lies adjacent to another free area, is combined with the FQE of the adjacent free area.

A test is then made to determine if the resulting free area contains any free 2048-byte blocks of space that begin on a 2048-byte boundary. If it does, and the block is adjacent to an existing free 2048-byte block, the number of bytes to be freed are added to the count field in the FBQE representing the existing free space and, if necessary, the FBQE is relocated. If the block being freed is not adjacent to any existing free 2048-byte block, a new FBQE is constructed. The number-of-bytes count in the appropriate DQE is then decremented to reflect the number of blocks being removed from the subpool. When this count reaches zero, the DQE is eliminated.

The freeing of space in the region may permit a rollin to occur, if the region was obtained through rollout. If the rollout feature is included in the system, the FREEMAIN routine branches to the FREBRF routine. This routine tests the region's attributes, and if possible, releases the region from the current job step and schedules linkage to the RO/RI module (IEAQRORI).

The FREBRF routine performs the following functions for the job step whose space is being freed:

- Examines the partition queue element (PQE) that describes each region allocated to the job step.
- Determines if the region is "borrowed" and "free." The region is borrowed if its PQEBOR flag is set, indicating that the region is not owned by the job step. The region is "free" if none of its space is assigned to a subpool.
- If the region is borrowed and free, does the following:

  1. Releases the region from the current (borrowing) job step. It does this by removing the PQE from the current job step's PQE queue.
  2. Schedules linkage to the RO/RI module to attempt the rollin of the job step that owns the region. This is done via a branch to the SCHEDRRI routine.
  3. Determines if the region was allocated from unassigned space in the dynamic area. (Such a condition is indicated by zero in the PQETCB field.)
  4. Frees the region, if it was allocated from unassigned space, via a branch to the MRELEASE routine.

5. If the multiprocessing feature was selected, and the region was allocated from unassigned space, determines if any part of the region is to be removed from available main storage via a branch to the Vary Storage Offline routine (IFSVRYOF). (For a description of the Vary Storage Offline routine, see "Freeing Space Assigned to a Region.")

If rollin is warranted, the SCHEDRRI routine schedules linkage to the RO/RI module. The routine's processing is simular to that of the SHEDRO routine, which schedules the RO/RI module to perform rollout. (See "Scheduling Linkage to the Rollout/Rollin Module" in "Processing If the Requested Space Is Not Available.")

FREEING ONE OR MORE BORROWED REGIONS THROUGH ROLLIN

The RO/RI module is entered at location IEAQRORI from the Dispatcher, when it dispatches the RO/RI task via a Load PSW instruction. The RO/RI module determines that rollin is needed by observing that the input parameter-list address is negative. Accordingly, it branches to the RO/RI Criterion routine.

The RO/RI routine (ROLLIN) coordinates all functions performed during rollin. These functions consist of:

- Freeing the space occupied by the borrowing job step's PQE.
- Determining whether the rolled out job step should be rolled in.
- Transferring the rolled out job step to main storage, if the step should be rolled in. Performs special processing if I/O error occurred during the transfer. The processing consists of reconstructing free block queue elements (FBQEs) and scheduling the abnormal termination of the partially rolled-in step.
- Restarting deferred I/O requests for the rolled-in step.
- Restarting deferred operator replies for the rolled-in step.
- Making dispatchable the tasks of the rolled-in step.
- Performing final common housekeeping, primarily for the borrowing job step.
- Scheduling rollout for deferred rollout requests.

Freeing the Borrowing Job Step's PQE

The RO/RI Criterion routine first saves from the borrower's PQE the addresses of the region and the owner's job step TCB. It then invokes the FREEMAIN routine (IGC010), via supervisor linkage. The FREEMAIN routine frees the space occupied by the borrower's PQE, since this PQE is no longer needed. There is now only one PQE that describes the region, the owner's PQE.

Determining Whether the Rolled-Out Job Step Should Be Rolled In

To determine whether the rolled out step should be rolled in, the RO/RI Criterion routine does the following:

- Determines if the region was allocated to the borrower by means of a rollout, or whether the region was allocated from free space in the dynamic area. (If the region was allocated from free space, the owner's TCB address in the PQE (PQETCB) is zero.)
- Branches to location RIN08 to do final housekeeping, bypassing rollin, if the region was allocated from free space. (See "Performing Final Common Housekeeping.")
- Determines if any of the owner's PQEs represent the region that is being freed.[1] If so, clears the "in use" flag (PQEUSE) in the PQE to indicate that the region is not being used by a borrower.
- Bypasses rollin if the owning job step has any borrowed region that is still in use. In this case, the RO/RI Criterion routine branches to location RIN08 to perform final housekeeping. If, however, the owning step has no borrowed region that is still in use, the routine begins the rollin of the step.

Transferring the Rolled-Out Job Step to Main Storage

The RO/RI Criterion routine transfers to main storage the contents of the job step's region(s). It also does some housekeeping. For each region whose contents are to be rolled in, the routine does the following:

- Changes the storage protection key of all 2-K blocks from the borrower's key to that of the owner. This is done via a branch to location SETKEYS1.

[1] A rolled out step normally has only one PQE and region, since rollout of a step that has itself caused rollout is forbidden.

- Branches to the Start Transfer routine (STARTIO) to enable I/O interruptions and transfer the region's contents to main storage. (See "Transferring the Contents of the Selected Region to the Rollout Data Set.")

- Writes the rollin message "IEA124I jobname, stepname ROLLIN," if permanent I/O error did not occur during the transfer. The message is written via the Write-to-Operator routine (IGC003E).

- Disables I/O interruptions and tests for I/O error during the transfer. If there was permanent I/O error, the job step cannot be returned to its region. The RI/RO Criterion routine, in this case, reconstructs the free block queue elements (FBQEs) of the region, so that invalid FBQEs will not cause an ABEND recursion when the job step is abnormally terminated. The reconstruction is accomplished through the use of the MRELEASE routine in module IEAQM00.

- Resets the "rollout" flag (PQERO) in the owner's PQE to indicate that the region's contents are not rolled out.

- Sets the free 2-K blocks of the region to zero protection key. This is done so that the blocks may not be used by the job step until they have been allocated by the GETMAIN routine. (Sets zero protection key by branching to location SETKEYS.)

- If there was permanent I/O error during the transfer, branches to location ERRIN to invoke the supervisor's ABTERM routine. This routine schedules the abnormal termination of the partially rolled-in job step. As part of the abnormal termination, the ABEND routine (ABEND4) frees the region's space, via the Release Main Storage routine (IEAQSPET). The ABTERM routine returns control to location RIN06 in the RO/RI module. (See "Making Dispatchable the Tasks of the Rolled-In Job Step.")

Restarting Deferred I/O Requests for the Rolled-In Job Step

The RO/RI Criterion routine uses its SVC Restore Interface routine (RSTRIO) to restart I/O requests belonging to the rolled-in job step. These I/O requests were deferred when the job step was rolled out. (See "Processing If a Suitable Job Step Can Be Found.")

For each task of the rolled-in step, the SVC Restore Interface routine does the following:

- Selects the task's TCB address from a rollout I/O queue element (RIQE) on the RIQE queue. (See Section 12, "Control Blocks and Tables," for the RIQE format.)

- Prepares for the redispatching of the SVC Restore Interface routine under control of the selected TCB. The I/O Supervisor associates the restored I/O requests with the TCB under which the RESTORE macro instruction is issued. The preparation consists of:

1. Dequeuing the RO/RI IRB from the RO/RI TCB (or from a previously selected TCB), and placing it at the head of the RB queue of the selected task. The shifting of the IRB makes the RO/RI task nondispatchable.

2. Sets the "prevent asynchronous exits" flag (TCBFX) in the selected TCB. The purpose is to prevent the scheduling of an asynchronous exit routine by the Stage 3 Exit Effector when the Dispatcher is entered. Such scheduling would interfere with the issuance of the RESTORE macro instruction.

3. Places in the IRB old PSW the reentry address (RSTRI04) of the SVC Restore Interface routine. This address is the point at which the routine will be redispatched to issue the macro instruction.

4. Makes the selected task temporarily dispatchable by clearing nondispatchability flags in its TCB. (The TCBFRO flag was set in each TCB of the job step during rollout processing.) The flags are saved so that they may later be restored.

5. Saves the register contents that were stored in the selected TCB. Stores the RO/RI module's register contents in the selected TCB. This is necessary because the Dispatcher always loads registers from the TCB whose task it will dispatch.

6. Indicates to the Dispatcher that it should dispatch the selected task. The routine does this by zeroing the "new" TCB pointer (IEATCBP) and invoking the supervisor's Task Switching routine. The Task Switching routine, detecting that the RO/RI task is nonready, places the selected TCB address in the "new" TCB pointer.

- Branches to the Dispatcher to redispatch the SVC Restore Interface routine at location RSTRI04.

- Issues the RESTORE macro instruction, specifying the list origin (RIQEIOB) of a chain of IOBs. The IOBs represent channel programs deferred during rollout. The RESTORE macro instruction causes supervisor linkage to the I/O Supervisor, which sets I/O request elements to schedule the channel programs.

- Dequeues the RIQE for the selected task and frees its storage space (via the FREEMAIN routine), since the RIQE is no longer needed.

- Restores the selected task to its previous status by restoring its saved "prevent asynchronous exits" flag and its nondispatchability flags. Places the task's saved general register contents in the selected TCB.

When it has issued the RESTORE macro instruction for all tasks of the job step, the SVC Restore Interface routine causes the redispatching of the RO/RI task, as follows:

- Dequeues the RO/RI IRB from the last-selected TCB and queues it from the RO/RI TCB. This action makes the RO/RI task ready.

- Places the return address (RSTRI06) of the SVC Restore Interface routine in the IRB old PSW, in preparation for the redispatching of the RO/RI task.

- Places the RO/RI module's saved register contents in the RO/RI TCB. The Dispatcher will load the registers from this TCB.

- Branches to the Task Switching routine with the address of the RO/RI TCB. It does this to indicate to the Dispatcher that it should next dispatch the RO/RI task instead of the last-selected task.

- Branches to the Dispatcher to redispatch the RO/RI task. When redispatched (at location RSTRI06), the SVC Restore Interface routine returns control to the RO/RI Criterion routine.

Restarting Deferred Operator Replies for the Rolled-In Job Step

The RO/RI Criterion routine branches to the Reply Restore routine (RSTRQE) to restart operator replies that were received while the step was rolled out. The Reply Restore routine examines each reply queue element on the reply queue. Each element represents an operator reply that either was received or will be received. (The origin of the reply queue is UCMRPYQ in the unit control module.) The Reply Restore routine performs as follows for each reply queue element that is flagged "rolled out" and which belongs to the rolled-in step:

- Clears the "rolled out" flag (RQERO) in the reply queue element. (This flag was set by the Reply Purge routine (PRGRQE) when the step was rolled out.) The cleared flag will indicate to the communications task Reply Processor routine (IGC1203D) that it may move the reply to the user's buffer.

- Tests the "temporary-buffer" pointer (RQEXB) in the reply queue element. (In the program listing, it is called the "purging message address.") If the pointer is nonzero, the Reply Processor routine received a reply and placed it in the temporary buffer, while the job step was rolled out.

- Issues an MGCR macro instruction to restart the reply, if it was received while the job step was rolled out. The macro instruction causes supervisor linkage to the Reply Processor routine (IGC1203D), via the Command Processing routine and the MGCR Router routine. The Reply Processor routine determines that the temporary buffer is full (RQEXB is nonzero), moves the reply to the user's buffer, frees the temporary buffer, and completes the processing of the reply. (See "Reply Processing" in Section 7, "Console Communications and System Log.")

- Continues the examination of the reply queue until all reply queue elements that belong to the rolled-in step have been processed. The routine begins each new scan at the reply queue origin, because the Reply Processor routine reorders the queue each time that it is entered.

- Returns control to location RIN06 in the RO/RI Criterion routine.

Making Dispatchable the Tasks of the Rolled-In Job Step

The RO/RI Criterion routine next makes dispatchable the tasks of the rolled-in job step if all the regions, in both hierarchy 0 and 1, belonging to the task are in storage. (These tasks were set nondispatchable during rollout by the RO/RI Criterion routine, just before it deferred the job step's I/O requests.)

The RO/RI Criterion routine (at location RIN06) issues the STATUS macro instruction to cause supervisor linkage to the Set

Status routine (IGC079). This routine clears the "rolled out" nondispatchability flag (TCBFRO) in each TCB of the step.

The operands of the STATUS macro instruction, as used above, have these meanings:

RESET, ND	(6)	(12)
Causes the clearing of the nondispatchability flag specified by the mask operand (12).	Indicates that the TCB whose address is in register 6 (JSTREG) and its descendants are to be reset as specified.	This mask number indicates that the "rolled out" nondispatchability flag (TCBFRO) should be cleared.

Performing Final Common Housekeeping

The RO/RI Criterion routine next performs final common housekeeping, primarily for the borrower's job step. The housekeeping, begun at location RIN08, is common to three types of rollin:

1. A rollin of a job step that is completed without I/O error.
2. An attempted rollin of a job step that has produced a permanent I/O error.
3. A rollin to a region that was allocated from free space in the dynamic area.

The common housekeeping consists of:

- Decreasing by a count of one the "rollouts invoked" counter (ROICTR) to indicate that the current rollout is no longer in effect. The RO/RI Criterion routine tests the count each time a rollout is requested. Unless permitted by a user appendage (IEAQAPG1) the RO/RI Criterion routine will prevent another rollout (defer the request) if the count equals one.
- Clearing the "rollout invoked" flag (TCBFRI) in the borrower's job step TCB, if the job step has no other borrowed regions. The flag is tested by the TESTSTEP routine during rollout processing, to determine if a selected job step has invoked rollout. A job step is not suitable to be rolled out if it has itself invoked rollout.

The RO/RI Criterion routine then branches to its Dequeue routine to schedule rollout for deferred rollout requests.

Scheduling Deferred Rollout Requests

A rollout request (IQE) was deferred during rollout and placed on the rollout request queue for either of two reasons: another rollout was in effect and concurrent rollouts were prohibited, or a job step suitable to be rolled out could not be found.

Deferred rollout requests are scheduled by the RO/RI module's Dequeue routine (DEQUEUE). This routine is entered either from the RO/RI Criterion routine, via a branch, or from the Set Status routine (IGC079), during scheduling of the RO/RI module. In either case, a new region is available for rollout.

The Dequeue routine schedules deferred rollout requests as follows:

- Stores zero in the pointer to the rollout request queue (IEAROQUE). It does this because the queue is being temporarily eliminated.
- Places zero in the count of deferred rollout requests (IEAROQCT). During rollout this count can be used by an optional user appendage (IEAQAPG3) to determine whether to abnormally terminate a job step, if a step suitable for rollout cannot be found.
- If there are no IQEs on the rollout request queue, branches to the RETEXIT routine to queue the current IQE to the "next available list" (RBNEXAV) and exit from the RO/RI module. (See "Exiting from the Rollout/Rollin Module.")
- If there is at least one IQE on the rollout request queue, does the following for each IQE:

  1. Complements the IQE address to serve as an input parameter for the Stage 2 Exit Effector. This indicates to Stage 2 that the element is an IQE, not an I/O request element. (Stage 2 handles both types of elements.)
  2. Clears the "wait for core" nondispatchability flag (TCBWFC) in the requestor's TCB. (This TCB is the one whose address is contained in the IQE.) The flag is cleared because the task's main storage request is being reactivated.
  3. Branches to the Stage 2 Exit Effector (IEA0EF00) with the complemented IQE address. Stage 2 schedules linkage to the RO/RI module for the request. It does this by

placing the IQE on one of the asynchronous exit queues (AEQJ). (See "Scheduling Linkage to the Rollout/Rollin Module" in "Processing If the Requested Space Is Not Available.")

- Branches to the RETEXIT routine, when all IQEs on the rollout request queue have been scheduled. (See "Exiting from the Rollout/Rollin Module.")

FREEING SPACE IN THE SYSTEM QUEUE AREA

To free space in the system queue area, the GETMAIN routine first locates the descriptor queue element that represents the system queue area. It next checks to determine whether space within subpools 253 and 254 is to be freed. If so, 8 bytes are added to the size of the area to be freed (to include the AQE that is contained in the area), and 8 bytes are subtracted from the address of the space.

For subpools 253 and 254, the address of the appropriate AQE is obtained from the TCBAQE field of the associated TCB. If the entire area defined by the AQE is to be freed, the AQE is simply removed from the AQE queue. Otherwise, it is altered by changing its byte count. For all requests (subpools, 253, 254, and 255), any resulting contiguous free areas are combined by combining FQEs.

The Timer Supervision routines extend the capabilities of the IBM System/360 interval timer feature. By using the timer, these routines service the macro instructions by which programmers can obtain the date and time of day, measure periods of time, or schedule activity for a specific time of day.

The need for timer services is always signaled by an interruption, after which control is automatically given to an appropriate timer supervision routine. SVC interruptions occur when a TIME, STIMER, or TTIMER macro instruction is executed, and timer interruptions occur when a value in the interval timer expires. After an SVC interruption, one of three macro service routines is given control.

The TIME routine supplies 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 routines change the date at midnight and keep track of 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 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 queue element and places into it a summary of the information in the STIMER macro instruction, including the information that will be needed when the interval expires. It then positions the element in the timer queue by its time of expiration. When a timer interruption occurs (a value in the timer expires), timer interruption handling routines perform any requested actions and obtain new intervals to be placed into the timer from elements in the timer queue.

The TTIMER routine supplies the time remaining in a previously requested interval or it cancels previous requests for remaining time. 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.

In a multiprocessing system, each CPU has an interval timer located in its prefixed storage area (PSA). One timer is designated as active and is used by timing routines; the second, alternate timer is always set to a value X'80000000' greater than the active timer and thus never expires. In a partitioned multiprocessing system, the alternate timer is set, but does not decrement. Timer routines access the interval timer by adding the PSA displacement value of the timer to the value in PREFTMRA, an index to the PSA that contains the active timer. PREFTMRA is a PSA word which contains zeroes if the active timer is in the same PSA, or the address of the other PSA if the timer is located in the other PSA.

TIMER SVC INTERRUPTION HANDLING

The handling of interruptions resulting from issuance of timer-related macro instructions, is shown in Figure 6-1.

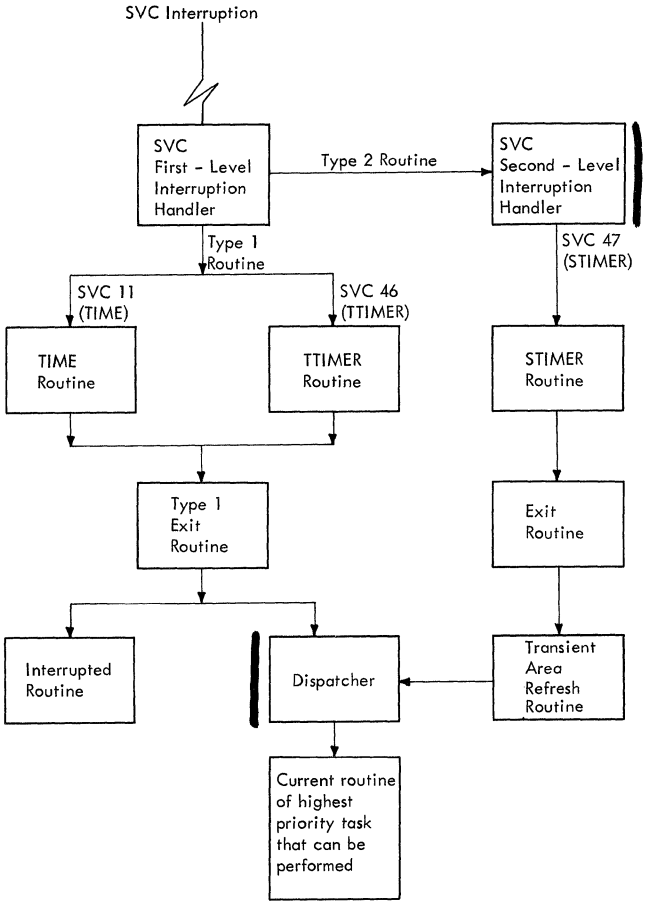


Figure 6-1. Timer SVC Interruption Handling

The expansions of the TIME, STIMER, and TTIMER macro instructions contain SVC 11, SVC 47, and SVC 46 instructions, respectively. When these SVC instructions are executed, SVC interruptions occur and control is given to the SVC First-Level Interruption Handler, which saves information about the interrupted program and routes control accordingly.

Both the TIME and TTIMER routines are type-1 SVC routines, and control is given directly to them by the SVC First-Level Interruption Handler. The STIMER routine, however, is a type-2 SVC routine, and control is first given to the SVC Second-Level Interruption Handler, which creates a supervisor request block, places into it the information about the interrupted program, and then gives control to the STIMER routine. (For a complete description of SVC first-level and second-level interruption handling, see "SVC Interruption Handling" in Section 2.

After type-1 SVC routines have been executed, control is given to the Type-1 Exit routine, which determines whether the task for which the SVC instruction was given should be resumed. If so, the Type 1 Exit routine restores the saved contents of registers and returns control to the routine in which the SVC instruction was encountered. If another task is to be performed, the Type-1 Exit routine saves register contents in the appropriate TCB, saves the contents of the old PSW in the appropriate request block, and gives control to the Dispatcher.

After type-2 SVC routines have been executed, control is given to the Exit routine, which performs functions similar to the Type-1 Exit routine and also gives control to the Dispatcher.

The Dispatcher routes control to the highest priority task that can be performed.

TIME ROUTINE

The TIME routine determines the current date and time of day and returns both values to requesting routines in general registers. It obtains the date from a location in 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.

After the operator gives a starting time, the interval timer must be kept continually operating, so that an elapsed time can be measured. The interval timer automatically decrements any value placed into it and causes an interruption when the value becomes negative. For timekeeping purposes, 6-hour intervals are used. 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, two pseudo clocks are used with the interval timer. Each time a 6-hour value is placed into the timer, one is also placed into a 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 a 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 given by the operator to arrive at the current time of day.

The values used in the timer are timer units equalling 13 microseconds; the values used in the pseudo clocks are timer units equalling 26.04 microseconds. Timer values are converted to units of 26.04 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.

STIMER ROUTINE

The STIMER routine builds and positions on the timer queue the elements that represent time intervals requested with STIMER macro instructions. If necessary, this routine first converts requested time from hours, minutes, and seconds to timer units. Then, using either an existing element or creating a new one, it places into the element information specified in the STIMER macro instruction. Finally, it uses the Timer Enqueue subroutine to position the elements on the timer queue.

THE TIMER QUEUE

The timer queue provides a means of scheduling values representing time intervals for placement into the interval timer to cause interruptions to occur at appropriate times.

All elements in the timer queue are arranged by a time of expiration. 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 is placed into the interval timer, and the procedure begins again.

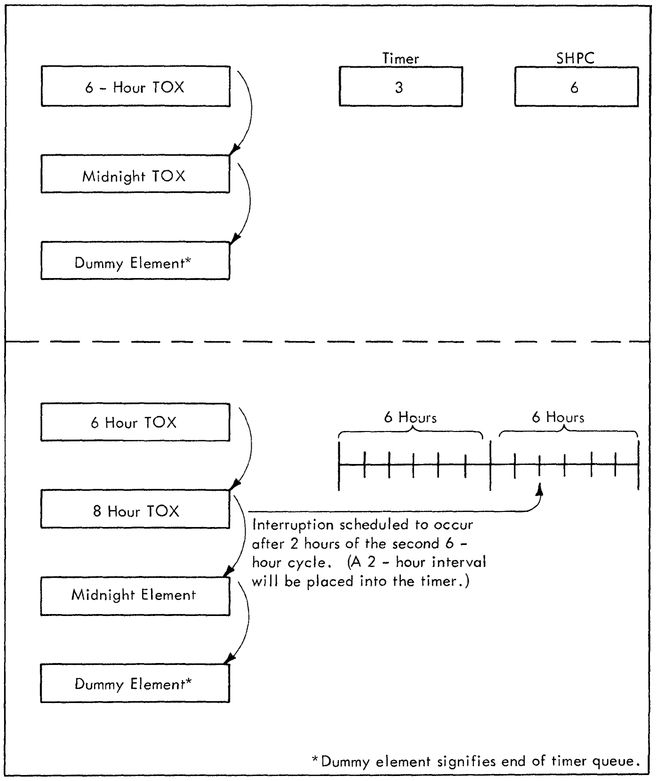
The time of expiration, by which elements are ordered on the timer queue, is based upon a 6-hour cycle. To determine a time of expiration (TOX), the STIMER routine subtracts the value in the interval timer from the value in the 6-hour pseudo clock (SHPC) and adds the interval requested:

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 6-2 shows the timer queue and the values in both the timer and the 6-hour pseudo clock at this time. Assume now that an 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.



Timer
SHPC
6 - Hour TOX
3
6
Midnight TOX
Dummy Element*
6 Hour TOX
6 Hours
6 Hours
8 Hour TOX
Interruption scheduled to occur after 2 hours of the second 6 - hour cycle. (A 2 - hour interval will be placed into the timer.)
Midnight Element
Dummy Element*
*Dummy element signifies end of timer queue.

Figure 6-2. Positioning of Elements on the Timer Queue

CONVERTING TIMES OF EXPIRATION TO INTERVALS

Using the STIMER macro instruction, programmers can specify a time interval in three ways: the location of a doubleword containing the interval in decimal form; a singleword containing the interval in binary form; or a doubleword containing a desired time of expiration in binary form. In the latter case, the STIMER routine must first convert the desired time of expiration to a time interval. In all cases, it must convert the intervals to timer units (one timer unit = 26.04 microseconds) before using them to calculate the times of expiration by which elements are ordered on the timer queue.

If a time of expiration is specified, the STIMER routine converts it to an interval by subtracting the current time of day from the specified time of expiration. It determines the current time of day with the following formula:

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.

BUILDING TIMER QUEUE ELEMENTS

The STIMER routine builds queue elements using information provided by the programmer in the STIMER macro instruction. It first checks to determine if an existing element can be used. A usable element may be available if an STIMER macro instruction has been given for the same task in which the current STIMER macro instruction was encountered. 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, or if no usable element exists, the STIMER routine obtains space for and builds a new element. It places in the current TCB a pointer (TCBTME) to the timer queue element that it has created. The STIMER routine then uses the Timer Enqueue subroutine to position the completed element on the timer queue (see Figure 6-3).

TIMER INTERRUPTION HANDLING

When a time interval that was placed into the timer expires, an external interruption occurs and control is automatically given to the External First-Level Interruption Handler (see Figure 6-4).

Basically, the 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

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 to determine what action to take, examines bits 6 and 7 of the first word in the element (see Table 6-1).

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 an exit is specified and the TQE is REAL, the TSLIH determines if the issuer of the STIMER was an initiator. If an initiator did not issue the STIMER macro instruction, the TSLIH proceeds as if an exit was specified and the TQE was TASK type. If an initiator did issue the STIMER macro instruction, further processing must be performed.

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 (TCBLTC field of initiator 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.

0	TQEFLGS (Indicators)	1 TQETCB (Address of TCB)
4	Zeros 1	TQEFLNK (Address of next queue element)
8	Zeros 1	TQEBLNK (Address of preceding queue element)
12	TQEVAL (Time of expiration/time remaining)	
16	TQELHPSW (First word of current PSW- used when TQE serves as IRB)	
20	Reserved	
24	TQESAADR (Address of processing program save area)	
28	Zeros	29 TQEEXIT (Address of timer asynchronous exit routine)
32	TQEGRS (Register save area - used when TQE serves as IRB)	
96	TQEECB (Used for interruption queue element when TQE serves as IRB) (16 bytes) TQEIQE (Used for ECB when WAIT parameter is given in STIMER macro-instruction.	

Note: See Section 12 for Description of Flags.

Figure 6-3. Timer Queue Element (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 into both the interval timer and the 6-hour pseudo clock. (The element representing the recently expired interval has been removed from the queue.)

RETURNING 6-HOUR AND MIDNIGHT ELEMENTS TO THE QUEUE

When intervals represented by either the 6-hour or midnight supervisor queue 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

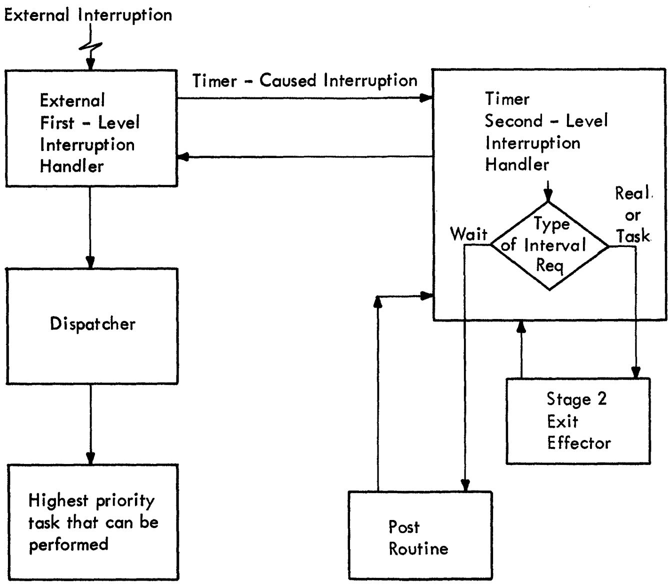

External Interruption
External First - Level Interruption Handler
Timer - Caused Interruption
Timer Second - Level Interruption Handler
Type of Interval Req
Wait
Real or Task
Dispatcher
Stage 2 Exit Effector
Highest priority task that can be performed
Post Routine

Figure 6-4. Timer Interruption Handling

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 Enqueue 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.

TTIMER ROUTINE

The TTIMER routine performs the two functions that can be requested with the TTIMER macro instruction. These are to provide the time remaining in a previously requested time interval or to cancel a previously requested interval.

Table 6-1. Actions Taken After Timer Expiration

Bits 6 & 7 of TQE	Indicate That:	Elapsed Time Represents:	Action Taken by Timer Second-Level Interruption Handler
00	TASK parameter was used in STIMER macro instruction.	Time used to perform task for which the STIMER macro instruction was given.	Checks bit 5, which contains a 1 if an asynchronous exit routine is to be entered. If so, passes control to Stage 2 Exit Effector.
01	WAIT parameter was used in STIMER macro instruction.	Total elapsed time, measured from time that interval was placed into timer.	Gives POST macro instruction. (Performance of task for which macro instruction was issued cannot be resumed until POST is given.)
10	Interval that expired was a 6-hour supervisor interval.	Total elapsed time, measured from time that interval was placed into timer.	Checks bit 5, which will contain a 1 if a 24-hour period has passed. If so, increments date by one.
11	REAL parameter was used in STIMER macro instruction.	Total elapsed time, measured from time that interval was placed into timer.	Checks bit 5, which will contain a 1 if an asynchronous exit routine is to be entered. If so, gives control to Stage 2 Exit Effector.*

* If an Initiator issued the STIMER macro instruction, the TQE will be converted to TASK type, and control will be passed to ABTERM.

DETERMINING REMAINING TIME

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 places 0 time into general register 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.

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.

SUPPORTING CONSOLE COMMUNICATIONS

The supervisor console support routines provide for input and output for one or more console devices. Input results from an unplanned interruption from an external device or from the main console; output results from the macro instructions WTO (Write to Operator) and WTOR (Write to Operator with Reply).

The operator causes an I/O interruption by pressing the REQUEST key on the 1052 Printer-Keyboard, or the START key on a card reader. The I/O First-Level Interruption Handler passes control to the I/O Supervisor, which determines that an operator interruption service has been requested. Control then passes to the resident Attention routine.

When the operator presses the INTERRUPT key on the operator control panel (OCP), he causes an external interruption. In this case, control passes from the External First-Level Interruption Handler to the communications task resident External Interruption Handler routine (IEEBC1PE).

The basic function of both the Attention routine and the External Interruption Handler routine is to prepare for the performance of the communications task. This task is represented by a TCB built into the nucleus at system generation. Routines operating under this TCB perform all input/output functions related to console communications. The communications task is performed by three modules in the system link library and by the transient SVC routine CHATR. The link modules are: an Initialization module, the Unit Control module, and a Wait module. The Initialization module sets up control blocks when the nucleus is initialized. The Unit Control module (UCM) is set up by the Initialization routine, and is the primary control table for console communications. The Wait module receives control when the communications task becomes active.

The Wait module issues a WAIT macro instruction, specifying a list of event control block (ECB) addresses. The address of this list is contained in the UCM. When one of the ECBs is posted, the communications task becomes a ready task. When it becomes the active task, it issues the CHATR SVC macro instruction (SVC 72). This SVC includes a common module, the Router module, and four service modules. The services that are performed, in order of priority, are: external interruption processing, attention processing, input/output completion processing, and WTO(R) processing.

The Router routine selects the service to be performed and passes control to one of four process modules. One of the process modules provides external interruption services. The other three provide console input/output services: one handles input/output for the 1052 Printer-Keyboard; the second handles input from unit record devices; and the third, output to unit record devices. Each of the three input/output process modules is associated with an Open/Close support module, which provides control blocks for Data Management and the I/O Supervisor.

The flow of control following an external or input/output interruption from a console is shown in Figure 7-1. This figure also serves as a module directory for console input services.

Console output is initiated when a user or system program issues the WTO or WTOR macro instruction. Both macro instructions result in the performance of the transient SVC 35 routine. This routine adjusts the console queues and prepares for the performance of the communications task.

There are two console queues, the buffer queue and the reply queue. The buffer queue points to messages that are to be written to the operator as a result of the WTO or WTOR macro instruction. The reply queue points to buffers for operator replies to the WTOR macro instruction. The SVC 35 service routine queues messages on the appropriate queue.

The extent of both queues may be limited when the system is generated. An attempt to exceed the limit results in an ENQ macro instruction for the requesting task. The task will receive control again when the number of elements in the queue falls below the limit.

The flow of control for console support output is similar to that for input. The Router module has the additional responsibility of selecting an output device. The process modules issue the EXCP command for the 1052 Printer-Keyboard, or the WRITE macro instruction for a printer. For an operator reply (to the WTOR macro instruction), the I/O Completion Process module

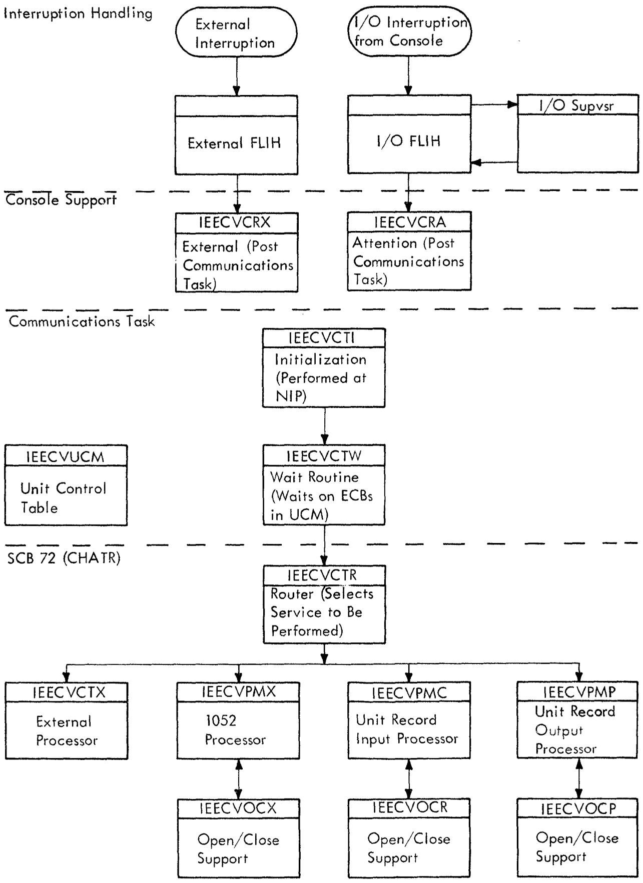


Figure 7-1. Console Support: Input

issues an SVC 34 instruction (Command Processing). The Command Processing service routine determines that the incoming command is a response to the WTOR macro instruction, and passes control to the Reply Processor routine.

Control flow for console support output is shown in Figure 7-2, which also serves as a routine directory.

REPLY PROCESSING

The WTOR macro instruction causes a message to the operator to be written on a console device, and permits a reply from the operator to be returned to the requesting routine. A WAIT macro instruction is also issued by the requestor, specifying the ECB address contained in the WTOR macro instruction. When the operator enters his reply on a console device, the reply is placed in a buffer in the requestor's region, and the specified ECB, also in the requestor's region, is posted.

An operator reply is processed by the communications task Reply Processor routine (IGC1203D). The routine is entered when a reply is received, or when the Rollin Reply Processing routine (RSTRQE) restarts replies that were deferred during rollout of a job step. (See "Freeing One or More Borrowed Regions Through Rollin" in Section 5, "Main Storage Supervision.")

The Reply Processor routine first edits the reply for proper format and length, then finds the reply queue element that represents the specified reply. Subsequent processing depends on whether the job step for which the reply was issued is currently rolled out.

If the job step is currently rolled out (RQERO flag set), the Reply Processor routine invokes the GETMAIN routine to obtain 144 bytes from the system queue area. This space provides a temporary buffer in which the Reply Processor routine saves the reply until the job step is rolled in. The address of the temporary buffer, provided by the GETMAIN routine in register 1, is stored in the RQEXB field of the reply queue element. The Reply Processor routine then moves the current reply to the temporary buffer. Since further reply processing is not possible while the job step is rolled out, the routine returns control

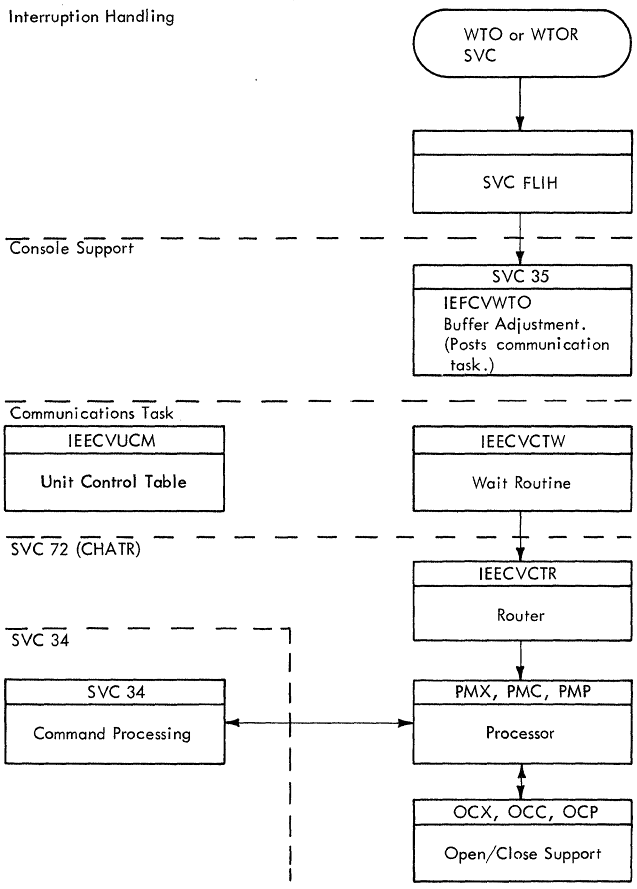


Figure 7-2. Console Support: Output

to the highest priority ready task, via the Exit routine and the Dispatcher.

If the job step is not currently rolled out (RQERO flag not set), the Reply Processor routine examines the RQEXB ("temporary buffer") pointer in the reply queue element. (In the program listing this pointer is called the "purging message address.") If the RQEXB pointer is zero, there is no temporary buffer. This means either that the reply was not received during a previous period when the job step was rolled out, or that the job step was not rolled out. In this case, the routine moves the reply from the system buffer to the user's buffer. It then returns control to the routine's main line to complete the processing of the reply. If, however, the RQEXB pointer is not zero, there is a temporary buffer in which the routine placed a reply during a previous period when the job step was rolled out. In this case, the routine moves the reply from the temporary buffer to the user's buffer, then clears the pointer to the temporary buffer, and invokes the FREEMAIN routine to free the buffer's space. It then returns control to the routine's main line to complete the processing of the reply.

The Reply Processor routine completes the processing of the reply by:

- Removing the reply queue element from the reply queue and freeing its storage space.

- Returning (queueing) the reply identification to the identification assignment pattern (UCMRPYI) in the unit control module. The reply identification is then available for reuse when a new WTOR macro instruction is issued.

- Decreasing by a count of one the identication assignment counter in the unit control module. This count indirectly indicates the number of reply identifications that are available for use.

- Invoking the Post routine to post the message-issuing routine's ECB.

- Returning control to the highest priority ready task, via the Exit routine and the Dispatcher.

SUPPORTING THE SYSTEM LOG

The system log is a pair of data sets maintained by the system for storage of statistical information; it is an optional feature of the operating system. The log is placed on a permanently mounted volume and cataloged when the system is generated. It is available for the use of any program.

The log is initialized by the IEEVLIN routine during nucleus initialization. This routine operates as part of the master scheduler task. The IEEVLIN routine searches the catalog to locate the log. If the search is not successful, the operator is notified that the log option is not included in the system; in this case, log requests will be ignored by the control program. If the IEEVLIN routine locates the log, it opens one of the data sets, creates a DCB for the log, and sets up the resident log control area.

Users communicate with the log through the macro instruction WTL (write to log) and the commands LOG and WRITELOG. The WTL routine (SVC 36) schedules the entering of designated information into the log. The LOG command is used to enter information into the log from the console. The WRITELOG command is used to request that the contents of the log be written by a SYSOUT writer of a particular class.

The supervisor controls responses to all three types of requests through the log control area. This area is an 8-word block that contains control information about the log. It includes an ECB that is used by the LOG command, and the address of an ECB that is used by the WTL macro instruction and the WRITELOG command.

Figure 7-3 shows the internal organization and control flow of log-related activity. The LOG and WRITELOG commands are both initially handled by the SVC 34 service routine (command processing). This processing includes issuing a POST macro instruction for the ECB associated with the particular command. In the case of a LOG command, the SVC 34 routine issues the WTL macro instruction after posting the ECB.

The WTL routine (SVC 36) may be entered directly as a result of a WTL macro instruction, or indirectly through a LOG command. The routine obtains main storage for the requested message, places the message on a chain of requested messages (called the log chain), and issues a POST macro instruction, specifying the WTL ECB.

When either log ECB is posted, the IEEVWAIT routine of the master task is made ready to be performed. This routine passes control to the Log Writer routine. If the request was to write a message on the log (either a WTL macro instruction or a LOG command), the Log Writer routine writes the message. If the request resulted from a WRITELOG command, the Log Writer routine branches to the IEEVLOPN routine, which opens the SYSOUT data set. The Log Writer routine then attaches the Log Dispatcher task, which schedules the output operation.

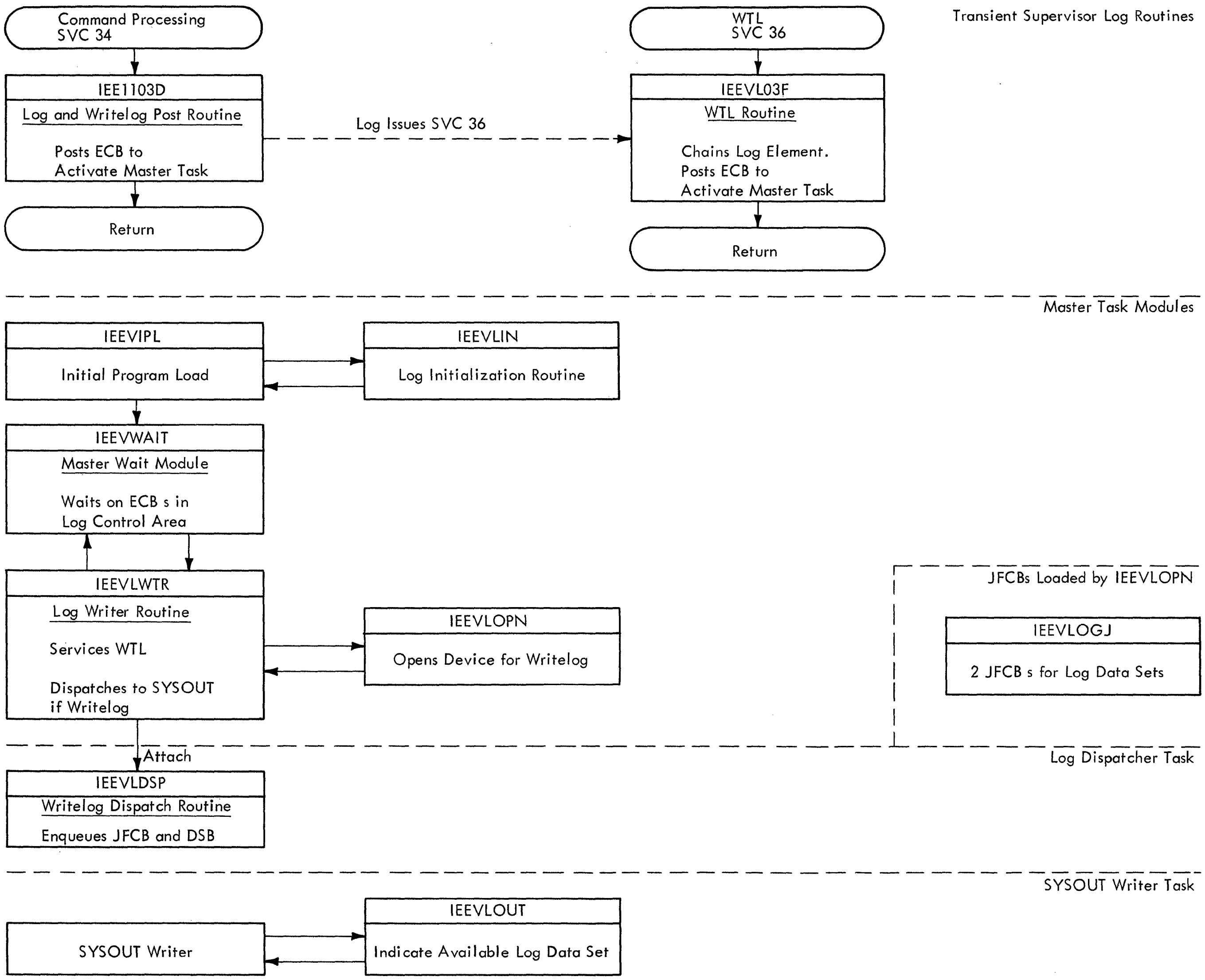

Transient Supervisor Log Routines
Command Processing SVC 34
IEE1103D
Log and Writelog Post Routine
Posts ECB to Activate Master Task
Return
Log Issues SVC 36
WTL SVC 36
IEEVL03F
WTL Routine
Chains Log Element. Posts ECB to Activate Master Task
Return
Master Task Modules
IEEVIPL
Initial Program Load
IEEVLIN
Log Initialization Routine
IEEVWAIT
Master Wait Module
Waits on ECB s in Log Control Area
IEEVLWTR
Log Writer Routine
Services WTL
Dispatches to SYSOUT if Writelog
IEEVLOPN
Opens Device for Writelog
JFCBs Loaded by IEEVLOPN
IEEVLOGJ
2 JFCB s for Log Data Sets
Attach
Log Dispatcher Task
IEEVLDSP
Writelog Dispatch Routine
Enqueues JFCB and DSB
SYSOUT Writer Task
SYSOUT Writer
IEEVLOUT
Indicate Available Log Data Set

Figure 7-3. Log Functions

The Checkpoint/Restart facility allows a job to be restarted after an abnormal termination. The retry can begin at the start of a job step, or within a job step, and prior steps and portions of a step can be skipped if they executed successfully before the termination. The supervisor provides the following two Type 4 SVC routines to handle restart within a job step (called a checkpoint restart):

- The Checkpoint routine, called by the CHKPT macro instruction (SVC 63) in the problem program at points where the programmer wishes a reexecution to begin.

- The Restart routine (SVC 52), called by a job management routine when the restarting step is scheduled.

Restart at the beginning of a step (a step restart) is handled by job management, and is documented in the publication IBM System/360 Operating System: MVT Job Management Program Logic Manual, Form Y28-6660.

The Checkpoint routine creates a series of records (a checkpoint entry) in a data set provided by the calling task. The records include a copy of the task's main storage region, descriptions of data sets, and system control information. The Restart routine interprets the information in the checkpoint entry and uses it to restore the task to main storage, mount, verify and position its data sets, and give it control at the point where the checkpoint entry was written.

CHECKPOINT (SVC 63)

The Checkpoint routine is called by a problem program with the CHKPT macro instruction to create a checkpoint entry. The calling program supplies a DCB for the checkpoint data set and, optionally, a name (CHECKID) for the entry. The Checkpoint routine writes four types of records in the checkpoint entry:

- A Checkpoint Header Record (CHR). The CHR describes a checkpoint and contains checkpoint/restart tables and flags.

- Data Set Descriptor Records (DSDRs). Each DSDR describes a data set and contains a Job File Control Block (JFCB), a JFCB extension, or the Generation Data Group Bias Count Table (GDGBCT).

- Core Image Records (CIRs). The CIRs contain a copy of the caller's main storage region at the time he issued CHKPT.

- Supervisor Records (SURs). The SURs contain the supervisor control blocks that will be needed to restart the task.

The Checkpoint routine is logically divided into several functions, which are listed below with the names of load modules that implement them:

- Checking parameters and system environment (IGC0006C, IGC0106C, and IGC0206C). The Housekeeping routine tests the CHKPT operands for validity, and ensures that the task is eligible for a checkpoint. A work area is obtained and formatted, the Job Control Table (JCT) is read in, and the CHR is built.

- Purging I/O requests (IGC0506C). The Check I/O routine removes the caller's pending I/O requests from the logical channel queues, and allows any active requests to complete.

- Describing the caller's data set status (IGC0A06C and IGC0D06C). The Preserve routine writes out the CHR, and then builds and writes out a DSDR for each data set.

- Copying the caller's region (IGC0F06C, IGC0G06C, and IGC0H06C). The Checkmain routine creates the CIRs by copying the caller's region(s), except for the checkpoint work area, into the checkpoint entry, and then builds and writes out the SURs from information in system control blocks.

- Reissuing the I/O requests (IGC0N06C). The Resume I/O routine returns the caller's pending I/O requests to the logical channel queues.

- Clean up, report, and exit (IGC0Q06C and IGC0S06C). The Checkpoint Exit routine returns the storage obtained with GETMAIN, returns the JCT to the input queue, writes a console message noting success or failure to write a checkpoint, closes the checkpoint data set if checkpoint opened it, and returns to the caller with an SVC 3 (EXIT) instruction.

The first module of the Checkpoint routine is loaded by the SVC SLIH, and subsequent modules are called into the SVC transient area by XCTL. Figure 8-1 shows the order in which the routines are executed, and the information each routine processes.

If an error is detected at any point during checkpoint processing, the Checkpoint Exit routine is called. An error message is written, and an error code is returned to the caller, so that execution may continue without the checkpoint.

PARAMETER AND ENVIRONMENT CHECK

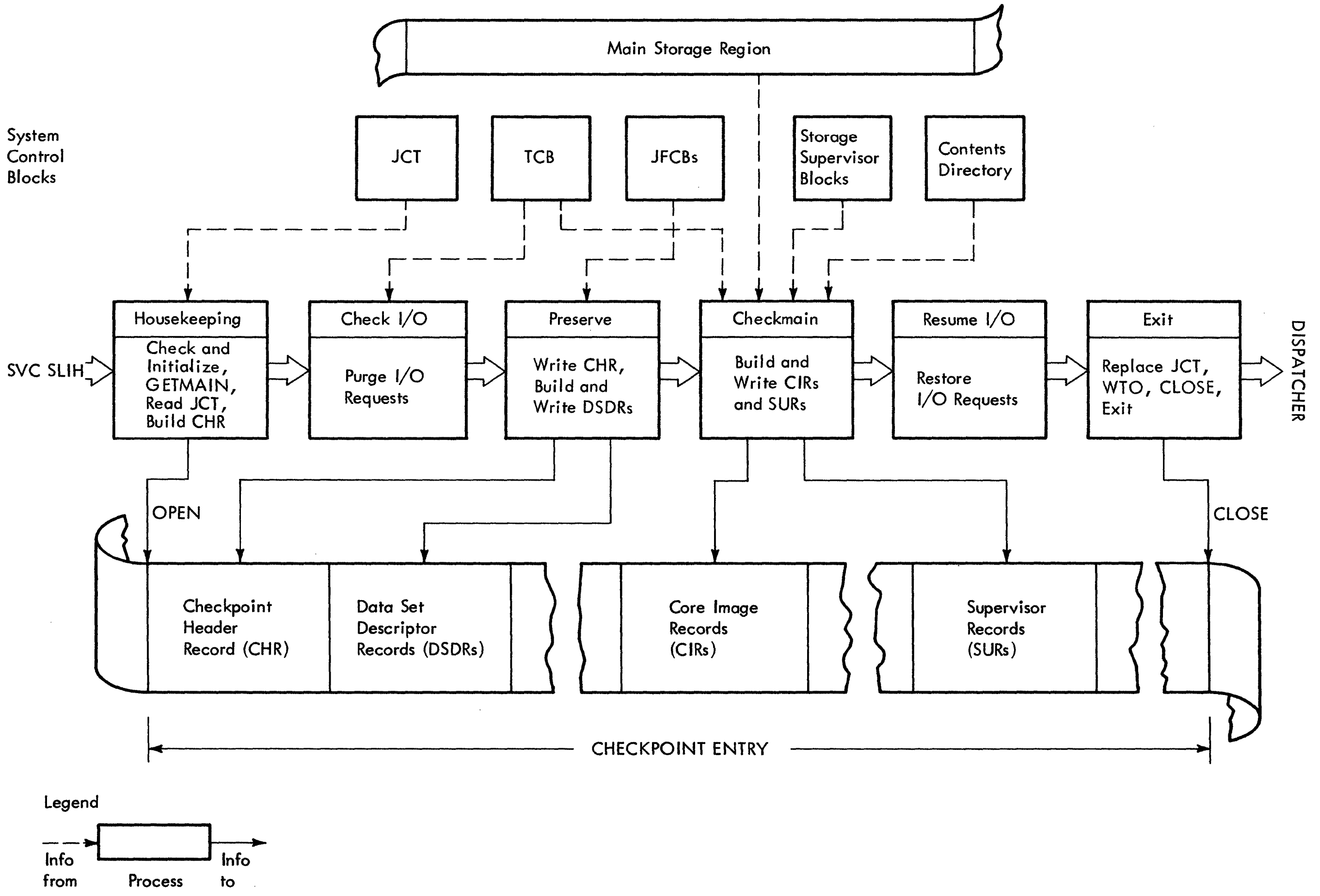
The first three load modules of the Checkpoint routine test the calling parameters and system environment for conditions that would prevent successful checkpoint processing. If no errors are detected, a work area is obtained and formatted, and the JCT is read. A CHR is built in the output buffer, and the Check I/O routine is called.

Parameter Check (IGC0006C)

The first module sets the system mask to allow all interruptions, then inspects the checkpoint flags in the TCB to determine if checkpoint entries have been suppressed by the RD parameter of the job control statements. If they have been suppressed, SVC 3 (EXIT) is issued to return to the caller. A test is also made for the CANCEL operand of the CHKPT macro instruction. CANCEL processing is discussed below.

For normal checkpoint processing, the first housekeeping module calls the supervisor's Validity Check routine (IEA0VL00) to ensure that the addresses supplied for the checkpoint DCB and CHECKID field are within the problem program region. An invalid address prevents further processing, and the Checkpoint Exit routine is called.

The checkpoint DCB, supplied by the caller, shows whether the checkpoint data


Main Storage Region
System Control Blocks
JCT
TCB
JFCBs
Storage Supervisor Blocks
Contents Directory
SVC SLIH
Housekeeping
Check and Initialize, GETMAIN, Read JCT, Build CHR
Check I/O
Purge I/O Requests
Preserve
Write CHR, Build and Write DSDRs
Checkmain
Build and Write CIRs and SURs
Resume I/O
Restore I/O Requests
Exit
Replace JCT, WTO, CLOSE, Exit
DISPATCHER
OPEN
CLOSE
Checkpoint Header Record (CHR)
Data Set Descriptor Records (DSDRs)
Core Image Records (CIRs)
Supervisor Records (SURs)
CHECKPOINT ENTRY
Legend
Info from
Process
Info to

• Figure 8-1. Checkpoint Processing Routines set has been opened. If it has not, an OPEN is issued for it, and a flag is set indicating it must be closed before exit. Then the size of the checkpoint work area is calculated by the following formula:

WA = TIOT + 1108 + 48 (DEBs - 2)

where TIOT is the length of the Task I/O Table (dependent on the number of DD entries).

1108 is a fixed table area.

48 (DEBs - 2) is the number of Data Extent Blocks less two, times 48.

A conditional GETMAIN, specifying Subpool 250, is issued for this area. If the GETMAIN is not successful, the Checkpoint Exit routine is called. If an area is returned, its upper and lower boundaries are checked to see that it does not come within 18 bytes of the region limits. (Because the work area will not be copied into the checkpoint entry with the rest of the region, a 17-byte or smaller "leftover" would be too small to write as a tape record.) If the work area is too close to the upper region boundary, all but the invalid part is released with FREEMAIN, and a second GETMAIN is issued. If the second GETMAIN is successful, the invalid portion of the first area is released. If the second GETMAIN is not successful, or if the first area returned is too close to the lower region boundary, the Checkpoint Exit routine is called, and no checkpoint entry is written.

When the work area is obtained, the region boundaries, the address of the checkpoint DCB, and offsets to input buffers are stored in it, and the second housekeeping module is called.

Environment Check (IGC0106C)

The second housekeeping module tests characteristics of the checkpoint data set and the calling task for checkpoint suitability. If any error is detected, the Checkpoint Exit routine is called and no checkpoint is written. The invalid conditions, in the order tested, are:

- Checkpoint data set not on a direct access or magnetic tape device.
- Key length not equal to zero when the checkpoint data set is on a direct access device.
- Record format is not "undefined."
- Blocksize specified in the DCB is not zero or not greater than 600.
- The data set was not opened for output.
- Physical sequential or partitioned organization was not specified.
- A timer interval is pending.
- An IRB or SIRB is pending on the RB chain.
- A Type 3 or 4 SVRB is pending (other than IGG0551A, EOV.)
- Rollout is being invoked.
- The calling task is or has a subtask.
- A WTOR is pending.
- The CHECKID is missing, is too long, or contains invalid characters.

In addition to these tests, a check is made for active ENQs. If any are pending, a warning message will be issued, at completion of processing, informing the programmer it will be the program's responsibility to reestablish the ENQs on a restart.

If the caller has not specified the checkpoint entry blocksize in the DCB, a DEVTYPE macro instruction is issued to obtain the maximum blocksize for the device, which is entered in the DCB. (The blocksize will be reset to zero before return to the caller.) Normal exit from this module is to IGC0206C, for JCT processing.

JCT Processing (IGC0206C)

The third housekeeping module constructs a channel program and I/O control blocks in the work area, and reads in the JCT from the input queue. The Checkpoint Exit routine is called if an I/O error occurs.

The count of the number of checkpoints taken for the current job is incremented in the JCT, and, if no CHECKID was supplied by the caller, one is generated (C'C' plus the seven-digit number of checkpoints taken).

A Checkpoint Header Record (CHR), shown in Figure 8-2, is constructed in the work area and padded to 400 bytes with binary 1's. The CHR is left in the output buffer and written later. Normal exit is to the Check I/O routine.

CANCEL Processing

The CANCEL operand of the CHKPT macro instruction indicates that the caller does

dec	hex				
0				Number of CHKPTS	CHECKID Length
4	4	CHECKID (left justified) (Checkpoint Entry Identification)			
20	14	DDNAME of CHECKPOINT Data Set			
28	1C	Lower Boundary of Problem Program Storage		Upper Boundary of Problem Program Storage	
36	24	CHKPT Blocksize	TIOT Length	CHECKPOINT Work Area Size	
44	2C	CHECKPOINT Work Area Address		CHECKPOINT SVRB Address	
52	34	Lower Boundary of IBM 2361 Core Storage Area (Hierarchy 1)		Upper Boundary of IBM 2361 Core Storage Area (Hierarchy 1)	

• Figure 8-2. CHECKPOINT Header Record (CHR)

not want a checkpoint entry to be created, but wants to suppress an automatic restart at any preceding checkpoints. If CANCEL is specified, IGC0006C issues a GETMAIN for a small work area, and calls IGC0206C module to read the JCT. Control is then passed to the Checkpoint Exit module (IGC0Q06C), where the checkpoint taken flag is set off, the JCT is returned to the input queue, and control is returned to the caller. In case of a subsequent abnormal termination, there is no indication in the JCT that checkpoint entries exist for the failing step, and no checkpoint restart is performed. The entries are retained in the checkpoint data set; the programmer may restart the step from one of these entries by submitting the proper restart Job Control Language at a later time.

PURGING I/O REQUESTS

The Check I/O routine consists of one module, IGC0506C, which intercepts pending I/O requests initiated by the caller. Check I/O obtains a pointer to the chain of DEBs from the caller's TCB, and issues the PURGE macro instruction, specifying the QUIESCE option, for each DEB in the chain. The SVC Purge routine removes any I/O requests associated with the specified DEB from the Logical Channel Queues of the I/O Supervisor. If a request has already been started, SVC Purge allows it to complete normally before returning to Check I/O.

If an error occurs in completing an active I/O request for a QSAM or QISAM data set, Check I/O tests if the user has specified the QSAM ACC option ("accept errors") in the DCB. If ACC is specified, the error is ignored. Otherwise, the Resume I/O routine is called, and no checkpoint is written. An error message (IHJ00-0I) is written to the operator indicating unsuccessful completion because of an I/O error. Data sets with other organizations are not checked for I/O errors. When all of the caller's I/O activity has subsided, the Preserve routine is called.

DESCRIBING DATA SET STATUS

The Preserve routine consists of two load modules, IGC0A06C and IGC0D06C. The first writes out the CHR already built (by IGC0206C) in the output buffer; the second builds and writes out Data Set Descriptor Records (DSDRs) for each data set. If either module detects an end-of-volume for the checkpoint data set on tape, IGC0206C is called to reprocess with a new tape. If the checkpoint data set is on a direct access device, or if end-of-volume is reached a second time on tape, the Resume I/O routine is called, and no further processing takes place.

Writing Out the CHR (IGC0A06C)

The Preserve routine writes out the CHR, which is always 400 bytes long. If the checkpoint data set is a partitioned data set, a NOTE macro instruction is issued, and the relative track address returned is saved in the work area. Control is passed to the second module.

Building and Writing DSDRs (IGC0D06C)

The second module of the Preserve routine reads JFCBs from the input queue, obtaining the track addresses from the TIOT. For each JFCB, a Type 1 DSDR, consisting of a 2-byte identification (X'0000'), the 176-byte JFCB, the DDNAME, and the UCBTYP field from the UCB, is constructed in the output buffer. If JFCB extensions are associated with the JFCB, they are read in, and a Type 2 DSDR is constructed for each, consisting of the identification X'0004', and the 176-byte JFCB extension. Whenever the 400-byte buffer is filled, it is written out to the checkpoint data set. When the end of the TIOT is reached, Preserve checks for the existence of a Generation Data Group Bias Count Table (GDGBCT). If one exists, as many Type 3 DSDRs as necessary to contain it are built and written. A Type 3 DSDR has the identification code X'0008' and a 176-byte segment of the GDGBCT. The format of the DSDRs is shown in Figure 8-4. Normal exit is to Checkmain.

COPYING THE REGION

The Checkmain routine consists of three load modules, IGC0F06C, IGC0G06C, and IGC0H06C. The first copies the caller's region into the Checkpoint data set as CIRs, and the second and third create SURs from the main storage supervision and contents supervision control blocks. Any I/O error within these modules causes the Resume I/O routine to be called, with an error code returned to the caller. If end-of-volume is detected on tape for the first time, control is transferred to IG0206C to attempt reprocessing with a new tape. If end-of-volume is detected on tape for a second time, or on a direct access device, Resume I/O is called with an error code.

Writing CIRs (IGC0F06C)

The first checkmain module determines the limits of the checkpoint work area, which is the only portion of the caller's main storage region which will not be written in the checkpoint entry. The first area copied is the portion of the region extending from the top of the checkpoint work area to the upper limit of the region. The next portion copied is from the lower limit of the region up to the lower boundary of the work area. If necessary, storage assigned to the task in hierarchy one is copied last. Blocks are written out according to the blocksize supplied by the caller, or the maximum blocksize of the device, if the caller did not specify blocksize. Data is not moved to a buffer for writing, but is copied from its location in the region. The last record written out for each of the three storage areas is normally shorter than the specified blocksize. Such short records are extended to at least 18 bytes.

Building and Writing SURs (IGC0G06C and IGC0H06C)

The SURs are constructed in a 200-byte output buffer in the work area, which is written out whenever it is full. The fields within the SUR may vary in content and length, so each is prefixed by a one-byte type code and one-byte length field as it is placed in the buffer. IGC0G06C first inserts the PQEs associated with the problem program TCB, then the SPQEs and DQEs associated with the TCBs of the system task control routine, the initiator, and the problem program. Next the SVRBs and PRBs are added in the order they are found on the RB chain, and the LLEs are added last. IGC0H06C adds CDEs, the address of the PIE, the address of the first problem program save area from the TCB, the address of the pointer to the problem program save area from the TQE, the general registers from the checkpoint SVRB, each problem program DEB, any IRBs attached to these DEBs, the floating-point registers, the checkpoint DCB, the address of the SYNAD routine in the checkpoint DCB, and the TIOT. Normal exit is to the Resume I/O routine.

RESTORING I/O REQUESTS

The Resume I/O routine (IGC0N06C) searches through the chain of DEBs from the caller's TCB. If a DEB has an entry in the DEBUSRPG field, it indicates I/O requests were purged for that data set. Resume I/O issues a RESTORE macro instruction for each DEB with such an entry. The Restore routine returns the purged I/O requests to the Logical Channel Queues of the I/O Supervisor. When all the DEBs have been checked, the Checkpoint Exit routine is called.

CHECKPOINT EXIT ROUTINE

The Checkpoint Exit routine is normally entered from the Resume I/O routine, but may be called from prior modules if an error is detected. The routine consists of two modules: IGC0Q06C, a general clean-up procedure, and IGC0S06C, a message routine.

General Clean-up (IGC0Q06C)

The first exit module checks to see if a checkpoint work area was obtained. If no work area exists, processing did not begin, and the message module is called to report the error and return to the caller. A test is also made for the CHKPT CANCEL operand. Processing for CANCEL was discussed above under "Parameter and Environment Check."

If the checkpoint entry was written, and the checkpoint data set has partitioned organization, a STOW macro instruction, specifying the CHECKID as member name, is issued to add the entry to the data set directory. If no checkpoint entry was written because of an error, a FREEMAIN is issued for the checkpoint work area, and control is passed to the message module.

The CHECKID is placed in the JCT to identify the most recent checkpoint entry for the job, and the checkpoint volume serial number or track address are placed in the appropriate JCT fields. (If automatic restarts have been suppressed by job control statements, only the CHECKID is moved to the JCT.) The updated JCT is written out to the input queue, the checkpoint work area is returned via FREEMAIN, and the message module is called.

Message Module (IGC0S06C)

The last checkpoint module issues a GETMAIN for a message buffer and small work area, and determines the type of message to be issued from the return code and error code passed in the extended SVRB save area. One of the following messages may be written: IHJ000I, IHJ001I, IHJ002I, IHJ004I, or IHJ005I. The jobname, checkpoint DDNAME, and, if a checkpoint entry was created, the volume serial number, unit name, and CHECKID of the entry, are moved into the message area, and a WTO macro instruction is issued.

If the Housekeeping routine opened the checkpoint data set, a CLOSE is issued. The message area is released via FREEMAIN, and one of the following return codes is placed in Register 15:

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 at RESTART.

SVC 3 (EXIT) is then issued to return to the Dispatcher.

RESTART (SVC 52)

Restart of a program within a job step is accomplished by using the information stored in a checkpoint entry to recreate the conditions that existed when CHKPT was issued. Interpretation of the checkpoint entry is done by both job management and supervisor routines. When the step to be restarted is scheduled, job management inserts an extra job step (IEFDSDRP) in front of it, which adjusts the input queue, and reads the DSDRs to build JFCBs for the restarting step's data sets. IEFDSDRP also ensures device allocations that are compatible with those that existed at CHKPT time. Just before exit, IEFDSDRP changes the name of the restarting step to IEFRSTRT. When this program is brought into storage and given control, it issues SVC 52, causing the first load module of the Restart routine to be brought into an SVC transient area. Restart uses the TCB and other control blocks assigned to IEFRSTRT to recreate the system environment for the restarting step.

The major functions of restart, and their relationship to the job management routines and the checkpoint entry, are shown in Figure 8-3. The functions are listed below, with the names of the load modules implementing them:

- Obtaining and formatting storage (IGC0005B and IGC0105B). The Housekeeping routine issues a GETMAIN for storage in the problem program region, builds work tables and buffers for the following routines, and positions the checkpoint entry to the first CIR.

- Restoring the step to main storage (IGC0505B, IGC0605B, IGC0705B, IGC0805B, and IGC0905B). The Repmain routine reads the CIRs to restore the step to its region in main storage, and processes the SURs to rebuild the task supervision control blocks and queues.

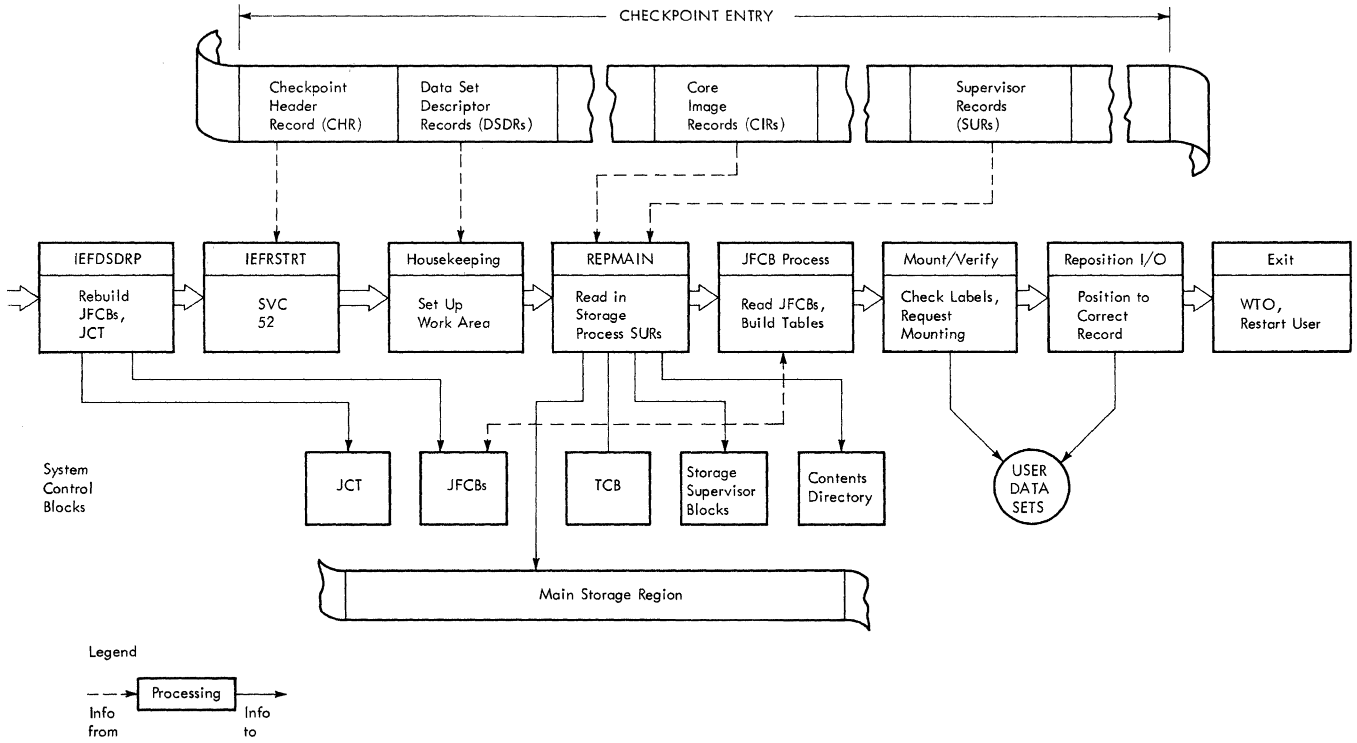
- JFCB Processing (IGC0G05B and IGC0I05B). The JFCB processor interprets the JFCBs (already rebuilt by IEFDSDRP) and builds tables describing each open data set in the restart work area.

- Mounting and verifying volumes (IGC0K05B and IGC0M05B). The Mount/Verify routine processes volume lables (calling a user label routine if necessary), and requests the operator to mount missing volumes.

- Positioning open data sets (IGC0N05B, IGC0Q05B, IGC0P05B and IGC0R05B). The Data Set Processor adjusts the problem program's data sets to the record being processed when CHKPT was issued.

- Restarting I/O requests (IGC0T05B). If the problem program had I/O requests pending when CHKPT was issued, the Access Method-Disposition routine returns these requests to the Logical Channel Queues to be restarted. This routine also adjusts Partitioned Data Set directories.

- Returning control to the step (IGC0V05B). The Restart Exit routine frees the restart work area, writes a



• Figure 8-3. Restart Processing Routines

message to the console, and returns control to the restarting step through the Dispatcher.

OBTAINING AND FORMATTING STORAGE

The Housekeeping routine consists of two load modules, IGC0005B and IGC0105B. The first obtains storage, transfers information into it, and opens the checkpoint data set. The second constructs the I/O blocks and channel programs needed to read the checkpoint entry.

Obtaining Storage (IGC0005C)

The first load module of Restart receives a parameter list in the extended SVRB save area containing information from the checkpoint header record and DSDRs, processed by IEFDSDRP. From this parameter list, the Housekeeping routine obtains the limits of the restarting step's region, and issues a GETMAIN for all of it. The parameter list also contains a pointer to the checkpoint work area within the region, and Restart Housekeeping sets up the same area as a work area. A BSAM DCB for the checkpoint data set is constructed in the work area, and an OPEN is issued. Part of the problem program region is temporarily freed with FREEMAIN for the OPEN routine. The second module of the Housekeeping routine is called after the OPEN.

Checkpoint Data Set Initialization (IGC0105B)

The second Housekeeping routine module moves the checkpoint data set IOB and channel program to the work area. If the data set is on a tape device, successive records are read until the tape is positioned at the first CIR. If the data set is on a direct access device, a POINT macro instruction is issued to position the data set at the first CIR. Exit is to the Repmain routine.

RESTORING THE STEP TO MAIN STORAGE

The Repmain routine consists of five modules (IGC0505B, IGC0605B, IGC0705B, IGC0805B, and IGC0905B). The first copies the CIRs into their original positions in the step's region; the other four read and process the SURs to recreate the system control blocks and queues that existed for the restarting task at CHKPT time. An I/O error or end-of-volume in any of the modules causes transfer to the Restart Exit routine for termination.

Restoring Main Storage (IGC0505B)

The first module of the Repmain routine reads CIRs into the areas of main storage from which they were written. The first CIRs are read into the area between the upper limit of the restart work area (which corresponds to the checkpoint work area) and the top of the region. The second area copied is from the lower limit of the region to the bottom of the restart work area. Hierarchy 1 is restored last, if present. The first Repmain module also restores the PQEs, the SPQEs and the DQEs for the TCBs of the initiator, the system task control routine, and the restarting task. These elements are the first fields in the SURs.

SUR Processing (IGC0605B, IGC0705B, IGC0805B, IGC0905B)

The other Repmain modules continue the processing of the SURs. The contents supervision blocks are replaced with the saved CDEs, Extent List, and LLEs. The contents supervision blocks are then freed. Internal queue pointers within these blocks are adjusted as they are returned to system queue space. Finally the current TCB (originally assigned to IEFRSTRT) is updated with the information saved from the restarting task's TCB. The TIOT is the last control block read in, before control is passed to the JFCB Processing routine. If an I/O error or EOV occurs, control is passed to IGC0905B which frees all partially restored chains.

JFCB PROCESSING

This routine counts the data sets that were open at CHKPT time, and builds a data set description table in the restart work area for each data set. In another part of the work area, a set of I/O control blocks (DCB, IOB, DEB, and a channel program) is constructed for each data set. The JFCBs processed were constructed from the DSDRs in the checkpoint entry by IEFDSDRP. The first module of the JFCB Processing routine (IGC0G05B) builds the tables and control blocks, the second (IGC0I05B) makes adjustments for data sets residing on more than five volumes.

Table Build Module (IGC0G05B)

The JFCB Processing routine assigns a 304-byte section within the restart workarea to each DEB chained to the restarting task's TCB. Then the "new" TIOT is searched for the DDNAME corresponding to each open data set. The disk addresses of the JFCBs are obtained from the TIOT, and the JFCBs are read in. If an I/O error occurs, or a DDNAME is missing, control is passed to the Restart Exit routine. When all JFCBs have been read in, a DCB, DEB, IOB, ECB, and channel program are constructed for each data set. Up to five volume identifications are moved from the JFCB to the associated data set description table, and a flag is set if it will be necessary to read JFCB extensions later for additional volumes.

Table Complete Module (IGC0I05B)

The second JFCB processing module reads the JFCB extension for those data sets residing on more than five volumes. For non-concatenated partitioned data sets and sequential data sets, the volume in use at CHKPT time is placed at the top of the description table list of volumes, and it will be the only one mounted later. A flag is set for multi-volume ISAM, BDAM, and concatenated partitioned data sets to indicate that all volumes on which they reside will have to be mounted.

MOUNTING AND VERIFYING VOLUMES

The Mount/Verify routine ensures that the correct volumes are mounted for the user's data sets, and requests the operator to mount any that are missing. The user's non-standard tape label routine is called to verify data sets with non-standard labels. The routine consists of two modules: IGC00K05B, which processes all data sets not on a direct access device, and IGC0M05B, for direct access device data sets.

Non-Direct Access Processing (IGD0K05B):

The non-direct access Mount/Verify module checks each of the data sets description tables, and processes all except those for direct access data sets and null data sets. For SYSIN, SYSOUT, unit record, and graphic data sets, the DEB is adjusted, and no mount verification is performed.

For data sets on magnetic tape, the volume serial number in the data sets description table (the number saved at CHKPT time) is compared to the volume serial number in the primary UCB specified by the data set's TIOT entry. If the volume serial numbers do not match, the secondary UCBs, if any are checked. If no match is found, a suitable UCB is selected from the TIOT list, and the operator is requested to mount the volume.

When the volume serial number is located, or when the volume is mounted, the tape label is read and checked, and the tape is rewound. If it is not the correct volume, or if a standard label is present and the JFCB indicates it should not be, a message is written to the operator, and the tape is unloaded. If it is the correct volume, the UCB and DEB are adjusted, and the UCB becomes the primary UCB in the TIOT entry.

When the end of the description tables is reached, a second pass is made through, checking for input volumes with non-standard labels. A user-supplied label verification routine is called if any are present. On completion, the direct access Mount/Verify module is called, unless there are no direct access data sets. In this case the first Position I/O module is called.

Direct Access Mount/Verify Module (IGC0M05B)

The second module of the Mount/Verify routine compares the volume serial number in the data set description table (saved at CHKPT) with the volume serial number in the primary UCB listed in the TIOT entry for each direct access data set. If the numbers match, the DEB and UCB are adjusted. If the numbers do not match, the secondary UCBs listed in the TIOT are checked. If no match is found, a suitable UCB is selected, and the operator is requested to mount the volume.

For sequential and single partitioned data sets, only the volume in use at CHKPT time is mounted. All volumes on which ISAM, BDAM, or concatenated partitioned data sets reside are mounted. If necessary, JFCB extensions are read to find the volume identifications.

If an error occurs in either of the Mount/Verify modules, Restart is terminated by calling the Exit routine. If no error occurs, control is passed to the Data Set Processor routine. The tape module is called first, unless all data sets are on direct access devices.

POSITIONING OPEN DATA SETS

The SYSIN/SYSOUT Data Set Processor 1 module (IGC0N05B) adjusts the DCB, DEB and channel programs for SYSIN or SYSOUT direct access data sets on a deferred restart. These data sets which existed at the time checkpoint was issued have been deleted, and new SYSIN or SYSOUT data sets have been allocated at restart time. The name of the reallocated data set is obtained from the JFCB, and the VTOC is searched for the DSCB. Extents from the DSCB are used to construct a new DEB. If the number of extents in the DSCB equal the number of extents in the old DEB in use at checkpoint time, the new DEB is constructed in the same space as the old DEB. Otherwise, GETMAIN is issued to obtain space for the new DEB, and the old DEB space is freed.

For SYSIN data sets, the following absolute disk addresses (MBBCCHHR) that point to the old data set are changed to absolute disk addresses that point to the same positions in the new data set:

1. Full disk address in the DCB - This address is changed to point to the next record to be read from SYSIN in the new data set.

2. IOB seek addresses of the current and next IOB - At Restart time, these addresses point to the old data set (because the channel program for the next read is built during the current read) and now are changed to point to the new data set.

The old disk addresses (MBBCCHHR) are converted into TTR form using the old DEB; after the new DEB is constructed, the addresses are converted back into MBBCCHHR form using the new DEB. The TTR to MBBCCHHR conversion is performed in the next module, IGC0Q05B.

The SYSIN/SYSOUT Data Set Processor 2 (Direct-Access) module (IGC0Q05B) calculates the number of tracks in each extent in each SYSIN or SYSOUT DEB and places the number in the DEB. For SYSOUT data sets, the lower limit of the first extent is placed in the full disk address field of the DCB; the track capacity for the device is also placed in the DCB. For SYSIN data sets, the absolute disk addresses which were converted to TTR form in IGC0N05B are now converted back to MBBCCHHR form using the new DEB. Control is then passed to the Data Set Processor 1 module (IGC0P05B) if there are any non direct-access data sets. Otherwise, control is passed to the Data Set Processor 2 module (IGC0R05B).

The Data Set Processor 1 (IGC0P05B) module works from the data set description tables, processing only entries for magnetic tape data sets. 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 non-standard labeled tapes, which are positioned at the first data record by the user label routine.

Data Set Processor 1 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 BLKCT field is zero or negative, the data set is positioned at the first record or end-of-file, depending whether the forward or backward processing was taking place at CHKPT time. An I/O error in repositioning causes Restart termination. The Data Set Processor 2 module is called on completion, unless there are no direct access data sets.

The Data Set Processor 2 module (IGC0R05B) checks each data set residing on a direct access device for a difference in the space allocation limits in the DEB saved at CHKPT time, and the space allocation limits in current Data Set Control Blocks (DSCBs) in the Volume Table of Contents (VTOC). Any discrepency between a DEB and the associated DSCB for input data sets causes Restart termination, since the data set has been modified since CHKPT.

For output data sets, the smaller of the two space allocations is placed in both the DEB and the DSCB. If the DSCB extents are reduced, the Partial Release module of the CLOSE routine is called to return the released space to the free area on the volume. ENQ and DEQ are used to protect the VTOC from other users during any modification. When all direct access data sets have been checked, the Access Method-Disposition module is called.

RESTARTING I/O REQUESTS

The Access Method-Disposition module (IGC0T05B) checks each output partitioned data set for members added since CHKPT was issued. The partitioned data set directory is read, and if the relative track and record address of any member is greater than that of the member being processed at CHKPT, it is deleted, using the STOW macro instruction.

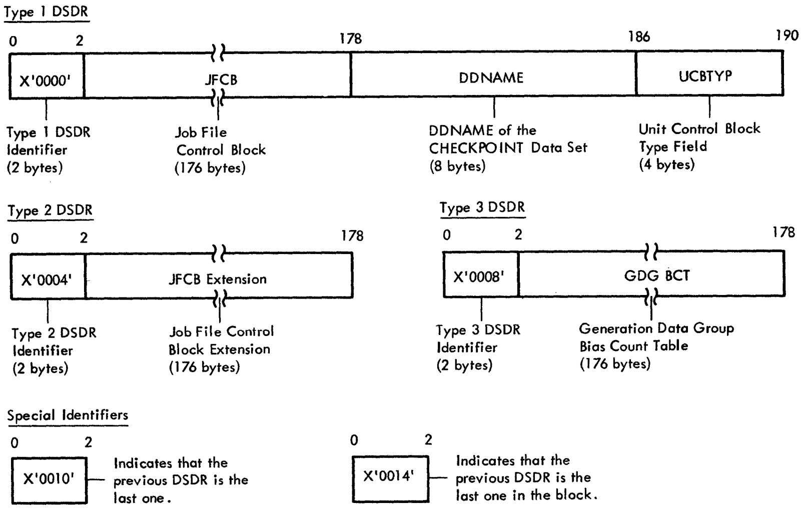
After all partitioned data sets have been checked, the chain of DEBs associated with the problem program TCB is inspected for entries in the DEBUSRPG field. These entries point to a chain of IOBs for user I/O requests which were pending at CHKPT time. The RESTORE macro instruction is issued for each DEB with intercepted requests. This returns the I/O requests to the I/O Supervisor's logical channel queues, where they will be started. Control is then passed to the Exit module.

RESTART EXIT ROUTINE

The Restart exit module (IGC0V05B) tests the error code field in the restart work area to determine if entry was caused by an error in one of the earlier modules. If an error code is present the exit routine places it in the "nn" field of the console message IHJ007I. The message is written with WTO, and ABEND is issued to return to the Dispatcher.

If no error code is found, WTO is used to write console message IHJ008I. The restart work area is released with FREEMAIN, and if the checkpoint routine opened the checkpoint data set, the restart exit routine issues a CLOSE for it.

The exit routine places a return code of X'04' in register 15 to inform the restarting program that a restart has taken place, and exits with an SVC 3 (EXIT). Since the TCB and SVRB have been updated with information saved at CHKPT time, the problem program will be started as though CHKPT had just been issued.


Type 1 DSDR
0
2
178
186
190
X'0000'
JFCB
DDNAME
UCBTYP
Type 1 DSDR Identifier (2 bytes)
Job File Control Block (176 bytes)
DDNAME of the CHECKPOINT Data Set (8 bytes)
Unit Control Block Type Field (4 bytes)
Type 2 DSDR
0
2
178
X'0004'
JFCB Extension
Type 2 DSDR Identifier (2 bytes)
Job File Control Block Extension (176 bytes)
Type 3 DSDR
0
2
178
X'0008'
GDG BCT
Type 3 DSDR Identifier (2 bytes)
Generation Data Group Bias Count Table (176 bytes)
Special Identifiers
0
2
X'0010'
Indicates that the previous DSDR is the last one.
0
2
X'0014'
Indicates that the previous DSDR is the last one in the block.

• Figure 8-4. Data Set Descriptor Records (DSDRs)

Exiting procedures consist of the preparation for return and the actual return of control from a completed program or routine. The program may be a user or system program that has issued a RETURN macro instruction, a completed SVC routine, or a user (asynchronous) exit routine. Control may pass to a user program or to a supervisor termination routine that performs normal termination of the completed program's task. Exiting procedures fall into three main classes:

- Preparing for return from a type-1 SVC routine. This class of exiting procedure is performed by the Type-1 Exit routine.
- Preparing for return from all other types of programs. This class of exiting procedure is performed by the Exit routine.
- Performing the actual return of control. This class of exiting procedure is performed by the Dispatcher (except when the return is from a type-1 SVC routine that returns control directly to the caller).

HANDLING RETURN FROM TYPE-1 SVC ROUTINES

The Type-1 Exit routine handles the return to a user program from a completed type-1 SVC routine. It determines whether control should be returned directly to the caller of the SVC routine, or to the Dispatcher. Control will pass to the Dispatcher if the completed SVC routine has indicated the need for a task switch by altering the "new" TCB pointer, IEATCBP.

The Type-1 Exit routine is entered from any Type-1 SVC routine via a branch. Its first step, a housekeeping step, is to reset the "type-1 switch" to indicate that registers are no longer stored in the lower main storage save area. The ABTERM routine tests this switch during an abnormal task termination to determine whether the routine that called the ABTERM routine is a type-1 SVC routine.

The Type-1 Exit routine then determines whether to return control directly to the caller or to branch to the Dispatcher; it does this by testing if the exiting SVC routine has indicated the need for a task switch. Some type-1 SVC routines, such as the Wait and Post routines, normally place a program in a wait condition or make a program ready, thus requiring a task switch. The Type-1 Exit routine recognizes this condition by testing the doubleword TCB pointers IEATCBP and IEATCBP+4. If both pointers contain the address of the current TCB, no task switch is required; the Type-1 Exit routine restores registers from lower main storage and returns control to the caller. If the two pointers are not equal, a task switch has been indicated, and the Type-1 Exit routine must branch to the Dispatcher. Before branching, the Type-1 Exit routine saves the SVC old PSW in the current request block, and the contents of the caller's registers in the current TCB; this is for eventual return to the caller.

In a multiprocessing system, the routine also inspects the doubleword TCB pointers of the second CPU. If they are unequal, a task switch is required, and the Dispatcher must be entered. The Dispatcher is also entered if the External FLIH bit in FLRETFLG is set, indicating that an external interruption has not been processed (in which case the Dispatcher then passes control to External FLIH). In addition, in a multiprocessing system, before returning control to the caller, zeros are placed in the supervisor lock and CPU identity bytes (the system is unlocked) if the SVC old PSW is completely enabled for interruptions. A completely enabled SVC old PSW indicates that the system was unlocked during the calling routine and must be returned to an unlocked state after completion of the type-1 SVC routine.

PREPARING FOR RETURN FROM PROGRAMS OTHER THAN TYPE-1 SVC ROUTINES

The Exit routine, itself a type-1 SVC routine, handles the exiting procedures for all programs other than type-1 SVC routines. User or system programs gain supervisor-assisted Linkage to the Exit routine by issuing a RETURN macro instruction; SVC routines obtain a similar result by using an SVC-3 instruction. The Exit routine determines the type of program that is exiting. The program can be a user program-check exit routine, a user asynchronous exit routine, an SVC routine, or a user program. For each type of exiting program, some special processing is performed.

If the completed program was the first executed program of its task, and therefore is considered to be at the "highest control level" within that task, the Exit routine recognizes an end-of-task condition, and branches to the End-of-task routine (EOT) to perform normal termination of the caller's task.

The Exit routine dequeues the RB under which the completed program was operating for all types of completed programs except user program check routines, which have no RBs. If the RB had been dynamically acquired via a GETMAIN macro instruction, the Exit routine frees the space occupied by the RB.

When it has completed its processing, the Exit routine branches to the Transient Area Refresh routine, which determines whether an SVC routine that was overlaid in its transient area block (TAB) may be restored to the block. The process of restoring an overlaid SVC routine is called "refreshing" the TAB. If a TAB may be refreshed, the Transient Area Refresh routine initiates the refresh process before branching to the Dispatcher. If no SVC routines were using a TAB, no processing occurs, and the Transient Area Refresh routine branches to the dispatcher.

PREPARING FOR RETURN FROM A USER PROGRAM CHECK ROUTINE

The Exit routine tests whether to perform special processing needed during the return from a user program check routine. When a user program check routine issues a RETURN macro instruction, a branch to an SVC-3 instruction results. The SVC instruction is located in lower main storage, just before the entry point to the Program Interruption FLIH. When the SVC interruption occurs, the address of the next executable instruction (the entry point of the Program Interruption FLIH) is placed by the CPU in the SVC old PSW. The Exit routine compares the address in the SVC old PSW with the address in the program interruption new PSW; if the two addresses are equal, the return is from a user program check routine.

The Exit routine clears the "first-time logic" switch in the user's program interruption element (PIE). The first execution of the SPIE routine for the current task had created a PIE, in which the program old PSW and certain registers are stored during a program interruption. The "first-time logic" switch must be cleared to indicate to the Program Interruption FLIH that the PIE is not active; without such a resetting of the switch, the FLIH would interpret a second program interruption as occurring in the program check routine, and would cause abnormal termination of the current task.

The Exit routine then transfers register contents and the RB old PSW, belonging to the user program that had been interrupted by the program check, to the current TCB. The Exit routine sets up the right half of the RB old PSW in the program's RB from information stored in the PIE. It sets up the left half of the PSW by transferring information from the left half of the SVC old PSW, which was stored when the user program check routine issued a RETURN macro instruction. The reason for constructing the RB old PSW from these two different sources is that (1) the user program check routine has the option of specifying a return point in the interrupted program that is different from the point of interruption, and therefore may store this return address in the right half of the program old PSW in the PIE; and (2) the user program check routine may have accidently altered the left half of the program old PSW stored in the PIE.

After transferring register contents to the TCB and setting up the RB old PSW in the RB, the Exit routine branches to the Dispatcher, which returns control to the interrupted user program. The Dispatcher loads the user's register contents from the current TCB and loads the RB old PSW set up by the Exit routine in the RB. This branch to the Dispatcher is an exception to the normal procedure of branching to the Transient Area Refresh routine.

PREPARING FOR RETURN FROM PROGRAMS CONTROLLED BY RBS

If the returning program is not a user program check routine, the Exit routine determines the type of program by finding out the type of RB under whose control the returning program was operating. The actual test is of the RBSTAB field of the current RB queued to the caller's TCB; this RB is the one which was controlling the returning program. Depending on the type of RB, the Exit routine performs one of three general types of processing.

- If the RB is an SVRB, representing a type 2, 3, or 4 SVC routine, the Exit routine branches to the SVC Second Level Interruption Handler to perform special handling for transient routines.

- If the RB is an SIRB or an IRB, representing a user exit routine, the Exit routine performs special processing for exit routines.

- If the RB is a PRB, representing a user program, the Exit routine performs an exiting procedure needed for contents supervision.

If the Returning Routine Is an SVC Routine

For an SVC routine, the Exit routine branches to the TAHEXIT subroutine (entry point IEAQTR01). The TAHEXIT subroutine performs two functions. (1) It moves saved registers from the SVRB to its TCB, and stores registers 0, 1, and 15 in the TCB. It does this so that the caller of the SVC routine will be redispatched with the proper register values. (2) It removes the SVRB for an exiting transient routine from the transient area queues. Both functions are performed if the exiting program is a transient SVC routine.

The TAHEXIT subroutine manipulates the register save areas so that when the caller of the exiting SVC routine is reentered, its registers 2-14 will contain the same values they had when the SVC was issued. Registers 15, 0, and 1 will contain the values which the SVC routine provided -- normally parameters passed back to the caller.

If the exiting routine is resident (type 2), the TAHEXIT subroutine returns control to the Exit routine. But if the exiting routine is nonresident, TAHEXIT performs additional processing to remove the SVRB from the transient area queues. To do this, the TAHEXIT routine determines the address of the TACT entry for the transient area occupied by the exiting routine. This address is obtained by adding the displacement of the TACT entry (contained in the exiting SVRB) to the address of the transient area control table (IEAQTAQ). (See Figure 9-1.) The TAHEXIT subroutine then searches the user queue associated with the TACT entry, looking for an SVRB which is "using" the exiting routine. (An SVRB is "using" the exiting routine if the TTR address in the SVRB is the same as the TTR address in the TACT entry.)

When an SVRB that is using the exiting routine is found, the TAHEXIT subroutine checks if it is the SVRB that was controlling the exiting routine. If it is, it is dequeued. If it is not, the SVRB represents another request for the routine, and the TAHEXIT subroutine cannot flag the transient area as free. In either case, the entire queue is checked.

When the end of the queue is reached, the TAHEXIT subroutine decreases by one the count of the total number of users of all the transient areas. This count is used by the Transient Area Refresh routine to determine if a search for a routine that should be refreshed is necessary.

The TAHEXIT subroutine flags the associated TACT entry either "in use" or "free," according to whether or not another SVRB for the exiting routine is still in the user queue. The TAHEXIT subroutine then returns control to the Exit routine.

If the Returning Routine Is a User Program

If the test of the RB type indicates a PRB, meaning that a user program is returning control, the Exit routine first moves the user's register contents from their save area in lower main storage, where they had been saved by the SVC FLIH, to the save area in the current TCB. This action is in preparation for the Dispatch- er's restoring of registers just before it returns control to the caller's task.

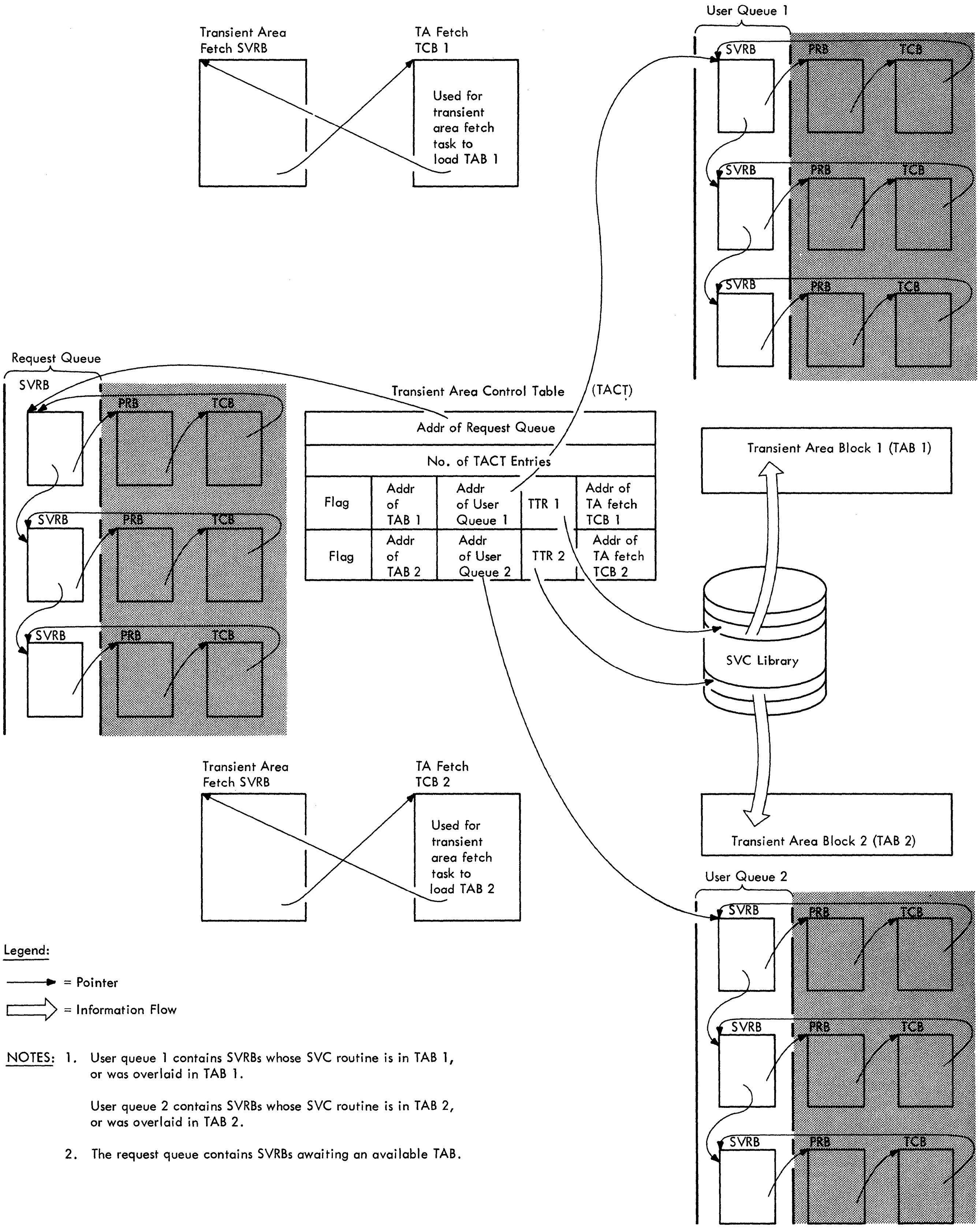
If the returning program is the last to be executed for its task, the Exit routine branches to the End-of-Task (EOT) routine to perform normal task termination. The Exit routine determines this condition by testing the RBTCBNXT flag of the PRB. This flag, if set, indicates that the RBLINK field points directly to the TCB. In this case, the PRB represents the last executed routine of its task.

If the returning program is not the last to be executed for its task, the Exit routine branches to the CDEXIT subroutine to determine if there are other requests for the use of the completed program, and to prepare for reentry to the program if there are such requests. The CDEXIT routine tests if the exiting program has a contents directory entry (CDE); the existence of a CDE is indicated in the CDE field of the PRB. If there is no CDE, the exiting program was entered via use of the SYNCH macro instruction, which does not build a CDE; in this case, the CDEXIT routine returns control to the Exit routine. If there is a CDE, the CDEXIT routine continues processing.

The CDEXIT routine determines the type of CDE. There are two types of CDE -- a major CDE, which is associated with the major entry-point of its program; and a minor CDE, which is associated with an alias or with an entry point set up by the execution of an IDENTIFY macro instruction. If the CDE pointed to by the PRB is a minor CDE, the CDEXIT routine finds the associated major CDE. It then reduces the use/responsibility count in the major CDE.

The use/responsibility count is the number of times the ATTACH, LINK, XCTL, or LOAD macro instructions have been issued for the module. It is used to keep track of the number of outstanding requests for a completed load module or program.

If the exiting program is serially reusable and there is at least one outstanding request for its use (indicated by a nonzero


Transient Area
Fetch SVRB
TA Fetch
TCB 1
Used for transient area fetch task to load TAB 1
User Queue 1
SVRB
PRB
TCB
Request Queue
Transient Area Control Table
(TACT)
Transient Area Block 1 (TAB 1)
SVC Library
Transient Area Block 2 (TAB 2)
Transient Area
Fetch SVRB
TA Fetch
TCB 2
Used for transient area fetch task to load TAB 2
User Queue 2

Addr of Request Queue				
No. of TACT Entries				
Flag	Addr of TAB 1	Addr of User Queue 1	TTR 1	Addr of TA fetch TCB 1
Flag	Addr of TAB 2	Addr of User Queue 2	TTR 2	Addr of TA fetch TCB 2

Legend:

⟶ = Pointer

⇨ = Information Flow

NOTES: 1. User queue 1 contains SVRBs whose SVC routine is in TAB 1, or was overlaid in TAB 1.

User queue 2 contains SVRBs whose SVC routine is in TAB 2, or was overlaid in TAB 2.

2. The request queue contains SVRBs awaiting an available TAB.

Figure 9-1. The Transient Area Queues

RBPGMQ field in the PRB), the CDEXIT routine updates the RB address in the CDE so it points to the next PRB that will control the program. This next PRB is associated with a task different from that of the caller. The address of the next PRB is obtained via the RBPGMQ field of the PRB. The CDEXIT routine makes the new PRB ready by placing zero in its wait count field; the Dispatcher will test this field before dispatching the program. The CDEXIT routine also sets the right half of the old PSW field in the new PRB, in preparation for later entry to the Contents Supervision subroutine CDEPILOG.

The CDEPILOG subroutine will be executed when the Dispatcher recognizes the new PRB's task as the highest priority ready task. (The CDEPILOG subroutine performs final preparation for linkage to the requested program.)

After preparing the next PRB to control the program, the CDEXIT routine branches to the Task Switching routine. This routine tests whether the TCB for the previously waiting PRB may replace the current TCB. It does this by comparing dispatching priorities. If a task switch is needed, the Task Switching routine places the address of the new TCB in the "new" TCB pointer. This pointer will later be tested by the Dispatcher. The Task Switching routine returns control to the CDEXIT routine, which in turn returns control to the Exit routine.

If there are no other requests for the exiting program, the CDEXIT routine uses its subroutine, the CDHKEEP routine. The CDHKEEP routine sets the "non-functional" flag in the CDE to indicate that the program has been executed. Although this flag is meaningful only for nonreusable programs, it is always set at this point in the processing.

The CDHKEEP routine tests the use/responsibility count in the CDE to determine if there are other requests for the exiting program. (This test is necessary, since CDHKEEP can be invoked separately by other parts of the supervisor.) If the use/responsibility count is not zero, there is at least one outstanding request for the program, and CDHKEEP returns control to the Exit routine (or to CDHKEEP's caller). If, however, the use/responsibility count is zero, there is no outstanding request for the program. In this case, the routine tests the program's attributes. If the program is in the link pack area, control is immediately returned to the caller, since the program must not be purged. If the program is not in the link pack area and is either serially reusable or reenterable, the routine sets the "release" flag (CDATTR2 field) in the program's CDE and the "purge" flag[1] for the job pack queue. These flags will be tested by the GETMAIN routine (CDPURGE subroutine) to determine which program's space should be freed, if space is requested and is otherwise unavailable. If the program is neither serially reusable nor reenterable, or was fetched[2] by a job step that has invoked rollout, the CDHKEEP routine branches to another subroutine of CDEXIT, the CDDESTRY routine. The CDDESTRY routine frees the storage areas used by the program frees the storage areas used by the program and certain related control blocks.

The CDDESTRY routine uses the extent list for the exiting program to set up input parameters to be passed to the FREEMAIN routine. The extent list is a control block set up by routines of contents supervision; it contains the length of the module (program) and its starting address, or the length and address of each separately loaded control section of a module that was scatter loaded. After setting up the parameters, the CDDESTRY routine branches to the FREEMAIN routine, which then frees the storage space occupied by the exiting program.

When control returns from the FREEMAIN routine, the CDDESTRY routine branches to the ORDERCDQ routine. This routine locates the contents directory queue on which the CDE resides, searches for the CDE, and dequeues the major CDE and any minor CDEs that may have been created for the program. Such dequeuing is necessary so that the job pack queue of the contents directory reflects the freeing of the space occupied by the program. The ORDERCDQ routine returns control to the CDDESTRY routine, which again branches to the FREEMAIN routine to free the space occupied by the dequeued CDEs and their associated extent list. After this operation has been performed, the CDDESTRY routine returns control to the Exit routine (or to CDDESTRY's caller).

If the Returning Program Is a User Exit Routine

If the exiting program was controlled by an SIRB, special processing is not required; control passes to the IRB-handling portion of the Exit routine, then right back to a user program. If the

[1] The "purge" flag is the high order bit of the TCBJPQ field of the current TCB.

[2] If the program was fetched via the LOAD macro instruction, the CDHKEEP routine returns control to the caller, and does not branch to the CDDESTRY routine to purge the program.

program was controlled by an IRB, special processing is required.

The Exit routine checks whether the use count in the IRB is zero. The use count may indicate that the parent task has requested multiple use of the same end-of-task exit routine (ETXR) for different subtasks. If the use count is not zero, indicating an additional need for the exiting user routine, the Exit routine branches to the Transient Area Refresh routine. But if the use count is zero, indicating that the IRB is no longer needed, the Exit routine tests whether there is a register save area that the requestor of the user Exit routine had originally reserved, that may now be freed for reuse. If there is such an area, which is indicated by a nonzero RBPPSAV field in the IRB, the Exit routine branches to the FREEMAIN routine to free it. When return is made from the FREEMAIN routine, the area occupied by the RB is freed.

Common Processing

Regardless of the type of special processing required (depending on RB type), control always returns to the same point in the Exit routine. This return point (EDTNX) is the address of a test. The test determines if the exiting program is under the control of the last RB on its RB queue. If it is, the Exit routine removes the current TCB from the TCB queue, since it is no longer needed. In addition, the Exit routine sets the "normal termination" flag (TCBFE) in the TCB as an indicator to the Detach routine. The purpose of this indicator is to avoid an incorrect branch to the ABTERM routine when the subtask is eventually detached.

The Exit routine next tests if the next RB on the RB queue is in a wait condition. If this RB is in wait condition -- indicated by a nonzero RBWCF field -- the Exit routine tests if a task switch has been indicated. The test is a comparison of the two words of the TCB pointer (IEATCBP and IEATCBP+4). If these two words are equal, the need for a task switch has not already been indicated by another routine, and the Exit routine sets the first word to zero. This later indicates to the Dispatcher that it should search down the TCB queue for a TCB representing the next highest priority ready task.

The Exit routine then flags the RB for the exiting program "inactive" and removes the RB from the RB queue. Inactive status means that the RB is not queued from a TCB. The RB is removed so that its program will not be mistakenly rescheduled for execution.

The Exit routine next determines whether to free the storage space occupied by the RB of the exiting program. If the RB was dynamically acquired, the space may be freed. This condition is indicated in the RBSTAB field of the RB. If the RB is a permanent system request block, such as an SIRB, its space may not be freed. If the space may not be freed, the Exit routine branches directly to the Transient Area Refresh routine; otherwise, it first frees the space.

THE TRANSIENT AREA REFRESH ROUTINE

The Transient Area (TA) Refresh routine is contained in the Transient Area Handler module at entry point IEAQTR02). It determines if it is necessary to reload an overlaid SVC routine in a transient area. If reloading (refreshing) is necessary, the routine initiates a task switch to the appropriate transient area fetch task to reload the needed routine.

The TA Refresh routine first checks if there are any "user" SVRBs for the transient areas by checking the user count for the transient area. If the count is zero, the TA Refresh routine branches to the Dispatcher, since there are no users for any transient area. If the count is not zero, the TA Refresh routine searches the user queue associated with each entry of the transient area control table (TACT). The routine searches for indication of a routine that needs to be refreshed (see Figure 9-1).

If a flag in the TACT entry indicates that the associated transient area is in process of being loaded, the user queue for that TACT entry is not searched. Otherwise, the queue is searched for the highest priority "ready user" SVRB. A user SVRB is an SVRB that was created when the associated SVC routine was requested. It is ready if it is the top RB on its RB queue and its TCB is not set nondispatchable. If a ready user SVRB is found, the TA Refresh routine checks if the associated routine is already in the transient area. If the TTR field in the SVRB is the same as that in the TACT entry, the routine is in the transient area. If the routine is not in the area, the TA Refresh routine prepares to overlay the routine that is currently in the area.

The TA Refresh routine saves the RB wait count of the current RB and sets a new wait count of 'FF' (decimal 255) in each user SVRB. The routine readies the TA Fetch TCB pointed to by the TACT entry. It then branches to the Task Switching routine to prepare for a task switch to the TA Fetch task by the Dispatcher. The TA Fetch TCB controls the TA Fetch routine. (See "Loading the Routine" in "Fetching a Nonresident Routine from Auxiliary Storage" in Section 2.)

The TA Refresh routine then tests the next TACT entry.

If no ready user SVRB is found tor a transient area, either the transient area is free or all user SVRBs are waiting. The TA Refresh routine indicates that deferred requests can be removed from the request queue, and then checks the next TACT entry.

When all TACT entries have been checked, the TA Refresh routine tests whether it has indicated that deferred requests can be removed. If they cannot, the routine branches to the Dispatcher. If they can, the routine removes all SVRBs on the request queue, clears the wait count field in each SVRB, and invokes the Task Switching routine to determine if the associated task is of higher priority than the current task. If the selected task is of higher priority, the Task Switching routine indicates to the Dispatcher the need for a task switch, by placing in the "new" TCB pointer the address of the selected TCB. The TA Refresh routine then branches to the Dispatcher.

DISPATCHING (PERFORMING THE ACTUAL RETURN OF CONTROL)

The Dispatcher is entered via a branch at the end of most interruption processing sequences. It receives control from any of the following supervisor routines, depending on the type of routine that is returning control and/or the type of processing that should next be performed:

- Type-1 Exit routine, when a type-1 SVC routine has been completed and the need for a task switch has been indicated.
- Exit routine, when a user program-check routine has been completed.
- Transient Area Refresh routine, when the return is from any routine except a type-1 SVC routine, a user program-check routine, or the I/O Supervisor.
- I/O First-Level Interruption Handler, when the return is from the I/O Supervisor.
- External First-Level Interruption Handler, when an external interruption has been serviced.
- Program Check First-Level Interruption Handler when the multiprocessing feature has been selected.
- SVC Second-Level Interruption Handler, when a transient area fetch task is to be given control to load a transient SVC routine.
- Transient Area Fetch routine, when a transient SVC routine has been loaded and no error has been detected by the Program Fetch routine.
- ABEND3, when it has selected another terminating task whose resources are to be purged.

In a multiprocessing system, the first operation of the Dispatcher is a test for external interruptions that have occurred during program check or I/O FLIH routines and have not been processed. If there are any (FLRETFLG is not equal to zero), control is passed to the External FLIH routine.

The main function of the Dispatcher is to determine the next task whose current routine is to be given control, and to pass control to that routine.

Other functions of the Dispatcher are:

- Completing the scheduling of user (asynchronous) exit routines.
- Handling task and job step timing.
- Recognizing that a priority level is time-sliced, determining which task within the group to dispatch, and dispatching the task for the maximum time interval (if time-slicing is included in the system).

DETERMINING AND GIVING CONTROL TO THE CURRENT ROUTINE OF THE TASK NEXT TO BE DISPATCHED

The Dispatcher decides the task next to be dispatched and passes control to the current routine of that task. The task next to be dispatched is one of the following:

- The current task, whose performance is being resumed.
- Another ready task of higher priority than the current task.
- Another ready task of lower priority than the current task, if the current task is waiting or is nondispatchable.
- Another task in the same time-sliced group (if time slicing is included in the system).

The interrupted routine of the current task is given control if no supervisor routine has indicated the need for a task switch. If, however, a task switch has been indicated, the Dispatcher gives control to the current routine of the highest priority ready task. This task may be of higher or lower priority than the current task. The address of the "new" task's TCB is found either in the "new" TCB pointer (IEATCBP), or through a search of the TCB queue.

If the Dispatcher does not find a ready TCB whose current routine it may dispatch, it dispatches a special pseudo, or dummy, task which is part of the nucleus. The pseudo task has no associated routines, and places the CPU in an enabled wait state. After a future interruption, one of the nonready tasks may be readied by an interruption handler, and CPU execution can continue.

The preceding paragraphs have provided a general discussion of the Dispatcher's main function. The following text is a detailed explanation of the same information. Dispatcher processing in a system without the time-slicing feature will be discussed first, followed by a description of the differences when the feature is present.

Normal Dispatcher Processing (Without Time-Slicing)

The Dispatcher determines which task should be performed next: the current task or another ready task. It does this by comparing the contents of the "old" and "new" TCB pointers, IEATCBP+4 and IEATCBP. These locations are obtained via a pointer in the communications vector table, called CVTTCBP. They contain the addresses of the current ("old") TCB and the "new" TCB for the task next to be dispatched.

If the two TCB pointers are equal, no supervisor routine has indicated the need for a task switch since the Dispatcher was last executed. The Dispatcher restores registers from the save area of the current TCB, and returns control to the interrupted routine by loading the RB old PSW from the routine's RB.

If the two TCB pointers are not equal, a task switch is required. The Dispatcher saves the floating point register contents in the floating point register save area of the current TCB. The general register contents were previously saved in the current TCB by one of the following routines, depending on the linkage path to the Dispatcher:

Type-1 Exit Routine
Exit routine
Transient Area Exit routine
SVC Second-Level Interruption Handler
External First-Level Interruption Handler
I/O First-Level Interruption Handler
ABEND3

The Dispatcher then determines the next task whose current routine it will give control.

If the two TCB pointers are not equal and the "new" TCB pointer (IEATCBP) does not contain zero, it points to the "new" TCB whose current routine will be given control. This condition is usually the result of recognition by the Task Switching routine that a task higher in priority than the current task is ready. The Dispatcher restores registers, both general and floating point, from the "new" TCB. It then gives control to the new task's current routine by loading the RB old PSW from the task's current RB. (The TCBRBP field of the TCB points to the task's current RB.) In a multiprocessing system, if the two TCB pointers are not equal, and the "new" TCB pointer does not contain zero, the Dispatcher searches down the TCB queue. The search begins with the TCB whose address is in the "new" TCB pointer of the executing CPU. The address of the next highest priority ready task is placed in the "new" TCB pointer of the second CPU.

If the two TCB pointers are unequal and the "new" TCB pointer contains zero, then the current task has been placed in a wait condition. In this case, the Dispatcher must determine the next highest priority ready task. The Dispatcher searches down the TCB queue, starting from the current TCB. It locates each successive TCB through the TCB link field (TCBTCB). The current routine associated with the first TCB that meets the following conditions is given control, via a Load PSW instruction:

1. The TCB's current RB must not be in wait condition (i.e., the RBWCF field must contain zero).

2. The nondispatchability flags in the TCB must not be set (see Table 9-2).

In either of the two cases in which the two TCB pointers are not equal, the Dispatcher sets both pointers equal to the address of the "new" TCB. Thus, for future processing the TCB pointers no longer indicate the need for a task switch.

In a multiprocessing system, if the two TCB pointers are unequal and the "new" TCB pointer for the executing TCB contains zero, the Dispatcher searches down the TCB queue to determine the two highest priority ready tasks. The search begins from the top of the queue when the "new" TCB pointer of the second CPU also contains zero; otherwise, the Relative Priority routine determines whether the current TCB on the executing CPU, or the TCB whose address is in the "new" TCB pointer of the second CPU, is higher on the TCB queue, and the search begins from the higher TCB. The highest priority ready task that is not the current task on the second CPU becomes the new TCB on the executing CPU. If the highest priority ready task is not the current TCB on the second CPU and the "new" TCB pointer for that CPU is not set, the search continues down the TCB queue for the next ready TCB. The address of this TCB is placed in the "new" TCB pointer of the second CPU.

If the Dispatcher in its search of the TCB queue finds no ready task, it selects a special TCB that represents a pseudo task. The Dispatcher then loads the RB old PSW from the permanent RB that is part of the pseudo TCB. This RB old PSW, when loaded, places the CPU in an enabled wait state. After a future interruption, one of the nonready tasks may be made ready by an interruption handler, and CPU processing can continue.

In a multiprocessing system, if the two TCB pointers of the second CPU are not equal (after the TCB queue has been searched) control is given to the SHOLDTAP routine which interrupts the second CPU with an indication (in STMASK) that the Dispatcher routine must gain control. Before dispatching the next task on the executing CPU, the old PSW is examined, and, if it is completely enabled, zeros are placed in the supervisor lock and CPU identity bytes. An enabled old PSW indicates that the supervisor lock byte was not set by the task that is to be dispatched, and therefore the lock byte is cleared before this task receives control.

Dispatcher Processing with Time-Slicing (Differences)

When the "new" and "old" TCB pointers are equal, the Dispatcher tests whether "old" represents a time-sliced task. If it does not, normal dispatcher processing continues. If it does the Dispatcher tests whether the time-slice interval has expired; it has expired if the time-slice TQE is off the timer queue. When this is the case, a task switch (to the next ready TCB in the time-slice group) is indicated, and the Dispatcher sets "new" to zero to force the task switch. If the interval has not expired, special processing is not required.

When the "new" TCB pointer, contains zero, it indicates the current task has been forced to wait and no higher-priority task is dispatchable. The Dispatcher again must test "old" for time-slicing; if it represents a time-sliced task, the next ready task in the time-slice group should be dispatched.

When "new" contains an address not equal to the TCB address in "old," it indicates (1) a higher-priority task has become ready to be dispatched, or (2) another task in the same time-slice group has become ready. The Dispatcher tests to determine the case. If the task represented by "new" is in the same time-slice group as the one represented by "old", the Dispatcher ignores therequested task switch; the new task must wait its turn.

When the next task to be dispatched is a time-sliced task (whether or not it is in the same time-slice group as the previous task), the Dispatcher updates the TSCE pointers for the new task's group. The Dispatcher finds the next TCB in the time-slice group on the TCB queue and places its address in the Next field of the TSCE. It also enqueues the time-slice TQE.

COMPLETING THE SCHEDULING OF USER EXIT ROUTINES

A minor function of the Dispatcher is to ensure that user (asynchronous) exit routines, partially scheduled by the Stage 2 Exit Effector, are completely scheduled. The Dispatcher tests the stage 3 switch (IEA0DS01) to determine whether there is at least one queue element (interruption queue element or request queue element) on a user (asynchronous) exit queue. (The switch is set by the Stage 2 Exit Effector when it places a queue element on either of the exit queues.) If the stage 3 switch is set, the Dispatcher branches to its subroutine, the Stage 3 Exit Effector (IEA0EF03), to complete the scheduling of the user exit routine(s). (See "Scheduling a User Exit Routine" in Section 3, Task Supervision.)

HANDLING TASK AND JOB STEP TIMING

If a task switch is to occur, the Dispatcher updates the timer queue, and if necessary, the timer itself. The purpose is to alter the timing of task intervals because a different task is about to control the CPU. The processing is different for the two types of timing handled by the Dispatcher, task timing and job step timing.

Task timing is requested by a routine of a task, via an STIMER macro instruction that specifies the TASK operand. If a task switch is needed, the Dispatcher tests whether the current task has an unexpired task-type[1] interval. If it has, the Dispatcher stops the timing of the current ("old") task's interval. If the ("new") task to be dispatched has requested the timing of a task-type interval, the Dispatcher restarts the timing of the "new" task's interval.

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 two conditions are met: (1) a task switch is needed, and the (2) job step timing option was specified during system generation. If these conditions are met, the Dispatcher suspends timing of the job step whose task has given up control and restarts timing of the job step whose task is next to be dispatched.

The handling of task and job step timing, just described in general, will now be discussed in greater detail.

Handling Task Timing

If a task switch is needed, the Dispatcher performs task timing. The need for a task switch is indicated by the inequality of the two TCB pointers, IEATCBP and IEATCBP+4. (The address of these pointers is in the CVTTCBP field of the communications vector table.)

If the task that is relinquishing control (the "old" or current task) requested task timing, the Dispatcher branches to the Timer Second-Level Interruption Handler (entry point IEAQTD01) to stop the timing of the requested interval. The "old" task requested task timing if it has a timer queue element (TQE) and if a task-type request is indicated in its TQE. The task has a TQE if the TCBTME field of its TCB does not contain zero.[2]

The Timer Second-Level Interruption Handler (see Chart EE) tests whether the "old" task's TQE is on the timer queue. If the TQE is not on the queue, the "old" task's interval is not being timed, and the Timer Second-Level Interruption Handler (hereafter called the Timer SLIH) returns control to the Dispatcher. If, however, the "old" task's TQE is on the timer queue, an interval is being timed for this task. In this case, the Timer SLIH determines the absolute time remaining in the requested interval, stores this time in the TQE for future use, and removes the TQE from the timer queue. If the removed TQE was at the top of the timer queue, the Timer SLIH updates the interval timer. It places the time of expiration (TOX) value of the new top TQE in both the interval timer and the six-hour pseudo clock (see Charts EE and ED). The Timer SLIH then returns control to the Dispatcher.

If the "new" task to be given control requested interval timing, the Dispatcher branches to the Timer SLIH (entry point IEAQTE00) to restart timing of the interval. The "new" task requested interval timing if it has a TQE, as indicated by a nonzero TCBTME field in its TCB. The Timer SLIH tests whether the TQE for the "new" task is on the timer queue. (The TQEFLGS field of the TQE indicates if the TQE is on the timer queue.) If it is, the requested time interval is already being timed. In this case, the Timer SLIH immediately returns control to the Dispatcher. If, however, the "new" task's TQE is not on the timer queue, processing is needed to restart the timing of the requested interval (see Chart EE).

The Timer SLIH computes a new time of expiration (TOX) for the requested interval and places the recomputed TOX value in the "new" task's TQE. (See Section 6, "Timer Supervision" for information on the computation of the TOX.) If the recomputed TOX value is smaller than the current value in the interval timer, the Timer SLIH places the new value in the timer. It then places the TQE on the timer queue in the relative position that is appropriate for the new TOX value. (TQEs are ordered on the queue according to their relative times of expiration.) When the TQE is on the timer queue, timing of the "new" task's requested interval is resumed. The Timer SLIH then returns control to the Dispatcher.

Handling Job Step Timing

The Dispatcher performs the following main functions for job step timing, if the need for a task switch is indicated, and if the job step timing option was specified during system generation:

----

[1] The type of interval request is indicated in the TQEFLGS field of the task's timer queue element.

[2] The TCBTME field is set by the STIMER routine when it services a "set timer" request. It contains zero in any of the following cases: no STIMER macro instruction has been issued for this task; or the TTIMER routine has serviced a TTIMER macro instruction for this task that specifies the CANCEL operand; or the task has been terminated, normally or abnormally. (See Charts EC and GB in Section 11.)

- Removes from the timer queue the TQE for the initiator of the job step associated with the "old" task if the TQE is TASK type.

- Places on the timer queue the TQE for the initiator of the job step associated with the "new" task to be dispatched if the TQE is TASK type. If the TQE is REAL and off the timer queue, it must be converted to TASK type and placed on the timer queue. If the TQE is REAL and on the timer queue, it must be removed from the queue, converted to a TASK TQE and placed on the queue.

REMOVING FROM THE TIMER QUEUE THE TQE FOR THE INITIATOR OF THE JOB STEP ASSOCIATED WITH THE TASK WHICH HAS GIVEN UP CONTROL: The Dispatcher must determine whether the "old" task was the dummy (pseudo) task. For the meaning of the dummy (pseudo) task, see "Normal Dispatcher Processing (Without Time Slicing"). It does this by comparing the "old" TCB address to the RB pointer (TCBRBP) in the "old" TCB. If they are equal the dummy task had previously been dispatched, and there is no TQE to be removed from the timer queue. If the "old" TCB was not the dummy task, the Dispatcher finds the address of the TCB for the initiator of the job step whose task has just given up control. It finds this initiator TCB by following the TCB pointers illustrated in Figure 9-2. The Dispatcher then determines if the step requested timing by testing for the presence of a TQE pointer (TCBTME) in the initiator TCB. If the field is zero, the user has specified that job step timing is not to be applied to this job and there is no TQE. If there is a TQE, the Dispatcher tests it for non-expired TASK type TQE. If the TQE is REAL, it should not be removed from the timer queue because it represents a 30-minute interval enqueued by WAIT and dequeued by POST. When the Dispatcher finds an unexpired TASK type TQE as the initiator's TQE, it branches to the Timer Second-Level Interruption Handler (entry point IEAQTD01) to suspend job step timing for the "old" task.

In a multiprocessing system, when two tasks of the same job step are running concurrently, the time to expiration value of the job is halved. Therefore, when job step timing is suspended for a task, the Dispatcher must determine whether the task on the second CPU belongs to the same job step. If so, the Dispatcher must double the time to expiration value of the job step TQE to restore nonconcurrent timing. The Dispatcher branches to a subroutine (entry point DJS00) to obtain the address of the job step TQE for the task on the second CPU. If this is the same TQE scheduled for removal from the timer queue, because it is associated with the "old" task on the first CPU, the TQE is not removed, and the time to expiration value is doubled.

PLACING ON THE TIMER QUEUE THE TQE FOR THE INITIATOR OF THE JOB STEP ASSOCIATED WITH THE TASK TO BE DISPATCHED: The Dispatcher finds the address of the TCB for the initiator of the job step whose task is to be dispatched. It then determines whether job step timing was suppressed by testing the pointer to the TQE in the initiator TCB (TCBTME). If the field contains zero, no job step timing will be done. If the field is non-zero, the Dispatcher examines the TQE type for a non-expired TASK TQE. If the TQE is this type, the Dispatcher branches to the Timer SLIH (entry point IEAQTE00) to restart job step timing for the task it is about to dispatch. If the TQE is REAL, it indicates that a user's asynchronous exit is to be given control and it should be job step timed. Therefore, the Dispatcher branches to the Timer SLIH (entry point IEAQTD01) to remove the element from the timer queue. Next, the dispatcher moves the job step time remaining value from the saved field to the TQEVAL field, and changes the TQE type from REAL to TASK by setting to an off position the two low-order bits (bits 6 and 7) in the flag byte in the TQE (TQEFLGS). The Dispatcher then branches to the Timer SLIH (entry point IEAQTE00) to restart job step timing for the task it is about to dispatch.

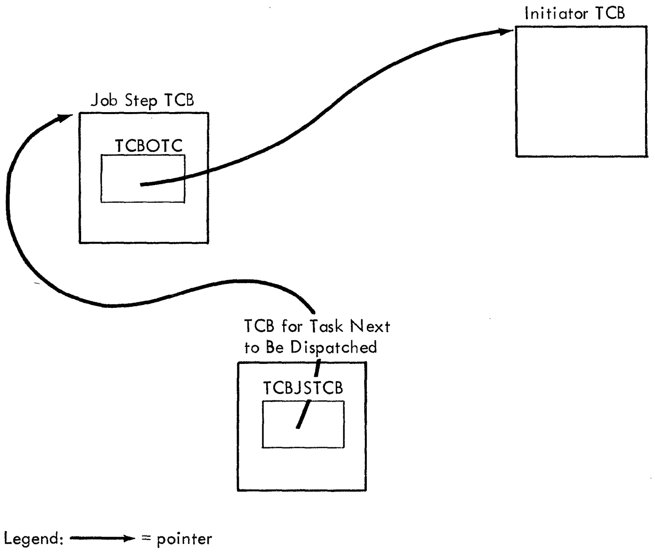


Figure 9-2. Locating the Initiator TCB Associated with the Task Next to be Dispatched

In a multiprocessing system, the Dispatcher branches to a subroutine (entry point DJS00) to obtain the address of the job step TQE for the task on the second CPU. If this is the TQE scheduled for placement on the timer queue, because the task to be dispatched on this CPU belongs to the same job step as the task on the second CPU, the time to expiration value of the TQE is halved. In this way, the execution time of the job step is the same as if two tasks were not running concurrently.

Termination procedures free the resources and control blocks belonging to the terminating task. The freed resources include exclusively used programs in main storage, enqueued resource requests, unexpired timer requests, incomplete operator communications, exclusively used data sets, and unshared subpools of main storage. The control blocks that are removed from their queues and freed include one or more:

- Task control blocks (TCBs).
- Request blocks (RBs).
- Interruption queue elements (IQEs).
- Queue elements (QELs).
- Queue control blocks (QCBs).
- Subpool queue elements (SPQEs).
- Contents directory elements (CDEs).
- Timer queue elements (TQEs).
- The Program Interruption Element (PIE) for the task, if one exists.

There are two types of termination procedures, normal and abnormal. Normal termination occurs when a task is complete; that is, when the last program to be executed for the task has completed its execution. Abnormal termination occurs when some type of unrecoverable error, such as a machine check, I/O error, or program check, has taken place. The task must be terminated to prevent waste of system resources.

Normal and abnormal termination differ in their scope of action. Normal termination frees resources only for the completed task, not for its subtasks or higher level tasks. Abnormal termination allows two options, task and step termination. In task termination the resources of only the malfunctioning task and its incomplete subtasks are freed. This option permits a program belonging to a higher level task in the job step to decide whether to continue the job step. But in step termination the resources used for the entire job step are freed, and the job scheduler ignores later steps of the same job. A task termination of the job step task, the highest level task in the job step, produces the same result as a step termination.

NORMAL TERMINATION (EOT ROUTINE)

Normal task termination is performed by the End-of-Task (EOT) routine, which receives control from the Exit routine upon its detecting an end-of-task condition. The EOT routine is strictly an internal supervisor routine; that is, it does not receive control directly via an SVC. It frees the previously mentioned resources and their control blocks. If an event control block (ECB) had been specified when the terminating task was attached, the EOT routine posts the ECB with a completion code for examination by a program belonging to the parent task. To allow other programs to continue execution, the EOT routine modifies the TCB pointer to ensure a task switch, and then branches to the Dispatcher to return control to the current routine of the highest priority ready task.

The EOT routine releases resources no longer needed when a task is completed. Its functions include:

- Purging the operator communication queues.
- Closing data sets opened for the completed task.
- Releasing unexpired timer elements.
- Releasing the program interruption element (PIE), if one exists.
- Freeing storage acquired for this task.
- Releasing programs loaded for the task.
- Removing the task's deferred rollout requests (if any) from the rollout request queue.
- Dequeuing the TCB for the task from the TCB queue and (conditionally) from the subtask queue and freeing its space.
- Ensuring that the need for a task switch has been indicated.

After performing these functions, the EOT routine returns control to the Exit routine to free the RB for the last executed program of the task. Then, via the Dispatcher, control is given to the current program of the highest priority ready task.

The EOT routine receives control, via a branch, from the Exit routine when it detects an end-of-task condition. The Exit routine recognizes that the PRB for an exiting user program points to its TCB instead of to another RB. (The RBTCBNXT status bit in the PRB, when set, indicates that the RBLINK field points to a TCB.)

The first step of EOT processing is to check whether there are any subtasks of the completed task that have not been detached. All subtasks should have been previously removed for the completed task. If there is at least one subtask that has not been detached (as indicated in the TCB by the subtask pointer TCBLTC), the EOT routine sets up an error code (hexadecimal 80A03000). It then issues an ABEND macro instruction to produce supervisor linkage to the ABEND routine in order to abnormally terminate the completed task.

If there are no remaining subtasks, the EOT routine stores in the task's TCB the completion code that will be provided to its parent task in the return code register. The parent task will examine the completion code to determine the status of its subtask. (The status of the subtask is examined by the parent task only if the subtask was attached with either the ECB or the ETXR operand specified.)

After storing the completion code, the EOT routine tests whether a program interruption element (PIE) exists and should be freed. If a PIE exists, its address appears in the TCBPIE field of the TCB, placed there earlier when the SPIE routine created the program interruption element. If the PIE exists, the EOT routine makes its space available for reuse by branching to the FREEMAIN SVC routine to release the space.

After freeing the PIE, or if no PIE existed for the task, the EOT routine branches to the Purge Timer subroutine. The subroutine's purpose is to test for and remove any remaining timer queue elements. Such an element represents a request for a timer interval that has not yet expired. If a timer element exists (queued from the TCBTME field of the TCB), the subroutine cancels the timer request and frees, via the FREEMAIN routine, the space occupied by the timer queue element (TQE) and any associated problem-program register save area.

The EOT routine next tests for any serially reusable resources that were enqueued and not later dequeued. If there is such a resource, the "enqueue" count (TCBQEL) in the TCB is not zero. (The enqueue count in the TCB is increased by the ENQ routine and decreased by the DEQ routine. The count is stored in the high-order byte of the TCBFSA field.) If the enqueue count indicates that a resource was not dequeued, the EOT routine sets up an error code (hexadecimal 80D03000), and issues an ABEND macro instruction to abnormally terminate the task.

Next, a branch is made to the "WTOR purge" routine (IEECVPRG). The routine removes from the buffer queue and the reply queue those elements that are associated with the completed task. The elements represent messages to the operator and the operator's replies. The "WTOR Purge" routine issues a "voiding" message to inform the operator to cancel outstanding replies.

To ensure that all data sets used for the task have been closed, the EOT routine next branches to the "close data sets" subroutine. This subroutine checks the TCBDEB field of the TCB. If the field is not zero, it contains the address of a data extent block, or DEB. The subroutine uses the DEB to obtain the address of a data control block, or DCB, which it supplies as an input parameter to the Close routine of data management. The subroutine then issues a CLOSE macro instruction to gain supervisor linkage to the Close routine. As part of its processing, the Close routine updates the DEB address in the TCBDEB field. The "close data sets" subroutine repeats the CLOSE macro instruction for each DEB on the queue. When the DEB chain has been exhausted, all data sets for the task have been closed.

After each execution of the Close routine, the "close data sets" subroutine checks for an error that might have occurred during execution of the Close routine. It does this by noting whether the TCBDEB field has been updated. If the field has not been updated, the subroutine recognizes that incorrect DEB information has been supplied. The subroutine sets up an error code (hexadecimal 80C03000) and issues an ABEND macro instruction to abnormally terminate the task.

If there is no error detected during the closing of data sets, the EOT routine branches to the CDEXIT subroutine. The CDEXIT subroutine either frees the task's last executed program, or schedules the program's execution for a waiting requestor. (For a detailed discussion, see "If the Returning Routine Is a User Program" in Section 9, "Exiting Procedures.")

The EOT routine next releases modules that were loaded for the task (via the LOAD macro instruction) and are no longer needed for other tasks. It does this by branching to the "release loaded programs" subroutine (IEAQABL).

This subroutine releases modules that were loaded for the task, via a LOAD macro instruction, but which were not released via a DELETE macro instruction.

To determine the number of outstanding requests for each module, the "release loaded programs" subroutine examines, in turn, each load list element in the task's load list. Each load list element represents a module that was loaded for the task, via a LOAD macro instruction. (The list origin of the load list is the TCBLLS field of the TCB.) To determine the number of outstanding requests for the module, the subroutine subtracts the responsibility count from the use/responsibility count. The responsibility count in the module's load list element records the number of load requests for the module. The use/responsibility count in the module's contents directory entry records the total number of requests for the module. (Each load list element points to an associated contents directory entry.)

The "release loaded programs" subroutine then branches to subroutine CDHKEEP to test the number of outstanding requests for the module. If there is a least one outstanding request for the module, CDHKEEP immediately returns control to the "release loaded programs" subroutine. If, however, there are no outstanding requests for the module, CDHKEEP either frees the module and its control blocks, or sets flags to inform Main Storage Supervision that space may be purged, depending on the attributes of the module. (For further details, see "If the Returning Routine Is a User Program" in Section 9, "Exiting Procedures.")

On return from the CDHKEEP subroutine, the "release loaded programs" subroutine frees the load list element for the module just tested and perhaps freed. The process is repeated until all the load list elements, and possibly their associated modules, have been freed.

The EOT routine next branches to the "release main storage" subroutine (IEAQSPET) to release space that was obtained for the task via a macro instruction. This subroutine performs an additional function if the completed task is the job step task (the highest level task in the job step). The subroutine ensures that programs remaining in the job pack area are freed. Such programs are reentrant or serially reusable programs that were used during the execution of the job step. Their release was previously invoked, but since they were still needed for other tasks of the job step, their storage space was not freed.

For any terminating task, the "release main storage" subroutine frees unshared subpools of main storage allocated to the task. The subpools are represented by subpool queue elements (SPQEs), which have their list origin in the TCBMSS field of the TCB. The subroutine examines each SPQE on the main storage queue. If an SPQE represents a subpool not shared with another task, the subpool and the SPQE are freed, via a branch to the FREEMAIN SVC routine. The main storage queue is updated, and the next element is examined. If, however, an SPQE represents a shared subpool, that subpool cannot be freed. The "release main storage" subroutine updates the queue to indicate that the subpool is no longer shared. When all elements have been examined, subpool 253 (supervisor queue area) is explicitly freed, since there is no SPQE for this subpool. As a minor additional function, the subroutine frees space occupied by a parameter list created during the execution of the "close data sets" subroutine.

If the completed task is the job step task, any remaining modules in the job pack area must be freed. A check is made of the job pack area queue (whose list origin is the TCBJPQ field) to discover if there is at least one contents directory entry (CDE) on the queue. If there is at least one CDE, the "release main storage" subroutine branches to entry point CDDESTRY in the CDEXIT routine to free remaining modules, CDEs, and extent lists. (For further information, see "If the Returning Routine Is a User Program" in Section 9, "Exiting Procedures.")

After freeing unshared subpools of main storage, the EOT routine initiates the scheduling of an end-of-task exit routine (ETXR), if one had been originally requested by the ETXR operand when the task was attached. If the use of the ETXR routine had been requested, the Attach routine would have created an interruption request block (IRB) and an interruption queue element (IQE). The IRB provides future control of the ETXR routine and aids in its scheduling, while the IQE represents the queued request. In addition, the Attach routine would have placed the address of the IQE in the newly created TCB, and set the TCBFETXR flag in the TCBFLGS field to indicate the presence of the ETXR request. Now, during end-of-task processing, the EOT routine checks the TCBFETXR flag to learn whether the use of an ETXR routine had been requested when the task was attached. If the flag is set, the EOT routine initiates scheduling of the ETXR by passing the address of the IQE to the Stage 2 Exit Effector. (See "Scheduling User Exit Routines" in Section 3, "Task Supervision.") The Stage 2 Exit Effector places the IQE representing the ETXR request on a queue of requests for user exit routines. Later, during the execution of the Dispatcher, the Stage 3 Exit Effector will complete the EXTR scheduling. It will place the IQE on a queue of IQEs belonging to the IRB, and place the IRB as the "current" RB on the RB queue of the attaching task. The ETXR routine is thus scheduled as the next program to be executed for the parent of the terminating task.

Any deferred rollout requests (IQEs) belonging to the terminating task are next removed from the rollout request queue. (This queue's address is IEAROQUE in the rollout/rollin module, IEAQRORI.) The rollout request queue represents rollout requests that could not be serviced either because another rollout was in effect for a different requesting job step, or because a job step suitable for rollout could not be found.

The EOT routine, via its "dequeue TCB" subroutine, removes the TCB of the terminating task from the TCB queue. Since the current task is now terminated, its TCB must be removed from consideration by the Dispatcher.

If the time-slicing feature is included in the system, the EOT routine tests the time-slice bit (TCBFTS) in the TCB. If it is not set, normal EOT processing continues. If it is set, indicating that the terminating task is a member of a time-sliced group, the EOT routine locates the TSCE for the group. The address fields (First, Last, and Next) in the TSCE are compared to the address of the terminating TCB.

- If none of the address fields match the TCB, the EOT routine turns off the time-slice bit and normal EOT processing continues.

- If all of the address fields match the TCB, the EOT routine places zeroes in them to indicate that the time-sliced group is without members. Normal EOT processing then continues.

- If the First field matches, the EOT routine places the address of the next lower TCB on the TCB queue in First.

- If the Next field matches and Last does not, the EOT routine places the address of the next lower TCB on the TCB queue in Next.

- If the Last field matches, the address of the next higher TCB on the TCB queue is placed in Last, and the address of the First TCB is placed in Next.

Normal EOT processing continues after each case.

The EOT routine next sets two completion flags in the TCB: the "normal completion" flag (TCBFE) and the "nondispatchable completion" flag (TCBFC). The "normal completion" flag is of significance only during completion of the job step task. If the terminating task is the job step task, the "normal completion" flag indicates to an initiator of the Job Scheduler that the job step has been normally terminated. The "nondispatchable completion" flag is tested by the Detach SVC routine to determine whether to remove the subtask TCB from the TCB queue, or to abnormally terminate the subtask. If this flag is not set, the Detach routine assumes that the subtask to be detached is incomplete, and therefore schedules it for abnormal termination.

If the attaching routine of the parent task had specified an event control block (ECB), the EOT routine must now post the normal completion of the subtask for examination by a routine of the parent task. If no ECB was specified, posting is bypassed. For any terminating task except the job step task, the "EOT posting" subroutine checks for an ECB address in the TCBECB field of the current TCB. If an ECB address exists, the subroutine tests its validity by determining if the ECB contains a valid RB address. This is necessary, since the Post routine will not check the ECB address. The ECB resides in a user storage area and therefore is subject to alteration by a user program. If the job step task is being terminated, the validity of the ECB address is not checked, since this ECB resides in system-protected storage and cannot be altered by a user program. Validity checking, performed by a check subroutine, consists of a series of tests that reasonably ensure that the specified ECB address is valid and will not produce a program check during Post processing. The EOT routine branches to the Post SVC routine to place in the ECB of the parent task the completion code that was stored in the subtask TCB.

The EOT routine next determines whether to remove the TCB for the terminating task from its parent's subtask queue, and free the TCB's storage space. If neither an ECB nor an ETXR routine was specified when the task was attached, information in the subtask's TCB will not be needed by any program of the parent task. In this case, the "erase phase" subroutine removes the TCB from its parent task's subtask queue and frees its storage space. But if either an ETXR routine or an ECB was specified when the task was attached, a program belonging to its parent task may later examine information in the terminating task's TCB. In this case, the TCB and the pointers needed to gain access to it must be retained. The Detach SVC routine, later invoked for the parent task, will remove the TCB from its parent's subtask queue and free its space.

The EOT routine next ensures that the need for a task switch is indicated. The routine sets the "new" TCB pointer (IEATCBP) equal to zero, as an indication to the Dispatcher that it must search down the TCB queue to find the highest priority ready task. Control is returned to the Exit routine to free the space occupied by the last RB of the terminating task.

The Exit routine then branches to the Transient Area Refresh routine to "refresh" a transient area block that may have been overlaid by the terminating task. (See "The Transient Area Refresh Routine" in Section 9, "Exiting Procedures.") The Transient Area Refresh routine will branch to the Dispatcher to give control to the current routine of the highest priority ready task.

ABNORMAL TERMINATION

Abnormal termination is implemented primarily by three supervisor routines: the ABTERM routine, the ABEND routine, and the ABDUMP routine.

The ABTERM routine schedules the execution of the ABEND routine. It does this for system routines that detect an error but cannot themselves issue an ABEND macro instruction. The ABTERM routine ensures that, after redispatching, the first instruction to be executed for the defective task is an SVC 13 (ABEND) instruction. Thus, the ABTERM routine indirectly issues an ABEND macro instruction for the task specified for termination. (See Figure 10-1.)

The ABEND routine frees resources for the terminating task and its incomplete subtasks. The resources include programs, main storage, data sets, queued requests for serially reusable resources, and the control blocks that implement the allocation of these resources to the task.

The ABEND routine, if the terminating task is the job step task, frees the resources belonging to all tasks of the job step. The job step task is terminated in any of the following cases:

- The invoking ABEND macro instruction specifies the STEP option.

- The operator has issued a CANCEL command.

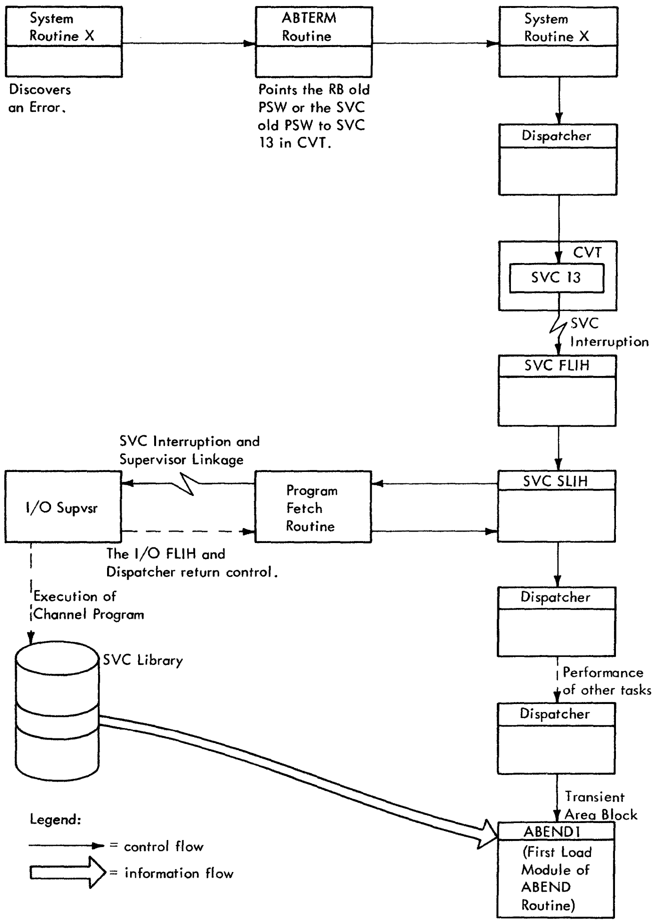


Figure 10-1. Scheduling of the ABEND Routine by the ABTERM Routine

- The job step timer interval has expired.

- The Machine-Check Handler for Model 65 (MCH/65)[1] is unable to recover from a machine check that occurs during the job step, but determines that the failure is not permanent.

The ABDUMP routine may be invoked by the ABEND routine as part of an abnormal termination, or it may be invoked at any time to perform a dynamic dump for a normal task. When invoked by the ABEND routine, the ABDUMP routine displays programs and control blocks belonging to the terminating task, and control blocks belonging to the task's descendants and direct ancestors. The ABDUMP routine is always invoked via a SNAP macro instruction.

----

[1] The Machine-Check Handler for Model 65 (MCH/65) is a system generation option available with System/360 Model 65.

SCHEDULING AN ABNORMAL TERMINATION (ABTERM)

The ABTERM routine is a disabled, serially reusable, resident non-SVC routine. It schedules the execution of the ABEND routine. It does this for the following types of callers:

- First-level interruption handlers.
- Type-1 SVC routines, which cannot issue an SVC instruction.
- System routines that must terminate a task other than the current task.
- The SER1 System Environment Recording routine or the Machine-Check Handler.
- The Program-Check First-Level Interruption Handler. Since it has special requirements, it cannot branch to the ABTERM routine directly, but must enter via a preliminary routine called the ABTERM Prologue routine. This routine performs housekeeping functions for the ABTERM routine.

In scheduling the execution of the ABEND routine, the ABTERM routine performs the following major functions:

- Interrogates flags to decide if the specified task should be scheduled for ABEND processing and/or if its subtasks should be set nondispatchable.
- Saves the address of the next executable instruction at the time of the last interruption (contained in either the SVC old PSW or in the RB old PSW of the current RB) for display by ABDUMP during ABEND processing.
- Stores the completion code and dump option in the TCB of the terminating task, for use by the ABEND routine.
- Schedules abnormal termination of the specified task by pointing either the RB old PSW of the current RB or the SVC old PSW to an SVC 13 (SVC ABEND) instruction, in the communication vector table. Conditionally indicates to the Dispatcher that a task switch to the scheduled task is needed.
- Sets nondispatchable incomplete subtasks of the terminating task, except for subtasks that are either being terminated or are in "must complete" status.
- In a multiprocessing system, determines, through a branch to the Task Removal routine, whether the current task on the second CPU has been set nondispatchable. If it has, the second CPU is interrupted with an indication (in STMASK) that the Dispatcher routine must gain control.
- Returns control to an address specified by the caller.

There are two entry points to the ABTERM routine: one (IEA0AB01) is for type-1 SVC routines, which need special processing; the other (IEA0AB00) is for all other system routines that wish to schedule an abnormal termination.

If entry is from a type-1 SVC routine, the ABTERM routine first obtains the TCB address of the current task, then adds the dump option flag to the completion (error) code that was passed by the SVC routine. The dump option flag specifies to the ABEND routine that it must invoke the ABDUMP routine, if possible, during ABEND processing. A branch is then made to the main entry point of the ABTERM routine (IEA0AB00).

The ABTERM routine, when entered at IEA0AB00, first saves the caller's register contents. Then it interrogates flags to determine if the specified task should be scheduled for ABEND processing, and/or if its subtasks should be set nondispatchable. Various combinations of ABTERM processing are possible, depending on the condition of the task specified for termination. The following discussion will describe each condition of the specified task and the resultant processing, as outlined in Table 10-1.

Processing if Specified Task Has Already Been Terminated

(See Table 10-1, condition 1.) In this case, the ABTERM routine does not schedule entry to the ABEND routine, nor does it attempt to set subtasks nondispatchable. Instead, the ABTERM routine simply restores the caller's register contents, and returns control to the routine whose address the caller had placed in the return register. A terminating task can be specified for abnormal termination if an operator's CANCEL command or the expiration of a job-step timer interval occurs concurrently with the execution of the EOT routine or the ABEND routine for the task.

Processing if the Task Has Already Been Scheduled for Abnormal Termination

(See Table 10-1, condition 2.) If the specified task has already been scheduled for abnormal termination but the ABEND routine has not yet been entered, the ABTERM routine does not reschedule ABEND processing for the task. It conditionally sets incomplete subtasks nondispatchable to

Table 10-1. ABTERM Processing

Conditions	Resultant Processing
1. Specified task[1] has already been terminated, normally or abnormally (TCBFC flag is set).	No processing beyond the restoring of the caller's register contents and return of control to an address specified by the caller.[2]
2. Specified task has already been scheduled for abnormal termination.	ABTERM conditionally sets the incomplete subtasks of the specified task nondispatchable.
3. Specified task is the job step task and is: a. Not already in the process of abnormal termination (TCBFA is not set).	Prepares for scheduling of the termination by clearing nondispatchability flags (except "must complete" nondispatchability) in the specified task's TCB. Stores parameters (dump option flag and completion code) in the TCB. Saves old PSW and wait count (if applicable). Schedules the task for entry to ABEND. Conditionally sets incomplete subtasks nondispatchable.
b. Already being abnormally terminated, and the Initiator is not the caller.	Schedules the task for entry to ABEND. Conditionally sets incomplete subtasks nondispatchable.
c. Already being abnormally terminated, and the Initiator is the caller and: (1) Dump option flag specifies a dump.	ABTERM assumes that a CANCEL command has occurred or job-step timer has expired, concurrently with ABEND execution. The processing is the same as in step 2.
(2) No dump is specified.	ABTERM assumes that a CANCEL command has been issued to stop a prolonged dump (possible infinite loop). Sets flags in the task's TCB to give the appearance of a first-time entry to ABEND. Remainder of processing is the same as in step 3a, except that parameters are not stored in the TCB and the old PSW and wait count are not saved during scheduling of the termination.
4. Specified task is not the job step task and: a. Specified task was previously set nondispatchable by ABTERM or ABEND (TCBABWF is "set").	Same processing as in step 2.
b. Specified task is not in the process of termination by ABEND.	Same processing as in step 3a, except that nondispatchability flags, if previously set, are not cleared.
c. Specified task is in the process of termination by ABEND.	Same processing as in step 3b.

[1] The "specified" task is the one whose TCB address is passed by the caller to ABTERM.
[2] All processing options include the processing performed under condition 1.

prevent their competing for system resources for the terminating parent task. This condition, wherein the task has been scheduled for abnormal termination but has not yet been terminated, can readily occur. The Dispatcher can allow other tasks to be performed after ABTERM processing, before it dispatches the ABEND routine for the given task.

If the specified task has at least one subtask (TCBLTC is not equal to zero), the ABTERM routine branches to its SETSUBS subroutine to determine which subtasks should be set nondispatchable.

The SETSUBS subroutine uses its SCANTREE subroutine to find each TCB that represents a subtask or descendant (subtask of a subtask) of the specified task. (See Figure 10-2.) For each such TCB that the SCANTREE subroutine finds, the SETSUBS subroutine tests if the associated subtask or descendant should be set nondispatchable. The tests are repeated for each subtask or descendant in the "subtask tree."

A subtask or descendant is set nondispatchable if none of the following conditions exists:

- Subtask is complete (thus no need for setting the subtask nondispatchable).

- Subtask is in the process of abnormal termination (the ABEND routine is being executed for the subtask). In this case, nondispatchability would prevent the further execution of the ABEND routine for the subtask.

- Specified task is nondispatchable, but its subtask is dispatchable. This subtask may be in "must complete" status and should not be terminated or set nondispatchable. (For further discussion of the "must complete" status, refer to "Serializing the Use of a Resource" in Section 3, "Task Supervision.")

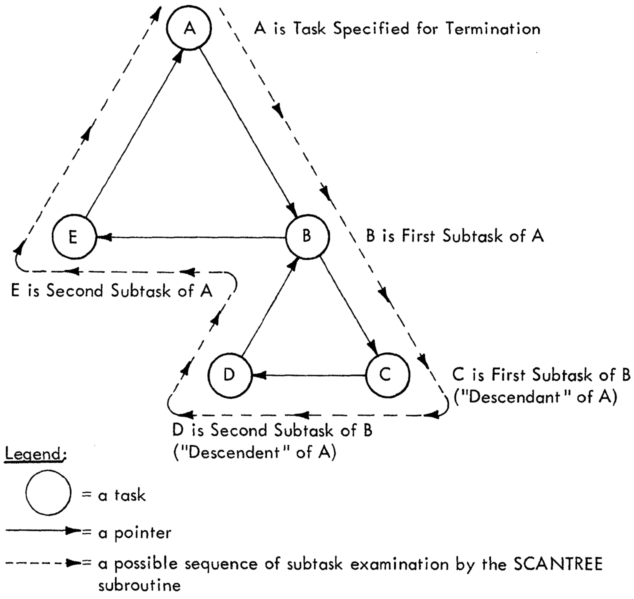

A
A is Task Specified for Termination
E
B
B is First Subtask of A
E is Second Subtask of A
D
C
C is First Subtask of B ("Descendant" of A)
D is Second Subtask of B ("Descendent" of A)
Legend:
= a task
= a pointer
= a possible sequence of subtask examination by the SCANTREE subroutine

Figure 10-2. A Tree of Subtasks and a Possible Sequence of Examination

The SETSUBS subroutine sets a subtask nondispatchable by setting the TCBABWF flag in the TCBFLGS field of the subtask's TCB. The subroutine also prevents the scheduling of asynchronous exits for the subtask. The Dispatcher will test the nondispatchability flags and will not dispatch any routine for the subtask, until the ABEND routine later clears the flags in preparation for terminating the subtask.

Processing if the Specified Task is the Job Step Task

(See Table 10-1, condition 3.) A job step task is a task attached by an Initiator of the job scheduler and is the highest level task within the family of tasks of a job step. The entry to the ABTERM routine may be the result of a direct branch from the Initiator because of either an operator's CANCEL command or the expiration of the job-step timer interval. Another possibility is that an error has occurred in an routine operating for the job step task. The type of ABTERM processing depends on the particular condition of the task. Processing for each of the following conditions will be discussed separately:

- The task is not already in the process of abnormal termination.

- The task is already being abnormally terminated and the Initiator is not the caller.

- The task is already being abnormally terminated and the Initiator is the caller.

THE TASK IS NOT ALREADY IN THE PROCESS OF ABNORMAL TERMINATION: (See Table 10-1, condition 3a.) In this case, the ABTERM routine proceeds to schedule the task for abnormal termination. (The clear state of the TCBFA flag in the job step TCB indicates that the job step TCB is not being terminated.) The ABTERM routine schedules the termination by:

- Ensuring that the task is dispatchable.

- Storing parameters for use by the ABEND routine.

- Scheduling the dispatching of the ABEND routine.

- Conditionally setting incomplete subtasks of the specified task nondispatchable.

- Returning control to the preloaded return address.

The ABTERM routine ensures that the ABEND routine can be dispatched for the terminating task. It does this by clearing all nondispatchability flags in the terminating task's TCB, except the "must complete" nondispatchability flags (TCBSYS and TCBSTP). The Dispatcher will later examine all these flags to determine that they are clear before dispatching the ABEND routine as the "current" routine for the terminating task.

Note: The nondispatchability flags are set by the supervisor for reasons such as: the resources of a task in the job step are being dumped by the ABDUMP routine, or the SER1 routine is in progress, or another task is in "must complete" status. (For further information on the TCB nondispatchability flags, refer to Table 10-2.)

The ABTERM routine next stores in the specified task's TCB the parameters that are needed by the ABEND routine. These parameters consist of the dump option flag, if a dump has been requested, and the completion code supplied by the caller. The parameters are stored in the "completion code" field of the TCB, called TCBCMP. The dump option flag, if set, later causes the ABEND routine to invoke the ABDUMP routine to display the programs and control blocks of the terminating task. The completion code is displayed during the dump as part of the TCB, and is made available

• Table 10-2. The TCB Nondispatchability Flags

Name of Flag	Offset of Flag in TCB	Meaning of Flag
TCBNDUMP	32.0	This task is nondispatchable while the resources of a task in this job step are being dumped.
TCBSER	32.1	This task is nondispatchable while the SER1 routine is being executed for this task.
	32.5	This task is nondispatchable while VARY or QUIESCE processing is being performed in a multiprocessing system.
TCBONDSP	32.7	This task is nondispatchable while the Open routine is being executed for this task as part of ABEND processing.
TCBFC	33.0	This task is nondispatchable because it has been normally or abnormally terminated.
TCBABWF	33.1	This task is nondispatchable as part of a tree of tasks being abnormally terminated.
TCBWFC	33.2	This task is nondispatchable because it has issued an unconditional GETMAIN not yet satisfied by rollout.
TCBFRO	33.3	This task is nondispatchable because it has been rolled out. (Meaningful in all TCBs except system task TCBs.)
TCBSYS	33.4	This task is nondispatchable while another task in the system is in "system must complete" status.
TCBSTP	33.5	This task is nondispatchable while another task in the same job step is in "step must complete" status.
TCBFCD1	33.6	This task is nondispatchable because it is an initiator task that is waiting for a requested region of main storage.

to the parent task, via the ABEND routine. (The parent of the job step task is the Initiator.)

The ABTERM routine next schedules the dispatching of the ABEND routine for the specified task. In essence, the scheduling consists of:

- Determining if the caller of the ABTERM routine is a type-1 SVC routine.

- Modifying the old PSW for the current routine so that it points to an SVC 13 instruction in the communication vector table (CVT). The old PSW may be either the RB old PSW of the task's "top" RB, or the SVC old PSW in lower main storage (if the task's current routine has no SVRB).

- Removing an RB wait condition (if it exists).

- Permitting the Dispatcher, on a task priority basis, to cause execution of the SVC 13 instruction.

When the SVC instruction is eventually executed, the SVC Second-Level Interruption Handler will fetch the ABEND routine from auxiliary storage (if it is not already in a transient area of main storage) and pass control to it. The ABEND routine is controlled during its execution as a part of the terminating task.

As a first step in the "scheduling" of the ABEND routine, the ABTERM routine determines which of two possible paths of processing will be followed. One path is used if the caller of the ABTERM routine is a type-1 SVC routine, and therefore is not controlled by an RB. The other path is followed if the caller is not a type-1 SVC routine, and therefore is controlled by an RB. This discussion will first consider the case in which the caller is not a type-1 SVC routine, as determined by a test of the "type-1" switch, IEATYPE1.

The Caller is not a Type-1 SVC Routine: If the caller is not a type-1 SVC routine, the RB old PSW and the wait count to be altered are in the "top" or current RB for the specified task. (The current RB is the one pointed to directly by the TCB.) Before pointed to directly by the TCB.) Before altering these fields, the ABTERM routine must first save the existing RB old PSW and the wait count, for display during ABDUMP processing. The second word of the RB old PSW, which contains the restart address, is saved in the RBABOPSW field of the current RB. For the same reason, the RB wait count, which the ABTERM routine clears, is also saved in the current RB. (If the current RB is an IRB, however, the wait count is not saved.) The RB wait count is cleared to prepare for supervisor linkage to the ABEND routine.

To permit the Dispatcher to place in execution an SVC-13 instruction for the terminating task, the ABTERM routine branches to the supervisor's Task Switching routine. The ABTERM routine passes to the Task Switching routine the TCB address of the specified task. The Task Switching routine compares the dispatching priority of the task to be terminated with the dispatching priority of the current task. If the task to be terminated is of higher priority than the current task, the Task Switching routine informs the Dispatcher by placing the higher priority TCB address in the "new" TCB pointer, IEATCBP. Without an alteration of the "new" TCB pointer, the Dispatcher would dispatch a routine belonging to either the current task or a lower-priority ready task.

After control is returned from the Task Switching routine, the ABTERM routine completes the scheduling of entry to the ABEND routine by pointing the previously mentioned RB old PSW to the SVC-13 instruction. It then sets the ABTERM flag (TCBABTRM) in the specified task's TCB, as an indication to both the ABTERM and ABEND routines that this task has been scheduled via the ABTERM routine. This indication, as described previously, limits ABTERM processing if a second branch to the ABTERM routine occurs for the same task.

In addition, the routine sets the "prevent asynchronous exits" flag (TCBFX) in the specified task's TCB. Its purpose is to prevent the scheduling of a user exit routine for the task by the Stage 3 Exit Effector during Dispatcher processing, before entry to the ABEND routine occurs. The execution of a user exit routine would be a waste of CPU time for a task that is no longer productive, and is potentially harmful. Before returning control to the caller, the ABTERM routine conditionally sets incomplete subtasks of the specified task nondispatchable, as discussed in "Processing if the Task Has Already Been Scheduled for Termination."

The Caller is a Type-1 SVC Routine: If the caller has been a type-1 SVC routine, the processing is similar to the foregoing. Instead of saving and altering the RB old PSW in the "top" RB of the specified task, the ABTERM routine does the saving in the "top" RB and the altering in the SVC old PSW in lower main storage. This variation is necessary, since type-1 SVC routines do not operate under the control of an RB. In addition, the Task Switching routine is not invoked, since the caller's register contents are still in their lower main-storage save area (IEASCSAV), and may be lost by another SVC interruption following a task switch.

THE TASK IS ALREADY BEING ABNORMALLY TERMINATED AND THE INITATOR IS NOT THE CALLER: (See Table 10-1, condition 3b.) If the job step task is in the process of abnormal termination by the ABEND routine and the Initiator is not the caller, an attempt is being made to repeat an abnormal termination for the same task. This means that an error condition has occurred during ABEND processing, which leads to a new request for abnormal termination of the task that is already being terminated. A new entry to the ABEND routine must be scheduled so that it can try, if possible, to complete termination procedures. Such a reentry to the ABEND routine for the same task is called a recursion. A recursion is valid only if an error occurs during the execution of the ABDUMP, Open, or Close routine during ABEND processing. If the recursion is valid, the ABEND routine continues the termination procedures. If, however, the recursion is invalid, the ABEND routine branches to the System Quiesce routine, which averts a CPU wait state by abnormally terminating only the failing task and its subtasks and by permiting the system to quiesce.

Since the original ABEND parameters (completion code and the dump option flag) must be used by the ABEND routine, new parameters are ignored and are not placed in the specified TCB. The scheduling of the ABEND routine and the flagging of incomplete tasks as nondispatchable are performed, as described in the topic "The Task is Not Already in the Process of Abnormal Termination." Similar also is the return of control. There are, however, two differences. The old PSW and the RB wait count, if applicable, are not saved, since on a recursion to ABEND, a dump is not provided.

THE TASK IS ALREADY BEING ABNORMALLY TERMINATED AND THE INITIATOR IS THE CALLER: (See Table 10-1, condition 3c.) There are two possible causes of an ABEND request by the Initiator while the job step task is already being abnormally terminated. The operator has issued a CANCEL command, or the job step timer has expired.

The processing varies, depending on whether a dump is specified. If a dump is specified, the ABTERM routine does not schedule entry to the ABEND routine, since the ABEND routine is already in execution to terminate the same task. But to prevent waste of system resources, the ABTERM routine conditionally sets any incomplete subtasks nondispatchable, as previously described. It then restores registers and returns control to the caller.

If a dump option is not specified, a CANCEL command was issued, probably to stop a prolonged dump that may be in an infinite loop. In this case, entry to the ABEND routine is urgent. In order to stop the dump, the ABTERM routine must give the appearance of a first-time request for termination, this time with a dump not requested.

The ABTERM routine gives the appearance of a first-time request for termination by clearing those flags in the TCB of the terminating task that indicate ABEND processing. The flags to be cleared are: TCBOPEN, which indicates that an OPEN macro instruction has been issued by the ABEND routine for the dump[1] data set; TCBFOINP, which indicates that the dump data set is in the process of being opened; and the dump option flag in the completion code. After clearing the flags, the ABTERM routine clears all nondispatchability flags, except the "must complete" nondispatchability flag, that may be set in the job step TCB. The purpose is to force the dispatching of ABEND for the job step task to end the prolonged dump. The ABTERM routine does not save the RB old PSW and the wait count of the current RB, since the new termination request will not cause a dump.

The remainder of ABTERM processing is similar to that previously described: the Task Switching routine is invoked, the ABEND routine is scheduled (RB old PSW and wait count are altered), the ABTERM flag and the "prohibit asynchronous exits" flag are set, incomplete subtasks are conditionally set nondispatchable, and control is returned as specified by the caller.

Processing if the Specified Task is not the Job Step Task

If the task specified for abnormal termination is not the job step task, as indicated by the TCBJSTCB field in its TCB, there are three possible paths of processing. The path taken depends on whether the specified task had been previously set nondispatchable by either the ABTERM or ABEND routine, and on whether the branch to the ABTERM routine represents an attempted recursion. The following discussion will consider each case separately.

THE TASK WAS PREVIOUSLY SET NONDISPATCHABLE BY ABTERM OR ABEND: (See Table 10-1, condition 4a.) In this case, entry to the

[1] The dump data set is either SYSABEND or SYSUDUMP.

ABEND routine is not scheduled. The reason is that an ancestor of the specified task is already in the process of abnormal termination. There is no need for an explicit request for termination of the specified task, since its resources will be released as part of the termination of its ancestor.

The processing for this condition consists of setting subtasks of the specified task nondispatchable (TCBABWF flag set), if they were not all previously placed in this condition. This prevents any use of system resources by a subtask of the terminating task. Possibly during a previous entry to the ABTERM routine, a subtask was not set nondispatchable because it was in "must complete" status. If a routine of the subtask has reset the "must complete" status, the subtask can now be set nondispatchable. The ABTERM routine then restores the caller's register contents from the TCB, and returns control to an address the caller specified.

THE TASK IS NOT IN THE PROCESS OF TERMINATION BY ABEND: (See Table 10-1, condition 4b.) For this condition (indicated by the clear state of the TCBFA flag), the processing is similar to that performed if the caller specified the job step task. The only difference is that in this case the ABTERM routine does not clear nondispatchability flags in the TCB of the specified task. These flags must be cleared by the routine that set them, before the ABEND routine can be executed for the task.

THE TASK IS IN THE PROCESS OF TERMINATION BY ABEND: (See Table 10-1, condition 4c.) In this case, it is necessary to schedule reentry to the ABEND routine to test for valid recursion.

Preparation for ABTERM Processing After a Program Interruption (ABTERM Prologue)

After a program interruption, the Program-Check First Level Interruption Handler (PC FLIH) cannot branch directly to the main entry point of the ABTERM routine (IEA0AB00). First, certain housekeeping functions needed by the ABTERM routine must be performed. These functions are performed by a routine of ABTERM, called the ABTERM Prologue routine.

Note: If a program check occurs in a user program, the Program Check FLIH does not branch to the ABTERM Prologue routine if both of the following conditions exist:

- A program interruption element (PIE) has been specified, and
- The program interruption control area (PICA) specifies this particular interruption type to be handled by a user routine.

The ABTERM Prologue routine performs five main functions:

- Obtains the TCB address of the task to be terminated and places it in a parameter register for use by the ABTERM routine.
- Sets up a completion code (system error code) that indicates the type of program check and places the error code in a parameter register for initial use by the ABTERM routine and ultimate use by the ABEND routine.
- Conditionally saves the program-interruption old PSW for later display by the ABDUMP routine. address for the ABTERM routine represents a location ABTERM routine will return control when its processing is complete.
- Sets up the dump option flag as an indication to the ABEND routine that it should invoke the ABDUMP SVC routine.

The ABTERM Prologue routine (hereafter called the Prologue routine) first gets the current TCB address. The TCB address specifies the task to be scheduled for abnormal termination. If the program check occurred in the I/O Supervisor, as indicated by the "set" condition of the "I/O original interruption" switch (IORGSW), the Prologue routine gets the TCB address from a request queue element (RQE) whose address has been placed in register 1 by the I/O Supervisor. If the program check did not occur in the I/O Supervisor, the Prologue routine obtains the TCB address from the "current" TCB pointer (IEATCBP+4).

After the determination of the TCB address, there are three streams of processing, depending on the source of the program check: a system or user program, a type-1 SVC routine (or the SVC FLIH), or the I/O Supervisor. The source of the program check is determined by tests of the "I/O original interruption" switch and the "type-1" switch. This discussion will first consider the path followed if the program check occurred in a routine of a system or user program.

The first stream of processing is for a system routine (except the I/O Supervisor or a type-1 SVC routine) or for a user program. The Prologue routine saves the registers and the address of the next executable instruction of the interrupted routine. This information is displayed during the dump that later occurs as part of ABEND processing. the Prologue routine saves the address of the instruction by storing the program interruption old PSW in the RB old PSW field of the current RB. This RB is the "top" RB on the RB queue for the current TCB. The old PSW, so saved, cannot be lost by a new program check occurring before the original information can be displayed by ABDUMP. The register contents belonging to the interrupted program are moved from the program-interruption save area in lower main storage to the register save area of the current TCB (TCBGRS field). They will eventually be placed in the ABEND routine's SVRB.

The Prologue routine next sets up a completion code (system error code) and a return address for later use by the ABTERM routine. The completion code indicates the type of interruption and suggests the source of the error, e.g., 0C6 = specification error. (See the publication Messages and Codes.) The ABTERM routine stores the completion code in the TCB for the task to be terminated. It then places in the return register the address of the Dispatcher, to which the ABTERM routine will return control when its processing is complete.

If the program check has occurred in a type-1 SVC routine (or in the SVC First-Level Interruption Handler), as indicated by the "type-1" switch (IEATYPE1), the Prologue routine sets up a completion code and a return address for use by the ABTERM routine. This processing is similar to that previously described, except that the error code is 0F2, indicating that a program check occurred in a type-1 SVC routine (or in the SVC FLIH). The purpose is still the same: to indicate to the programmer, via a later dump, the type of program in which the error occurred. The ABTERM routine will store the completion code in the TCB belonging to the task to be terminated. After setting the completion code, the Prologue routine places in a register the address of the Type-1 Exit routine, to which the ABTERM routine will return control.

In the third case, if the program check has occurred in the I/O Supervisor, as indicated by the "I/O original interruption" switch (IORGSW), the Prologue routine ignores the TCB address that it had previously obtained from the "current" TCB pointer. Instead, it gets the TCB address for the task to be terminated from a request queue element (RQE), whose address has been placed in register 1 by the I/O Supervisor. The TCB address so obtained may possibly not be that of the current TCB, since I/O errors do not occur synchronously with the operation of the CPU. The TCB address for the terminating task is then placed in a parameter register for use by the ABTERM routine.

The Prologue routine then sets up a completion code and return address for the ABTERM routine, in a manner similar to that previously described. In this case, however, the I/O error is indicated by a code of "0F1", indicating that a program check occurred in the I/O Supervisor. The return address for the ABTERM routine represents a location in the I/O First-Level Interruption Handler, called DISMISS.

Regardless of the source of the program check, the Prologue routine sets the dump option flag and places the flag and the completion code in the parameter register. The dump option flag will cause the ABEND routine to invoke the ABDUMP SVC routine. The position of the completion code in the parameter register indicates to the ABDUMP routine whether a system error or a user error has occurred (see Figure 10-3).

The Prologue routine next branches to the main entry point of the ABTERM routine (IEA0AB00).

DUMPING SELECTED AREAS OF MAIN STORAGE (ABDUMP)

ABDUMP is an SVC routine which may be invoked through issuance of a SNAP macro instruction, either by the ABEND routine during an abnormal termination, or at any time by a user program. It can therefore provide an abnormal dump or a dynamic dump. If it is invoked by the ABEND routine, it displays major control blocks, programs, and dynamically acquired storage belonging to the terminating task, its subtasks, and its direct ancestors.

In systems with Main Storage Hierarchy Support, ABDUMP dumps main storage in each hierarchy associated with the terminating

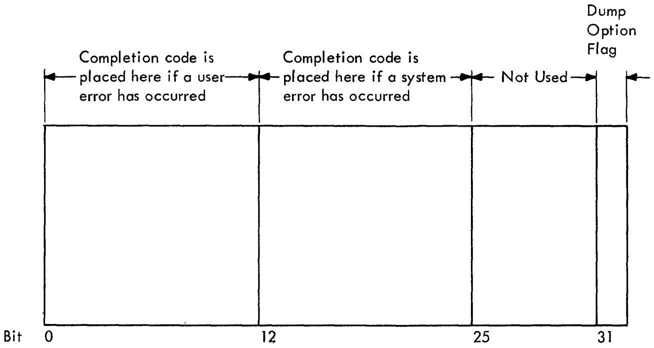

Dump Option Flag
Completion code is placed here if a user error has occurred
Completion code is placed here if a system error has occurred
Not Used
Bit
0
12
25
31

Figure 10-3. Format of the Completion Code and the Dump Option Flag in the Parameter Register

job step. Storage limits are determined by examining the PQE chain.

The SNAP macro instruction (whose expansion contains an SVC 51 instruction) causes the SVC Second-Level Interruption Handler (SLIH) to search for and fetch the ABDUMP routine, one module at a time. Only those modules of the ABDUMP routine whose functions are requested are fetched and executed.

The ABDUMP routine consists of nine nonresident modules each of which is separately fetched and executed, and one "resident" module, which remains in main storage for the entire dump procedure. The multiprocessing ABDUMP routine includes one additional non-resident module. The resident module (IEAQAD0A), loaded by the first segment of the ABDUMP routine, contains several format and output subroutines used by the other modules. The ABDUMP routine provides either a formatted printed display, or a series of blocked records on tape or on a direct-access medium, such as disk. In either case, the output consists of a group of control blocks, followed by the programs and/or dynamically acquired storage of the task, depending on the areas requested.

The first module, ABDUMP1, tests that a dump data set has been opened for the BSAM access method, provides a work area for use by the entire routine, loads the so-called "resident" module (format and output routines), conditionally gets storage space for preserving the trace table and for the blocking of records, and displays "indicative" information, such as job name, step name, time, date, etc.

ABDUMP2 formats and displays the old PSW, if requested, the TCB for the specified task, the request blocks on its RB queue, and the load list for the task. Optionally, it displays the TCB register save area.

ABDUMP3 formats and displays contents directory entries, their extent lists (one for each major CDE), the data extent blocks (DEBs), and the task I/O table (TIOT).

ABDUMP4 identifies, formats, and displays the control blocks of main storage supervision: subpool queue elements (SPQEs), descriptor queue elements (DQEs), free queue elements (FQEs), the dummy partition queue element, the partition queue elements (PQEs), and the free block queue elements (FBQEs).

ABDUMP5 formats and displays the control blocks that schedule serially reusable resources -- queue control blocks (QCBs) and queue elements (QELs) -- and register save areas belonging to interruption request blocks (IRBs).

ABDUMP6 formats and displays the register save areas for each user program of the task. For each save area the following information is displayed: the address of the save area, the contents of the save area, the type of linkage (LINK or CALL), the entry point identification, and a "call" identification (if the CALL macro instruction was used to obtain linkage).

ABDUMP11, executed in a multiprocessing system between ABDUMP6 and ABDUMP7, displays the trace table, if requested, and displays the prefixed storage area in the nucleus. If the multisystem mode is operating, the prefixed storage area at upper main storage is also displayed.

ABDUMP7 formats and displays the nucleus of main storage, the register contents of the user program at entry to ABDUMP, and dynamically acquired storage (if STORAGE is a keyword operand included with the SNAP macro instruction).

ABDUMP8 formats and displays load modules represented by contents directory entries. Each module fetched to main storage for the terminating or requesting task is displayed.

ABDUMP9 formats and displays storage obtained dynamically by user programs within the task. Each block of main storage is identified by a search of the subpool queue element (SPQE) queue. If requested, the trace table is displayed. The trace table provides information on most interruptions, start I/O instructions, and executions of the Dispatcher.

Processing During ABDUMP1 (Entry Point IGC0005A)

After having been fetched by the SVC SLIH, ABDUMP1 first tests two input parameters: the DCB for the dump data set, and the TCB for the task whose resources are to be displayed. (See Section 12, "Control Blocks and Tables," for the content and format of the ABDUMP parameter list.) The DCB is associated with the data set on which the dump will appear. The caller -- either the ABEND routine or a user program -- must previously have opened the DCB for the dump data set. If the DCB has not beenopened, ABDUMP1 sets up an error return code (4) and, via the Exit routine and the Dispatcher, returns control to the caller. Otherwise, processing continues. If a TCB address is provided as an input parameter, the resources of a task other than the current task are to be dumped. To avoid a program check, ABDUMP1 checks the validity of the TCB address. If the address is invalid, the routine sets up an error code (8), and returns control to the caller, via the Exit routine and the Dispatcher. If the test suggests a valid TCB address, processing continues.

If the task whose resources are to be dumped is not the current task (as indicated by the TCB address), ABDUMP1 sets all tasks of the job step nondispatchable except the current task. It does this to prevent concurrent dump requests issued by programs belonging to different tasks of the same job step from causing a possible "interlock" if one of the tasks abnormally terminates. Later, during ABDUMP9, when dynamically acquired storage has been displayed, the tasks will again be set dispatchable. If the multiprocessing feature was selected, control is passed to the Task Removal subroutine, which determines whether the current task on the second CPU has been set nondispatchable. If it has, the second CPU is interrupted with an indication (in STMASK) that the Dispatcher must gain control.

To provide a work area for use by all load modules of the ABDUMP routine, ABDUMP1 next obtains storage space. This area will later be used to save registers, to serve as an output buffer and as a work area, and to hold pointers and flags. Later, after ABDUMP9, the Where-to-Go routine of the "resident" module will free the space obtained by ABDUMP1.

ABDUMP1, via a LOAD macro instruction, next causes the fetching of the "resident" module of the ABDUMP routine (IGC0A05A) to the job step's region of main storage. If the "resident" module is resident in the link pack area, its execution is scheduled by the common subroutines of Contents Supervision. This module consists of format and output routines that are used during the entire dump. Included are three format routines, an Output routine, a Where-to-Go routine, and a "TCB selection" routine.

One format routine determines the position a labeled field will occupy on a print line. Another format routine determines the number of 32-byte lines of print needed to format a block of storage, and the number of bytes to be placed in the last incomplete print line. The third format routine unpacks a block of main storage and formats it in 4-byte fields in preparation for printout.

The Output routine issues the WRITE and CHECK macro instructions to print a line on a printer or write a block of storage on tape or a direct access device.

The Where-to-Go routine tests the flags in the input parameter list to determine which of several possible transient load modules of the ABDUMP routine should next receive control, after a given module's processing is complete. It also performs final housekeeping before control is returned to the caller of the ABDUMP routine.

The "TCB selection" routine permits certain modules of the ABDUMP routine to scan the TCBs of the job step in order to set the tasks nondispatchable or dispatchable. It is necessary to set tasks other than the current task nondispatchable. This prevents routines for other tasks from altering control blocks while the blocks are being displayed. If the multiprocessing feature was selected, control is passed to the Task Removal subroutine, which determines whether the current task on the second CPU has been set nondispatchable. If it has, the second CPU is interrupted with an indication (in STMASK) that the Dispatcher must gain control. After loading the resident module, if entry is not from the ABEND routine, ABDUMP1 issues an ENQ macro instruction for the dump data set. (The ABEND routine issues its own ENQ macro instruction for the dump data set.) It does this to prevent a program belonging to task in the same job step from concurrently causing a new dump to the same data set. This could occur during a period of dispatchability, before the current dump is complete.

Next, if a display of the trace table was requested as an option of the SNAP macro instruction, ABDUMP1 issues a conditional GETMAIN macro instruction for space so that it can move the contents of the table. The purpose is to prevent the table's further alteration during the ABDUMP and ABEND routines. If the table is moved, ABDUMP1 sets an indicator for ABDUMP9. If no space is available to which the trace table can be moved, the message "NO SPACE FOR TRACE TABLE" is issued. In this case, during ABDUMP9, the trace table will not be displayed.

The trace table was built by the Trace routine, a part of the nucleus. The trace table contains entries describing certain conditions at each SVC interruption, external interruption, program interruption, and I/O interruption. Other entries indicate conditions at each issuance of a Start I/O instruction and at each execution of the Dispatcher. Most entries contain the old PSW, the contents of three registers, the current TCB address, and the time of the interruption. In a multiprocessing system, the trace entries also contain the address of the TCB current on the second CPU and the contents of the CPU identity byte. (For the format of the trace table, see Section 12, "Control Blocks and Tables.")

If the output device for the dump data set is not a printer, records must be blocked. ABDUMP1 issues a conditional GETMAIN macro instruction to obtain space for the blocking of records. If space is not available, the processing continues without the setup for the blocking of records.

ABDUMP1 enables interruptions and initializes the work area it previously obtained. It then displays, via a format routine and the Output routine, the identification code (if specified), the job name, step name, time, and date. (See sample dump in Section 12, "Control Blocks.") If the ABEND routine is the caller, ABDUMP1 displays the completion code from the TCB for the specified task. Control is then passed to the next applicable module of the ABDUMP routine, via a branch to the Where-to-Go routine. This routine, by testing the dump option flags in the parameter list, determines the next module of the ABDUMP routine needed to satisfy the caller's dump options. The Where-to-Go routine obtains the needed module by issuing an XCTL macro instruction. The XCTL request, via the SVC SLIH, fetches and causes control to be given to the selected module of the ABDUMP routine.

<u>Processing During ABDUMP2</u>
<u>(Entry Point IGC0105A)</u>

ABDUMP2, if entered, first displays (via the format and output routines) the old PSW stored when the ABDUMP routine was entered. The old PSW is displayed if it was requested as an operand of the SNAP macro instruction. Next, ABDUMP2 unconditionally displays all labeled fields of the task's TCB, except its register save area. The register save area is displayed only if the caller requests the dump of task resources other than its own.

ABDUMP2 scans the request blocks of the RB queue for the specified task in order to display the labeled fields of each request block (RB). If any RB contains more than 32 bytes, as indicated by a test of the RBSIZE field, its register save area and extended save area, if they exist, are also displayed.

After all RBs on the RB queue have been displayed, ABDUMP2 displays the load list for the task, if the TCB (TCBLLS field) indicates that a load list exists. The load list contains pointers to contents directory entries for all modules that were fetched for the task via the LOAD macro instruction. After all the load list elements have been displayed, or if there was no load list for the task, ABDUMP2 invokes the next module, ABDUMP3, via an XCTL macro instruction.

<u>Processing During ABDUMP3</u>
<u>(Entry Point IGC0205A)</u>

ABDUMP3 displays the contents directory entries for the task, their extent lists (one for each major CDE), the data extent blocks (DEBs) chained from the TCB, and the task I/O table (TIOT). The contents directory entries and their associated extent lists are obtained via two searches. The first search consists of a scan of the RB queue to find PRBs, each of which may point to a CDE. The second search examines the load list for the task. Each load list element also points to a CDE. Each major CDE points to its associated extent list.

When all CDEs and their extent lists have been displayed, the data extent blocks (DEBs) chained from the TCBDEB field and the task I/O table (TIOT) are displayed. This completes the processing of ABDUMP3. ABDUMP3 invokes ABDUMP4, via an XCTL macro instruction.

<u>Processing During ABDUMP4</u>
<u>(Entry Point IGC0305A)</u>

ABDUMP4 displays all the main storage control blocks associated with the specified task, if two conditions are met: the task is not complete (TCBFC flag is not set) and there is at least one subpool queue element (TCBMSS pointer is not zero). If these conditions exist, the following control blocks are displayed:

<u>For the Specified Task:</u>

subpool queue elements (SPQEs)
descriptor queue elements (DQEs)
free queue elements (FQEs)

<u>For the Job Step's Region(s):</u>

partition queue elements (PQEs)
free block queue elements (FBQEs).

If the specified task is complete or there are no subpool queue elements, ABDUMP4 displays only the PQEs and the FBQEs for the job step's region(s).

ABDUMP4 then branches to the Where-to-Go routine of the resident module to determine the next applicable module of the ABDUMP routine.

The display of main storage control blocks is implemented as follows. The first step is to set nondispatchable all tasks in the job step except the current task. This is accomplished via a branch to the Task Select routine of the resident module. (This action may already have been done in ABDUMP1 if the specified task is not the current task.) The purpose is to prevent any program belonging to another task in the job step from being executed during an I/O wait condition of the current task. During such execution the program of the other task could issue a GETMAIN or FREEMAIN macro instruction, changing the main storage queues that are being displayed for the specified task. If the multiprocessing feature was selected, control is passed to the Task Removal subroutine, which determines whether the current task on the second CPU has been set nondispatchable. If it has, the second CPU is interrupted with an indication (in STMASK) that the Dispatcher must gain control.

ABDUMP4 then formats and displays each SPQE in the SPQE queue and its associated DQEs and FQEs. If a subpool is shared, both the owner's and the sharer's SPQEs are displayed. When all SPQEs and their associated DQEs and FQEs have been displayed, if the current task is the one specified for the dump, ABDUMP4 branches to the Task Select routine to make dispatchable other tasks in the job step. Dispatchability is now feasible, since all main-storage control blocks that are readily alterable have been displayed. But if the task specified for the dump is not the current task, the other tasks of the job step remain nondispatchable, as set by ABDUMP1, and the branch to the Task Select routine is bypassed.

After making other tasks dispatchable (if necessary), the partition queue elements (PQEs) and the free block queue elements (FBQEs) for the job step's region(s) are displayed. ABDUMP4 then branches to the resident module's Where-to-Go routine to determine the next applicable module of the ABDUMP routine needed to satisfy the current dump request.

Processing During ABDUMP5 (Entry Point IGC0405A)

ABDUMP5 displays queue control blocks (QCBs) and queue elements (QELs) for the entire job step, and/or save areas belonging to interruption request blocks (IRBs), depending on the dump options requested, as indicated by the option flags of the parameter list. If the ABDUMP routine was invoked by the ABEND routine, all these items are displayed.

If a display of QCBs and QELs for the job step is requested, the first step is to obtain the QCB origin address in the nucleus. Then, if the current task is the one specified for the dump, all other tasks in the job step are (via the Task Select routine) set nondispatchable. The purpose, as with the display of the main storage queues, is to prevent alteration of the QCB queues and QEL queues by programs belonging to other tasks while these control blocks are being displayed. If the task specified for the dump is not the current task, all tasks but the current task have been nondispatchable since the execution of ABDUMP1. If the multiprocessing feature was selected, control is passed to the Task Removal subroutine, which determines whether the current task on the second CPU has been set nondispatchable. If it has, the second CPU is interrupted with an indication (in STMASK) that the Dispatcher must gain control.

The QEL queue chained from each minor QCB is searched to find QELs that belong to either the specified task's job step, or to its Initiator. For the first QEL that schedules a given job step resource, ABDUMP5 displays both the QEL and its associated QCB. For each other QEL for the resource, only the QEL is displayed. ABDUMP5 compares two PQE pointers to determine whether a given QEL belongs to the current job step (including its Initiator). One of the PQE pointers (TCBPQE) is in the TCB whose address is contained in the QEL. The other PQE pointer is in the current TCB under whose control the ABDUMP routine is operating. If the two PQE pointers are equal, both TCBs belong to the same job step (or one represents the Initiator), since they both refer to the same region of main storage. In this case, the QEL is displayed. The examination and display of QELs belonging to the job step continue until all QELs have been examined, as indicated by a major-QCB chain address of zero.

If the current task is the one specified for the dump, all other tasks are next set dispatchable. But if the specified task is not the current task, other tasks are still nondispatchable as set by ABDUMP1, and this step is bypassed.

The next step is to display user-program save areas belonging to IRBs, if a save-area trace has been requested. If a save-area trace has not been requested, no further processing occurs in ABDUMP5, and a branch is made to the Where-to-Go routine of the resident module to determine the next module of ABDUMP to be invoked.

If a save-area trace has been requested as an option of the SNAP macro instruction, ABDUMP5 examines each RB on the RB queue of the specified TCB. For each IRB on the queue, the register save area is displayed. When the save areas of all IRBs on the RB queue have been displayed, ABDUMP5 processing is complete. ABDUMP6 is invoked, via an XCTL macro instruction, to continue the display of save-area information.

Processinq Durinq ABDUMP6 (Entry Point IGC0505A)

ABDUMP6 provides the heading line "SAVE AREA TRACE." The heading identifies the following lines as a trace of the program-provided register save areas for the task being dumped. Each save area is displayed in three printable lines, starting with the supervisor-provided save area for the first user routine of the task.

Save areas are displayed initially in a "forward" order, the order in which the associated routines were invoked by LINK or CALL macro instructions. The forward trace continues until all program-provided save areas have been displayed, or until incorrect forward or back chaining of save areas is discovered. Then, ABDUMP6 performs a partial "backward" trace, displaying the save areas for the two most recently executed user routines.

Besides the address and contents of each save area, ABDUMP6 displays the following messages:

- An "interruption" message, giving the address of the next executable instruction of the newest user routine of the task.
- A messaqe stating the type of linkage macro instruction (LINK or CALL) that was first used for the task.
- A message identifying the display of the backward trace.

The save area trace will now be described in greater detail. (See Figure 10-4.)

The forward trace begins as ABDUMP6 obtains the address of the supervisor-provided save area for the first executed user routine of the task. This save area is pointed to by the TCBFSA field of the TCB. ABDUMP6 checks the validity of the save area address. If the address is invalid (zero or not on a fullword boundary), most of the save area trace is bypassed and only the save areas for the two last executed routines of the task are displayed. But if the address of the supervisor-supplied save area is valid, information for the first-executed routine of the task is displayed (Figure 10-4, part 1). The information includes the type of linkage (LINK or CALL), the module name (obtained from the module's CDE), and the entry point identifier (if it was specified as an operand of the LINK or CALL macro instruction).

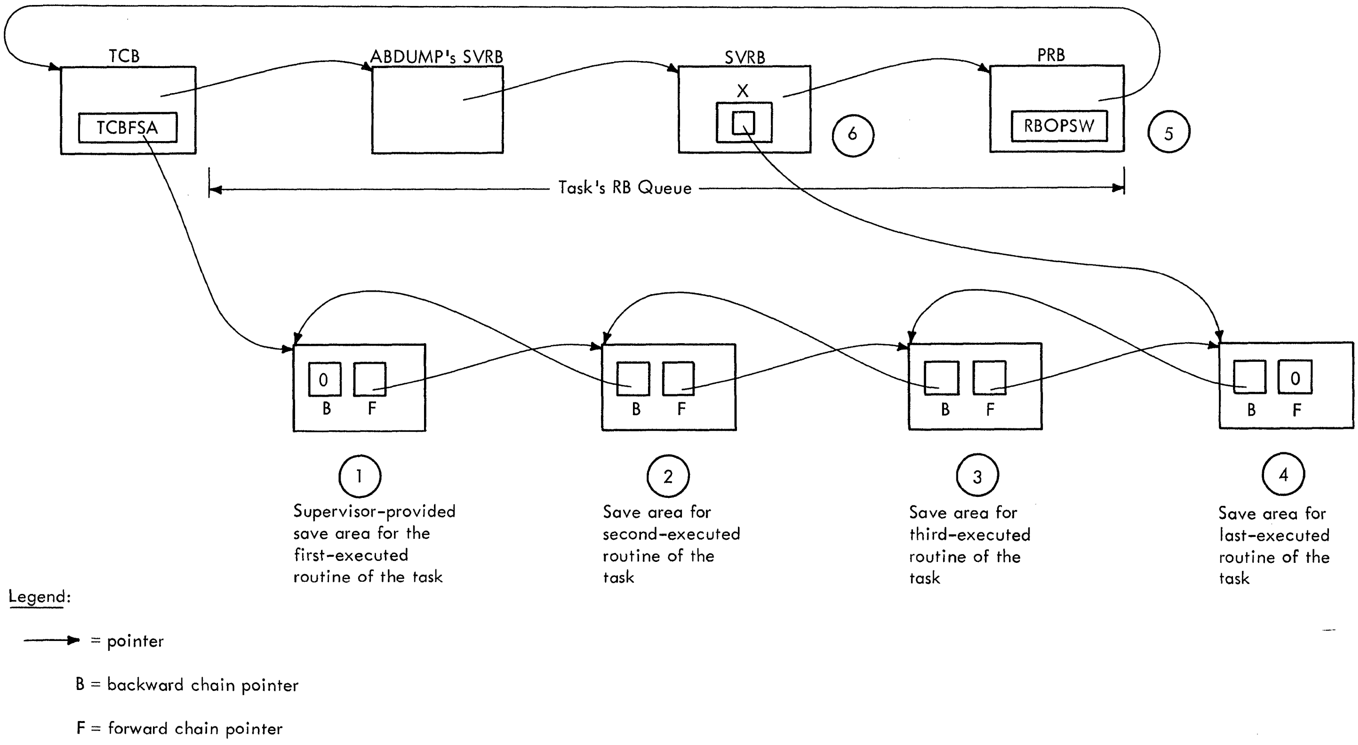

TCB
TCBFSA
ABDUMP's SVRB
SVRB
X
6
PRB
RBOPSW
5
Task's RB Queue
0
B F
B F
B F
0
B F
1
Supervisor-provided save area for the first-executed routine of the task
2
Save area for second-executed routine of the task
3
Save area for third-executed routine of the task
4
Save area for last-executed routine of the task
Legend:
= pointer
B = backward chain pointer
F = forward chain pointer

Figure 10-4. Pointers Used During the Save Area Trace

ABDUMP6 tries to complete the forward save area trace by performing the following steps:

- It obtains the forward chain pointer from the third word of the supervisor-provided save area and checks the pointer for validity (Figure 10-4, part 1, block F).

- If the forward chain pointer is valid, it obtains the backward chain pointer from the second word of the next save area and checks the pointer for validity (Figure 10-4, part 2, block B).

- If the backward chain pointer is valid, it displays the save area and its address (Figure 10-4, part 2).

These steps are repeated for each save area until all save areas have been displayed, as indicated by a forward chain pointer of zero, or until an invalid forward chain pointer or backward chain pointer has been detected. If ABDUMP6 detects an invalid backward chain pointer, it issues an error message "INCORRECT BACK CHAIN" and displays the associated save area.

ABDUMP6 next prepares for the partial backward trace that displays the register save areas for the two most recently executed user routines. It first obtains the address of the newest PRB on the task's RB queue (see Figure 10-4, part 5). This PRB represents the last executed user routine. ABDUMP6 then writes the interruption message consisting of the words "INTERRUPT AT," followed by the second half (address word) of the RB old PSW in the PRB. As a heading for the backward trace, ABDUMP6 issues the message "PROCEEDING BACK VIA REG 13."

ABDUMP6 then performs the partial backward trace. It first obtains the address of the save area for the last executed user routine of the task. This address is in the register-13 save location in the SVRB that precedes the newest PRB on the task's RB queue (Figure 10-4, part 6, block X). This save area address and the associated backward chain pointer (Figure 10-4, part 4, block B) are validity checked, and the two save areas and their addresses are displayed (Figure 10-4, parts 3 and 4). ABDUMP6 then branches to the Where-to-Go routine of the resident module to determine the next transient module of the ABDUMP routine to be invoked. The Where-to-Go routine makes the decision on the basis of the dump options specified by the ABDUMP routine's caller (as indicated by the option flags in the dump parameter list).

Processing During ABDUMP11 (Entry Point IGC0B05A)

ABDUMP11 is executed only in a multiprocessing system between ABDUMP6 and ABDUMP7. ABDUMP11 displays the trace table if it exists in the system and was requested as part of the dump. The trace table is displayed as in a uniprocessing system (see Processing During ABDUMP9). The character A or B is printed on each line/entry to identify the CPU to which the line/entry applies.

ABDUMP11 also displays the prefixed storage area(s) in the nucleus of main storage. If the partitioned mode is operating, only the prefixed storage areas at the lower end of main storage is displayed, preceded by the heading "CPU A PSA" or "CPU B PSA." If the multisystem mode is operating, both prefixed storage areas are displayed, preceded by the headings "CPU A PSA" and "CPU B PSA."

In a multiprocessing system, ABDUMP7 omits the prefixed storage area from the display of the nucleus, and ABDUMP9 does not display the trace table.

Processing During ABDUMP7 (Entry Point IGC0605A)

ABDUMP7 displays any combination or all of the following resources of the specified task, depending on the options requested by the caller.

- The nucleus of main storage.

- The register contents when the ABEND routine was entered, or when the SNAP macro instruction was issued.

- Selected blocks of main storage (if STORAGE is included as a keyword operand of the SNAP macro instruction).

If the caller has requested a dump of the nucleus of main storage, ABDUMP7 displays the nucleus, preceded by the heading NUCLEUS. If there is a trace table in the system, and it lies in the nucleus, only the part of the nucleus below the trace table is displayed (see ABDUMP1 for a discussion of the trace table). Then the heading NUCLEUS CONT and the rest of the nucleus above the trace table are displayed. ABDUMP7 bypasses the current copy of the trace table because the table now contains misleading information. This information was inserted after SVC interruptions, I/O interruptions, and entries to the Dispatcher, during execution of the ABDUMP routine. The original copy of the trace table was saved by ABDUMP1, if space was available, and will be displayed by ABDUMP9.

In a multiprocessing system, the prefixed storage area(s) are displayed by ABDUMP11 (IGC0B05A). Therefore, ABDUMP7 displays the nucleus starting at location X'1000.'

ABDUMP7 next displays the register contents as they appeared when the SNAP macro instruction was issued. If the ABEND routine was the caller, the register contents are obtained from the ABEND routine's SVRB. Otherwise, the register contents saved in the ABDUMP routine's SVRB are used for the display. The display is preceded by either of two messages: "REGS AT ENTRY TO ABEND" or "REGS AT ENTRY TO SNAP."

If a SNAP macro instruction was issued with the keyword STORAGE, the areas of main storage requested by the caller are formatted and displayed. To protect private information, storage is displayed only if it lies within the caller's region. Each eight words of storage is preceded by its starting address. ABDUMP7, its processing now complete, branches to the Where-to-Go routine of the resident module to determine the next transient module of the ABDUMP routine to be invoked.

Processing During ABDUMP8 (Entry Point IGC0705A)

ABDUMP8 displays load modules for the task whose resources are being dumped. The information needed to display each load module is obtained from the contents directory entry (CDE) for the module and from the associated extent list.

There are two possible sources of information needed to dump load modules. One source is the group of CDEs pointed to by PRBs belonging to the task. These CDEs represent modules requested by an ATTACH, LINK, or XCTL macro instruction. The other source is the group of CDEs pointed to by elements of the load list for the task. These CDEs represent modules requested by a LOAD macro instruction. (For a review of the contents directory and the load lists, see Section 4, "Contents Supervision.") If the task specified for the dump has already been terminated, either normally or abnormally, as indicated by the "set" condition of the TCBFC flag, all PRBs have been removed from the task's RB queue and have been freed. To determine if the RB queue still exists and can be examined, ABDUMP8 examines the TCBFC flag to test for previous task termination. If the task was not terminated, both the RB queue and the load list are scanned for pointers to CDEs. (For the content and format of a PRB, a CDE, a load list element, and an extent list, see Section 12, "Control Blocks and Tables.")

ABDUMP8 obtains the following information from the CDEs:

- Whether the module is already in main storage or in the process of being fetched.
- The address of the module's extent list. The extent list contains the main storage address and length of each loadable section of the module.
- The module's entry point name.
- Whether the module is in the area of main storage specified by the caller (job pack area or link pack area).

If the module is in the specified main storage area, ABDUMP8 displays a heading line, containing "LOAD MODULE" and the module's name, followed by the contents of the module itself. The normal line of the display contains eight words of storage preceded by their starting address.

When the load modules described by all CDEs have been displayed, ABDUMP8 branches to the Where-to-Go routine of the resident module. This routine determines whether ABDUMP9 should be invoked, or whether control should be returned to the caller of the ABDUMP routine.

Processing During ABDUMP9 (Entry Point IGC0805A)

ABDUMP9 displays the trace table if it exists in the system and was requested as part of the dump, and if the table was saved during ABDUMP1. (Refer to ABDUMP1 for a brief description of the trace table, and to Section 12 for the format of its entries.) In a multiprocessing system, the trace table is displayed by ABDUMP11 (IGC0-B05A) and, therefore, is not displayed by ABDUMP9. ABDUMP9 also displays user subpools of main storage that have subpool numbers not greater than 127. When all user subpools have been displayed, ABDUMP9 branches to the Where-to-Go routine of the resident module (IEAQAD0A), to prepare for and return control to the caller of the ABDUMP routine.

The details of the processing in ABDUMP9 will now be described.

ABDUMP9 displays the trace table in two parts. (The program listing calls this procedure "unfolding" the trace table.) ABDUMP9 starts the display at the trace table entry immediately after the current entry, and proceeds to the end of the table. It then displays the rest of the table by starting at the first entry and proceeding to the current entry. Pointers to the trace table exist in a triple word whose address is obtained from the secondary communication vector table (see Section 12, "Control Blocks and Tables"). The first word points to the address of the current entry of the trace table; the second word points to the start of the table; the third word points to the end of the table.

After displaying the trace table, ABDUMP9 frees the space previously obtained for the table.

ABDUMP9 next displays user-obtained main storage if two conditions exist: there is at least one subpool queue element (SPQE) on the task's main storage queues (TCBMSS flag is not zero), and the SPLS operand was specified in the SNAP macro instruction. Otherwise, ABDUMP9 branches to the Where-to-Go routine in the resident module (IEAQAD0A) to end the dump and return control to the caller.

If user main storage is to be displayed, the job step is set temporarily nondispatchable to prevent alteration of the main storage queues during the display. If the multiprocessing feature was selected, control is passed to the TESTDSP subroutine which determines whether the current task on the second CPU has been set nondispatchable. If it has, the second CPU is interrupted with an indication (in STMASK) that the Dispatcher must gain control.

For each subpool queue element (SPQE), ABDUMP9 checks that the subpool number is for a user area of storage, as indicated by a subpool number not greater than 127, and that the SPQE does not represent a shared area of storage. (For the content and format of an SPQE see Section 12, "Control Blocks and Tables.") If the SPQE indicates that the area is shared, the "owner" SPQE is obtained via an SPQE pointer in the DQE-pointer field of the SPQE. The "owner" SPQE is the element created by the GETMAIN routine when the block of storage was first requested.

ABDUMP9 obtains from the descriptor queue element (DQE), pointed to by the SPQE, the starting address of the block of main storage for the original GETMAIN request and the number of bytes allocated for the request. ABDUMP9 displays a header line giving the subpool and block number. The subpool number is obtained from the SPQE, the block number from the DQE. It then formats the block, normally eight words to a line, and displays it. There may be one or more free areas in the block to be displayed, as indicated by the existence of a free queue element (FQE) pointed to by the DQE. In this case, ABDUMP9 divides the block into sections separated by free areas. It then formats and displays the block, bypassing each free area, so that free areas do not appear in the dump output. The process is repeated for each DQE belonging to an SPQE and for each SPQE in the queue. When all SPQEs have been processed, ABDUMP9 sets all other tasks of the job step dispatachable. Since the display of user-acquired main storage is finished, GETMAIN requests will not now affect the dump. ABDUMP9 then branches to the Where-to-Go routine of the resident module (IEAQAD0A) to prepare for return of control to the caller: the ABEND routine or the issuer of the SNAP macro instruction.

Cleanup in the Where-to-Go Routine

The cleanup procedure of the Where-to-Go routine of the resident module (IGC0A05A) ends the dump and prepares for return of control to the caller by:

- Displaying message "END OF DUMP."

- Freeing all areas obtained during the execution of ABDUMP1 (i.e., the work area and the optional area for the blocking of records).

- Issuing a DEQ macro instruction for the dump data set, if the ABDUMP routine was invoked by a user routine. If the ABDUMP routine was invoked by the ABEND routine, the ABEND routine issues the DEQ macro instruction, specifying the dump data set. The data set can then be used by the ABDUMP routine for another caller specifying the same data set.

- Deletes the resident module and returns control to the caller, via the Exit routine and the Dispatcher. It does this by moving a DELETE macro instruction and an SVC-3 instruction to the extended save area of the ABDUMP routine's SVRB, and then executing these instructions. In the program listing this process is called "self delete."

PERFORMING ABNORMAL TERMINATION (ABEND ROUTINE)

Abnormal termination occurs when some type of unrecoverable error, such as a machine check, I/O error, or program check has taken place. It may also be initiated by a system or user program that detects an abnormal condition that could cause a program check or incorrect processing. The task whose program or I/O operation has malfunctioned is abnormally terminated because reliable results can no longer be obtained. The task must be terminated to prevent waste of system resources, such as CPU time or main storage.

The purpose of abnormal termination is to free the resources of the malfunctioning task so that they can be made available to other tasks in the system. The freed resources include programs in main storage, enqueued resource requests, unexpired timer requests, incomplete operator communications exclusively used data sets, and unshared subpools of main storage (if dynamically acquired). These resources belong to the specified task itself and its previously unterminated descendants. In addition, control blocks used by the terminating task and its descendants are dequeued from their lists and in most cases freed. These control blocks include: TCBs, RBs, IQEs, QELs, QCBs, SPQEs, CDEs, TQEs, and PIEs (if they exist).

Abnormal termination allows two options: task and job step termination. These are normally user options, specified by an operand of the ABEND macro instruction. In task termination the resources of only the malfunctioning task and its previously unterminated descendants are released. This option permits a program belonging to a higher level task in the job step to decide whether to continue or terminate the other tasks of the job step. But in step termination the resources used by all tasks of the job step are freed. Step termination may be elected by a user program (via the STEP operand of the ABEND macro instruction), or caused by the ABEND routine if it cannot obtain sufficient storage for the closing of data sets, and must "steal" storage from a block shared with other tasks in the job step.

The termination procedure is performed by the ABEND routine, a type-4 (reentrant, nonresident, segmented, partially disabled) SVC routine. As stated before, the ABEND routine frees resources and control blocks belonging to the offending task and its previously unterminated descendants (subtasks, tasks, and subtasks of subtasks). For several unusual conditions (a task in "must complete" status, a terminating system task, and an invalid recursion), the ABEND routine branches to the system quiesce routine which abnormally terminates only the failing task, sets its subtasks nondispatchable, and permits the system to quiesce.

If the dump option had been selected (either by the user program or by the ABTERM routine), the ABEND routine causes the loading and execution of the ABDUMP SVC routine. The ABDUMP routine displays the programs, control blocks, and dynamically acquired storage of the terminating task, its descendants, and its ancestors, including the job step task.

The ABEND routine may be invoked directly or indirectly. The invocation is direct when a system or user routine issues an ABEND macro instruction to terminate the current task. The invocation is indirect when a system routine, after detecting an abnormal condition, branches to the ABTERM routine. The ABTERM routine schedules the execution of an SVC 13 (ABEND macro instruction) for the task to be terminated. The SVC 13 instruction, executed when the task to be terminated is next dispatched, causes supervisor linkage to the ABEND routine.

The entry is indirect in the following situations:

- A type-1 SVC routine, which is not permitted to issue an SVC instruction, decides to terminate the current task.
- A supervisor routine decides to terminate a task other than the current task.
- The I/O Supervisor, whose execution is asynchronous with task performance, decides to terminate a task for which an unrecoverable I/O error has occurred.
- A program check occurs during the performance of any task.

The ABEND routine is composed of six nonresident or transient modules. The first module determines whether STAE processing, rather than ABEND processing, should be performed on the failing task. The other five modules perform normal ABEND processing, releasing task resources and invoking the ABDUMP routine. Each module is fetched and given control by the transient area handler. The loading of the first segment occurs after the SVC 13 (SVC ABEND) instruction is issued (either by the caller or indirectly by the ABTERM routine). The issuance of an XCTL macro instruction at the end of each module causes supervisor linkage to the transient area handler to fetch and pass control to the next module. ABEND6, usually the last executed module of the ABEND routine, returns control to the current routine of the highest priority ready task, by branching to the supervisor Exit routine and the Dispatcher (via an SVC 3 instruction).

The System Quiesce routine, which is part of the nucleus, may be branched to by ABEND under special conditions to abnormally terminate the failing task, set its related tasks nondispatchable, and permit the system to quiesce.

Processing During ABEND1
(Entry Point IGC0001C)

The SVC SLIH fetches the first module of the ABEND routine (ABEND1) from the SVC library. The SVC SLIH then gives control to ABEND1, via the Dispatcher.

The main function of ABEND1 is to determine if normal ABEND processing should be executed for the failing task or if the ABENDing task issued a STAE macro instruction and the ABEND/STAE interface routine (ASIR) should be invoked. ABEND1 first tests the TCBNSTAE field of the TCB to determine if a STAE was issued. If a STAE environment is not in effect (TCBNSTAE = 0), ABEND2 is invoked to continue normal ABEND processing.

If a STAE was issued by the failing task, ABEND1 next determines, by testing a bit in the TCBNSTAE field, if the failure occurred while ASIR was attempting to quiesce I/O. If so, control is returned to ASIR which will halt the I/O in progress.

Even if the failing task has issued a STAE, normal ABEND processing will continue if one of the following conditions exists:

- The completion code passed in register 1 is a 13E, signifying that the jobstep task issued a DETACH macro instruction during the processing of a subtask.
- A bit in the TCBFLGS field of the job step TCB indicates that the ABEND was caused by either expiration of the job step timer or by an operator's CANCEL command.
- The STAE recursion bit in the TCBNSTAE field indicates that ABEND was entered because of a failure during STAE processing (other than a failure during the quiesce I/O pruge operation).

If any one of the above conditions is indicated, ABEND2 is invoked and normal ABEND processing is continued.

Before passing control to ASIR for STAE processing, ABEND1 tests the TCBPIE field in the TCB, and, if a program interruption element (PIE) exit exists, it is purged. If the ABTERM bit in the TCBFLGS field indicates that the completion code passed in register one has not been stored in the TCBCMP field, ABEND1 stores it there. ABEND1 (via an SVC 7 instruction) passes control to ASIR1 to begin STAE processing. (See the description of the STAE macro instruction in Section 3.)

Processing During ABEND2
(Entry Point IGC0401C)

ABEND2 performs the following functions:

- Determines if the task which is ABENDing is a graphics job. If so, branches to the graphics exit routine to attempt error recovery.
- Clears the ABDUMP nondispatchability flag (TCBNDUMP) in all TCBs of the job step.
- Recognizes whether a serious program error condition has occurred (such as a termination of a system task). If such a condition exists, branches to the system quiesce routine which terminates the failing task, sets its subtasks nondispatchable, and informs the operator that a CPU wait state has been averted and the system must be allowed to quiesce.
- Determines whether the entry to the ABEND routine is a first-time entry or a reentry from a previous execution of the ABEND routine, and thus decides whether to bypass ABEND3.
- Determines if the rollout/rollin feature is in the system. If so, invokes the Rollout Purge routine (ROLLPRGC) to remove the appropriate IQEs from the rollout queue.

PROCESSING FOR GRAPHICS JOBS: If the task being terminated is a graphics job, ABEND2 sets the subtasks of the ABENDing tasks nondispatchable. ABEND2 then determines whether (1) enough storage is available for a dump and (2) the task is resumable, i.e., the ABEND was issued by a user program or caused by a program check in a user routine. Control is next passed to the Graphics Exit routine. At the completion of this routine if the task is resumable and the user chose to resume processing, the subtasks are set dispatchable, the right half of the PSW is set to the user's specifications and control is returned to the caller via an SVC3 instruction. Otherwise, control is passed to ABEND3.

CLEARING THE ABDUMP NONDISPATCHABLE FLAG: The first main step for non-graphics jobs performed by ABEND2 is to clear in each TCB of the job step a non-dispatchability flag (TCBNDUMP) set during a possible previous execution of the ABDUMP routine. ABDUMP sets this flag in each TCB of the job step (except that of the current task) in order to prevent alteration of dynamic queues during their display. ABEND2 must now clear these non-dispatchability flags in order that the Dispatcher may restart normal (nonterminating) tasks of the job step.

If the multiprocessing feature was selected, control is passed to the TESTDSP subroutine, which determines whether the current task on the second CPU has been set nondispatchable. If it has, the second CPU is interrupted with an indication (in STMASK) that the Dispatcher must gain control.

The following example illustrates this point. Assume that normal task A has requested a dynamic dump of task B's resources. The ABDUMP routine, when it gets control, sets all tasks in the job step nondispatchable to prevent alteration of dynamic queues during the dump. While the dump is in process and before the ABDUMP routine can reset non-dispatchability, an error occurs that abnormally terminates task A. All tasks of the job step would remain nondispatchable if the ABEND2 routine did not clear ABDUMP nondispatchability soon after it gained control.

RECOGNIZING A SEVERE ERROR CONDITION: ABEND2 next tests whether the current entry to the ABEND routine represents an error condition serious enough to warrant branching to the System Quiesce routine (entry point IECIWTST). This routine places the failing task in a wait state, sets its subtasks nondispatchable, and issues a message to the operator indicating that a CPU wait state has been averted and the system should be allowed to quiesce. The three conditions of extreme severity are:

1. The attempted abnormal termination of any system task because of a program check during its performance.

   Note: The CPU is not placed in the wait state, nor is an error message issued to the operator, if ABEND2 is entered from the Machine-Check Handler[1] because of a machine check. In this case, the formatted dump normally produced during ABEND5 is prevented by the setting of the "prevent dump" indicator (TCBPDUMP) by ABEND2 in the job step TCB. The dump is prevented because a dump is not informative after a machine check. An entry to ABEND2 from the Machine-Check Handler is indicated by a condition code of 0F3.

2. The attempted abnormal termination of a task in "must complete" status. A task in "must complete" status must be completed in order for the system to remain intact. It should not be abnormally terminated.

3. An invalid reentry ("recursion") to the ABEND routine for a task that is already being terminated. A recursion is valid only if caused by an error in one of the following situations:

   - The execution of the Open routine of data management (during ABEND4) to open the SYSABEND or SYSUDUMP data set.

   - The execution of the Close routine of data management (during ABEND5) to close a data set belonging to the terminating task.

   - The execution of the ABDUMP routine (during ABEND5).

If one of the severe error conditions is detected, ABEND2 branches to the system quiesce routine (IEAQTWST) whose address is obtained from word 72 (dec.) of the CVT. The system quiesce routine places the failing task in a wait condition, sets its subtasks nondispatchable, isolates the region associated with the failing task for later analysis, and issues a message to the operator indicating that a CPU wait state has been averted and instructing him to permit the system to quiesce by not scheduling any additional jobs. If none of the extremely serious conditions exists (as indicated by flags in the current TCB), ABEND2 continues processing.

DISCRIMINATING BETWEEN FIRST-TIME ENTRY AND VALID RECURSION: At this point in ABEND2 processing there are two possible paths, depending on whether the entry for the current task is a first-time occurrence or a valid recursion. (A valid recursion is indicated by the set condition of both the TCBREC flag and any one of three other flags: TCBADUMP, TCBOPEN, or TCBCLOSE.) If entry to the ABEND routine is due to valid recursion, most of ABEND2 processing is bypassed, since it was performed during a previous entry to the ABEND routine. ABEND3 is invoked to handle the recursion. There are, however, several steps which are performed if the system has the rollout feature. These include the removal of IQEs from the rollout queue and the asynchronous exit queue.

REMOVING IQES FROM THEIR QUEUES: IQEs are removed from the rollout queue and from the asynchronous exit queue by the Rollout Purge routine. Input to this routine consists solely of the address of the TCB for the abending task.

IQEs on the rollout queue are examined first. If the TCB address in the first

[1] The Machine-Check Handler is a system generation option available with the System/360 Model 65. See Section 2, "Interruption Handling."

word of the parameter list addressed by an IQE on the rollout queue is equal to the TCB address passed to this routine, the count of queued rollout requests is decremented by one, and the IQE is removed from the queue and returned to the available list (whose origin is the rollout IRB).

If a TCB match is not made against an IQE on the rollout queue, or if the queue is empty, IQEs on the asynchronous exit queue (AEQ2) are examined. If a match is made against the TCB address in the parameter list addressed by an IQE on this queue, the IQE is removed from the queue and returned to the available list.

If no match is obtained against an IQE on the AEQ2 or if the queue is empty, the queue of IQEs originating from the rollout IRB is examined. If a match is obtained against the TCB address in the parameter list addressed by an IQE on this queue and the IQE is not at the head of the queue (not addressed by the RBIQE field and therefore not currently being processed by Rollout), the IQE is removed from the queue and returned to the available list. If a match is made and the IQE is at the head of the queue (currently being processed by Rollout), a flag is set in the parameter list addressed by the IQE to indicate to the Rollout routine that the task is in the process of terminating abnormally.

Processing During ABEND3 (Entry Point IGC0A01C)

Control is passed to ABEND3 from ABEND2 after IQEs have been removed from the rollout queue or the asynchronous exit queue.

ABEND3 performs the following functions:

- Clears the "ABTERM" and "valid recursion" flags in the task's TCB if entry is due to a valid recursion.
- Purges those resources of the terminating task and its descendants that can operate asynchronously with the CPU and can cause needless processing during the course of the termination. The resources include unexpired timer intervals, I/O requests and I/O operations that are in process, outstanding WTOR requests, unscheduled requests for user (asynchronous) exit routines, and the use of a program interruption routine (if a PIE exists).
- Checks the validity of free queue elements in the main storage queues in order to avoid program checks during later issuance of GETMAIN and FREEMAIN macro instructions.
- Tests for the availability of main storage, to be used during the closing of data sets by ABEND5. If main storage is not available, ABEND3 prepares for a step termination, prevents a dump, and "steals" main storage from the job step for use by the Close routine during ABEND5.

PROCESSING FOR A VALID RECURSION: The first step for a valid recursion is to clear the ABTERM and "valid recursion" flags in the TCB for the current task. (The flags are TCBABTRM and TCBREC.) The ABTERM flag, when set, prevents the ABTERM routine from scheduling multiple entries to the ABEND routine for the same task, before the ABEND routine can be entered. Since the ABEND routine has now been entered for the task, the TCB ABTERM flag must be cleared. The "valid recursion" flag (TCBREC), if set during a previous partial termination of the task, must be cleared. Otherwise, ABEND3 will misinterpret as valid a future invalid recursion. Remember that the only valid recursions during ABEND processing are those occurring during the execution of the Open, ABDUMP, and Close routines.

The next step is to purge I/O requests and requests for asynchronous exit routines that were possibly generated during the previous entry to the ABEND routine. Older requests of the same type, initiated by system or user programs of the task, were purged during the earlier ABEND3 execution. New queue elements and I/O operations created or begun for the ABEND open, dump, and close functions must be eliminated. This is done for two reasons: to prevent waste of system resources, and to avoid the posting by the I/O Supervisor of ECBs that will no longer exist after the ABEND routine has purged main storage.

The remainder of recursion processing in ABEND3 is the same for both recursion and first-time entry. This processing consists largely of obtaining main storage for the close function of ABEND5, and will be described under the heading "Stealing Main Storage."

PROCESSING FOR A FIRST-TIME ENTRY: When ABEND3 has determined that a severe error has not occurred (such as the termination of a system task), and that the current entry to the ABEND routine is not a recursion, it then tests the scope of the termination request.

Determining the Scope of the Termination Request: The termination request may be for a single task and its unterminated descendants or for the entire job step. The choice, an option of the ABEND macro instruction, is indicated in the input parameter list. The tree of tasks to be terminated originates with the current task, unless the STEP option has been specified in the ABEND macro instruction. If the STEP option has been specified, the terminating tree of tasks must originate with the job step task.

An "alternate TCB" pointer is preloaded with the address of the job step TCB, on the initial assumption that the caller has specified the STEP option or belongs to the job step task. If the assumption is incorrect, ABEND3 places in the "alternate TCB" pointer the address of the current or caller's TCB. In this case, the "alternate TCB" pointer specifies the current task as the "top" task of the tree of tasks to be terminated. The "top" or specified task and all its previously unterminated descendants are terminated during the course of ABEND processing.

Setting Descendants Nondispatchable and Preventing Asynchronous Exits: ABEND3 sets nondispatchable all incomplete descendants of the specified task, and prevents asynchronous exits for these descendants. The main purpose is to avoid the possibility of a subtask gaining control during an I/O wait period and causing a new abnormal termination. Such a new termination would be interpreted by ABEND4 as an invalid recursion, and would cause a branch to the System Quiesce routine. A secondary purpose is to prevent the waste of system resources for subtasks that are planned for termination but are not yet terminated.

To accomplish the foregoing purposes, and to indicate that the tasks of the tree are in the process of abnormal termination, ABEND3 sets the following three flags in the TCB of each descendant:

- "Abnormal wait" flag (TCBABWF) which indicates to the Dispatcher that it may not place into execution any routine of the task.

- "Prevent asynchronous exits" flag (TCBFX), which indicates to the Stage 3 Exit Effector that it may not transfer interruption queue elements from an asynchronous exit queue to a queue belonging to an IRB. It also prevents Stage 3 from queuing an IRB to a TCB, and thus prevents the scheduling of an asynchronous exit routine.

- "Termination in process flag" (TCBFA), which indicates to ABEND3, on later reentry for the same task, that a recursion has occurred.

In order to obtain the address of each TCB whose flags must be set, ABEND3 uses a "task select" subroutine. This subroutine, used in various modules of the ABEND routine, and in the ABTERM and ABDUMP routines, scans the tree of TCBs whose tasks are to be terminated. It starts with the newest descendant of the "top" TCB. It then examines the tree of TCBs from the newest descendant to the top TCB. For each selected TCB, the three aforementioned flags are set.

If, during the scan of the tree of tasks, a TCB is found that indicates (by its TCBOINP flag) that the SYSABEND or SYSUDUMP data set is being opened for the task, the task is not set nondispatchable. (That is, the three flags in its TCB are not set.) The opening of the dump data set (SYSABEND or SYSUDUMP) must continue without interruption. Otherwise, ABEND4 would prevent dumps for the entire job step.

In addition, if the dump data set is being opened for a task, its TCB "top" flag (TCBFT) is cleared. The reason is that the task, partially terminated as the top task of its tree, is no longer the top task. Its "top" flag is therefore altered to reflect its more recent place in the tree.

When the three flags have been set in all TCBs of the tree, ABEND3 clears the "abnormal wait flag" (TCBABWF) in the current TCB, so that the next module of the ABEND routine may be dispatched for the current task when ABEND3 is complete. The "top" flag (TCBFT) is also set in the TCB for the top or oldest task of the tree, to indicate to the ABEND routine that this task and all its incomplete descendants are to be terminated.

If the multiprocessing feature has been selected, control is passed to the Task Removal routine, which determines whether the current task on the second CPU has been set nondispatchable. If it has, the second CPU is interrupted with an indication (in STMASK) that the Dispatcher must gain control.

Purging Resources for the Specified Task and Its Descendants: ABEND3 (via separate routines) purges for the tree of tasks the timer queue, I/O requests and I/O operations in process, the WTOR queues, the asynchronous exit queue for non-I/O requests, and the program interruption element (PIE), if one has been specified. Using the "task select" subroutine, and starting with the newest descendant task of the specified or "top" task, ABEND2 purges the resources and resource requests for each task in the tree. During the scan of the tree of tasks, only resources belonging to previously unterminated descendant tasks are released. Tasks that were previously terminated, either normally or abnormally as indicated by the "completion" flag (TCBFC), are ignored and the next task selected for release of resources.

Purging the Timer Queue: The first group of resource requests to be purged for each task is contained in the timer queue. The Timer Purge routine removes from the timer queue those elements that represent unexpired timer requests for the task. It also frees the space occupied by these elements. The purpose is to minimize the number of external interruptions for tasks that are terminating. The Timer Purge routine also frees the problem program register save area associated with each user (asynchronous) exit routine. (The associated save area is pointed to by its TQE.)

Purging I/O Requests and I/O Operations in Process: ABEND3 purges I/O requests and I/O operations in order to avoid errors that can cause recursion to the ABEND routine. Since the ABEND routine frees main storage, an I/O operation that is not halted can cause information to be read into main storage that may have been reallocated. The result is that data or programs can be destroyed. Furthermore, an event control block may be posted in reallocated main storage, thus causing an additional error. ABEND3 removes (via the SVC Purge routine) I/O requests (RQEs) that have not yet been serviced. By Halt I/O instructions, the SVC Purge routine stops I/O operations in process for each task of the tree. RQEs removed from the request queue are returned to a list of available RQEs for reuse by the I/O supervisor. Besides purging I/O operations in process and outstanding I/O requests, the SVC Purge routine dequeues, from the SIRB, elements representing scheduled requests for the use of I/O error handling routines.

Purging the Operator Communication Queues: After removing I/O requests and halting current I/O operations for a task, ABEND3 branches to the resident WTOR Purge routine. This routine removes elements from the buffer queue and from the reply queue that represent both messages to the operator and the operator's replies associated with the terminating task. The purpose of purging these elements from the queues is threefold: to save processing time, to prevent errors, and to prevent posting of meaningless ECBs for the communications task. These ECBs may not exist after ABEND6 frees dynamically acquired main storage.

Removing Requests for User (Asynchronous) Exit Routines: After purging the operator communication queues, ABEND3 branches to another subroutine to remove asynchronous exit requests. Those IQEs on the asynchronous exit queue that represent exit requests for the terminating task are dequeued. The elements will be freed later during ABEND6 when subpools of main storage are released. Note that IQEs are removed from the asynchronous exit queue but not from an IRB's queue if they have already been scheduled by the Stage 3 Exit Effector. The purpose of removing the IQEs from the queue is to minimize the scheduling of asynchronous exit routines that can occur after the "prevent asynchronous exits" flag is cleared at the end of ABEND3. The execution of an asynchronous exit routine, before ABEND processing is complete, can cause invalid recursion if the exit routine abnormally terminates. Such execution can also slow up the termination processing. (See "Scheduling of User Exit Routines" in Section 3, "Task Supervision.")

Releasing the Program Interruption Element (PIE): The release of the PIE, although occurring later in ABEND3 (after the validity checking of the main storage queues), will be discussed now, since the processing is functionally related to the release of other task resources.

ABEND3 tests whether a program interruption element (PIE) exists for the task. (If a PIE exists, its address appears in the TCBPIE field of the TCB, placed there earlier when the SPIE routine created the program interruption element.) If the PIE exists, ABEND3 branches to the FREEMAIN SVC routine to free the storage space.

Checking the Validity of the Main Storage Queues: After purging the asynchronous exit queue and before freeing the PIE's space, ABEND3 checks the validity of the addresses of free queue elements (FQEs), and checks the correctness of their length fields. It does this for all subpools belonging to the task currently selected by the "task select" subroutine. (The reader may recall that the "task select" subroutine selects in turn each task in the tree of tasks being terminated.) The purpose of checking the FQEs is to prevent program checks, with resulting recursions to the ABEND routine, during the later issuance of GETMAIN and FREEMAIN macro instructions.

Since FQEs are not in supervisor-protected areas of main storage, they may be altered by a user program, or by the ABEND routine itself during the performance of its "steal core" function. When the GETMAIN or FREEMAIN routine tries to gain access to an altered FQE to satisfy a request, the result is a program check. The need for the validity check of FQEs is thus apparent.

ABEND3, via the MSSLOOP subroutine, scans the subpool queue for a selected task, searching for descriptor queue elements (DQEs). ABEND3 examines all FQEs for each DQE for an owned subpool. It makes three general checks for each FQE. ABEND2 first examines the validity of the free-area address in the FQE. It determines if the address specifies a location that is on a fullword boundary, is within the bounds of main storage, and specifies a location that is in the area described by the associated DQE. ABEND3 next verifies that the free-area length specified in the FQE length field does not exceed the length described by the associated DQE. As a last test, ABEND3 verifies that the next FQE (pointed to by the FQE being examined) is at a higher main storage location than the FQE under examination.

ABEND3 nullifies the effect of an invalid FQE as follows. If a free-area address is invalid, the address is replaced with zero. If the length field is incorrect or if the next FQE is out of sequence, ABEND2 replaces with zero either the length field of the given FQE or the pointer to the next FQE. In this case, the examination of other FQEs associated with the same DQE stops, and continues with the FQEs belonging to the next DQE of the subpool. When all FQEs belonging to all subpool queue elements of the task have been examined and altered if need be, the validity check of the main storage queues is complete.

After the check of the main storage queues, the PIE (if it exists for the selected task) is freed, as previously described. When all tasks in the tree of terminating tasks (the specified task and its previously unterminated descendants) have been processed, and their resources freed or examined (as described), ABEND3 obtains main storage for its "close data sets" function. This processing and the remainder of ABEND3 are performed for both types of entry to the ABEND routine: first-time entry and a recursion entry.

Obtaining Main Storage for the "Close Data Sets" Function: ABEND3 next (via its "steal core" subroutine) tests whether storage is available for use during ABEND5 by the Close routine of data management. If sufficient storage is available (512 bytes), another module of the ABEND routine can be invoked to continue the termination procedures. But if the required storage area cannot be obtained, ABEND3 follows either of two paths, depending on whether the current termination is for the entire job step or only for the specified task and its descendants.

If the current termination is for the job step, ABEND3 "steals" (frees) previously allocated main storage for the Close routine, preferably from space allocated to the job step task. This space is in subpool 252. ABEND3 also sets a "prevent dump" indicator (TCBPDUMP) in the job step TCB to prevent dumps for all tasks of the job step during ABEND5. The prevention of dumps is necessary, since if space for the Close routine is not available, space needed to open the dump data set and space for a dump work area are also not available.

If the current termination is not for the job step task, the task termination must be converted to a job step termination. This is because the "stealing" of allocated main storage has made impossible the normal continuation of the job step. ABEND3 prepares for a job step termination. It does this by setting the top-task or "alternate" task pointer to the address of the job step TCB. It then branches to an early point in ABEND2 to again purge task resources, this time for an enlarged tree of tasks whose "top" task is the job step task. When the "steal core" subroutine is entered for the second time, the test for available main storage, and the "stealing" of needed storage (if necessary) occur just as they would for an original job step termination.

Before dealing with the processing that occurs in ABEND4, the present discussion will amplify the previous description of the process by which the availability of main storage is tested and unavailable storage is "stolen."

The "steal core" subroutine, used by ABEND3, tests for the availability of main storage for the Close routine, as follows. It issues a conditional GETMAIN macro instruction for 512 bytes of main storage. The availability or unavailability of this amount of storage is indicated by the code returned by the GETMAIN routine in the return code register. If the space is available, it is immediately freed, since the purpose of the GETMAIN macro instruction is merely to test availability. In this case, the main-line ABEND3 processing continues. The assumption is that if the space needed for the Close routine is available, additional space may also be available for the opening of the dump data set and the creation of an ABDUMP work area. (The opening of the dump data set occurs in ABEND4; the closing of data sets and the ABDUMP processing occurs in ABEND5.)

If main storage is not available (as indicated by the previous tests), the "steal core" subroutine tests whether the current termination is for the entire job step. If it is not, the conversion to a "step" termination and the purging of resources for the enlarged tree of tasks occurs as previously described. But if the termination is for the entire job step, the subroutine tries to obtain main storage for the Close routine of ABEND5 by attempting to "steal" allocated storage from one of the tasks of the job step.

The "steal core" subroutine next sets the "prevent dump" indicator (TCBPDUMP) in the job step TCB. ABEND5 will test this indicator to determine whether to invoke the ABDUMP routine. Dumps of any task in the job step must be prevented for two reasons:

- The freeing of any storage area allocated to the job step makes the results of a dump misleading.
- Main storage is stolen only for the ABEND5 function of closing data sets. If storage is not available for this function, additional space needed by the Open routine (in opening the dump data set), and space needed for an ABDUMP work area, are also not available.

The "steal core" subroutine tries to find previously allocated subpool-252 space belonging to the job step task. It tries to locate this space in preference to other subpools, since subpool 252 within a region never holds data control blocks (DCBs). Thus, the "stealing" of allocated space from this subpool will not cause recursions to the ABEND routine when ABEND5 refers to the DCBs to close data sets. The subroutine searches the subpool queue of the job step TCB, looking for a subpool queue element (SPQE) for subpool 252. If it finds such an SPQE before it exhausts the SPQE queue, it issues a FREEMAIN macro instruction to free the entire subpool, and tests for the availability of main storage. It makes this test by issuing a macro instruction conditional GETMAIN for 512 bytes, followed by a FREEMAIN macro instruction. If storage is now available (as indicated by the condition code returned by the GETMAIN routine), the work of the subroutine is finished.

But if the "steal core" subroutine cannot find an area assigned to subpool 252 that belongs to the job step task, it then looks for any allocated subpool that belongs to any task of the job step. By use of the "task select" subroutine and the MSSLOOP subroutine, the main storage queues of each task in the job step are searched for a descriptor queue element (DQE). If a DQE exists, main storage is already allocated to the associated subpool. The "steal core" subroutine places zero in the "free-queue element" pointer in the DQE. It does this to simulate an absence of free storage in the blocks described by the DQE. The purpose is to prevent the error of trying to free an area that is already free. To make storage available for the Close routine, the subroutine frees (via a branch to the FREEMAIN SVC routine) a block of 2K bytes of the area described by the DQE. Its work finished, the "steal core" subroutine returns control to ABEND3.

<u>Deciding the Next Module of ABEND to Be Invoked</u>: ABEND3 next clears the "prevent asynchronous exits" flag (TCBFX) in the current TCB. It does this to allow the Stage 3 Exit Effector to schedule asynchronous exit routines for system functions that may be needed during I/O operations in ABEND4.

After allowing the scheduling of I/O exit routines, ABEND3 performs tests to determine which module of the ABEND routine it should invoke: ABEND4 or ABEND5. ABEND4 is invoked if the current entry to the ABEND routine is either a first-time entry or is a recursion due to an error detected during the Open routine. If such a recursion has occurred, ABEND4 during its first execution set two flags in the current TCB: TCBREC and TCBOPEN. If neither flag is set, a first-time entry to the ABEND routine has occurred. ABEND5 is invoked if the current entry to the ABEND routine is a recursion because of an error detected in the Close routine or in the ABDUMP routine. In this case, ABEND5 during its first execution set two flags in the current TCB: TCBREC, and either TCBCLOSE or TCBADUMP.

<u>Processing During ABEND4 (Entry Point IGC0101C)</u>

ABEND4 performs several main functions. If a dump is requested, it opens the dump data set, if possible, in preparation for the dump to occur during ABEND5. If the data set can be opened, ABEND4 ensures that it remains accessible for the duration of the associated job step. If, however, the current entry to the ABEND routine is a recursion because of an error during the Open routine, ABEND4 sets the "prevent dump" indicator in the job step TCB. It does this to prevent abnormal termination dumps for the entire job step.

ABEND4 also releases programs that are in the process of being loaded for nondispatchable tasks of the job step, so that they will be available for other requestors. Of particular interest are the ABDUMP and BSAM routines, which are needed for ABEND processing.

Lastly, ABEND4 determines whether processing of the current ABEND request should continue, or be stopped to allow a concurrent ABEND request to be processed. The concurrent request, if for a higher level task (e.g., the parent of the current task), would permit the termination of a larger tree of tasks than does the current request.

To accomplish these purposes, ABEND4 performs the following main functions:

- Determines whether to open the dump data set (SYSABEND or SYSUDUMP).
- Releases partially loaded modules.
- Prepares to open the dump data set.
- Ensures that the dump data set remains open for the duration of the job step.
- Indicates whether the dump data set has been opened.
- Prepares to restart deferred terminations.
- Determines whether the current termination should continue.
- Ends its processing by either branching to the Dispatcher or invoking ABEND5.

DETERMINING WHETHER TO OPEN THE DUMP DATA SET: ABEND4 performs tests to determine whether it should attempt to open the dump data set (SYSABEND or SYSUDUMP) for use during ABEND5. (The SYSUDUMP data set, if allocated, permits the dump of the entire job-step region of main storage.) These tests determine:

- Whether the current entry to ABEND4 is due to recursion.
- Whether the "prevent dump" indicator is set.
- Whether the dump data set is already in the process of being opened for another task in the job step.
- Whether the caller of the ABEND routine has requested a dump.
- Whether the SYSUDUMP data set was originally allocated by means of a DD card.
- Whether the SYSABEND data set was originally allocated by means of a DD card.
- Whether the dump data set was previously opened for another task in the job step.

Determining if the Entry to ABEND4 is Due to Recursion: If the entry to ABEND4 is due to recursion, the TCBREC and TCBOPEN flags were set in the current TCB by ABEND4. A previous execution of the Open routine of data management for this task produced an error. The error caused a reentry to the ABEND routine. Since the previous attempt to open the dump data set for the current task failed, ABEND4 does not try again. Instead, it sets the "prevent dump" indicator (TCBPDUMP) in the job step TCB to prevent abnormal termination dumps for any task in the job step. ABEND5, finding this flag set, will bypass the dump portion of its processing.

Determining if the "Prevent Dump" Indicator Is Set: If the entry is not a recursion, ABEND4 next tests if the "prevent dump" indicator (TCBPDUMP) is set in the job step TCB. The indicator may have been set for any of the following reasons:

- The test for availability of main storage failed during ABEND3. (Both the Open and the ABDUMP routines need storage for their processing.)
- The Open routine could not open the dump data set (SYSABEND or SYSUDUMP) for a previously terminating task in the same job step.
- An error that caused entry to the ABEND routine occurred while the ABDUMP routine was being executed for another task in the same job step.

If the "prevent dump" indicator is set, ABEND4 cannot prepare for dumps. It therefore bypasses the rest of its processing by invoking ABEND5 to close the data sets used by the terminating tasks.

Determining if the Dump Data Set is Already Being Opened: If the "prevent dump" indicator is not set in the job step TCB, ABEND4 tests whether the dump data set is already in the process of being opened for another task in the job step. (The test checks the TCBFOINP flag in the job step TCB.) If an open procedure is already in process, a new OPEN macro instruction is not issued. Instead, ABEND4 defers the current ABEND request. It does this by setting the SVRB RB old PSW for reentry to ABEND4, and making the current task nondispatchable (setting the TCBONDSP flag) until the open procedure is complete. It allows the system to continue the execution of routines for other tasks by preparing for a task switch and branching to the Dispatcher. It signals the Dispatcher to search the TCB queue for a ready task by storing zero in the "new" TCB pointer (IEATCBP). At a future execution of ABEND4, the task for which the OPEN macro instruction was issued will clear the non-dispatchability flag of the "waiting" task, regardless of whether the open procedure was successful.

Note: The release of partially loaded modules occurs next. Since it is unrelated to the opening of the dump data set, it will be discussed in a later topic.

Testing the Dump Request and the Status of the Dump Data Set: ABEND4 makes three tests to determine if it should open the dump data set, or whether it should invoke ABEND5 immediately. Two of these tests check whether a dump cannot or should not occur. The third test determines if the dump data set has already been opened for another task of the job step.

One test determines whether a dump was requested by the caller of the ABEND routine. ABEND4 tests the high-order bit of the completion code in the TCBCMP field of the current TCB. Another test examines the task I/O table (TIOT) to determine if a SYSABEND or SYSUDUMP DD card was recognized by the Reader/Interpreter of the Job Scheduler, and thus whether the SYSABEND or SYSUDUMP data set was allocated by the Job Scheduler. If the caller did not request a dump, or if neither data set was allocated, ABEND4 sets the "prevent dump" indicator in the job step TCB and invokes ABEND5. In this case, ABEND5 will close data sets without causing a dump.

The remaining test determines if the dump data set (SYSABEND or SYSUDUMP) has already been opened for another task of the job step, and therefore whether the open function may be bypassed. (The test checks the TCBDSOP flag in the job step TCB.) If the data set was previously opened, ABEND4 obtains the associated DCB address, needed by ABEND5, by searching the DEB queue of the job step task for the DEB belonging to the dump data set. It then extracts the DCB address from the DEB for use during I/O operations of the dump procedure of ABEND5. ABEND4 sets a special identifier bit in the DEB.

After obtaining the DCB address, ABEND4 invokes ABEND5 to perform both its major functions.

RELEASING OF PARTIALLY LOADED MODULES: If the open procedure is not in process, before making the foregoing three tests, ABEND4 releases "partially loaded" modules for terminating tasks. Such modules are in the process of being loaded for the current task or for other tasks that are being abnormally terminated (TCBABWF flag set).

When a task is set permanently nondispatchable (TCBABWF flag set), the Contents Supervision routines do not complete the loading of a module requested for the task. The routines also do not begin a new fetch of a module whose loading process has started. Other requestors waiting for the module cannot gain access to it. Of primary interest are the modules containing the ABDUMP and BSAM routines, needed by ABEND5. These modules may have been requested by a subtask of a task that is now terminating. Since ABEND2 may have placed the subtask in the abnormal wait state (TCBABWF flag set), the routines may be permanently unavailable. The problem of "frozen" partially loaded modules is solved by the "release" routine of ABEND4, called PARRLSE.

For each module in process of being loaded for a terminating task, the PARRLSE routine performs the following purge functions:

- Frees the module's extent list and program area, if they exist.
- Removes from the job pack area queue one or more contents directory entries (CDEs), which represent the partially loaded module, and frees the space they occupy.
- If there are other requestors which are awaiting the loading of the module, prepares one or more RBs for reentry to Contents Supervision at CDCONTRL to refetch the module for another task.

The PARRLSE routine searches the job pack area queue for CDEs whose modules are in the process of being loaded for a terminating task. (The CDENIC flag was set in each CDE whose module is being loaded.) For each CDE whose module is being loaded, the purge functions are performed if either of two requirements is met:

- The loading was initiated for the current task, for which ABEND4 is being executed. In this case, the current task is being terminated and its I/O operations have already been purged by ABEND2.
- The loading was initiated for another task which is being abnormally terminated (TCBABWF flag set), and whose I/O operations, initiated for loading, have already been purged by ABEND3 (TCBFA flag set).

The CDE, its extent list, and program area may not be freed until the task's I/O operations have been purged by ABEND3. Otherwise, the Main Storage Supervision routines may reallocate for another task the freed program area. The requested module may later arrive in main storage, overlaying the reallocated program area, which now belongs to another task.

PREPARING TO OPEN THE DUMP DATA SET: Preparation for the opening of the dump data set occurs if the following conditions exist, as indicated by the previously described tests: dumps have not been prevented by ABEND3, an open procedure is not already in process, a dump was requested by the caller, the data set was not previously opened for another task of the job step, and the data set was allocated by the Job Scheduler.

If all these conditions exist, three flags are set and space is obtained for a preassembled DCB. The flags indicate the status of the current ABEND request. One, the "open in process" flag (TCBOINP), is set in the job step TCB to indicate that the dump data set is in the process of being opened. If another termination should occur before the open procedure is finished and the flag is reset, the new terminating task is set "open nondispatchable" (i.e., the termination is deferred) until the open process is complete. (The TCBONDSP flag is set in the new requestor's TCB.) The other two flags (TCBOPEN and TCBREC) indicate a valid recursion to the ABEND routine, if an error occurs during the execution of the GETMAIN or Open routine (soon to be executed). After setting the three flags, ABEND4 issues a GETMAIN macro instruction to obtain space for a preassembled DCB and moves the DCB to the space. This DCB will be specified in the OPEN macro instruction during ABEND4 and in the I/O macro instructions used by the ABDUMP routine during ABEND5.

ENSURING THAT THE DUMP DATA SET REMAINS OPEN: Both before issuing the OPEN macro instruction, and after the execution of the Open routine (if the open request has been successful), ABEND4 tries to ensure that the dump data set will remain "open" for the remainder of the job step. By "open" is meant the retention of the BSAM access method routines in main storage and the retention of the DEB and DCB for the dump data set. Special efforts are needed to keep the dump data set open, since ABEND5 will close data sets belonging to the terminating tree of tasks, and ABEND6 will free the load lists belonging to these tasks and the associated program areas (if there are no outstanding requests for the programs). Unless special precautions are taken, the DEB for the dump data set will be removed and freed from its DEB list during ABEND5, and the BSAM routines possibly released during ABEND6. With the dump data set no longer "open", further abnormal dumps during the remaining life of the job step would not be possible. Repeated opening of the data set, after it has been initially opened, is avoided for two reasons: such repetition would waste time, and each reissue of the OPEN macro instruction (if acted upon) could possibly reposition the data set volume undesirably (depending on the DISP operand of the user's DD card -- see Job Control Language publication).

ABEND4 does two things to preserve the "open" status of the dump data set:

1. It prevents the deletion of the BSAM routines by forcing the creation of new load list elements for them. It queues the new load list element to the load list for the job step TCB.

2. It places the DEB for the dump data set on the DEB queue belonging to the job step TCB.

These two steps will be discussed separately in the following paragraphs.

ABEND4 prevents deletion of the BSAM routines from main storage by saving the load list pointer (TCBLLS) for the current task and replacing the pointer with zeros. When ABEND4 issues the OPEN macro instruction, the Open routine requests the loading of the BSAM module. Since the Contents Supervision routines cannot find load list elements representing the BSAM routines on the current task's load list (the zeroed load list pointer indicates that there is no load list), they create new load list elements for the BSAM routines, and place the load list pointer (TCBLLS) in the current TCB. After ABEND4 issues the OPEN macro instruction, it places the newly created load list elements for BSAM on the load list belonging to the job step TCB. Thereafter, the BSAM routines cannot be deleted from main storage by the Close routine of data management, until data sets belonging to the job step TCB are closed at normal or abnormal step termination. ABEND4 then issues the OPEN macro instruction. Regardless of whether the data set is actually "opened", ABEND4 restores the original load list pointer (TCBLLS) to the current TCB. If a recursion entry to the ABEND routine occurs because of an error during the execution of the Open routine, the load list pointer is also restored. It is placed in the TCB that was current during the previous execution of ABEND4.

If the open attempt is successful (as indicated by a flag in the DCB of the dump data set), ABEND4 uniquely labels the data extent block (DEB) associated with the dump data set. ABEND4 uniquely labels the DEB so that it can later find the DEB and obtain from it the associated DCB address. ABEND4 will pass the DCB address to ABEND5. The position of the DEB in the DEB queue can vary because of processing by the End-of-Volume (EOV) data management routine.

ABEND4 then places the DEB on the job step task's DEB queue. It does this to prevent ABEND5 from closing the dump data set, when it closes data sets belonging to the terminating tree of tasks.

INDICATING WHETHER THE DUMP DATA SET HAS BEEN OPENED: When the request to open the dump data set has been issued and the Open routine of data management has been executed, ABEND4 tests the appropriate flag of the DCB to learn if the data set has been actually opened. According to the results of the test, ABEND4 indicates, via flag bits in the job step TCB and in the current TCB, whether the open request has been successful.

If the data set could not be opened, and therefore abnormal dumps are not possible, ABEND4 sets the "prevent dump" indicator (TCBPDUMP) in the job step TCB. This action prevents ABEND5 from invoking the ABDUMP routine to perform an abnormal dump for any task of the job step. Although abnormal dumps requested by ABEND5 are prevented for the rest of the job step, dynamic dumps requested via a SNAP macro instruction for unterminated tasks of the job step can be performed.

If the dump data set was actually opened, ABEND4 manipulates two other indicators in the job step TCB. It sets the "data set open" indicator (TCBDSOP) so that two types of ABEND requests (first-time terminations and restarted deferred terminations) may bypass much of ABEND4 processing and invoke ABEND5. Deferred terminations are first-time ABEND terminations that find an "open in process" and are set nondispatchable (TCBONDSP), awaiting the completion of the open process. (A later phase of ABEND4 will clear "open nondispatchability" for deferred terminations.) Besides setting the "data set open" indicator, ABEND4 clears the "open in process" flag (TCBOINP) to prevent new or restarted terminations from being deferred.

Regardless of whether the dump data set could be opened, ABEND4 clears the "open" and "recursion" indicators (TCBOPEN and TCBREC) in the current TCB. It does this to indicate that any new recursion is not due to open processing and is invalid.

PREPARATION FOR THE RESTART OF DEFERRED TERMINATIONS: Deferred terminations await the completion of the open procedure that was initiated for another terminating task. Deferred terminations are prepared for restart under any of three conditions: a successful open procedure, an unsuccessful open procedure, or a recursion to the ABEND routine because of an error during the execution of the Open routine. However, only if the open procedure is successful can abnormal dumps occur during the job step. If the open procedure is unsuccessful, a permanent error probably exists (such as an incorrectly specified data set), and further attempts to open the dump data set are useless.

In all three cases, ABEND4 (via the "task select" routine) examines all TCBs in the job step, clearing any "open nondispatchable" flags (TCBONDSP) that it finds. ABEND4 must examine all TCBs in the job step, since it has no record of which tasks represent deferred terminations. After the flag is reset in any TCB, the supervisor's Task Switching routine is invoked to determine if the associated task is ready and can, by virtue of its dispatching priority, replace the current task at the next entry to the Dispatcher. (A task is ready if no non-dispatchability flag is set in its TCB, and its current RB has a wait count (RBWCF) of zero.)

DETERMINING WHETHER THE CURRENT TERMINATION SHOULD CONTINUE: After preparing to restart deferred terminations, ABEND4 tests whether the current termination should continue, or whether it should be stopped to allow one of the deferred terminations to proceed. The deferred termination is allowed to proceed if it is for a direct ancestor of the current task. The advantage of the higher level termination is that a larger array of resources is freed, including the resources of the task whose termination is stopped (see Figure 10-5).

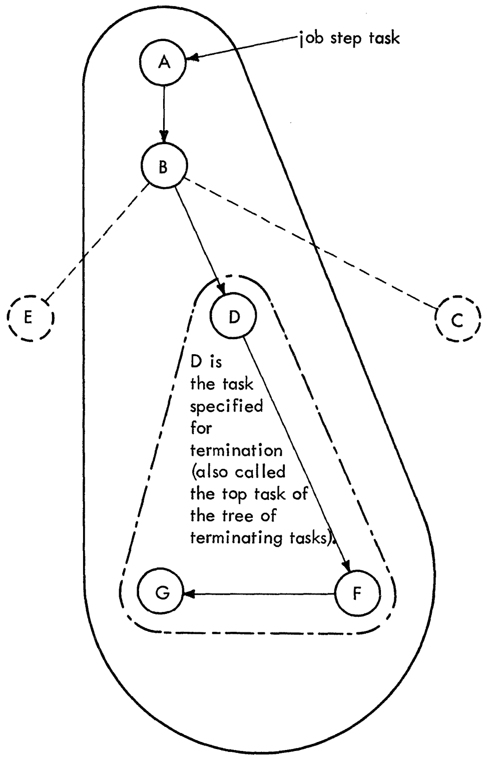
As a result of its tests, ABEND4 either continues the current termination, or stops it in favor of the terminating ancestor. It does this by setting the current task nondispatchable and branching to the Dispatcher. The Dispatcher causes reentry to the ABEND routine for the higher level task, if its dispatching priority exceeds that of the current task.

The circumstances in which ABEND4 continues the termination of the current task, or discontinues its termination in favor of its terminating ancestor, will now be described.

ABEND4 continues the termination of the current task for two cases:

1. The current task is the "top" task in a tree of terminating tasks, as indicated by the "top" flag (TCBFT) in its TCB. In this case, no direct ancestor is terminating.

2. A "step" termination has been requested (the "top" flag is set in the job step TCB), but the requesting routine does not belong to the job step task. The indication of this


job step task
A
B
E
C
D
D is the task specified for termination (also called the top task of the tree of terminating tasks)
G
F

Legend:

○ Represents a task

→ Represents a pointer

Represents the direct line of ancestors and descendants for task D within the job step.

Represents a "tree" of terminating tasks whose top task is D.

Notes:

1. Tasks shown by dashed lines are not direct ancestors of task D.
2. The job step consists of all the tasks shown in the figure.
3. The figure shows a tree of tasks during a "task" termination, as opposed to a "step" termination. During a step termination the job step task is the top task of the tree, and all tasks of the job step belong to the tree.

Figure 10-5. Task Relationships During an Abnormal Termination

condition is that the ABEND routine's SVRB is not queued to the job step TCB. ABEND4 must continue the current task termination, since the ABEND routine is controlled by one SVRB which may not now be queued to another TCB. (Such shifting of the SVRB can occur during ABEND5.)

ABEND4 discontinues the termination of the current task in favor of its terminating ancestor for two other cases:

1. A direct ancestor of the current task has requested a "task" termination, but this ancestor is not the job step task. As evidence of this condition, the "top" flag (TCBFT) has not been set in either the current TCB or in the job step TCB.

2. A routine belonging to the job step task has requested a termination (either step or task option). In this case, the higher level ancestor whose termination may proceed is the job step task, and a step termination is performed. The evidence of this condition is that the "top" flag is set in the job step TCB, and the ABEND routine's SVRB (with its unique identifier) is queued to the job step TCB. This condition involves two known SVRBs for the ABEND routine: the one belonging to the current task and the one generated for the job step task. The latter SVRB will be used to continue the step termination; the SVRB for the stopped current task will be released during ABEND5.

CONCLUSION OF ABEND4: ABEND4 processing is completed either by a branch to the Dispatcher, if termination of the current task must be stopped (as just described), or by invocation of ABEND5 for the current task if its termination will continue. ABEND5 is invoked according to the results of tests already described in the topic "Determining Whether to Open the Dump Data Set".

Processing During ABEND5 (Entry Point IGC0201C)

ABEND5 has four main purposes:

- Determining the scope of ABEND5 processing.
- Performing ABDUMP processing. This includes the dump of resources belonging to the terminating tasks of the tree (i.e., the specified task and its descendants).
- Closing data sets that belong to the terminating tasks of the tree. This includes the purge of associated queues of data extent blocks (DEBs).
- Removing request blocks (SVRBs) belonging to transient SVC routines that are being executed for the terminating tasks. The SVRBs are purged as the first part of a general RB removal begun in ABEND5 and finished in ABEND6. The SVRBs are removed from the RB queues of their respective TCBs and from the transient area queues, and their space is then freed.

ABEND5 uses a number of SVC routines to perform its functions. Of primary importance are the ABDUMP routine and the Close routine of data management. For each invocation, ABDUMP displays the resources of the selected task. ABEND5 invokes the ABDUMP routine separately for the "top" task, its descendants, and its direct ancestors. The Close routine closes data sets and purges the DEB queues.

There are two types of entries to ABEND5, just as there were to ABEND2, ABEND3, and to ABEND4: first-time entries and recursion entries. A first-time entry represents a first-time request for the abnormal termination of a task. It occurs via an XCTL macro instruction from ABEND4. A recursion entry represents a request for abnormal termination generated by either the ABDUMP or Close routine because of an error detected during its processing. A recursion entry to ABEND5 is always made directly from ABEND3, via an XCTL macro instruction.

The scope of ABEND5 processing varies, depending on the particular type of entry. A first-time entry permits ABEND5 to perform all its functions: causing dumps, closing data sets and purging associated DEBs, and dequeuing and freeing SVRBs for transient SVC routines. A recursion because of an ABDUMP error causes the bypassing of the dump function, but permits all other functions to be performed. Lastly, if a recursion occurs because of an error detected during the Close routine, the dump function is bypassed, since it may already have been performed, and the closing of data sets and the purging of DEBs continue from the point of error. As with other types of entries, SVRBs for transient routines are also purged.

DETERMINING THE SCOPE OF PROCESSING: Before ABEND5 can perform any of its major functions, it must test certain indicators to learn the type of processing that it must perform. It first tests for a recursion because of an error during ABDUMP processing or during close processing. (Either the ABDUMP or Close routine can request a reentry to ABEND if it discovers an error. For example, the Close routine may find a DCB that has been overlaid by a user program.)

If a recursion has occurred from the Close routine (as indicated by the "set" condition of the "Close recursion" flag TCBCLOSE in the current TCB), ABEND5 performs special processing in order to continue the closing of data sets and purging of DEBs. The special processing will be described under "Special Handling of a Recursion".

If a recursion has occurred from the ABDUMP routine (as indicated by the "set" condition of the "ABDUMP recursion" flag TCBADUMP in the current TCB), ABEND5 resets the flag to avoid a false indication, and sets the "prevent dump" indicator (TCBPDUMP) in the job step TCB to prevent abnormal dumps now and for the remainder of the job step. This is necessary because a serious error detected during ABDUMP processing has made unreliable further dumps on the same data set.

If no recursion from the ABDUMP routine has occurred, ABEND5 makes two other tests before performing ABDUMP processing. Either of these tests can cause the bypassing of ABDUMP processing. The first test examines the "prevent dump" indicator in the job step TCB to learn if ABEND3 or ABEND4 had discovered an abnormal condition. (ABEND3 sets the indicator if main storage is not available and must be "stolen". ABEND4 sets the indicator if the dump data set has not been allocated by the Job Scheduler, or if the data set cannot be opened.)

If the "prevent dump" indicator is not set, ABEND5 tests the DCB address passed by ABEND4 in a register. If the address is zero, the caller of the ABEND routine has not requested a dump. Accordingly, ABEND5 bypasses the ABDUMP processing and closes data sets. But if the address is not zero, the DCB register contains the address of the DCB that ABEND4 used to open the dump data set.

If the "prevent-dump" indicator is set, or if the DCB address is zero, dumps for the current tree of terminating tasks is bypassed. There is, however, an important distinction between the meaning of the two indicators. If the DCB address is zero, abnormal dumps are bypassed only for the processing of the current ABEND request, since the ABEND routine's caller has not requested dumps. But if the "prevent dump" indicator is set, no abnormal dump will be performed for the remainder of the current job step, since an error exists, possibly associated with the dump data set for this job step.

If the "prevent dump" indicator is not set, and the DCB address passed to ABEND5 is not zero, ABDUMP processing is performed as described in the following section.

PERFORMING ABDUMP PROCESSING: The ABDUMP processing consists of three functional parts: preparation for the dumps, performance of the dumps, and a cleanup procedure after the dumps. Each of the functional parts will be explained separately.

Preparation for the Dumps: Before ABEND5 can issue a SNAP macro instruction to invoke ABDUMP for each task to be dumped, it must take certain precautions. To prevent repetitious loading, it ensures that the ABDUMP routine's "resident" module (IGC0A05A) remains in main storage throughout the series of dumps for the current task, its descendants, and its ancestors. Otherwise, the ABDUMP routine would load its resident module before each dump, and delete the module after each dump. ABEND5 issues a LOAD macro instruction to load the resident module before it invokes the ABDUMP routine for the first task, and deletes the module after the ABDUMP routine has been executed for the last task of the tree. ABEND5 thus prevents the actual reloading and deletion of the resident module by the ABDUMP routine each time that it is entered. (Although the ABDUMP routine issues a LOAD macro instruction to load the resident module, no loading occurs since the module is already in main storage.)

As another precaution, ABEND5 ensures that the dumps associated with one ABEND request will appear consecutively on the dump data set, not interspersed with dumps for a concurrently terminating tree in the same job step. To prevent such interleaved dumps, ABEND5 issues an ENQ macro instruction (with the "exclusive" option) for the dump data set before the first ABDUMP execution and issues a DEQ macro instruction after the last ABDUMP execution.

The ENQ routine, besides its normal processing, performs a special service for ABEND5. It makes possible the servicing of the currently issued ENQ request for the dump data set. Such action is necessary if a subtask of the current task was previously abnormally terminated. The data set may still be enqueued for the previously requested dump of the subtask's resources. In this case, the servicing of the current ENQ request would await the issuance of a DEQ macro instruction by ABEND5, when the dump of the subtask's resources is complete. But since the subtask is now nondispatchable (its TCBABWF flag set during ABEND2), the DEQ macro instruction cannot be issued by ABEND5 for the subtask.

When the ENQ routine detects that the ABEND routine is the caller, it removes from the resource queues, via its "autopurge" subroutine, all queue elements belonging to the current task and any of its subtasks. The current ENQ request issued by ABEND5 can then be serviced.

After return of control from the ENQ routine, ABEND5 again tests the "prevent dump" indicator (TCBPDUMP) in the job step TCB. This test is necessary because while the current task was enqueued and waiting for the data set, ABEND5, executed for another task in the job step, may have responded to an error in ABDUMP processing by setting the "prevent dump" indicator. If the indicator is set, the ABDUMP routine is not invoked, and ABEND5 dequeues the dump data set and deletes ABDUMP's resident module. The remaining processing in ABEND5 (closing of data sets, purging of DEBs and SVRBs for transient routines) is the same as if dumps had originally been prohibited or not requested.

<u>Performing the Dumps</u>: Dumps of the terminating tasks of the tree are made if the following conditions have been met, tested by both ABEND4 and ABEND5: a dump data set has been provided (as indicated by a search of the TIOT by ABEND4), a preassembled DCB can be opened (also by ABEND4), the caller of ABEND has requested dumps, and no last-minute ABDUMP errors have been detected by ABEND5.

Via issuance of the SNAP macro instruction (SVC51), ABEND5 invokes the ABDUMP routine separately for each task whose resources are to be displayed. The current task is dumped first, then its descendants, then its direct ancestors, including the job step task (see Figure 10-5).

Each task of the tree of tasks (D, G, and F in Figure 10-5) is selected by means of the "task select" (TASKSEL) subroutine. As each task is selected by the subroutine, ABEND5 tests the "S" flag (TCBFS) in its TCB to determine if the task's resources have already been dumped. If the "S" flag is set, the resources have already been dumped, and the next task is selected. For each task that has not already been dumped, ABEND5 issues a SNAP macro instruction (SVC 51) to dump the task's resources. The operands of the macro instruction include the address of the selected TCB, the DCB address received from ABEND4, and the fact that the ABEND routine is the caller (via a bit that is set in the ABEND routine's SVRB). On each return of control from the ABDUMP routine for a subtask, ABEND5 sets the "S" flag in the subtask TCB to indicate that the subtask's resources have been dumped.

The ABDUMP routine displays for the current task the following resources: job name, step name, date, time, completion code, PSW at entry to ABEND, TCB, RBs, load list, CDEs, extent lists, TIOT, DEBs, SPQEs, DQEs, FQEs, PQEs, FBQEs, save area trace, QCBs, address of the last point of interruption (old PSW), register contents at entry to ABEND, the nucleus, load modules, and the subpool blocks.

The resources displayed for the subtasks and ancestors of the current task do not include the following items: PSW at entry to ABEND, register contents at entry to ABEND, the nucleus, load modules, and the subpool blocks. A subtask dump is identified by the number 001, that of an ancestor by the number 002.

Cleanup After the Dumps: After the dumps of the ancestors, ABEND5 performs several cleanup steps. It first dequeues the dump data set so that it is available for use by the next requestor of the ABEND routine. The parameters of the DEQ macro instruction are obtained from the extended save area of the ABEND routine's SVRB, where they were stored when the ENQ macro instruction was issued. Next, ABEND5 deletes the resident module of ABDUMP (IGC0A05A), so that its space, no longer needed for the current dumps, may be freed for other use. (Although the ABDUMP routine has already issued a DELETE macro instruction for the same module, it is ineffective in releasing it, since the ABEND routine's request for the module is still outstanding.) When ABEND5 issues the DELETE macro instruction, the Delete routine decreases the CDE use/ responsibility count. If the count is now zero, the Delete routine releases the space occupied by the module, its load list element, CDE, and extent list.

After dequeuing the dump data set[1] and issuing a DELETE macro instruction for ABDUMP's resident module, ABEND5 clears the ABDUMP and recursion flags in the current TCB. These had been set to indicate a valid recursion if an error had occurred during ABDUMP processing. Since this processing is finished, the bits are reset. This action completes the post-dump cleanup procedure.

CLOSING DATA SETS THAT BELONG TO THE TERMINATING TASKS: Data sets that are opened during task operation are normally closed (if they are not already closed) by the End-of-Task (EOT) routine. But since terminating tasks cannot reach the end-of-task condition, the ABEND routine must close all their data sets that are still open.

The closing of data sets and the purging of SVRBs are performed separately for each task of the tree while the task is currently active. The "task select" subroutine (TASKSEL) selects the tasks, one at a time, starting with the newest descendant of the task specified for termination. The previously active task is set nondispatchable, the newly selected task is made ready, the ABEND routines's SVRB is queued to the selected task's RB queue, a task switch is invoked, and a branch is made to the Dispatcher. When ABEND5 is dispatched for the selected task it closes data sets, purges DEBs, and removes SVRBs belonging to transient routines. When all such SVRBs have been purged, the TASKSEL sub routine selects the next higher level task in the tree, and the process is repeated.

Selecting Each Task of the Tree: On each iteration of the loop in which data sets are closed and SVRBs are purged, the TASKSEL subroutine selects another task of the terminating tree. Its first selection is the newest descendant of the task specified for termination. For example, in Figure 10-4 the newest descendant is task G, and the task specified for termination is task D. On each iteration, the next higher level task is selected. The next higher level task is F. On the final iteration of the loop, the highest level or "top" task of the tree is selected. The top task of the tree is D. The direction of selection is thus from bottom to top of the tree.

For each selection, the TASKSEL subroutine tests the "completion" flag (TCBFC) to learn whether the task has already been terminated (either normally or abnormally). If the selected task has already been terminated, and its TCB is thus no longer needed, ABEND5 branches to the resident "erase" routine (part of the EOT routine) with the address of the terminated TCB. The "erase" routine dequeues the selected TCB from the subtask queues (whose pointers are TCBLTC and TCBNTC in each TCB) and frees the space it occupies. On return of control from the "erase" routine, the TASKSEL subroutine makes another selection and again checks the "completion" flag. When an incomplete or not-already terminated task has been selected, the next part of ABEND5 is prepared for dispatching under control of the selected TCB.

Preparation for the Dispatching of ABEND5 Under Control of the Selected Task: As stated before, the closing of data sets and the purging of SVRBs are done under the control of the task to which these resources belong. For this purpose, the ABEND routine's SVRB must be queued to the selected task's RB queue. The selected task must then become the active task, replacing that which was previously current. Under control of the newly selected TCB, ABEND5 is redispatched (at location ENTRY2) to begin execution of its "close data sets" function.

Preparation for the redispatching of ABEND5 occurs as follows. First, the current task is set nondispatchable (TCBABWF flag is set) so that ABEND5 temporarily cannot be redispatched for this task. The task selected by the TASKSEL subroutine is then made dispatchable (two bytes of nondispatchability flags are cleared in its TCB) in preparation for the branch to the Dispatcher. ABEND5 stores in the RB old

----

[1] The dump data set can be specified as SYSABEND or SYSUDUMP.

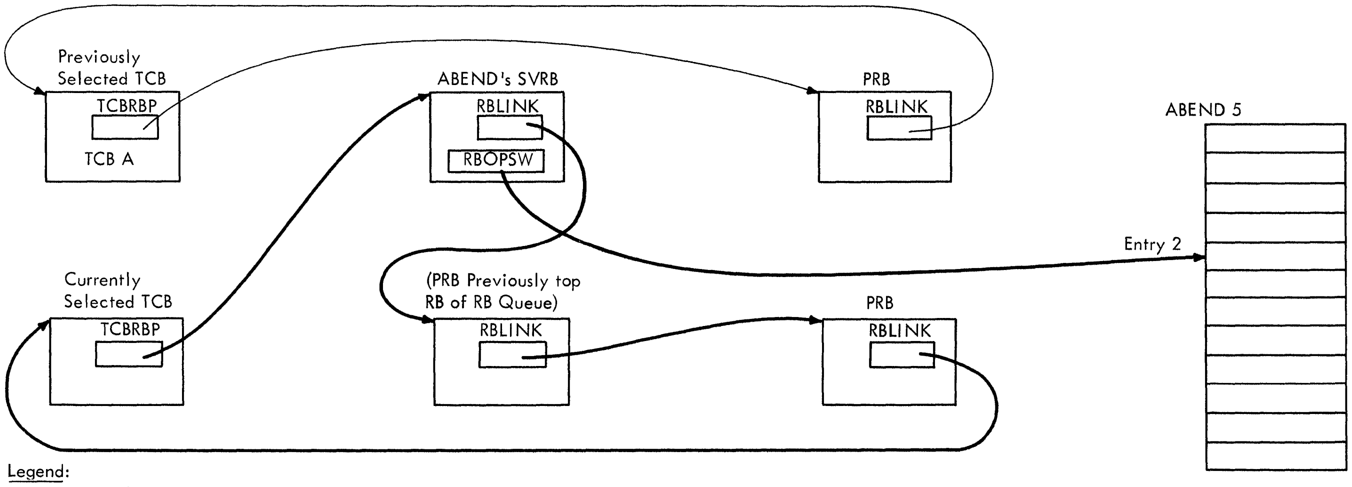
PSW field (RBOPSW) of ABEND's SVRB the entry point (ENTRY2) to its "close data sets" function, for later use by the Dispatcher. Then, to permit ABEND5 to control processing for the selected task, the ABEND routine's SVRB is placed at the head of the selected task's RB queue. (The TCBRBP field of the selected TCB is altered to point to the ABEND routine's SVRB, and the ABEND routine's SVRB points to the previously "top" RB of the queue. (Refer to Figure 10-6.) The ABEND routine's SVRB is then removed from the RB queue of the previously current TCB, since ABEND5 can service only one task at a time.

To ensure that the registers will contain the correct values when ABEND5 is redispatched for the selected task, the address of the selected TCB is placed in the TCB register (register 4), and the current general register contents are stored in the register save area (TCBGRS) of the selected TCB. The Dispatcher, when invoked, will load the registers from this TCB save area.

ABEND5 next branches to the Task Switching routine, making available the address of the selected TCB. The Task Switching routine compares the dispatching priority of the input TCB with the dispatching priority of the last dispatched (previously current) TCB. If the selected task is of higher priority, the Task Switching routine places the selected TCB address in the "new" TCB pointer (IEATCBP) as an indication to the Dispatcher. Otherwise, the Dispatcher would try to select either the previously current task (now nondispatchable), or another lower priority task by a search of the TCB queue in a downward-priority direction.

Finally, ABEND5 branches to the Dispatcher to pass control to ABEND's "close data sets" function for the selected task. When this task has highest priority among the ready tasks (perhaps after a delay in which other tasks are active), ABEND5 is redispatched (at location ENTRY2) to purge data sets and SVRBs for the selected task.

Closing Data Sets for a Selected Task: The closing of data sets for a selected task consists of the issuing of a CLOSE macro instruction (with resulting supervisor linkage to the Close routine of data management) for each data set opened for the task. Each data set is specified by a DCB whose address is contained in a DEB whose queue belongs to the task. After a data set is closed, its associated DEB is removed from the task's DEB queue, and its space is freed. If a recursion to the ABEND routine occurs because of a defective DCB, or an incorrect DEB address in a DCB, the DEB is dequeued and freed, although its data set is not closed. When all DEBs on the queue have been dequeued and freed, ABEND5 branches to its SVRB purge function to release SVRBs representing transient routines. After all such SVRBs have been released, the "task select" (TASKSEL) subroutine selects the next higher level task for similar processing, unless the top task has just been processed. When the top task of the terminating tree has been processed, ABEND5 invokes ABEND6 to continue the purge of resources.



Legend:
——► = pointer

Note: ABEND's SVRB is shifted to the RB aueue of the currentlv selected task.

• Figure 10-6. Preparation for the Dispatching of ABEND5 for the Selected Task

The closing of data sets and the DEB purge will now be discussed in greater detail.

If the pointer to the task's DEB queue (TCBDEB) is not zero, there are data sets belonging to the task that must be closed. To prepare for the issuance of CLOSE macro instructions, ABEND5 stores in the parameter list for the Close routine the flag byte, and the DCB address which was obtained from the first DEB. The parameter list is the second word of the extended save area of the ABEND routine's SVRB. (See SVRB format in Section 12, "Control Blocks and Tables.") The high-order bit of the parameter list is set to indicate to the Close routine that the specified data set is the last in a list of data sets. (See Supervisor and Data Management Macro Instructions.)

After setting up parameters for the Close routine, ABEND5 saves the current DEB address in the extended save area of the ABEND routine's SVRB. The purpose is to check whether the Close routine is able to perform its secondary functions: the updating of the DEB pointer (TCBDEB), and the freeing of the current DEB. After regaining control from the Close routine, ABEND5 will compare the DEB pointer with the saved DEB address to determine if the Close routine has both removed the current DEB from the DEB queue and freed its space.

Next, before invoking the Close routine, ABEND5 sets the "Close" and "recursion" flags (TCBCLOSE and TCBREC) in the selected TCB. If an error occurs during the Close routine (possibly caused by an invalid DCB), the set condition of these indicators indicates a valid recursion to the ABEND routine, and causes reentry to ABEND5 from ABEND2 to continue the DEB processing.

ABEND5 invokes the Close routine of data management by issuing a CLOSE macro instruction specifying the parameters it had previously stored in its SVRB. The resultant SVC interruption causes the SVC Second-Level Interruption Handler to create an SVRB for the Close routine and to place it on the RB queue of the selected task. When the Close routine finishes its processing, the resultant SVC interruption causes supervisor linkage to the Exit routine, which removes the Close routine's SVRB from the RB queue, and frees its space. The ABEND routine's SVRB is then left as the current RB for the task.

After the execution of the Close routine, the Dispatcher returns control to ABEND5 as the current routine for the still-active current task. ABEND5, to indicate that a recursion (if it now occurs) is not valid, resets the "Close" and "recursion" flags in the selected TCB. They will again be set just before the issuance of the CLOSE macro instruction for the next data set.

The TCBDEB pointer is compared with the saved DEB address to determine if the current DEB was dequeued and freed by the Close routine. If the two DEB addresses are unequal, the current DEB has been freed. ABEND5 then branches to location ENTRY2 within its own module to repeat the processing for the next DEB on the queue. But if the two DEB addresses are equal, the Close routine did not process the current DEB. Accordingly, ABEND5 removes the DEB from the DEB queue, determines its size, and frees the space it occupies. It then branches to location ENTRY2 to repeat the processing for the next DEB.

Special Handling of a Recursion: It is possible for an error to occur during the execution of the Close routine, causing a recursion or reentry to ABEND. One possible cause of such an error is the overlaying of one or more DCBs because of the "steal core" function of ABEND2. Regard less of the cause, a recursion because of an error during the Close routine needs special handling to permit the continued closing of data sets after the point of error.

If a recursion to the ABEND routine occurs because of an error during the Close routine, ABEND2 invokes ABEND5 directly. The purpose is to continue the closing of data sets and the purging of DEBs. A test at the beginning of ABEND5 recognizes the set condition of the "Close" indicator in the selected TCB, and branches to perform special handling.

ABEND5 must first locate the DEB on which the Close routine was operating when the error occurred. It locates the DEB address by searching the RB queue to find the SVRB that was used during the previous execution of the ABEND routine. That SVRB holds the last used DEB address in its extended save area. This DEB address was placed in the extended save area by ABEND5 during its previous execution, just before it issued the last CLOSE macro instruction. The last used DEB address, when found, is saved in the extended save area of the SVRB used for the current execution of the ABEND routine.

ABEND5 next purges two unneeded SVRBs on the current task's RB queue that are consuming supervisor queue space. One of the SVRBs was created for the Close routine during the ABEND routine's last execution. The other SVRB was used for the previous execution of the ABEND routine, during which the "close" error occurred. ABEND5 removes these SVRBs by removing and freeing all SVRBs on the task's RB queue that represent transient SVC routines, except the SVRB controlling the current execution of the ABEND routine. In addition, (via the TAHABEND subroutine) ABEND5 removes from the transient area queues all pointers to these SVRBs. (The purge of the transient area queues will be described in greater detail under "Removal of Request Blocks for Transient SVC Routines.")

After the purging of the extra SVRBs on the RB queue, ABEND5 continues processing as if control had just been returned from the Close routine after normal DEB processing. The "close" and "recursion" flags are cleared, and the current DEB (since it was not freed by the Close routine) is dequeued from the DEB queue and freed. Normal DEB processing for the next DEB then continues, as previously described.

When all DEBs on the selected task's DEB queue have been processed, ABEND5 branches to its "purge SVRB" subroutine to dequeue and free SVRBs representing transient routines. (If an error has occurred during the execution of the Close routine, these SVRBs have already been released.)

REMOVAL OF REQUEST BLOCKS FOR TRANSIENT SVC ROUTINES: The purge of SVRBs representing transient routines is the beginning of a general purge of request blocks that is continued in ABEND6. The purge of SVRBs for transient SVC routines is performed immediately after the closing of data sets and the DEB purge for a given task, while that task is current and before the next higher level task is selected.

ABEND5 locates SVRBs for transient routines by a search of the RB queue belonging to the selected task. (Each next RB is pointed to by the RBLINK field of the previous RB, beginning with the ABEND routine's SVRB.) Each RB is examined to determine that it is an SVRB and that it represents a transient routine, as indicated by the bit settings in the RBSTAB field. Each SVRB for a transient routine is removed from the task's RB queue. Then ABEND5 branches to subroutine TAHABEND to remove the SVRB from the transient area queues.

The TAHABEND subroutine first tests the transient area block number (RBTABNO field) of the SVRB to determine if the represented routine is currently in a transient area block (TAB). If this field is zero, the SVC routine is not in a TAB. In this case, the subroutine searches the transient area request queue for a pointer to the SVRB. If the SVRB address is found, it is removed from the request queue and the purge of the transient area is complete.

If, however, the TAB number (RBTABNO) in the specified SVRB is not zero, the SVRB address is on a user queue and the associated routine is either in a TAB or was overlaid before it could be completed. In this case, the transient area user count is decreased by one to indicate one less outstanding request for the routine in the TAB. Then, by use of the TAB number as a displacement, the associated entry in the transient area control table (TACT) is found. By means of the TACT entry, the appropriate user queue is located and searched for the SVRB address. When the specified SVRB address is found, it is dequeued from the user queue, since the requestor that originally generated the SVRB is being terminated. The user queue for the TAB is then searched to determine if there are other users of the routine in the TAB. (The relative track and record address--TTR--in the TACT entry, representing the routine now in the TAB, is compared with the TTR in the remaining SVRBs on the user queue.) If the search indicates that there are other users of the routine, the purge of the transient area queues is complete. But if it indicates that there are no other users of the routine in the TAB, the associated TACT entry is flagged to indicate a free TAB.

Upon return of control from the TAHABEND subroutine, ABEND5 determines the size of the SVRB just processed (from its RBSIZE field) and frees the space it occupies, specifying subpool 255 (one of the subpools of supervisor queue space).

When all RBs on the selected task's RB queue have been examined, and if necessary dequeued and freed, the "task select" (TASKSEL) routine is invoked to select the next higher level task of the tree. Preparation is then made for redispatching ABEND5 (at location ENTRY2) under control of the selected TCB, as previously discussed.

When data sets, DEBs, and SVRBs have been purged for all tasks of the tree (as indicated by a test after SVRB processing), ABEND5 invokes ABEND6, via an XCTL macro instruction, to continue the purge of task resources.

Processing During ABEND6 (Entry Point IGC0301C)

ABEND6 performs the following main functions for each task of the tree:

- Purges request blocks (RBs) and updates the contents directory.
- Purges the load list.

- Purges dynamically acquired main storage.

- Releases the task control block.

- Provides final processing for the top task of the tree.

ABEND6 completes the RB purge begun in ABEND5, and purges the remaining resources of the specified task and its descendants. The released resources include: RBs not yet purged (PRBs and IRBs); control blocks of Contents Supervision and their associated load modules, if they are no longer needed; requests for enqueued resources (QELs); the load lists; and dynamically acquired main storage, if exclusively owned.

Like ABEND5, ABEND6 purges the various resources of each task of the tree, one task at a time. Similarly, ABEND6 selects each task by means of the previously described "task select" subroutine (TASKSEL), starting with the newest descendant of the specified task and ending with the specified or "top" task itself. But unlike the processing of ABEND5, in which each task was redispatched under the control of ABEND's SVRB to purge its own resources, ABEND6's processing is all done under the control of one task, the specified or top task of the terminating tree. This task remains currently dispatchable throughout the execution of ABEND6 for the current request.

After ABEND6 has purged all resources belonging to a selected task, it removes the task's TCB from the TCB queue and from the subtask queues, and frees the storage that the TCB occupies. The subtask queues indicate task relationships of the tree. The next higher level task is then selected and its resources are purged in the same way. The process continues until the top task of the tree has been selected and its resources purged. This time the TCB is not removed from its queues, nor is its space freed, since this TCB is still needed after ABEND6 exits.

As a last function, ABEND6 loads the return register with the completion code obtained from the top TCB. This completion code is then available to the top task's parent, the next higher level task, for its examination.

ABEND6 at its completion twice causes supervisor linkage to the Exit routine. The Exit routine during its first execution updates the transient area control table and the transient area queues, via its TAXEXIT subroutine. It does this because ABEND6, a transient SVC routine, is finished. During its first execution the Exit routine also removes the ABEND routine's SVRB from the top task's RB queue and frees its storage space. During its second execution the Exit routine, detecting an end-of-task condition, branches to the EOT routine.

The EOT routine performs final termination procedures for the top task of the tree. These procedures consist of:

- Passing control to an end-of-task exit routine (ETXR), if one has been specified.

- Posting an event control block (ECB) for the parent task, if an ECB has been specified.

- Removing the top TCB from its queues and freeing its storage space.

Upon regaining control from the EOT routine, the Exit routine removes the last RB from the top task's RB queue and frees its storage space. The Exit routine then branches to the Transient Area Refresh routine to refresh (if necessary) a transient area. The transient area may have been overlaid by the modules of the ABEND routine.

The Transient Area Refresh routine, when its processing is complete, branches to the Dispatcher. The Dispatcher then gives control to the current routine of the highest priority ready task.

<u>THE PURGING OF REQUEST BLOCKS AND THE UPDATING OF THE CONTENTS DIRECTORY</u>: The resources first purged by ABEND6 for a selected task are the request blocks (RBs). The RBs are processed by a subroutine called RBREMOVE. The processing varies according to the type of RB: SVRBs for resident routines, IRBs, and PRBs. The type of RB is determined by a test of the RBFTP bits of the RBSTAB field. For the format and contents of each type of RB, see Section 12, "Control Blocks and Tables." The processing for each type of RB will be separately discussed in the following paragraphs.

<u>Purging an SVRB</u>: Since SVRBs for transient routines have already been released by ABEND5, any SVRB detected by the RBREMOVE subroutine must represent a resident SVC routine. If the SVRB is not the last RB of the "top" TCB, the RBREMOVE subroutine removes the SVRB from the task's RB queue, determines its size from its RBSIZE field, sets up the subpool operand (255) for a FREEMAIN macro instruction, and frees the space occupied by the SVRB. If the SVRB is not the last RB on the selected task's RB queue, the next RB is obtained and RB processing for the task continues. But if

the released SVRB was the last RB on the queue, the RBREMOVE subroutine branches to the ENQ/DEQ Purge routine to purge QELs belonging to the selected task.

If the SVRB is the last SVRB on the top task's RB queue, the RBREMOVE subroutine does not release the SVRB. This last SVRB, used for the ABEND routine, is released by the Exit routine after ABEND6 is finished.

Purging an IRB: If the RBREMOVE subroutine finds an IRB that represents a user or system exit routine, it dequeues all IQEs or RQEs that are queued to it. The subroutine then decreases the "use count" in the IRB, according to the number of IQEs or RQEs that it removed. (The use count, stored in the IRB when the user exit routine was first requested, indicates multiple use of the same exit routine for different subtasks.)

The RBREMOVE subroutine removes the IRB from the task's RB queue, and resets the "active" flag (RBFACTV) in the IRB to indicate to the Stage 3 Exit Effector that the IRB is not on a task's RB queue.

Then two tests are made to determine if space belonging to the IRB may be freed. If the IRB's storage space was not dynamically acquired (as indicated by a test of the RBFDYN flag in the RBSTAB field), the RB is a permanent system interruption request block and may not be freed. Or if the IRB's use count is greater than zero, it is still needed, and should not be freed. In either case, the RBREMOVE subroutine processes the next RB on the selected task's RB queue, or if there is no other RB on the queue, branches to the ENQ/DEQ Purge routine. But if the IRB is not a system RB and contains a use count of zero, it is no longer needed and its space is freed. If it has a user register save area, originally reserved by the requestor of the user exit routine, the save area space is freed. (Existence of the save area is indicated by a nonzero RBPPSAV field in the IRB.) The 72-byte save area is freed from subpool 250 by a branch to the FREEMAIN routine (address FMBRANCH). The RBREMOVE subroutine then branches to the FREEMAIN routine to free the IRB's space from subpool 253. If there is another RB on the selected task's RB queue, the subroutine processes it. Otherwise, it branches to the ENQ/DEQ Purge routine to purge QELs belonging to the task.

Purging a PRB and Updating the Contents Directory: If the RB examined by the RBREMOVE subroutine is a PRB, there is an associated contents directory entry (CDE) which must be examined, and if necessary, purged from the contents directory. To examine the CDE, which represents a load module, the RBREMOVE subroutine branches to its subroutine PRBPROC to test the CDE and possibly update the associated queues.

If the CDE pointer (RBCDE) in the PRB is zero, there is no CDE associated with the PRB. There is therefore no need for PRBPROC to update the contents directory. It returns control to the RBREMOVE subroutine (at entry point TESTOCT) to test the PRB and, if possible, remove the PRB from the selected task's RB queue and free its storage area.

If the CDE pointer (RBCDE) in the PRB is not zero, there is a CDE, which means that a load module is associated with the PRB. The load module may be in the process of being loaded, or may already be residing in main storage. The status of the module may be determined by a test of the CDE's CDATTR flags. For the format and contents of a CDE, see Section 12, "Control Blocks and Tables."

If the module described by the CDE is in the process of being loaded (as indicated by the "set" condition of its NIC flag) PRBPROC frees the fetch work area, obtains the major CDE if the currently examined CDE is a minor (since alterations are always made in the major CDE), and processes according to two possible situations:

1. The module is being loaded under control of another PRB, not the selected PRB. A test of the CDERB field shows that it does not point to the selected PRB. The selected PRB is on a queue[1] of PRBs waiting for the module to be loaded for another task in the job step. In this case, the removal of the selected PRB from the waiting queue has no effect on other tasks whose PRBs are waiting. Accordingly, PRBPROC branches to its DQRBS subroutine to remove the selected PRB from the queue of waiting PRBs. PRBPROC then returns control to the RBREMOVE subroutine to remove the selected PRB from its task's RB queue and free its storage area.

2. The module is being loaded under the control of the selected PRB. (The test of the CDERB field shows that it points to the selected PRB.) In this case, the processing depends on whether there are other PRBs queued[1] to the selected PRB.

Purging the Module and Related Storage Areas: If a test of the RBPGMQ field indicates that no other FRBs in the job step are awaiting the loading of the

[1] Queuing field is RBPGMQ.

module, the CDE and its related areas can be removed without adversely affecting other tasks. Accordingly, the PRBPROC subroutine branches to the FREEMAIN routine to free the module's storage area and its associated extent list (if they exist). It then dequeues the major CDE for the module and any minor CDEs and, via the FREEMAIN routine, frees the space they occupy.

Preparation for Refetching the Module: But if PRBs for other tasks of the job step are queued, awaiting the loading of the module, the loading process cannot be stopped without ensuring that the module will be loaded under the control of one of the waiting PRBs. Otherwise, the tasks whose PRBs are waiting would be permanently nondispatchable, awaiting a resource that is never available. Accordingly, the PRBPROC subroutine prepares for the refetching of the module by the routines of Contents Supervision (as described in the next paragraph). It then frees the program area and extent list, if they exist, and dequeues and frees the major CDE and any minors.

Preparation for the refetching of the module consists of making the waiting PRBs ready to enter the CDSEARCH routine of Contents Supervision at entry point CDCONTRL. This routine initializes the request for the module. Other routines of Contents Supervision perform the fetch and update the contents directory. The PRBPROC subroutine prepares the waiting PRBs for entry to location CDCONTRL by performing the following steps for each queued PRB, via two subroutines, RBQUEUED and QDRBS:

1. Frees the fetch work area, if one has been allocated. (This action has already been done for the PRB to be removed.)

2. Stores the address CDCONTRL in the old PSW field of the PRB, in preparation for the dispatching of the CDSEARCH routine, mentioned above.

3. Reinitializes the RB by: placing zero in its wait count field (RBWCF), thus removing it from the wait condition; placing zero in the CDE pointer (RBCDE), since Contents Supervision will store a new CDE pointer in this field; and placing zero in the queuing field (RBPGMQ), since the RB is no longer in the waiting queue. Contents Supervision will create a new waiting queue of requesting SVRBs during the reinitialized fetch process.

4. Decreases by a count of one the use/responsibility count in the major CDE, in order to indicate that there is one less outstanding request for the module.

5. Locates the address of the TCB associated with the PRB by chaining through the RB queue, following the RBLINK fields, and branches to the supervisor's Task Switching routine with this TCB address. The Task Switching routine may alter the "new" TCB pointer (IEATCBP) to permit the Dispatcher to eventually pass control to location CDCONTRL for a task which is of higher priority than the current task.

The reinitialized request for the module will cause execution as soon as one of the tasks whose RBs have been readied is next dispatched.

After the PRBPROC subroutine has reinitialized the module request, it frees the program area and extent lists, if they have been acquired, and dequeues and frees the major CDE and any minors. When Contents Supervision is eventually dispatched, it will not find a CDE for the module, since the CDE has been removed from the contents directory. Contents Supervision therefore begins the process of fetching the module.

Processing if the Module is Already in Main Storage: If the module described by the CDE is already in main storage, the PRBPROC subroutine performs processing roughly parallel to that which it performs if the module is in the process of being loaded. There are, however, several differences:

- No fetch work areas are freed, since Contents Supervision has already freed these areas.

- Preparation for the refetching of the module occurs only if the module is serially reusable. The reasoning is that the module may now not be reusable, either because of a program check during its execution or because it could not finish and therefore could not reinitialize itself. In either case, waiting queued PRBs are made ready and pointed to location CDCONTRL, as previously described. But to force refetch by Contents Supervision, the PRBPROC subroutine clears the "serially reusable" flag and sets the "nonreusable" flag in the CDE.

- Instead of freeing the program area, extent list, and the CDE unconditionally if the selected PRB is in control of the module, the PRBPROC subroutine branches to entry point CDHKEEP, a subroutine of the Exit routine, to test the CDE use/responsibility count. If this count is zero, indicating that there are no outstanding requests for the module, the PRBPROC subroutine branches to the CDHKEEP subroutine and to two other subroutines of the Exit routine (CDDESTRY and ORDERCDQ) to free the program area, extent list, and the CDE. (For further information about CDHKEEP, CDDESTRY, and ORDERCDQ, see Section 9 "Exiting Procedures.")

If contents directory processing by the CDHKEEP subroutine is not needed because the selected PRB is not in control of the module, the selected PRB is removed from the RB "wait" queue[1] that originates in the CDE. The use/responsibility count is then decreased by a count of one to indicate that there is one less outstanding request for the module.

The result of the processing by PRBPROC is that the selected PRB has been removed from the CDE's RB queue of waiting requestors, or the request for the module has been reinitialized and, if necessary, the program area, extent list, and CDE have been freed.

Removing the PRB: The PRBPROC subroutine then returns control to the RBREMOVE subroutine (at location TESTOCT). The RBREMOVE subroutine removes the PRB from its task's RB queue and frees its storage area. The freeing of the PRB's storage area is similar to that for any other RB's storage area except that there is no user register save area to be freed, and the RB size and subpool number pertain to a PRB. The RBREMOVE subroutine branches to the FREEMAIN routine at entry point FMBRANCH to free the PRB's storage space from subpool 255. Then, if there is another RB on the selected task's RB queue, the RBREMOVE subroutine purges that RB in the manner previously described. But if there are no more RB's belonging to the selected task, the RBREMOVE subroutine branches to the ENQ/DEQ Purge routine.

Special Processing for the Last RB of the "Top" Task: If the RB just processed is the last RB of the "top" task of the terminating tree, the RBREMOVE subroutine performs special processing for this last RB. (The last RB is not the ABEND routine's SVRB but is the RB pointed to by its RBLINK field.) The last RB needs special processing to satisfy the needs of the supervisor's Exit routine, where the final purging of resources is performed after the completion of ABEND6. The Exit routine expects that the last RB belonging to a completed task is a PRB. The RBREMOVE subroutine therefore converts its last-processed RB into a PRB and ensures linkage to the Exit routine by altering certain RB fields. It converts the RB into a PRB by clearing the RBFTP subfield of the RBSTAB field. To avoid manipulation of the contents directory by the CDEXIT subroutine of the Exit routine, the RBREMOVE subroutine clears the CDE pointer (RBCDE). To permit dispatching of the Exit routine for the top task when ABEND6 processing is complete, the RBREMOVE subroutine removes any existing wait condition by clearing the RBWCF field. Also for this purpose it points the RB old PSW (second word of the RBOPSW field) to an SVC 3 instruction in the communication vector table (CVT) at location CVTEXIT. The SVC 3 instruction, when placed in execution by the Dispatcher, will cause supervisor linkage to the Exit routine. After altering the last RB of the top task to cause eventual linkage to the Exit routine, the RBREMOVE subroutine branches to the ENQ/DEQ Purge routine, as it does after all RB's have been processed for any selected task of the tree.

THE PURGING OF REQUESTS FOR ENQUEUED RESOURCES: ABEND6 enters the resident ENQ/DEQ Purge routine (at entry point IEAQEQ01) to remove resource requests generated by the issuance of the ENQ macro instruction for the selected task. This is the task selected by the TASKSEL subroutine from those belonging to the terminating tree. The queue elements (QELs), representing resource requests, must be removed. This is done so that routines belonging to other tasks could gain access to the enqueued resource, if the abnormal termination occurred before the DEQ routine could be executed for the selected task. Otherwise, the resource would remain inaccessible.

The ENQ/DEQ Purge routine searches the system QCB chain for QELs that were constructed by the ENQ routine for the selected task. Each QEL contains a pointer to the TCB under whose control it was constructed. Each QEL belonging to the selected task is removed from its QEL queue, and its space is freed, via a branch to the FREEMAIN routine. If all QELs queued to a minor QCB are removed, the minor QCB is also dequeued from its major QCB and its space is freed. If the major QCB has no minor QCB, it is also removed from its queue and its space is freed. When all the task's enqueued requests have been removed from the system QCB chain and its related queues, the ENQ/DEQ Purge routine branches to the Load List Purge routine to continue the purging of the selected task's resources.

The ABEND6 routine must also terminate device reservations acquired through the RESERVE macro instruction and not released through a subsequent DEQ macro instruction. These device reservations occur only in systems with the shared DASD option.

----
[1] Queuing field is RBPGMQ.

Outstanding reservations are reflected in the TCB enqueue count (offset 112 in the TCB). When such a reservation is detected, the ABEND6 routine branches to the EXCP interface subroutine in the ENQ/DEQ module. This subroutine prepares control blocks for an EXCP command and issues theEXCP command that results in the release of the reserved device. (See "Processing in Systems With Shared DASD".)

<u>PURGING THE LOAD LIST</u>: The resident Load List Purge routine (entered at location IEAQABL) releases load list elements and modules that were loaded for the selected task and are now no longer needed. This is the same routine that performs a similar function for the EOT routine during a normal task termination. The Load List Purge routine releases modules that were loaded for the selected task, but which were not released before the task was abnormally terminated. The modules would normally have been released by either the Delete SVC routine or the CDEXIT subroutine of the Exit routine.

The Load List Purge routine examines each load list element in the load list, representing all modules that were loaded for the selected task. (The list origin for the load list is in the TCBLLS field of the TCB.) The routine subtracts the responsibility count (number of load requests for each module) stored in its load list element, from the use/responsibility count (total number of requests for the module) stored in the CDE for the module. Each load list element points to its associated CDE. The purpose of subtracting the responsibility count from the use/responsibility count is to determine the number of outstanding requests for the loaded module.

The Load List Purge routine branches to the CDEXIT subroutine (location CDHKEEP). The subroutine tests the number of outstanding requests for the module. If there is no outstanding request for the module, the routine tests the module's attributes. If the module is in the link pack area, control is returned immediately to the caller. If the module is not in the link pack area and is either reenterable or reusable, the routine sets the "release" flag in the module's CDE and the "purge" flag for the job pack queue. (These flags are tested by the GETMAIN routine to determine which module's space may be freed, if needed space is otherwise unavailable.) If the module is neither serially reusable nor reenterable, CDEXIT (via its CDDESTRY subroutine) removes the module's CDE from the job pack queue, and frees the space occupied by the module, its extent list, and its CDEs (major and minor).

On return of control from the CDHKEEP subroutine, the Load List Purge routine frees the load list element. The process is repeated until all load list elements have been examined.

<u>THE PURGING OF DYNAMICALLY ACQUIRED MAIN STORAGE</u>: After control is returned from the Load List Purge routine, ABEND6 branches to the Subpool End-of-Task routine (SPEOT), whose entry point is IEAQSPET. SPEOT is part of the EOT routine. The SPEOT routine releases subpools exclusively "owned" by the selected task and frees the associated subpool queue elements (SPQEs).

The SPEOT routine frees unshared subpools of main storage allocated to the selected task. The subpools are represented by SPQEs, which have as their list origin the TCBMSS field of the selected TCB. The routine examines each SPQE on the queue. If an SPQE represents a shared subpool that may not yet be freed, the queue is updated (the "shared" SQPE is freed) to reflect the new unshared ownership of the subpool. If, however, an SPQE represents a subpool not shared with another task of the job step, the subpool and its SPQE are freed, via a branch to the FREEMAIN routine. The SPQE list is updated, and the next element is examined. When all elements have been examined, subpool 253, one of the numbers assigned to supervisor queue space, is explicitly freed since there is no SPQE for this subpool.

If the selected task is the job step task, a job step termination must be occurring. In this case, besides freeing subpools and SPQEs, the SPEOT routine dequeues and frees CDEs and extent lists that describe modules in the job pack area (subpool 251) and in the step link pack area (subpool 252). These CDEs and extent lists are released during termination of the job step task because the SPQEs for these two subpools are queued to the job step TCB and have not previously been released. The SPEOT routine checks the job pack area control queue (JOBPACQ), whose list origin is the TCBJQP field in the TCB, to discover if there is at least one CDE. If there is at least one CDE, the SPEOT routine branches to a part of the CDEXIT subroutine of the Exit routine (CDDESTRY) to free the remaining CDEs and their associated extent lists.

<u>RELEASING THE TASK CONTROL BLOCK (TCB)</u>: After freeing main storage and SPQEs for the selected task, ABEND6 follows either of two paths of processing, depending on whether the selected task is the top task of the tree. If the selected task is the top task, final processing is performed, as described in "Final Processing for the Top Task." But if the selected task is not the

top task, ABEND6 sets the "completion" flag (TCBFC) in the selected TCB, to indicate that the task has been terminated, removes the TCB from its queues, and frees its space.

The dequeuing and freeing of the selected TCB is performed by two resident routines belonging to EOT: the "dequeue TCB" routine (DQTCB), whose address is IEADQTCB, and the "erase" routine (address IEAQERA). The "dequeue TCB" routine removes the address of the selected TCB from the TCB queue. The reader may recall that the TCB queue consists of pointers connecting the TCBs of the system in the order of their dispatching priorities. The Dispatcher may examine this queue to determine the next task whose current routine should be dispatched. Since the selected task is now terminated, its TCB must be removed from consideration by the Dispatcher.

The ABEND routine does not contain special code in systems with the time-slicing feature. The EOT routine contains special code for time-slicing, and this code performs the preceding functions for ABEND6.

The "erase" routine removes the selected TCB from its subtask queues, updating the TCBLTC and TCBNTC pointers in the next higher TCB of the tree. The "erase" routine then branches to the FREEMAIN routine to free the space occupied by the selected TCB.

FINAL PROCESSING FOR THE TOP TASK: After the release of a selected TCB, ABEND6 branches to its "task select" subroutine to select the next higher level task of the tree of terminating tasks. The resources of the newly selected task are released in a manner similar to that described. When the resources of the "top" task of the tree have been released, as indicated by a test after the SPEOT routine has returned control, ABEND6 begins final processing for the top task. It loads the return register with a completion code that it obtains from the TCBCMP field of the top TCB. The parent of the top task may examine the completion code, via an end-of-task exit routine or a posted ECB, when its current routine is dispatched. After placing the completion code in the return register, ABEND6 causes linkage to the supervisor Exit routine by issuance of an SVC 3 instruction.

The supervisor Exit routine and the EOT routine provide final cleanup of the top task. The Exit routine removes the ABEND routine's SVRB from the top task's RB queue and frees its space. The Exit routine then branches to the Dispatcher to return control to the current routine of the highest priority ready task. When the top task of the terminating tree is next dispatched, its RB old PSW causes control to be passed to an SVC 3 instruction in the communication vector table (address CVTEXIT). Control is passed to the Exit routine, via Supervisor linkage, this time to remove the last RB from the top task's RB queue. This RB is the one that was converted to a PRB by the RBREMOVE subroutine (see "Special Processing for the Last RB"). The Exit routine, after removing the "dummied" PRB, detects an end-of-task condition and branches to the EOT routine.

The EOT routine, via its DQTCB and "erase" routines, removes the top TCB from the TCB queue and its subtask queues and frees its space. (If, however, an end-of-task exit routine (ETXR) or an ECB was specified when the top task was attached, the top TCB is not removed from its subtask queues.) The EOT routine next clears the "new" TCB pointer (IEATCBP) to zero, indicating to the Dispatcher that it must search the TCB queue to find the highest priority ready task from among those that remain in the system. A task switch is thus ensured. The EOT routine returns control to the Exit routine to free the space occupied by the last RB (the "dummied" PRB) of the top task. The Exit routine then branches to its Transient Area Refresh routine to refresh (if necessary) a transient area block that was overlaid by the various modules of the ABEND routine. The Transient Area Refresh routine, after performing its processing, branches to the Dispatcher to return control to the current routine of the highest priority task of those that remain in the system.

Processing by ABEND6, the usually last-executed segment of ABEND, is now complete.

Permitting the System to Quiesce (System Quiesce Routine)

System quiescence involves the abnormal termination of the failing task and its associated tasks, the isolation of the failing task's region, the scheduling of no further jobs, and the attempt of the previously scheduled tasks in the non-failing regions of the system to reach normal processing termination.

The System Quiesce routine (IEAQTWST, entry point IECIWTST) serves as an emergency exit for the ABEND routine. When a severe error condition is detected during ABEND processing, the System Quiesce routine places the failing task in a wait condition, sets the TCBs of all related tasks nondispatchable, frees the resources allocated to the failing tasks, issues a message to the operator indicating that a CPU wait state has been averted and instructing him to permit the system to quiesce. Control is returned to the supervisor enabling the system for interruptions, and processing in the other regions of the system is continued.

The address of the System Quiesce routine is located in the CVTXWTO field of the CVT. The routine is branched to by ABEND2 if any of the following three conditions is detected:

- The task scheduled for ABEND processing is in "must complete" status, indicated by either the TCBFJMC or the TCBFSMC flag.
- The task scheduled for ABEND processing is a system task.
- Entry to the ABEND routine was caused by an invalid recursion -- a failure that occurred during a previous execution of the ABEND routine. (A second entry to the ABEND routine is considered a valid recursion only if the failure in ABEND processing occurred during the Open routine, the ABDUMP routine, or the Close routine. Second entries from any other part of ABEND processing are considered invalid recursions.)

The System Quiesce routine causes the printing of the appropriate error message, which warns the operator that a severe error has occurred, a CPU wait state has been averted, and the system must be allowed to quiesce.

The System Quiesce routine first saves the program check new PSW. The wait state code, passed to the System Quiesce routine in register 0, is translated and saved for later incorporation in the error message.

To determine if the failing task is a system task, the TIOT field in the current TCB is tested. If the TIOT field is zero, the failing task is a system task, and an appropriate message is moved into the buffer area. If not, the step name in the TIOT is replaced by the related wait state code and all related subtasks are set nondispatchable.

The System Quiesce routine next branches to the ENQ/DEQ Purge routine which frees the resources that were allocated to the TCB of the failing task. Upon return from the ENQ/DEQ Purge routine, a branch is executed to the POST routine to post the Write-to-Log ECB in the Unit Control Module with a unique wait state code associated with the communications task, which is later recognized by the Router, module IEECVCTR.

Upon re-entry to the system quiesce routine from POST, the address of the program check new PSW is now restored. The region of the failing task must be isolated to preserve the contents of main storage for analysis. To accomplish this, the system quiesce routine clears a dummy ECB, and issues a WAIT macro instruction specifying this ECB.

The System Quiesce routine sets up two different error messages, depending on whether the failing task is a problem program or a system task. Both error messages contain the wait state code that describes the error condition. If an invalid recursion is detected, the message contains the completion code for the original entry into the ABEND routine. When the communications task is entered, module IEECVCTB invokes the Router (SVC 72), which schedules the error message by issuing a WTO (SVC 35) specifying the address of the system quiesce routine buffer area.

After the WAIT macro instruction has been issued on the dummy ECB, the system quiesce routine returns control to the supervisor, enabling the system for interruptions and permitting the other regions of the system to continue processing. A stand-alone dump program must be scheduled when the system has quiesced.

2250 SYSTEM OPERATOR'S CONSOLE

The IBM 2250 Model 1 Display Unit can be used as a system operator's console with the IBM System/360 Models 50, 65, or 75; it is standard with the Model 91. If this option is selected, 2250 System Operators Console programming support is provided for displaying system status and reference information, and system and problem program messages to the operator.

Console support is part of the Communication Task in Operating System/360 with MVT (see Section 7: Console Communications and System Log). The following modifications are made to the Communications Task to accommodate the 2250.

- Pointers to the 2250 Processor routine and Display Control Module (DCM) are added to the Unit Control Module (UCM).
- The Console Device Support routines, comprised of the 2250 Processor, Display, I/O, Options, Asynchronous Error, and Open/Close routines, are added to SVC 72.

COMMUNICATION TASK CONTROL FLOW

Control flow for the communication task with 2250 support is identical with normal MVT flow up to the point when SVC 72 is issued (see Figure 11-1). Where necessary, these functions are described below to provide continuity for the 2250 information.

The Router routine, part of module IEECVCTR (SVC 72), examines the UCM to determine the ECB which has been posted and the Console Device Support routine which should get control. This is accomplished by locating the first Device Control Entry (DCE) which indicates it can process a request to display a message or a request to read a command. After completion of processing by the Console Device Support routines, control is returned to the Router, which re-examines the UCM and returns control to the Communication Task Wait routine only if no other ECBs have been posted. When a posted ECB is found by the Router routine, the Router again passes control to the appropriate Console Device Support routine.

The Console Device Support routines perform reading and writing operations on the 2250 console by using the EXCP macro instruction. These support routines consist of the 2250 Processor, Display, Input/Output, Options, Open/Close, and Asynchronous Error routines.

The 2250 Processor routine receives control from the Router routine. After determining the actions required, control passes to the Open/Close, Display, Input/Output, or Asynchronous Error routines.

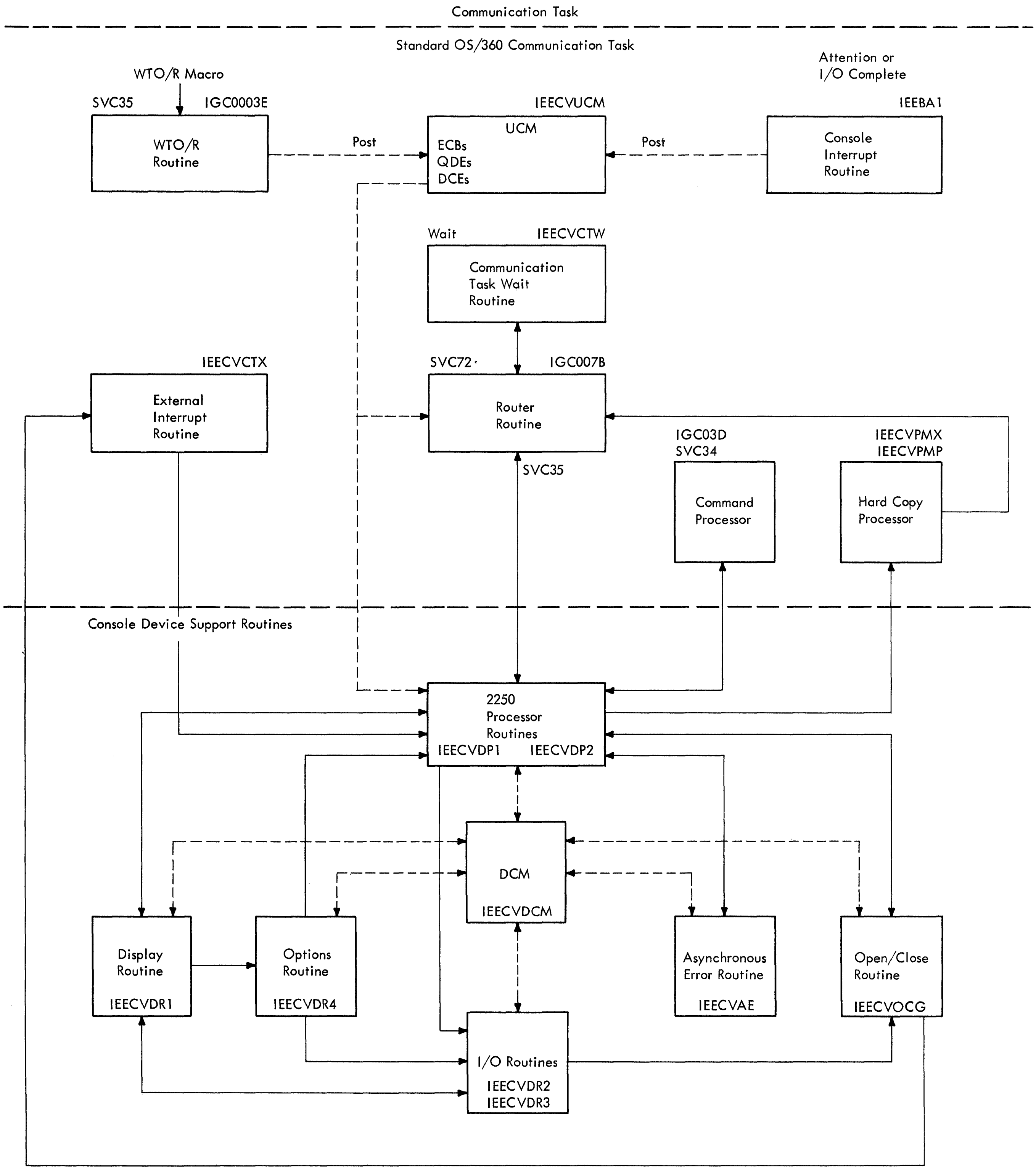
To accomplish their functions, the Display, Input/Output, and Options routines access the DCM as necessary to (1) examine information stored by the 2250 Processor routine, and to (2) store information to be examined by the 2250 Processor routine. Upon completion of an input/output operation involving the alternate console, the 2250 Processor routine receives control so that it can manage the message buffers and access the DCM to record and obtain information pertinent to the 2250 operator's console.

The Input/Output routine reads a command into a buffer area. The 2250 Processor routine calls the Command Processor routine via an SVC. The Command Processor routine analyzes the command; if the command is acceptable, it is moved from the buffer into which it was read to a local buffer, and is processed by the appropriate command execution routines. If the command is invalid, a system message informing the operator of the invalid command appears on the screen. When a valid command has been processed by the Command Processor routine, control returns to the 2250 processor routine.

The Input/Output routines write messages in the 2250 buffer, causing them to be displayed. After a message has been displayed, control returns to the 2250 Processor routine.

UNIT CONTROL MODULE MODIFICATION

The Unit Control Module (UCM) is the primary control table for console communication. It is a non-executable module containing ECBs used in the WAIT/POST mechanism in the Write-To-Operator and Console Interrupt routines. It contains pointers to WQEs and RQEs, and pointers to routines that support the 2250, a typewriter, or a reader-printer as consoles. The name of the 2250 Processor routine is contained in location UCMNAME of the UCM.



UCM - Unit Control Module
ECB - Event Control Block
QDE - Queue Descriptor Entry
DCE - Device Control Entry
DCM - Display Control Module

Figure 11-1. Control Flow of 2250 System Operator's Console Support

The modification in the UCM is the insertion of a pointer to the Display Control Module (DCM) at location UCMXB. This pointer is inserted at system generation when the user specifies a 2250 as either a primary or an alternate console.

CONSOLE DEVICE SUPPORT ROUTINES

The Console Device Support routines perform read and write operations to display system and problem program messages to the operator, cause a hard-copy of these messages to be produced, receive commands issued by the operator, process light pen attentions, and display unit status and command formats. The routines are:

- Display Control Module (IEECVDCM): maintains an image and status of the display according to the options selected (non-executable).
- 2250 Processor (IGC3107B): determines the event that occurred and passes control to the appropriate Console Device Support routine, and to other routines associated with the Communication Task. It accesses the Display Control Module to record and obtain information pertinent to the 2250 operator's console.
- Display (IGC3407B): accepts input from the 2250 Processor routine and alters the main storage representation of the display. It also relates a light pen attention to a selected function and, if necessary, computes a list of buffer addresses for use by the Input/Output routines in building channel programs.
- Input/Output Routines (IGC3507B, IGC3607B): process alphameric keyboard attentions including reading commands and replies, perform delete functions of WTOs and WTORs, and construct and execute all necessary channel programs.
- Options (IGC3807B, IEECVDRS): displays unit status and command formats if sufficient storage is available. Otherwise, a WTO message is displayed.
- Open/Close (IGC3I97B): performs open and close functions for the DCB associated with the 2250 console device.
- Asynchronous Error (IGC3707B): restores the 2250 buffer if an asynchronous error occurs during regeneration of the display, and initializes the display after console switching has occurred. Restoration of the buffer is accomplished by rewriting the buffer using the image maintained in the DCM.

MODEL 91 DECIMAL SIMULATOR (IEAXDS00) ROUTINE

The Decimal Simulator (IEAXDS00) routine is provided to perform decimal arithmetic instructions since the decimal feature on the Model 91 includes only the "EDIT" and "EDMK" instructions.

Note: In this publication, the Decimal Simulator routine is also referred to as the simulator.

The execution of the following instructions is simulated by the Decimal Simulator routine:

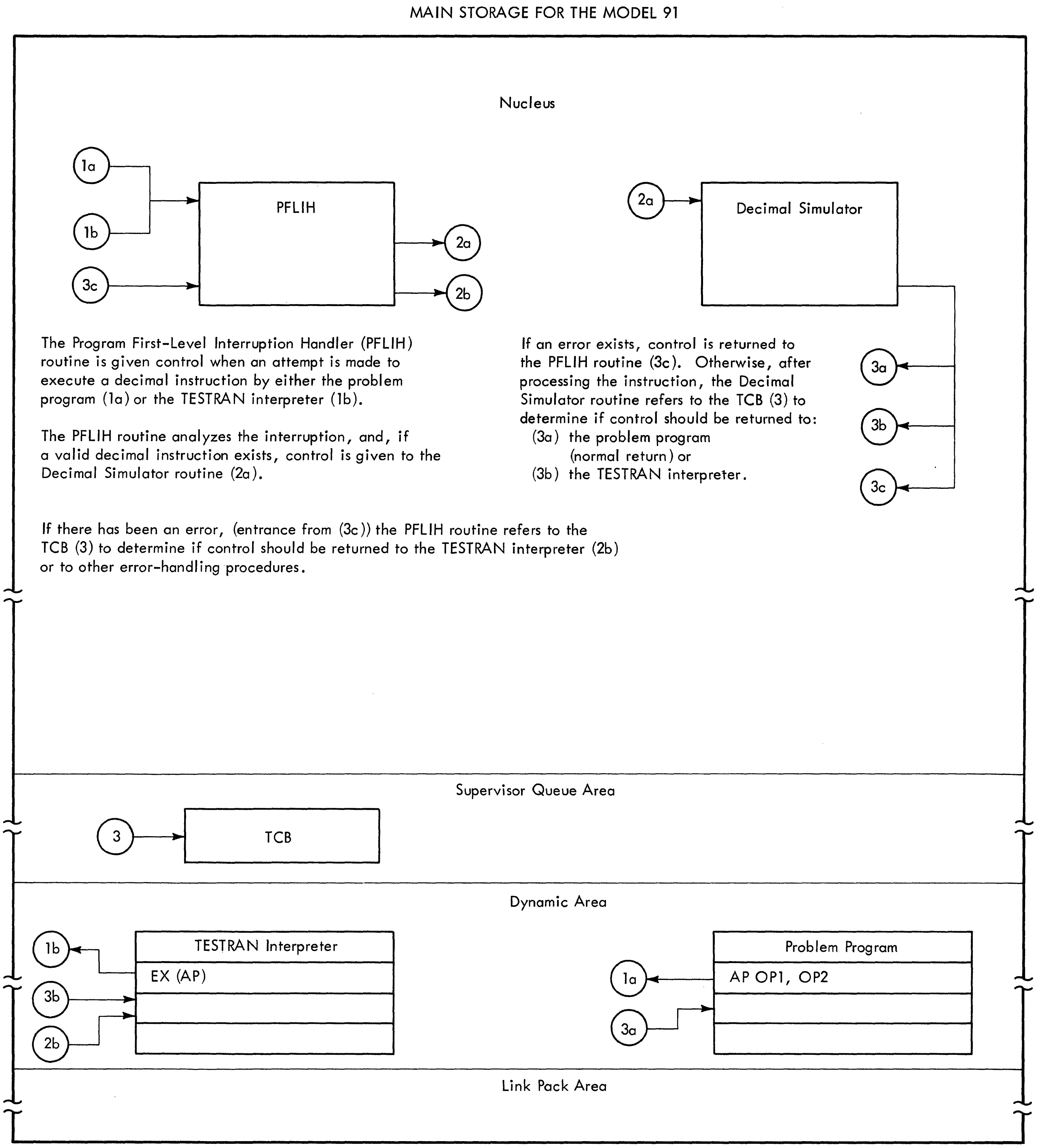
Instruction	Assembler Mnemonic	Operation Code
Add Decimal	AP	X'FA'
Subtract Decimal	SP	X'FB'
Zero-and-Add Decimal	ZAP	X'F8'
Multiply Decimal	MP	X'FC'
Divide Decimal	DP	X'FD'
Compare Decimal	CP	X'F9'

RELATIONSHIP TO THE OPERATING SYSTEM

Figure 11-2 indicates the relationship of the Decimal Simulator routines to the operating system. On the Model 91, the attempted execution of a decimal instruction causes a precise interruption. This interruption causes control of the CPU to be given to the Program First-Level Interruption Handler (PFLIH) routine.

After determining that the cause of an interruption was due either to a decimal instruction or to an EXECUTE instruction that addresses a decimal instruction, the PFLIH routine transfers control to the Decimal Simulator (IEAXDS00) routine. The Decimal Simulator routine, operating in the supervisor state with all interruptions masked, interprets the instruction, checks it for validity, and performs operations that simulate the execution of the instruction. At the completion of the simulation, control is given back to the CPU until another decimal instruction is encountered.

A decimal instruction that causes an interruption may have been fetched directly from a problem program, or fetched remotely while the TESTRAN interpreter routine was attempting to execute the instruction indirectly. If the simulation of the instruction execution is completed without error, the Decimal Simulator routine refers to the task control block (TCB) of the current task to determine where control is to be given. The possibilities are to return control to either the TESTRAN interpreter or the problem program containing the decimal instruction.



The Program First-Level Interruption Handler (PFLIH) routine is given control when an attempt is made to execute a decimal instruction by either the problem program (1a) or the TESTRAN interpreter (1b).

The PFLIH routine analyzes the interruption, and, if a valid decimal instruction exists, control is given to the Decimal Simulator routine (2a).

If an error exists, control is returned to the PFLIH routine (3c). Otherwise, after processing the instruction, the Decimal Simulator routine refers to the TCB (3) to determine if control should be returned to:

(3a) the problem program (normal return) or

(3b) the TESTRAN interpreter.

If there has been an error, (entrance from (3c)) the PFLIH routine refers to the TCB (3) to determine if control should be returned to the TESTRAN interpreter (2b) or to other error-handling procedures.

• Figure 11-2. Relationship of the Decimal Simulator Routine (IEAXDS00) to the Operating System

• Table 11-1. Organization of the Decimal Simulator (IEAXDS00) Routine

Routine	Function
Simulator Control (DECENT)	Main routine of the simulator: • Monitors overall operation. • Directs control to simulating routine once per execution of the simulator. • Checks for errors in data validity, protection, and overlap.
Add/Subtract Decimal and Zero-and-Add Decimal (DECASP)	Simulates execution of the following instructions: • Add Decimal • Subtrack Decimal • Zero-and-Add Decimal
Multiply Decimal (DECMP)	Simulates execution of the Multiply Decimal instruction.
Divide Decimal (DECDP)	Simulates execution of the DividIQDecimal instruction.
Compare Decimal (DECCP)	Simulates execution of the Compare Decimal instruction.
Analyzer and End	Determines where control is to be returned after simulating a decimal instruction.

If an error has occurred during the simulated execution of an instruction, the Decimal Simulator routine gives control back to the PFLIH routine routine as indicated in Figure 11-2.

SIMULATOR ORGANIZATION

Table 11-1 and Figure 11-3 indicate the organization and flow of the Decimal Simulator routine. A discussion of the major routines in the simulator is given in the sections which follow.

Simulator Control (DECENT) Routine

The Simulator control routine (hereinafter referred to as the control routine) performs the initialization of the Decimal Simulator routine. The control routine is entered from the Program First-level Interruption Handler (PFLIH) routine when it is determined that a decimal instruction is to be simulated. The functions of this control routine are to:

- Simulate address checking and protection checking.

- Simulate data checking, except for a decimal divide exception and a specification exception.

- Direct the processing to the appropriate arithmetic routine.

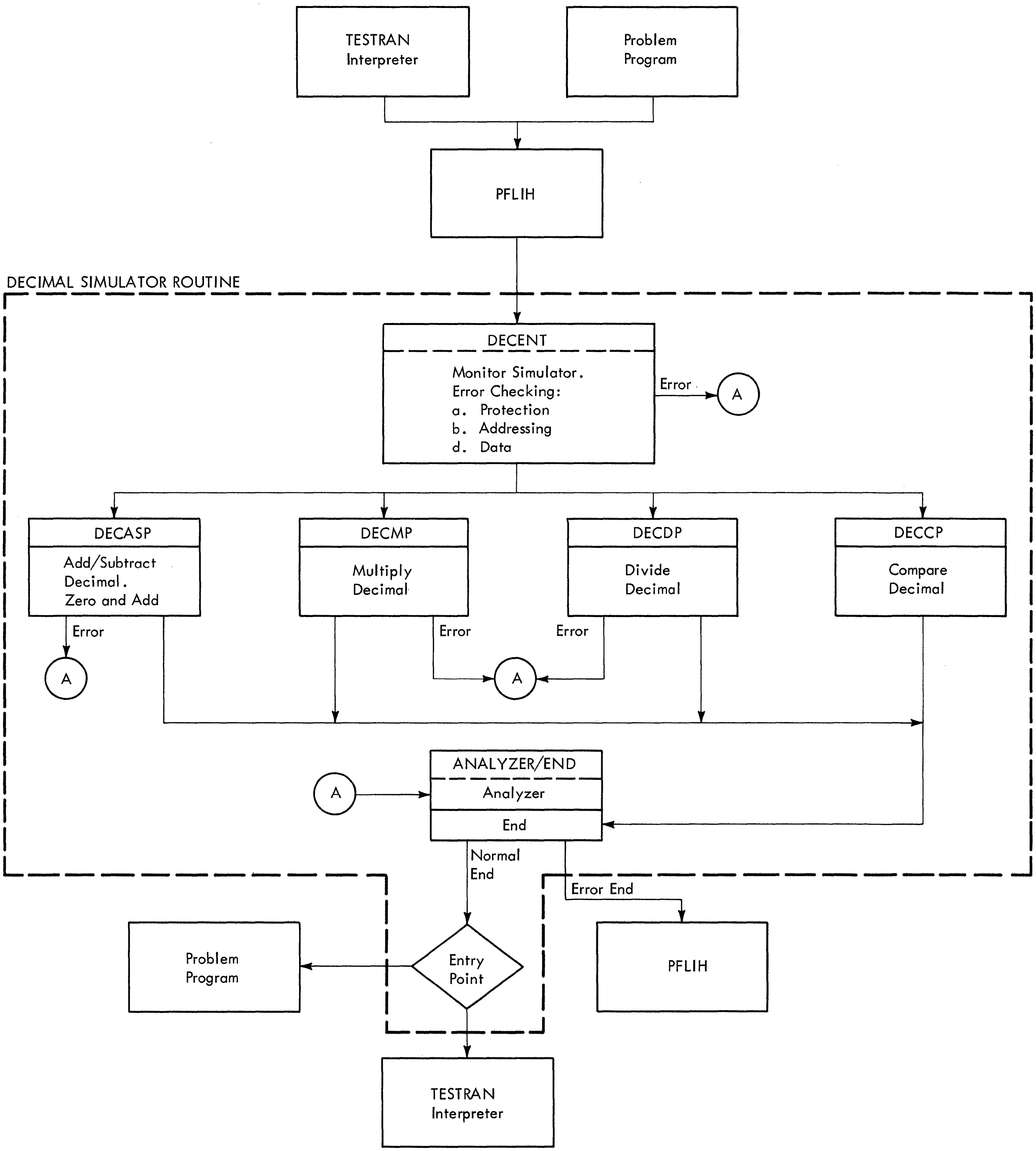
In performing the preceding functions, the control routine first obtains the size of main storage from the communications vector table. The main storage addresses of both the first operand and the second operand of the decimal instruction are determined, the length (in bytes) of each operand is computed, and the Decimal Simulator routine's work area is cleared.

With the addresses established, the control routine carries out the following simulated hardware checks in the order given.

1. The operand addresses are examined to determine if either (or both) is within the available storage limitations for the installation. If an address is outside the storage limit, an addressing exception occurs.

2. Using the data addresses and the length of the data fields, a check is made to determine if invalid overlapping has occurred. If it has, a data-check exception results. This check is not made for a Zero-and-Add Decimal (ZAP) instruction.

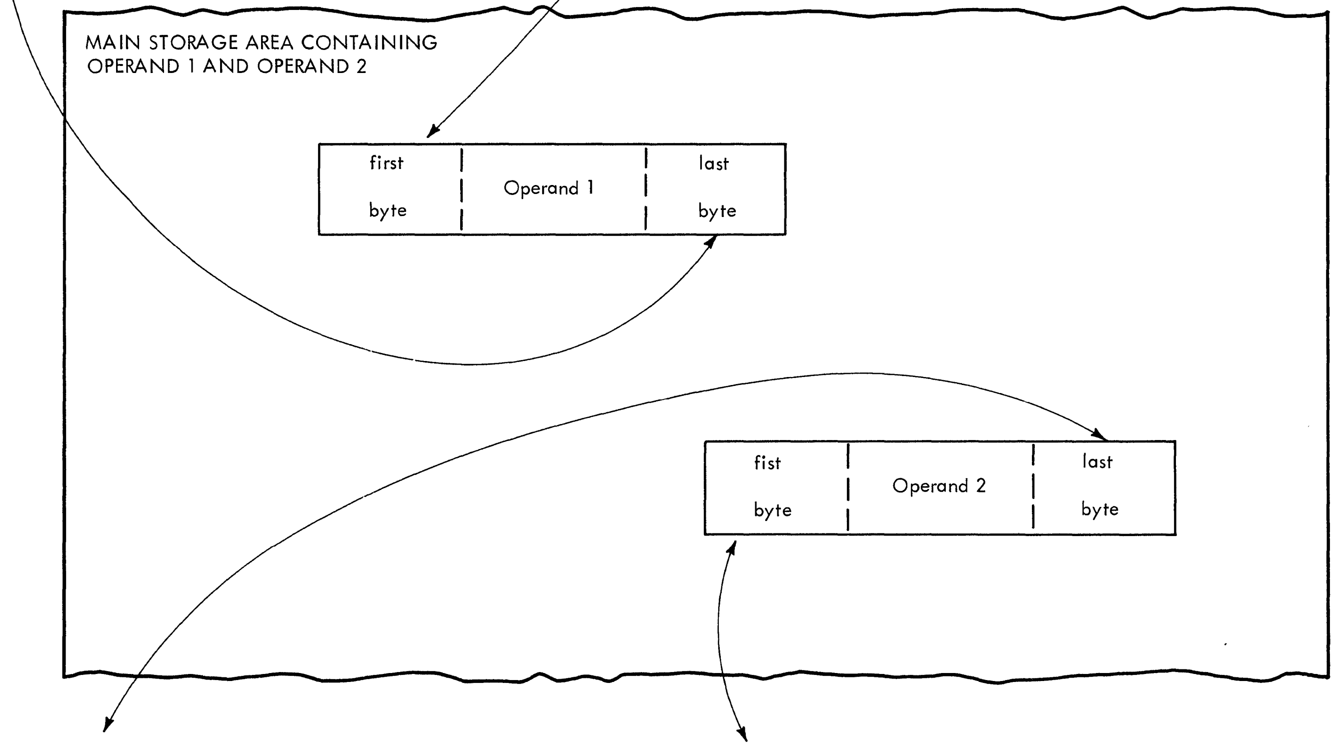
3. If the program old PSW protection key is zero, a protection check is not made. Otherwise, the addresses of the left-most and the right-most bytes of the data are checked for fetch and/or store protection violations. This is to ensure that boundary violations of



• Figure 11-3. Decimal Simulator (IEAXDS00) Routine Organization and Flow of Control

(1) The address of the last byte specified in operand 1 is checked for both fetch and store protection.* (For a Compare Decimal instruction, only a fetch protection check is made.) If the check is satisfactory, step (2) is performed. Otherwise, a protection exception results.

(2) The address of the first byte specified in operand 1 is checked against the address of the last byte specified in operand 1 to see if both bytes are in the same 2K-byte storage block. If they are, a protection check is not made since the storage block has already been checked in step (1). Step (3) is then performed. If a different 2K-byte storage block is indicated, a complete fetch and store protection check is made.* (For a Compare Decimal instruction, only a fetch protection check is made.) If the check is satisfactory, step (3) is performed. Otherwise, a protection exception results.



(3) The address of the last byte specified in operand 2 is checked against the address of the first byte specified in operand 1 to see if both bytes are in the same 2K-byte storage block. If they are, a protection check is not made since the storage block has already been checked in either step (1) or step (2). Step (4) is then performed. If a different 2K-byte storage block is indicated, a fetch protection check is made.* If the check is satisfactory, step (4) is performed. Otherwise, a protection exception results.

(4) The address of the first byte specified in operand 2 is checked against the address of the last byte specified in operand 2 to see if both bytes are in the same 2K-byte storage block. If they are, a protection check is not made since the storage block has already been checked in either step (1), step (2), or step (3). If a different 2K-byte storage block is indicated, a fetch protection check is made.* If the check is satisfactory, the protection checking procedure is through, and the sign code checking is initiated. Otherwise, a protection exception results.

* In the performance of the protection check, the protection key in the old PSW is compared with the protection key for the 2K-byte block of main storage in which the address (of the data) is located. The 2K-byte storage block address is determined by setting to zero the eleven low-order bits of the address of the byte that is being checked. The remaining bits of the address indicate the storage block address.

- Figure 11-4. Storage Protection Checking restricted storage do not occur. (See below for details of protection check). Since the results of most decimal operations are always placed in the storage location given by the first operand address of an instruction, the check applied to a second operand address is only for a fetch of protection violation. Except for a Compare Decimal instruction, the first operand address of a decimal instruction is checked for both a fetch and a store violation before the data is moved to the simulator's work area. With a Compare Decimal instruction, the first operand address is checked only for a fetch type of protection violation. If a violation occurs as a result of any of the preceding checks, a protection exception results.

4. The operands addressed in the instruction are then moved into the work area. (In the case of a Zero-and-Add Decimal instruction, only the second operand is moved into the work area, and the first operand is set to a plus zero.)

<u>Note</u>: For all arithmetic operations, all data handling and movement is done in and/or between the operand work areas.

5. Each operand is checked for a valid sign code (i.e., decimal values 10-15). An invalid sign code results in a data-check exception.

   <u>Note</u>: If the instruction is a ZAP instruction, the <u>sign code check</u> is made for the first operand by comparing against the value that has been pre-set to prevent an error condition.

6. The digit codes of both the first and second operands are checked against the permissible codes for numeric-type information. Invalid digit codes result in a data-check exception. Only the second operand of a ZAP instruction is checked for validity.

After performing the preceding checks for decimal arithmetic exceptions, the control routine forces the sign code (in the work area) of both the first and second operands to EBCDIC format. These sign codes, together with the EBCDIC sign code for the result of the decimal operation are recorded in the work area of the Decimal Simulator routine.

If, during the simulated hardware checks, an error condition (i.e., a condition that would have caused an operation exception to occur) is detected, further checking by the control routine is terminated and control is immediately given to the Analyzer/End routine. Operands one and two of the decimal instruction are not changed. The old PSW indicates the appropriate error condition as if the condition had been detected by the hardware itself.

<u>PROTECTION CHECK FEATURE</u>: To ensure against boundary violations, the complete storage areas occupied by both operand 1 and operand 2 are checked in the order (1) through (4) as indicated in Figure 11-4.

<u>Simulator Routine for Add, Subtract, Zero-and-Add Decimal Instruction (DECASP)</u>

The DECASP routine is invoked by the Simulator Control (DFCENT) routine if either the Add Decimal, the Subtract Decimal, or the Zero-and-Add Decimal instruction has been encountered and if the DECENT routine has detected no error. Whenever it is given control, this routine simulates the processing of one of the three mentioned instructions.

If the instruction is a Subtract Decimal instruction, the sign of the second operand is reversed so as to permit the processing to be carried out in the same manner as for an Add Decimal instruction. Then, for all three instructions, depending on whether neither, one, or both of the operands of the instruction are zero, the processing is performed with the following considerations:

- If the first operand is not zero, then each operand is converted to binary format. The two operands are added together if their signs are alike (after the reversal of the sign of the second operand as previously indicated). If the signs are unlike, the smaller operand is subtracted from the larger operand.

  In the preceding processes, if the actual length of either operand is greater than five bytes (i.e., the length code L_2 or L_1 is greater than four), the addition or subtraction is done in groups of four bytes at a time.

  At the completion of the arithmetic operations, the condition code is set in the PSW. If overflow has occured in addition (as, for example, a result of an indicated operation to add two numbers of like signs), an exit is made to the analyzer section of the Analyzer/End routine. Otherwise, the exit is to the end section of the Analyzer/End routine.

- If the absolute values of the two operands are equal and the signs of the operands are opposite, a plus zero result is supplied.

- If the first operand is zero, the second operand is moved to the first operand work area, and the condition code in the old PSW is set. If overflow has occurred, an exit is made to the analyzer section of the Analyzer/End routine. Otherwise an exit is made to the end section of the Analyzer/End routine. If the instruction is a Zero-and-Add Decimal (ZAP) instruction, the first operand has been previously forced to zero (see Simulator Control section). Therefore, the ZAP instruction is treated at this point as an Add Decimal instruction.

- If both operands are zero, the first operand is given a positive sign, and the condition code in the PSW is set to zero.

Simulator Routine for Multiply Decimal Instruction (DECMP)

The Simulator Control routine (DECENT) gives control to the DECMP routine if either a Multiply Decimal or a Divide Decimal instruction has been encountered. If a specification exception exists for either of the following conditions, control is given to the Analyzer-End routine:

- The actual length of the second operand is greater than eight bytes (i.e., instruction length field, denoted by L2, is greater than seven).

- The actual length of the second operand is equal to or greater than the actual length of the first operand.

After performing the preceding specification checks, control either is given to the divide decimal routine (DECDP), which is discussed in the next section, if the instruction is a Divide Decimal instruction, or remains in the DECMP routine for a Multiply Decimal instruction.

The first operand of the Multiply Decimal instruction is examined to see if it has at least as many bytes of leading zeros as there are bytes in the second operand. If it does not have the required zeros, control is given to the data-check portion of the Analyzer/End routine. Otherwise, multiplication is performed according to one of the three following conditions:

- If either operand is zero, the product field is cleared to zeros (if the first operand was not already zero), and the proper sign is inserted.

- If the multiplicand (the first operand) does not exceed five bytes, both the multiplicand and the multiplier are changed to binary format (since the multiplicand may then be contained in a general register), and the multiplication is carried out using binary multiplication.

- For all other cases (i.e., both operands non-zero and multiplicand size over five bytes), both the multiplier and the multiplicand are split into groups of four digits starting with the low-order sign positions. Initially, the 'lowest-order' group actually consists of three digits and a sign, but the sign is replaced by a zero for the actual processing. For multiplication purposes, each four-digit group is considered to have a positive sign. The algebraically-determined sign for the final product is saved and affixed to the result.

  Each four-digit group is converted to a binary format, and all possible combination of products of the four-digit groups are formed. (Note that each combination consists of one group from the multiplicand and one group from the multiplier.) From these products, partial sums are then formed according to the relative positions of the original four-digit groups. Beginning with the low-order partial sum, each partial sum is converted to decimal, and the low-order four digits (five digits for the first partial sum) of the sum are placed in the product field to the left of the digits already there. The remaining digits of the partial sum constitute a carry-over that is added to the next partial sum, the addition being performed in binary. In the case of the first (low-order) partial sum, the two zeros that had replaced the initial sign digits are truncated (i.e., only three digits are moved to the product field).

  After the last partial sum has been converted to decimal and moved into the product field, the sign of the product is inserted, and the multiplication is finished. Control is then transferred to the Analyzer/End routine.

<u>EXAMPLE OF MULTIPLICATION BY DECIMAL SIMULATOR</u>: Assume a 7-byte multiplicand (operand 1) of 0000000099999C and a 4-byte multiplier (operand 2) of 9999999C. The four bytes of leading zeros in the multiplicand satisfies the requirement of providing space to contain the complete answer. The low-order hexadecimal character ('C') of operand 1 is replaced by a 0, and the resulting low-order four decimal digits

(9990) are converted to binary format (2706_1). [Note: In this example, all binary formatted numbers are expressed in hexadecimal notation.] The next four digits (0099) of operand 1 are also converted to binary foramt (0063_1). The remaining bytes in operand 1 remain as zeros.

In a similar manner, for operand 2, the sign is replaced by a 0, and each group of four digits is converted to binary format. This results in the two groups of four digits: 270F (high-order) and 2706 (low-order).

The pertinent groups of digits from both operands can be arranged in the following manner for describing the next steps.

	B	A
Operand 1 (OP1)	0063	2706
Operand 2 (OP2)	270F	2706

The partial products are formed and stored as the indicated partial sums:

OP2(A) x OP1(A) = 2706 x 2706 = Partial Sum A.

OP2(A) x OP1(B) = 2706 x 0063 = Partial Sum B1.

OP2(B) x OP1(A) = 270F x 2706 = Partial Sum B2.
Partial Sum B1 + Partial Sum B2 equals Partial Sum B.

OP2(B) x OP1(B) = 270F x 0063 = Partial Sum C.

If a storage dump is taken at this point in the problem, the partial sums would appear as the values shown in the boxes that follow.

Partial Sum C	Partial Sum B = (Partial B2 + Partial B1)	Partial Sum A
$000F1ACD_{16}$	$06034AAC_{16}$	$05F2D424_{16}$

The formation of the final product procedes in the following manner:

- Sum A is converted to decimal with a plus sign: ($05F2D424_{16}$=$099800100C_{10}$).
- The two low-order hexadecimal characters (0C) are dropped, and the next three digits (010) are stored in the operand 1 work area as the three low-order digits of the answer.
- The remaining digits (09980) are converted back to binary (26FC) and are added to the initial Sum B to form a new Sum B, ($06034AAC_{16}$+$26FC_{16}$=$060371A8_{16}$).
- Sum B ($060371A8_{16}$) is converted to decimal with a plus sign: ($100889000C_{10}$).
- The sign ('C') is dropped, and the low-order four digits (9000) are placed in the operand 1 work area as part of the final product. The product (at this point) is 9000010.
- The remaining digits (10088) are converted to binary (2768) and are added to the initial Sum C to form a new Sum C, ($000F1ACD_{16}$+2768_{16}=$000F4235_{16}$).
- Sum C ($000F4235_{16}$) is converted to decimal with a plus sign: ($0999989C_{10}$). Since this is the last partial sum in this example, the entire number (without the sign) is placed in the operand 1 area as the rest of the final product. Thus the final value in the operand 1 work area is 09999899000010_{10}.
- The sign of the product is determined by the usual rules of multiplication and replaces the low-order digit in the operand 1 work area. In this example, the sign is plus ('C').
- The answer that is returned to the operand 1 area of the problem program is the 7-byte value $0999989900001C_{10}$.

Simulator Routine for Divide Decimal Instruction (DECDP)

Routine DECDP simulates the decimal divide feature for the)ecimal Simulator. For a Divide Decimal instruction, the Multiply Decimal (DECDMP) routine gives control to the DECDP routine if the specification checks performed by the DECMP routine do not result in an error exit.

Preceding the actual division, if the second operand is found to be zero, a divide check exception occurs, and the error section of the Analyzer/End routine is given control.

To determine if the quotient will fit into the area (i.e., the number of bytes, that is alloted to it, the divisor (the second operand) is aligned with the next to the left-most digit of the dividend (the first operand). When so aligned, the divisor must be larger than the 'aligned' portion of the dividend if the quotient is to fit. If the quotient cannot fit, a divide-check exception occurs, and control is given to the error portion of the Analyzer/End routine.

If the maximum length of the first operand is five bytes or less, the operands are converted to binary format and the division is performed in a general register, using the fixed-point division instruction. Otherwise, the division is carried out by repeated subtraction of multiples of the divisor. Before the actual subtraction process begins, the divisor is left-aligned with the third digit from the left of the dividend. The divisor multiples have the values, respectively, of 8, 4, 2, and 1 times the divisor, and they are subtracted from the dividend at various stages in the process. Each multiple of the divisor corresponds to an appropriate bit to be entered in each 4-bit BCD quotient digit that is formed. If a multiple can be subtracted, its corresponding bit in the appropriate quotient digit is set to 1.

After each quotient digit is formed, the divisor and its multiples are shifted right one digit, and the subtractions are performed again to form the next quotient digit. After each set of four divisor multiples has been subtracted (or at least checked to see if it can be subtracted) from the dividend, the portion of the dividend that remains is referred to as a 'partial dividend.' When the last quotient digit has been formed (as indicated by the divisor and its multiples being right-adjusted in their fields and the partial dividend being less than the divisor), the remaining contents of the dividend field are moved to the remainder field of the answer. The appropriate signs of the quotient and the remainder are inserted, and control is given to the end portion of the Analyzer/End routine.

EXAMPLE OF DIVISION BY DECIMAL SIMULATOR: Assume a six-byte dividend (operand 1) of 00097000000C, and a four-byte divisor (operand 2) of 1000000D. The dividend has been prefaced by leading zeros so that the pre-division check will indicate that the quotient can fit in the alloted area. When this check is performed, the alignment of dividend and quotient appears as follows:

```
Dividend 00097000000C
 Divisor 01000000000C
```

where the divisor appears with a leading zero and three low-order zeros for purposes of alignment. For comparison purposes during simulation, all signs are set positive. When aligned as shown, the divisor is greater than the dividend. This indicates that the quotient will fit in the alloted number of bytes.

To begin the actual simulated division, the divisor is again shifted one digit-place to the right (toward the low-order end), and multiples of the divisor are formed. The alignment then looks like this: (The divisor and its multiples are given a plus ('C') sign.)

Dividend (D)	00097000000C
Divisor First Multiple (D1M)	00100000000C
Divisor Second Multiple (D2M)	00200000000C
Divisor Fourth Multiple (D4M)	00400000000C
Divisor Eighth Multiple (D8M)	00800000000C

The divisor multiples are compared against the dividend in one of the three orders:

- Fourth followed by Eighth followed by First if the eighth multiple is less than or equal to the dividend.

- Fourth followed by Eighth if the fourth multiple is less than or equal to the dividend, followed by Second if the eighth multiple is greater than the dividend, followed by the First.

- Fourth followed by Second if the fourth multiple is greater than dividend, followed by First.

If the dividend is less than the first multiple, a zero (0) is entered as the corresponding quotient digit and all divisor multiples are shifted one place toward the low-order side (to the right), and a new round of comparisons is undertaken.

For each multiple that can be subtracted, the appropriate bit in the quotient digit is set to 1. Each round of comparisons seeks to locate the largest multiple(s) than can be subtracted from the dividend.

Steps 1 through 3b in Figure 11-5 illustrate the compare and shifting operations that are performed and the formation of the quotient and remainder digits.

	Step 1	Step 2	Step 2a
Dividend (D)	00097000000C	00097000000C	00017000000C
Divisor First Multiple (D1M)	00100000000C 3rd comp.	00010000000C	00010000000C 3rd comp.
Divisor Second Multiple (D2M)	00200000000C 2nd comp.	00020000000C	00020000000C
Divisor Fourth Multiple (D4M)	00400000000C 1st comp.	00040000000C 1st comp.	00040000000C
Divisor Eighth Multiple (D8M)	00800000000C	00080000000C 2nd comp.	00080000000C
	Since D1M>D, a zero is entered as the first quotient digit. All multiples are shifted one digit to the right, and step 2 is performed.	Since D8M<D, the second quotient digit's "8-bit" is set to 1. The D8M is subtracted from D to form a new D (value) for step 2a.	Since a given quotient digit cannot be greater than 9 and the second digit's "8-bit" is set to 1, the only comparison that can be made is with the D1M (corresponding to a "1-bit"). Since D1M<D, the second digit's "1-bit" is set to 1. Thus the second digit is 9 as a result of both the "8-bit" and the "1-bit" being set to 1. The D1M is subtracted from D to form a new D (value) for step 3. All multiples are shifted one digit to the right.

	Step 3	Step 3a	Step 3b
Dividend (D)	00007000000C	00003000000C	00001000000C
Divisor First Multiple (D1M)	00001000000C	00001000000C	00001000000C 4th comp.
Divisor Second Multiple (D2M)	00002000000C	00002000000C 3rd comp.	00002000000C
Divisor Fourth Multiple (D4M)	00004000000C 1st comp.	00004000000C	00004000000C
Divisor Eighth Multiple (D8M)	00008000000C 2nd comp.	00008000000	00008000000C
	Since D4M<D, and D8M>D, the third quotient digit's "4-bit" is set to 1. The D4M is subtracted from D to form a new D (value) for step 3a.	Since D2M<D, the third digit's "2-bit" is set to 1. The D2M is subtracted from D to form a new D (value) for step 3b.	Since both the "4-bit" and the "2-bit" have been set for this quotient digit, the only comparison that can be made is with the D1M (see step 2a). Because the D1M=D, the "1-bit" for this digit can be set to 1. Thus, the third digit is 7 as a result of the "4-bit," the "2-bit," and the "1-bit" all being set to 1. The D1M is subtracted from D to give the value zero, which becomes the remainder in this example. (Both the dividend and the divisor multiples are now right-adjusted so no further shifting occurs and the division is complete.)

After step 3b in the preceding process has been completed, the three quotient digits that were formed are 097, and the remainder is zero. The sign of the remainder becomes the same as the sign of operand 1 (the dividend). In this example, the sign is plus ('C'). The sign of the quotient is plus if both operands have the same sign. Otherwise, the quotient sign is minus. In this example, the quotient sign is minus ('D'). The final result is 097D0000000C.

• Figure 11-5. Example of Division by Decimal Simulator

Simulator Routine for Compare Decimal Instruction (DECCP)

The comparison of the two decimal operands is made by the DECCP routine. If both operands are zero, they are considered equal regardless of their signs. If one operand is non-zero and the other one is zero, the non-zero operand is considered greater if it is positive, and it is considered less if it is negative.

If two non-zero operands are to be compared, each is extended in the work area (by adding leading zeros) to 31 digits plus the sign before comparison. The absolute values of the operands are than compared logically. The operand that is greater in absolute value is considered to be greater if it is positive but less if it is negative.

If both operands have the same absolute value and sign, they are equal. If the absolute values are equal but the signs are different, the positive operand is considered to be the greater.

Control is given to the end portion of the Analyzer/End routine after the result of the comparison has been determined.

Analyzer/End Routine

The Analyzer/End routine is given control to handle the termination procedures for the Decimal Simulator routine. If errors have been recognized by any of the preceding simulation routines, control is given to the analyzer (or error-handling) section of the routine. When the simulation of an instruction is completed successfully, or after the error-handling section has performed certain functions, control is given to the end section of the Analyzer/End routine.

The analyzer section establishes the appropriate interruption code and places this code in the old PSW. In the case of a decimal overflow exception, bit 37 of the PSW is checked to determine whether the user or the operating system is to handle the error. If the decimal overflow is to be handled as a system error, the analyzer section retains control. Otherwise, control is given to the end section.

If there exists data that is not to be returned to the user, register addresses are moved to preclude the transfer of this data to the user's result area.

The end section of the Analyzer/End routine handles the return of control to the source from which the Decimal Simulator routine received control. For a successful simulation, the result obtained from the appropriate simulation routine is moved to the user's area. In the case of a decimal overflow condition which the user chooses to ignore, a result truncated to the length specified in the instruction is moved to the user's area.

The end section gives control to the PFLIH routine if an error condition arises during the simulation process. (Note: If a decimal overflow condition is to be ignored, the Analyzer/End routine does not consider the overflow as an error condition.) Otherwise, by testing the 'return-to-TESTRAN' flag bit in the task control block, the end section determines the routine (e.g., problem program or TESTRAN) to which control is to be returned.

The following control blocks, tables, and related areas are included in this section.

Name of Table, Control Block, or Related Area   Page

ABDUMP Parameter List.. 295
Allocated Queue Element (AQE).. 286
Block Extent List and Note List.. 276
Communications Vector Table (CVT).. 246
Contents Directory Element (CDE).. 271
Control and Relocation Dictionary Record.. 282
Control Record.. 280
Descriptor Queue Element (DQE).. 285
Display Control Module (DCM).. 297
Dummy Partition Queue Element (DPQE).. 289
Entry Table.. 284
Event Control Block (ECB).. 265
Fail Soft Storage Element Map (FSSEMAP).. 301
Free Block Queue Element (FBQE).. 289
Free Queue Element (FQE).. 286
GOVRFLB (Origin List for Main Storage Queues).. 287
Interruption Request Block (IRB).. 256
Interruption Queue Element (IQE).. 269
Load List Element (LLE).. 272
Major Queue Control Block (QCB).. 267
Minor Queue Control Block (QCB).. 267
Multiprocessing Communications Vector Table (MPCVT).. 300
Parameter List Element for the ENQ/DEQ Routines.. 266
Partition Queue Element (PQE).. 286
Partitioned Data Set Directory Entry.. 272
Program Fetch Buffer Table.. 279
Program Fetch Work Area.. 278
Program Interruption Control Area (PICA).. 263

Name of Table, Control Block, or Related Area (Continued) Page

Program Interuption Element (PIE).......... 263

Program Request Block (PRB).......... 258

Queue Element (QEL).......... 268

Relocation Dictionary (RLD) Record.......... 281

Reply Queue Element.......... 290

Request Queue Element (RQE).......... 270

Rollout I/O Queue Element (RIQE).......... 290

Sample Dump.......... 302

Scatter Extent List.......... 275

Scatter Translation Record.......... 277

Secondary Communications Vector Table.......... 294

Segment Table.......... 283

STAE Control Block (SCB).......... 264

Subpool Queue Element (SPQE).......... 285

Supervisor Request Block (SVRB) -- for Nonresident Routine.......... 255

Supervisor Request Block (SVRB) -- for Resident Routine.......... 254

SVC Purge Parameter List.......... 291

SVC Table.......... 245

System Interruption Request Block (SIRB).......... 257

Task Control Block (TCB).......... 250

Time-Slice Control Element (TSCE).......... 296

Timer Queue Element (TQE).......... 292

Trace Table (Uniprocessing Systems).......... 260

Trace Table (Multiprocessing Systems).......... 261

Transient Area Control Table (TACT).......... 262

Vary Queue Element (VQE).......... 300

SVC TABLE

Entries for resident SVC routines

Bytes	1 byte		3 bytes
0	10	Zeros	1 Main Storage Address
4	10	0 ——— 0	5
8	10	0 ——— 0	9
12	10	0 ——— 0	13
16	10	0 ——— 0	17
20	10	0 ——— 0	21
24	10	0 ——— 0	25
28	10	0 ——— 0	29
32	10	0 ——— 0	33
36	10	0 ——— 0	37
40	10	0 ——— 0	

Entries for transient SVC routines

11	2 Length 13	14 Relative Track and Record Address 31
11		
11		
11		
11		
11		
11		
11		
11		
11		
11		
11		

Entry for each SVC routine = 4 bytes

NOTE : '10' flag in two high-order bits indicates a resident routine.

'11' flag in two high-order bits indicates a transient (non-resident) routine.

COMMUNICATIONS VECTOR TABLE (CVT)

The Communication Vector Table provides the means whereby nonresident routines may refer to information in the nucleus of the control program. The CVT is part of the resident nucleus.

The symbolic displacements below are generated in nonresident routines by use of the CVT macro instruction. The address of the first location of the CVT is placed in main storage location hex 10 during nucleus initialization.

The following table shows the relative locations of the entries in the CVT.

(Part 1 of 4)

Hex	Dec	4 bytes
-4	-4	CVTRELNO release number of operating system in use
0	0	CVTTCBP pointer to addresses for next and current TCB
4	4	CVT0EF00 addr Stage 2 Exit Effector
8	8	CVTLINK addr of DCB for SYS1. LINKLIB
C	12	CVTJOB addr of work queue control blocks
10	16	CVTBUF addr of buffer for Resident Console Interruption routine
14	20	CVTXAPG addr of IOS appendage table
18	24	CVT0VL00 entry-point addr of Validity Check routine
1C	28	CVTPCNVT entry-point addr of routine for converting relative track addr to absolute
20	32	CVTPRLTV entry-point addr of routine for converting absolute track addr to relative
24	36	CVTILK1 addr of channel and control unit section in UCB lookup table
28	40	CVTILK2 addr of UCB addr list section in UCB lookup table
2C	44	CVTXTLER entry-point addr to Stage 3 Exit Effector for system error routines
30	48	CVTSYAD addr of system residence volume entry in UCB lookup table
34	52	CVTBTERM entry-point addr of ABTERM routine
38	56	CVTDATE current date in packed decimal

• COMMUNICATIONS VECTOR TABLE (CVT) (Part 2 of 4)

Hex	Dec		
3C	60	CVTMSLT addr of master schedule linkage table	
40	64	CVTZDTAB addr of I/O device characteristic table	
44	68	CVTXITP addr of Error Interpreter routine	
48	72	CVTXWTO addr of System Qiesce routine	
4C	76	CVT0FN00 reserved	
50	80	CVTEXIT an SVC 3 instruction	CVTBRET a BCR 15, 14 instruction
54	84	CVTSVDCB addr of DCB for SYS1. SVCLIB data set	
58	88	CVTTPC addr of pseudo clocks for timer routine (SHPC first)	
5C	92	CVTPBLDL branch and link entry-point addr to BLDL routine	
60	96	CVTSJQ reserved	
64	100	CVTCUCB addr of table of pointers to console unit control blocks	
68	104	CVTQTE00 addr of Timer Enqueue routine (IEAQTE00) in Timer Second-Level Interruption Handler	
6C	108	CVTQTD00 addr of Timer Dequeue routine (IEAQTD00) in Timer Second-Level Interruption Handler	
70	112	CVTSTB addr of I/O device statistics table	
74	116	System Configuation X'10'- Uniprocessing X'14'- Multiprocessing	CVTDCB addr of DCB for SYS1. LOGREC data set

COMMUNICATIONS VECTOR TABLE (CVT) (Part 3 of 4)

Hex	Dec	
78	120	CVTIOQET Addr of I/O Request Element Table
7C	124	CVTIXAVL Addr of IOS Freelist Pointer
80	128	CVTNUCB Lowest Storage Addr Not in Nucleus
84	132	CVBOSV Addr of Program Fetch Routine
88	136	CVTODS Entry-Point Addr of Dispatcher
8C	140	CVTILCH Addr of Logical-Channel Word Table
90	144	CVTIERLC Addr of Logical-Channel Error Queue
94	148	CVTMSER Addr of Master Scheduler Resident Data Area
98	152	CVTOPT01 Branch Entry-Point Addr of Post Routine Used by the I/O Supervisor
9C	156	CVTTRMTB Addr of Terminal Table for QTAM
A0	160	CVTHEAD Addr of Highest Priority TCB in TCB Queue
A4	164	CVTMZ00 Highest Storage Addr in Machine
A8	168	CVT1EF00 Addr of Stage 1 Exit Effector
AC	172	
B0	176	

• COMMUNICATIONS VECTOR TABLE (CVT) (Part 4 of 4)

Hex	Dec	
B4	180	CVTQCDSR addr of search routine for contents directory
B8	184	CVTQLPAQ pointer to address of first CDE in LPA queue
BC	188	CVTMPCVT addr of Multiprocessing Communications Vector Table
C0	192	CVTQPGTM entry-point addr to EOT Purge Timer routine
C4	196	CVTABEND addr of secondary CVT
C8	200	CVTQABL entry-point addr to Release Loaded Programs routine
CC	204	CVTQSPET entry-point addr to Release Main Storage routine
D0	208	CVTQABST an SVC 13 instruction
D4	212	CVTTSCE address of first time-slice control element (TSCE)
D8	216	

• TASK CONTROL BLOCK (TCB)

Byte Displacement						
-32	TCBFRS0 Save Area for Floating Point Register Zero					
-24	TCBFRS2 Save Area for Floating Point Register 2					
-16	TCBFRS4 Save Area for Floating Point Register 4					
-8	TCBFRS6 Save Area for Floating Point Register 6					
0	0	1 TCBRBP Address of Last RB in RB Queue		4 0		5 TCBPIE Address of PIE
8	0	9 TCBDEB Address of Last DEB in DEB Queue		12 0		13 TCBTIO Address of Task I/O Table
16	TCBCMP Task Completion Code			20 0		21 TCBTRN Address of TESTRAN Control Area
24	TCBNROC (See Description)	25 TCBMSS Address of Last SPQE in SPQE Queue		28 TCBPKF Bits 0-3 = Protect Key; Bits 4-7 =0		29 TCBFLGS Flags: See Description
32	TCBFLGS Cont.	34 TCBLMP Limit Priority	35 TCBDSP Dispatching Priority	36 0		37 TCBLLS Address of Last Load List Element in The Load List
40	0	41 TCBJLB Address of The Job Library DCB		JPQ Purge Flag 44.0	0 44.1	45 TCBJPQ Address of The Last CDE for The Job Pack Area
48	TCBGRS0 Save Area for Gen. Reg. 0			TCBGRS1		
56	TCBGRS2			TCBGRS3		
64	TCBGRS4			TCBGRS5		
72	TCBGRS6			TCBGRS7		
80	TCBGRS8			TCBGRS9		
88	TCBGRS10			TCBGRS11		
96	TCBGRS12			TCBGRS13		
104	TCBGRS14			TCBGRS15		
112	TCBQEL Enqueue Count	113 TCBFSA Address of First P.P. Save Area for This Task		116 0		117 TCBTCB Address of Next TCB on TCB Queue (In Rollout TCB, contains Address of First Transient Area TCB)
120	0	121 TCBTME Address of Timer Queue Element for This Task		124 0		125 TCBJSTCB Address of the Job Step TCB
128	0	129 TCBNTC Address of Next TCB Attached by Originating Task (Always 0 in Rollout TCB)		132 0		133 TCBOTC Address of Originating or Parent TCB
136	0	137 TCBLTC Address of Last TCB on Subtask Queue (Always 0 in Rollout TCB)		140 0		141 TCBIQE Address of The IQE for Scheduling an End-of-Task Exit Routine
144	0	145 TCBECB Address of ECB to be Posted When This Task is Complete		148 0		149 Reserved
152	0	153 TCBPQE Address of Dummy PQE -8 (First Element on PQE List for Job Step)		156 0		157 TCBAQE List Origin of Allocated Queue Elements for This Task
160	STAE Flags	161 TCBNSTAE Address of STAE Control Block (SCB)		164 0		165 TCBTCT Reserved
168	TCBUSER Field to be used by users					

Description of Field

TCBNROC: "Nonrolloutable Count" field. Meaningful only in a job step TCB. When zero, indicates that job step is eligible to be rolled out. When nonzero, indicates that job step is not eligible for rollout. Initialized by the Attach routine from input parameters provided by job step's initiator. The count is increased by the ENQ routine and decreased by the DEQ routine; it is tested by the TESTSTEP routine of the rollout module.

Description of TCB Flags

Offset	Symbol	Meaning (when flag is set)
29.0	TCBFA	Indicates that abnormal termination, performed by the ABEND routine, is in progress for this task.
29.1	TCBFE	Indicates that normal termination, performed by the EOT routine, is in progress for this task.
29.2	TCBFERA	Indicates that the Erase Phase routine is to be entered when the ABEND routine is again executed for this task.
29.3-29.4		Reserved.
29.5	TCBFT	Indicates that this task is currently the top task of a tree of tasks being abnormally terminated.
29.6	TCBFS	Indicates that an abnormal termination dump has been performed for this task.
29.7	TCBFX	Prohibits asynchronous exits from being scheduled for this task.
30.0	TCBFOINP	Indicates that the dump data set for the job step is being opened.
30.1	TCBFSTI	Indicates that a job step interval requested by an initiator has expired. (Set in the TCB of an initiator task.)
30.2	TCBFRA	Meaningful only in a job step TCB. When '1', indicates that job step can cause rollout. When '0', indicates that job step cannot cause rollout. Initialized by the Attach routine from input parameters provided by job step's initiator.
30.3	TCBFSMC	Indicates that this task is in "system must complete" status.
30.4	TCBFJMC	Indicates that this task in in "job step must complete" status.
30.5	TCBFDSOP	Indicates that the ABEND routine has previously opened the dump data set for this job step. (Set in the job step TCB.)
30.6	TCBFETXR	Indicates that an end-of-task exit (ETXR) routine is to be scheduled for the task that attached this task.
30.7	TCBFTS	Indicates member of time-slice group.
31.0	TCBFSM	Indicates that the RB old PSW for all programs executed as part of this task should be set to supervisor state.
31.1	TCBFRI	"Rollout Invoked" flag. Meaningful only in a job step TCB. When '1', indicates that job step has had one or more main storage requests satisfied from outside its region (via the rollout mechanism). "Borrowed" space is still allocated to the step. When '0', indicates that job step has not invoked rollout.
31.2	TCBABTRM	Prevents multiple scheduling of the ABEND routine by the ABTERM routine. Also indicates that the operands of the ABEND macro instruction have been saved in the TCBCMP field.

31.3 TCBOPEN Indicates that an OPEN macro instruction has been issued for the dump data set for this task. Also used with TCBREC (31.7) to indicate a valid reentry to the ABEND routine.

31.4 TCBADUMP Indicates that the ABDUMP routine is in progress for this task. Also used with TCBREC (31.7) to indicate a valid reentry to the ABEND routine.

31.5 TCBPDUMP Indicates that no abnormal termination dumps are to be taken for any task within the job step. Set in the job step TCB.

31.6 TCBCLOSE Indicates that a CLOSE macro instruction has been issued during ABEND processing. Also used with TCBREC (31.7) to indicate a valid reentry to the ABEND routine.

31.7 TCBREC In conjunction with either TCBADUMP (31.4), TCBOPEN (31.3), or TCBCLOSE (31.6), indicates a valid reentry to the ABEND routine.

32.0 TCBNDUMP Indicates that the ABDUMP routine has made this task nondispatchable while it is displaying dynamic queues.

32.1 TCBSER Indicates that this task is nondispatchable while the SER1 routine is being executed for this task.

32.2 TCBRQENA Indicates to the I/O Supervisor that there are no more request queue elements.

32.3-32.4 Reserved as status bits to indicate nondispatchability.

32.5 Indicates that this task is nondispatchable because VARY or QUIESCE processing is being done in a multiprocessing system.

32.6 Reserved as status bit to indicate nondispatchability.

32.7 TCBONDSP Indicates that the current task, which is abnormally terminating, is nondispatchable while the dump data set is being opened for another task in the same job step.

33.0 TCBFC Indicates that this task has terminated, normally or abnormally, and is nondispatchable.

33.1 TCBABWF Indicates that this task is nondispatchable because it is to be terminated by the ABEND routine.

33.2 TCBWFC "Wait for Core" nondispatchability flag. If set, indicates that this task is waiting for a space request to be satisfied by the rollout mechanism. Meaningful in all TCBs except those for permanent system tasks.

33.3 TCBFRO "Rolled Out" nondispatchability flag. If set, indicates that this task is nondispatchable because it has been rolled out. This flag is set in all TCBs of a rolled-out job step, including the TCB of the associated initiator.

33.4 TCBSYS Indicates that this task is nondispatchable because another task is in "system must complete" status.

33.5 TCBSTP Indicates that this task is nondispatchable because another task in the same job step is in "step must complete" status.

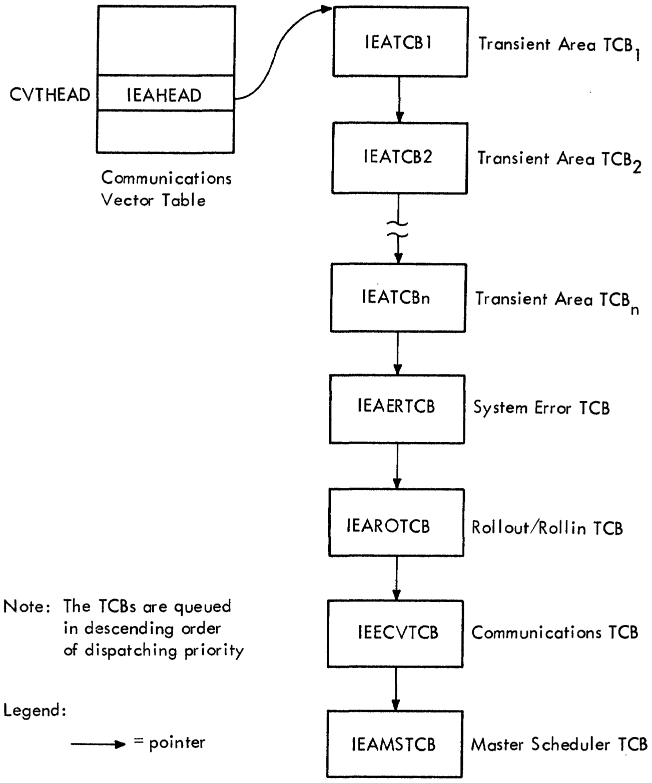
33.6 TCBFCD1 Indicates that this task is nondispatchable because it is an initiator task that is waiting for a requested region of main storage.

33.7 Reserved.

Description of STAE flags (byte 160)

Bit	Indication
0	The ABEND routine was entered because of an error which occurred during STAE processing.
1	The STAE routine invoked the Purge I/O routine with the quiesce I/O option.
2	The current SCB has the XCTL=YES option.
3	The SCB was created by a program that is scatter loaded.
4	The Purge I/O routine did not successfully quiesce I/O, but I/O was halted.
5	The program using STAE is in supervisor mode.
6	The STAE user requested that a retry routine be scheduled but that the RB chain not be purged.
7	The retry routine and parameter list addresses are both valid.

Positions of Permanent System TCBs on TCB Queue



SUPERVISOR REQUEST BLOCK (SVRB) -- FOR RESIDENT ROUTINE

Bytes 0	Reserved			4 RBABOPSW Bits 32-63 of User PSW	
8	RBWCSA Wait Count Save Area	9 RBSIZE Size in Double Words, of RB	10 RBSTAB * Status and Attribute Bits	12 RBCDFLGS * Contents Control Flags	13 RBCDE Address of Contents Directory Entry Used by The Link Routine When Forming a PRB
16	RBOPSW				
24	0	25 RBPGMQ Queue Field for Serially Reusable Programs		28 RBWCF Wait Count	29 RBLINK Address of Next RB on RB Queue
32	RBGRSAVE Register 0			36	Register 1
40	Register 2			44	Register 3
48	Register 4			52	Register 5
56	Register 6			60	Register 7
64	Register 8			68	Register 9
72	Register 10			76	Register 11
80	Register 12			84	Register 13
88	Register 14			92	Register 15

RBEXSAVE

Extended Save Area for SVC Routines; Length = 48 Bytes

* Described under "Description of RB Flags"

SUPERVISOR REQUEST BLOCK (SVRB) -- FOR NONRESIDENT ROUTINE

<table>
<tr><td>Bytes 0</td><td colspan="2">RBTABNO
Displacement of TACT Entry</td><td colspan="2">RBRTLNTH
Length in Bytes of SVC Routine
2</td><td colspan="4">RBABOPSW
Bits 32 63 of User PSW
4</td></tr>
<tr><td>8</td><td>RBWCSA
Wait Count
Save Area</td><td>RBSIZE
Size in Double
9 Words of RB</td><td colspan="2">RBSTAB *
Status and Attribute Bits
10</td><td colspan="4">RBSVTON
Address of Next RB on Transient Area Queue
12</td></tr>
<tr><td>16</td><td colspan="8">RBOPSW
RB Old PSW</td></tr>
<tr><td>24</td><td>RBTAWCSA
Wait Count Save
Area for Transient
Area Handling</td><td colspan="3">RBSVTTR
TTR for SVC Routine
25</td><td>RBWCF
Wait Count
28</td><td colspan="3">RBLINK
Address of Next RB on RB
Queue for Task</td></tr>
<tr><td>32</td><td colspan="8">Remainder of SVRB Same as SVRB for Resident Routine</td></tr>
</table>

* Described under "Description of RB Flags"

INTERRUPTION REQUEST BLOCK (IRB)

Bytes				
0	RBTMFLD[1] Flags for Timer Routines	1 RBPPSAV Address of PP Register Save Area	4 RBABOPSW Bits 32-63 of User's PSW	
8	RBWCSA Wait Count Save Area	9 RBSIZE Size, in Double Words of RB 10 RBSTAB[1] Status and Attribute Bits	12 RBEP Entry Point Address	
16	RBOPSW			
24	RBUSE[2] Attach Use Count	25 RBIQE[3] List Origin for IQEs	28 RBWCF Wait Count	29 RBLINK Address of Next RB or RB Queue
** 24	Reserved	26 RbIQE[3] List Origin for RQEs		
32	Register 0	RBGRSAVE	36 Register 1	
40	Register 2		44 Register 3	
48	Register 4		52 Register 5	
56	Register 6		60 Register 7	
64	Register 8		68 Register 9	
72	Register 10		76 Register 11	
80	Register 12		84 Register 13	
88	Register 14		92 Register 15	
96	RBNEXAV[4] Address of Next Available IQE		100	
	IQE Work Space ***			

1 Described under topic "Description of RB Flags".

2 The RBUSE field is used only when the IRB schedules an end - of - task exit (ETXR) routine.

3 The RBIQE field will be either 2 or 3 bytes in length, depending on the type of queuing element (IQE or RQE).

4 The RBNEXAV field and the IQE work space are available only in IRBs for which this work space was requested via CIRB macro instruction.

SYSTEM INTERRUPTION REQUEST BLOCK (SIRB)

Bytes				
0	RBEXRTNM 1-8 Character Name of Error Exit Routine. First Four Characters are IGE0. Last Four are Unpacked Decimal Characters.			
8	RBWCSA Wait Count Save Area	9 RBSIZE Size in Double Words of RB	10 RBSTAB* Status and Attribute Bits	12 RBEP Entry Point Address
16	RBOPSW			
24	Reserved	26 RBIQE List Origin for RQEs	28 RBWCF Wait Count	29 RBLINK Address of Next RB on RB Queue
32	RBGRSAVE Register 0		36 Register 1	
40	Register 2		44 Register 3	
48	Register 4		52 Register 5	
56	Register 6		60 Register 7	
64	Register 8		68 Register 9	
72	Register 10		76 Register 11	
80	Register 12		84 Register 13	
88	Register 14		92 Register 15	

* Described under "Description of RB Flags"

PROGRAM REQUEST BLOCK (PRB)

<table>
<tr><td>Bytes 0</td><td colspan="3">Reserved</td><td colspan="2">RBABOPSW
Bits 32-63 of User's PSW
4</td></tr>
<tr><td>8</td><td>RBWCSA
Wait Count Save Area</td><td>RBSIZE
Size, in Double Words, of RB
9</td><td>RBSTAB *
Status and Attribute Bits
10</td><td>RBCDFLGS *
Contents Control Flags
12</td><td>RBCDE
Address of Contents Directory Entry
13</td></tr>
<tr><td>16</td><td colspan="5">RBOPSW</td></tr>
<tr><td>24</td><td>0</td><td colspan="2">RBPGMQ
Queue Field for Serially Reusable Programs
25</td><td>RBWCF
Wait Count
28</td><td>RBLINK
Address of Next RB on RB Queue
29</td></tr>
</table>

* Described under "Description of RB Flags"

Description of RB Flags

RBSTAB field (all RB types):

Offset	Symbol	Meaning (bit is set, unless otherwise indicated)
00-01	RBFTP	RB type: 00 = PRB 01 = IRB 10 = SIRB 11 = SVRB
02	reserved	
03	RBFNSVRB	Indicates SVRB for a transient (nonresident) SVC routine.
04	reserved	
05	reserved	
06	reserved	
07	reserved	
08	RBTCBNXT	Indicates that the RBLINK field points to a TCB.
09	RBFACTV	Indicates that the IRB or SIRB is queued to a TCB.
10	reserved	
11	reserved	
12-13	RBIQETP	(Meaningful only with an IRB or SIRB.) Distinguishes asynchronous exit queue element type: 00 = RQE is not to be queued to "next available" list (IECNXAVL) by the Exit routine. (Since the RB is an SIRB, the RQE has already been queued by the error exit routine.) 01 = IRB has asynchronous exit queue elements that are RQEs. 10* = IQE is not to be queued to "next available" list (RBNEXAV) by the Exit routine. (These bit settings are used with the rollout IRB.)

		11* = IRB has asynchronous exit queue elements that are IQEs. IQE is to be queued to "next available" list by the Exit routine.
		*The RETIQE operand of the CIRB macro instruction determines these settings. If RETIQE = YES, '11' is set. If RETIQE = NO, '10' is set. (If the operand is not specified, '11' is set.) <u>Note</u>: If rollout is included during system generation, the Nucleus Initialization Program issues the CIRB macro instruction to create and initialize the rollout IRB.
14	RBFDYN	RB space can be freed at exit.
15	RBECBWT	0 = Wait for single event or for N of N events.
		1 = Wait for M of N events (where M is less than N).

RBTMFLD field (IRB only)

00	RBTMQUE	Timer element not on queue
01	RBTMTOD	Local TOD option used
02-03	RBTMIND1	00 = TUINTVL requested
		01 = BINTVL requested
		10 = reserved
		11 = DECINTVL requested
04	RBTMCMP	Interval is complete.
05	RBTMIND2	Indicates midnight supervisory timer element.
06-07	RBTMIND3	00 = task request
		01 = Wait request
		10 = Supervisory element
		11 = RBAL request

RBCDFLGS field (PRB and SVRB for resident routine)

00	reserved	
01	reserved	
02	reserved	
03	reserved	
04	WAE	Work area exists.
05	RBCDSYNC	SYNCH macro instruction issued.
06	RBCDXCTL	XCTL macro instruction issued.
07	RBCDLD	LOAD macro instruction issued.

• TRACE TABLE (UNIPROCESSING SYSTEMS)

NOTE: Each entry is eight words

SIO Instruction:

Words	0	1	2	3	4	5	6	7
	(See Below)	Channel Address Word	Channel Status Word		Reg. 1	0	TCB	Timer Contents

First Word of SIO Entry:

0	2	3	13		21 31
			0		Device Address

SIO Condition Code

I/O Interruption:

Words	0	1	2	3	4	5	6	7
	Bit 13 = 1 Bits 16 - 19 = 0101		Channel Status Word		Reg. 1	0	TCB	Timer Contents

I/O Old PSW

SVC Interruption:

Words	0	1	2	3	4	5	6	7
	Bit 13 = 1 Bits 16 - 19 = 0010		Reg. 15	Reg. 0	Reg. 1	0	TCB	Timer Contents

Program Interruption:

Words	0	1	2	3	4	5	6	7
	Bit 13 = 1 Bits 16 - 19 = 0011		Reg. 15	Reg. 0	Reg. 1	0	TCB	Timer Contents

Program OPSW

External Interruption:

Words	0	1	2	3	4	5	6	7
	Bit 13 = 1 Bits 16-19 = 0001		Reg. 15	Reg. 0	Reg. 1	0	TQE if Timer Interruption Otherwise, Zero	Timer Contents

External Old PSW

Dispatcher:

Words	0	1	2	3	4	5	6	7
	Bit 13 = 1 Bits 16 - 9 = 1101		Reg. 15	Reg. 0	Reg. 1	0	TCB	Timer Contents

The addresses of the trace table are contained in a 12 - byte field whose address is at hex loc 54. The format of the field is:

Bytes	0 3	4 7	8
	Address of Last Entry	Address of Table Beginning	Address of Table End

• TRACE TABLE (MULTIPROCESSING SYSTEMS)

TRACE TABLE (Multiprocessing Systems)
NOTE: Each entry is eight words
SIO Instruction:

Words	0	1	2	3	4	5	6	7	
	(See Below)	Channel Address Word	Channel Status Word		TCB	'Old' TCB of CPUA	'Old' TCB of CPUB	Timer Contents	CPUID

First Word of SIO Entry:

0	2	3		13	21	31
			0		Device Address	

SIO Condition Code

I/O Interruption:

Words	0	1	2	3	4	5	6	7	
	Bit 13 = 1 Bits 16 - 19 = 0101		Reg 15	Reg 0	Reg 1	'Old' TCB of CPUA	'Old' TCB of CPUB	Timer Contents	CPUID

I/O Old PSW

SVC Interruption:

Words	0	1	2	3	4	5	6	7	
	Bit 13 = 1 Bits 16 - 19 = 0010		Reg 15	Reg 0	Reg 1	'Old' TCB of CPUA	'Old' TCB of CPUB	Timer Contents	CPUID

Program Interruption:

Words	0	1	2	3	4	5	6	7	
	Bit 13 = 1 Bits 16 - 19 = 0011		Reg 15	Reg 0	Reg 1	'Old' TCB of CPUA	'Old' TCB of CPUB	Timer Contents	CPUID

Program Old PSW

SSM Program Interruption (Multisystem Mode)

Words	0	1	2	3	4	5		6	7	
	Bit 13 = 1 Bits 16 - 19 = 0100		Reg 15	Reg 0	Reg 1		'Old' TCB of CPUA	'Old' TCB of CPUB	Timer Contents	CPUID

Program Old PSW

CPUID of locking CPU

External Interruption:

Words	0	1	2	3	4	5	6	7	
	Bit 13 = 1 Bits 16 - 19 = 0001		Reg 15	Reg 0	Reg 1	STMASK of other CPU	TQE if timer interruption otherwise, zero	Timer Contents	CPUID

External Old PSW

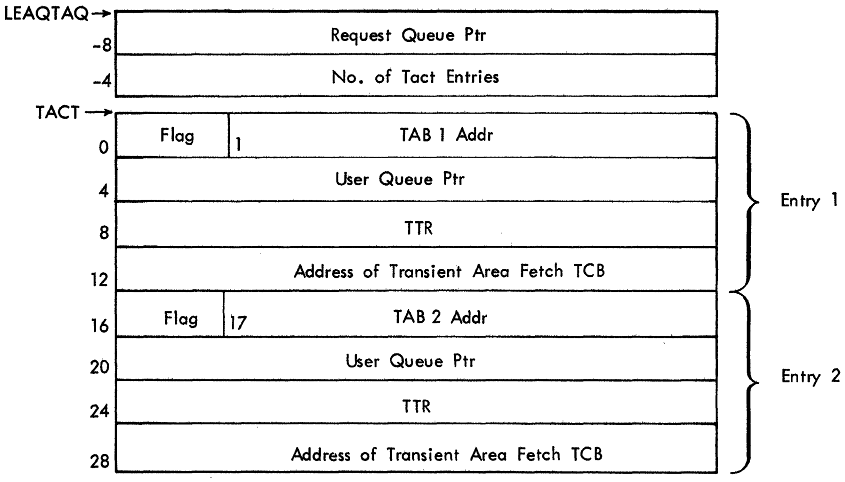
Dispatcher:

Words	0	1	2	3	4	5	6	7	
	Bit 13 = 1 Bits 16 - 19 = 1101		Reg ı5	Reg 0	Reg 1	'New' TCB of CPUA	'New' TCB of CPUB	Timer Contents	CPUID

The addresses of the trace table are contained in a 12-byte field whose address is at hex loc 54. The format of the field is:

Bytes	0 3	4 7	8
	Address of Last Entry	Address of Table Beginning	Address of Table End

TRANSIENT AREA CONTROL TABLE (TACT)

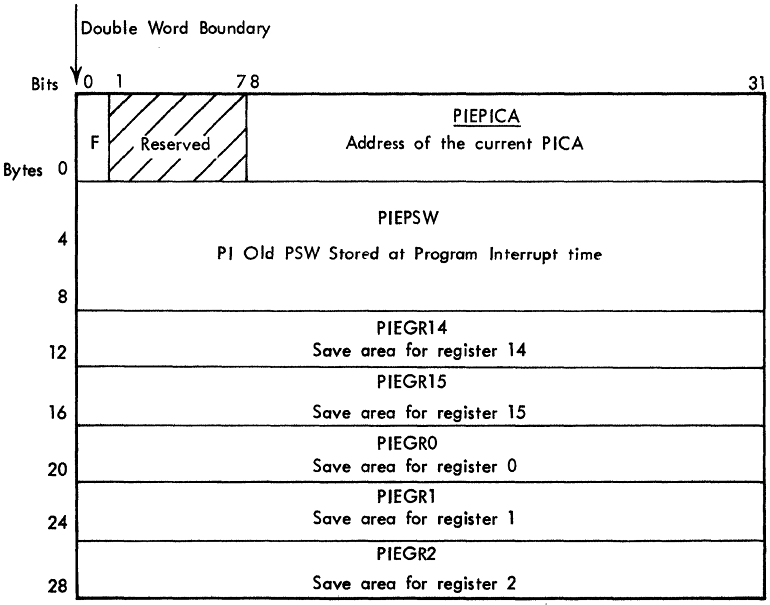


Description of the Transient Area Control Table

There is one four-word entry for each transient area block (TAB) in the system. Each entry has the format:

Byte	Contains
0	Flags: X'40' - TAB is being loaded X'20' - TAB is free (unoccupied) X'00' - TAB is being used
1-3	Address of associated TAB.
4-7	Pointer to user queue for associated TAB.
8-11	Track address in SVC Library of the routine currently in the TAB. This address is used to identify the routine.
12-15	Address of transient area fetch TCB under whose control routines are fetched to the TAB.

PROGRAM INTERRUPTION ELEMENT (PIE)



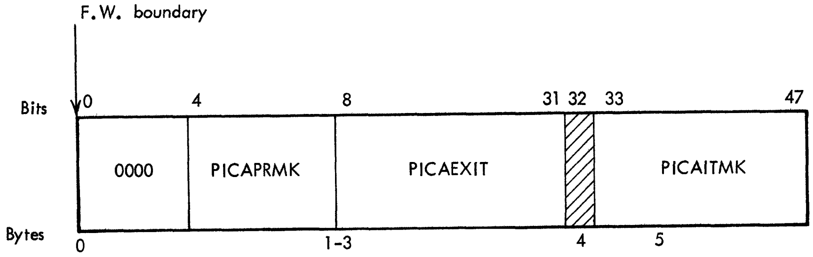
Description of Field

F: Flag bit which, if set to one, indicates that the task cannot accept further PI's. (This bit is set whenever a user PI exit routine is entered. It is reset by the SVC Exit routine).

This bit is called the first-time logic switch.

(All other fields are described in the figure above.)

PROGRAM INTERRUPTION CONTROL AREA (PICA)



Description of Fields

PICAPRMK: Program mask to be used in the PSW when the programs of the task are executing.

PICAEXIT: Address of the user's program interruption exit routine to be given control when a program interruption of specified type occurs.

PICAITMK: Mask which indicates 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 user interest in that type. Bit 0 in the field is reserved.

• STAE CONTROL BLOCK (SCB)

Bytes		
0	Reserved	1 Address of Previous SCB
4	Address of STAE Exit Routine	
8	Reserved	9 Address of STAE Exit Routine Parameter List
12	Flags	13 Address of RB

Description of Fields

Byte **Contents**

0 Reserved

1-3 Address of the previous SCB for this task or zero if this is the first SCB created for this task.

4-7 Address of the user-written STAE exit routine as specified in the STAE macro instruction.

8 Reserved

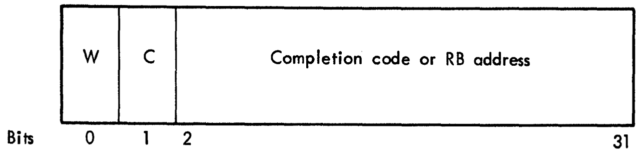
9-11 Address of the parameter list to be passed to the STAE exit routine as specified in the STAE macro instruction.

12 STAE flags

Bit	Indication
0	SCB will not be cancelled by Exit routine when XCTL is issued
1	ISAM/TAM switch
2-7	Reserved

13-15 Address of the Request Block of the task issuing the STAE macro instruction.

EVENT CONTROL BLOCK (ECB)



Description of ECB Fields

W = Wait flag
C = Completion flag

Condition of W and C Flags		Meaning of the Flags and the Contents of Bits 2-31
W	C	
0	0	The event has not been awaited and has not been posted complete. Bits 2-31 may contain meaningless information.
0	1	The event has been posted complete, but it has not yet been awaited. Bits 2-31 contain the completion code in the high-order positions; 0's in the low-order positions.
1	0	The event has been awaited, but has not yet been posted complete. Bits 2-7 are zero, and bits 8-31 contain the address of the RB under which the WAIT macro instruction was issued.
1	1	This combination of conditions cannot normally occur.

Completion Code	Meaning
111111	Normal completion (no errors).
000001	I/O permanent error code.
000010	Extent permanent error code. This code indicates that the seek address specified in the IOB is out of the extent specified in the DEB.
000100	IOB intercept code. Whenever an error occurs after a channel end interruption for a device, the I/O request for that device has already been posted complete and the request element returned to the freelist. To handle the error, the I/O supervisor sets the UCB intercept flag to indicate that the next I/O request for that device must be intercepted. When intercepted, the IOB for the new I/O request and the CSW and sense data for the error are passed to the error recovery procedures for the device. If a permanent error exists, the ECB for the intercepted IOB is posted complete with the IOB intercept code.
001000	Not started or purged. This code signifies either that the I/O request has not been started or that it has been purged.
001111	Error could not be retried. This code signifies that the home address and/or R0 could not be read during error recovery procedures.

PARAMETER LIST ELEMENT (FOR THE ENQ/DEQ ROUTINES)

Bytes				
0	LISTEND	1 LMINOR	2 PARMCDS	3 Return
4	Major Name			
8	Minor Name			

Description of Fields

Field — Meaning (bit is set unless otherwise specified)

LISTEND — Indicates the last element in the parameter list. The last element must have hexadecimal 'FF' in this field. All other elements in the list may have any other value.

LMINOR — The length of the minor name whose address is at offset 8, or zero. If LMINOR contains zero, the length of the minor name is assumed to be in the first byte of the name field whose address is at offset 8. In this case, the length byte does not include its own length.

PARMCDS — ENQ/DEQ parameters:

Bit 0 — Indicates a shared request. If bit is 0, indicates an exclusive request.

Bit 1 — Indicates that the scope of the minor name is SYSTEM. If bit is 0, the minor name is known only to the job step.

Bit 2 — Indicates that "Set Must Complete" = SYSTEM.

Bit 3 — Indicates that "Set Must Complete" = STEP.

Bit 4 — Reserved

Bit 5 — Indicates that RET = TEST (see note below).

Bit 6 — Indicates that RET = TEST/USE (see note below).

Bit 7 — Indicates that RET = TEST/USE/HAVE (see note below).

Note: Bits 5, 6, and 7 must have one of the following configurations:

RET = TEST	1 1 1
RET = USE	0 1 1
RET = HAVE	0 0 1
RET = NONE	0 0 0

RETURN — Return code field for codes returned to the issuer of the ENQ or DEQ macro instruction.

MAJOR NAME — The address of the major resource name (Qname).

MINOR NAME — The address of the minor resource name (Rname).

MAJOR QUEUE CONTROL BLOCK (QCB)

Bytes	
0	Address of next major QCB (if last, equals zero)
4	Address of previous major QCB (if first, equals IEAQQCB)
8	Address of first minor QCB on queue of minors
12	Major QCB name (first four characters)
16	Major QCB name (last four characters)

MINOR QUEUE CONTROL BLOCK (QCB)

Bytes			
0	Address of the first QEL on the QEL queue		
4	Address of the previous minor QCB (If first, equals major QCB)		
8	Address of the next minor QCB (If last, equals zero)		
12	Length of QCB name	QCBPKF 13 (see below)	Minor QCB name (variable in length 14 from 1-255 characters)

Description of Field

QCBPKF: If field is 'FF', the name is known to the entire system. If field is '00', '10', '20', '30', or 'F0', it is the protection key of the TCB under which the request was enqueued. In this case, the name is known only to the job step.

QUEUE ELEMENT (QEL)

Bytes 0	SMC	1 Address of Next QEL Zero if this is last QEL
4	CODE	5 Address of Previous, QEL Address of Minor QCB if This QEL is First on QEL Queue
8	Address of TCB That Was Current When ENQ Macro Instruction Was Issued	
12	Address of SVRB for ENQ Routine	

Description of Fields

Byte 0 Indicates whether or not the QEL represents a request for "must complete" status.

X'20' represents a "system must complete" request
X'10' represents a "step must complete" request
X'00' represents "must complete" status not requested

Bytes 1-3 The address of the next QEL on the queue. In the last QEL the field is zero.

Byte 4 Bit 0: If set to one, indicates a shared request; if set to zero, indicates an exclusive request.
Bit 1: If shared DASD is included in the system: if set to one, indicates that a UCB address appears at Byte 12 of this QEL, and this QEL is associated with a RESERVE macro instruction, rather than an ENQ macro instruction.

Bytes 5-7 The address on the previous QEL on the queue. In the first QEL on the queue, this field points to the minor QCB.

Byte 8 The address of the TCB under which the ENQ macro instruction was issued.

Byte 12 The address of the SVRB under which the ENQ routine is operating. In systems with shared DASD, if the QEL represents a RESERVE request that has been satisfied, this byte contains the address of the UCB of the direct-access device on which the requested resource resides.

INTERRUPTION QUEUE ELEMENT (IQE)

<table>
<tr><td></td><td>Bytes 0</td><td>Reserved</td><td>IQELNK
1</td><td colspan="2">IQEPARAM
4</td></tr>
<tr><td></td><td>8</td><td>Reserved</td><td>IQEIRB
9</td><td>Reserved
12</td><td>IQETCB
13</td></tr>
<tr><td>Rollout/Rollin
Parameter List
(Optional)</td><td>16</td><td colspan="2">RPLTCB</td><td>Reserved
20</td><td>RPLSZPQE
21</td></tr>
</table>

Description of Fields

IQELNK: Address of the next IQE on the IQE queue.

IQEPARAM: The parameter that is to be passed to the asynchronous exit routine.

IQEIRB: Address of the IRB that is to be scheduled because of this request.

IQETCB: Address of the TCB with which this request is associated.

RPLTCB: Address of the TCB for the task requiring or releasing an extension to a region.

RPLSZPQE: Size of region requested (rollout request), or address of PQE describing area (rollin request).

REQUEST QUEUE ELEMENT (RQE)

Bytes 0	RQELNK	2 RQEUCB	F	4 Contains 'FF' if RQE Is on The Free List. Otherwise, Contains Zero.	5 RQEIOB
8	RQEPRI	9 RQEDEB		12 Contains Protection Key of Requestor's Task	13 RQETCB

Description of Fields

RQELNK: Pointer to next RQE on the RQE queue.

RQEUCB: Pointer to UCB.

F: If 1, bit indicates that the RQE represents a request for a system error routine that operates under an SIRB.

RQEIOB: Address of the associated IOB.

RQEPRI: Dispatching priority of requestor's task.

RQEDEB: Address of the associated DEB.

RQETCB: Address of the TCB with which the I/O request is associated.

CONTENTS DIRECTORY ELEMENT (CDE)

Bytes 0	CDATTR	1	CDCHAIN
4	CDROLL	5	CDRBP
8	CDNAME		
12	CDNAME		
16	CDUSE		CDENTPT
20	CDATTR 2		CDXLMJP

Description of Fields

CDATTR: Attribute field.

Bit	Symbolic name	Meaning (when set)
0	NIP	Module was loaded by NIP.
1	NIC	Module is in process of being loaded.
2	REN	Module is reenterable.
3	SER	Module is serially reusable.
4	NFN	Module may not be reused.
5	MIN	This is a minor CDE.
6	JPA	Module is in the job pack area.
7	NLR	Module is not loadable-only.

CDCHAIN: Address of next CDE in queue (either the JPACQ or the LPACQ).

CDROLL: Reserved

CDRBP: RB address. If the module is reenterable, this field contains the address of the last RB that controlled the module. If the module is serially reusable, this field contains the address of the RB at the top of the waiting (RBPGMQ) queue. If the module was requested only through LOAD macro instructions, contains zero.

CDNAME: Either a module name, an alias name, or a name that has been identified via an IDENTIFY macro instruction.

CDUSE: The use/responsibility count. This represents the number of outstanding requests for the module's use.

CDENTPT: Entry point address.

CDATTR2: A second attribute field.

Bit	Symbolic Name	Meaning (when set)
0		Reserved
1	REL	Module is inactive and may be released by the GETMAIN routine (CDPURGE) subroutine).
2	XLE	An extent list has been built for the module.
3	RLC	This CDE contains a minor entry point address that has been relocated by the Program Fetch routine.
4	REFR	Module is refreshable.
5-7		Reserved

CDXLMJP: Extent list address, or major CDE address if this CDE is a minor. (If this CDE is a minor, the MIN bit is also set in its CDATTR field.)

LOAD LIST ELEMENT (LLE)

Bytes	0	1	4	5 6 7
	0	LLCHAIN	LLCOUNT	LLCDPTR

Description of Fields

LLCHAIN: Address of the first byte of the next element on the load list.

LLCOUNT: Responsibility count. The number of requests for the module, via the LOAD macro instruction.

LLCDPTR: The address of the CDE for the module.

PARTITIONED DATA SET DIRECTORY ENTRY

Bytes			
0	Name of load module (Member or alias name)		
8	Relative (to beginning of data set) disk address of module (TTR)		11 Alias indicator and miscellaneous information
12	Relative (to beginning of data set) disk address of first text record (TTR)		15 Byte of binary zeroes
16	Relative (to beginning of data set) disk address of NOTE list or Scatter/ translation record (TTR)		19 NOTE List * Number of entries in
20	Module Attributes (see description of attributes) 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, R, R	22-24 Total contiguous main storage required for the module.	
24		25 Length (in bytes) of first text record	27 Module's linkage
28	Editor assigned entry point address	30 - 32 Linkage editor assigned origin of first text record	
32			

For load modules in scatter format, add:

Bytes			
		33 Length of scatter list (in bytes)	35 - 36 Length of translation table (in bytes)
36		37 ESDID (CESD entry number of control section name) for first text record	39 - 40 ESDID (CESD entry number of control section name) containing entry point
40		41	

For load modules with RENT or REUS attribute and Alias names add:

Bytes		
		41 Entry point address of the member name
44	Member name	
52	SSI Bytes - Aligned on a half-word boundary at the end of the PDS record	

Description of Fields

Alias indicator and miscellaneous information:

Bit Number	Meaning
0	Alias indicator: 0 signifies none, 1 signifies alias.
1-2	Number of relative disk addresses in user data field.
3-7	Length of user data field in halfwords

PDS Directory Record Size:

Block format	34 bytes (when rounded to a halfword boundary)
Block format with alias names	44 bytes
Scatter format	42 bytes
Scatter format with alias names	52 bytes

Note: For SSI, add 4 bytes to sizes given above.

Module Attributes:

Bit number	Attribute	Bit setting	Indication
0	RENT	0	Not reenterable
		1	Reenterable
1	REUS	0	Not reusable
		1	Reusable
2	OVLY	0	Not an overlay module
		1	Overlay module
3	TEST	0	Not under test
		1	Under test
4	LOAD	0	Not loadable only
		1	Loadable only[1]
5	Format	0	Block format
		1	Scatter format
6	Executable	0	Not executable
		1	Executable
7	Format	0	Module contains more than one text record and/or RLD record(s).
		1	Module contains only one text record and no RLD record.
8	Compatibility	0	Module can be processed by all levels of linkage editor.
		1	Module cannot be reprocessed by Linkage Editor-E.
9	Format	0	Linkage editor assigned origin of first text record is not zero.
		1	Linkage editor assigned origin of first text record is zero.

[1] Module can be loaded only with the LOAD macro instruction. When the module is in main storage, it will be entered directly and not through the use of an XCTL, LINK or ATTACH macro instruction.

10	Format	0	Linkage editor assigned entry point is not zero.
		1	Linkage editor assigned entry point is zero.
11	Format	0	Module contains RLD record(s)
		1	Module does not contain an RLD record.
12	Editability	0	Module can be reprocessed by linkage editor.
		1	Module cannot be reprocessed by linkage editor.
13	Format	0	Module does not contain TESTRAN symbol records.
		1	Module contains TESTRAN symbol records.
14	Reserved		
15	Refreshability	0	Module is not refreshable.
		1	Module is refreshable.

SCATTER EXTENT LIST

Bytes	1 byte	3 bytes
0	EXLLNTH (Total size of extent list)	
4	Number of relocation factors	
8	Length of first non-contiguous block	
12	Length of second non-contiguous block	
16	Length of third non-contiguous block	
	Hex. 80*	Length of last non-contiguous block
	0	Address of first non-contiguous block
	0	Address of second non-contiguous block
	0	Address of third non-contiguous block
		• • •
	0	Address of last non-contiguous block

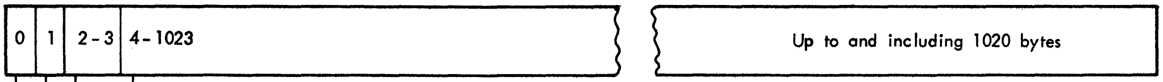
* Indicates the end of the immediately preceding length-of-block list. Used by the GETMAIN routine.

BLOCK EXTENT LIST AND NOTE LIST

Bytes			
0	EXLLNTH Total Size of Block Extent List		Block Extent List
4	Number of Relocation Factors		
8	Hex. 80	9 Length of Main Storage Block	
12	Zero	13 Address of Main Storage Block	
16	Zero	17 Relocation Factor	Note List (overlay modules only)
20		23 Concatenation Number *	
24	Relative Disk Address (TTR) of First Segment of Module	27 Zero	
28	Relative Disk Address (TTR) of Second Segment of Module	31 Zero	
32	Relative Disk Address (TTR) of Third Segment of Module	35 Zero	
	• • •	• • •	
	Relative Disk Address (TTR) of Last Segment of Module	Zero	

* Contatenation number is a value that specifies this data set's sequential position in a group of concatenated data sets.

SCATTER/TRANSLATION RECORD



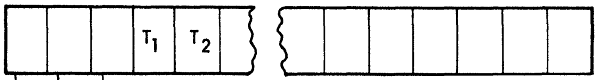
Data - may contain translation table, translation table and scatter table or scatter table only

Count - in bytes, of data field

Zero - one byte of binary zeros

Identification - identifies this as a scatter-translation record - bit configuration is: 0001 0000

Translation Table



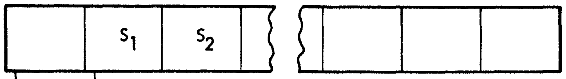
Padding (2 bytes) - if necessary, to force full-word boundary alignment of scatter table.

Pointer (2 bytes) - to the scatter table entry that contains the address of the control section containing this CESD entry.
Number of translation table entries = number of CESD entries + 1.
Pointer will be zero if its corresponding CESD entry is not SD, PC, CM or LR.

Zero - 2 bytes of binary zeros

NOTE: (One 2-byte entry for each external symbol)

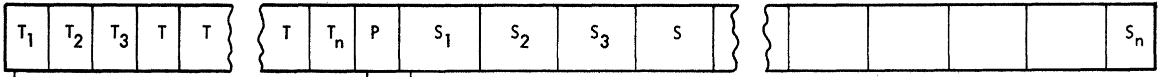
Scatter Table



Assigned address (4 bytes) - of a control section (SD, PC or CM) (one entry for each CSECT)

Zero - 4 bytes of binary zeros

Translation Table and Scatter Table



Scatter data

Padding (2 bytes) if necessary to align scatter table to a full-word boundary.

Translation data

NOTE: Translation table follows extent list in main storage.
Translation table entries are two bytes in length, scatter table entries four bytes in length.

Legend for Types of Entries in Composite External Symbol Dictionary (CESD)

SD = section definition
LR = label reference
PC = private code
CM = common

PROGRAM FETCH WORK AREA -- (DISPLACEMENTS IN BYTES)

Displacement	Definition	Length
0	IOB	8 full words
32	IOB Seek Address	2 full words
40	Seek Buffers (4)	12 full words
88	Search and TIC CCWs	3 double words
112	RLD Buffer 1	33 double words
376	Channel Program 1	5 double words
416	RLD Buffer 2	33 double words
680	Channel Program 2	5 double words
720	RLD Buffer 3	33 double words
984	Channel Program 3	5 double words
1024	I/O ECB	1 full word
1028	ECB	1 full word
1032	Buffer Table Pointer	2 full words
1040	Buffer Table	9 full words
1076	Register Save Area	16 full words
1140	Addr of Translation Table	1 full word
1144	Addr of Scatter List	1 full word
1148	Addr of R-Pointer	1 full word
1152	Addr of P-Pointer	1 full word
1156	Boundary Word for Relocation	1 full word
1160	Fetch Flags	2 full words
1168	ECB List	2 full words
1176	Last Table Entry	1 full word

Description of Fetch Flags

Byte	Content	Meaning
0		Reserved
1	FF	Program is being scatter-loaded.
	00	Program is being block-loaded.
2	FF	All buffers are full.
	0F	Channel-End Appendage routine is unable to restart a channel program because all buffers were full when the channel-end interruption occurred.

	00	Normal condition. There is at least one empty buffer.
3	FF	End condition. Only termination processing by the Program Fetch routine is needed.
	0F	End condition. Buffer processing is needed.
4-7	0	A read operation was just completed. A text record, followed by an RLD or control record, was read. The restart buffer is the last one to be filled.
	Address	A read operation was just completed. An RLD or control record was read. The contents of the restart-seek address buffer is saved to be used when channel-program restart is needed.

PROGRAM FETCH BUFFER TABLE

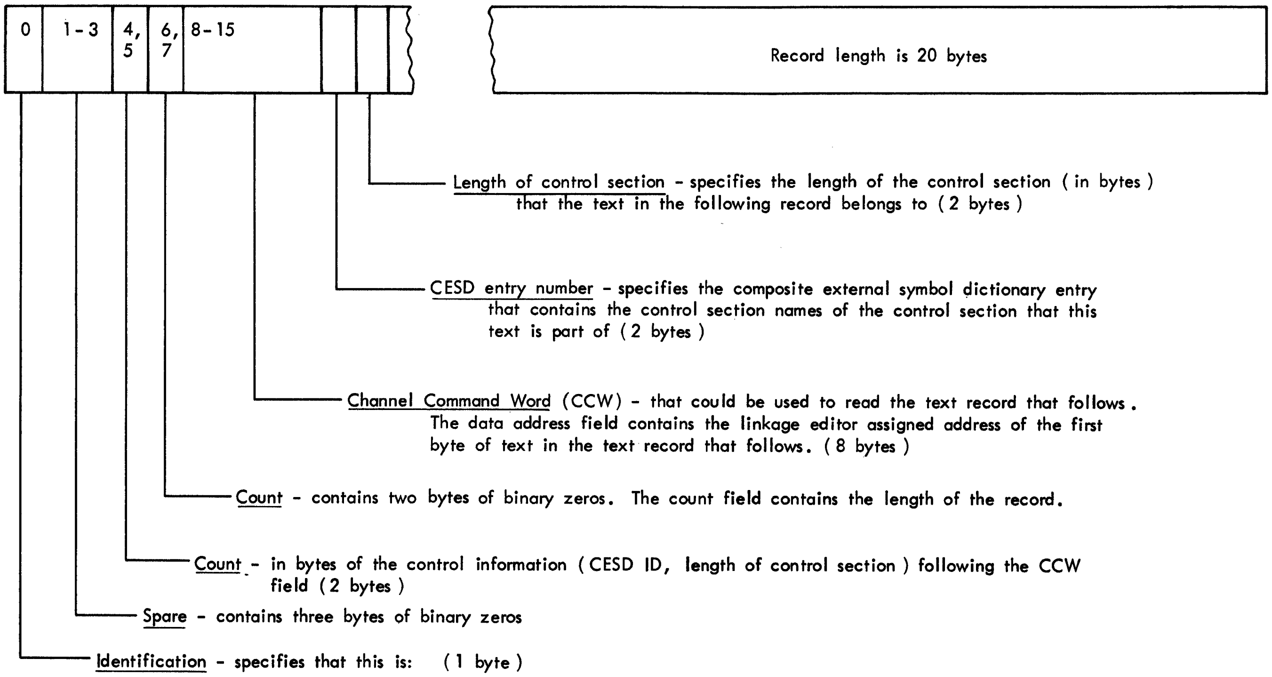
Bytes						
0	Buffer Code	Pointer to Next Entry (12) 1	TIC Command 4	Address of Channel Program 2 5	Zero 8	Address of Buffer 1 9
12	Buffer Code	Pointer to Next Entry (24) 13	TIC Command 16	Address of Channel Program 3 17	Zero 20	Address of Buffer 2 21
24	Buffer Code	Pointer to First Entry (0) 25	TIC Command 28	Address of Channel Program 1 29	Zero 32	Address of Buffer 3 33

Note : Each entry consists of 12 bytes

Description of Buffer Codes

Content	Meaning
00	Buffer Empty
80	Buffer Full

CONTROL RECORD



Length of control section - specifies the length of the control section (in bytes) that the text in the following record belongs to (2 bytes)

CESD entry number - specifies the composite external symbol dictionary entry that contains the control section names of the control section that this text is part of (2 bytes)

Channel Command Word (CCW) - that could be used to read the text record that follows. The data address field contains the linkage editor assigned address of the first byte of text in the text record that follows. (8 bytes)

Count - contains two bytes of binary zeros. The count field contains the length of the record.

Count - in bytes of the control information (CESD ID, length of control section) following the CCW field (2 bytes)

Spare - contains three bytes of binary zeros

Identification - specifies that this is: (1 byte)

- A control record - 0000 0001
- The control record that precedes,the last text record of this overlay segment - 0000 0101
- The control record that precedes the last text record of the module - 0000 1101

RELOCATION DICTIONARY (RLD) RECORD

0	1 - 3	4, 5	6, 7	8-15	16-255

Record length can be between 24 and 256 bytes

RLD data -- see below

Spare - contains 8 bytes of binary zeroes

Count - in bytes of the relocation dictionary information following the spare 8 byte field (2 bytes)

Count - contains two bytes of binary zeroes

Spare - contains three bytes of binary zeroes

Identification - specifies that this is: (1 byte)
A relocation dictionary record - 0000 0010
The last record of the segment - 0000 0110
The last record of the module - 0000 1110

RLD Data

R	P	F	A	F	A

F	A	R	P	F	A	R	P	F	A

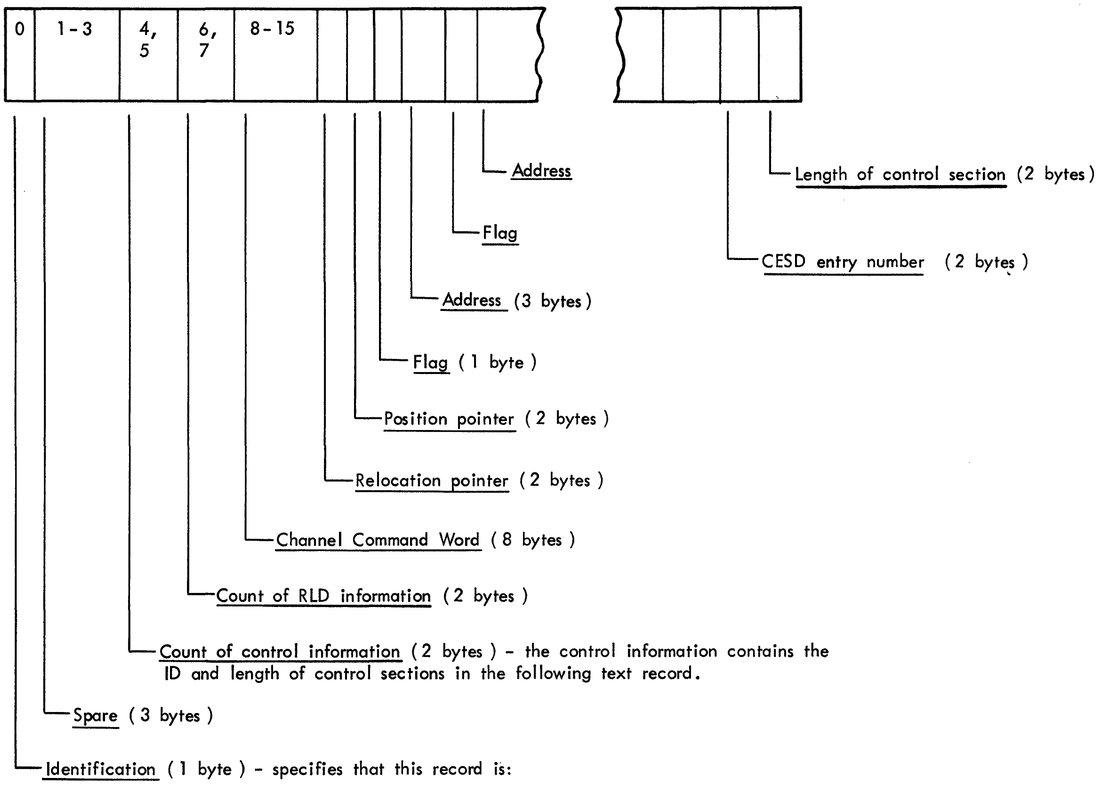
Address - Linkage editor assigned address of the address constant (3 bytes)

Flag - specifies miscellaneous information as follows: (1 byte) when byte format is xxxxLIST:
xxxx specifies the type of this RLD item (address constant)
0000 -- non-branch type in assembler language, a DC A (name)
0001 -- branch type (in assembler language, a DC V (name)
0010 -- pseudo register displacement value
0011 -- pseudo register cumulative displacement value
1000 and 1001 -- this address constant is not to be relocated, because it refers to an unresolved symbol.
LL specifies the length of the address constant
01 -- two byte
10 -- three byte
11 -- four byte
S specifies the direction of relocation
0 -- position
1 -- negative
T specifies the type of RLD item following this one
0 -- the following RLD item has a different relocation and/or position pointer
1 -- the following RLD item has the same relocation and position pointers as this one, and therefore is omitted

Position pointer - contains the entry number of the CESD entry (or translation table entry) that indicates which control section the address constant is in (2 bytes)

Relocation pointer - contains the entry number of the CESD entry (or translation table entry) that indicates which symbol's value is to be used in the computation of the address constant's value (2 bytes)

CONTROL AND RELOCATION DICTIONARY RECORD



Address

Flag

Address (3 bytes)

Flag (1 byte)

Position pointer (2 bytes)

Relocation pointer (2 bytes)

Channel Command Word (8 bytes)

Count of RLD information (2 bytes)

Count of control information (2 bytes) - the control information contains the ID and length of control sections in the following text record.

Spare (3 bytes)

Length of control section (2 bytes)

CESD entry number (2 bytes)

Identification (1 byte) - specifies that this record is:

- A control and RLD record - 0000 0011
- A control and RLD record that is followed by the last text record of a segment - 0000 0111
- A control and RLD record that is followed by the last text•record of a module - 0000 1111

Note: For detailed descriptions of the data fields see:

Relocation Dictionary Record
Control Record

The record length will vary from 20 to 260 bytes.

SEGMENT TABLE

Bytes 0	TEST ind.	Bit 1 = 0: Not in Test Bit 1 = 1: In Test	1 Address of data control block (DCB) used to load module *		
4	0		Address of note list *		
8	Last segment number of region 1		Highest segment no. in storage-region 1 9	Last segment number of region 2 10	Highest segment no. in storage-region 2 11
12	Last segment number of region 3		Highest segment no. in storage-region 3 13	Last segment number of region 4 14	Highest segment no. in storage-region 4 15
16	Address of ECB to be posted when SEGLD request has been serviced *				
20	Reserved *				
24	Previous segment number for segment 1 *		25		Status Indctr
28	Previous segment number for segment 2		29 Address of entry table entry (when caller chain exists) *		Status Indictr
	Previous segment number for segment N		Address of entry table entry (when caller chain exists) *		Status Indctr

4 bytes

Description of Fields

TEST indicator
specifies that this module is "under test" using TESTRAN. Is initialized by Program Fetch routine.

Highest segment number in storage
is initially set to 00 except for region 1 which is initially set to 01 by linkage editor.

Status indicator
indicates the status of this segment, with the two last bits of the entry table address field as follows:

00 -- segment is in main storage as a result of a branch to the segment.
10 -- segment is in main storage, no caller chain exists.
01 -- segment is not in main storage, but is scheduled to be loaded.
11 -- segment is not in main storage.

The status indicator for segment 1 is initially set to 10. All the rest are initially set to 11.

*Set to zero by linkage editor.

Note: "Region" refers to the regions of a multiregion overlay structure, not to a job-steps's region of main storage (see Linkage Editor SRL).

ENTRY TABLE

<table>
<tr><td></td><td colspan="2">Unconditional branch to ·ast entry-
BC 15, DISP(15,0)
0</td><td colspan="2">Address of referred to symbol
4</td><td>"to" seg number
8</td><td>Previous Caller (zero initially)
9</td></tr>
<tr><td></td><td colspan="2">Unconditional branch to last entry-
BC 15, DISP(15,0)
12</td><td colspan="2">Address of referred to symbol
16</td><td>"to" seg number
20</td><td>Previous Caller (zero initially)
21 22 23</td></tr>
<tr><td></td><td colspan="2">|</td><td colspan="2">|</td><td>|</td><td>|</td></tr>
<tr><td></td><td colspan="2">Unconditional branch to last entry-
BC 15, DIPS(15,0)</td><td colspan="2">Address of referred to symbol</td><td>"to" seg number</td><td>Previous Caller (zero initially)</td></tr>
<tr><td>Last Entry</td><td>SVC 45 instruction</td><td colspan="2">L 15,4(0,15) Loads GR15 with the value of the ADCON.</td><td>BCR 15, 15</td><td>"from" seg no.</td><td>Address of segment table (SEGTAB)</td></tr>
<tr><td></td><td>2 bytes</td><td>2 bytes</td><td>2 bytes</td><td>2 bytes</td><td>1 byte</td><td>3 bytes</td></tr>
</table>

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.

SUBPOOL QUEUE ELEMENT (SPQE)

	Bits				
Bytes 0	0	1	2	Reserved	1 SPQEPTR
4	SPID				5 DQEPTR

Description of Fields

Byte 0

Bit 0: When bit is zero, indicates that subpool belongs to the associated task. When bit is one, indicates that the subpool is shared.

Bit 1: Is usually zero. When bit is one, indicates that this element is the last SPQE on the queue.

Bit 2: When bit is one, indicates that the subpool is shared with another task.

SPQEPTR: Pointer to the next SPQE. When field is zero, indicates that this element is the last SPQE on the queue.

SPID: Identifying number of the subpool.

DQEPTR: Pointer to the first DQE for the subpool. If subpool is shared, field points to the "owning" SPQE.

• DESCRIPTOR QUEUE ELEMENT (DQE)

Bytes		
0	Reserved	1 FQEPTR
4	Reserved	5 DQEPTR
8	DQEHRID	9 Block Address
12	Reserved	13 Length

Description of Fields

FQEPTR: Pointer to first free area.

DQEPTR: Pointer to next DQE. Is zero in last DQE.

DQEHRID:

Bit	Meaning
0-6	zero
7	when 0 indicates that the DQE describes core obtained from hierarchy 0; when 1 indicates that the DQE describes core obtained from hierarchy 1.

Block address: Address of the first 2K block described by this DQE.

Length: Length in bytes described by this DQE. (Length is always a multiple of 2K bytes.)

FREE QUEUE ELEMENT (FQE)

Bytes 0	Reserved	1 FQEPTR
4	Reserved	5 LENGTH

Description of Fields

FQEPTR: Pointer to next lower free area.

LENGTH: Number of bytes in free area.

ALLOCATED QUEUE ELEMENT (AQE)

Bytes 0	Reserved	1 AQEPTR
4	Reserved	5 LENGTH

Description of Fields

AQEPTR: Pointer to next allocated area.

LENGTH: Number of bytes in allocated area.

- GOVRFLB (Origin list for Main Storage Queues)

Bytes		
0	Reserved	SQBOUND
4	Reserved	DQESQES
8	Reserved	PQEPTR
12	Reserved	SZDPRS
16	Reserved	SZDLCS
20	Reserved	VQEPTR

Description of Fields

SQBOUND: Address of the first byte beyond supervisor queue area.

DQESQES: Address of the DQE describing supervisor queue area.

PQEPTR: Address of the PQE describing unassigned main storage (storage not assigned to any region).

SZDPRS: Amount of storage available in hierarchy 0 after NIP.

SZDLCS: Amount of storage available in hierarchy 1 after NIP.

VQEPTR: Address of the first VQE describing storage areas scheduled for removal in a multiprocessing system. Zero if no VQEs exist.

• PARTITION QUEUE ELEMENT (PQE)

0	PQEFFBQE			
4	PQEBFBQE			
8	PQEFPQE			
12	PQEBPQE			
16	PQETCB			
20	PQESIZE			
24	PQEREGN			
28	PQERFLGS	29 PQEHRID	30 Reserved	31 Reserved

Description of Fields

PQEFFBQE: Address of the first FBQE in the region described by this PQE. If there are no FBQEs, contains the PQE address.

PQEBFBQE: Address of the last FBQE in the region described by this PQE. If there are no FBQEs, contains the PQE address.

PQEFPQE: Address of the next PQE on the queue. Contains 0's in the last PQE.

PQEBPQE: Address of the preceding PQE on the queue. Contains 0's in the first PQE.

PQETCB: Address of the TCB for the job step to which the space belongs. Contains 0's if the space was obtained from unassigned free space.

PQESIZE: Size of the region described by this PQE. (Always a multiple of 2048)

PQEREGN: Address of the first byte of the region described by this PQE.

Description of Rollout Flags (PQERFLGS)

Bit	Meaning
0	When 0, indicates space described by this PQE is owned. When 1, indicates space is borrowed.
1	When 1, indicates region has been rolled out. Meaningful only if bit 0=0.
2	When 1, indicates region has been borrowed. Meaningful only if bit 0=0.
3-7	Reserved.

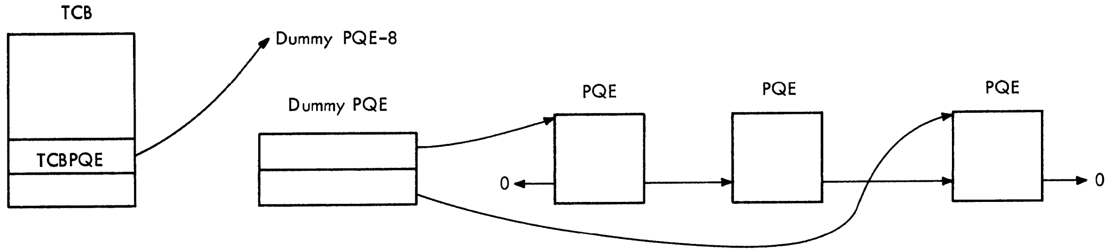
Description of Hierarchy Identifier (PQEHRID)

Bit	Meaning
0-6	zero
7	when 0 indicates that the PQE describes a region in hierarchy 0. when 1 indicates that the PQE describes a region in hierarchy 1.

DUMMY PARTITION QUEUE ELEMENT (DPQE)

Address of First PQE in Chain	Address of Last PQE in Chain
0	4

• Relationship of Dummy PQE to TCB and PQE Chain



FREE BLOCK QUEUE ELEMENT (FBQE)

Bytes		
0	Reserved	1 FWDPTR
4	Reserved	5 BCKPTR
8	Reserved	9 SIZE

Description of Fields

FWDPTR: Pointer to the next higher address FBQE in the region. In the highest address FBQE, contains the address of the PQE.

BCKPTR: Pointer to the next lower FBQE in the region. In the lowest FBQE, contains the address of the PQE.

SIZE: Number of bytes in the set of 2K blocks.

ROLLOUT I/O QUEUE ELEMENT (RIQE)

0	4	8	12
Address of Next RIQE	Address of Rolled-Out Job Step's TCB	Address of I/O-Purged TCB	Beginning Address of IOB Chain

REPLY QUEUE ELEMENT

0	RQERQE		
4	RQEID	6	RQEXA
8	RQETCB		
12	RQEXB		
16	RQELNTH	17	RQERPTR
20	RQEECB		

Description of Fields

RQERQE: Address of next reply queue element.

RQEID: Reply identification number.

RQEXA: Flags, with the following meanings when on:

Bit 6.0: Associated reply will be purged.

Bit 6.1: Associated task has been rolled out.

RQETCB: Address of TCB for task that issued message for which this RPQE represents a reply.

RQEXB: Address of purging message buffer, or temporary buffer if reply was deferred by rollout.

RQELNTH: Maximum length of reply.

RQERPTR Address of user's buffer.

RQEECB Address of user's ECB.

SVC PURGE PARAMETER LIST

PURGPARM

0	PURGOPT	1 PURGDEB
4		5 PURGTCB/PURGECB
8		9 PURGIOB

Description of Fields

Offset

0 PURGOPT: Purge options (always X'02' for rollout, requesting 'TCB' and 'quiesce' options).

1 PURGDEB: Address of DEB (not used for rollout).

4 Completion code to be placed in ECB.

5 PURGTCB: During input: Address of TCB whose request elements are to be purged.

PURGECB: During output: Address of ECB to be posted when purge is complete.

8 Count field for quiesce option. Number of request elements whose I/O operations have not yet completed.

9 PURGIOB: Address of an IOB chain field. The IOBs queued from the chain field represent channel programs to be restarted by the SVC Restore routine after rollin has occurred. These IOBs belong to the task whose TCB address is recorded in the PURGTCB field.

TIMER QUEUE ELEMENT (TQE)

0	TQEFLGS (Indicators)	1	TQETCB (Address of TCB)
4	Zeros	5	TQEFLNK (Address of next queue element)
8	Zeros	9	TQEBLNK (Address of preceding queue element)
12	TQEVAL (Time of expiration/time remaining)		
16	TQELHPSW (First word of current PSW - used when TQE serves as IRB)		
20	TQESAV (Used to save contents of TQEVAL when TQE is converted from TASK to REAL)		
24	TQESAADR (Address of processing program save area)		
28	Zeros	29	TQEEXIT (Address of timer asynchronous exit routine)
32	TQEGRS (Register save area - used when TQE serves as IRB)		
	TQEECB		(Used for interruption queue element when TQE serves as IRB) (16 bytes)
96	TQEIQE		(Used for ECB when WAIT parameter is given in STIMER macro-instruction.

Description of Fields

TQEFLGS:

Bit	Meaning (when bit is set)
0	Timer element is not on timer queue.
1	Local TOD option used.
2-3	00 = TUINTVL requested 01 = BINTVL requested 10 = reserved 11 = DECINTVL requested
4	Interval is complete.
5	Exit specified*
6-7	00 = task request 01 = wait request 10 = supervisory element* 11 = real request
*5-7	110 = Denotes the midnight supervisory timer element

TQETCB: Address of the TCB for the task for which this timer element is being used.

TQEFLNK: Forward link field. This field contains the address of the first byte of the next TQE on the timer queue.

TQEBLNK: Backward link field. This field contains the address of the first byte of the previous TQE on the timer queue.

TQELHPSW: Left half PSW. This field contains the left half PSW to be used when the TQE serves as an IRB. The left half PSW determines the machine states (enabled, supervisor, etc.) when a timer exit routine is entered.

TQEVAL: Interval value. This field contains the timer interval. If the element is on the timer queue, the contents of this field represent the time of expiration (TOX) of the interval relative to the 6-hour interval. If the element is off the timer queue, the contents of this field represent the remaining time in the interval.

The value of the interval in microseconds can be calculated by multiplying by 26 the decimal value represented by the field.

TQEEXIT: Exit routine address. This field contains the address of the timer exit routine, if one was specified by the user in the calling sequence of the STIMER macro instruction. Otherwise this field is zero.

TQEGRS: General register save area. This field becomes the general register save area when the TQE is used as an IRB for the scheduling of a timer exit routine.

TQESAADR: Address of the problem program register save area.

TQEIQE: Interruption queue element (begins at offset 96). This field is used as the IQE passed to the Stage 2 Exit Effector in order to schedule a timer exit routine.

TQEECB: Event control block (begins at offset 96). ECB to be posted for a "wait" type interval.

A TQE is required in systems with the time-slicing feature. It is used by the Dispatcher to set the time interval to the specified time-slice length when a time-sliced task is dispatched. The first 16 bytes of the TQE are used in this application.

SECONDARY COMMUNICATIONS VECTOR TABLE

This table appears in module IEAQED00, beginning at symbolic location IEABEND. It consists of a list of address constants that point to routine entry points or system control blocks. The address constants that appear are:

Displacement in Bytes	Symbolic Name	Meaning
0	IEAQPGTM	Address of EOT Purge Timer routine.
4	IEECVPRG	Address of WTOR Purge routine.
8	IEAQSPET	Address of Release Main Storage routine.
12	IEAQTAQ	Address of TACT.
16	IEAQERA	Address of EOT Erase Phase routine.
20	IEAQQCBO	Address of QCB origin.
24	IEA0EQ01	Address of ENQ/DEQ Purge routine.
28	RMBRANCH	Address of REGMAIN branch entry.
32	IGC016	Address of SVC Purge routine.
36	IECXTRA	Address of Trace routine switch.
40	IEA0DS02	Address of Task Switching routine.
44	IEAQCS02	Address of CDCONTRL in common subroutines of Contents Supervision.
48	FMBRANCH	Branch entry point to the FREEMAIN routine.
52	IEAQABL	Address of Release Loaded Programs routine in EOT.
56	IEADQTCB	Address of Dequeue TCB routine in EOT.
60	CDHKEEP	Address of CDHKEEP in the CDEXIT routine.
64	TRPTR	Address of trace table pointers.
68	GMBRANCH	List Format GETMAIN branch entry point.
72	TAUSERCT	Transient area user count.
76	IEARCTRS	Address of rollout counters.
80	IEAROQUE	Address of rollout queue.
84	IEAROIRB	Address of rollout IRB.
88	IEAROTCB	Address of rollout TCB.

ABDUMP PARAMETER LIST

Bytes					
0	ID	1	0	2	Option Flags
4	0	5	Pointer to DCB		
8	0	9	Pointer to TCB		
12	0	13	Pointer to Storage List		

Description of Option Flags

Byte	Bit	Symbolic Name	Meaning (when bit is set)
2	0	PFABEND	0 = Abend request; 1 = SNAP request.
	1	PFTCB	TCB address is given.
	2	PFSUPDAT	Display all supervisor data.
	3	PFTRACE	Display trace table (if possible).
	4	PFNUC	Display the nucleus.
	5	PFSNAP	Snapshot list is given.
	6	PFID	ID given.
	7	PFQCB	Display the QCBs.
3	0	PFSAVE	Save area (see next flag).
	1	PFSAVE2	0 = display entire save area; 1 = display headings only.
	2	PFREGS	Display registers on entry to Abend or SNAP.
	3	PFLPA	Display link pack area.
	4	PFJPA	Display job pack area.
	5	PFPSW	Display PSW on entry to Abend or SNAP.
	6	PFSPALL	Display all subpools less than subpool 128.
	7		Reserved.

TIME-SLICE CONTROL ELEMENT (TSCE)

0	Dispatching Priority	1	Address of First TCB
4	0	5	Address of Last TCB
8	0	9	Address of Next TCB to be Dispatched
12	TSCE Flags	13	Length of Time-Slice

There is one TSCE for each priority that is time-sliced. The address of the first TSCE is in the CVTTSCE field of the CVT. All TSCEs are contiguous.

Description of Fields

Byte **Contains**

0 Dispatching priority of time-slice group.

1-3 Address of first TCB on the TCB queue that is a member of the time-slice group.

4 Zero

5-7 Address of the last TCB on the TCB queue that is a member of the time-slice group.

8 Zero

9-11 Address of the next TCB to be dispatched when the priority obtains control.

12 TSCE Flags

Bit	Means (when set to one)
0	Last TSCE
1-7	Reserved

13-15 Length of time-slice. In milliseconds before NIP; in timer units after (26 micro-seconds per timer unit).

• DISPLAY CONTROL MODULE (DCM)

Display Control Module (DCM)

	8 Bytes							
0	DCMAPA							
8	DCMHCP							
16	DCMEXTN				20 DCMUCM22			
24	DCMUCMHC				28 DCMWTBUF			
32	DCMTCB1				36 DCMTCB2			
40	DCMTCB3				44 DCMTCB4			
48	Zeros	49 DCMACT	50 DCMDSRC	51 DCMDSTA	52 DCMAP	53 DCMCS	54 DCMIOC1	55 DCMIOC2
56	DCMIOC3	57 DCMIOC4	58 Data computed by the Display Routine and passed to the I/O Delete Routine.(10 bytes)					
					68 DCMSNSTS			
	Buffer addresses for points in the Basic Display Order program. (68 bytes)							
72					140 Not Used			
144	DCMENTLG		146 Not Used		148			
	DCMENTR2 (148 bytes)							
	DCMNULL (148 bytes)							
296					444			
	Basic Display Order Program.(approximately 3616 bytes)							
4060	Work area for CCW chains. (80 bytes)							
4140	Fields used by Console Device Support Routines to control displays of options and error messages. (316 bytes)							

Description of Fields

DCMAPA
contains the name of the routine to which control is to be passed when console switching has occurred.

DCMHCP
contains the name of the Hard Copy Processor routine.

DCMGXTN
contains a pointer to the DCM extension which starts with the work area for CCW chains.

DCMUCM22
contains a pointer to the 2250 UCM entry.

DCMUCMHC
contains a pointer to the hard copy UCM entry.

DCMWTBUF
contains a pointer to the message to be displayed.

DCMTCB1, 2, 3, 4
contains the pointers to the task control blocks associated with WTOR 1, 2, 3, and 4.

DCMACT

Bit	Indication (when bit is set)
1	Light pen option
2	WTOR1 delete option
3	WTOR2 delete option
4	WTOR3 delete option
5	WTOR4 delete option

DCMDSRC

Bit	Indication (when bit is set)
0	Full buffer
1	Permanent Error
2	Input
3	Do not print hard copy
4	Retry bit for asynchronous error routine

DCMDSTA

Bit	Indication (when bit is set)
0	Hold message
1	Unit Status displayed
2	Format Options displayed
3	Command Format displayed

DCMAP

Bit	Indication (when bit is set)
0	Stop display regeneration
1	WTO Entry
2	WTOR Entry

DCMCS

Bit	Indication (when bit is set)
0	Closed request
1	Re-open condition

IOC1

Bit	Indication (when bit is set)
0	Write WTO
1	Write WTOR
4	WTO delete

5	WTOR delete
7	Stop display regeneration

IOC2

Bit	Indication (when bit is set)
0	Message Print 'YES'
1	Message Print 'NO'
2	Message Hold 'YES'
3	Message Hold 'NO'
4	Keyboard Attention
6	WTO Heading delete
7	WTOR Heading delete

IOC3

Bit	Indication (when bit is set)
0	WTO Warning is displayed
1	WTOR Warning is displayed
2	WTO Warning should be displayed
3	WTOR Warning should be displayed
4	CCW chain
5	Sound audible alarm

IOC4

Bit	Indication (when bit is set)
0	Option 1 has been selected
1	Option 2 has been selected

DCMSNSTS
contains the contents of Register 15 when a wait state is entered due to a permanent input/output or asynchronous error.

DCMASYN
contains the buffer address at which an asynchronous error occurred.

DCMGD1
contains the X,Y coordinates for MESSAGE PRINT NO.

DCMGD2
contains the X,Y coordinates for MESSAGE HOLD YES.

DCMENTLG
contains the length of the incoming keyboard message.

DCMENTR2
contains the incoming message as passed to the Log and WRITELOG Post routine (SVC 34) and the Write-to-Operator routine (SVC 35).

DCMNULL
contains an area used to blank the screen.

• MULTIPROCESSING COMMUNICATIONS VECTOR TABLE (MPCVT)

The Multiprocessing Communications Vector Table is part of the resident nucleus and begins at symbolic location IEAMPCVT. The address of the first location of the MPCVT is contained in the CVTMPCVT entry of the CVT and also in the MPCVTPTR field of the Prefixed Storage Area. The entries in the MPCVT are:

Hex	Dec	4 bytes		
0	0	CVTAFFLK CPU Identity C1 = CPU A C2 = CPU B 00 = Neither	Supervisor Lock FF = Set 00 = not Set	Reserved
4	4	CVTSTPTR addr of SHOLDTAP routine		
8	8	CVTWTTCB address of Dispatcher Wait Task		
C	12	CVTTKRM addr of task removal (TESTDSP) routine		
10	16	CVTGOV addr of GOVRFLB table		
14	20	CVTIOTIO address of Multiprocessing Unit TIO routine in IOS		
18	24	CVTIOTCH address of Multiprocessing Channel TCH routine in IOS		

• VARY QUEUE ELEMENT (VQE)

The VQE describes the main storage area to be logically removed from a multiprocessing system due to a VARY STORAGE offline command. The address of the Vary Queue is located in the GOVRFLB table.

0	0	1	Address of next VQE on Vary Queue
4	0	5	Lower address of area specified in VARY command
8	0	9	Length of area specified in VARY command
12	0	13	ECB - posted by FREEPART

FAIL SOFT STORAGE ELEMENT MAP (FSSEMAP)

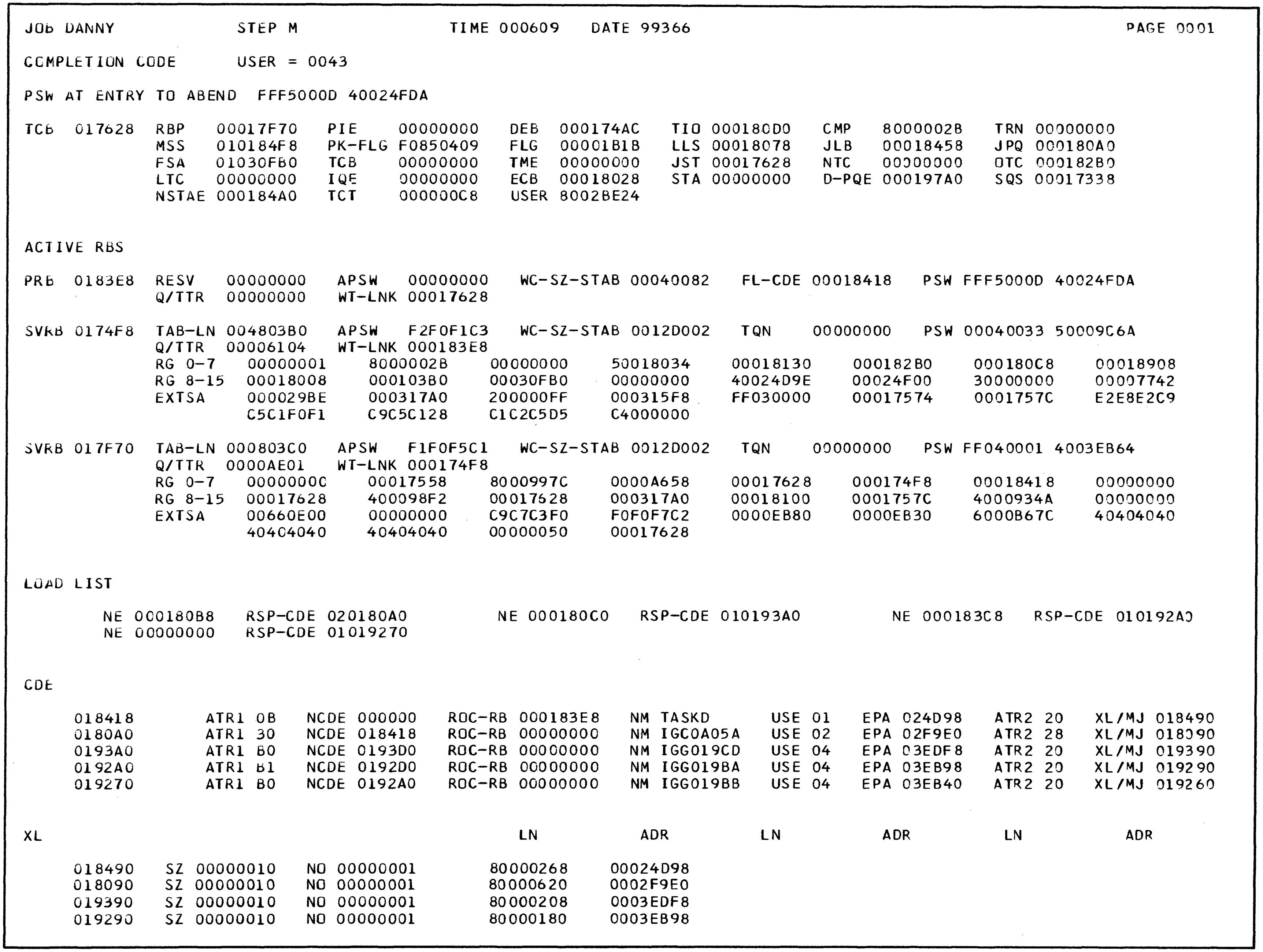
The FSSEMAP is a 128-byte (1024 bits) field located at hex location 300 in a multiprocessing system. Each 2K block of main storage is described by two bits which can have the following values:

Setting	Indication
00	Normal-described by an FBQE or PQE
10	Reserved
01	Reserved
11	Logically removed from the system - not described by an FBQE or PQE

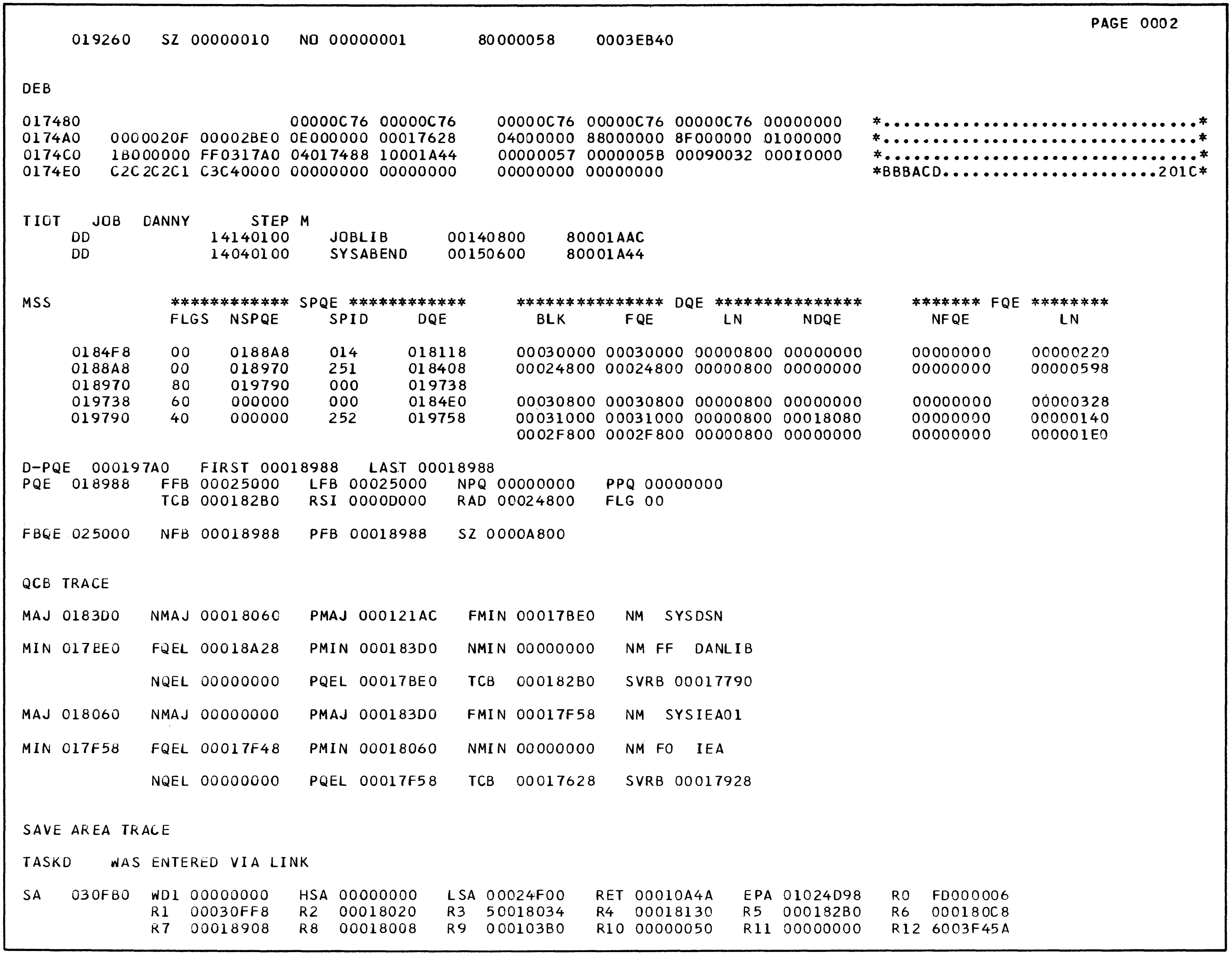
Given a main storage address (X), the corresponding 2K block (b) is:

$$b = \frac{X}{2048} \quad \text{(disregard remainder).}$$

The number (n) of the first of the two bits which describe the 2K block is: n = 2*b.



```
JOb DANNY             STEP M                   TIME 000609   DATE 99366                               PAGE 0001

CCMPLETIUN CODE       USER = 0043

PSW AT ENTRY TO ABEND  FFF5000D 40024FDA

TCb  017628  RBP    00017F70   PIE      00000000   DEB   000174AC   TIO 000180D0   CMP    8000002B   TRN 00000000
             MSS    010184F8   PK-FLG F0850409   FLG   00001B1B   LLS 00018078   JLB    00018458   JPQ 000180A0
             FSA    01030FB0   TCB      00000000   TME   00000000   JST 00017628   NTC    00000000   OTC 000182B0
             LTC    00000000   IQE      00000000   ECB   00018028   STA 00000000   D-PQE 000197A0   SQS 00017338
             NSTAE 000184A0   TCT      000000C8   USER 8002BE24


ACTIVE RBS

PRb  0183E8  RESV    00000000   APSW    00000000   WC-SZ-STAB 00040082   FL-CDE 00018418   PSW FFF5000D 40024FDA
             Q/TTR   00000000   WT-LNK 00017628

SVRB 0174F8  TAB-LN 004803B0   APSW    F2F0F1C3   WC-SZ-STAB 0012D002   TQN     00000000   PSW 00040033 50009C6A
             Q/TTR   00006104   WT-LNK 000183E8
             RG 0-7    00000001    8000002B    00000000    50018034    00018130    000182B0    000180C8    00018908
             RG 8-15   00018008    000103B0    00030FB0    00000000    40024D9E    00024F00    30000000    00007742
             EXTSA     000029BE    000317A0    200000FF    000315F8    FF030000    00017574    0001757C    E2E8E2C9
                       C5C1F0F1    C9C5C128    C1C2C5D5    C4000000

SVRB 017F70  TAB-LN 000803C0   APSW    F1F0F5C1   WC-SZ-STAB 0012D002   TQN     00000000   PSW FF040001 4003EB64
             Q/TTR   0000AE01   WT-LNK 000174F8
             RG 0-7    0000000C    00017558    8000997C    0000A658    00017628    000174F8    00018418    00000000
             RG 8-15   00017628    400098F2    00017628    000317A0    00018100    0001757C    4000934A    00000000
             EXTSA     00660E00    00000000    C9C7C3F0    F0F0F7C2    0000EB80    0000EB30    6000B67C    40404040
                       40404040    40404040    00000050    00017628


LUAD LIST

        NE 000180B8   RSP-CDE 020180A0          NE 000180C0   RSP-CDE 010193A0          NE 000183C8   RSP-CDE 010192A0
        NE 00000000   RSP-CDE 01019270


CDE

     018418        ATR1 0B   NCDE 000000   ROC-RB 000183E8   NM TASKD      USE 01   EPA 024D98   ATR2 20   XL/MJ 018490
     0180A0        ATR1 30   NCDE 018418   ROC-RB 00000000   NM IGC0A05A   USE 02   EPA 02F9E0   ATR2 28   XL/MJ 018090
     0193A0        ATR1 B0   NCDE 0193D0   ROC-RB 00000000   NM IGG019CD   USE 04   EPA 03EDF8   ATR2 20   XL/MJ 019390
     0192A0        ATR1 B1   NCDE 0192D0   ROC-RB 00000000   NM IGG019BA   USE 04   EPA 03EB98   ATR2 20   XL/MJ 019290
     019270        ATR1 B0   NCDE 0192A0   ROC-RB 00000000   NM IGG019BB   USE 04   EPA 03EB40   ATR2 20   XL/MJ 019260

XL                                                   LN          ADR          LN          ADR          LN          ADR

     018490   SZ 00000010   NO 00000001           80000268    00024D98
     018090   SZ 00000010   NO 00000001           80000620    0002F9E0
     019390   SZ 00000010   NO 00000001           80000208    0003EDF8
     019290   SZ 00000010   NO 00000001           80000180    0003EB98
```



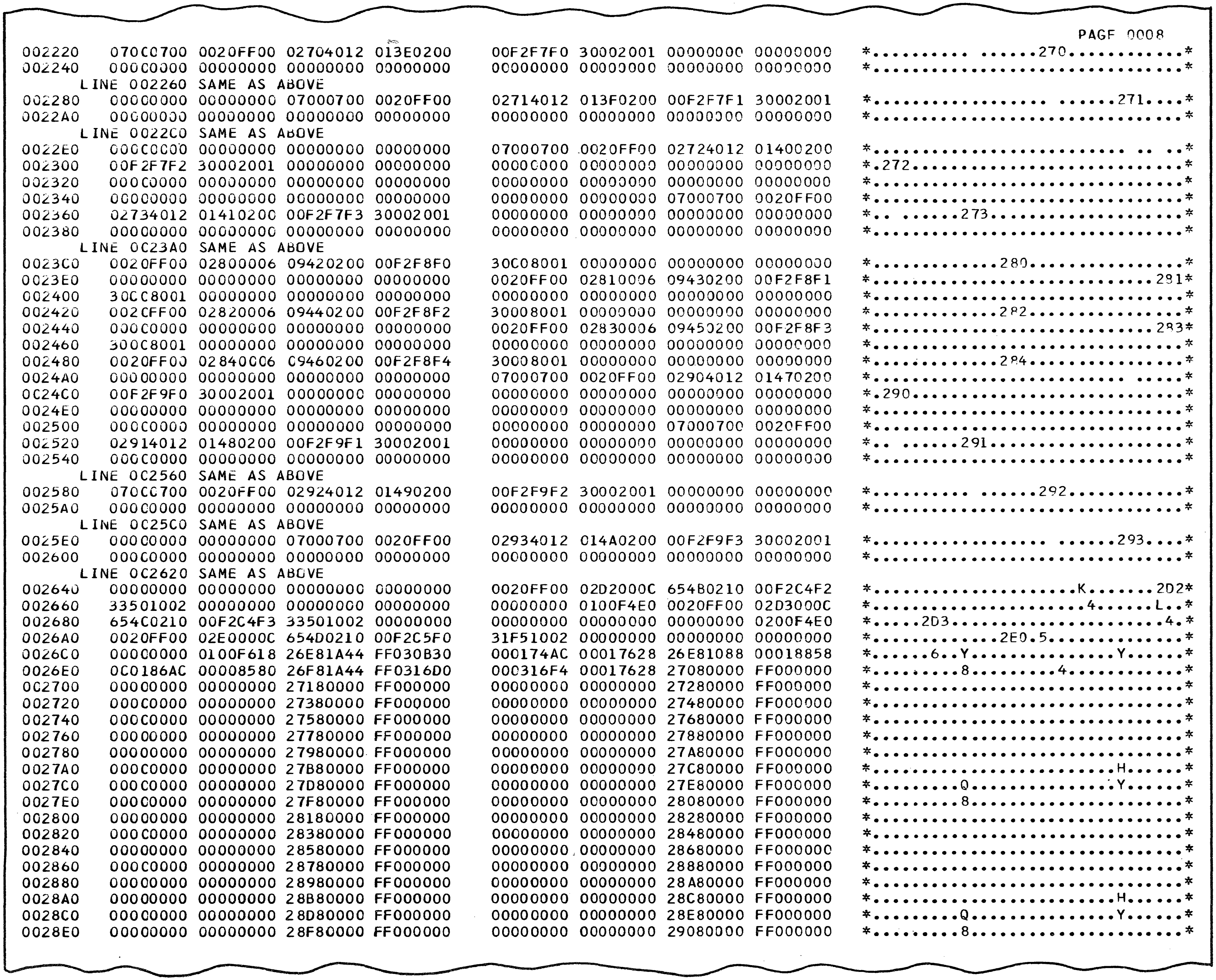
```
                                                                                                     PAGE 0002
    019260    SZ 00000010    NO 00000001        80000058     0003EB40


DEB

017480                          00000C76 00000C76     00000C76 00000C76 00000C76 00000000     *................................*
0174A0    0000020F 00002BE0 0E000000 00017628     04000000 88000000 8F000000 01000000     *................................*
0174C0    1B000000 FF0317A0 04017488 10001A44     00000057 0000005B 00090032 00010000     *................................*
0174E0    C2C2C2C1 C3C40000 00000000 00000000     00000000 00000000                       *BBBACD......................201C*


TIOT    JOB   DANNY       STEP M
     DD             14140100    JOBLIB       00140800     80001AAC
     DD             14040100    SYSABEND     00150600     80001A44


MSS              ************ SPQE ************     *************** DQE ***************     ******* FQE ********
                 FLGS   NSPQE      SPID      DQE       BLK        FQE        LN        NDQE       NFQE          LN

     0184F8      00     0188A8     014      018118     00030000 00030000 00000800 00000000     00000000      00000220
     0188A8      00     018970     251      018408     00024800 00024800 00000800 00000000     00000000      00000598
     018970      80     019790     000      019738
     019738      60     000000     000      0184E0     00030800 00030800 00000800 00000000     00000000      00000328
     019790      40     000000     252      019758     00031000 00031000 00000800 00018080     00000000      00000140
                                                       0002F800 0002F800 00000800 00000000     00000000      000001E0

D-PQE  000197A0    FIRST 00018988    LAST 00018988
PQE   018988    FFB 00025000    LFB 00025000    NPQ 00000000    PPQ 00000000
                TCB 000182B0    RSI 0000D000    RAD 00024800    FLG 00

FBQE 025000     NFB 00018988    PFB 00018988    SZ 0000A800


QCB TRACE

MAJ 0183D0     NMAJ 00018060     PMAJ 000121AC     FMIN 00017BE0     NM   SYSDSN

MIN 017BE0     FQEL 00018A28     PMIN 000183D0     NMIN 00000000     NM FF   DANLIB

               NQEL 00000000     PQEL 00017BE0     TCB   000182B0     SVRB 00017790

MAJ 018060     NMAJ 00000000     PMAJ 000183D0     FMIN 00017F58     NM   SYSIEA01

MIN 017F58     FQEL 00017F48     PMIN 00018060     NMIN 00000000     NM F0    IEA

               NQEL 00000000     PQEL 00017F58     TCB   00017628     SVRB 00017928


SAVE AREA TRACE

TASKD     WAS ENTERED VIA LINK

SA     030FB0    WD1 00000000     HSA 00000000     LSA 00024F00     RET 00010A4A     EPA 01024D98     R0   FD000006
                 R1   00030FF8    R2   00018020    R3   50018034    R4   00018130    R5   000182B0    R6   000180C8
                 R7   00018908    R8   00018008    R9   000103B0    R10 00000050     R11 00000000     R12 6003F45A
```

● SAMPLE DUMP (Part 2 of 11)

● SAMPLE DUMP (Part 3 of 11)

```
                                                                                                              PAGE 0003
SA    024F00  WD1 0106A2D2    HSA 00030FB0    LSA 2706A302    RET 2A06A329    EPA 3306A353    R0   3406A386
              R1   1906A3BA   R2   2C06A3D3   R3   1606A3FF   R4   2706A415   R5   3106A43C   R6   2D06A46D
              R7   3006A49A   R8   2706A4CA   R9   2706A4F1   R10 2006A518    R11 2206A538    R12 2E06A55A


INTERRUPT AT 024FDA


PROCEEDING BACK VIA REG 13

SA    024F00  WD1 0106A2D2    HSA 00030FB0    LSA 2706A302    RET 2A06A329    EPA 3306A353    R0   3406A386
              R1   1906A3BA   R2   2C06A3D3   R3   1606A3FF   R4   2706A415   R5   3106A43C   R6   2D06A46D
              R7   3006A49A   R8   2706A4CA   R9   2706A4F1   R10 2006A518    R11 2206A538    R12 2E06A55A

TASKD    WAS ENTERED VIA LINK

SA    030FB0  WD1 00000000    HSA 00000000    LSA 00024F00    RET 00010A4A    EPA 01024D98    R0   FD000006
              R1   00030FF8   R2   00018020   R3   50018034   R4   00018130   R5   000182B0   R6   000180C8
              R7   00018908   R8   00018008   R9   000103B0   R10 00000050    R11 00000000    R12 6003F45A


NUCLEUS

000000    00000000 00000000 00000000 00000000    0000A658 00000000 01040080 2003BAE2   *...............................S*
000020    FF040001 6000F680 00000000 00000000    0000FF00 00000000 FF060190 00000000   *......6.........................*
000040    000C87C8 0C000000 00000EE8 0000A658    083F9560 00003DE4 00040000 00006888   *...H.......Y...........U........*
000060    00040000 00006EF0 00040000 00006926    00000000 0000D330 00040000 000068F2   *.......0..............L........2*
000080    00000000 00000000 00000000 00000000    00000000 00000000 00000000 00000000   *................................*
      LINES 0000A0-000140 SAME AS ABOVE
000160    000C0000 00000000 00000000 82000170    00040000 0003B610 00000000 00000000   *................................*
000180    FF040001 4003EB64 0000018A 018A018A    FF000190 FF000190 00000001 00017388   *.... ...........................*
0001A0    000317A0 00030B3C 00017388 0003EB40    0002F9E0 00019800 000173D9 00030FB0   *............... ..9........R....*
0001C0    000001E7 A002FC0C 400087D2 00017340    6002FD10 0003EB40 00000000 00000000   *...X.... ..K... ....... ........*
0001E0    00000000 00000000 00000000 00000000    00000000 00000000 00000000 00000000   *................................*
      LINE 000200 SAME AS ABOVE
000220    000C0A82 00000000 41500800 1A551821    92800E5C 41C002B4 1B111804 58420014   *................................*
000240    5834002C D5022015 30194770 078091F0    00214780 0270D300 0E573018 18A0D500   *....N..........0......L.......N.*
000260    0E57A01C 47700780 41F00E4B 45E00712    1B9918A9 91FE3010 47700288 48730022   *.........0......................*
000280    91707012 47800298 4393001C 43A20020    89A09000 587A3020 91F07002 47E00780   *.........................0......*
0002A0    58B20004 1BAA43A7 000A89A0 000341DA    0F8007FC 4012001E D7082008 2008D403   *.................... ...P.....M.*
0002C0    20000E48 927F2004 501B0000 458002E8    47F002E0 47F00362 47000000 45E0068A   *...............Y.0...0..........*
0002E0    181258E0 0DE007FE 48100DAC 12114740    033E9101 10014710 03344071 00029023   *............... .......... .....*
000300    10045001 000C9200 1004D300 100C0021    D2010DAC 10009102 20004710 032691C0   *..........L.....K...............*
000320    402C4710 03B491EA 70064770 800848AD    000607FA D5022015 0DE94780 02FA58A0   * ...................N....Z......*
000340    00244BA0 0E5250A0 002418B0 9620B020    92F009F5 1B9958A0 0DD05090 A00047F0   *.................0.5...........0*
000360    02E04180 02D09111 70064770 03949102    20004710 038A91C0 402C4780 038A4710   *........................ .......*
000380    03B09120 20004788 00084397 000748A9    0F4A07FA 91107006 47800382 D201200C   *............................K...*
0003A0    7014927E 2004D201 20027016 47F004A2    45E006BC 92482004 47F00C5A D2000E10   *......K......0...........0..K...*
0003C0    201841A0 0E1041C0 045450A0 00489120    20004710 03F058F3 001C58FF 000405EF   *.....................0.3........*
0003E0    47F003F0 41E009DC 94F37006 47F006BC    92000048 91017006 47800404 91102001   *.0.0.....3...0..................*
000400    471C040A D3000048 100C4367 000594B7    70064010 70144060 70049C00 600047D0   *....L............. ... .........*
000420    04269250 004405A0 92140DD8 88A00018    42A20010 58900DDC 05B91B99 43907004   *...........Q....................*
000440    1A954079 0F7296A0 70069130 20104710    049605CC 18E84780 06BC910E 00454770   *.. ..................Y..........*
000460    04AA9110 00444780 04AA945B 70069120    00444718 00049608 70069608 0E5C9140   *............................... *
```

• SAMPLE DUMP (Part 4 of 11)



```
                                                                                                          PAGE 0008
002220    070C0700 0020FF00 02704012 013E0200    00F2F7F0 30002001 00000000 00000000    *.......... ......270............*
002240    00000000 00000000 00000000 00000000    00000000 00000000 00000000 00000000    *................................*
      LINE 002260 SAME AS ABOVE
002280    00000000 00000000 07000700 0020FF00    02714012 013F0200 00F2F7F1 30002001    *.................. ......271....*
0022A0    00000000 00000000 00000000 00000000    00000000 00000000 00000000 00000000    *................................*
      LINE 0022C0 SAME AS ABOVE
0022E0    000C0000 00000000 00000000 00000000    07000700 0020FF00 02724012 01400200    *.......................... .. ..*
002300    00F2F7F2 30002001 00000000 00000000    00000000 00000000 00000000 00000000    *.272............................*
002320    000C0000 00000000 00000000 00000000    00000000 00000000 00000000 00000000    *................................*
002340    00000000 00000000 00000000 00000000    00000000 00000000 07000700 0020FF00    *................................*
002360    02734012 01410200 00F2F7F3 30002001    00000000 00000000 00000000 00000000    *.. ......273....................*
002380    00000000 00000000 00000000 00000000    00000000 00000000 00000000 00000000    *................................*
      LINE 0C23A0 SAME AS ABOVE
0023C0    0020FF00 02800006 09420200 00F2F8F0    30C08001 00000000 00000000 00000000    *.............280................*
0023E0    00000000 00000000 00000000 00000000    0020FF00 02810006 09430200 00F2F8F1    *.............................281*
002400    30CC8001 00000000 00000000 00000000    00000000 00000000 00000000 00000000    *................................*
002420    002CFF00 02820006 09440200 00F2F8F2    30008001 00000000 00000000 00000000    *.............282................*
002440    000C0000 00000000 00000000 00000000    0020FF00 02830006 09450200 00F2F8F3    *.............................283*
002460    300C8001 00000000 00000000 00000000    00000000 00000000 00000000 00000000    *................................*
002480    0020FF00 02840006 C9460200 00F2F8F4    30008001 00000000 00000000 00000000    *.............284................*
0024A0    00000000 00000000 00000000 00000000    07000700 0020FF00 02904012 01470200    *.......................... .....*
0C24C0    00F2F9F0 30002001 00000000 00000000    00000000 00000000 00000000 00000000    *.290............................*
0024E0    00000000 00000000 00000000 00000000    00000000 00000000 00000000 00000000    *................................*
002500    000C0000 00000000 00000000 00000000    00000000 00000000 07000700 0020FF00    *................................*
002520    02914012 01480200 00F2F9F1 30002001    00000000 00000000 00000000 00000000    *.. ......291....................*
002540    000C0000 00000000 00000000 00000000    00000000 00000000 00000000 00000000    *................................*
      LINE 0C2560 SAME AS ABOVE
002580    070CC700 0020FF00 02924012 01490200    00F2F9F2 30002001 00000000 00000000    *.......... ......292............*
0025A0    000C0000 00000000 00000000 00000000    00000000 00000000 00000000 00000000    *................................*
      LINE 0C25C0 SAME AS ABOVE
0025E0    00000000 00000000 07000700 0020FF00    02934012 014A0200 00F2F9F3 30002001    *.................. ......293....*
002600    000C0000 00000000 00000000 00000000    00000000 00000000 00000000 00000000    *................................*
      LINE 0C2620 SAME AS ABOVE
002640    00000000 00000000 00000000 00000000    0020FF00 02D2000C 654B0210 00F2C4F2    *.....................K.......2D2*
002660    33501002 00000000 00000000 00000000    00000000 0100F4E0 0020FF00 02D3000C    *......................4......L..*
002680    654C0210 00F2C4F3 33501002 00000000    00000000 00000000 00000000 0200F4E0    *.....2D3......................4.*
0026A0    0020FF00 02E0000C 654D0210 00F2C5F0    31F51002 00000000 00000000 00000000    *.............2E0.5..............*
0026C0    00000000 0100F618 26E81A44 FF030B30    000174AC 00017628 26E81088 00018858    *......6..Y...............Y......*
0026E0    0C0186AC 00008580 26F81A44 FF0316D0    000316F4 00017628 27080000 FF000000    *.........8.........4............*
002700    00000000 00000000 27180000 FF000000    00000000 00000000 27280000 FF000000    *................................*
002720    000C0000 00000000 27380000 FF000000    00000000 00000000 27480000 FF000000    *................................*
002740    000C0000 00000000 27580000 FF000000    00000000 00000000 27680000 FF000000    *................................*
002760    00000000 00000000 27780000 FF000000    00000000 00000000 27880000 FF000000    *................................*
002780    00000000 00000000 27980000 FF000000    00000000 00000000 27A80000 FF000000    *................................*
0027A0    000C0000 00000000 27B80000 FF000000    00000000 00000000 27C80000 FF000000    *.........................H......*
0027C0    000C0000 00000000 27D80000 FF000000    00000000 00000000 27E80000 FF000000    *.........Q...............Y......*
0027E0    000C0000 00000000 27F80000 FF000000    00000000 00000000 28080000 FF000000    *.........8......................*
002800    00000000 00000000 28180000 FF000000    00000000 00000000 28280000 FF000000    *................................*
002820    000C0000 00000000 28380000 FF000000    00000000 00000000 28480000 FF000000    *................................*
002840    00000000 00000000 28580000 FF000000    00000000 00000000 28680000 FF000000    *................................*
002860    000C0000 00000000 28780000 FF000000    00000000 00000000 28880000 FF000000    *................................*
002880    00000000 00000000 28980000 FF000000    00000000 00000000 28A80000 FF000000    *................................*
0028A0    00000000 00000000 28B80000 FF000000    00000000 00000000 28C80000 FF000000    *.........................H......*
0028C0    00000000 00000000 28D80000 FF000000    00000000 00000000 28E80000 FF000000    *.........Q...............Y......*
0028E0    00000000 00000000 28F80000 FF000000    00000000 00000000 29080000 FF000000    *.........8......................*
```

```
                                                                                                                PAGE 0020
        LINES C0A340-00A560 SAME AS ABOVE
00A580    0000A584 00000000 00000000 00000000    00000000 00000000 00000000 0F00A554    *................................*
00A5A0    0400FE50 C0001AAC 00000006 00000055    00090320 00000000 0000A5BC 00000000    *................................*
00A5C0    000C0000 00000000 00000000 00000000    00000000 0F00A58C 0400FE50 C0001A44    *................................*
00A5E0    0000000D 0000002A 0009012C 00018EC0    7B000000 00000000 00000000 00002676    *................................*
00A600    8B0C0000 00000000 00000000 00002676    9B000000 00000000 00000000 00002676    *................................*
00A620    AB0C0000 00000000 00000000 00002676    BB000000 00000000 00000000 80002676    *................................*
00A640    00000000 00000000 80000000 00000000    00000000 00000000 0000721C 00006984    *................................*
00A660    0000A554 000100A0 00000000 00000E64    00007018 0000C33E 0000C2C0 00000F98    *......................C...B.....*
00A680    00000FCC 00006E5C 00001A44 000101CE    0099366F 000103B0 0000C3CC 00010478    *..........................C.....*
00A6A0    00007550 00000000 0A0307FE 0000A58C    000074CC 0000C088 00000000 0000EB30    *................................*
00A6C0    00007468 0000724E 00002CFA 100104CC    000026C8 00000DAC 00019800 0000F692    *...................H..........6.*
00A6E0    000C6A60 00000F80 0000018A 000103B0    00007916 00000000 00007B78 0003FFFF    *................................*
00A700    00000000 00000000 00010608 00000000    000106D4 0000A5EC 00000000 00000000    *...................M............*
00A720    00011674 00000000 00000000 0A0D0000    0000A5F0 00007048 00000000 00000000    *...................0............*
0CA740    000C0000 00000000 0580D200 8D6E0021    92F48D63 92048D6F 58F08DCE 059F91C0    *..........K......4.......0......*
00A760    10084740 802C4710 809245D0 83C047F0    83AE47F0 8184182A 41D083C0 58A20000    *... ...........0...0............*
00A780    41AA0000 05DD47F0 807247F0 81845921    00000789 41700004 1B279680 20005020    *.......0...0....................*
00A7A0    8EE25821 000058B1 000441D0 87C845E0    86EA98B3 8EC69780 200007F9 91802000    *.S...........H.......F.....9....*
00A7C0    920C8D62 471083AE 41220004 58B08EC6    41BB0004 41D083C8 47F08032 182A58A2    *...............F.......H.0......*
00A7E0    000445D0 83C047F0 80D845D0 8CC847F0    821291FF 8D614710 80B447F0 81CE58A2    *.......0.Q...H.0...........0....*
00A800    00001255 47B080C4 41D00008 1B6D19A6    472081A8 18A645D0 83C847F0 80D847F0    *.......D.................H.0.0.0*
00A820    818C58B0 8EC650AB 000447F0 83AE18F8    05805880 8C8A50F0 8DEA41F0 800A8000    *.....F.....0...8.......0...0....*
00A840    8D9A9027 8DD2909E 8DEE9200 8D6E92FF    8D6005EF 92008D60 982E8DD2 07FE0000    *.....K.....................K....*
00A860    058C5880 8C5AD200 8D6E0021 121147A0    812C92FA 8D634160 000B920A 8D6F18A0    *......K.........................*
00A880    41AA0000 18508850 001841B0 8D86501B    00009400 B0001211 47808732 47208718    *................................*
00A8A0    45D083C0 47F0816A 45D08CC8 47F082E2    47F0821A 181747F0 83B818F8 05805880    *.....0.....H.0.S.0.....0...8....*
00A8C0    8BFE50F0 8DEA41F0 812C47F0 80F445D0    8CC847F0 821291FF 8D614780 821A95FA    *...0...0...0.4...H.0............*
00A8E0    8D634780 81B891C0 10084790 81A84590    80449120 100847E0 81B841F0 000447F0    *...........................0...0*
00A900    83B85550 8DA24720 82DE47F0 82A450D0    8D6A58B0 8DAE07FB 95FF8D62 478081E2    *...........0...................S*
00A920    92FF8D62 50608D66 47F082E2 59608D66    47D0820A 59620000 47B080B4 125547B0    *.........0.S....................*
00A940    823A41D0 00081B6D 59620000 47D081A8    47F0823A 58608D66 47F081EA 92FF8D64    *.................0.......0......*
00A960    47F082E2 55508DA2 472082DE 91FF8D61    471082A4 95FF8D62 4780823A 92FF8D62    *.0.S............................*
00A980    47F082E2 91F08D63 4780826A 95FA8D63    4780825E 91C01008 47908256 45908044    *.0.S.0..........................*
00A9A0    91201008 471081B0 41D083B6 50D08D6A    47F08272 41D08194 47F081C4 58300010    *.................0.......0.D....*
00A9C0    58430000 58440004 91C0401D 475082A4    5894007C 9120901E 478082A4 59A08D82    *.......... .....................*
00A9E0    472C82A4 58B08DAA 58908DB2 07F9D705    8D628D62 4160000B 47F08D38 58608D8E    *.............9P..........0......*
00AA00    9118601E 477082D0 58660074 12664770    82B65800 8F0E47F0 82D45800 8F1258D0    *.......................0.M......*
00AA20    001058DD 004807FD 459082EA 98B38EC6    47F083EE 18644110 000241D0 82FC5840    *...............F.0............. *
00AA40    8D8E47F0 83065844 00741244 4780831E    9180402C 478082FC 91104021 471082FC    *...0.............. ....... .....*
00AA60    411C0000 47F08CC8 184647F1 832407F9    41600800 1B1118F1 457085E6 47F0833A    *.....0.H...1...9.......1...W.0..*
00AA80    47FC82B2 59E08D76 477082B2 58208D7A    58720000 41770000 5862000C 58C70004    *.0...........................G..*
00AAA0    1AC719CE 4770839E 5AF70004 50F70004    41660800 5062000C 58E08D7E 58EE0008    *.G.......7...7..................*
00AAC0    582E0014 4B208D5E 502E0014 58208D76    41220800 50208D76 502E0018 58200010    *................................*
00AAE0    D2032080 8D7607F9 50FE0004 507E0000    50E20000 47F08366 91201008 47E083B8    *K......9.........S...0..........*
00AB00    1BFFD705 8D628D62 07FE4590 8AF847F0    869012AA 47A083E6 91201008 471083DE    *..P..........8.0.......W........*
00AB20    4160000B 47F08D38 41F00004 47F083B8    45908BF6 90B38EC6 58008D9A 55508DA2    *.....0...0...0.....6...F........*
00AB40    47D0840C 45908484 47F0847C 12554740    843447F0 841E4160 0FF0196A 4740843C    *.........0..... ...0.....0... ..*
00AB60    459C8484 47F0844E 45908620 98B38EC6    507B0000 50D08D6A 58B08DAE 07FB4590    *.....0.........F................*
00AB80    866247F0 84221BBB 582C0004 14204780    845418C2 47F0843E 12AA4780 84224160    *...0...............B.0..........*
00ABA0    07FF1816 1A6A1661 176145D0 851047F0    842254B0 8D9A4780 847A586B 000419F6    *...............0...............6*
00ABC0    47D0847C 186F98B3 8EC647FD 00041B66    18B6582C 00041420 07895550 8DA24720    *.........F......................*
00ABE0    84C65814 00985811 000CD502 20091019    4740849C 95FF8D62 478084BE 9180101C    *.F........N...... ..............*
00AC00    471C850A 47F084C6 9180101C 4780850A    18E2587E 00001470 4780850A 58F70004    *.....0.F.........S..........7..*
00AC20    1BFA47B9 00041AFA 19F647D0 84E8186F    18B718E7 9107E003 47708502 587E0000    *.........6...Y.....X............*
```

```
                                                                                                                  PAGE 0049
0191E0   B0019210 00000000 C9C7C7F0 F1F9C3C5    0303E998 200191D0 00019228 02019210    *........IGG019CE..Z.............*
019200   00000010 00000001 800000B8 0003EA08    B0019240 00000000 C9C7C7F0 F1F9C1D1    *................... ....IGG019AJ*
019220   0203EA08 20019200 00019258 02019240    00000010 00000001 80000080 0003EAC0    *............... ................*
019240   B0019270 00000000 C9C7C7F0 F1F9C1C9    0303EAC0 20019230 00019288 03019270    *........IGG019AI................*
019260   00000010 00000001 80000058 0003EB40    B00192A0 00000000 C9C7C7F0 F1F9C2C2    *............... ........IGG019BB*
019280   0403EB40 20019260 000192B8 030192A0    00000010 00000001 80000180 0003EB98    *... ............................*
0192A0   B10192D0 00000000 C9C7C7F0 F1F9C2C1    0403EB98 20019290 000192E8 020192D0    *........IGG019BA...........Y....*
0192C0   00000010 00000001 80000068 0003ED18    B0019300 00000000 C9C7C7F0 F1F9C3D1    *........................IGG019CJ*
0192E0   0203ED18 200192C0 00019318 02019300    00000010 00000001 80000078 0003ED80    *................................*
019300   B0019330 00000000 C9C7C7F0 F1F9C3C9    0203ED80 200192F0 00019348 02019330    *........IGG019CI.......0........*
019320   000C0010 00000001 80000070 0003F038    B0019360 00000000 C9C7C7F0 F1F9C3C8    *..............0.........IGG019CH*
019340   0203F038 20019320 00019378 02019360    00000010 00000001 80000058 0003F0A8    *..0...........................0.*
019360   B10193A0 00000000 C9C7C7F0 F1F9C3C3    0203F0A8 20019350 000193B8 030193A0    *........IGG019CC..0.............*
019380   0003E800 00019100 0003E800 00000800    00000010 00000001 80000208 0003EDF8    *..Y.......Y....................8*
0193A0   B00193D0 00000000 C9C7C7F0 F1F9C3C4    0403EDF8 20019390 000193E8 020193D0    *........IGG019CD...8.......Y....*
0193C0   00000010 00000001 80000070 0003F100    B0019400 00000000 C9C7C7F0 F1F9C1D9    *..............1.........IGG019AR*
0193E0   0303F100 200193C0 00019418 02019400    00000010 00000001 80000078 0003F170    *..1...........................1.*
019400   B0019430 00000000 C9C7C7F0 F1F9C1D8    0203F170 200193F0 00019448 02019430    *........IGG019AQ..1....0........*
019420   000C0010 00000001 800000D8 0003F1E8    B0019460 00000000 C9C7C7F0 F1F9C1D2    *...........Q..1Y........IGG019AK*
019440   0203F1E8 20019420 00019478 02019460    00000010 00000001 80000068 0003F2C0    *..1Y..........................2.*
019460   B0019490 00000000 C9C7C7F0 F1F9C1C2    0203F2C0 20019450 000194A8 02019490    *........IGG019AB..2.............*
019480   00000010 00000001 80000068 0003F328    B00194C0 00000000 C9C7C7F0 F1F9C1C1    *..............3.........IGG019AA*
0194A0   0203F328 20019480 000194D8 010194C0    00000010 00000001 80000060 0003F390    *..3........Q..................3.*
0194C0   B90194F0 00000000 C9C5C6E2 C4F1F0F5    0103F390 280194B0 00019508 010194F0    *...0....IEFSD105..3............0*
0194E0   00000010 00000001 80000068 0003F3F0    B9019520 00000000 C9C5C5D7 D7D9C5E2    *..............30........IEEPPRES*
019500   0103F3F0 280194E0 00019538 01019520    00000010 00000001 80000268 0003F458    *..30..........................4.*
019520   B9019550 00018430 C9C5C6E2 C4F2F6F3    0203F458 20019510 00019568 01019550    *........IEFSD263..4.............*
019540   00000010 00000001 80000068 0003F6C0    B9019580 00000000 C9C5C6E2 C4F1F0F2    *..............6.........IEFSD102*
019560   0103F6C0 28019540 00019598 01019580    00000010 00000001 80000058 0003F808    *..6.... ......................8.*
019580   B90195B0 00000000 C9C5C6D8 C9D5E3E9    0103F808 20019570 000195C8 010195B0    *........IEFQINTZ..8........H....*
0195A0   00000010 00000001 80000070 0003F728    B10195F0 00000000 C9C5C5D7 D3C4E2D7    *..............7....0....IEEPLDSP*
0195C0   0103F728 280195A0 00019608 010195F0    0003F000 00019380 0003F000 00000800    *..7............0..0.......0.....*
0195E0   00000010 00000001 80000068 0003F798    B1019620 00000000 C9C5C5D7 C1D3E3D9    *..............7.........IEEPALTR*
019600   0103F798 200195E0 00019638 01019620    00000010 00000001 80000058 0003F860    *..7...........................8.*
019620   B1019650 00000000 C9C5C5E5 D4D5E3F1    0103F860 28019610 00019668 01019650    *........IEEVMNT1..8.............*
019640   00000010 00000001 80000038 0003F8B8    B9019680 00000000 C9C5C5D7 D9E3D540    *..............8.........IEEPRTN *
019660   0103F8B8 28019640 00019698 01019680    00000010 00000001 80000058 0003F8F0    *..8.... ......................80*
019680   B90196C0 00000000 C9C5C5E5 E2E3D9E3    0103F8F0 28019670 00000000 010196C0    *........IEEVSTRT..80............*
0196A0   00000010 00000001 800001B8 0003F948    0003FB00 000195D0 0003F800 00000800    *..............9...........8.....*
0196C0   B9000000 00000000 C9C5C5E5 E6C9D3D2    0303F948 200196A0 40000000 FC0196B0    *........IEEVWILK..9..... .......*
0196E0   000103B8 800196E8 80017D18 00000000    00018708 00001AAC 00001AAC 00D3C802    *.......Y.....................LH.*
019700   00000000 00000000 00000000 00000000    00019798 FB000000 00031800 00031800    *................................*
019720   000C0000 00000000 000179E0 00005000    00031800 00000000 60000000 000184E0    *................................*
019740   000196D8 FB017E70 00035800 00000000    00035800 00000800 00031000 00018080    *...Q............................*
019760   00031000 00000800 000362F8 00017D50    00036000 00000800 00039800 00000000    *...........8....................*
019780   00039800 00000800 00019740 00019778    40000000 FC019758 C0000000 00019738    *........... .... ...............*
0197A0   00019738 FB000000 00018988 00018988    80018960 000188A0 00019718 00019718    *................................*
0197C0   0B000000 00017D18 C9C5C5E5 E6C1C9E3    01036F88 28017E80 050197F8 00000000    *........IEEVWAIT...........8....*
0197E0   0000EB30 0000EB34 0000EB38 0000EB3C    8000EB80 00000000 0000EB80 00000000    *................................*


REGS AT ENTRY TO ABEND

     FLTR 0-6      0501741040000060       3101765840000005             0801761000000000       060174FC10000060

     REGS 0-7      00000001    8000002B    00000000    50018034        00018130    000182B0    000180C8    00018908
```

```
                                                                                                PAGE 0050
    REGS 8-15      00018008    000103B0    00030FB0    00000000            40024D9E    00024F00    30000000    00007742

LOAD MODULE   TASKD

024D80                                                             90ECD00C 05C050D0   *.............0......... ........*
024DA0    C16618AD 41D0C162 50DA0008 4510C030    001D0000 E2E4C2E3 C1E2D240 C440D5D6   *A.....A.............SUBTASK D NO*
024DC0    E64CC9D5 40C3D6D5 E3D9D6D3 40080A23    1B220700 4510C03E 000003E8 58010000   *W IN CONTROL .............Y....*
024DE0    0A0A0700 4510C04E 0E0005DC 58010000    0A0A1222 4770C036 4110C072 4100C07A   *................................*
024E00    41E00030 89E00018 160E0A2F 47F0C0B4    F0F0F1F5 F0F3F2F1 4510C0B2 00340000   *.............0..00150321........*
024E20    E3C1E2D2 C440E3C9 D4C540C9 D5E3C5D9    E5C1D340 C5E7D7C9 D9C5C460 E3C1E2D2   *TASKD TIME INTERVAL EXPIRED.TASK*
024E40    40C2C1C3 D240C9D5 40C3D6D5 E3D9D6D3    0A23D703 C12EC12E 4510C116 06024EC4   * BACK IN CONTROL..P.A.A...A....D*
024E60    00024ECC 004F0000 E3C1E2D2 C440C4D6    D5C540D9 C5D7D3E8 40D5D6D9 D4C1D36B   *........TASKD DONE REPLY NORMAL.*
024E80    C1C2D5C4 E3F06BC1 C2D5C4D1 F06BC1C2    D5C4E3C4 6BC1C2D5 C4D1C46B C7C5E3D4   *ABNDTO.ABNDJO.ABNDTD.ABNDJD.GETM*
024EA0    C1D56BC7 C5E3F8F8 F86BD6D9 40D7D9D6    C7C3D200 0A234110 C12E4100 00010A01   *AN.GET888.OR PROGCK.....A.......*
024EC0    47F0C1AA C1C2D5C4 E3C40000 40000000    D5D6D9D4 C1D3C1C2 D5C4E3F0 C1C2D5C4   *.0A.ABNDTD.. ...NORMALABNDTOABND*
024EE0    D1F0C1C2 D5C4E3C4 C1C2D5C4 D1C4C7C5    E3D4C1D5 C7C5E3F8 F8F8D7D9 D6C7C3D2   *JOABNDTDABNDJDGETMANGET888PROGCK*
024F00    0106A2D2 00030FB0 2706A302 2A06A329    3306A353 3406A386 1906A3BA 2C06A3D3   *...K...........................L*
024F20    1606A3FF 2706A415 3106A43C 2D06A46D    3006A49A 2706A4CA 2706A4F1 2006A518   *...........................1....*
024F40    2206A538 2E06A55A D505C126 C1324780    C24CD505 C126C138 4780C216 D505C126   *........N.A.A...B.N.A.A...B.N.A.*
024F60    C13E4780 C21ED505 C126C144 4780C22E    D505C126 C14A4780 C23ED505 C126C150   *A...B.N.A.A...B.N.A.A...B.N.A.A.*
024F80    4770C1EE 41200001 47F0C036 D505C126    C1564770 C2064100 00084510 C2000A0A   *..A......0..N.A.A...B.......B...*
024FA0    47F0C1F8 D505C126 C15C4770 C0B40000    00000000 41100029 0A0D0700 47F0C226   *.0A8N.A.A....................0B.*
024FC0    4000002A 5810C222 0A0D0700 47F0C236    8000002B 5810C232 0A0D0700 47F0C246   * .....B......0B.......B......0B.*
024FE0    C000002C 5810C242 0A0D58DD 000498EC    D00C92FF D00C41F0 000407FE D9C5C3E3   *......B................0....RECT*

LOAD MODULE   IGC0A05A

02F9E0    4180D099 1B114313 00001A81 41330001    95FF3000 47806068 1BEE1BFF 1B001B11   *................................*
02FA00    43E30000 43030001 8CE00004 88F0001C    8C000004 8810001C 1A2044E0 60701A1E   *.T...........0..................*
02FA20    41818001 44F06076 F384D069 D069DC07    D06962C6 41FFF001 44F0607C 418F8004   *.....0..3..........F..0..0......*
02FA40    413E3003 47F06010 41330001 47F060E2    D2008000 3002D200 D0692000 D2008000   *.....0.......0.SK.....K.....K...*
02FA60    D0695050 D08C5000 D12094FC D1235B00    D1201A10 5800D120 41110003 5010D064   *........J...J...J.....J.........*
02FA80    94FCD067 D703D06C D06C1810 54006290    19014780 60BC1B10 4010D06C 5810D064   *....P...........................*
02FAA0    4A10D06C 1B005D00 66184000 D06A1211    4770611C 4810D06A 12114770 60E85850   *.......... ..................Y..*
02FAC0    D08C5860 D12407F5 4120D121 45B06236    5820D120 413062C4 48A0D06A 4BA0D06C   *....J..5..J.......J....D........*
02FAE0    88A00002 45B0623E 46A06104 9640D112    455062DA 94BFD112 47F060DE 18A15810   *............. J.......J..0......*
02FB00    D12C4B10 D06C5010 D0704120 D07145B0    62364810 D06C8810 00011A31 5820D120   *J.............................J.*
02FB20    45B0623E 5020D120 9640D112 455062DA    94BFD112 4810D06C 12114770 61985810   *......J.. J.......J.............*
02FB40    00105810 10A41921 47B061A6 18125B10    6618D51F 10002000 477061A6 4810D06E   *..................N.............*
02FB60    41110001 4010D06E 41220020 5020D120    46A0615E 47F061B0 1B114010 D06C46A0   *.... .........J.......0... .....*
02FB80    611E47F0 60D44810 D06E1211 4780619E    5810D120 4800D06E 89000005 1B105010   *...0.M............J.............*
02FBA0    D0704120 D0714810 D06E0610 12114770    6202D203 D09F629D 41306294 45B0623E   *..................K.............*
02FBC0    D20DD0AA 62A29640 D1124550 62DA94BF    D112D701 D06ED06E 12AA4780 60D447F0   *K...... J.......J.P..........M.0*
02FBE0    619ED204 D09F629D 41306297 45B0623E    9260D0AB 5810D120 5B106618 5010D070   *..K...................J.........*
02FC00    4120D071 4130629A 45B0623E D20DD0B2    62A247F0 61E64130 62B047F0 65744180   *............K......0.W.....0....*
02FC20    D09995FF 30004780 627C1B00 43030000    1B114313 00011A80 44106284 F384D070   *............................3...*
02FC40    D070DC07 D07062C6 41111001 4410628A    41330002 41220004 47F0623E 41330001   *.......F.................0......*
02FC60    07FE90C8 D200D070 2000D200 8000D070    FFFFFFE0 0B02FF0C 02FF1302 FFD3C9D5   *...HK.....K..................LIN*
02FC80    C5E240E2 C1D4C540 C1E240C1 C2D6E5C5    0002FF09 0312031B 03240330 03390342   *ES SAME AS ABOVE................*
02FCA0    034B03FF 0903FF12 03FF1B03 FF2403FF    3003FF39 03FF4203 FF0098E0 D08012EE   *................................*
02FCC0    47806310 D27CF000 D09441FF 007D50F0    D084411F 007D1910 47D06330 1BFE40F0   *....K.0........0.............. 0*
02FCE0    D090D203 E000D090 41FE0004 50F0D084    4110D048 92201005 58F10008 58F0F030   *..K..........0...........1...00.*
02FD00    05EF4110 D04858E0 100858F0 E03405EF    41100001 4800D05C 9560D098 47406348   *...........0................. ..*
02FD20    47206346 1A011A01 1A019240 D098D277    D099D098 41100038 190147B0 636C4000   *........... ..K............... .*
02FD40    D05C9140 D1120715 47F060E2 4810D05E    41110001 4010D05E 92F1D098 D203D105   *... J....0.S........ ....1..K.J.*
```

```
                                                                                               PAGE 0051
02FD60   63AE4E10 D078F333 D10AD07C 96F0D10D    D201D05C 63B247F0 62DA98E0 D08012EE   *......3.J....0J.K......0........*
02FD80   0785411E 0004191F 078547F0 62FCD7C1    C7C5FFFF 07FEF0F1 F2F3F4F5 F6F7F8F9   *...........0..PAGE....0123455789*
02FDA0   C1C2C3C4 C5C69120 D1384710 64BE9101    D1384710 64CA9180 D1394710 64CA910C   *ABCDEF..J.......J.......J.......*
02FDC0   D1384750 64D29120 D1394710 64D29118    D1394750 64E29110 D1384710 64EE9102   *J....K..J....K..J....S..J.......*
02FDE0   D1394710 64EE58E4 007C13EE 45506504    47F06418 47F06434 947FE020 58100010   *J......U.........0...0..........*
02FE00   58B010C8 58BB0028 188E18AE 05EB18E8    47F0640C 94FE401D D20BD098 652CD203   *...H...........Y.0.... .K.....K.*
02FE20   D124D128 455062DA 4550639A 9879D080    12774780 64661B97 41000000 89000018   *J.J.............................*
02FE40   16091817 0A0A58E0 D0609180 E0024780    649E4110 D070D70B 10001000 41E0656C   *......................P.........*
02FE60   58F0D060 41FF0004 41300004 92FF1000    42301001 50E01004 50F01008 0A3058A0   *.0.......................0......*
02FE80   D1305800 6568181D 0A0A4100 6560D205    A06064B8 47FA0060 0A091BFF 0A0394DF   *J.............K.................*
02FEA0   D13841A0 653847F0 64FA41A0 654047F0    64FA94F3 D13894DF D13941A0 654847F0   *J......0..... .0...3J...J......0*
02FEC0   64FA94E7 D13941A0 655047F0 64FA94FD    D13994EF D13841A0 655841F0 D07050A0   *...XJ......0....J...J......0....*
02FEE0   F0000A07 12EE47A0 650E13EE 07F518FE    58EF0088 12EE0775 58EF0080 59FF007C   *0............5..................*
02FF00   47850004 58FF0084 47F06514 F0C5D5C4    40D6C640 C4E4D4D7 C9C7C3F0 F1F0F5C1   *.........0..0END OF DUMPIGC0105A*
02FF20   C9C7C3F0 F4F0F5C1 C9C7C3F0 F6F0F5C1    C9C7C3F0 F7F0F5C1 C9C7C3F0 F8F0F5C1   *IGC0405AIGC0605AIGC0705AIGC0805A*
02FF40   C9C7C3F0 C1F0F5C1 FD00013A E2E8E2C9    C5C1F0F1 4180D099 907AD09C 187858A0   *IGC0A05A....SYSIEA01............*
02FF60   D12088A0 000589A0 0005D21F 7057A000    4187001F 4197001F 41800001 925C7056   *J.........K.....................*
02FF80   925C7077 91C07057 471065B8 95407057    478065BC 920F7057 943F7057 877865A4   *............. ..................*
02FFA0   DC1F7037 65D4987A D09CD213 D09CD09B    47F0623E 4BC1C2C3 C4C5C6C7 C8C94B4B   *.....M....K......0...ABCDEFGHI..*
02FFC0   4B4B4B4B 4BD1D2D3 D4D5D6D7 D8D94B4B    4B4B4B4B 4B4BE2E3 E4E5E6E7 E8E94B4B   *.....JKLMNOPQR........STUVWXYZ..*
02FFE0   4B4B4B4B F0F1F2F3 F4F5F6F7 F8F94B4B    4B4B4B4B 40000000 00000020 00000000   *....0123456789...... ...........*

LOAD MODULE   IGG019CD

03EDE0                                                            909AD040 9120203C   * ...0...................... ....*
03EE00   4710F144 58A0200C 5810202C 48673006    9101203C 4780F024 4860205A 12664740   *..1...................0........ *
03EE20   F05A1B44 4340A007 18044340 20041B04    91982024 4710F046 1E0647F0 F04A4A00   *0.... ..... ..........0....00...*
03EE40   203E4900 201247C0 F1664900 A00447C0    F06C4810 F06A4100 00808900 00181610   *........1.......0...0...........*
03EE60   0A0D2000 41202005 42730004 183F184E    185B186C 187D58F0 001058F0 F02005EF   *.......................0...00...*
03EE80   589031FC 1E091880 58F00010 58F0F01C    05EF12FF 478030A8 96202037 47F0310A   *.........0...00..............0..*
03EEA0   95011010 4770310A 95011010 4770310A    91041008 47E0310A 1B991BAA 1BBB1BCC   *................................*
03EEC0   43920000 89900004 43A91029 43B9102D    43C20006 19CA4740 30E819BC 47B0310A   *.................B..... .Y......*
03EEE0   18081BBA 58902007 43A09003 1BAB46A0    30FA89A0 00101E0A 58F00010 58F0F01C   *.........................0...00.*
03EF00   05EF18F3 41900005 1B295880 20305830    204458A0 200C18D7 1B774373 000418B5   *...3...................P........*
03EF20   18C618E4 48673006 9101203C 4780F13C    4860205A 9120203C 4780F158 9620203C   *.F.U..........1...........1.....*
03EF40   96013014 5843000C 92404000 47F0F1F4    58A0200C D2012012 A0049200 200C1846   *.........  ..014....K...........*
03EF60   91982024 47C0F174 4840203E 4850A00A    1C448E40 00094340 A0061E54 43402004   *......1.. ......... ... ..... ..*
03EF80   1F544840 20121B45 40402012 D2073028    20051B44 4340200C 41440001 4240200C   *... ....  ..K........ ....... ..*
03EFA0   914C3000 4710F1B8 4243002F 47F0F1EC    43402040 18531A54 43402041 89400003   *. ....1......01.. . ..... ... ..*
03EFC0   1A54D204 50002008 4B50F200 91802034    4780F1E0 40605000 1B444340 20101B64   *..K.......2.......1. ...... ....*
03EFE0   40650008 41430008 18140A00 989AD040    07FE0000 00010000 00020700 07000700   * .............. ................*

LOAD MODULE   IGG019BA

03EB80                                                            90E8D014 188F5821   *..... ......Y............Y......*
03EBA0   00085830 20445833 000094BF 20304103    00085001 00105013 000C95FF 30044770   *................................*
03EBC0   80349228 100098E8 D01407FE 1B551B44    91C01005 474080F4 91202011 47808050   *.......Y............. .4........*
03EBE0   91802030 4780816E 1B449680 203058F0    204C4340 20409120 20114780 80729104   *...............0... . ..........*
03EC00   203D4780 80724144 00101A43 50430018    94013000 96403000 1B444340 20431874   *..................... ..... ....*
03EC20   1A439120 201147E0 80964350 201091C0    202447B0 80C89140 20244780 80D89120   *.....................H. .....Q..*
03EC40   100547E0 80D85861 000C1A65 D2013006    6000D201 60028010 48630006 47F080DC   *.....Q......K.....K..........0..*
03EC60   91801004 471080D8 48610006 47F080DC    4860203E 1A654064 0006D202 4001100D   *.......Q.....0........ ...K. ...*
03EC80   50302044 05EF98E8 D01407FE 43402042    18741A43 58F02048 94013000 96203000   *.......Y..... .......0..........*
03ECA0   91202011 47E08136 41E00008 9104203D    4780811E 1BEE410E 30285003 00189108   *................................*
03ECC0   20244710 8092947F 203047F0 809294F0    40005043 00189140 10054710 81629602   *...........0...0 ...... ........*
```

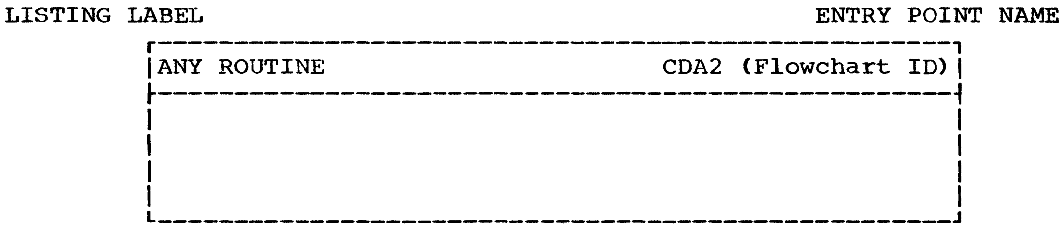
```
                                                                                                            PAGE 0052
03ECE0    400C9150 201147C0 809691C0 202447C0    80969604 400047F0 8096960C 40009640    * .................. ..0.... .. *
03ED00    203047F0 80961851 18120A19 18151B55    5013000C 47F08050                      *...0.................0..........*

LOAD MODULE    IGG019BB

03EB40    90E8D014 185F5821 00085831 00101841    917F4000 4710504C 47405028 18144100    *.Y............... ...... ......*
03EB60    00010A01 47F05010 18121803 4B005008    9001D040 41110000 0A3712FF 47805010    *.....0............. ...........*
03EB80    05EF9801 D0404111 00000A37 98E8D014    07FE0700 07000700                      *..... .......Y...........Y......*

SP 252

031140    065D0000 007D4040 40F0F3C5 C2F6F040    4040F0F0 F0F1F0C1 F0F140F4 F7C6F0F5    *......   03EB60   00010A01 47F05*
031160    F0F1F040 F1F8F1F2 F1F8F0F3 40F4C2F0    F0F5F0F0 F8404040 40F9F0F0 F1C4F0F4    *010 18121803 4B005008     9001D04*
031180    F04CF4F1 F1F1F0F0 F0F040F0 C1F3F7F1    F2C6C640 F4F7F8F0 F5F0F1F0 4040405C    *0 41110000 0A3712FF 47805010   .*
0311A0    4B4B4B4B 4BF04B4B 4B4B4B4B 4B4B4B4B    4B4B4B40 4B4B4B4B 4B4B4B4B 4B4B4B4B    *.....0.............. ...........*
0311C0    5C0C7D40 4040F0F3 C5C2F8F0 404040F0    F5C5C6F9 F8F0F140 C4F0F4F0 F4F1F1F1    *...   03EB80   05EF9801 D0404111*
0311E0    40F0F0F0 F0F0C1F3 F740F9F8 C5F8C4F0    F1F44040 4040F0F7 C6C5F0F7 F0F040F0    * 00000A37 98E8D014     07FE0700 0*
031200    F7FCF0F0 F7F0F040 40404040 40404040    40404040 40404040 40404040 5C4B4B4B    *7000700                        ....*
031220    4B4B404B 4B4B4B4B 4B4BE84B 4B4B4B4B    4B4B4B4B 4B4BE84B 4B4B4B4B 4B5C007D    *.. .......Y...........Y.........*
031240    4040F0E2 D740F2F5 F2404040 40404040    40404040 40404040 40404040 40404040    *  0SP 252                        *
031260    40404040 40404040 40404040 40404040    40404040 40404040 40404040 40404040    *                                 *
      LINE 031280 SAME AS ABOVE
0312A0    404C4040 40404040 40404040 40404040    40404040 40404040 40404000 7D4040F0    *                             ..  0*
0312C0    F0F3F1F1 F4F04040 40F0F6F5 C4F0F0F0    F040F0F0 F7C4F4F0 F4F040F4 F0C6F0C6    *031140    065D0000 007D4040 40F0F*
0312E0    F3C3F540 C3F2C6F6 C6F0F4F0 40404040    F4F0F4F0 C6F0C6F0 40C6F0C6 F1C6F0C3    *3C5 C2F6F040     4040F0F0 F0F1F0C*
031300    F040F4F0 C3F6C6F0 C3F640C6 F1C3F6C6    F0C3F340 40405CF3 C3F540C3 F2C6F6C6    *0 40C6F0C6 F1C6F0C3    .3C5 C2F6F*
031320    F0F4F040 404040F4 F0F4F0C6 F0C6F040    C6F0C6F1 C6F0C35C 007D4040 40F0F3F1    *040     4040F0F0 F0F1F0C...   031*
031340    F3F0F040 4040C6F0 F4F0C6F4 C6F040C3    F3C6F6C3 F6C6F040 C3F3C6F6 F4F0C3F6    *300    F040F4F0 C3F6C6F0 C3F640C6*
031360    40C6F1C3 F3C6F6C3 F6404040 40C6F0C3    F3C6F3F4 F040F4F0 F4F0F5C3 C6F340C3    * F1C3F6C6     F0C3F340 40405CF3 C*
031380    F3C6F5F4 F0C3F340 C6F2C3F6 C6F6C3F6    4040405C F040F4F0 C3F6C6F0 C3F640C6    *3F540C3 F2C6F6C6    .0 40C6F0C6 F*
0313A0    F1C3F6C6 F0C3F340 40404BF3 C3F540C3    F2C6F6C6 5C007D40 4040F0F3 F1F3F2F0    *1C6F0C3     .3C5 C2F6F...   031320*
0313C0    404040C6 F0C6F4C6 F0F4F040 F4F0F4F0    F4F0C6F4 40C6F0C6 F4C6F0C3 F640C6F0    *    F0F4F040 404040F4 F0F4F0C6 F0*
0313E0    C3F6C6F0 F4F04040 4040C3F6 C6F0C3F6    C6F140C3 F6C6F0C3 F3F5C340 F0F0F7C4    *C6F040     C6F0C6F1 C6F0C35C 007D*
031400    F4F0F4F0 40F4F0C6 F0C6F3C6 F1404040    5CF0F4F0 40404040 F4F0F4F0 C6F0C6F0    *4040 40F0F3F1    .040     4040F0F0*
031420    40C6F0C6 F1C6F0C3 4B4B4B40 4040F0F3    F15C007D 404040F0 F3F1F3F4 F0404040    * F0F1F0C...   031...    031340    *
031440    C6F3C6F0 C6F0F4F0 40F4F0F4 F0C3F6C6    F040C6F4 C6F0C3F6 C6F440C3 F6C6F0F4    *F3F0F040 4040C6F0 F4F0C6F4 C6F04*
031460    F0C3F340 404040C6 F3C3F6C6 F6C3F340    C6F6C3F6 C6F0F4F0 40C3F3C6 F3C3F6C6    *0C3      F3C6F6C3 F6C6F040 C3F3C6F*
031480    F640C6F4 C6F0C3F3 C6F64040 405CF3F0    F0404040 C6F0F4F0 C6F4C6F0 40C3F3C6    *6 F4F0C3F6    .300    F040F4F0 C3F*
0314A0    F6C3F6C6 F040C3F3 C6F6F4F0 C3F65C00    7D404040 F0F3F1F3 F6F04040 40F4F0C3    *6C6F0 C3F640C6...    031360    40C*
0314C0    F6C6F1C3 F340C6F3 C3F6C6F6 C3F340C6    F6F4F0F4 F0F4F040 F4F0C3F6 C6F0C3F3    *6F1C3 F3C6F6C3 F6404040 40C6F0C3*
0314E0    40404040 C6F3C3F6 C6F3C6F4 40C6F0F4    F0C6F4C6 F040C6F4 C6F0C6F5 C3F340C3    *     F3C6F3F4 F040F4F0 F4F0F5C3 C*
031500    F6C6F3F4 F0C3F340 40405C40 C6F1C3F3    C6F6C3F6 40404040 C6F0C3F3 C6F3F4F0    *6F340C3    . F1C3F6C6     F0C3F340*
031520    404C40C6 F0C3F3C6 F3F4F05C 007D4040    40F0F3F1 F3F8F040 4040C6F3 C3F6C6F5    *   F0C3F340...    031380    F3C6F5*
031540    C3F640C6 F0C3F3C6 F3C3F640 C6F3C6F4    C6F0F5C3 40F0F0F7 C4F4F0F4 F0404040    *C6 F0C3F3C6 F3F4F05C 007D4040    *
031560    40F4F0C6 F0C6F3C6 F140C6F3 C6F8C6F0    F4F040F4 F0F4F0C3 F6C6F340 C3F3C6F6    * 40F0F3F1 F3F8F040 4040C6F3 C3F6*
031580    C3F6C6F5 4040405C 404040C6 F0C3F3C6    F3F4F04B 4B4B4040 40F0F3F1 F3F8F040    *C6F5    .    F0C3F340...    031380 *
0315A0    4040C6F3 C3F6C6F5 5C007D40 4040F0F3    F1F5F4F0 404040C3 F3C6F6F4 F0C3F640    *  F3C6F5...    031540    C3F640C6 *
0315C0    C6F0C3F3 C6F3C3F6 40C6F3C3 F3C6F6F4    F040C3F6 C6F3C3F6 C6F44040 4040C3F6    *F0C3F3C6 F3C3F640 C6F3C6F4     C6*
0315E0    C6F0C6F5 C3F340F4 F0C6F0C6 F0C6F740    C3F4C6F4 C6F0C6F4 40C6F0F4 F0F4F0F4    *F0F5C3 40F0F0F7 C4F4F0F4 F040404*
031600    F0404040 5CC3F640 C6F0C3F3 C6F3C3F6    40C6F3C6 F4C6F0F5 C340F0F0 F7C4F4F0    *0    .C6 F0C3F3C6 F3F4F05C 007D40*
031620    F4FC4040 405C007D 404040F0 F3F1F5F6    F0404040 F4F0C6F4 C6F0C3F6 40C6F0C3    *40    ...    031560    40F4F0C6 F0C*
031640    F6C6F3C3 F640C6F1 F4F0C3F6 C6F340C3    F6C6F8C3 F6C6F040 404040C6 F4C6F0F4    *6F3C6 F140C6F3 C6F8C6F0     F4F04*
031660    F0C6F440 C6F0C6F4 C6F0C3F3 40C6F6C3    F6C6F3F4 F040C3F3 C6F3C3F6 C6F64040    *0F4 F0F4F0C3 F6C6F340 C3F3C6F6  *
031680    405C40F4 F0C6F0C6 F3C6F140 C6F3C6F8    C6F0F4F0 40F4F0F4 F0C3F6C6 F340C3F3    * . 40F0F3F1 F3F8F040 4040C6F3 C3*
0316A0    C6F65C00 7D404040 F0F3F1F5 F8F04040    40C3F3C6 F6C3F6C6 F540F4F0 F4F0F4F0    *F6...    031580    C3F6C6F5 404040*
```

```
                                                                                                                 PAGE 0053
0316C0    F5C340F4 F0F4F0F4 F0C3F640 C6F0C3F3    C6F3C3F6 40404040 C6F3C6F4 C6F0F4C2   *5C 404040C6 F0C3F3C6    F3F4F04B*
0316E0    40F4C2F4 C2F4F0F4 F040F4F0 C6F0C6F3    C6F140C6 F3C6F8C6 F0F4F040 40405CC3   * 4B4B4040 40F0F3F1 F3F8F040   .C*
031700    F6C6F540 40404B40 4040C6F0 C3F3C6F3    F4F04B4B 4B404040 F0F3F1F3 F8F0405C   *6F5   .   F0C3F340...   031380 .*
031720    007D4040 40F0F3F1 F5C1F040 4040F4F0    F4F0C3F6 C6F340C3 F3C6F6C3 F6C6F540   *..   0315A0   4040C6F3 C3F6C6F5 *
031740    C6F5C3F1 C6F0F4F0 40F4F0F4 F0C6F4C6    F0404040 40C6F4C6 F0C3F3C6 F640C3F6   *F5C1F040 4040F4F0    F4F0C3F6 C6*
031760    C6F3F4F0 C3F340C6 F3C3F6C6 F6C3F340    C6F6C3F6 C6F5F4F0 4040405C 4B4B4040   *F340C3 F3C6F6C3 F6C6F540   ...  *
031780    40FCF3F1 F5C1F040 4040F4F0 F4F0C3F6    C6F340C3 F3C6F6C3 F6C6F540 5C0180A0   * 0315A0   4040C6F3 C3F6C6F5 ....*
0317A0    00000100 14060000 00610009 0100C3D8    00210740 00000001 00004000 00000001   *..............CQ... ...... .....*
0317C0    00000001 54000000 002C0020 000172A4    9203EB98 0003EB40 0B000001 00000660   *....................... ........*
0317E0    30040048 41030B28 0103EDF8 0003EDF8    0000007D 00000001 4510902E 0A0A5010   *...........8...8................*

SP 014

030220    C10C4335 00014155 00020630 4430C0FA    41330001 D2032000 C1084232 00019067   *A...................K...A.......*
030240    20041A53 1A731578 4780C146 4122000C    47F0C0B6 D2007000 5000E2E8 E2C4E2D5   *..........A......0..K.....SYSDSN*
030260    40400000 4000D202 B0B44004 41D0B068    181A58F0 C01605EF 47FFC122 47F0C142   *  .. .K... ........0......A..0A.*
030280    05C047F0 C014CCCC C9C5C6E2 C4F1F6F2    05031966 CCCC1881 58608014 48306008   *...0....IEFSD162................*
0302A0    41003006 41E000FD 89E00018 160E4510    C0300A0A 18414030 10044100 10045000   *...................... .........*
0302C0    100C9680 10000630 12334740 C0504430    C48E4190 80F85830 90181B55 43530000   *........... ....D....8..........*
0302E0    12554780 C07A4100 00484510 C06C0A0A    18D14110 80F858F0 C4A205EF 4510C082   *.................J...8.0D.......*
030300    FD000048 58010000 0A0A18D1 4510C092    FD000018 58010000 0A0A5830 80105030   *...........J....................*
030320    10045080 10000700 45F0C0BA 00030334    00000000 C9C5C6E6 F2F1E2C4 0A06183F   *.........0..........IEFW21SD....*
030340    1821181D 47F0C0CA FD000048 5800C0C6    0A0A1233 4780C0F4 48140004 41010006   *.....0.........F.......4........*
030360    41E000FD 89E00018 160E1814 0A0A58F0    C49E1812 07FF9680 60AC4190 80F841D0   *...............0D............8..*
030380    80684110 80B05010 90209203 90089202    90205870 20045070 10005060 10085830   *................................*
0303A0    70005030 10045030 80105830 60005030    100C5030 80149200 100F9200 80171819   *................................*
0303C0    58F0C48A 05EF9200 902047FF C14C47F0    C16C4700 C14C4700 C14C4110 00B08B10   *.0D.........A..0A...A...A.......*
0303E0    000C4100 00808900 00181610 0A0D5830    90185833 00004133 00001233 4780C1A6   *..............................A.*
030400    18D3D700 90089008 181958F0 C4A605EF    183F181D 41000048 0A0A41D0 80681233   *.LP........0D...................*
030420    4780C1A6 47F0C1A2 41300024 4100002C    1A0341E0 00FD89E0 0018160E 18D04510   *..A..0A.........................*
030440    C1C00A0A D7171000 100018B1 06304430    C4944100 B02C5000 B00842D0 B0084110   *A...P...........D...............*
030460    8054D502 6005C47A 4780C1F0 D2021001    60055040 B0004110 B024D207 1000606C   *..N...D...A0K...... ......K.....*
030480    5010B02C 58306004 88300006 5430C4B2    5030B040 5830200C 5030B014 58508000   *..............D.... ............*
0304A0    5840500C 5040B010 D7033000 400CD703    400C3000 D7033000 400C5080 B00C58A4   *. ... ..P... .P. ...P... .......*
0304C0    000C1B00 91807005 4780C2C6 D5027020    60004770 C2B04100 00381840 064041E0   *..........BFN.......B...... . ..*
0304E0    00FD89E0 0018160E 4510C26A 0A0A1831    4440C4AA D2073028 A01C4110 30345010   *..........B...... D.K...........*
030500    3034D202 30253035 D2033034 C47E1812    92801000 94F01000 43E10000 50310000   *..K.....K...D........0..........*
030520    42E10000 0A139101 30304710 C3265030    80601B00 4300A018 12004780 C3901AA0   *............C...............C...*
030540    9540A01C 4780C2B2 91806043 4780C390    41000038 18400640 41E000FD 89E00018   *. ....B.......C...... . ........*
030560    160E4510 C2E40A0A 18314440 C4AAD207    3028A01C 41103034 50103034 D2023025   *....BU..... D.K.............K...*
030580    3035D203 3034C47E 18129280 100094F0    100043E1 00005031 000042E1 00000A13   *..K...D........0................*
0305A0    91013030 4780C38C D207B024 C4824100    00384140 00FD8940 00181604 18130A0A   *......C.K...D...... ... ........*
0305C0    5810B000 48410004 41040006 414000FD    89400018 16040A0A 41400008 41040006   *............. ... ....... ......*
0305E0    414000FD 89400018 16044510 C36C0A0A    5010B000 40401004 41001004 50001000   *. ... ......C.......  ..........*
030600    96801000 D2071006 A01C47F0 C3905030    B0305840 B0105830 B014D703 3000400C   *....K......0C...... ......P... .*
030620    D703400C 3000D703 3000400C 4110B020    D7031000 10005010 B0349102 60044780   *P. ...P... .....P...............*
030640    C3C49680 B0341B33 43308000 8B300004    4230B046 47F0C3FA 00000000 00000000   *CD...................0C.........*
030660    00000000 00000000 00000000 00000000    00000000 40404040 40404040 58A00010   *....................        ....*
030680    58A0A094 D2038030 C47AD501 A028C47A    4780C422 D50160A6 A02647B0 C422D201   *....K...D.N...D...D.N.......D.K.*
0306A0    803260A6 4110B018 5010B004 4130B020    50301000 41405018 50401004 96801004   *..................... ... ......*
0306C0    18170700 47F0C44A FD0000B0 5800C446    0A0A1816 47F0C45A FD0000B0 5800C456   *.....0D.......D......0D.......D.*
0306E0    0A0A1812 47F0C46A FD000018 5800C466    0A0A181B 58F0C49A 07FF0000 00000000   *.....0D.......D......0D.........*
030700    89000000 C9C5C6E2 C4F0E7E7 00030830    D2001006 6044D203 B02CC3D6 00030770   *....IEFSD0XX....K.....K...CO....*
030720    00030818 00030928 00030F08 D2003000    C4B60000 03000000 00000000 00000000   *............K...D...............*
030740    00000000 00000000 00000000 00000001    00000200 00000001 00000001 00000000   *................................*
```

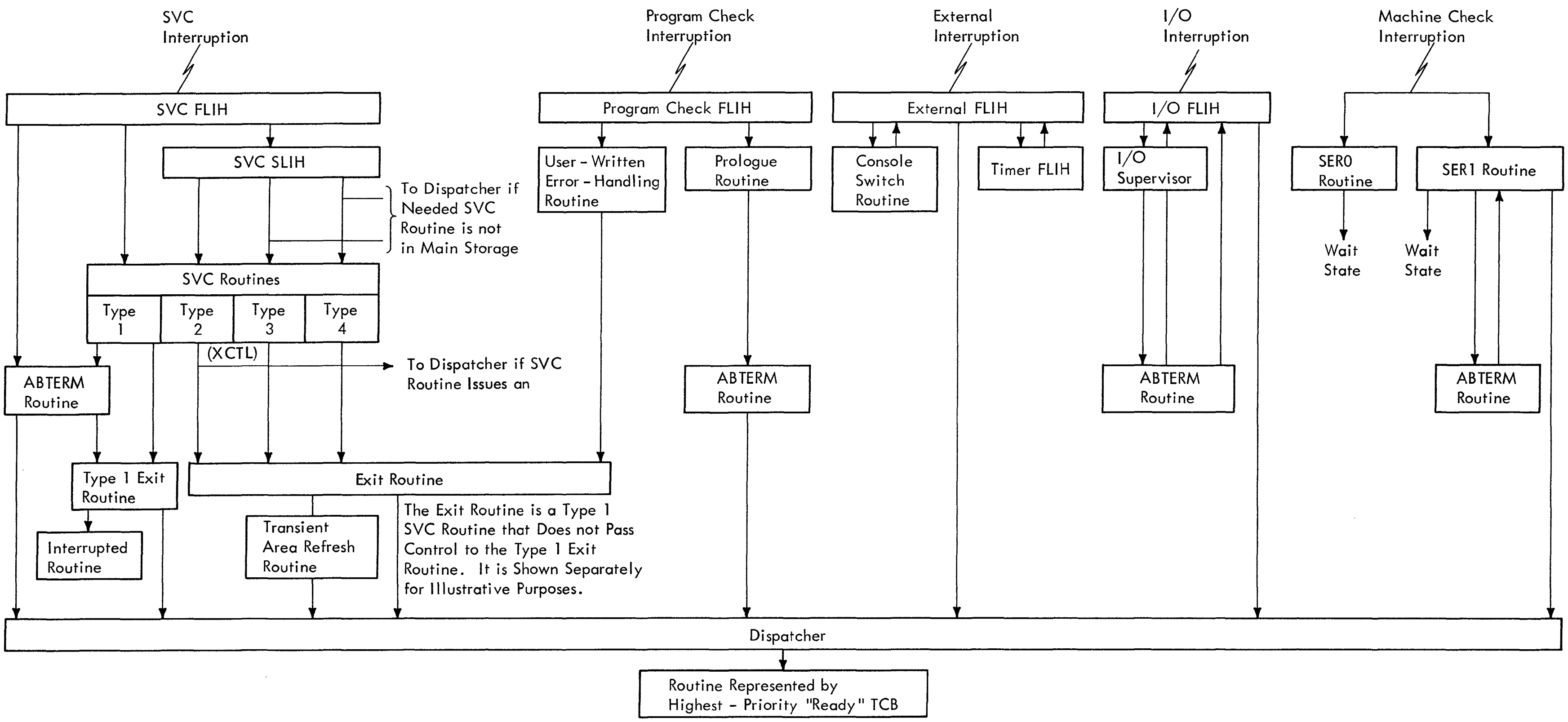
```
                                                                                                              PAGE 0054
030760    F0404040 40404040 0200D008 00000000    05C047F0 C014CCCC C9C5C6E2 C4F1F6F3    *0       ...........0....IEFSD163*
030780    06081966 CCCC9827 10005880 70045877    00001857 41A08008 18914110 00FF8910    *................................*
0307A0    0018180A 16014510 C0380A0A 41610008    50810004 4980C0A2 4740C060 D2FF6000    *......................... ..K...*
0307C0    70004170 71004160 61004B80 C0A247F0    C0421288 4780C06C 06804480 C09C5010    *...............0................*
030 7E0   90144180 10085080 10001815 180A0A0A    58B9000C 18190700 45F0C09A 00030804    *.........................0......*

SP 000

030B20                      41030B28 4803065D    00000000 7F017388 00030B78 0C000000    *...... .........................*
030B40    40030B58 000317A0 00000000 00000000    07000000 62000100 31030B53 40000005    * ........................... ...*
030B60    08030B58 00000000 1D030B78 A0000008    00031140 2000065D 00620001 0100065D    *................... ............*
030B80    7018989B 702418D5 97809006 47F080A0    1B554960 90044780 819C4950 90044780    *.......N.....0..................*
030BA0    812C5950 90004780 812C5890 900047F0    810C4850 B0804140 A0584120 90085020    *...............0....... ........*
030BC0    40104144 00204122 00B04650 8138185D    18489046 7018909B 70249180 90064780    * ...............................*
030BE0    817458F0 821A05EF 58870018 12FF4770    818E989B 70249780 900658F0 821A05EF    *...0.......................0....*
030C00    58870018 989B7024 D2019004 702212FF    47808196 58D7001C 47F08208 98467018    *........K............P...0......*
030C20    18D55990 A04C4780 81C45850 A04C1845    58550000 19594770 81A8D203 40009000    *.N.......D................K. ...*
030C40    D2039000 A04C5090 A04C5860 30205846    00001B22 8D200008 88300008 12224780    *K...............................*
030C60    82060620 41C90008 411000B0 1C021AC1    95043008 47808200 D2AFC000 40009680    *.....I.........A........K... ...*
030C80    900647F0 8206D2AF 4000C000 1BFF50FD    00101817 41000038 0A0A98EC D00C07FE    *...0..K. .......................*
030CA0    00030CA8 908892FF 05804860 B08050E7    00309180 90064780 80189205 701047F0    *...............X...............0*
030CC0    801C9206 70104150 A05818C6 41D00020    1CCC1AD5 50D70014 184D9505 70104770    *...........F.......N.P..........*
030CE0    804048D0 900447F0 80581B22 8D200008    41100001 19128C20 00084780 809648D7    *. .....0.......................P*
030D00    00224820 B07A4BD0 B0864740 807A1BCC    4810B07E 1DC11AD2 89D00008 41DCD001    *........... .........A.K........*
030D20    47F0808A 4AD0B086 4AD0B084 06208920    00081AD2 410D0000 89000008 47F0809E    *.0.................K.........0..*
030D40    58103020 58010004 4110B034 50070034    182758B0 001058F0 B01C05EF 58070034    *.......................0........*
030D60    5810301C D2071024 7000D204 40007003    D2005010 70104144 00054155 00204660    *....K.....K. ...K...............*
030D80    80DC47F0 81DA8C00 00108810 00184111    000158B0 001058B0 B00C4820 B07A1902    *...0............................*
030DA0    47B0810A 4820B084 4A20B086 4B208256    47F0810E 4820B07E 19124720 813043F7    *.................0.............7*
030DC0    000741FF 000142F7 00078910 00188D00    0010947F 500047F0 80C04110 00011A01    *.......7...............0........*
030DE0    89100018 8D000010 4110B034 50070034    41270008 58B00010 58F0B01C 05EF5807    *.........................0......*
030E00    0034D501 7003700B 47708170 D2027005    700D9680 500047F0 80C0D207 70007008    *..N.........K..........0..K.....*
030E20    41000018 4510817E 0A0A4121 00044B50    825AD207 10105018 50205018 920B5018    *..................K.............*
030E40    41C00006 9240501C 40C0501E 41C05018    4A50825A 50C1000C D70B1000 1000D201    *..... .. ............A..P.....K.*
030E60    10067003 58C70014 4BC08258 182C58CC    000012EC 477081C2 50120000 58070034    *.....G.................B........*
030E80    47FC80C0 5810301C 58210014 50270010    58A70028 4120A058 50210014 41110004    *.0..............................*
030EA0    0A005810 301C4100 00010A01 58270010    50210014 58C70014 4BC08258 18DC58CC    *.....................G..........*
030EC0    0000122C 47808238 58CC0000 5812000C    D2071000 20101812 41000018 0A0A47F0    *................K..............0*
030EE0    8218D703 D000D000 1BFF5810 301C957F    10004780 825041F0 000C58E7 003007FE    *..P....................0...X....*
030F00    00010008 00204900 90ECD00C 05801831    1B5558A0 301858AA 000041AA 00005890    *................................*
030F20    A04C1B66 91083008 4710807E 41000038    45108026 0A0A1871 91809006 47108040    *............................... *
030F40    58990000 12994780 80B447F0 802A1255    478080AE 184D1868 90467018 909B7024    *...........0....................*
030F60    45E080C2 58870020 98467018 989B7024    18D4186F 12FF4780 807047F0 80761255    *...B.............M.........0....*
030F80    47708032 18174100 00380A0A 41000048    45108086 0A0AD247 1000D000 18D15800    *......................K......J..*
030FA0    A048181A 0A0A18F6 58603018 D7026001    00000000 00000000 00024F00 00010A4A    *.......6....P...................*
030FC0    01024D98 FD000006 00030FF8 00018020    50018034 00018130 000182B0 000180C8    *...........8...................H*
030FE0    00018908 00018008 000103B0 00000050    00000000 6003F45A 80030FFC 00000CA8    *......................4.........*

END OF DUMP
```

The flowcharts are arranged, in general, in the order in which the routines are described in this publication. Each flowchart contains entry point names, common routine names, and labels that appear in the listings.

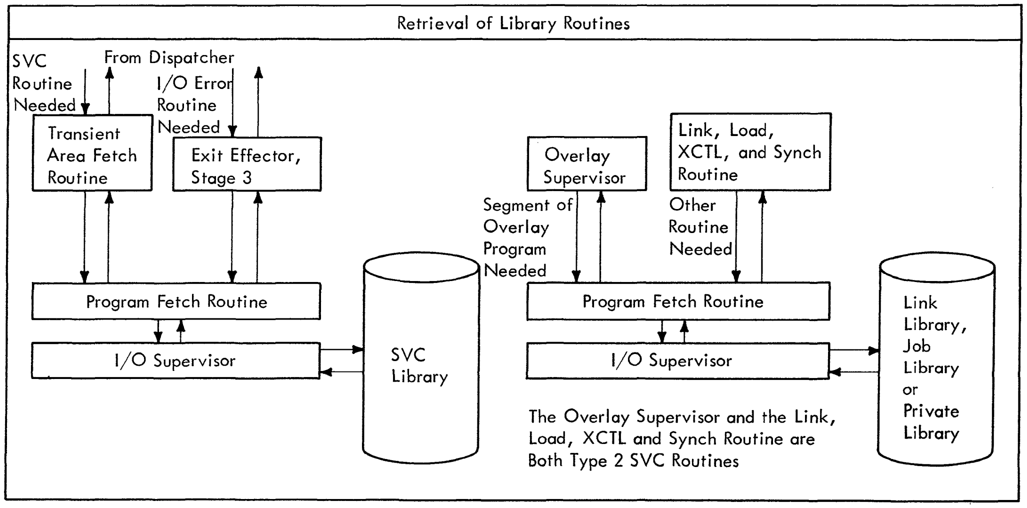
A subroutine block can contain as many as three names: the label from the listing used when the subroutine was invoked, the subroutine's common name used in both the listing and the manual, and the subroutine's entry point name. The label from the listing appears at the top left of the block; the common name and the flowchart identification appear inside the block; the entry point name appears at the top right of the block. An (S) after the entry point name means that the subroutine (or routine) is invoked via supervisor linkage (the SVC interruption handlers). The supervisor linkage path is not shown on each flowchart. It is shown, however, in the overall control flow, Chart 00.



• Chart 00. Overall Control Flow of MVT Supervisor


SVC Interruption
Program Check Interruption
External Interruption
I/O Interruption
Machine Check Interruption
SVC FLIH
SVC SLIH
To Dispatcher if Needed SVC Routine is not in Main Storage
SVC Routines
Type 1
Type 2
Type 3
Type 4
(XCTL)
To Dispatcher if SVC Routine Issues an
ABTERM Routine
Type 1 Exit Routine
Interrupted Routine
Exit Routine
Transient Area Refresh Routine
The Exit Routine is a Type 1 SVC Routine that Does not Pass Control to the Type 1 Exit Routine. It is Shown Separately for Illustrative Purposes.
Program Check FLIH
User - Written Error - Handling Routine
Prologue Routine
ABTERM Routine
External FLIH
Console Switch Routine
Timer FLIH
I/O FLIH
I/O Supervisor
ABTERM Routine
SER0 Routine
SER1 Routine
Wait State
Wait State
ABTERM Routine
Dispatcher
Routine Represented by Highest - Priority "Ready" TCB

Type 1 SVC Routines	Type 2 SVC Routines	Type 3 SVC Routines	Type 4 SVC Routines
CHAP EXIT EXTRACT FREEMAIN GETMAIN POST TIME TTIMER WAIT	Attach Delete DEQ Detach ENQ Exit Effector, Stage 1 Identify Link, Load, XCTL and Synch Overlay Supervisor Spie STIMER	STAE Service WTO, WTOR, WTL	ABDUMP ABEND Checkpoint Comm. Task Router Log and Writelog Post Restart

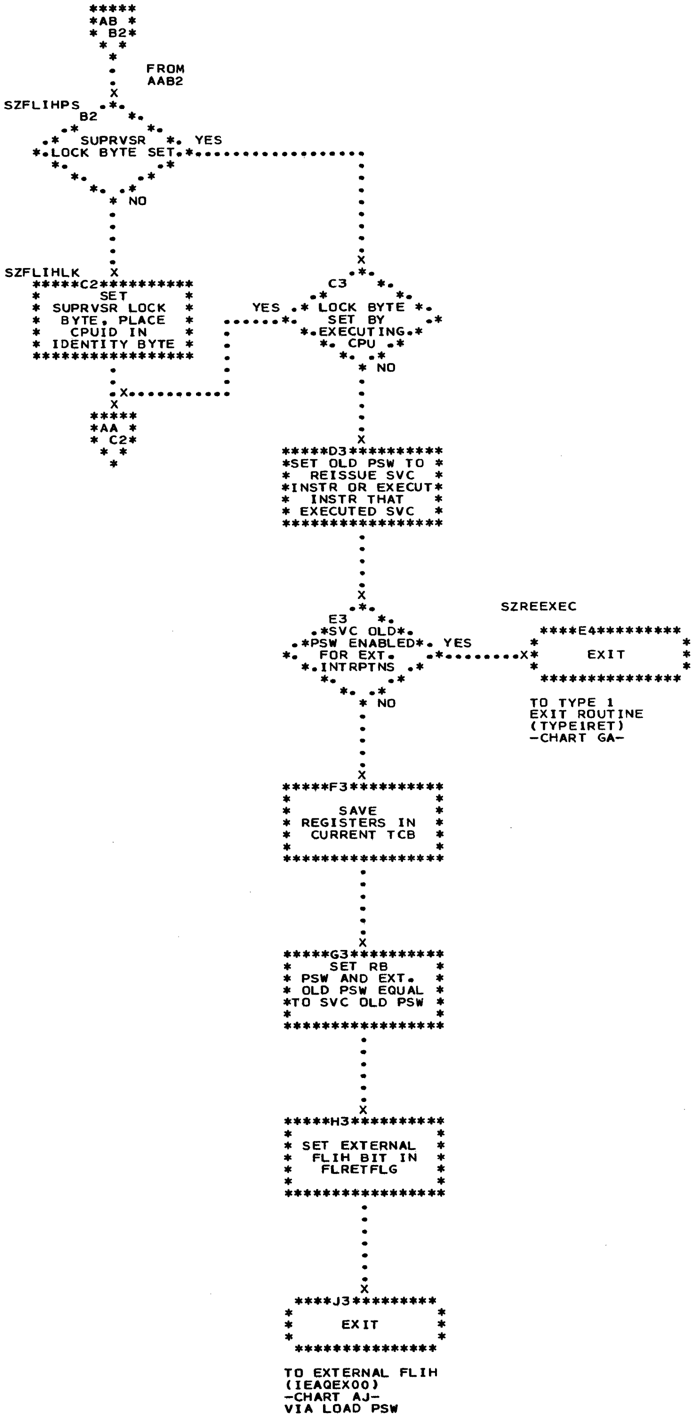

Retrieval of Library Routines
SVC Routine Needed
From Dispatcher
I/O Error Routine Needed
Transient Area Fetch Routine
Exit Effector, Stage 3
Program Fetch Routine
I/O Supervisor
SVC Library
Overlay Supervisor
Link, Load, XCTL, and Synch Routine
Segment of Overlay Program Needed
Other Routine Needed
Program Fetch Routine
I/O Supervisor
Link Library, Job Library or Private Library
The Overlay Supervisor and the Link, Load, XCTL and Synch Routine are Both Type 2 SVC Routines

• Chart AA. SVC First-Level Interruption Handler

```
           IEAQSC00

               ****A2*********
              *               *
              *     ENTRY     *
              *               *
               ***************
                      .
                      . ENTERED FOLLOWING
                      . SVC INTERRUPTION
                      .
                      .
                      .
                      .
                      X
              *****B2**********
              *PLACE REGISTER *
              * CONTENTS INTO *
              * SVC REGISTER  *
              * SAVE AREA LOC *
              *   IEASCSAV    *
              *****************
                      .
                      .  ****   NOTE - THIS EXIT APPLIES ONLY
                      .  *AB *         TO MULTIPROCESSING SYSTEMS
                      ..X* B1 *
                      .  *   *
                      .  ****
           TRSVC      X
              *****C2**********
              *   TRACE TRN   *
              *-*-*-*-*-*-*-*-*
           ..X*PLACE PERTINENT* (OPTIONAL)
           .  *INFO INTO TRACE*
FROM       .  *     TABLE     *
ABC2       .  *****************
           *          .
          * *         .
         *AA *        .
         * C2*        .
         *****        .
                      .
                      X
              *****D2**********
              *     USING     *
              *SVC NO. IN SVC *
              *OLD PSW. LOCATE*
              * ENTRY IN SVC  *
              *     TABLE     *
              *****************
                      .
                      .
                      .
                      .
                      .
                      X
                     .*.
                   E2   *.
                 .*       *.              ****E3*********     NOTE-REGISTER 14 SET TO
               .*           *. NO        *               *         CAUSE DISPATCHER TO
              *.VALID SVC NO..*.........X*     EXIT      *         BE ENTERED AFTER
                *.          .*           *               *         ABTERM ROUTINE.
                  *.     .*               ***************
                    *. .*
                      * YES                 TO ABTERM
                      .                     ROUTINE (IEA0AB00)
                      .                     --CHART HE--
                      .
                      .
                      X
           IBMSVC    .*.
                   F2   *.               *****F3*******
                 .*       *.             *             *            ****F4*********
               .*  REQ FOR  *. YES       *     SET      *          *               *
              *. TYPE-1 SVC  .*.........X* TYPE-1 SWITCH *........X*     EXIT      *
                *. ROUTINE .*            *   IEATYPE1   *          *               *
                  *.     .*              *             *            ***************
                    *. .*                **************
                      * NO                                           TO TYPE-1
                      .                                               SVC ROUTINE (IEA0XE00)
                      .                                               CHART GA
                      .
                      .
                      .
                      .
                      X
               ****G2*********
              *               *
              *     EXIT      *
              *               *
               ***************

                TO SVC SLIH (IEAQTR00)
               --CHART AC--
```

TYPE 1 SVC ROUTINES	
CHAP	BE
EXIT	GB-GC
EXTRACT. . .	BH
FREEMAIN . .	DB
GETMAIN. . .	DA
POST	BM
TIME	EA
TTIMER . . .	EC
WAIT	BK

• Chart AB. SVC First-Level Interruption Handler (Multiprocessing System)


AB B2
FROM AAB2
SZFLIHPS
B2
SUPRVSR LOCK BYTE SET
YES
NO
SZFLIHLK
C2
SET SUPRVSR LOCK BYTE, PLACE CPUID IN IDENTITY BYTE
C3
LOCK BYTE SET BY EXECUTING CPU
YES
NO
AA C2
D3
SET OLD PSW TO REISSUE SVC INSTR OR EXECUT INSTR THAT EXECUTED SVC
E3
SVC OLD PSW ENABLED FOR EXT. INTRPTNS
YES
NO
SZREEXEC
E4
EXIT
TO TYPE 1 EXIT ROUTINE (TYPE1RET) -CHART GA-
F3
SAVE REGISTERS IN CURRENT TCB
G3
SET RB PSW AND EXT. OLD PSW EQUAL TO SVC OLD PSW
H3
SET EXTERNAL FLIH BIT IN FLRETFLG
J3
EXIT
TO EXTERNAL FLIH (IEAQEX00) -CHART AJ- VIA LOAD PSW

• Chart AC. SVC Second-Level Interruption Handler

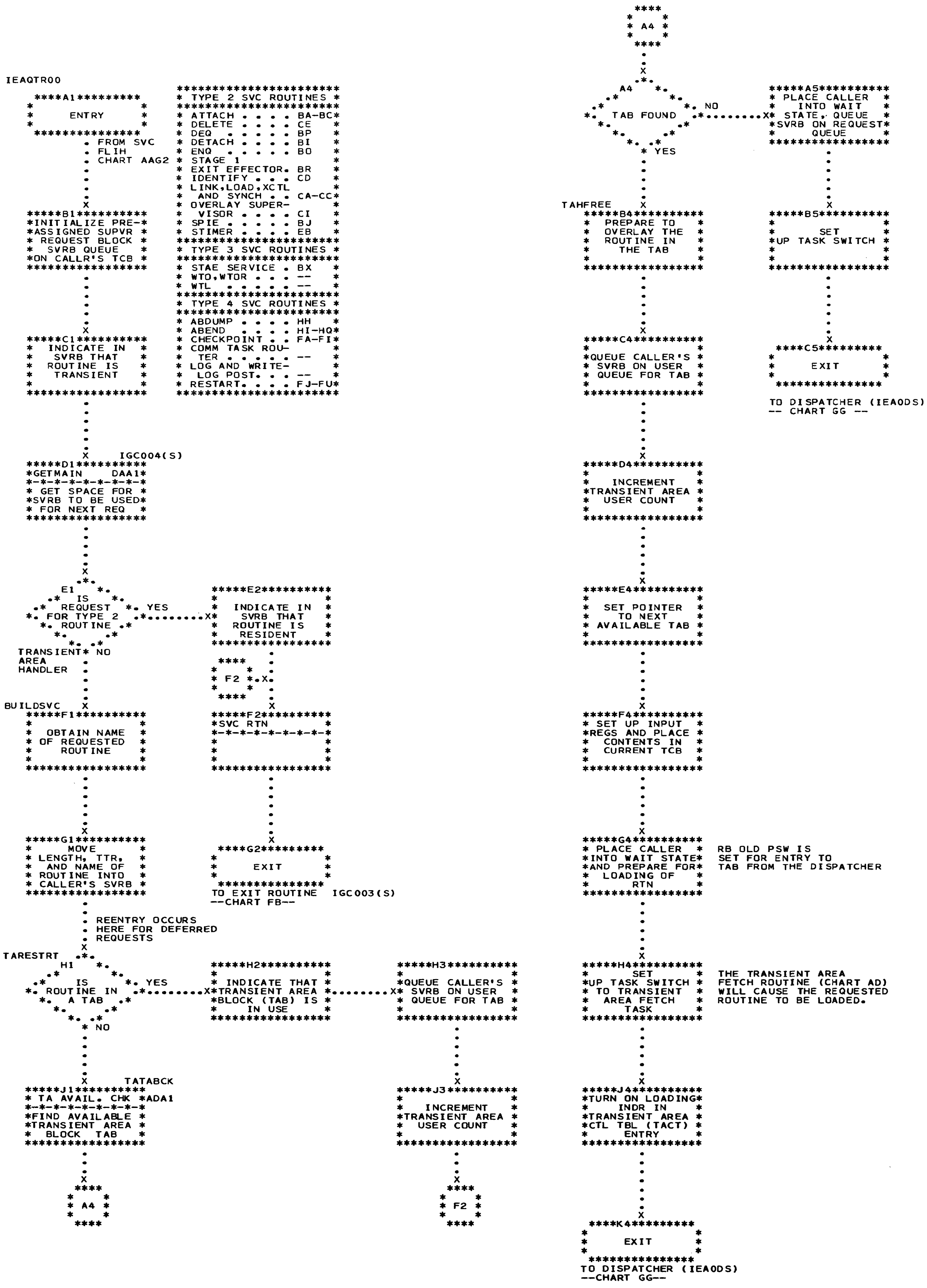

IEAQTR00
A1 ENTRY
FROM SVC FLIH CHART AAG2
TYPE 2 SVC ROUTINES
ATTACH . . . . BA-BC
DELETE . . . . CE
DEQ . . . . . BP
DETACH . . . . BI
ENQ . . . . . BO
STAGE 1
EXIT EFFECTOR. BR
IDENTIFY . . . CD
LINK,LOAD,XCTL AND SYNCH . . CA-CC
OVERLAY SUPERVISOR CI
SPIE . . . . . BJ
STIMER . . . . EB
TYPE 3 SVC ROUTINES
STAE SERVICE . BX
WTO,WTOR . . . --
WTL . . . . . --
TYPE 4 SVC ROUTINES
ABDUMP . . . . HH
ABEND . . . . HI-HQ
CHECKPOINT . . FA-FI
COMM TASK ROUTER . . . . . --
LOG AND WRITE-LOG POST. . . --
RESTART. . . . FJ-FU
B1 INITIALIZE PRE-ASSIGNED SUPVR REQUEST BLOCK SVRB QUEUE ON CALLR'S TCB
C1 INDICATE IN SVRB THAT ROUTINE IS TRANSIENT
IGC004(S)
D1 GETMAIN DAA1 GET SPACE FOR SVRB TO BE USED FOR NEXT REQ
E1 IS REQUEST FOR TYPE 2 ROUTINE
YES
NO
TRANSIENT AREA HANDLER
E2 INDICATE IN SVRB THAT ROUTINE IS RESIDENT
F2
BUILDSVC
F1 OBTAIN NAME OF REQUESTED ROUTINE
F2 SVC RTN
G1 MOVE LENGTH, TTR, AND NAME OF ROUTINE INTO CALLER'S SVRB
G2 EXIT
TO EXIT ROUTINE IGC003(S) --CHART FB--
REENTRY OCCURS HERE FOR DEFERRED REQUESTS
TARESTRT
H1 IS ROUTINE IN A TAB
YES
NO
H2 INDICATE THAT TRANSIENT AREA BLOCK (TAB) IS IN USE
H3 QUEUE CALLER'S SVRB ON USER QUEUE FOR TAB
TATABCK
J1 TA AVAIL. CHK ADA1 FIND AVAILABLE TRANSIENT AREA BLOCK TAB
J3 INCREMENT TRANSIENT AREA USER COUNT
A4
F2
A4
A4 TAB FOUND
NO
YES
A5 PLACE CALLER INTO WAIT STATE, QUEUE SVRB ON REQUEST QUEUE
TAHFREE
B4 PREPARE TO OVERLAY THE ROUTINE IN THE TAB
B5 SET UP TASK SWITCH
C4 QUEUE CALLER'S SVRB ON USER QUEUE FOR TAB
C5 EXIT
TO DISPATCHER (IEA0DS) -- CHART GG --
D4 INCREMENT TRANSIENT AREA USER COUNT
E4 SET POINTER TO NEXT AVAILABLE TAB
F4 SET UP INPUT REGS AND PLACE CONTENTS IN CURRENT TCB
G4 PLACE CALLER INTO WAIT STATE AND PREPARE FOR LOADING OF RTN
RB OLD PSW IS SET FOR ENTRY TO TAB FROM THE DISPATCHER
H4 SET UP TASK SWITCH TO TRANSIENT AREA FETCH TASK
THE TRANSIENT AREA FETCH ROUTINE (CHART AD) WILL CAUSE THE REQUESTED ROUTINE TO BE LOADED.
J4 TURN ON LOADING INDR IN TRANSIENT AREA CTL TBL (TACT) ENTRY
K4 EXIT
TO DISPATCHER (IEA0DS) --CHART GG--

Chart AD. Transient Area Availability Check Routine

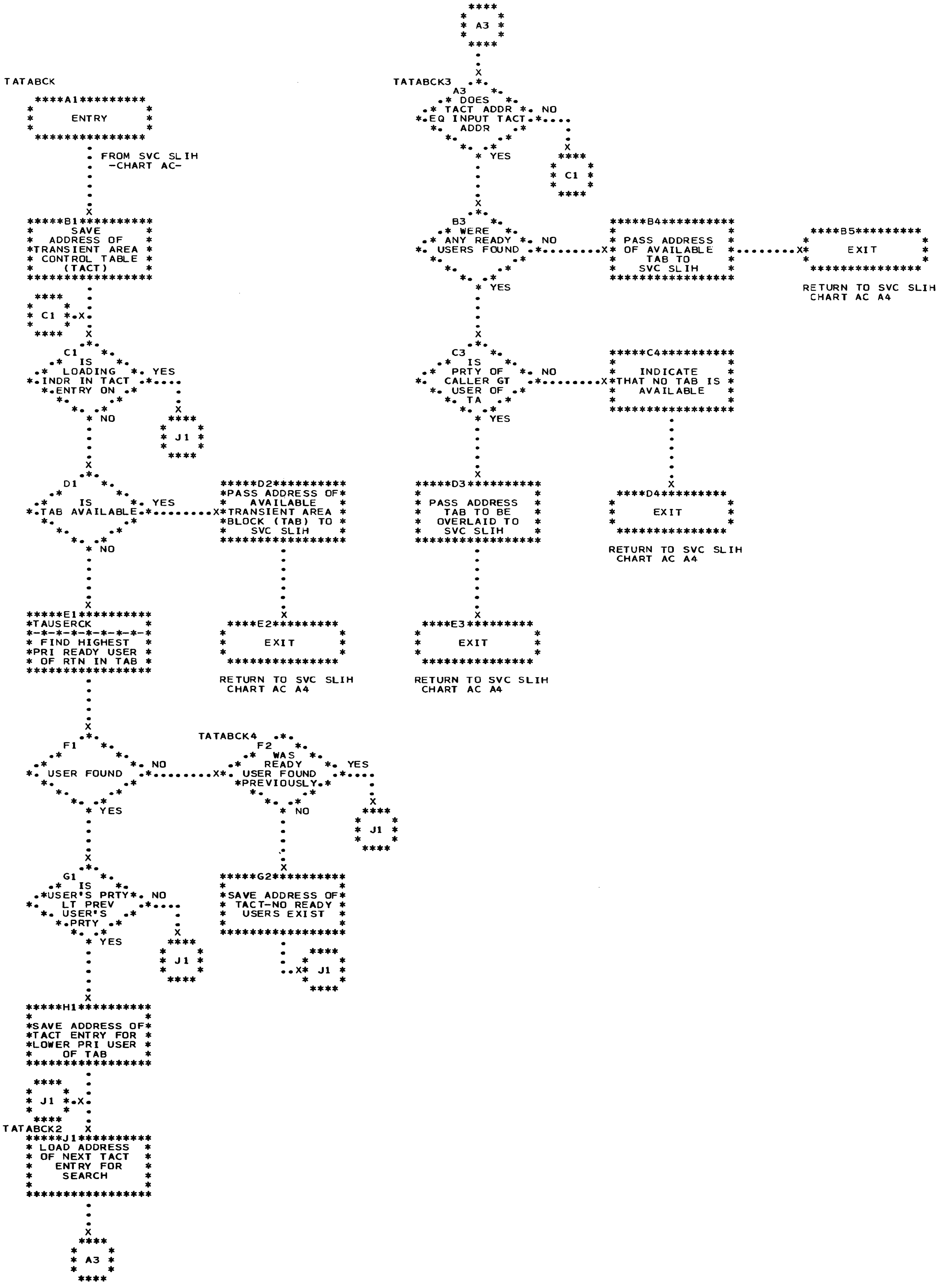
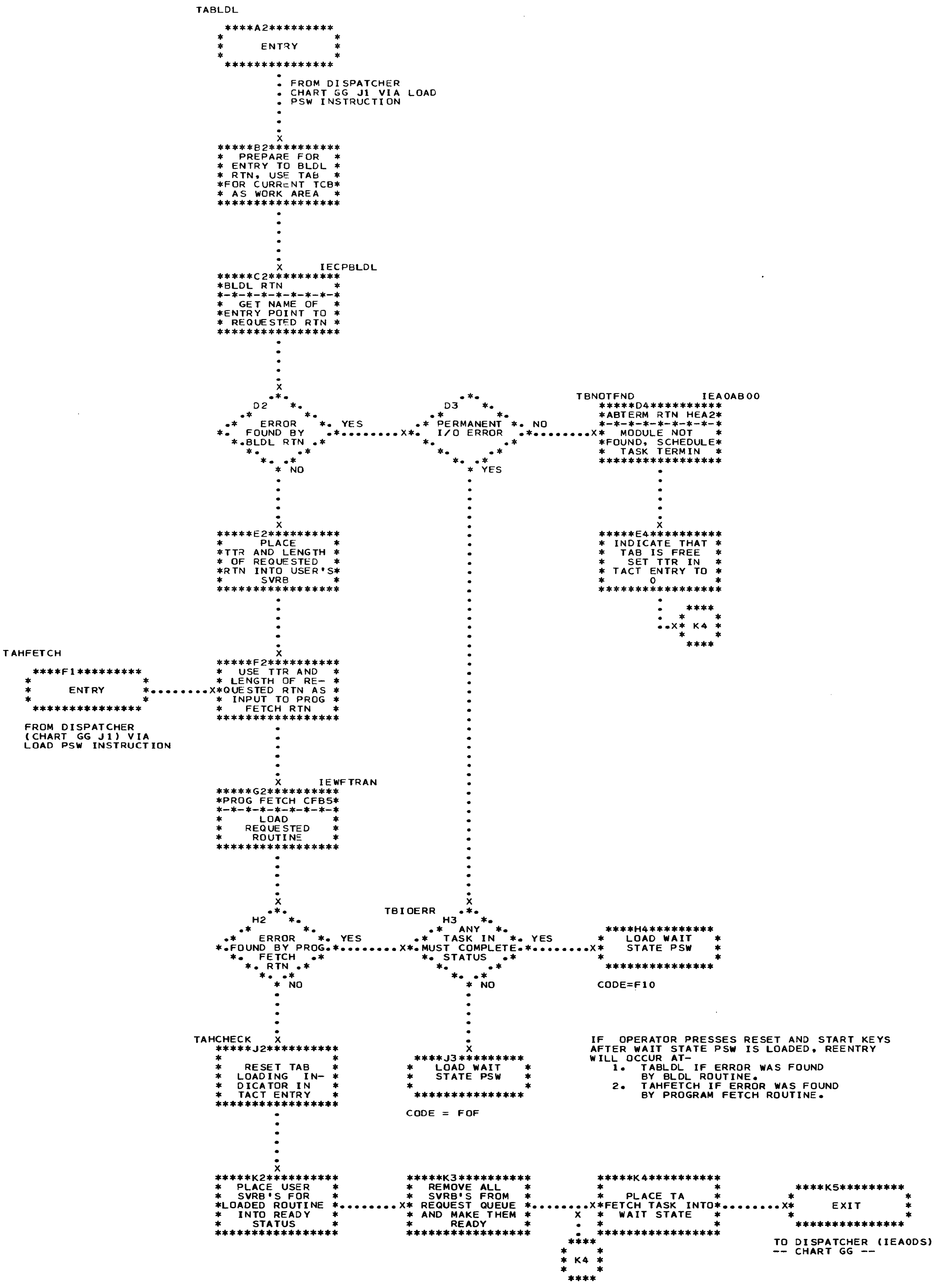
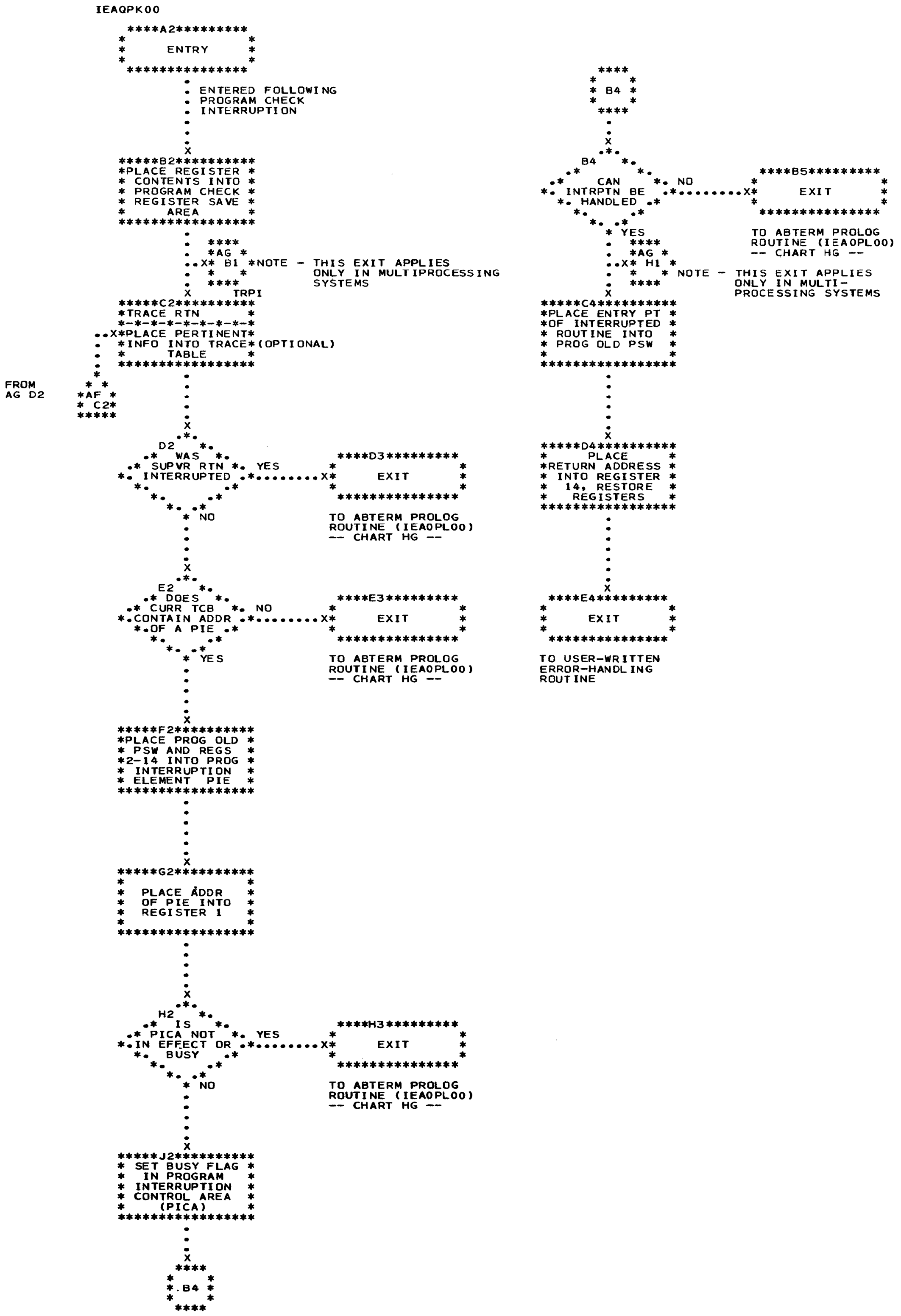

TATABCK
A1 ENTRY
FROM SVC SLIH -CHART AC-
B1 SAVE ADDRESS OF TRANSIENT AREA CONTROL TABLE (TACT)
C1
C1 IS LOADING INDR IN TACT ENTRY ON
YES
J1
NO
D1 IS TAB AVAILABLE
YES
D2 PASS ADDRESS OF AVAILABLE TRANSIENT AREA BLOCK (TAB) TO SVC SLIH
E2 EXIT
RETURN TO SVC SLIH CHART AC A4
NO
E1 TAUSERCK FIND HIGHEST PRI READY USER OF RTN IN TAB
F1 USER FOUND
NO
TATABCK4
F2 WAS READY USER FOUND PREVIOUSLY
YES
J1
NO
G2 SAVE ADDRESS OF TACT-NO READY USERS EXIST
J1
YES
G1 IS USER'S PRTY LT PREV USER'S PRTY
NO
J1
YES
H1 SAVE ADDRESS OF TACT ENTRY FOR LOWER PRI USER OF TAB
J1
TATABCK2
J1 LOAD ADDRESS OF NEXT TACT ENTRY FOR SEARCH
A3
A3
TATABCK3
A3 DOES TACT ADDR EQ INPUT TACT ADDR
NO
C1
YES
B3 WERE ANY READY USERS FOUND
NO
B4 PASS ADDRESS OF AVAILABLE TAB TO SVC SLIH
B5 EXIT
RETURN TO SVC SLIH CHART AC A4
YES
C3 IS PRTY OF CALLER GT USER OF TA
NO
C4 INDICATE THAT NO TAB IS AVAILABLE
D4 EXIT
RETURN TO SVC SLIH CHART AC A4
YES
D3 PASS ADDRESS TAB TO BE OVERLAID TO SVC SLIH
E3 EXIT
RETURN TO SVC SLIH CHART AC A4

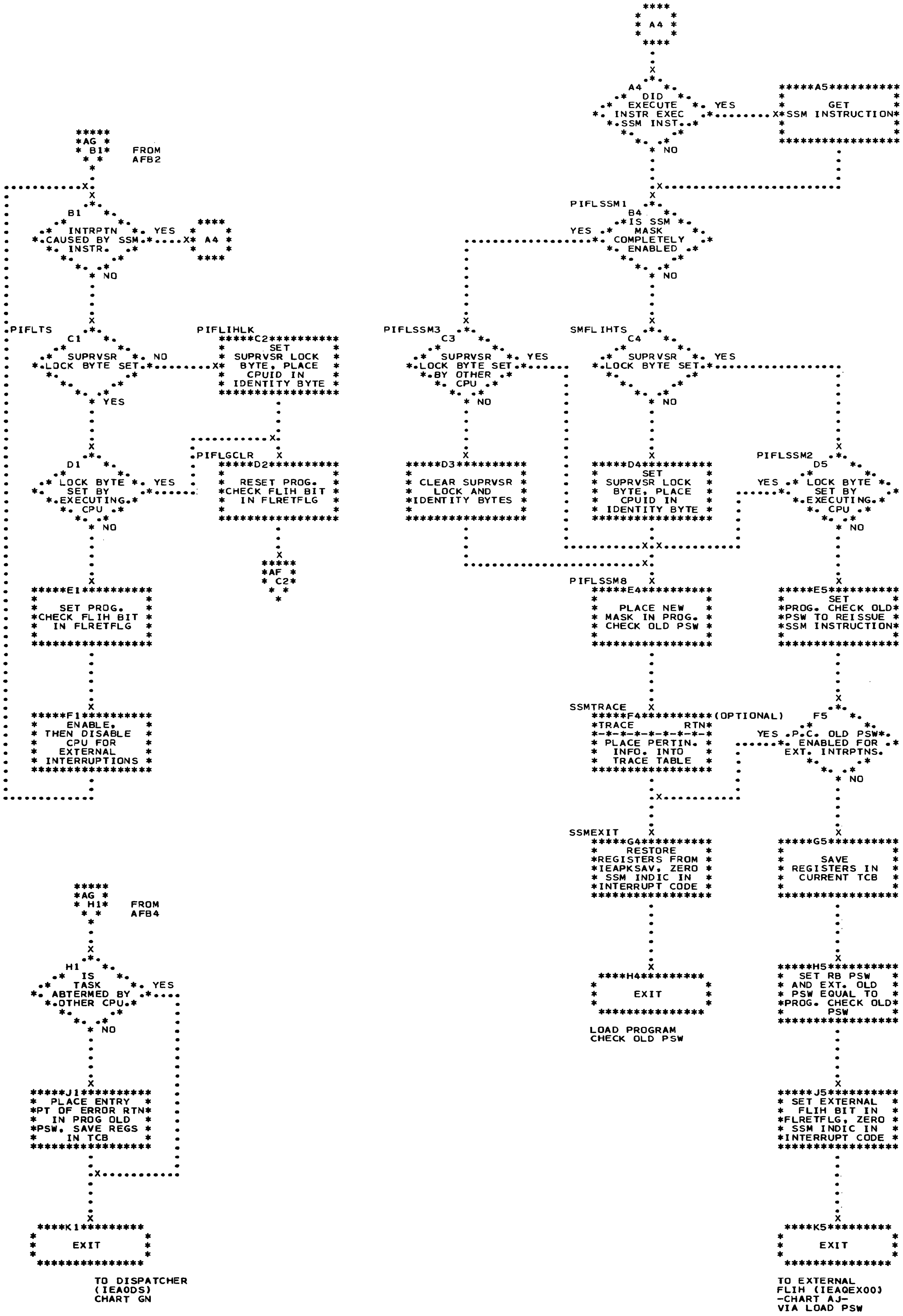
Chart AE. Transient Area Fetch Routine


TABLDL
A2 ENTRY
FROM DISPATCHER CHART GG J1 VIA LOAD PSW INSTRUCTION
B2 PREPARE FOR ENTRY TO BLDL RTN, USE TAB FOR CURRENT TCB AS WORK AREA
IECPBLDL
C2 BLDL RTN
GET NAME OF ENTRY POINT TO REQUESTED RTN
D2 ERROR FOUND BY BLDL RTN
YES
NO
D3 PERMANENT I/O ERROR
NO
YES
TBNOTFND
IEA0AB00
D4 ABTERM RTN HEA2
MODULE NOT FOUND, SCHEDULE TASK TERMIN
E2 PLACE TTR AND LENGTH OF REQUESTED RTN INTO USER'S SVRB
E4 INDICATE THAT TAB IS FREE SET TTR IN TACT ENTRY TO 0
K4
TAHFETCH
F1 ENTRY
FROM DISPATCHER (CHART GG J1) VIA LOAD PSW INSTRUCTION
F2 USE TTR AND LENGTH OF REQUESTED RTN AS INPUT TO PROG FETCH RTN
IEWFTRAN
G2 PROG FETCH CFB5
LOAD REQUESTED ROUTINE
H2 ERROR FOUND BY PROG. FETCH RTN
YES
NO
TBIOERR
H3 ANY TASK IN MUST COMPLETE STATUS
YES
NO
H4 LOAD WAIT STATE PSW
CODE=F10
TAHCHECK
J2 RESET TAB LOADING INDICATOR IN TACT ENTRY
J3 LOAD WAIT STATE PSW
CODE = F0F
IF OPERATOR PRESSES RESET AND START KEYS AFTER WAIT STATE PSW IS LOADED, REENTRY WILL OCCUR AT-
1. TABLDL IF ERROR WAS FOUND BY BLDL ROUTINE.
2. TAHFETCH IF ERROR WAS FOUND BY PROGRAM FETCH ROUTINE.
K2 PLACE USER SVRB'S FOR LOADED ROUTINE INTO READY STATUS
K3 REMOVE ALL SVRB'S FROM REQUEST QUEUE AND MAKE THEM READY
K4 PLACE TA FETCH TASK INTO WAIT STATE
K4
K5 EXIT
TO DISPATCHER (IEA0DS) -- CHART GG --

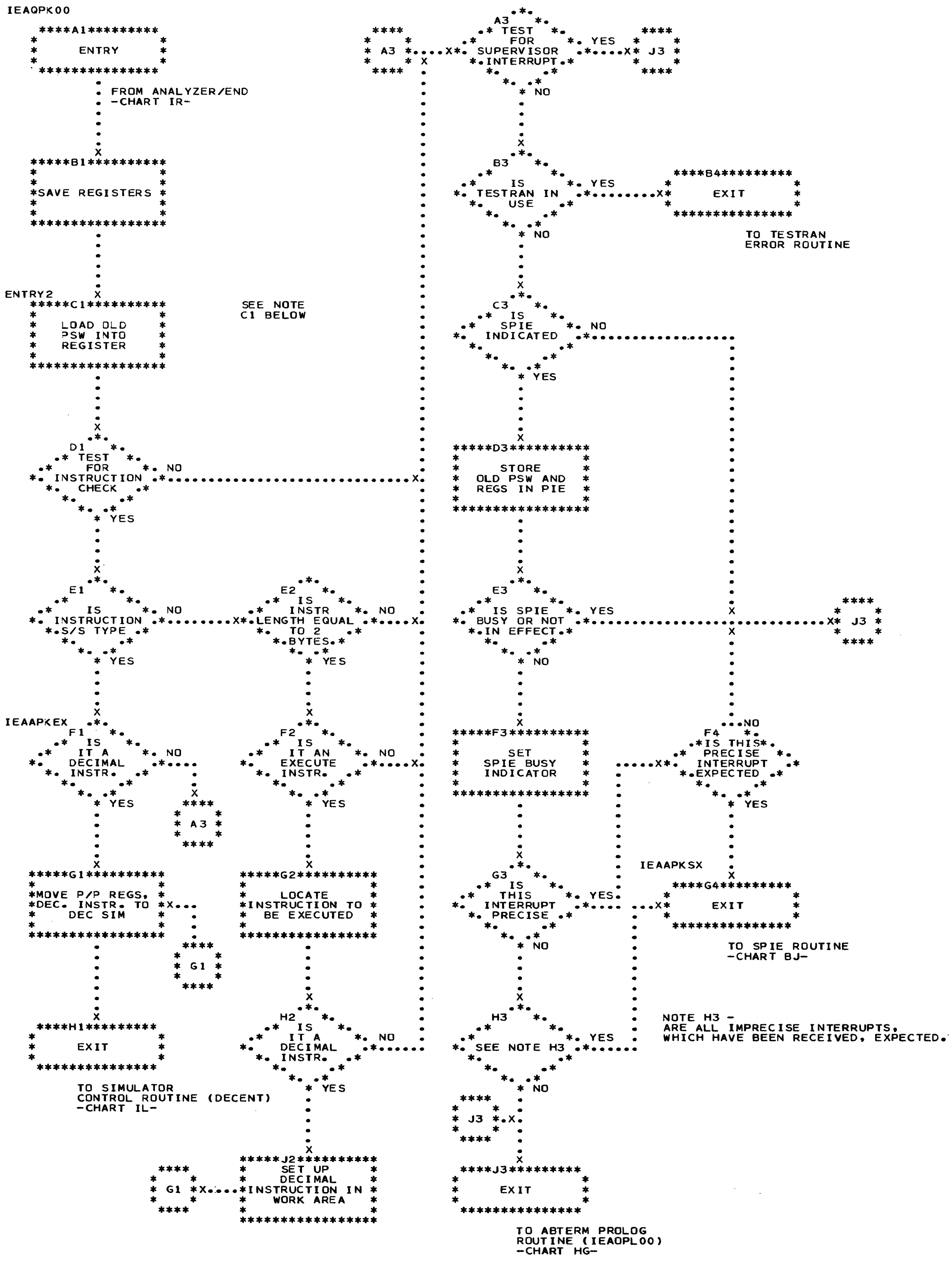
• Chart AF. Program Check First-Level Interruption Handler


IEAQPK00
A2 ENTRY
ENTERED FOLLOWING PROGRAM CHECK INTERRUPTION
B2 PLACE REGISTER CONTENTS INTO PROGRAM CHECK REGISTER SAVE AREA
AG B1
NOTE - THIS EXIT APPLIES ONLY IN MULTIPROCESSING SYSTEMS
TRPI
C2 TRACE RTN
PLACE PERTINENT INFO INTO TRACE TABLE
(OPTIONAL)
FROM AG D2
AF C2
D2 WAS SUPVR RTN INTERRUPTED
YES
NO
D3 EXIT
TO ABTERM PROLOG ROUTINE (IEAOPL00) -- CHART HG --
E2 DOES CURR TCB CONTAIN ADDR OF A PIE
NO
YES
E3 EXIT
TO ABTERM PROLOG ROUTINE (IEAOPL00) -- CHART HG --
F2 PLACE PROG OLD PSW AND REGS 2-14 INTO PROG INTERRUPTION ELEMENT PIE
G2 PLACE ADDR OF PIE INTO REGISTER 1
H2 IS PICA NOT IN EFFECT OR BUSY
YES
NO
H3 EXIT
TO ABTERM PROLOG ROUTINE (IEAOPL00) -- CHART HG --
J2 SET BUSY FLAG IN PROGRAM INTERRUPTION CONTROL AREA (PICA)
B4
B4
B4 CAN INTRPTN BE HANDLED
NO
YES
B5 EXIT
TO ABTERM PROLOG ROUTINE (IEAOPL00) -- CHART HG --
AG H1
NOTE - THIS EXIT APPLIES ONLY IN MULTI-PROCESSING SYSTEMS
C4 PLACE ENTRY PT OF INTERRUPTED ROUTINE INTO PROG OLD PSW
D4 PLACE RETURN ADDRESS INTO REGISTER 14, RESTORE REGISTERS
E4 EXIT
TO USER-WRITTEN ERROR-HANDLING ROUTINE

• **Chart AG. Program Check First-Level Interruption Handler (Multiprocessing System)**


AG B1 FROM AFB2
B1 INTRPTN CAUSED BY SSM INSTR. YES A4 NO
PIFLTS C1 SUPRVSR LOCK BYTE SET YES NO
PIFLIHLK C2 SET SUPRVSR LOCK BYTE, PLACE CPUID IN IDENTITY BYTE
D1 LOCK BYTE SET BY EXECUTING CPU YES NO
PIFLGCLR D2 RESET PROG. CHECK FLIH BIT IN FLRETFLG
AF C2
E1 SET PROG. CHECK FLIH BIT IN FLRETFLG
F1 ENABLE, THEN DISABLE CPU FOR EXTERNAL INTERRUPTIONS
AG H1 FROM AFB4
H1 IS TASK ABTERMED BY OTHER CPU YES NO
J1 PLACE ENTRY PT OF ERROR RTN IN PROG OLD PSW, SAVE REGS IN TCB
K1 EXIT
TO DISPATCHER (IEAODS) CHART GN
A4
A4 DID EXECUTE INSTR EXEC SSM INST. YES NO
A5 GET SSM INSTRUCTION
PIFLSSM1 B4 IS SSM MASK COMPLETELY ENABLED YES NO
PIFLSSM3 C3 SUPRVSR LOCK BYTE SET BY OTHER CPU YES NO
SMFLIHTS C4 SUPRVSR LOCK BYTE SET YES NO
D3 CLEAR SUPRVSR LOCK AND IDENTITY BYTES
D4 SET SUPRVSR LOCK BYTE, PLACE CPUID IN IDENTITY BYTE
PIFLSSM2 D5 LOCK BYTE SET BY EXECUTING CPU YES NO
PIFLSSM8 E4 PLACE NEW MASK IN PROG. CHECK OLD PSW
E5 SET PROG. CHECK OLD PSW TO REISSUE SSM INSTRUCTION
SSMTRACE F4 (OPTIONAL) TRACE RTN PLACE PERTIN. INFO. INTO TRACE TABLE
F5 P.C. OLD PSW ENABLED FOR EXT. INTRPTNS. YES NO
SSMEXIT G4 RESTORE REGISTERS FROM IEAPKSAV, ZERO SSM INDIC IN INTERRUPT CODE
G5 SAVE REGISTERS IN CURRENT TCB
H4 EXIT
LOAD PROGRAM CHECK OLD PSW
H5 SET RB PSW AND EXT. OLD PSW EQUAL TO PROG. CHECK OLD PSW
J5 SET EXTERNAL FLIH BIT IN FLRETFLG, ZERO SSM INDIC IN INTERRUPT CODE
K5 EXIT
TO EXTERNAL FLIH (IEAQEX00) -CHART AJ- VIA LOAD PSW

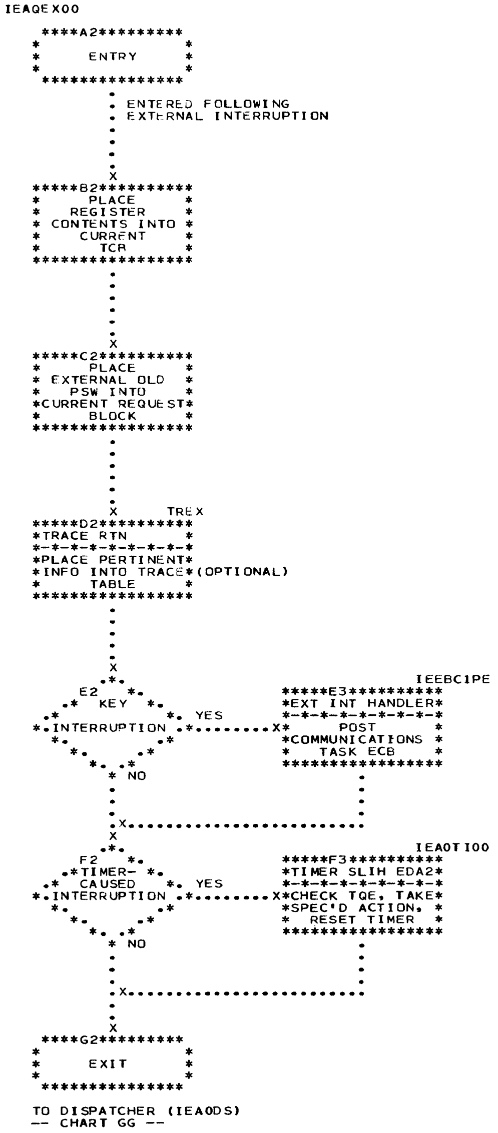
• Chart AH. Program Check First-Level Interruption Handler for Model 91


IEAQPK00
A1 ENTRY
FROM ANALYZER/END -CHART IR-
B1 SAVE REGISTERS
ENTRY2
C1 LOAD OLD PSW INTO REGISTER
SEE NOTE C1 BELOW
D1 TEST FOR INSTRUCTION CHECK
NO
YES
E1 IS INSTRUCTION S/S TYPE
NO
YES
IEAAPKEX
F1 IS IT A DECIMAL INSTR.
NO
YES
A3
G1 MOVE P/P REGS. DEC. INSTR. TO DEC SIM
G1
H1 EXIT
TO SIMULATOR CONTROL ROUTINE (DECENT) -CHART IL-
E2 IS INSTR LENGTH EQUAL TO 2 BYTES.
NO
YES
F2 IS IT AN EXECUTE INSTR.
NO
YES
G2 LOCATE INSTRUCTION TO BE EXECUTED
H2 IS IT A DECIMAL INSTR.
NO
YES
G1
J2 SET UP DECIMAL INSTRUCTION IN WORK AREA
A3
A3 TEST FOR SUPERVISOR INTERRUPT
YES
J3
NO
B3 IS TESTRAN IN USE
YES
B4 EXIT
TO TESTRAN ERROR ROUTINE
NO
C3 IS SPIE INDICATED
NO
YES
D3 STORE OLD PSW AND REGS IN PIE
E3 IS SPIE BUSY OR NOT IN EFFECT
YES
J3
NO
F3 SET SPIE BUSY INDICATOR
G3 IS THIS INTERRUPT PRECISE
YES
NO
H3 SEE NOTE H3
YES
NO
J3
J3 EXIT
TO ABTERM PROLOG ROUTINE (IEAOPL00) -CHART HG-
NO
F4 IS THIS PRECISE INTERRUPT EXPECTED
YES
IEAAPKSX
G4 EXIT
TO SPIE ROUTINE -CHART BJ-

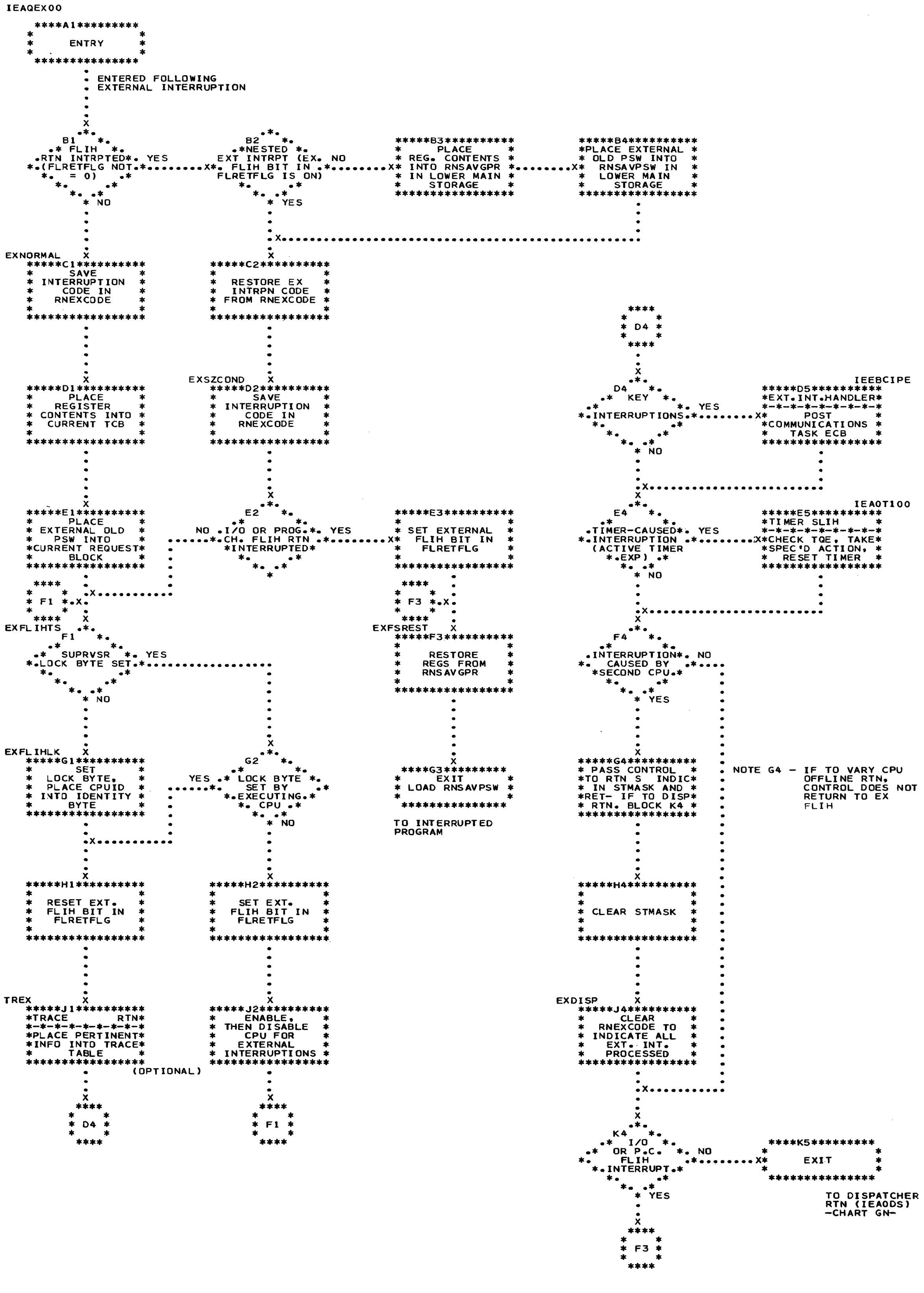
NOTE H3 - ARE ALL IMPRECISE INTERRUPTS, WHICH HAVE BEEN RECEIVED, EXPECTED.

NOTE C1 IF THE DECIMAL SIMULATOR IS NOT IN THE SYSTEM, BLOCKS D1-G1 AND E2-J2 ARE NOT TO BE USED (I.E., BLOCK C1 CONNECTS TO BLOCK A3). IF THE TESTRAN INTERPRETER IS NOT IN THE SYSTEM, BLOCK B3 AND A5 ARE NOT TO BE USED (I.E., BLOCK A3 CONNECTS TO BLOCK C3).

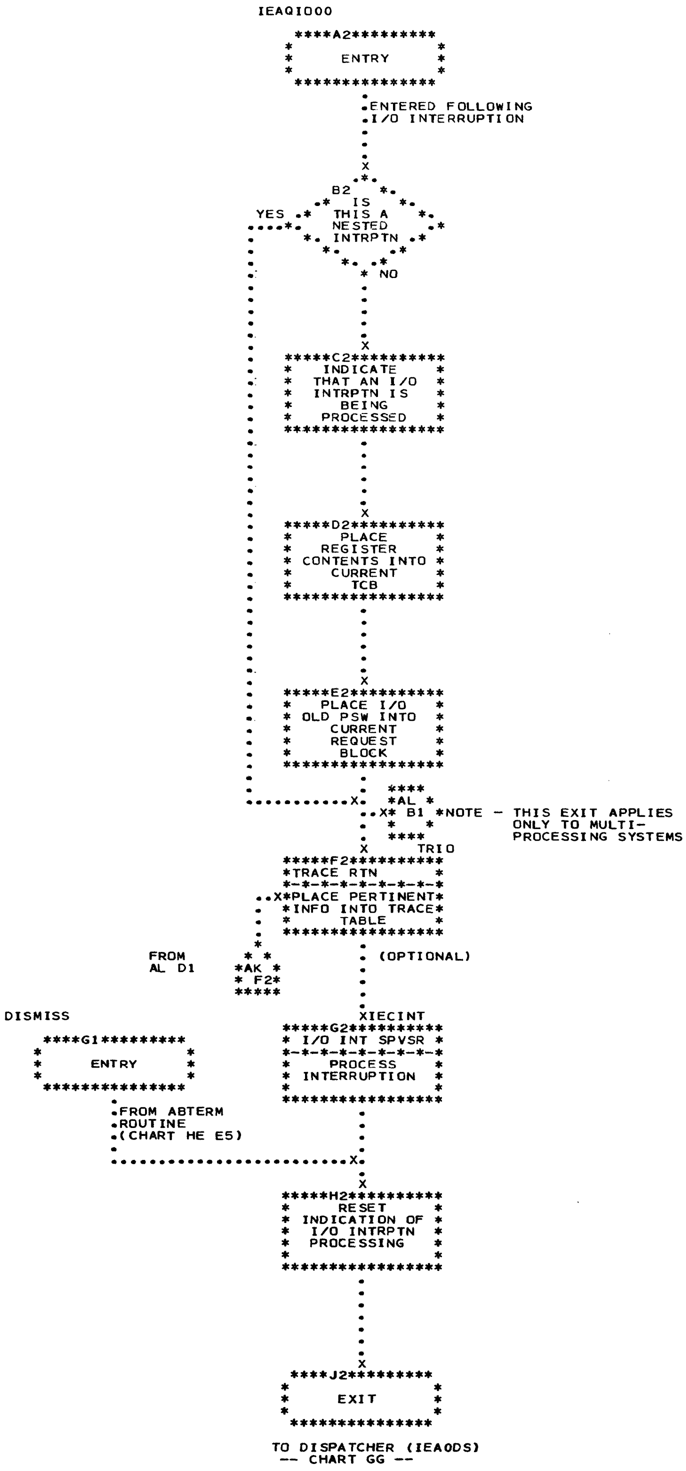
Chart AI. External First-Level Interruption Handler (Uniprocessing System)


IEAQEX00
A2
ENTRY
ENTERED FOLLOWING EXTERNAL INTERRUPTION
B2
PLACE REGISTER CONTENTS INTO CURRENT TCB
C2
PLACE EXTERNAL OLD PSW INTO CURRENT REQUEST BLOCK
TREX
D2
TRACE RTN
PLACE PERTINENT INFO INTO TRACE TABLE
(OPTIONAL)
E2
KEY INTERRUPTION
YES
NO
IEEBC1PE
E3
EXT INT HANDLER
POST COMMUNICATIONS TASK ECB
F2
TIMER-CAUSED INTERRUPTION
YES
NO
IEAOTI00
F3
TIMER SLIH EDA2
CHECK TQE, TAKE SPEC'D ACTION, RESET TIMER
G2
EXIT
TO DISPATCHER (IEAODS)
-- CHART GG --

• Chart AJ. External First-Level Interruption Handler (Multiprocessing System)


IEAQEX00
A1 ENTRY
ENTERED FOLLOWING EXTERNAL INTERRUPTION
B1 FLIH RTN INTRPTED (FLRETFLG NOT = 0)
YES
NO
B2 NESTED EXT INTRPT (EX. FLIH BIT IN FLRETFLG IS ON)
NO
YES
B3 PLACE REG. CONTENTS INTO RNSAVGPR IN LOWER MAIN STORAGE
B4 PLACE EXTERNAL OLD PSW INTO RNSAVPSW IN LOWER MAIN STORAGE
EXNORMAL
C1 SAVE INTERRUPTION CODE IN RNEXCODE
C2 RESTORE EX INTRPN CODE FROM RNEXCODE
D4
D1 PLACE REGISTER CONTENTS INTO CURRENT TCB
EXSZCOND
D2 SAVE INTERRUPTION CODE IN RNEXCODE
D4 KEY INTERRUPTIONS
YES
NO
IEEBCIPE
D5 EXT.INT.HANDLER POST COMMUNICATIONS TASK ECB
E1 PLACE EXTERNAL OLD PSW INTO CURRENT REQUEST BLOCK
E2 I/O OR PROG. CH. FLIH RTN INTERRUPTED
NO
YES
E3 SET EXTERNAL FLIH BIT IN FLRETFLG
E4 TIMER-CAUSED INTERRUPTION (ACTIVE TIMER EXP)
YES
NO
IEAOT100
E5 TIMER SLIH CHECK TQE. TAKE SPEC'D ACTION. RESET TIMER
F1
F3
EXFLIHTS
F1 SUPRVSR LOCK BYTE SET
YES
NO
EXFSREST
F3 RESTORE REGS FROM RNSAVGPR
F4 INTERRUPTION CAUSED BY SECOND CPU
NO
YES
EXFLIHLK
G1 SET LOCK BYTE. PLACE CPUID INTO IDENTITY BYTE
G2 LOCK BYTE SET BY EXECUTING CPU
YES
NO
G3 EXIT LOAD RNSAVPSW
TO INTERRUPTED PROGRAM
G4 PASS CONTROL TO RTN S INDIC IN STMASK AND RET- IF TO DISP RTN. BLOCK K4
NOTE G4 - IF TO VARY CPU OFFLINE RTN. CONTROL DOES NOT RETURN TO EX FLIH
H1 RESET EXT. FLIH BIT IN FLRETFLG
H2 SET EXT. FLIH BIT IN FLRETFLG
H4 CLEAR STMASK
TREX
J1 TRACE RTN PLACE PERTINENT INFO INTO TRACE TABLE
(OPTIONAL)
J2 ENABLE. THEN DISABLE CPU FOR EXTERNAL INTERRUPTIONS
EXDISP
J4 CLEAR RNEXCODE TO INDICATE ALL EXT. INT. PROCESSED
D4
F1
K4 I/O OR P.C. FLIH INTERRUPT
NO
YES
K5 EXIT
TO DISPATCHER RTN (IEAODS) -CHART GN-
F3

• Chart AK. I/O First-Level Interruption Handler


IEAQIOOO
A2
ENTRY
ENTERED FOLLOWING I/O INTERRUPTION
B2
IS THIS A NESTED INTRPTN
YES
NO
C2
INDICATE THAT AN I/O INTRPTN IS BEING PROCESSED
D2
PLACE REGISTER CONTENTS INTO CURRENT TCB
E2
PLACE I/O OLD PSW INTO CURRENT REQUEST BLOCK
AL B1
NOTE - THIS EXIT APPLIES ONLY TO MULTI-PROCESSING SYSTEMS
TRIO
F2
TRACE RTN
PLACE PERTINENT INFO INTO TRACE TABLE
FROM AL D1
AK F2
(OPTIONAL)
DISMISS
G1
ENTRY
FROM ABTERM ROUTINE (CHART HE E5)
XIECINT
G2
I/O INT SPVSR
PROCESS INTERRUPTION
H2
RESET INDICATION OF I/O INTRPTN PROCESSING
J2
EXIT
TO DISPATCHER (IEAODS) -- CHART GG --

• Chart AL. I/O First-Level Interruption Handler (Multiprocessing System)

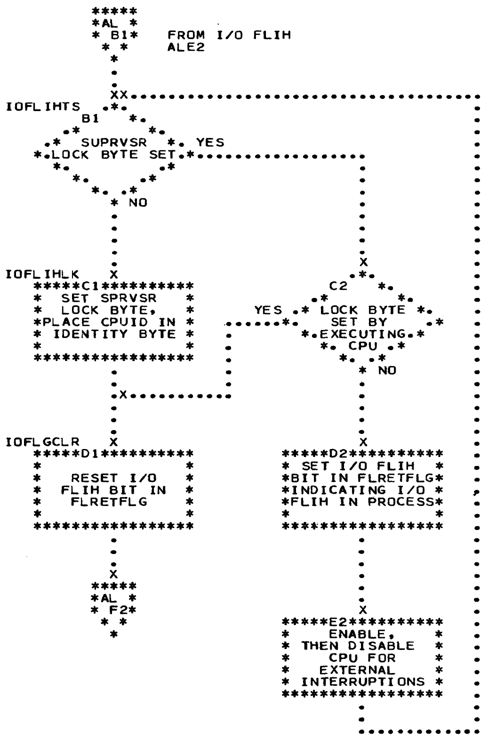

AL B1
FROM I/O FLIH ALE2
IOFLIHTS
B1
SUPRVSR LOCK BYTE SET
YES
NO
IOFLIHLK
C1
SET SPRVSR LOCK BYTE, PLACE CPUID IN IDENTITY BYTE
C2
LOCK BYTE SET BY EXECUTING CPU
YES
NO
IOFLGCLR
D1
RESET I/O FLIH BIT IN FLRETFLG
D2
SET I/O FLIH BIT IN FLRETFLG INDICATING I/O FLIH IN PROCESS
AL F2
E2
ENABLE, THEN DISABLE CPU FOR EXTERNAL INTERRUPTIONS

Chart AM. SER0 Routine

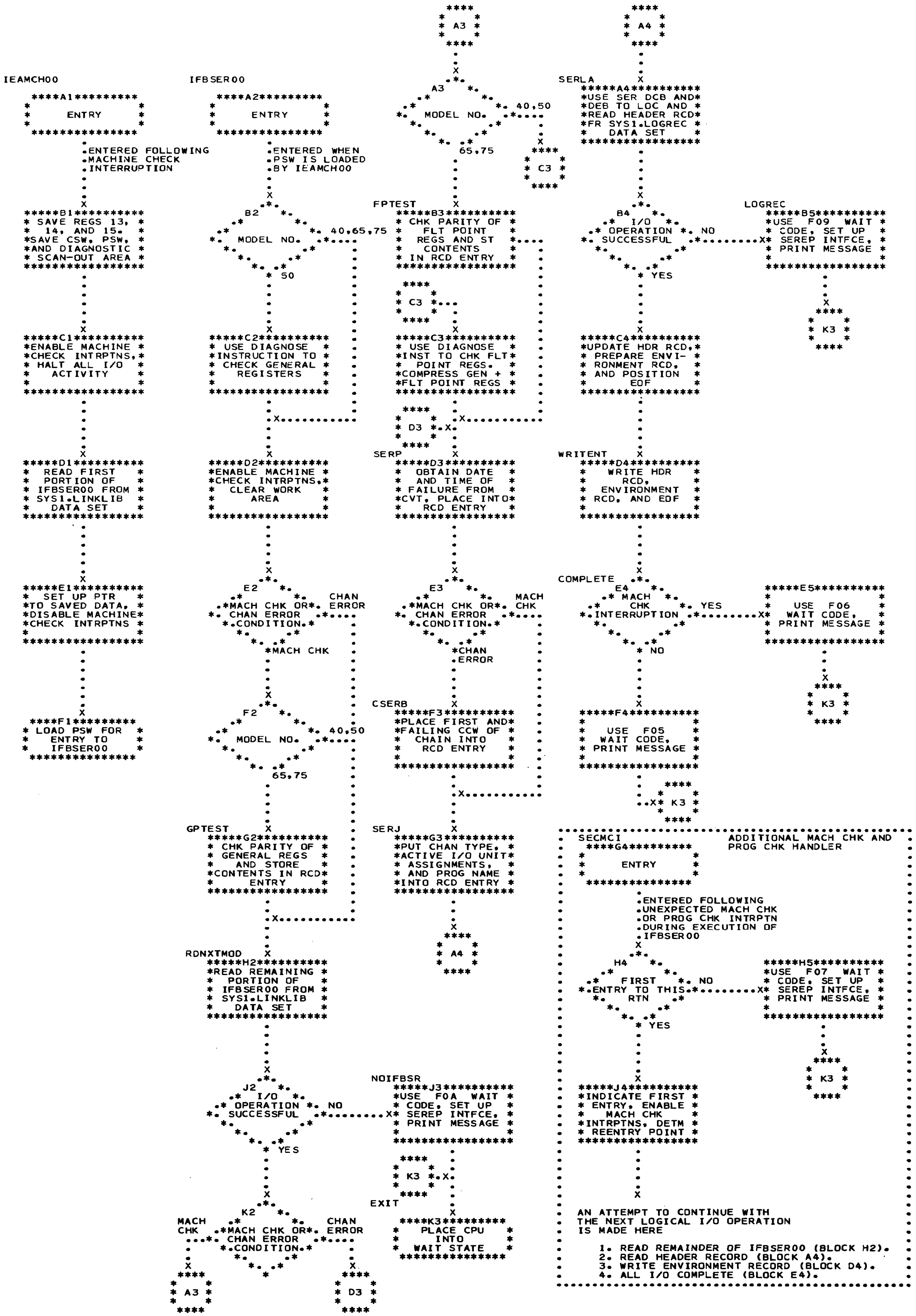
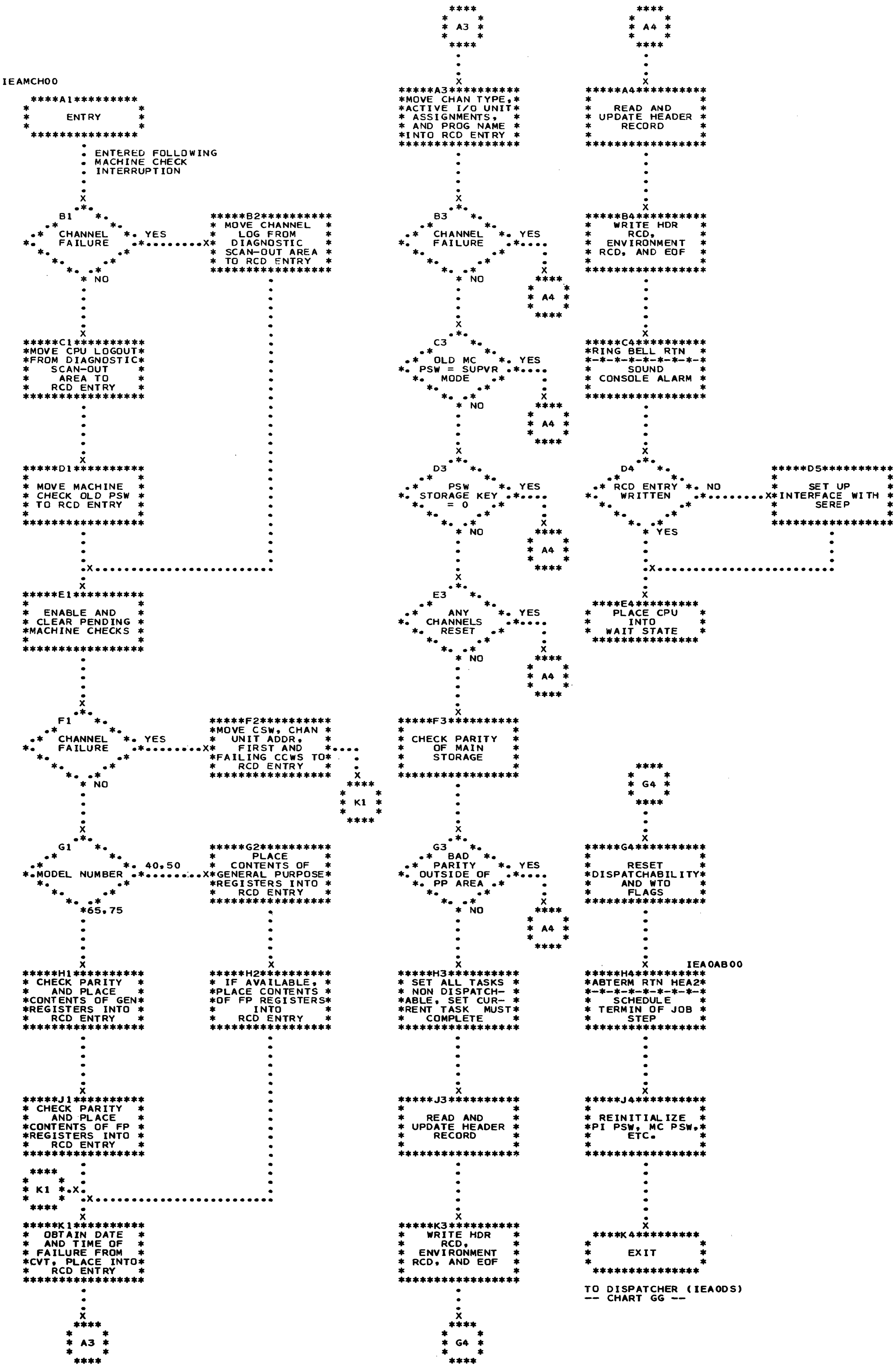
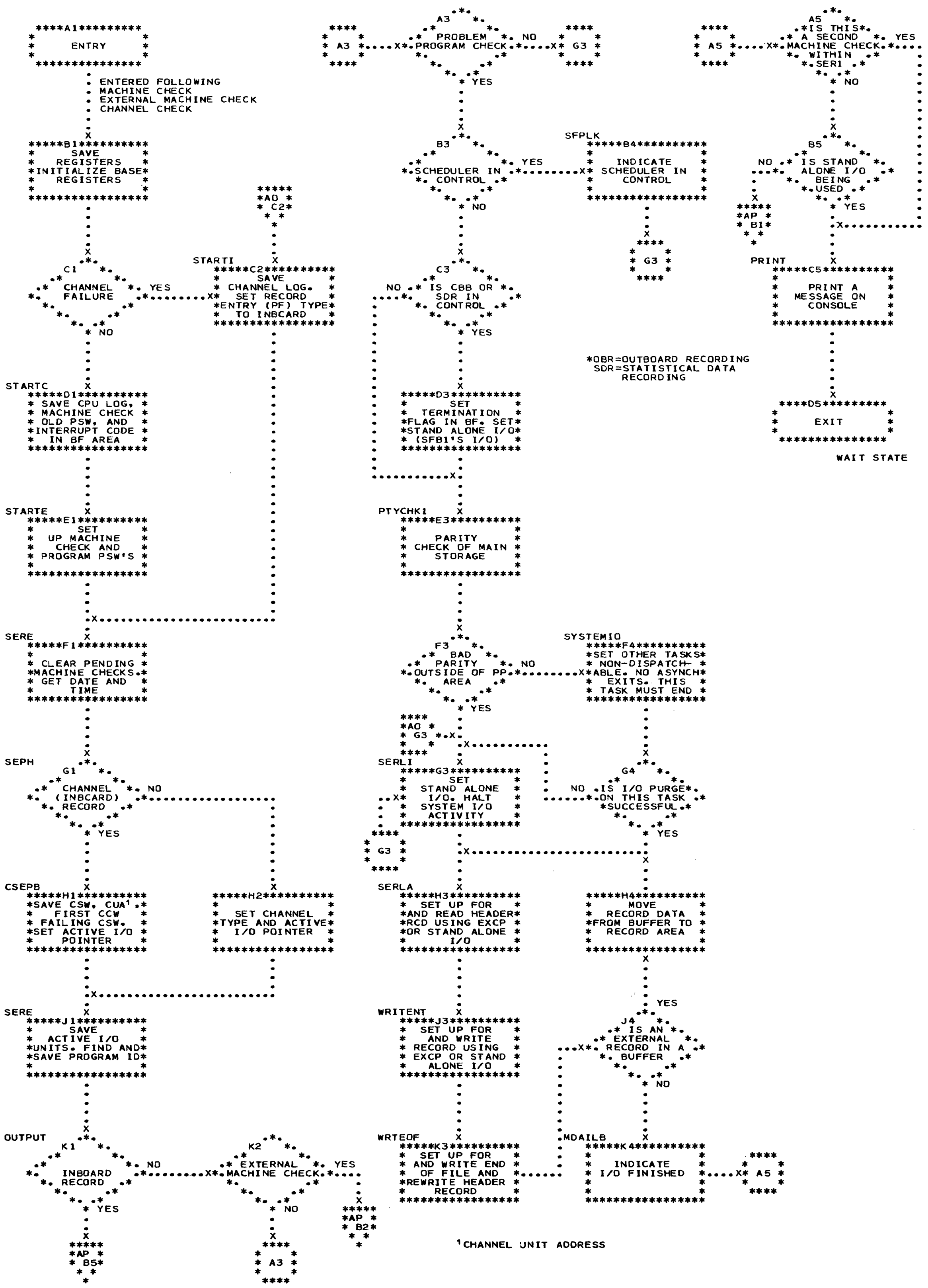

IEAMCH00
A1 ENTRY
ENTERED FOLLOWING MACHINE CHECK INTERRUPTION
B1 SAVE REGS 13, 14, AND 15. SAVE CSW, PSW, AND DIAGNOSTIC SCAN-OUT AREA
C1 ENABLE MACHINE CHECK INTRPTNS, HALT ALL I/O ACTIVITY
D1 READ FIRST PORTION OF IFBSER00 FROM SYS1.LINKLIB DATA SET
E1 SET UP PTR TO SAVED DATA, DISABLE MACHINE CHECK INTRPTNS
F1 LOAD PSW FOR ENTRY TO IFBSER00
IFBSER00
A2 ENTRY
ENTERED WHEN PSW IS LOADED BY IEAMCH00
B2 MODEL NO.
40,65,75
50
C2 USE DIAGNOSE INSTRUCTION TO CHECK GENERAL REGISTERS
D2 ENABLE MACHINE CHECK INTRPTNS, CLEAR WORK AREA
E2 MACH CHK OR CHAN ERROR CONDITION
CHAN ERROR
MACH CHK
F2 MODEL NO.
40,50
65,75
GPTEST
G2 CHK PARITY OF GENERAL REGS AND STORE CONTENTS IN RCD ENTRY
RDNXTMOD
H2 READ REMAINING PORTION OF IFBSER00 FROM SYS1.LINKLIB DATA SET
J2 I/O OPERATION SUCCESSFUL
NO
YES
K2 MACH CHK OR CHAN ERROR CONDITION
MACH CHK
CHAN ERROR
A3
D3
A3
A3 MODEL NO.
40,50
65,75
C3
FPTEST
B3 CHK PARITY OF FLT POINT REGS AND ST CONTENTS IN RCD ENTRY
C3
C3 USE DIAGNOSE INST TO CHK FLT POINT REGS. COMPRESS GEN + FLT POINT REGS
D3
SERP
D3 OBTAIN DATE AND TIME OF FAILURE FROM CVT, PLACE INTO RCD ENTRY
E3 MACH CHK OR CHAN ERROR CONDITION
MACH CHK
CHAN ERROR
CSERB
F3 PLACE FIRST AND FAILING CCW OF CHAIN INTO RCD ENTRY
SERJ
G3 PUT CHAN TYPE, ACTIVE I/O UNIT ASSIGNMENTS, AND PROG NAME INTO RCD ENTRY
A4
NOIFBSR
J3 USE F0A WAIT CODE, SET UP SEREP INTFCE, PRINT MESSAGE
K3
EXIT
K3 PLACE CPU INTO WAIT STATE
A4
SERLA
A4 USE SER DCB AND DEB TO LOC AND READ HEADER RCD FR SYS1.LOGREC DATA SET
B4 I/O OPERATION SUCCESSFUL
NO
YES
LOGREC
B5 USE F09 WAIT CODE, SET UP SEREP INTFCE, PRINT MESSAGE
K3
C4 UPDATE HDR RCD, PREPARE ENVIRONMENT RCD, AND POSITION EOF
WRITENT
D4 WRITE HDR RCD, ENVIRONMENT RCD, AND EOF
COMPLETE
E4 MACH CHK INTERRUPTION
YES
NO
E5 USE F06 WAIT CODE, PRINT MESSAGE
K3
F4 USE F05 WAIT CODE, PRINT MESSAGE
K3
SECMCI
ADDITIONAL MACH CHK AND PROG CHK HANDLER
G4 ENTRY
ENTERED FOLLOWING UNEXPECTED MACH CHK OR PROG CHK INTRPTN DURING EXECUTION OF IFBSER00
H4 FIRST ENTRY TO THIS RTN
NO
YES
H5 USE F07 WAIT CODE, SET UP SEREP INTFCE, PRINT MESSAGE
K3
J4 INDICATE FIRST ENTRY, ENABLE MACH CHK INTRPTNS, DETM REENTRY POINT
AN ATTEMPT TO CONTINUE WITH THE NEXT LOGICAL I/O OPERATION IS MADE HERE
1. READ REMAINDER OF IFBSER00 (BLOCK H2).
2. READ HEADER RECORD (BLOCK A4).
3. WRITE ENVIRONMENT RECORD (BLOCK D4).
4. ALL I/O COMPLETE (BLOCK E4).

Chart AN. SER1 Routine (for Models 40, 50, 60, 75)


A3
A4
IEAMCH00
A1
ENTRY
ENTERED FOLLOWING MACHINE CHECK INTERRUPTION
B1
CHANNEL FAILURE
YES
NO
B2
MOVE CHANNEL LOG FROM DIAGNOSTIC SCAN-OUT AREA TO RCD ENTRY
C1
MOVE CPU LOGOUT FROM DIAGNOSTIC SCAN-OUT AREA TO RCD ENTRY
D1
MOVE MACHINE CHECK OLD PSW TO RCD ENTRY
E1
ENABLE AND CLEAR PENDING MACHINE CHECKS
F1
CHANNEL FAILURE
YES
NO
F2
MOVE CSW, CHAN UNIT ADDR, FIRST AND FAILING CCWS TO RCD ENTRY
K1
G1
MODEL NUMBER
40,50
65,75
G2
PLACE CONTENTS OF GENERAL PURPOSE REGISTERS INTO RCD ENTRY
H1
CHECK PARITY AND PLACE CONTENTS OF GEN REGISTERS INTO RCD ENTRY
H2
IF AVAILABLE, PLACE CONTENTS OF FP REGISTERS INTO RCD ENTRY
J1
CHECK PARITY AND PLACE CONTENTS OF FP REGISTERS INTO RCD ENTRY
K1
K1
OBTAIN DATE AND TIME OF FAILURE FROM CVT, PLACE INTO RCD ENTRY
A3
A3
MOVE CHAN TYPE, ACTIVE I/O UNIT ASSIGNMENTS, AND PROG NAME INTO RCD ENTRY
B3
CHANNEL FAILURE
YES
A4
NO
C3
OLD MC PSW = SUPVR MODE
YES
A4
NO
D3
PSW STORAGE KEY = 0
YES
A4
NO
E3
ANY CHANNELS RESET
YES
A4
NO
F3
CHECK PARITY OF MAIN STORAGE
G3
BAD PARITY OUTSIDE OF PP AREA
YES
A4
NO
H3
SET ALL TASKS NON DISPATCHABLE, SET CURRENT TASK MUST COMPLETE
J3
READ AND UPDATE HEADER RECORD
K3
WRITE HDR RCD, ENVIRONMENT RCD, AND EOF
G4
A4
READ AND UPDATE HEADER RECORD
B4
WRITE HDR RCD, ENVIRONMENT RCD, AND EOF
C4
RING BELL RTN
SOUND CONSOLE ALARM
D4
RCD ENTRY WRITTEN
NO
YES
D5
SET UP INTERFACE WITH SEREP
E4
PLACE CPU INTO WAIT STATE
G4
G4
RESET DISPATCHABILITY AND WTO FLAGS
IEA0AB00
H4
ABTERM RTN HEA2
SCHEDULE TERMIN OF JOB STEP
J4
REINITIALIZE PI PSW, MC PSW, ETC.
K4
EXIT
TO DISPATCHER (IEA0DS) -- CHART GG --

• Chart AO. SER1 Routine (for Model 91; Part 1 of 2)


A1 ENTRY
ENTERED FOLLOWING MACHINE CHECK EXTERNAL MACHINE CHECK CHANNEL CHECK
B1 SAVE REGISTERS INITIALIZE BASE REGISTERS
C1 CHANNEL FAILURE
YES
NO
AO C2
STARTI
C2 SAVE CHANNEL LOG. SET RECORD ENTRY (PF) TYPE TO INBCARD
STARTC
D1 SAVE CPU LOG, MACHINE CHECK OLD PSW, AND INTERRUPT CODE IN BF AREA
STARTE
E1 SET UP MACHINE CHECK AND PROGRAM PSW'S
SERE
F1 CLEAR PENDING MACHINE CHECKS. GET DATE AND TIME
SEPH
G1 CHANNEL (INBCARD) RECORD
NO
YES
CSEPB
H1 SAVE CSW, CUA¹, FIRST CCW FAILING CSW. SET ACTIVE I/O POINTER
H2 SET CHANNEL TYPE AND ACTIVE I/O POINTER
SERE
J1 SAVE ACTIVE I/O UNITS. FIND AND SAVE PROGRAM ID
OUTPUT
K1 INBOARD RECORD
NO
YES
AP B5
K2 EXTERNAL MACHINE CHECK
YES
NO
AP B2
A3
A3
A3 PROBLEM PROGRAM CHECK
NO
G3
YES
B3 SCHEDULER IN CONTROL
YES
NO
SFPLK
B4 INDICATE SCHEDULER IN CONTROL
G3
C3 IS CBB OR SDR IN CONTROL
NO
YES
D3 SET TERMINATION FLAG IN BF. SET STAND ALONE I/O (SFB1'S I/O)
*OBR=OUTBOARD RECORDING SDR=STATISTICAL DATA RECORDING
PTYCHK1
E3 PARITY CHECK OF MAIN STORAGE
F3 BAD PARITY OUTSIDE OF PP AREA
NO
YES
SYSTEMIO
F4 SET OTHER TASKS NON-DISPATCH- ABLE. NO ASYNCH EXITS. THIS TASK MUST END
AO G3
SERLI
G3 SET STAND ALONE I/O. HALT SYSTEM I/O ACTIVITY
G3
G4 IS I/O PURGE ON THIS TASK SUCCESSFUL
NO
YES
SERLA
H3 SET UP FOR AND READ HEADER RCD USING EXCP OR STAND ALONE I/O
H4 MOVE RECORD DATA FROM BUFFER TO RECORD AREA
WRITENT
J3 SET UP FOR AND WRITE RECORD USING EXCP OR STAND ALONE I/O
J4 IS AN EXTERNAL RECORD IN A BUFFER
YES
NO
WRTEOF
K3 SET UP FOR AND WRITE END OF FILE AND REWRITE HEADER RECORD
MDAILB
K4 INDICATE I/O FINISHED
A5
A5
A5 IS THIS A SECOND MACHINE CHECK WITHIN SER1
YES
NO
B5 IS STAND ALONE I/O BEING USED
NO
YES
AP B1
PRINT
C5 PRINT A MESSAGE ON CONSOLE
D5 EXIT
WAIT STATE

¹CHANNEL UNIT ADDRESS

• Chart AP. SER1 Routine (for Model 91; Part 2 of 2)

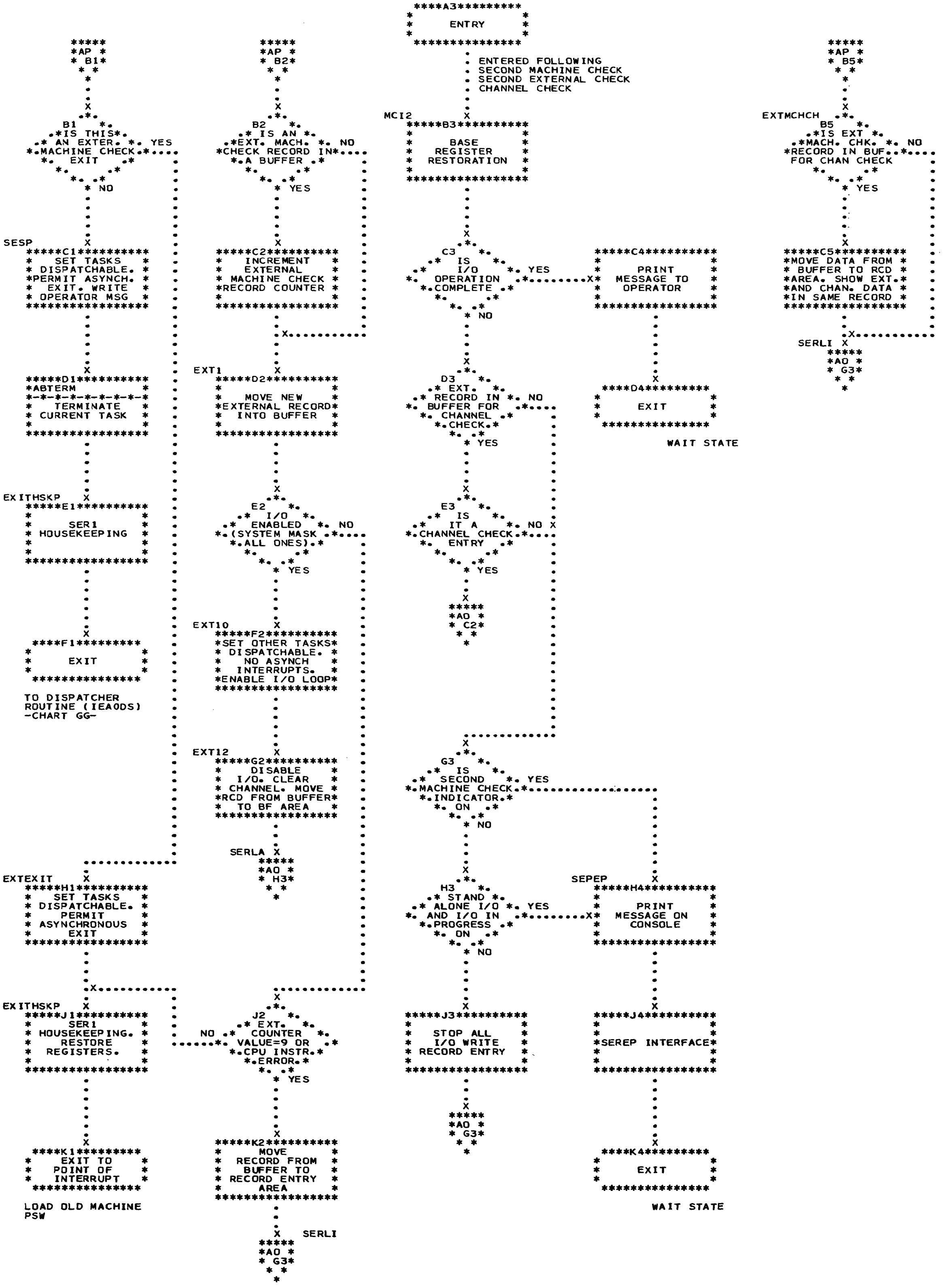
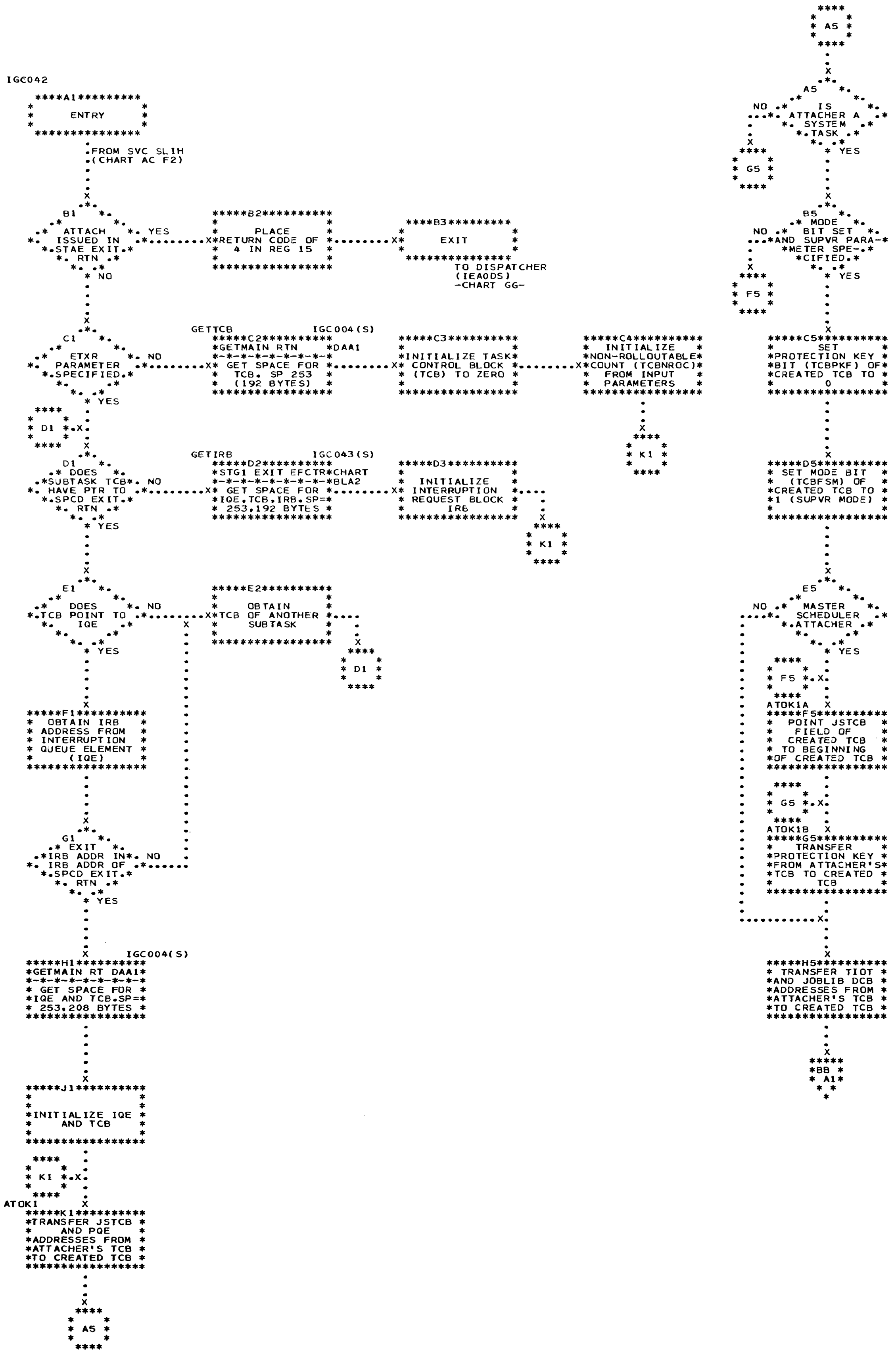
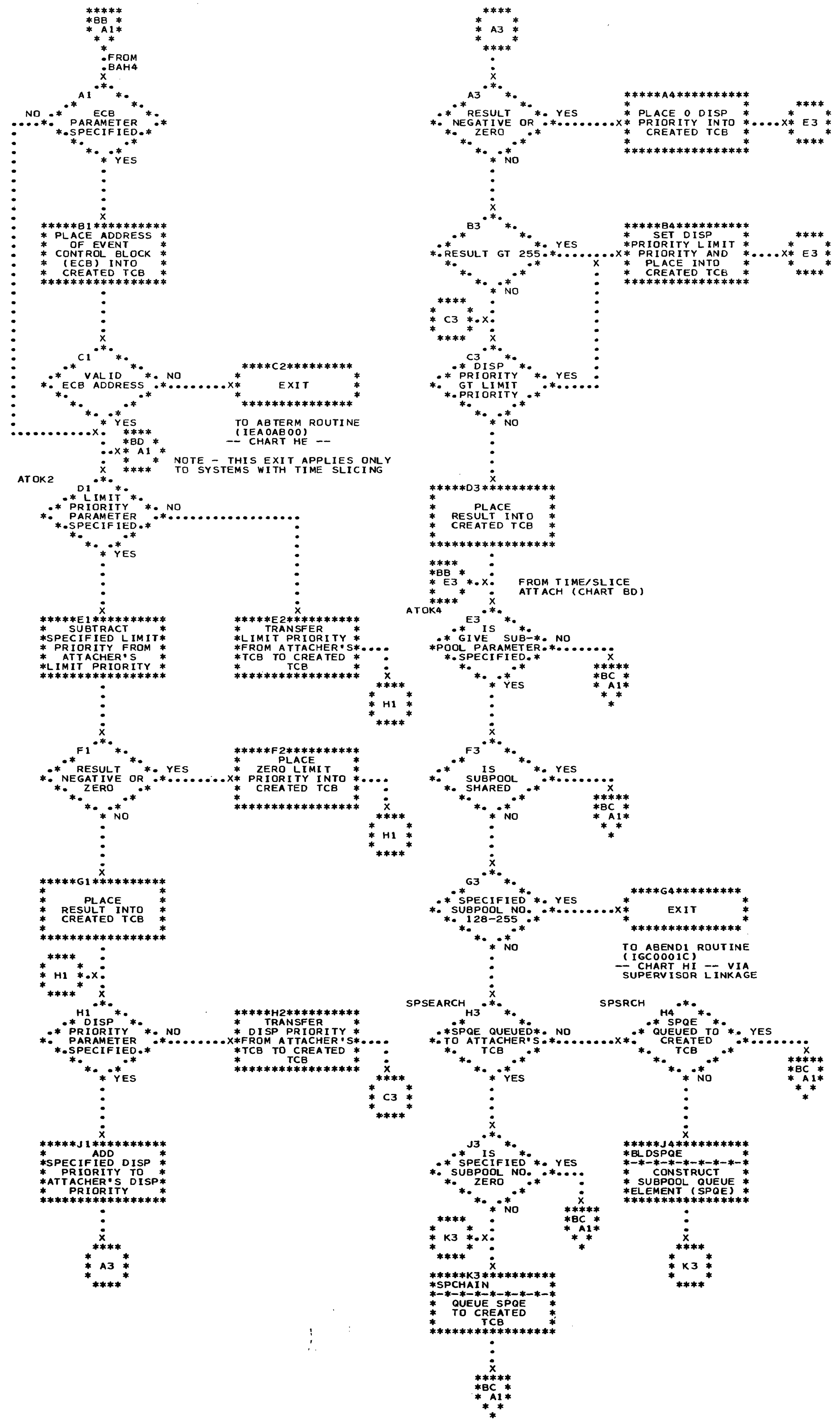

A3 ENTRY
ENTERED FOLLOWING SECOND MACHINE CHECK SECOND EXTERNAL CHECK CHANNEL CHECK
AP B1
B1 IS THIS AN EXTER. MACHINE CHECK EXIT
YES
NO
SESP
C1 SET TASKS DISPATCHABLE. PERMIT ASYNCH. EXIT. WRITE OPERATOR MSG
D1 ABTERM TERMINATE CURRENT TASK
EXITHSKP
E1 SER1 HOUSEKEEPING
F1 EXIT
TO DISPATCHER ROUTINE (IEAODS) -CHART GG-
EXTEXIT
H1 SET TASKS DISPATCHABLE. PERMIT ASYNCHRONOUS EXIT
EXITHSKP
J1 SER1 HOUSEKEEPING. RESTORE REGISTERS.
K1 EXIT TO POINT OF INTERRUPT
LOAD OLD MACHINE PSW
AP B2
B2 IS AN EXT. MACH. CHECK RECORD IN A BUFFER
NO
YES
C2 INCREMENT EXTERNAL MACHINE CHECK RECORD COUNTER
EXT1
D2 MOVE NEW EXTERNAL RECORD INTO BUFFER
E2 I/O ENABLED (SYSTEM MASK ALL ONES)
NO
YES
EXT10
F2 SET OTHER TASKS DISPATCHABLE. NO ASYNCH INTERRUPTS. ENABLE I/O LOOP
EXT12
G2 DISABLE I/O. CLEAR CHANNEL. MOVE RCD FROM BUFFER TO BF AREA
SERLA
AO H3
J2 EXT. COUNTER VALUE=9 OR CPU INSTR. ERROR.
NO
YES
K2 MOVE RECORD FROM BUFFER TO RECORD ENTRY AREA
SERLI
AO G3
MCI2
B3 BASE REGISTER RESTORATION
C3 IS I/O OPERATION COMPLETE
YES
NO
C4 PRINT MESSAGE TO OPERATOR
D4 EXIT
WAIT STATE
D3 EXT. RECORD IN BUFFER FOR CHANNEL CHECK.
NO
YES
E3 IS IT A CHANNEL CHECK ENTRY
NO
YES
AO C2
G3 IS SECOND MACHINE CHECK INDICATOR ON
YES
NO
H3 STAND ALONE I/O AND I/O IN PROGRESS ON
YES
NO
SEPEP
H4 PRINT MESSAGE ON CONSOLE
J3 STOP ALL I/O WRITE RECORD ENTRY
AO G3
J4 SEREP INTERFACE
K4 EXIT
WAIT STATE
AP B5
EXTMCHCH
B5 IS EXT MACH. CHK. RECORD IN BUF. FOR CHAN CHECK
NO
YES
C5 MOVE DATA FROM BUFFER TO RCD AREA. SHOW EXT. AND CHAN. DATA IN SAME RECORD
SERLI
AO G3

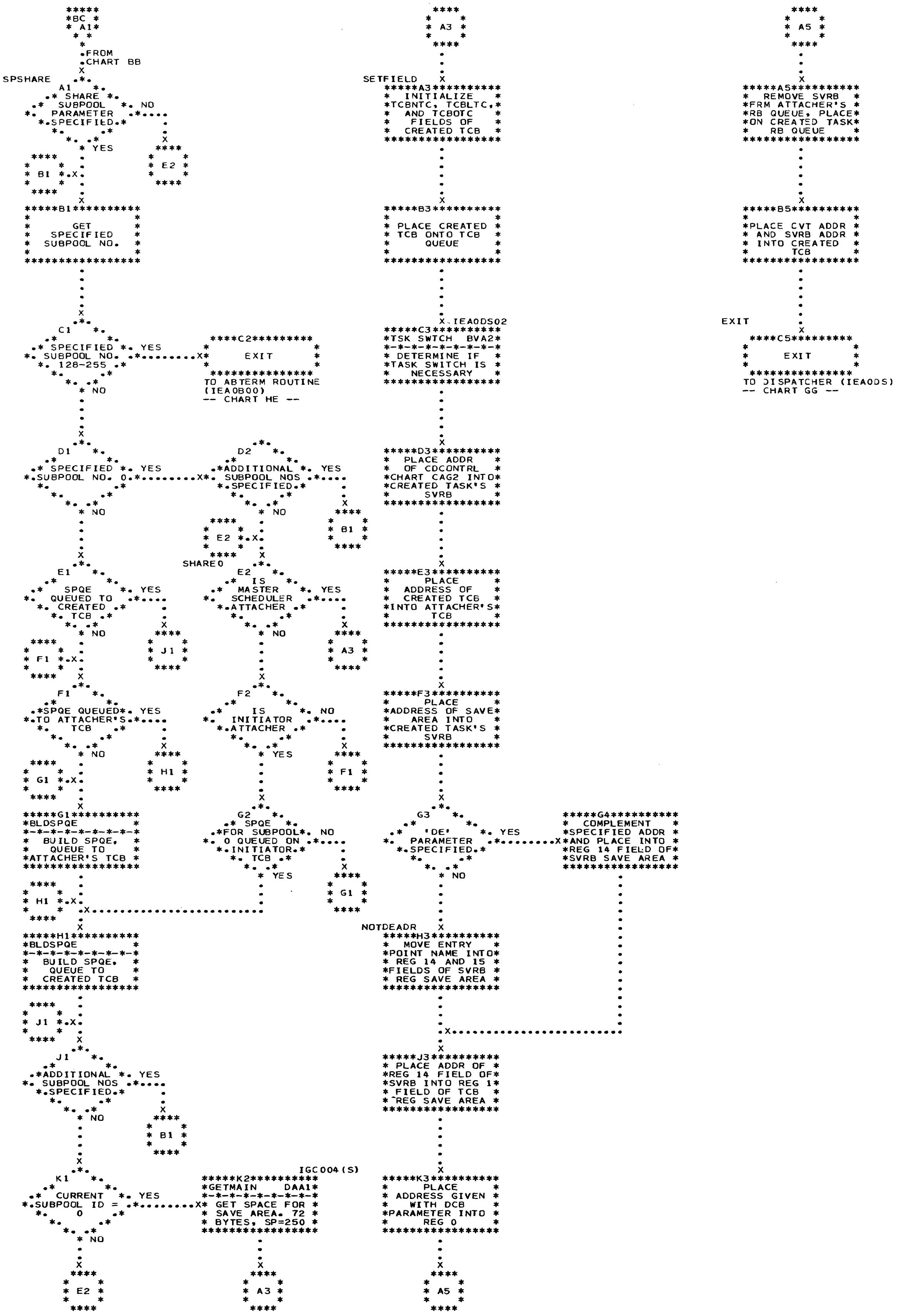
Chart BA. Attach Routine (Part 1 of 3)


IGC042
A1 ENTRY
FROM SVC SLIH (CHART AC F2)
B1 ATTACH ISSUED IN STAE EXIT RTN
YES
NO
B2 PLACE RETURN CODE OF 4 IN REG 15
B3 EXIT
TO DISPATCHER (IEAODS) -CHART GG-
C1 ETXR PARAMETER SPECIFIED
GETTCB IGC004(S)
C2 GETMAIN RTN DAA1 GET SPACE FOR TCB. SP 253 (192 BYTES)
C3 INITIALIZE TASK CONTROL BLOCK (TCB) TO ZERO
C4 INITIALIZE NON-ROLLOUTABLE COUNT (TCBNROC) FROM INPUT PARAMETERS
K1
D1
D1 DOES SUBTASK TCB HAVE PTR TO SPCD EXIT RTN
GETIRB IGC043(S)
D2 STG1 EXIT EFCTR CHART BLA2 GET SPACE FOR IQE,TCB,IRB.SP= 253,192 BYTES
D3 INITIALIZE INTERRUPTION REQUEST BLOCK IRB
K1
E1 DOES TCB POINT TO IQE
E2 OBTAIN TCB OF ANOTHER SUBTASK
D1
F1 OBTAIN IRB ADDRESS FROM INTERRUPTION QUEUE ELEMENT (IQE)
G1 EXIT IRB ADDR IN IRB ADDR OF SPCD EXIT RTN
IGC004(S)
H1 GETMAIN RT DAA1 GET SPACE FOR IQE AND TCB.SP= 253,208 BYTES
J1 INITIALIZE IQE AND TCB
K1
ATOK1
K1 TRANSFER JSTCB AND PQE ADDRESSES FROM ATTACHER'S TCB TO CREATED TCB
A5
A5
A5 IS ATTACHER A SYSTEM TASK
G5
B5 MODE BIT SET AND SUPVR PARAMETER SPECIFIED
F5
C5 SET PROTECTION KEY BIT (TCBPKF) OF CREATED TCB TO 0
D5 SET MODE BIT (TCBFSM) OF CREATED TCB TO 1 (SUPVR MODE)
E5 MASTER SCHEDULER ATTACHER
F5
ATOK1A
F5 POINT JSTCB FIELD OF CREATED TCB TO BEGINNING OF CREATED TCB
G5
ATOK1B
G5 TRANSFER PROTECTION KEY FROM ATTACHER'S TCB TO CREATED TCB
H5 TRANSFER TIOT AND JOBLIB DCB ADDRESSES FROM ATTACHER'S TCB TO CREATED TCB
BB A1

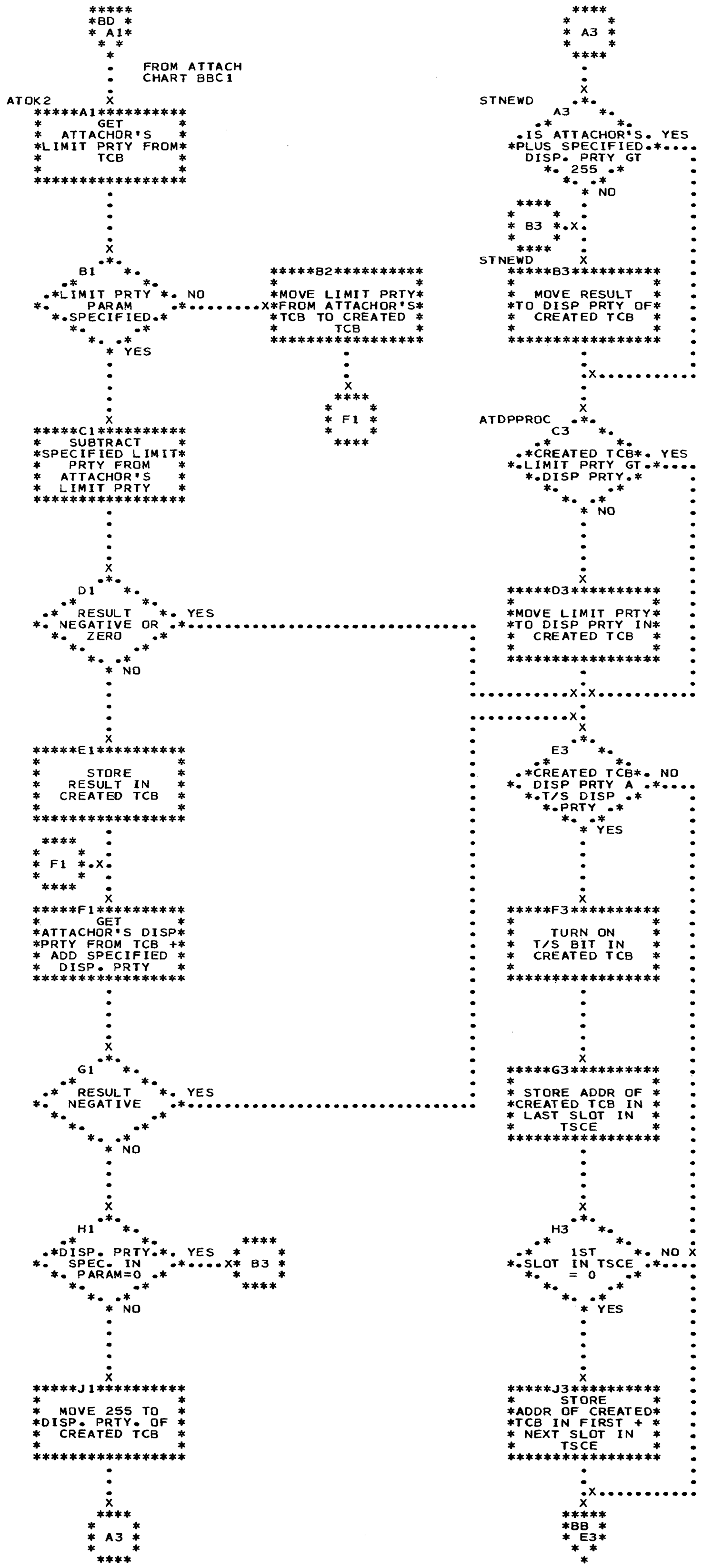
• Chart BB. Attach Routine (Part 2 of 3)


BB A1
FROM BAH4
A1
ECB PARAMETER SPECIFIED
NO
YES
B1
PLACE ADDRESS OF EVENT CONTROL BLOCK (ECB) INTO CREATED TCB
C1
VALID ECB ADDRESS
NO
C2
EXIT
TO ABTERM ROUTINE (IEA0AB00) -- CHART HE --
YES
BD A1
NOTE - THIS EXIT APPLIES ONLY TO SYSTEMS WITH TIME SLICING
ATOK2
D1
LIMIT PRIORITY PARAMETER SPECIFIED
NO
YES
E1
SUBTRACT SPECIFIED LIMIT PRIORITY FROM ATTACHER'S LIMIT PRIORITY
E2
TRANSFER LIMIT PRIORITY FROM ATTACHER'S TCB TO CREATED TCB
H1
F1
RESULT NEGATIVE OR ZERO
YES
NO
F2
PLACE ZERO LIMIT PRIORITY INTO CREATED TCB
H1
G1
PLACE RESULT INTO CREATED TCB
H1
H1
DISP PRIORITY PARAMETER SPECIFIED
NO
YES
H2
TRANSFER DISP PRIORITY FROM ATTACHER'S TCB TO CREATED TCB
C3
J1
ADD SPECIFIED DISP PRIORITY TO ATTACHER'S DISP PRIORITY
A3
A3
A3
RESULT NEGATIVE OR ZERO
YES
NO
A4
PLACE 0 DISP PRIORITY INTO CREATED TCB
E3
B3
RESULT GT 255
YES
NO
B4
SET DISP PRIORITY LIMIT PRIORITY AND PLACE INTO CREATED TCB
E3
C3
C3
DISP PRIORITY GT LIMIT PRIORITY
YES
NO
D3
PLACE RESULT INTO CREATED TCB
BB E3
FROM TIME/SLICE ATTACH (CHART BD)
ATOK4
E3
IS GIVE SUB-POOL PARAMETER SPECIFIED
NO
BC A1
YES
F3
IS SUBPOOL SHARED
YES
BC A1
NO
G3
SPECIFIED SUBPOOL NO. 128-255
YES
G4
EXIT
TO ABEND1 ROUTINE (IGC0001C) -- CHART HI -- VIA SUPERVISOR LINKAGE
NO
SPSEARCH
H3
SPQE QUEUED TO ATTACHER'S TCB
NO
SPSRCH
H4
SPQE QUEUED TO CREATED TCB
YES
BC A1
YES
NO
J3
IS SPECIFIED SUBPOOL NO. ZERO
YES
BC A1
NO
J4
BLDSPQE
CONSTRUCT SUBPOOL QUEUE ELEMENT (SPQE)
K3
K3
K3
SPCHAIN
QUEUE SPQE TO CREATED TCB
BC A1

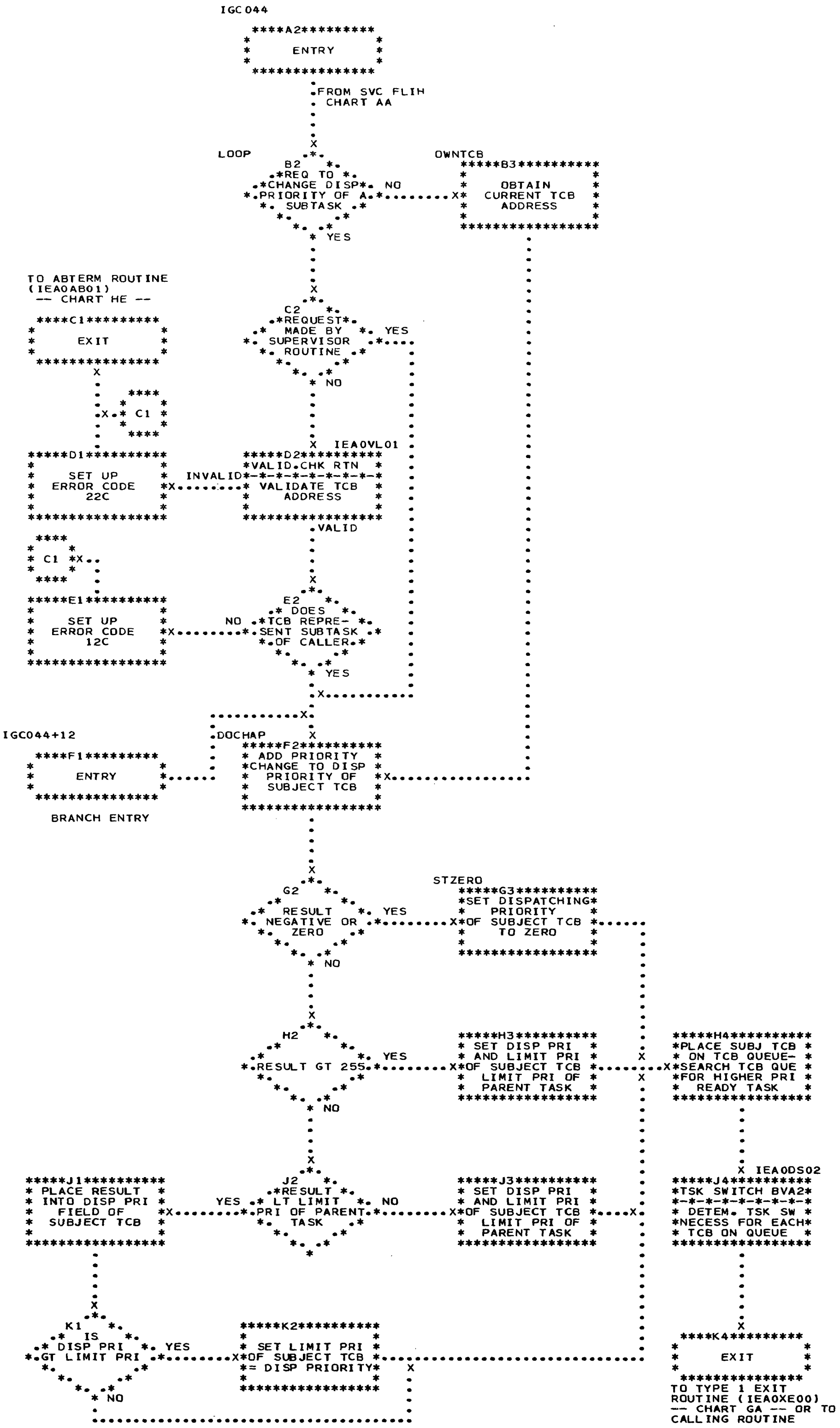
• Chart BC. Attach Routine (Part 3 of 3)


BC A1
FROM CHART BB
SPSHARE
A1 SHARE SUBPOOL PARAMETER SPECIFIED
NO
YES
E2
B1
B1 GET SPECIFIED SUBPOOL NO.
C1 SPECIFIED SUBPOOL NO. 128-255
YES
NO
C2 EXIT
TO ABTERM ROUTINE (IEA0B00) -- CHART HE --
D1 SPECIFIED SUBPOOL NO. 0
YES
NO
D2 ADDITIONAL SUBPOOL NOS SPECIFIED
YES
NO
E2
B1
SHARE0
E1 SPQE QUEUED TO CREATED TCB
YES
NO
J1
F1
E2 IS MASTER SCHEDULER ATTACHER
YES
NO
A3
F1 SPQE QUEUED TO ATTACHER'S TCB
YES
NO
H1
G1
F2 IS INITIATOR ATTACHER
NO
YES
F1
G1 BLDSPQE BUILD SPQE, QUEUE TO ATTACHER'S TCB
G2 SPQE FOR SUBPOOL 0 QUEUED ON INITIATOR TCB
NO
YES
G1
H1
H1 BLDSPQE BUILD SPQE, QUEUE TO CREATED TCB
J1
J1 ADDITIONAL SUBPOOL NOS SPECIFIED
YES
NO
B1
K1 CURRENT SUBPOOL ID = 0
YES
NO
E2
IGC004(S)
K2 GETMAIN DAA1 GET SPACE FOR SAVE AREA. 72 BYTES, SP=250
A3
A3
SETFIELD
A3 INITIALIZE TCBNTC, TCBLTC, AND TCBOTC FIELDS OF CREATED TCB
B3 PLACE CREATED TCB ONTO TCB QUEUE
IEA0DS02
C3 TSK SWTCH BVA2 DETERMINE IF TASK SWITCH IS NECESSARY
D3 PLACE ADDR OF CDCONTRL CHART CAG2 INTO CREATED TASK'S SVRB
E3 PLACE ADDRESS OF CREATED TCB INTO ATTACHER'S TCB
F3 PLACE ADDRESS OF SAVE AREA INTO CREATED TASK'S SVRB
G3 'DE' PARAMETER SPECIFIED
YES
NO
G4 COMPLEMENT SPECIFIED ADDR AND PLACE INTO REG 14 FIELD OF SVRB SAVE AREA
NOTDEADR
H3 MOVE ENTRY POINT NAME INTO REG 14 AND 15 FIELDS OF SVRB REG SAVE AREA
J3 PLACE ADDR OF REG 14 FIELD OF SVRB INTO REG 1 FIELD OF TCB REG SAVE AREA
K3 PLACE ADDRESS GIVEN WITH DCB PARAMETER INTO REG 0
A5
A5
A5 REMOVE SVRB FRM ATTACHER'S RB QUEUE. PLACE ON CREATED TASK RB QUEUE
B5 PLACE CVT ADDR AND SVRB ADDR INTO CREATED TCB
EXIT
C5 EXIT
TO DISPATCHER (IEA0DS) -- CHART GG --

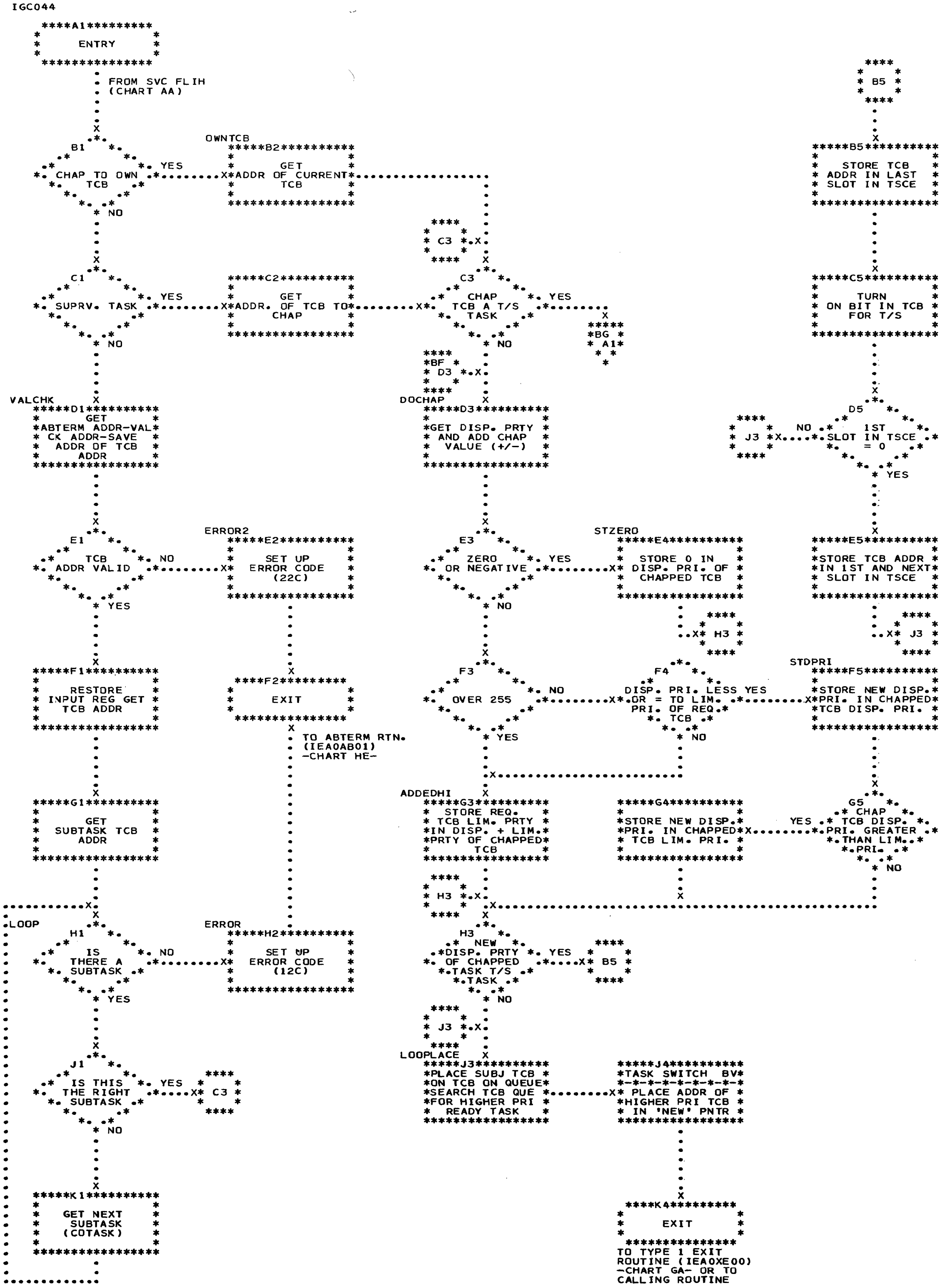
• Chart BD. Attach Routine (With Time-Slicing)


BD A1
FROM ATTACH CHART BBC1
ATOK2
A1 GET ATTACHOR'S LIMIT PRTY FROM TCB
B1 LIMIT PRTY PARAM SPECIFIED
NO
B2 MOVE LIMIT PRTY FROM ATTACHOR'S TCB TO CREATED TCB
F1
YES
C1 SUBTRACT SPECIFIED LIMIT PRTY FROM ATTACHOR'S LIMIT PRTY
D1 RESULT NEGATIVE OR ZERO
YES
NO
E1 STORE RESULT IN CREATED TCB
F1
F1 GET ATTACHOR'S DISP PRTY FROM TCB + ADD SPECIFIED DISP. PRTY
G1 RESULT NEGATIVE
YES
NO
H1 DISP. PRTY. SPEC. IN PARAM=0
YES
B3
NO
J1 MOVE 255 TO DISP. PRTY. OF CREATED TCB
A3
A3
STNEWD
A3 IS ATTACHOR'S. PLUS SPECIFIED DISP. PRTY GT 255
YES
NO
B3
STNEWD
B3 MOVE RESULT TO DISP PRTY OF CREATED TCB
ATDPPROC
C3 CREATED TCB LIMIT PRTY GT DISP PRTY.
YES
NO
D3 MOVE LIMIT PRTY TO DISP PRTY IN CREATED TCB
E3 CREATED TCB DISP PRTY A T/S DISP PRTY
NO
YES
F3 TURN ON T/S BIT IN CREATED TCB
G3 STORE ADDR OF CREATED TCB IN LAST SLOT IN TSCE
H3 1ST SLOT IN TSCE = 0
NO
YES
J3 STORE ADDR OF CREATED TCB IN FIRST + NEXT SLOT IN TSCE
BB E3

• Chart BE. Chap Routine


IGC044
A2
ENTRY
FROM SVC FLIH
CHART AA
LOOP
B2
REQ TO CHANGE DISP PRIORITY OF A SUBTASK
NO
YES
OWNTCB
B3
OBTAIN CURRENT TCB ADDRESS
TO ABTERM ROUTINE (IEA0AB01) -- CHART HE --
C1
EXIT
C1
C2
REQUEST MADE BY SUPERVISOR ROUTINE
YES
NO
IEA0VL01
D1
SET UP ERROR CODE 22C
INVALID
D2
VALID.CHK RTN
VALIDATE TCB ADDRESS
VALID
C1
E1
SET UP ERROR CODE 12C
NO
E2
DOES TCB REPRESENT SUBTASK OF CALLER
YES
IGC044+12
F1
ENTRY
BRANCH ENTRY
DOCHAP
F2
ADD PRIORITY CHANGE TO DISP PRIORITY OF SUBJECT TCB
G2
RESULT NEGATIVE OR ZERO
YES
NO
STZERO
G3
SET DISPATCHING PRIORITY OF SUBJECT TCB TO ZERO
H2
RESULT GT 255
YES
NO
H3
SET DISP PRI AND LIMIT PRI OF SUBJECT TCB = LIMIT PRI OF PARENT TASK
H4
PLACE SUBJ TCB ON TCB QUEUE- SEARCH TCB QUE FOR HIGHER PRI READY TASK
IEA0DS02
J1
PLACE RESULT INTO DISP PRI FIELD OF SUBJECT TCB
YES
J2
RESULT LT LIMIT PRI OF PARENT TASK
NO
J3
SET DISP PRI AND LIMIT PRI OF SUBJECT TCB = LIMIT PRI OF PARENT TASK
J4
TSK SWITCH BVA2
DETEM. TSK SW NECESS FOR EACH TCB ON QUEUE
K1
IS DISP PRI GT LIMIT PRI
YES
NO
K2
SET LIMIT PRI OF SUBJECT TCB = DISP PRIORITY
K4
EXIT
TO TYPE 1 EXIT ROUTINE (IEA0XE00) -- CHART GA -- OR TO CALLING ROUTINE

• Chart BF. Chap Routine (With Time-Slicing; Part 1 of 2)



IGC044
A1
ENTRY
FROM SVC FLIH (CHART AA)
B1
CHAP TO OWN TCB
YES
NO
OWNTCB
B2
GET ADDR OF CURRENT TCB
C3
C1
SUPRV. TASK
YES
NO
C2
GET ADDR. OF TCB TO CHAP
C3
CHAP TCB A T/S TASK
YES
NO
BG A1
BF D3
VALCHK
D1
GET ABTERM ADDR-VAL CK ADDR-SAVE ADDR OF TCB ADDR
DOCHAP
D3
GET DISP. PRTY AND ADD CHAP VALUE (+/-)
E1
TCB ADDR VALID
NO
YES
ERROR2
E2
SET UP ERROR CODE (22C)
E3
ZERO OR NEGATIVE
YES
NO
STZERO
E4
STORE 0 IN DISP. PRI. OF CHAPPED TCB
H3
F1
RESTORE INPUT REG GET TCB ADDR
F2
EXIT
TO ABTERM RTN. (IEA0AB01) -CHART HE-
F3
OVER 255
NO
YES
F4
DISP. PRI. LESS OR = TO LIM. PRI. OF REQ. TCB
YES
NO
G1
GET SUBTASK TCB ADDR
ADDEDHI
G3
STORE REQ. TCB LIM. PRTY IN DISP. + LIM. PRTY OF CHAPPED TCB
G4
STORE NEW DISP. PRI. IN CHAPPED TCB LIM. PRI.
H3
LOOP
H1
IS THERE A SUBTASK
NO
YES
ERROR
H2
SET UP ERROR CODE (12C)
H3
NEW DISP. PRTY OF CHAPPED TASK T/S TASK
YES
NO
B5
J3
J1
IS THIS THE RIGHT SUBTASK
YES
NO
C3
LOOPLACE
J3
PLACE SUBJ TCB ON TCB ON QUEUE SEARCH TCB QUE FOR HIGHER PRI READY TASK
J4
TASK SWITCH BV
PLACE ADDR OF HIGHER PRI TCB IN 'NEW' PNTR
K1
GET NEXT SUBTASK (COTASK)
K4
EXIT
TO TYPE I EXIT ROUTINE (IEA0XE00) -CHART GA- OR TO CALLING ROUTINE
B5
B5
STORE TCB ADDR IN LAST SLOT IN TSCE
C5
TURN ON BIT IN TCB FOR T/S
D5
1ST SLOT IN TSCE = 0
NO
YES
J3
E5
STORE TCB ADDR IN 1ST AND NEXT SLOT IN TSCE
J3
STDPRI
F5
STORE NEW DISP. PRI. IN CHAPPED TCB DISP. PRI.
G5
CHAP TCB DISP. PRI. GREATER THAN LIM. PRI.
YES
NO

Chart BG. Chap Routine (With Time-Slicing; Part 2 of 2)

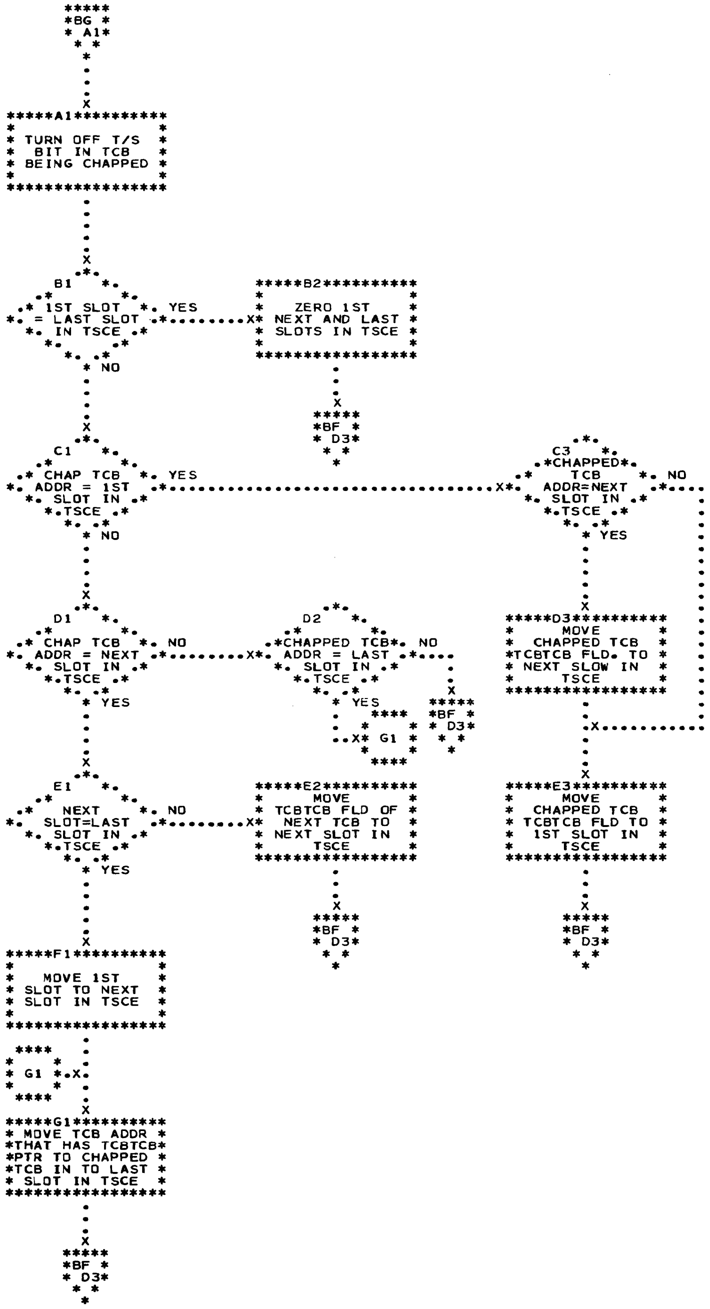

BG A1
A1
TURN OFF T/S BIT IN TCB BEING CHAPPED
B1
1ST SLOT = LAST SLOT IN TSCE
YES
NO
B2
ZERO 1ST NEXT AND LAST SLOTS IN TSCE
BF D3
C1
CHAP TCB ADDR = 1ST SLOT IN TSCE
YES
NO
C3
CHAPPED TCB ADDR=NEXT SLOT IN TSCE
NO
YES
D1
CHAP TCB ADDR = NEXT SLOT IN TSCE
NO
YES
D2
CHAPPED TCB ADDR = LAST SLOT IN TSCE
NO
YES
G1
BF D3
D3
MOVE CHAPPED TCB TCBTCB FLD. TO NEXT SLOW IN TSCE
E1
NEXT SLOT=LAST SLOT IN TSCE
NO
YES
E2
MOVE TCBTCB FLD OF NEXT TCB TO NEXT SLOT IN TSCE
BF D3
E3
MOVE CHAPPED TCB TCBTCB FLD TO 1ST SLOT IN TSCE
BF D3
F1
MOVE 1ST SLOT TO NEXT SLOT IN TSCE
G1
G1
MOVE TCB ADDR THAT HAS TCBTCB PTR TO CHAPPED TCB IN TO LAST SLOT IN TSCE
BF D3

Chart BH. Extract Routine

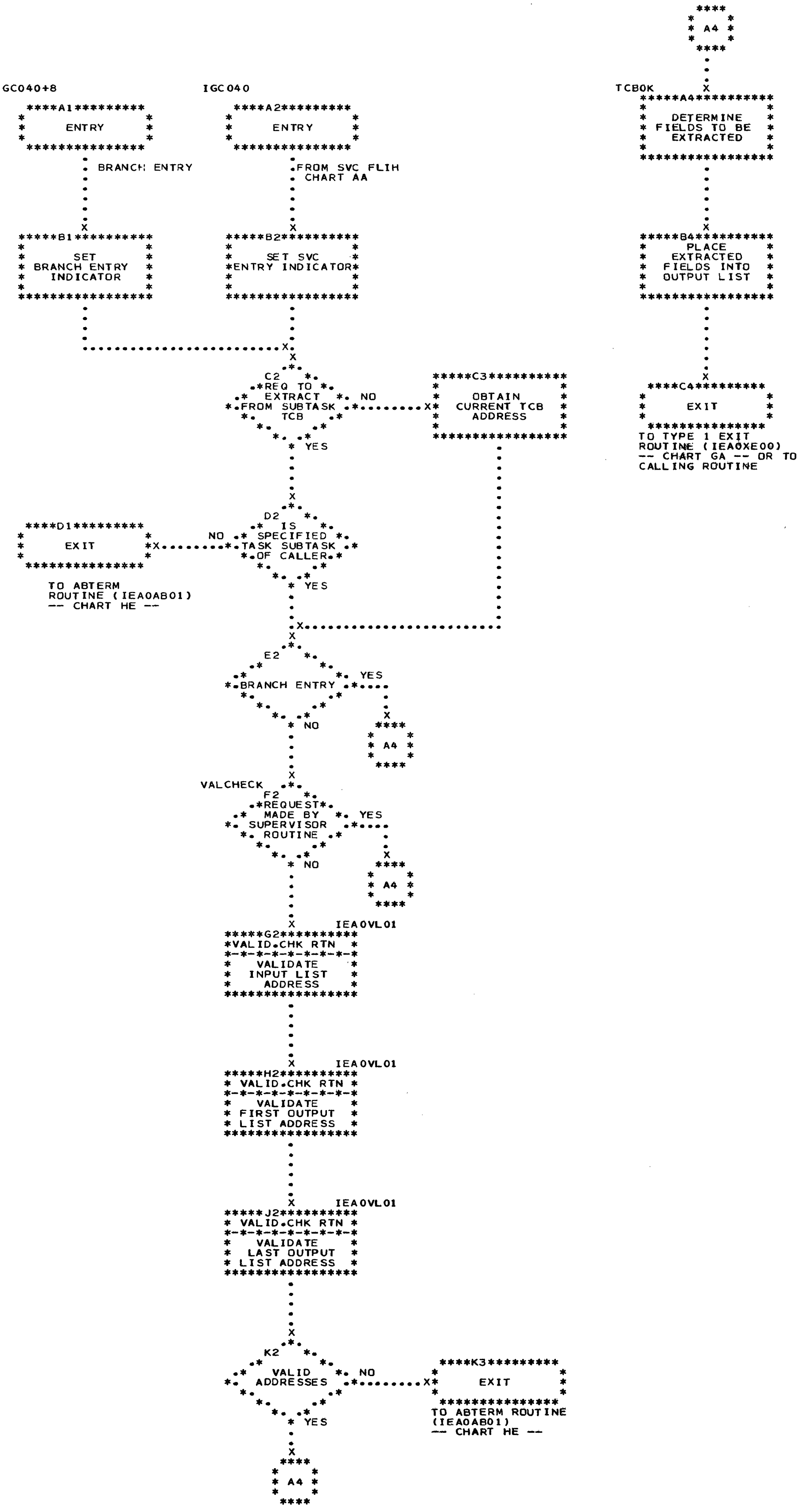

A4
IGC040+8
A1 ENTRY
BRANCH ENTRY
B1 SET BRANCH ENTRY INDICATOR
IGC040
A2 ENTRY
FROM SVC FLIH CHART AA
B2 SET SVC ENTRY INDICATOR
C2 REQ TO EXTRACT FROM SUBTASK TCB
NO
YES
C3 OBTAIN CURRENT TCB ADDRESS
D2 IS SPECIFIED TASK SUBTASK OF CALLER
NO
YES
D1 EXIT
TO ABTERM ROUTINE (IEA0AB01) -- CHART HE --
E2 BRANCH ENTRY
YES
NO
A4
VALCHECK
F2 REQUEST MADE BY SUPERVISOR ROUTINE
YES
NO
A4
IEA0VL01
G2 VALID.CHK RTN VALIDATE INPUT LIST ADDRESS
IEA0VL01
H2 VALID.CHK RTN VALIDATE FIRST OUTPUT LIST ADDRESS
IEA0VL01
J2 VALID.CHK RTN VALIDATE LAST OUTPUT LIST ADDRESS
K2 VALID ADDRESSES
NO
YES
K3 EXIT
TO ABTERM ROUTINE (IEA0AB01) -- CHART HE --
A4
TCBOK
A4 DETERMINE FIELDS TO BE EXTRACTED
B4 PLACE EXTRACTED FIELDS INTO OUTPUT LIST
C4 EXIT
TO TYPE 1 EXIT ROUTINE (IEA0XE00) -- CHART GA -- OR TO CALLING ROUTINE

Chart BI. Detach Routine

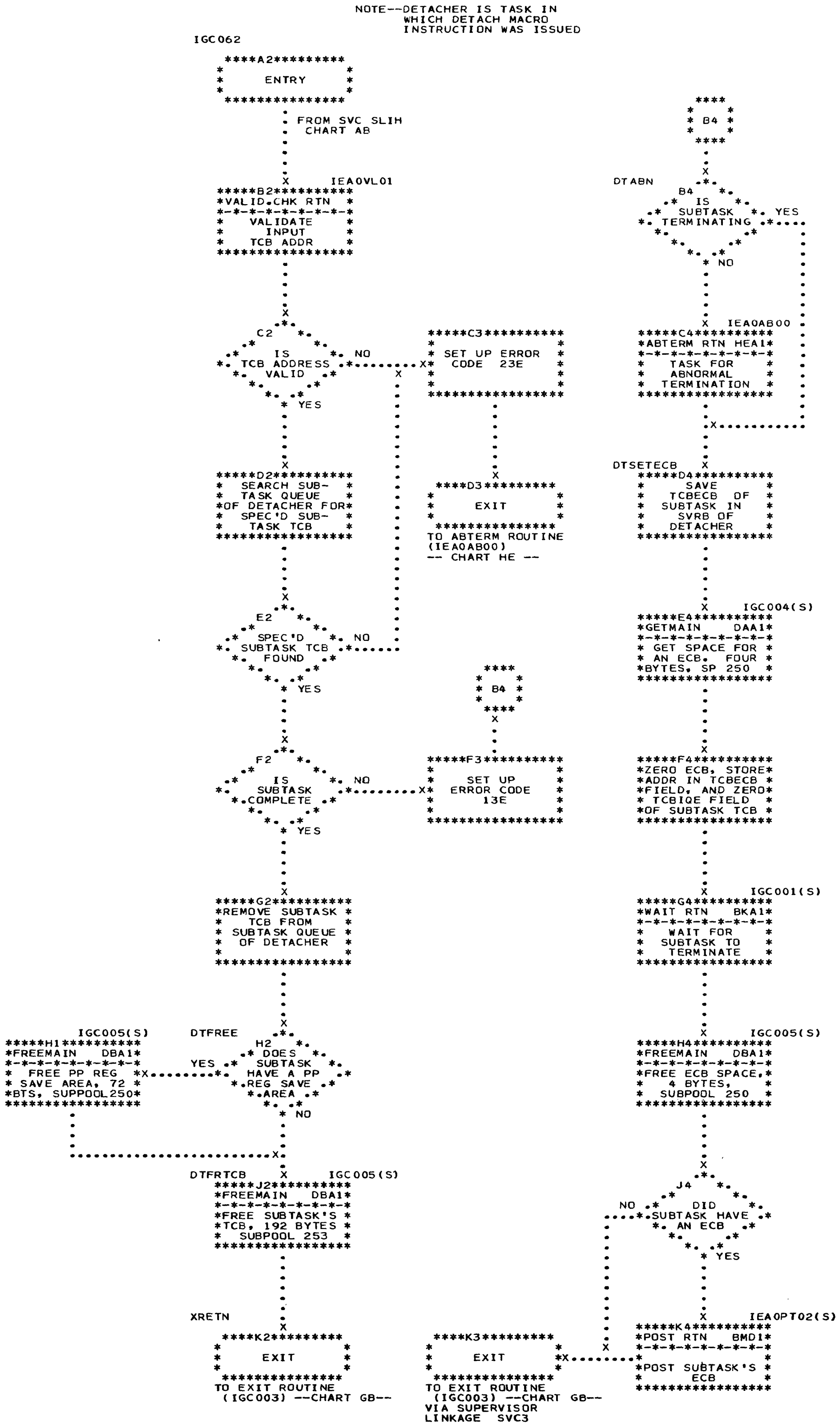
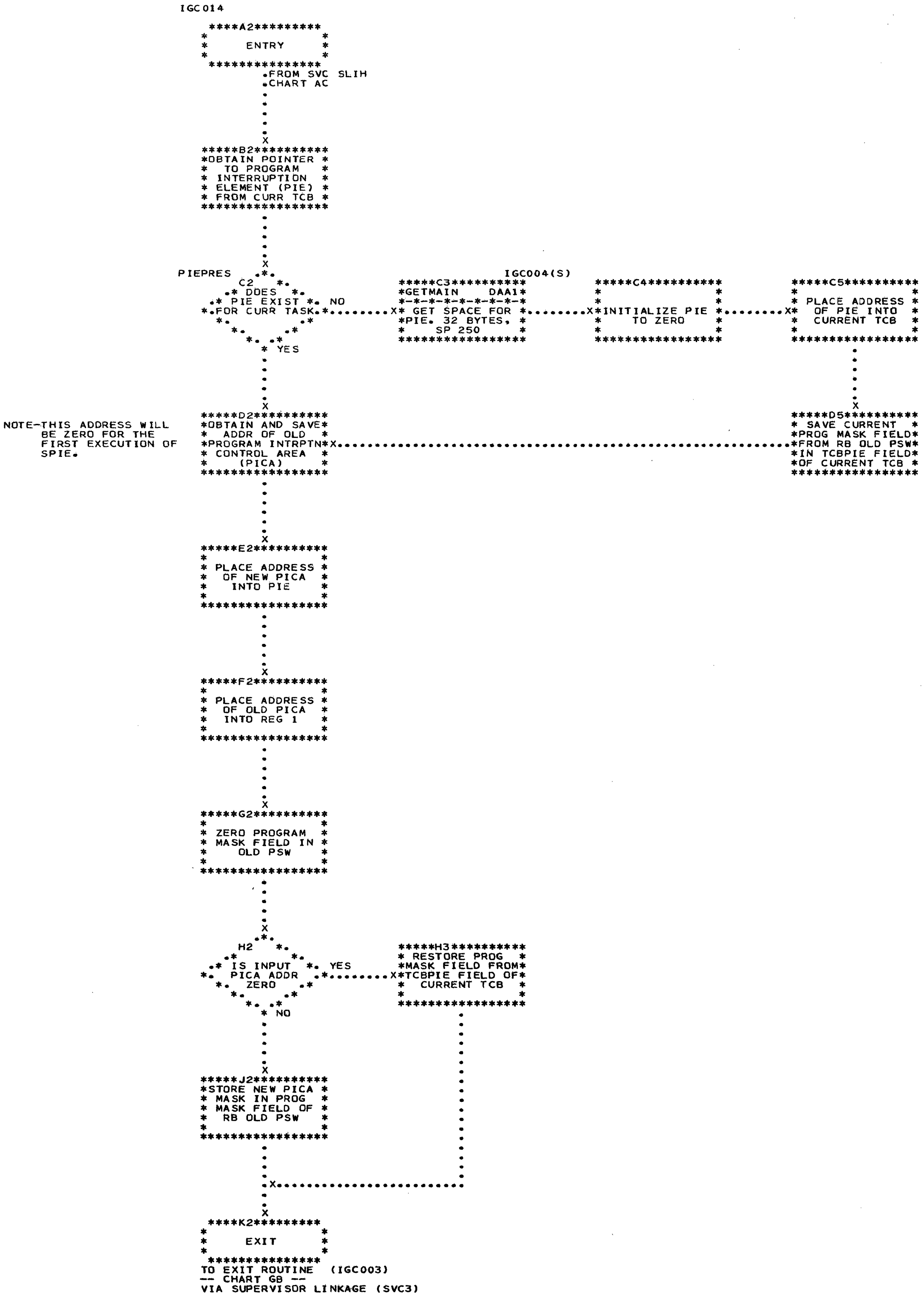

NOTE--DETACHER IS TASK IN WHICH DETACH MACRO INSTRUCTION WAS ISSUED
IGC062
A2
ENTRY
FROM SVC SLIH CHART AB
IEAOVL01
B2
VALID.CHK RTN
VALIDATE INPUT TCB ADDR
C2
IS TCB ADDRESS VALID
NO
YES
C3
SET UP ERROR CODE 23E
D2
SEARCH SUB-TASK QUEUE OF DETACHER FOR SPEC'D SUB-TASK TCB
D3
EXIT
TO ABTERM ROUTINE (IEA0AB00) -- CHART HE --
E2
SPEC'D SUBTASK TCB FOUND
NO
YES
B4
F2
IS SUBTASK COMPLETE
NO
YES
F3
SET UP ERROR CODE 13E
G2
REMOVE SUBTASK TCB FROM SUBTASK QUEUE OF DETACHER
IGC005(S)
H1
FREEMAIN DBA1
FREE PP REG SAVE AREA, 72 BTS, SUPPOOL250
DTFREE
H2
DOES SUBTASK HAVE A PP REG SAVE AREA
YES
NO
DTFRTCB
IGC005(S)
J2
FREEMAIN DBA1
FREE SUBTASK'S TCB, 192 BYTES SUBPOOL 253
XRETN
K2
EXIT
TO EXIT ROUTINE (IGC003) --CHART GB--
K3
EXIT
TO EXIT ROUTINE (IGC003) --CHART GB-- VIA SUPERVISOR LINKAGE SVC3
B4
DTABN
B4
IS SUBTASK TERMINATING
YES
NO
IEA0AB00
C4
ABTERM RTN HEA1
TASK FOR ABNORMAL TERMINATION
DTSETECB
D4
SAVE TCBECB OF SUBTASK IN SVRB OF DETACHER
IGC004(S)
E4
GETMAIN DAA1
GET SPACE FOR AN ECB. FOUR BYTES, SP 250
F4
ZERO ECB, STORE ADDR IN TCBECB FIELD, AND ZERO TCBIQE FIELD OF SUBTASK TCB
IGC001(S)
G4
WAIT RTN BKA1
WAIT FOR SUBTASK TO TERMINATE
IGC005(S)
H4
FREEMAIN DBA1
FREE ECB SPACE, 4 BYTES, SUBPOOL 250
J4
DID SUBTASK HAVE AN ECB
NO
YES
IEA0PT02(S)
K4
POST RTN BMD1
POST SUBTASK'S ECB

Chart BJ. SPIE Routine



IGC014
A2
ENTRY
FROM SVC SLIH
CHART AC
B2
OBTAIN POINTER TO PROGRAM INTERRUPTION ELEMENT (PIE) FROM CURR TCB
PIEPRES
C2
DOES PIE EXIST FOR CURR TASK
NO
YES
IGC004(S)
C3
GETMAIN DAA1
GET SPACE FOR PIE. 32 BYTES. SP 250
C4
INITIALIZE PIE TO ZERO
C5
PLACE ADDRESS OF PIE INTO CURRENT TCB
NOTE-THIS ADDRESS WILL BE ZERO FOR THE FIRST EXECUTION OF SPIE.
D2
OBTAIN AND SAVE ADDR OF OLD PROGRAM INTRPTN CONTROL AREA (PICA)
D5
SAVE CURRENT PROG MASK FIELD FROM RB OLD PSW IN TCBPIE FIELD OF CURRENT TCB
E2
PLACE ADDRESS OF NEW PICA INTO PIE
F2
PLACE ADDRESS OF OLD PICA INTO REG 1
G2
ZERO PROGRAM MASK FIELD IN OLD PSW
H2
IS INPUT PICA ADDR ZERO
YES
NO
H3
RESTORE PROG MASK FIELD FROM TCBPIE FIELD OF CURRENT TCB
J2
STORE NEW PICA MASK IN PROG MASK FIELD OF RB OLD PSW
K2
EXIT
TO EXIT ROUTINE (IGC003)
-- CHART GB --
VIA SUPERVISOR LINKAGE (SVC3)

Chart BK. Wait Routine

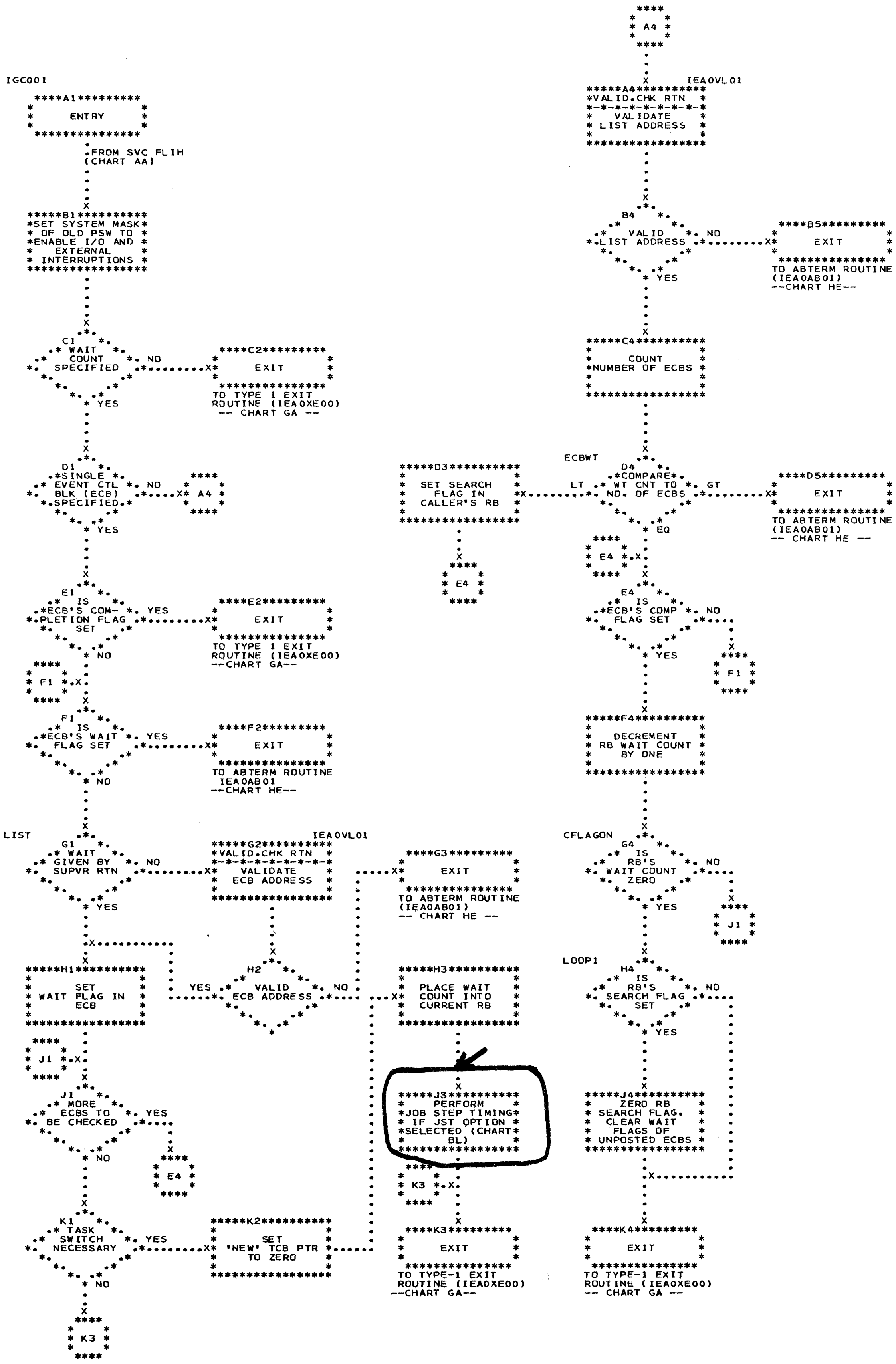

IGC001
A1
ENTRY
FROM SVC FLIH (CHART AA)
B1
SET SYSTEM MASK OF OLD PSW TO ENABLE I/O AND EXTERNAL INTERRUPTIONS
C1
WAIT COUNT SPECIFIED
NO
YES
C2
EXIT
TO TYPE 1 EXIT ROUTINE (IEA0XE00) -- CHART GA --
D1
SINGLE EVENT CTL BLK (ECB) SPECIFIED
NO
A4
YES
E1
IS ECB'S COMPLETION FLAG SET
YES
NO
E2
EXIT
TO TYPE 1 EXIT ROUTINE (IEA0XE00) --CHART GA--
F1
F1
IS ECB'S WAIT FLAG SET
YES
NO
F2
EXIT
TO ABTERM ROUTINE IEA0AB01 --CHART HE--
LIST
G1
WAIT GIVEN BY SUPVR RTN
NO
YES
IEA0VL01
G2
VALID.CHK RTN
VALIDATE ECB ADDRESS
H1
SET WAIT FLAG IN ECB
H2
VALID ECB ADDRESS
YES
NO
J1
J1
MORE ECBS TO BE CHECKED
YES
E4
NO
K1
TASK SWITCH NECESSARY
YES
NO
K3
K2
SET 'NEW' TCB PTR TO ZERO
G3
EXIT
TO ABTERM ROUTINE (IEA0AB01) -- CHART HE --
H3
PLACE WAIT COUNT INTO CURRENT RB
J3
PERFORM JOB STEP TIMING IF JST OPTION SELECTED (CHART BL)
K3
K3
EXIT
TO TYPE-1 EXIT ROUTINE (IEA0XE00) --CHART GA--
D3
SET SEARCH FLAG IN CALLER'S RB
E4
A4
IEA0VL01
A4
VALID.CHK RTN
VALIDATE LIST ADDRESS
B4
VALID LIST ADDRESS
NO
YES
B5
EXIT
TO ABTERM ROUTINE (IEA0AB01) --CHART HE--
C4
COUNT NUMBER OF ECBS
ECBWT
D4
COMPARE WT CNT TO NO. OF ECBS
LT
GT
EQ
D5
EXIT
TO ABTERM ROUTINE (IEA0AB01) -- CHART HE --
E4
E4
IS ECB'S COMP FLAG SET
NO
F1
YES
F4
DECREMENT RB WAIT COUNT BY ONE
CFLAGON
G4
IS RB'S WAIT COUNT ZERO
NO
J1
YES
LOOP1
H4
IS RB'S SEARCH FLAG SET
NO
YES
J4
ZERO RB SEARCH FLAG, CLEAR WAIT FLAGS OF UNPOSTED ECBS
K4
EXIT
TO TYPE-1 EXIT ROUTINE (IEA0XE00) -- CHART GA --

Chart BL. Wait Routine (Job Step Timing)

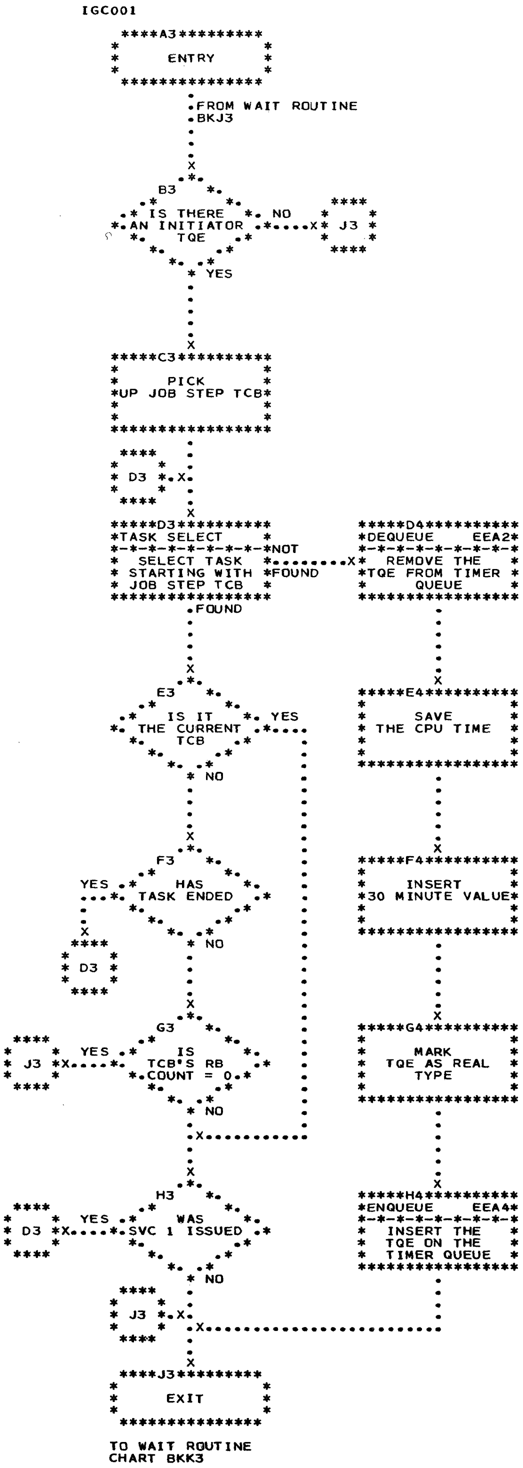


Chart BM. Post Routine

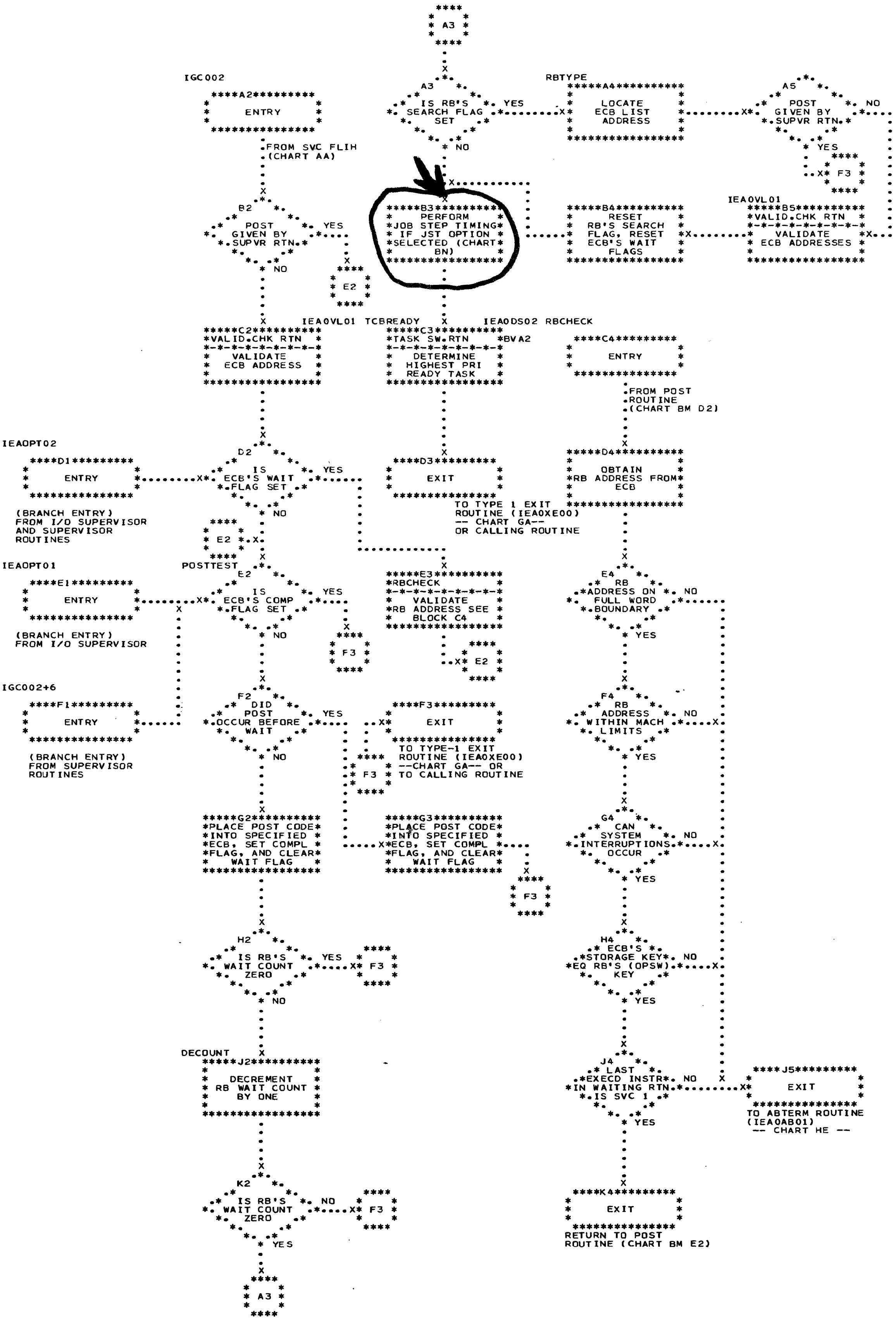

A3
IGC002
A2
ENTRY
FROM SVC FLIH (CHART AA)
A3
IS RB'S SEARCH FLAG SET
YES
NO
RBTYPE
A4
LOCATE ECB LIST ADDRESS
A5
POST GIVEN BY SUPVR RTN.
NO
YES
F3
B2
POST GIVEN BY SUPVR RTN.
YES
NO
E2
B3
PERFORM JOB STEP TIMING IF JST OPTION SELECTED (CHART BN)
B4
RESET RB'S SEARCH FLAG, RESET ECB'S WAIT FLAGS
IEAOVL01
B5
VALID.CHK RTN
VALIDATE ECB ADDRESSES
IEAOVL01
C2
VALID.CHK RTN
VALIDATE ECB ADDRESS
TCBREADY
C3
TASK SW.RTN
*BVA2
DETERMINE HIGHEST PRI READY TASK
IEAODS02 RBCHECK
C4
ENTRY
FROM POST ROUTINE (CHART BM D2)
IEAOPT02
D1
ENTRY
(BRANCH ENTRY) FROM I/O SUPERVISOR AND SUPERVISOR ROUTINES
D2
IS ECB'S WAIT FLAG SET
YES
NO
D3
EXIT
TO TYPE 1 EXIT ROUTINE (IEAOXE00) -- CHART GA-- OR CALLING ROUTINE
D4
OBTAIN RB ADDRESS FROM ECB
E2
IEAOPT01
E1
ENTRY
(BRANCH ENTRY) FROM I/O SUPERVISOR
POSTTEST
E2
IS ECB'S COMP FLAG SET
YES
NO
F3
E3
RBCHECK
VALIDATE RB ADDRESS SEE BLOCK C4
E2
E4
RB ADDRESS ON FULL WORD BOUNDARY
NO
YES
IGC002+6
F1
ENTRY
(BRANCH ENTRY) FROM SUPERVISOR ROUTINES
F2
DID POST OCCUR BEFORE WAIT
YES
NO
F3
F3
EXIT
TO TYPE-1 EXIT ROUTINE (IEAOXE00) --CHART GA-- OR TO CALLING ROUTINE
F4
RB ADDRESS WITHIN MACH. LIMITS
NO
YES
G2
PLACE POST CODE INTO SPECIFIED ECB, SET COMPL FLAG, AND CLEAR WAIT FLAG
G3
PLACE POST CODE INTO SPECIFIED ECB, SET COMPL FLAG, AND CLEAR WAIT FLAG
F3
G4
CAN SYSTEM INTERRUPTIONS OCCUR
NO
YES
H2
IS RB'S WAIT COUNT ZERO
YES
F3
NO
H4
ECB'S STORAGE KEY EQ RB'S (OPSW) KEY
NO
YES
DECOUNT
J2
DECREMENT RB WAIT COUNT BY ONE
J4
LAST EXECD INSTR IN WAITING RTN IS SVC 1
NO
YES
J5
EXIT
TO ABTERM ROUTINE (IEAOAB01) -- CHART HE --
K2
IS RB'S WAIT COUNT ZERO
NO
F3
YES
A3
K4
EXIT
RETURN TO POST ROUTINE (CHART BM E2)

Chart BN. Post Routine (Job Step Timing)

```
                IGC002
                 ****A3*********
                *               *
                *     ENTRY     *
                *               *
                 ***************
                        .
                        .FROM
                        .POST  ROUTINE
                        .BIA3
                        .
                        .
                        X
                       .*.
                 B3  .*   *.
                   .*  IS   *.
                 .*   THERE   *.  NO
                *. AN INITIA-   .*.....
                  *. TOR TQE  .*      .
                    *.      .*        .
                      *.  .*          .
                        *  YES        .
                        .             .
                        .             .
                        .             .
                        X             .
                       .*.            .
                 C3  .*   *.          .
                   .*  IS   *.        .
                 .*   IS IT A  *.  NO X
                *.REAL TQE + ON.*.....
                  *.  QUEUE   .*      .
                    *.      .*        .
                      *.  .*          .
                        *  YES        .
                        .             .
                        .             .
                        .             .
                        X             .
               *****D3**********      .
               *DEQUEUE         *     .
               *-*-*-*-*-*-*-*-*-*    .
               *  REMOVE THE    *     .
               * TQE FROM THE   *     .
               *  TIMER QUEUE   *     .
               ******************     .
                        .             .
                        .             .
                        .             .
                        .             .
                        X             .
               *****E3**********      .
               *                *     .
               *     REPLACE    *     .
               * CPU TASK TIME  *     .
               *     IN TQE     *     .
               *                *     .
               ******************     .
                        .             .
                        .             .
                        .             .
                        .             .
                        X             .
               *****F3**********      .
               *                *     .
               *      MARK      *     .
               *  TQE AS TASK   *     .
               *      TYPE      *     .
               *                *     .
               ******************     .
                        .             .
                        .X.............
                        .
                        .
                        .
                        X
                ****G3*********
               *               *
               *     EXIT      *
               *               *
                ***************
               TO POST ROUTINE
               -CHART BMC3-
```

Chart BO. ENQ Routine

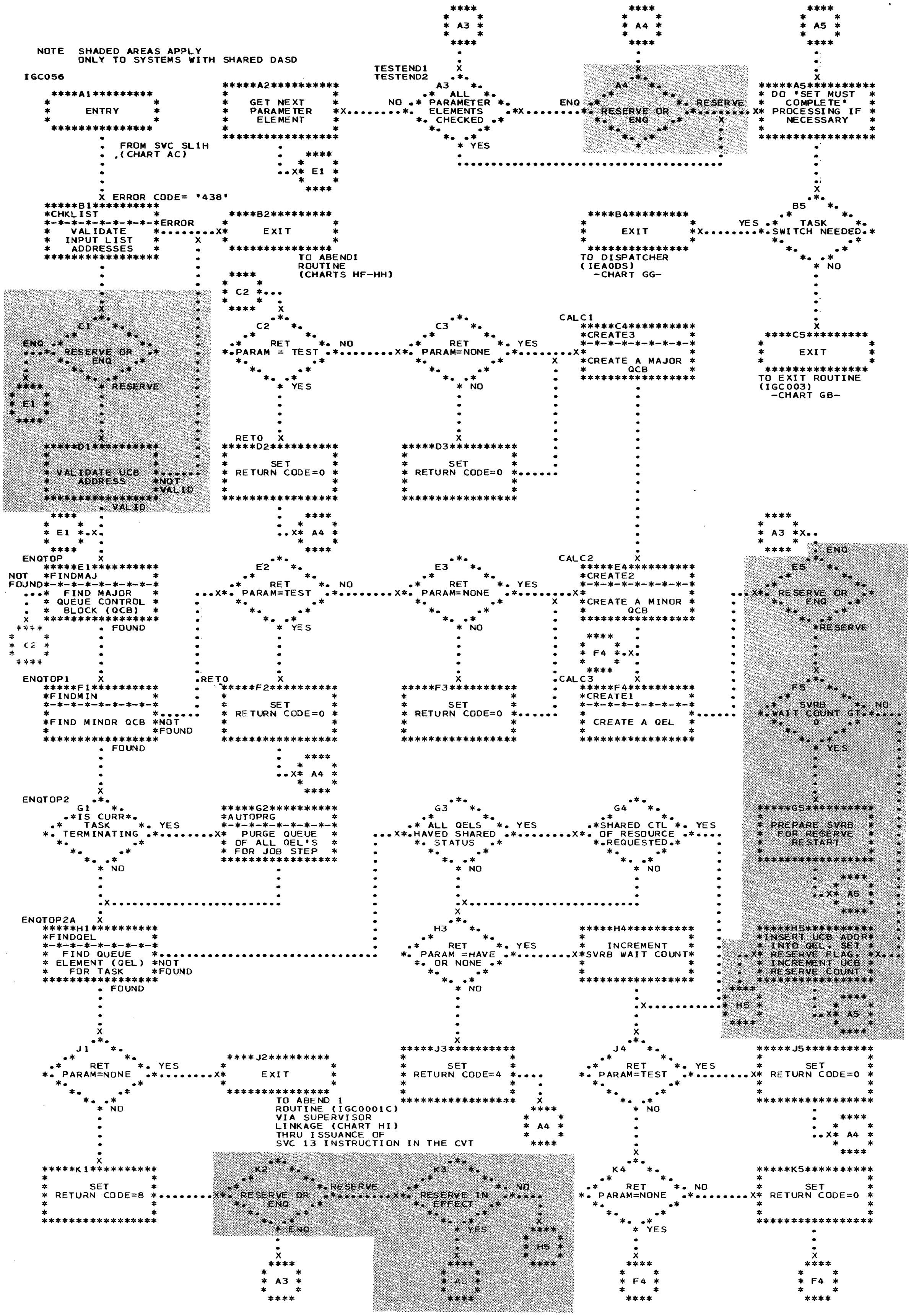

NOTE SHADED AREAS APPLY ONLY TO SYSTEMS WITH SHARED DASD
IGC056
A1 ENTRY
FROM SVC SL1H (CHART AC)
ERROR CODE= '438'
B1 CHKLIST VALIDATE INPUT LIST ADDRESSES
ERROR
C1 RESERVE OR ENQ
ENQ
RESERVE
E1
D1 VALIDATE UCB ADDRESS
NOT VALID
VALID
E1
ENQTOP
E1 FINDMAJ FIND MAJOR QUEUE CONTROL BLOCK (QCB)
NOT FOUND
FOUND
C2
ENQTOP1
F1 FINDMIN FIND MINOR QCB
NOT FOUND
FOUND
ENQTOP2
G1 IS CURR TASK TERMINATING
YES
NO
ENQTOP2A
H1 FINDQEL FIND QUEUE ELEMENT (QEL) FOR TASK
NOT FOUND
FOUND
J1 RET PARAM=NONE
YES
NO
K1 SET RETURN CODE=8
A2 GET NEXT PARAMETER ELEMENT
E1
B2 EXIT
TO ABENDI ROUTINE (CHARTS HF-HH)
C2
C2 RET PARAM = TEST
NO
YES
RETO
D2 SET RETURN CODE=0
A4
E2 RET PARAM=TEST
NO
YES
RETO
F2 SET RETURN CODE=0
A4
G2 AUTOPRG PURGE QUEUE OF ALL QEL'S FOR JOB STEP
J2 EXIT
TO ABEND 1 ROUTINE (IGC0001C) VIA SUPERVISOR LINKAGE (CHART HI) THRU ISSUANCE OF SVC 13 INSTRUCTION IN THE CVT
K2 RESERVE OR ENQ
RESERVE
ENQ
A3
A3
TESTEND1
TESTEND2
A3 ALL PARAMETER ELEMENTS CHECKED
NO
YES
C3 RET PARAM=NONE
YES
NO
D3 SET RETURN CODE=0
E3 RET PARAM=NONE
YES
NO
F3 SET RETURN CODE=0
G3 ALL QELS HAVED SHARED STATUS
YES
NO
H3 RET PARAM =HAVE OR NONE
YES
NO
J3 SET RETURN CODE=4
A4
K3 RESERVE IN EFFECT
NO
YES
H5
A4
A4
A4 RESERVE OR ENQ
ENQ
RESERVE
B4 EXIT
TO DISPATCHER (IEAODS) -CHART GG-
CALC1
C4 CREATE3 CREATE A MAJOR QCB
CALC2
E4 CREATE2 CREATE A MINOR QCB
F4
CALC3
F4 CREATE1 CREATE A QEL
G4 SHARED CTL OF RESOURCE REQUESTED
YES
NO
H4 INCREMENT SVRB WAIT COUNT
J4 RET PARAM=TEST
YES
NO
K4 RET PARAM=NONE
NO
YES
F4
A5
A5 DO 'SET MUST COMPLETE' PROCESSING IF NECESSARY
B5 TASK SWITCH NEEDED
YES
NO
C5 EXIT
TO EXIT ROUTINE (IGC003) -CHART GB-
A3
ENQ
E5 RESERVE OR ENQ
RESERVE
F5 SVRB WAIT COUNT GT 0
NO
YES
G5 PREPARE SVRB FOR RESERVE RESTART
A5
H5 INSERT UCB ADDR INTO QEL. SET RESERVE FLAG. INCREMENT UCB RESERVE COUNT
H5
A5
J5 SET RETURN CODE=0
A4
K5 SET RETURN CODE=0
F4

Chart BP. DEQ Routine

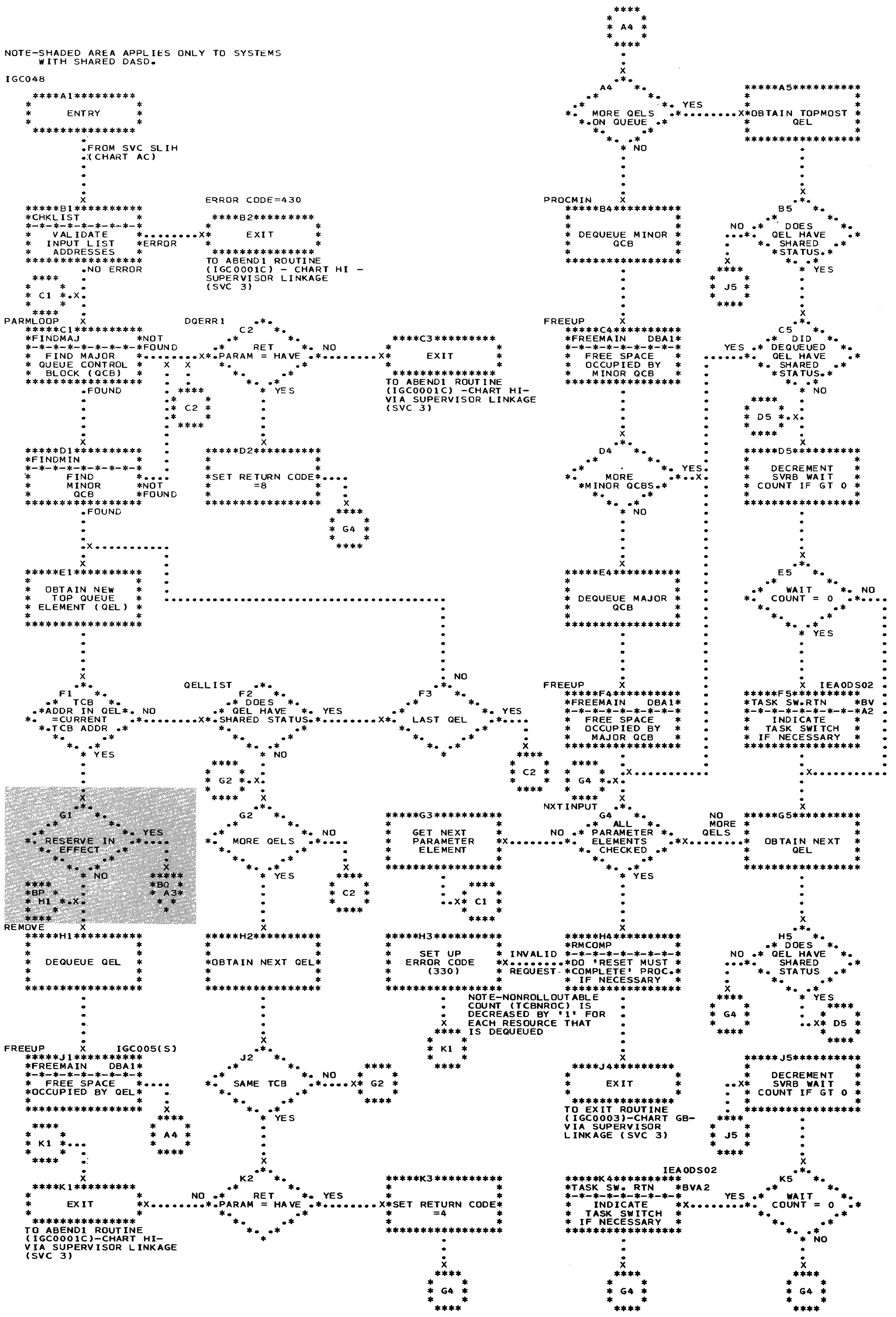

NOTE-SHADED AREA APPLIES ONLY TO SYSTEMS WITH SHARED DASD.
IGC048
A1 ENTRY
FROM SVC SLIH (CHART AC)
B1 CHKLIST VALIDATE INPUT LIST ADDRESSES
ERROR
NO ERROR
ERROR CODE=430
B2 EXIT
TO ABEND1 ROUTINE (IGC0001C) - CHART HI - SUPERVISOR LINKAGE (SVC 3)
C1
PARMLOOP
C1 FINDMAJ FIND MAJOR QUEUE CONTROL BLOCK (QCB)
NOT FOUND
FOUND
DQERR1
C2 RET PARAM = HAVE
NO
YES
C2
C3 EXIT
TO ABEND1 ROUTINE (IGC0001C) -CHART HI- VIA SUPERVISOR LINKAGE (SVC 3)
D1 FINDMIN FIND MINOR QCB
NOT FOUND
FOUND
D2 SET RETURN CODE =8
G4
E1 OBTAIN NEW TOP QUEUE ELEMENT (QEL)
F1 TCB ADDR IN QEL = CURRENT TCB ADDR
NO
YES
QELLIST
F2 DOES QEL HAVE SHARED STATUS
YES
NO
F3 LAST QEL
NO
YES
C2
G2
G1 RESERVE IN EFFECT
YES
NO
BO A3
BP H1
G2 MORE QELS
NO
YES
C2
G3 GET NEXT PARAMETER ELEMENT
C1
REMOVE
H1 DEQUEUE QEL
H2 OBTAIN NEXT QEL
H3 SET UP ERROR CODE (330)
INVALID REQUEST
NOTE-NONROLLOUTABLE COUNT (TCBNROC) IS DECREASED BY '1' FOR EACH RESOURCE THAT IS DEQUEUED
K1
FREEUP
IGC005(S)
J1 FREEMAIN DBA1 FREE SPACE OCCUPIED BY QEL
A4
K1
J2 SAME TCB
NO
G2
YES
K1 EXIT
TO ABEND1 ROUTINE (IGC0001C)-CHART HI- VIA SUPERVISOR LINKAGE (SVC 3)
K2 RET PARAM = HAVE
NO
YES
K3 SET RETURN CODE =4
G4
A4
A4 MORE QELS ON QUEUE
YES
NO
A5 OBTAIN TOPMOST QEL
PROCMIN
B4 DEQUEUE MINOR QCB
B5 DOES QEL HAVE SHARED STATUS
NO
YES
J5
FREEUP
C4 FREEMAIN DBA1 FREE SPACE OCCUPIED BY MINOR QCB
C5 DID DEQUEUED QEL HAVE SHARED STATUS
YES
NO
D5
D4 MORE MINOR QCBS
YES
NO
D5 DECREMENT SVRB WAIT COUNT IF GT 0
E4 DEQUEUE MAJOR QCB
E5 WAIT COUNT = 0
NO
YES
FREEUP
F4 FREEMAIN DBA1 FREE SPACE OCCUPIED BY MAJOR QCB
IEAODS02
F5 TASK SW.RTN BV A2 INDICATE TASK SWITCH IF NECESSARY
G4
NXTINPUT
G4 ALL PARAMETER ELEMENTS CHECKED
NO
NO MORE QELS
YES
G5 OBTAIN NEXT QEL
H4 RMCOMP DO 'RESET MUST COMPLETE' PROC. IF NECESSARY
H5 DOES QEL HAVE SHARED STATUS
NO
YES
G4
D5
J4 EXIT
TO EXIT ROUTINE (IGC0003)-CHART GB- VIA SUPERVISOR LINKAGE (SVC 3)
J5 DECREMENT SVRB WAIT COUNT IF GT 0
J5
IEAODS02
K4 TASK SW. RTN BVA2 INDICATE TASK SWITCH IF NECESSARY
K5 WAIT COUNT = 0
YES
NO
G4
G4
G4

Chart BQ. DEQ Routine (Shared DASD)

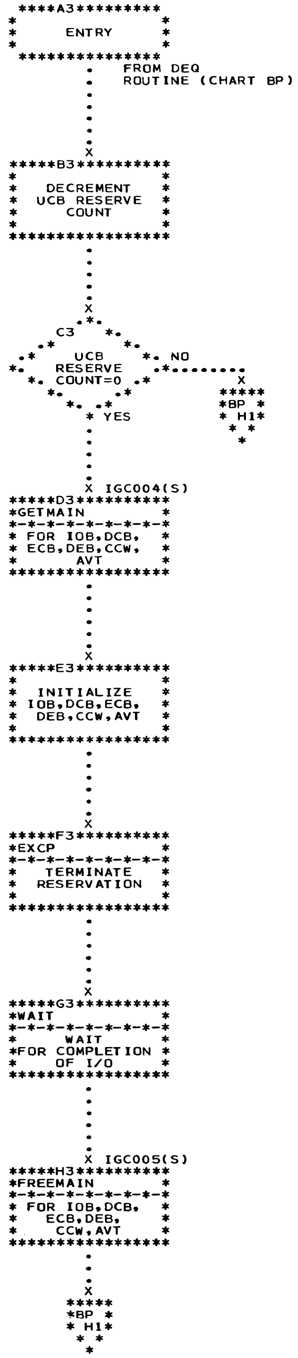
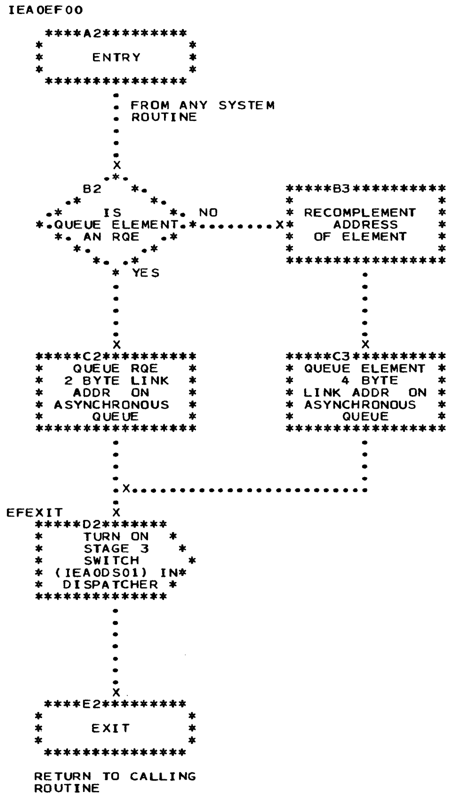

A3
ENTRY
FROM DEQ ROUTINE (CHART BP)
B3
DECREMENT UCB RESERVE COUNT
C3
UCB RESERVE COUNT=0
NO
BP H1
YES
IGC004(S)
D3
GETMAIN
FOR IOB,DCB, ECB,DEB,CCW, AVT
E3
INITIALIZE IOB,DCB,ECB, DEB,CCW,AVT
F3
EXCP
TERMINATE RESERVATION
G3
WAIT
WAIT FOR COMPLETION OF I/O
IGC005(S)
H3
FREEMAIN
FOR IOB,DCB, ECB,DEB, CCW,AVT
BP H1

Chart BR. Stage 1 Exit Effector

```
IGC043
    ****A2*********
   *               *
   *     ENTRY     *
   *               *
    ***************
            .FROM SVC SLIH
            .-CHART AC-
            .
            .
            .
            .
            .
            X     IGC004(S)
   *****B2**********
   *GETMAIN    DAA1*
   *-*-*-*-*-*-*-*-*
   * GET SPACE FOR *
   * INTERRUPTION  *
   * REQ BLK  IRB  *
   *****************
            .
            .
            .
            .
            .
            X
           .*.
        C2    *.
       .*        *.
     .*  IS SAVE   *.  NO
    *.    AREA      .*.....
     *.REQUESTED.*         .
       *.      .*          .
         *.  .*            .
           *  YES          .
            .              .
            .              .
            .              .
            .              .
            .              .
            XIGC004(S)     .
   *****D2**********       .
   *GETMAIN    DAA1*       .
   *-*-*-*-*-*-*-*-*       .
   *   GET SPACE   *       .
   *  FOR PP REG   *       .
   *   SAVE AREA   *       .
   *****************       .
            .              .
            .X..............
            .
            .
            X
   *****E2**********
   *               *
   *  INITIALIZE   *
   * IRB PER CIRB  *
   *   OPERANDS    *
   *               *
   *****************
            .
            .
            .
            .
            .
            .
            X
    ****F2*********
   *               *
   *     EXIT      *
   *               *
    ***************

   TO EXIT ROUTINE
   (IGC003)
    -- CHART GB --
   VIA SUPERVISOR
   LINKAGE (SVC 3)
```

Chart BS. Stage 2 Exit Effector


IEA0EF00
A2
ENTRY
FROM ANY SYSTEM ROUTINE
B2
IS QUEUE ELEMENT AN RQE
NO
YES
B3
RECOMPLEMENT ADDRESS OF ELEMENT
C2
QUEUE RQE 2 BYTE LINK ADDR ON ASYNCHRONOUS QUEUE
C3
QUEUE ELEMENT 4 BYTE LINK ADDR ON ASYNCHRONOUS QUEUE
EFEXIT
D2
TURN ON STAGE 3 SWITCH (IEA0DS01) IN DISPATCHER
E2
EXIT
RETURN TO CALLING ROUTINE

• Chart BT. Stage 3 Exit Effector (Part 1 of 2)

NOTE - SHADED AREA APPLIES ONLY TO MULTIPROCESSING SYSTEMS

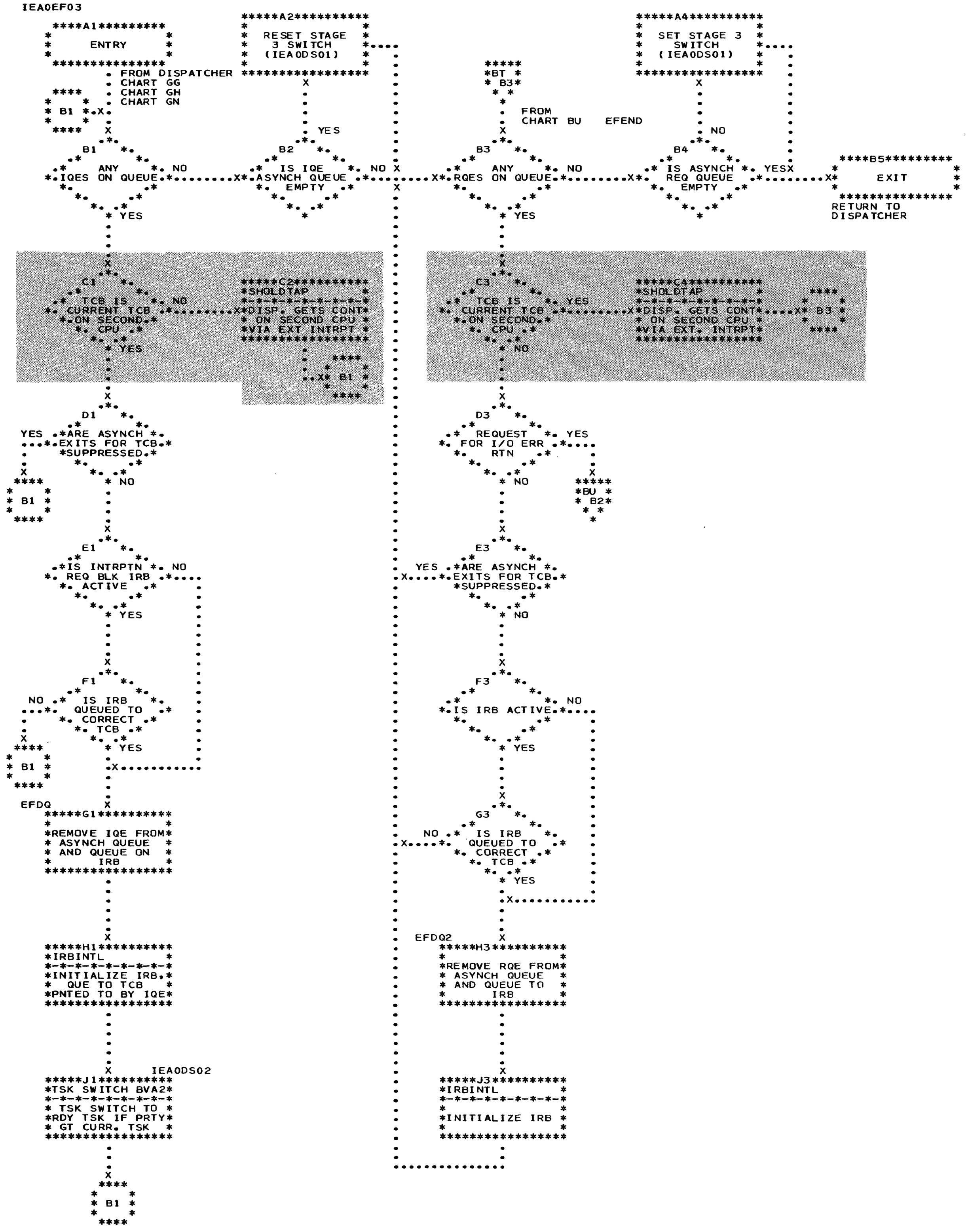

IEA0EF03
A1
ENTRY
FROM DISPATCHER
CHART GG
CHART GH
CHART GN
B1
B1
ANY IQES ON QUEUE
NO
YES
A2
RESET STAGE 3 SWITCH (IEA0DS01)
YES
B2
IS IQE ASYNCH QUEUE EMPTY
NO
BT B3
FROM CHART BU EFEND
B3
ANY RQES ON QUEUE
NO
YES
A4
SET STAGE 3 SWITCH (IEA0DS01)
NO
B4
IS ASYNCH REQ QUEUE EMPTY
YES
B5
EXIT
RETURN TO DISPATCHER
C1
TCB IS CURRENT TCB ON SECOND CPU
NO
YES
C2
SHOLDTAP
DISP. GETS CONT ON SECOND CPU VIA EXT INTRPT
B1
C3
TCB IS CURRENT TCB ON SECOND CPU
YES
NO
C4
SHOLDTAP
DISP. GETS CONT ON SECOND CPU VIA EXT. INTRPT
B3
D1
ARE ASYNCH EXITS FOR TCB SUPPRESSED
YES
NO
B1
D3
REQUEST FOR I/O ERR RTN
YES
NO
BU B2
E1
IS INTRPTN REQ BLK IRB ACTIVE
NO
YES
E3
ARE ASYNCH EXITS FOR TCB SUPPRESSED
YES
NO
F1
IS IRB QUEUED TO CORRECT TCB
NO
YES
B1
F3
IS IRB ACTIVE
NO
YES
EFDQ
G1
REMOVE IQE FROM ASYNCH QUEUE AND QUEUE ON IRB
G3
IS IRB QUEUED TO CORRECT TCB
NO
YES
H1
IRBINTL
INITIALIZE IRB, QUE TO TCB PNTED TO BY IQE
EFDQ2
H3
REMOVE RQE FROM ASYNCH QUEUE AND QUEUE TO IRB
IEA0DS02
J1
TSK SWITCH BVA2
TSK SWITCH TO RDY TSK IF PRTY GT CURR. TSK
J3
IRBINTL
INITIALIZE IRB
B1

Chart BU. Stage 3 Exit Effector (Part 2 of 2)

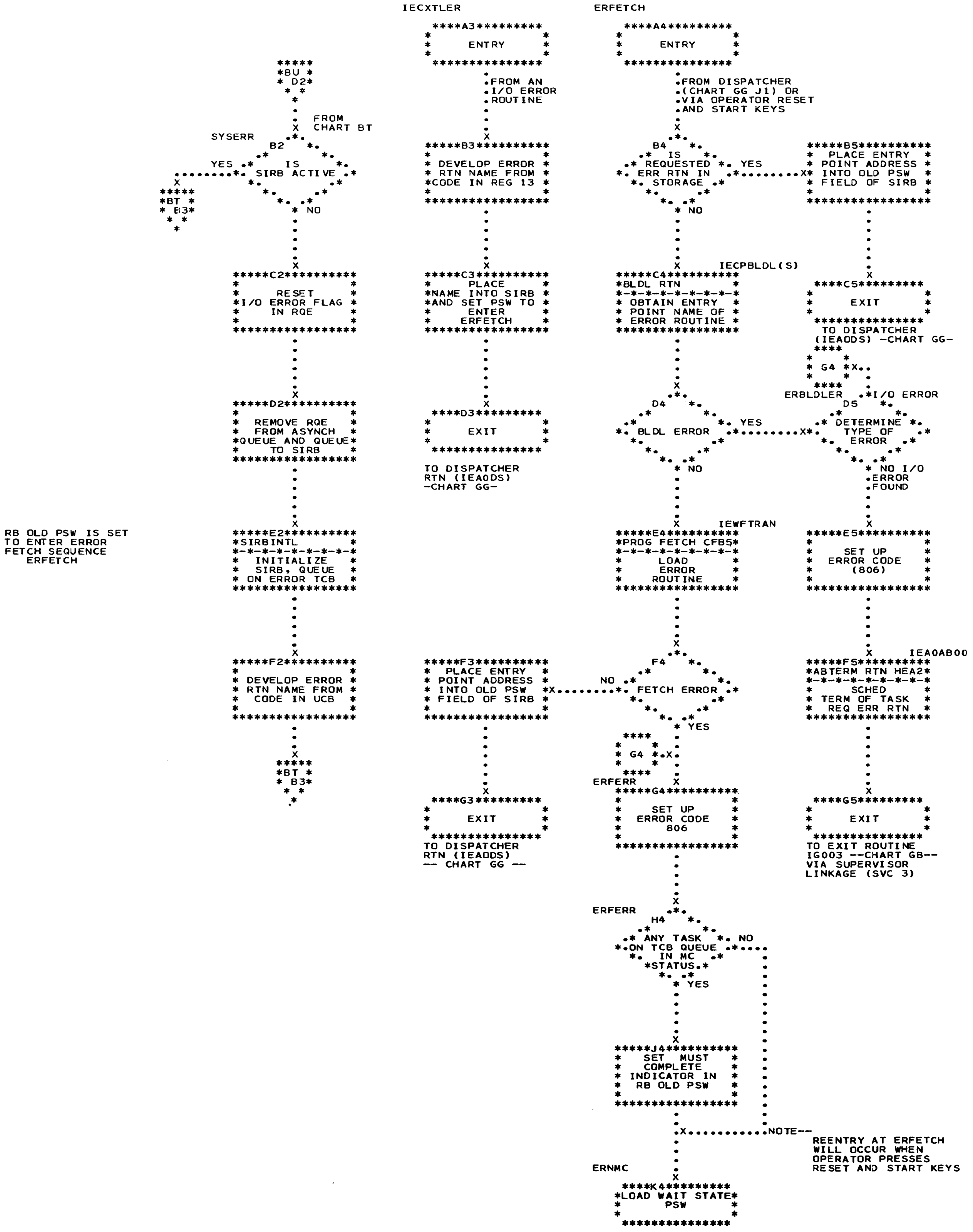
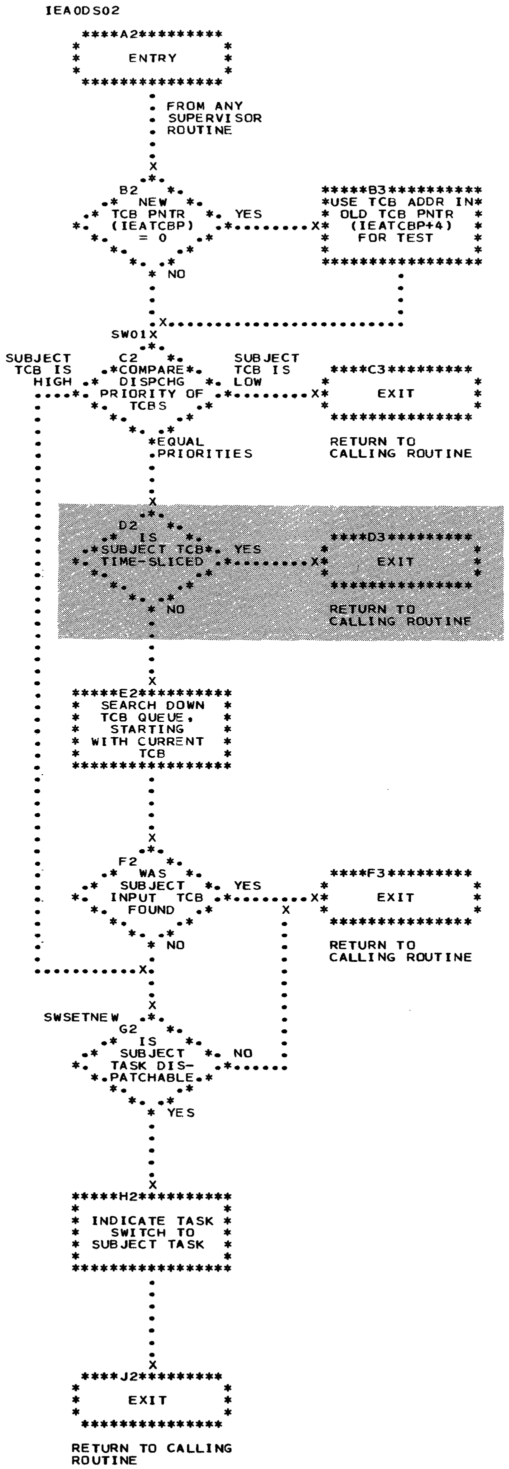

IECXTLER
A3 ENTRY
FROM AN I/O ERROR ROUTINE
B3 DEVELOP ERROR RTN NAME FROM CODE IN REG 13
C3 PLACE NAME INTO SIRB AND SET PSW TO ENTER ERFETCH
D3 EXIT
TO DISPATCHER RTN (IEAODS) -CHART GG-
ERFETCH
A4 ENTRY
FROM DISPATCHER (CHART GG J1) OR VIA OPERATOR RESET AND START KEYS
B4 IS REQUESTED ERR RTN IN STORAGE
YES
NO
B5 PLACE ENTRY POINT ADDRESS INTO OLD PSW FIELD OF SIRB
C5 EXIT
TO DISPATCHER (IEAODS) -CHART GG-
BU D2
FROM CHART BT
SYSERR
B2 IS SIRB ACTIVE
YES
NO
BT B3
C2 RESET I/O ERROR FLAG IN RQE
D2 REMOVE RQE FROM ASYNCH QUEUE AND QUEUE TO SIRB
RB OLD PSW IS SET TO ENTER ERROR FETCH SEQUENCE ERFETCH
E2 SIRBINTL
INITIALIZE SIRB, QUEUE ON ERROR TCB
F2 DEVELOP ERROR RTN NAME FROM CODE IN UCB
BT B3
IECPBLDL(S)
C4 BLDL RTN
OBTAIN ENTRY POINT NAME OF ERROR ROUTINE
D4 BLDL ERROR
YES
NO
G4
ERBLDLER
I/O ERROR
D5 DETERMINE TYPE OF ERROR
NO I/O ERROR FOUND
IEWFTRAN
E4 PROG FETCH CFB5
LOAD ERROR ROUTINE
E5 SET UP ERROR CODE (806)
IEAOAB00
F5 ABTERM RTN HEA2
SCHED TERM OF TASK REQ ERR RTN
G5 EXIT
TO EXIT ROUTINE IG003 --CHART GB-- VIA SUPERVISOR LINKAGE (SVC 3)
F3 PLACE ENTRY POINT ADDRESS INTO OLD PSW FIELD OF SIRB
NO
F4 FETCH ERROR
YES
G4
G3 EXIT
TO DISPATCHER RTN (IEAODS) -- CHART GG --
ERFERR
G4 SET UP ERROR CODE 806
ERFERR
H4 ANY TASK ON TCB QUEUE IN MC STATUS
NO
YES
J4 SET MUST COMPLETE INDICATOR IN RB OLD PSW
NOTE-- REENTRY AT ERFETCH WILL OCCUR WHEN OPERATOR PRESSES RESET AND START KEYS
ERNMC
K4 LOAD WAIT STATE PSW

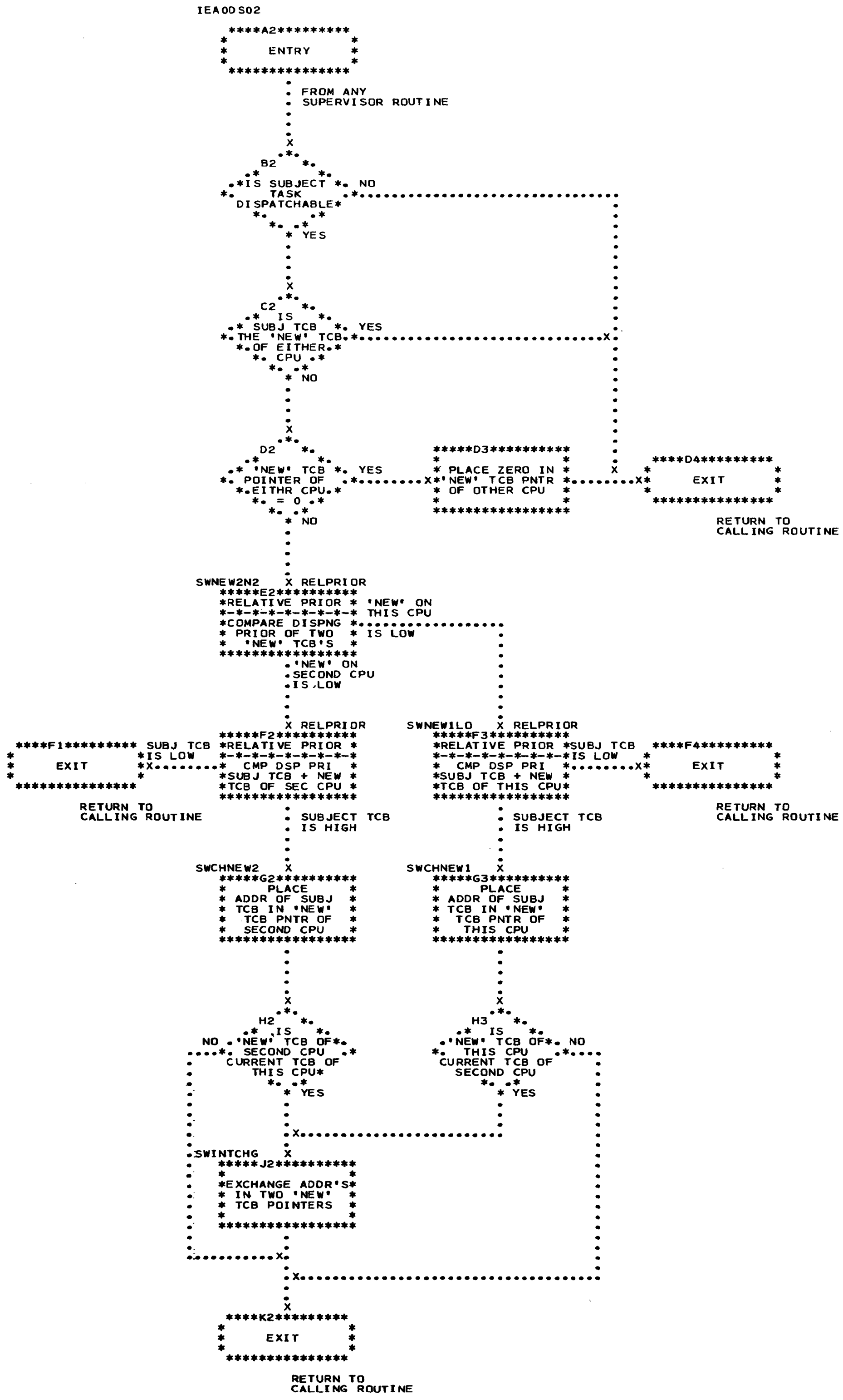
Chart BV. Task Switching Routine (Uniprocessing System)

NOTE - SHADED AREA APPLIES ONLY TO SYSTEMS WITH TIME-SLICING.

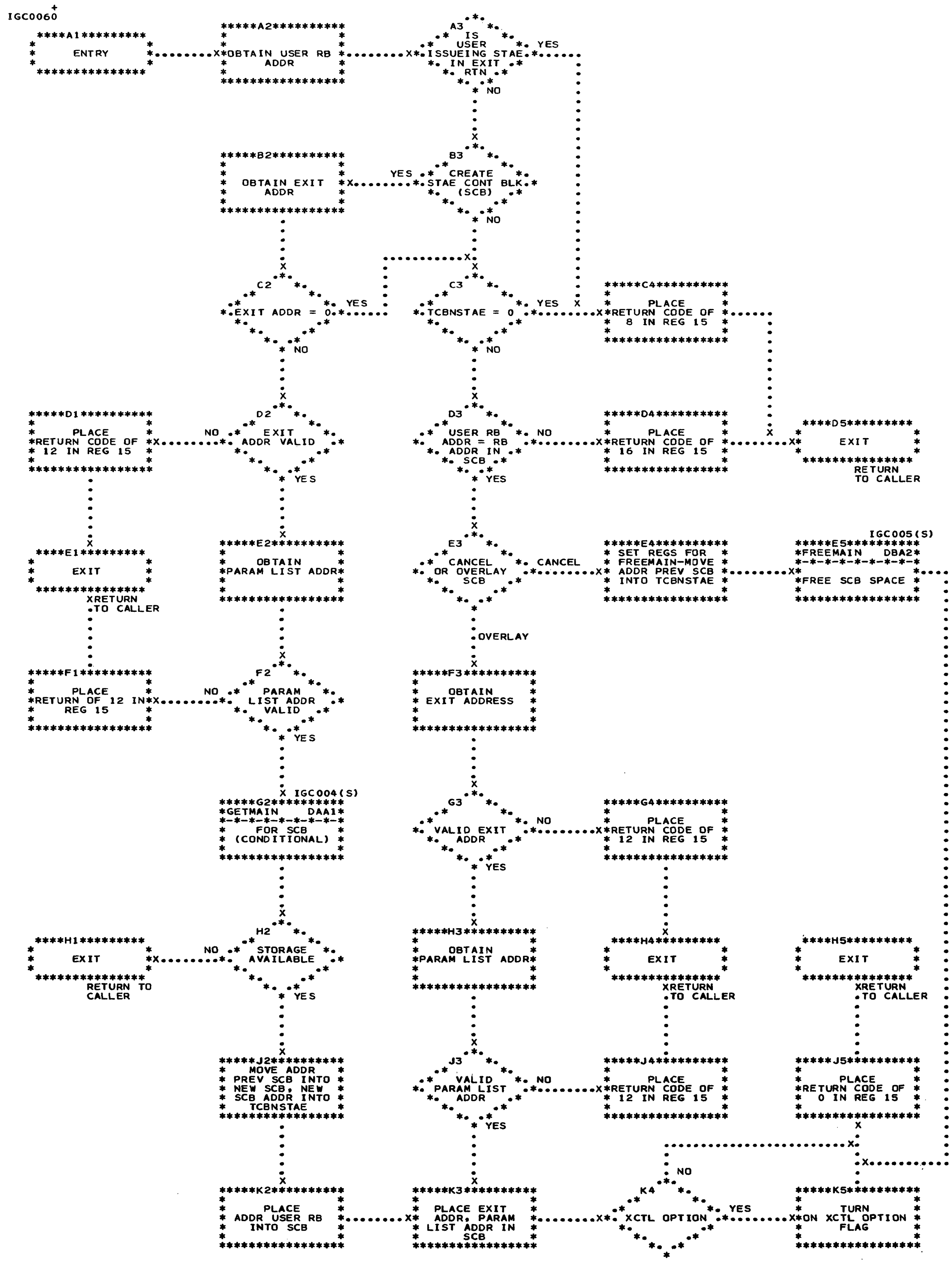

IEA0DS02
A2
ENTRY
FROM ANY SUPERVISOR ROUTINE
B2
NEW TCB PNTR (IEATCBP) = 0
YES
NO
B3
USE TCB ADDR IN OLD TCB PNTR (IEATCBP+4) FOR TEST
SW01
C2
COMPARE DISPCHG PRIORITY OF TCBS
SUBJECT TCB IS HIGH
SUBJECT TCB IS LOW
EQUAL PRIORITIES
C3
EXIT
RETURN TO CALLING ROUTINE
D2
IS SUBJECT TCB TIME-SLICED
YES
NO
D3
EXIT
RETURN TO CALLING ROUTINE
E2
SEARCH DOWN TCB QUEUE, STARTING WITH CURRENT TCB
F2
WAS SUBJECT INPUT TCB FOUND
YES
NO
F3
EXIT
RETURN TO CALLING ROUTINE
SWSETNEW
G2
IS SUBJECT TASK DIS-PATCHABLE
NO
YES
H2
INDICATE TASK SWITCH TO SUBJECT TASK
J2
EXIT
RETURN TO CALLING ROUTINE

NOTE-THE SUBJECT TASK IS REPRESENTED BY THE TCB WHOSE ADDRESS IS PASSED TO THIS ROUTINE.

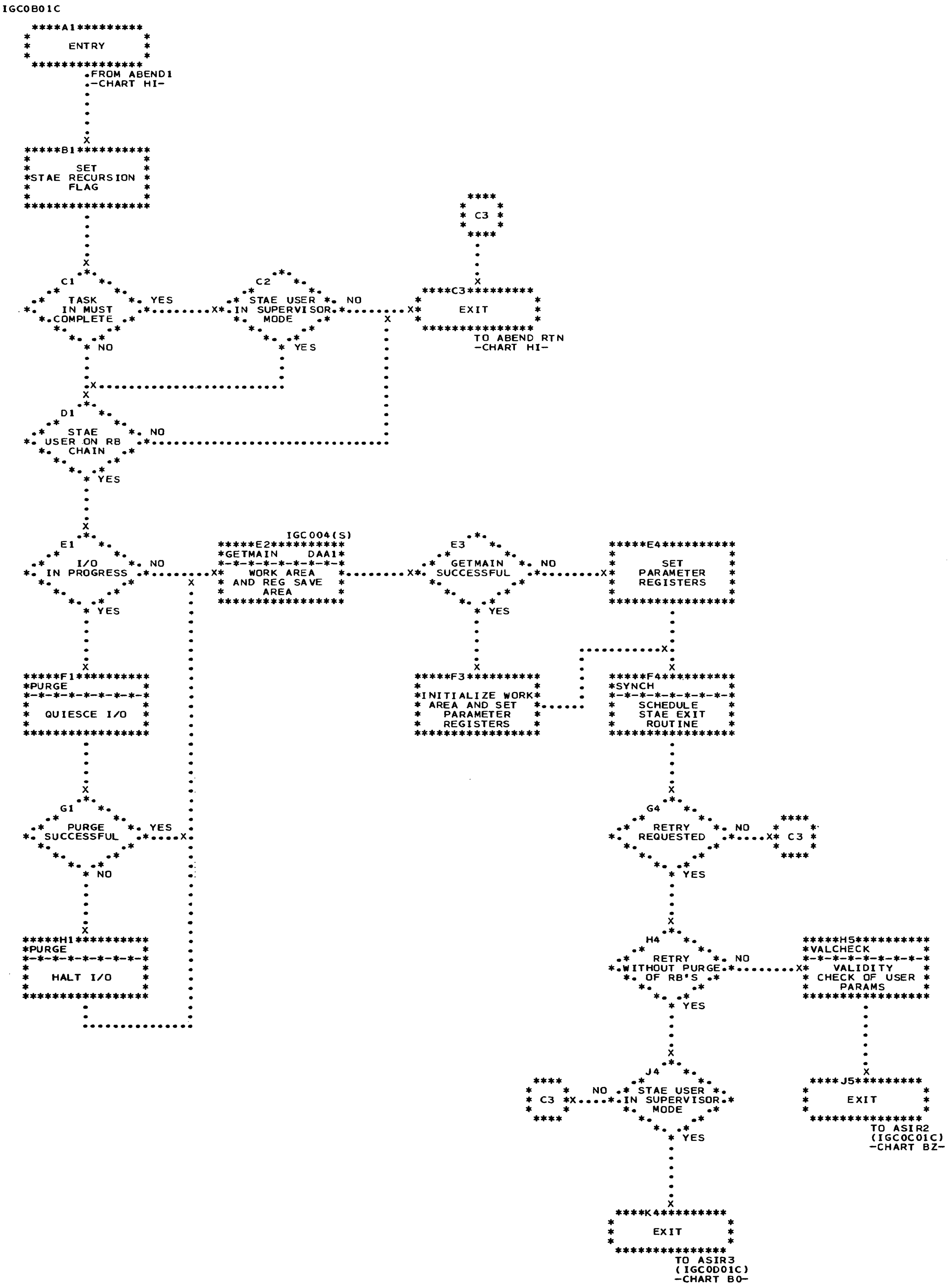
• Chart BW. Task Switching Routine (Multiprocessing System)


IEAODS02
A2 ENTRY
FROM ANY SUPERVISOR ROUTINE
B2 IS SUBJECT TASK DISPATCHABLE
NO
YES
C2 IS SUBJ TCB THE 'NEW' TCB OF EITHER CPU
YES
NO
D2 'NEW' TCB POINTER OF EITHR CPU = 0
YES
NO
D3 PLACE ZERO IN 'NEW' TCB PNTR OF OTHER CPU
D4 EXIT
RETURN TO CALLING ROUTINE
SWNEW2N2
RELPRIOR
E2 RELATIVE PRIOR
COMPARE DISPNG PRIOR OF TWO 'NEW' TCB'S
'NEW' ON THIS CPU IS LOW
'NEW' ON SECOND CPU IS LOW
RELPRIOR
F2 RELATIVE PRIOR
CMP DSP PRI SUBJ TCB + NEW TCB OF SEC CPU
SUBJ TCB IS LOW
F1 EXIT
RETURN TO CALLING ROUTINE
SWNEW1LO
RELPRIOR
F3 RELATIVE PRIOR
CMP DSP PRI SUBJ TCB + NEW TCB OF THIS CPU
SUBJ TCB IS LOW
F4 EXIT
RETURN TO CALLING ROUTINE
SUBJECT TCB IS HIGH
SUBJECT TCB IS HIGH
SWCHNEW2
G2 PLACE ADDR OF SUBJ TCB IN 'NEW' TCB PNTR OF SECOND CPU
SWCHNEW1
G3 PLACE ADDR OF SUBJ TCB IN 'NEW' TCB PNTR OF THIS CPU
H2 IS 'NEW' TCB OF SECOND CPU CURRENT TCB OF THIS CPU
NO
YES
H3 IS 'NEW' TCB OF THIS CPU CURRENT TCB OF SECOND CPU
NO
YES
SWINTCHG
J2 EXCHANGE ADDR'S IN TWO 'NEW' TCB POINTERS
K2 EXIT
RETURN TO CALLING ROUTINE

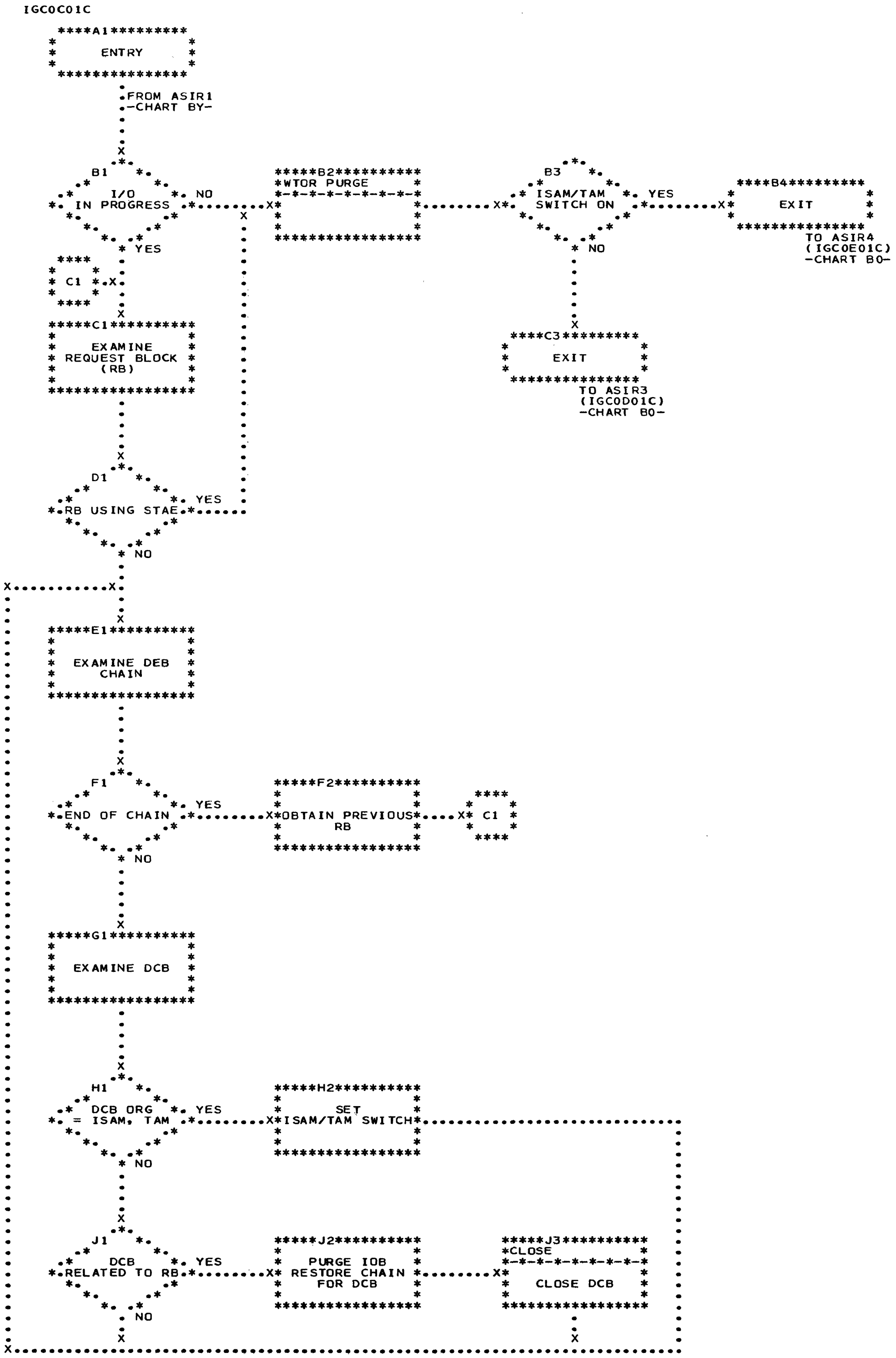
• Chart BX. STAE Service Routine



• Chart BY. ABEND/STAE Interface 1 Routine (ASIR1)



• Chart BZ. ABEND/STAE Interface 2 Routine (ASIR2)



• Chart B0. ABEND/STAE Interface 3 and 4 Routines (ASIR3, ASIR4)

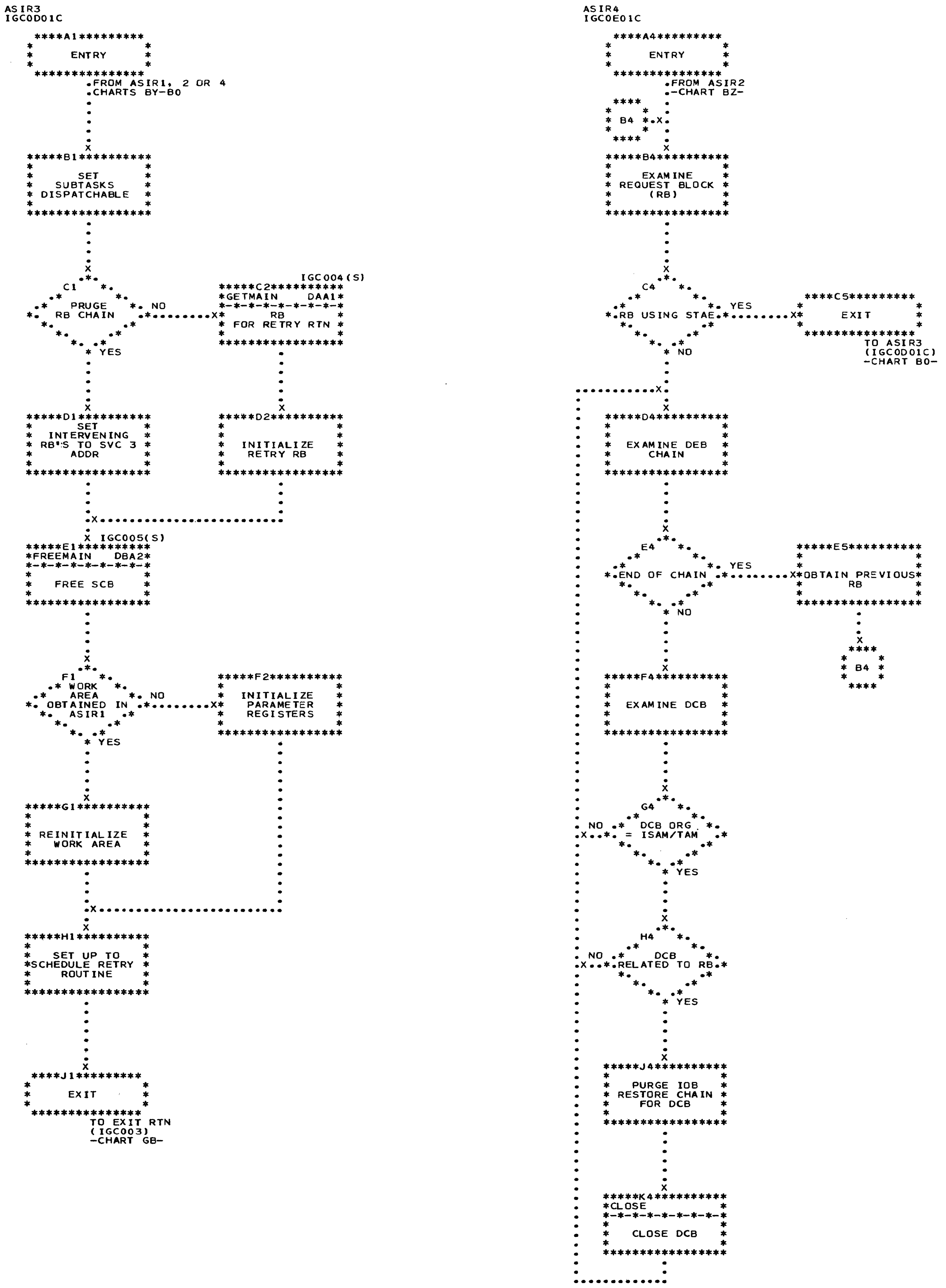
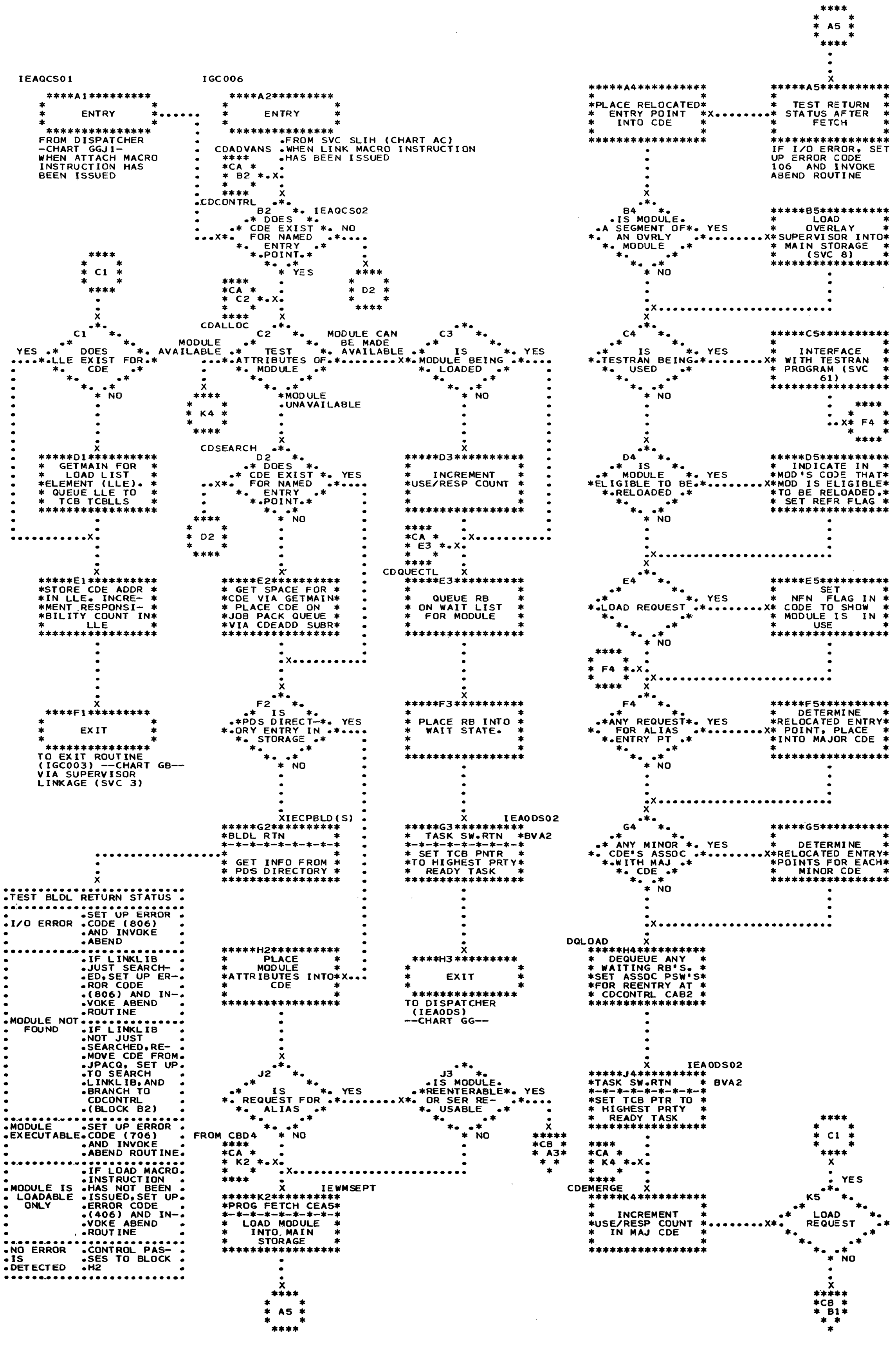
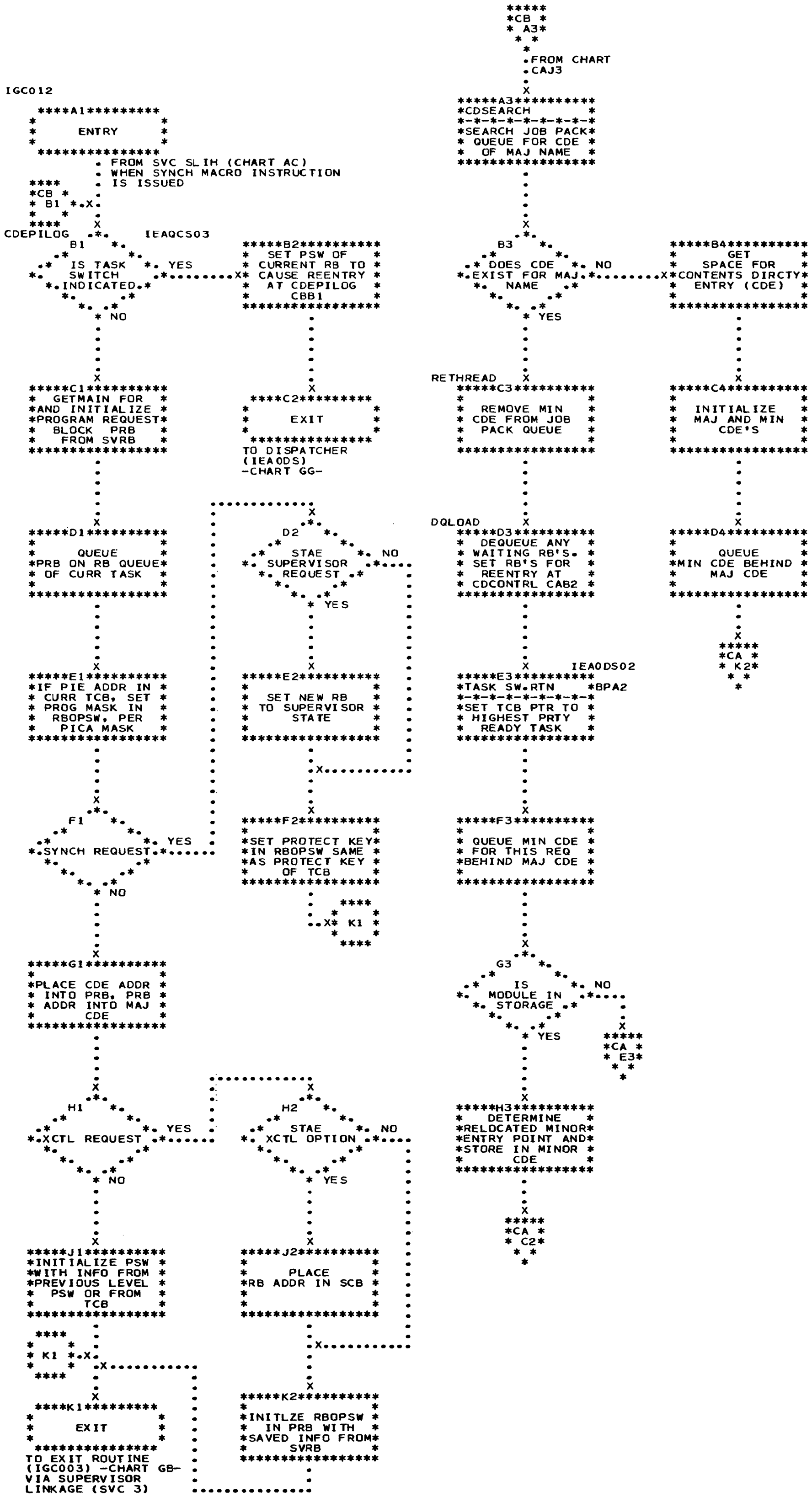


Chart CA. Link, Load, XCTL, and SYNCH Processing (Part 1 of 3)


A5
IEAQCS01
A1
ENTRY
FROM DISPATCHER -CHART GGJ1- WHEN ATTACH MACRO INSTRUCTION HAS BEEN ISSUED
IGC006
A2
ENTRY
FROM SVC SLIH (CHART AC) WHEN LINK MACRO INSTRUCTION HAS BEEN ISSUED
CDADVANS
CA B2
CDCONTRL
B2 IEAQCS02
DOES CDE EXIST FOR NAMED ENTRY POINT
NO
YES
D2
C1
CA C2
CDALLOC
C1
DOES LLE EXIST FOR CDE
YES
NO
C2
TEST ATTRIBUTES OF MODULE
MODULE AVAILABLE
MODULE CAN BE MADE AVAILABLE
MODULE UNAVAILABLE
K4
C3
IS MODULE BEING LOADED
YES
NO
D1
GETMAIN FOR LOAD LIST ELEMENT (LLE). QUEUE LLE TO TCB TCBLLS
CDSEARCH
D2
DOES CDE EXIST FOR NAMED ENTRY POINT
YES
NO
D2
D3
INCREMENT USE/RESP COUNT
CA E3
CDQUECTL
E1
STORE CDE ADDR IN LLE. INCREMENT RESPONSIBILITY COUNT IN LLE
E2
GET SPACE FOR CDE VIA GETMAIN PLACE CDE ON JOB PACK QUEUE VIA CDEADD SUBR
E3
QUEUE RB ON WAIT LIST FOR MODULE
F1
EXIT
TO EXIT ROUTINE (IGC003) --CHART GB-- VIA SUPERVISOR LINKAGE (SVC 3)
F2
IS PDS DIRECTORY ENTRY IN STORAGE
YES
NO
F3
PLACE RB INTO WAIT STATE.
XIECPBLD(S)
G2
BLDL RTN
GET INFO FROM PDS DIRECTORY
IEAODS02
G3
TASK SW.RTN
BVA2
SET TCB PNTR TO HIGHEST PRTY READY TASK
H2
PLACE MODULE ATTRIBUTES INTO CDE
H3
EXIT
TO DISPATCHER (IEAODS) --CHART GG--
J2
IS REQUEST FOR ALIAS
YES
NO
J3
IS MODULE REENTERABLE OR SER REUSABLE
YES
NO
CB A3
FROM CBD4
CA K2
IEWMSEPT
K2
PROG FETCH CEA5
LOAD MODULE INTO MAIN STORAGE
A5
A4
PLACE RELOCATED ENTRY POINT INTO CDE
A5
TEST RETURN STATUS AFTER FETCH
IF I/O ERROR, SET UP ERROR CODE 106 AND INVOKE ABEND ROUTINE
B4
IS MODULE A SEGMENT OF AN OVRLY MODULE
YES
NO
B5
LOAD OVERLAY SUPERVISOR INTO MAIN STORAGE (SVC 8)
C4
IS TESTRAN BEING USED
YES
NO
C5
INTERFACE WITH TESTRAN PROGRAM (SVC 61)
F4
D4
IS MODULE ELIGIBLE TO BE RELOADED
YES
NO
D5
INDICATE IN MOD'S CODE THAT MOD IS ELIGIBLE TO BE RELOADED, SET REFR FLAG
E4
LOAD REQUEST
YES
NO
E5
SET NFN FLAG IN CODE TO SHOW MODULE IS IN USE
F4
F4
ANY REQUEST FOR ALIAS ENTRY PT
YES
NO
F5
DETERMINE RELOCATED ENTRY POINT, PLACE INTO MAJOR CDE
G4
ANY MINOR CDE'S ASSOC WITH MAJ CDE
YES
NO
G5
DETERMINE RELOCATED ENTRY POINTS FOR EACH MINOR CDE
DQLOAD
H4
DEQUEUE ANY WAITING RB'S. SET ASSOC PSW'S FOR REENTRY AT CDCONTRL CAB2
IEAODS02
J4
TASK SW.RTN
BVA2
SET TCB PTR TO HIGHEST PRTY READY TASK
C1
CA K4
CDEMERGE
K4
INCREMENT USE/RESP COUNT IN MAJ CDE
YES
K5
LOAD REQUEST
NO
CB B1

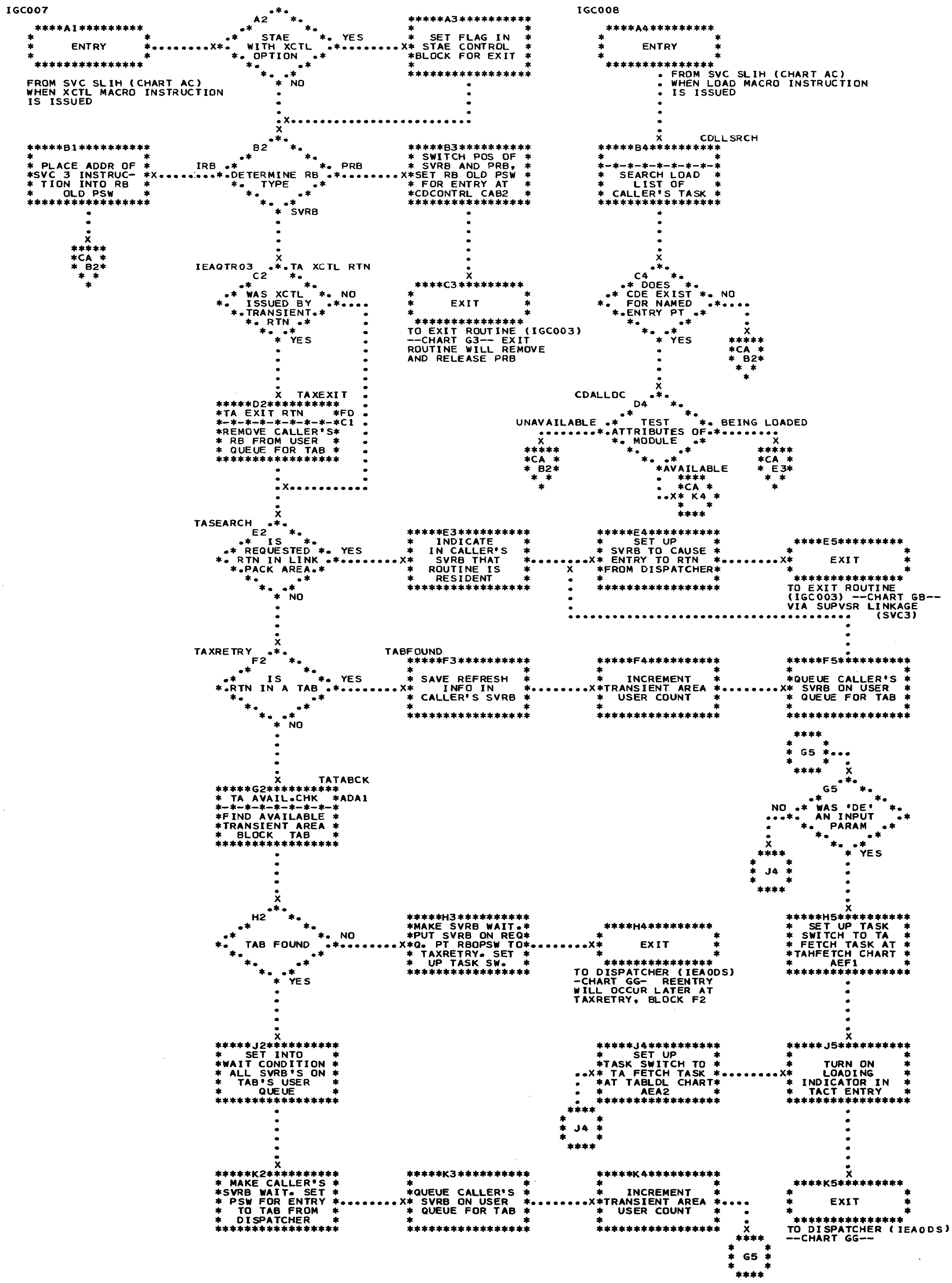
TEST BLDL RETURN STATUS	
I/O ERROR	SET UP ERROR CODE (806) AND INVOKE ABEND
MODULE NOT FOUND	IF LINKLIB JUST SEARCHED, SET UP ERROR CODE (806) AND INVOKE ABEND ROUTINE
	IF LINKLIB NOT JUST SEARCHED, REMOVE CDE FROM JPACQ, SET UP TO SEARCH LINKLIB, AND BRANCH TO CDCONTRL (BLOCK B2)
MODULE EXECUTABLE	SET UP ERROR CODE (706) AND INVOKE ABEND ROUTINE.
MODULE IS LOADABLE ONLY	IF LOAD MACRO INSTRUCTION HAS NOT BEEN ISSUED, SET UP ERROR CODE (406) AND INVOKE ABEND ROUTINE
NO ERROR IS DETECTED	CONTROL PASSES TO BLOCK H2

• Chart CB. Link, Load, XCTL, and SYNCH Processing (Part 2 of 3)



CB A3
FROM CHART CAJ3
IGC012
A1 ENTRY
FROM SVC SLIH (CHART AC) WHEN SYNCH MACRO INSTRUCTION IS ISSUED
CB B1
CDEPILOG
IEAQCS03
B1 IS TASK SWITCH INDICATED
YES
NO
B2 SET PSW OF CURRENT RB TO CAUSE REENTRY AT CDEPILOG CBB1
C1 GETMAIN FOR AND INITIALIZE PROGRAM REQUEST BLOCK PRB FROM SVRB
C2 EXIT
TO DISPATCHER (IEAODS) -CHART GG-
D1 QUEUE PRB ON RB QUEUE OF CURR TASK
D2 STAE SUPERVISOR REQUEST
NO
YES
E1 IF PIE ADDR IN CURR TCB, SET PROG MASK IN RBOPSW, PER PICA MASK
E2 SET NEW RB TO SUPERVISOR STATE
F1 SYNCH REQUEST
YES
NO
F2 SET PROTECT KEY IN RBOPSW SAME AS PROTECT KEY OF TCB
K1
G1 PLACE CDE ADDR INTO PRB, PRB ADDR INTO MAJ CDE
H1 XCTL REQUEST
YES
NO
H2 STAE XCTL OPTION
NO
YES
J1 INITIALIZE PSW WITH INFO FROM PREVIOUS LEVEL PSW OR FROM TCB
J2 PLACE RB ADDR IN SCB
K1
K1 EXIT
TO EXIT ROUTINE (IGC003) -CHART GB- VIA SUPERVISOR LINKAGE (SVC 3)
K2 INITLZE RBOPSW IN PRB WITH SAVED INFO FROM SVRB
A3 CDSEARCH
SEARCH JOB PACK QUEUE FOR CDE OF MAJ NAME
B3 DOES CDE EXIST FOR MAJ. NAME
NO
YES
B4 GET SPACE FOR CONTENTS DIRCTY ENTRY (CDE)
RETHREAD
C3 REMOVE MIN CDE FROM JOB PACK QUEUE
C4 INITIALIZE MAJ AND MIN CDE'S
DQLOAD
D3 DEQUEUE ANY WAITING RB'S. SET RB'S FOR REENTRY AT CDCONTRL CAB2
D4 QUEUE MIN CDE BEHIND MAJ CDE
CA K2
IEAODS02
E3 TASK SW.RTN BPA2
SET TCB PTR TO HIGHEST PRTY READY TASK
F3 QUEUE MIN CDE FOR THIS REQ BEHIND MAJ CDE
G3 IS MODULE IN STORAGE
NO
YES
CA E3
H3 DETERMINE RELOCATED MINOR ENTRY POINT AND STORE IN MINOR CDE
CA C2

• Chart CC. Link, Load, XCTL, and SYNCH Processing (Part 3 of 3)



IGC007
A1
ENTRY
FROM SVC SLIH (CHART AC) WHEN XCTL MACRO INSTRUCTION IS ISSUED
A2
STAE WITH XCTL OPTION
YES
NO
A3
SET FLAG IN STAE CONTROL BLOCK FOR EXIT
IGC008
A4
ENTRY
FROM SVC SLIH (CHART AC) WHEN LOAD MACRO INSTRUCTION IS ISSUED
B1
PLACE ADDR OF SVC 3 INSTRUCTION INTO RB OLD PSW
CA B2
B2
IRB
DETERMINE RB TYPE
PRB
SVRB
B3
SWITCH POS OF SVRB AND PRB. SET RB OLD PSW FOR ENTRY AT CDCONTRL CAB2
CDLLSRCH
B4
SEARCH LOAD LIST OF CALLER'S TASK
IEAQTR03
TA XCTL RTN
C2
WAS XCTL ISSUED BY TRANSIENT RTN
NO
YES
C3
EXIT
TO EXIT ROUTINE (IGC003) --CHART G3-- EXIT ROUTINE WILL REMOVE AND RELEASE PRB
C4
DOES CDE EXIST FOR NAMED ENTRY PT
NO
YES
CA B2
TAXEXIT
D2
TA EXIT RTN FD C1
REMOVE CALLER'S RB FROM USER QUEUE FOR TAB
CDALLOC
D4
UNAVAILABLE
TEST ATTRIBUTES OF MODULE
BEING LOADED
AVAILABLE
CA B2
CA E3
CA K4
TASEARCH
E2
IS REQUESTED RTN IN LINK PACK AREA
YES
NO
E3
INDICATE IN CALLER'S SVRB THAT ROUTINE IS RESIDENT
E4
SET UP SVRB TO CAUSE ENTRY TO RTN FROM DISPATCHER
E5
EXIT
TO EXIT ROUTINE (IGC003) --CHART G8-- VIA SUPVSR LINKAGE (SVC3)
TAXRETRY
F2
IS RTN IN A TAB
YES
NO
TABFOUND
F3
SAVE REFRESH INFO IN CALLER'S SVRB
F4
INCREMENT TRANSIENT AREA USER COUNT
F5
QUEUE CALLER'S SVRB ON USER QUEUE FOR TAB
G5
TATABCK
G2
TA AVAIL.CHK ADA1
FIND AVAILABLE TRANSIENT AREA BLOCK TAB
G5
WAS 'DE' AN INPUT PARAM
NO
YES
J4
H2
TAB FOUND
NO
YES
H3
MAKE SVRB WAIT. PUT SVRB ON REQ Q. PT RBOPSW TO TAXRETRY. SET UP TASK SW.
H4
EXIT
TO DISPATCHER (IEAODS) -CHART GG- REENTRY WILL OCCUR LATER AT TAXRETRY, BLOCK F2
H5
SET UP TASK SWITCH TO TA FETCH TASK AT TAHFETCH CHART AEF1
J2
SET INTO WAIT CONDITION ALL SVRB'S ON TAB'S USER QUEUE
J4
SET UP TASK SWITCH TO TA FETCH TASK AT TABLDL CHART AEA2
J4
J5
TURN ON LOADING INDICATOR IN TACT ENTRY
K2
MAKE CALLER'S SVRB WAIT. SET PSW FOR ENTRY TO TAB FROM DISPATCHER
K3
QUEUE CALLER'S SVRB ON USER QUEUE FOR TAB
K4
INCREMENT TRANSIENT AREA USER COUNT
G5
K5
EXIT
TO DISPATCHER (IEAODS) --CHART GG--

Chart CD. Identify Routine

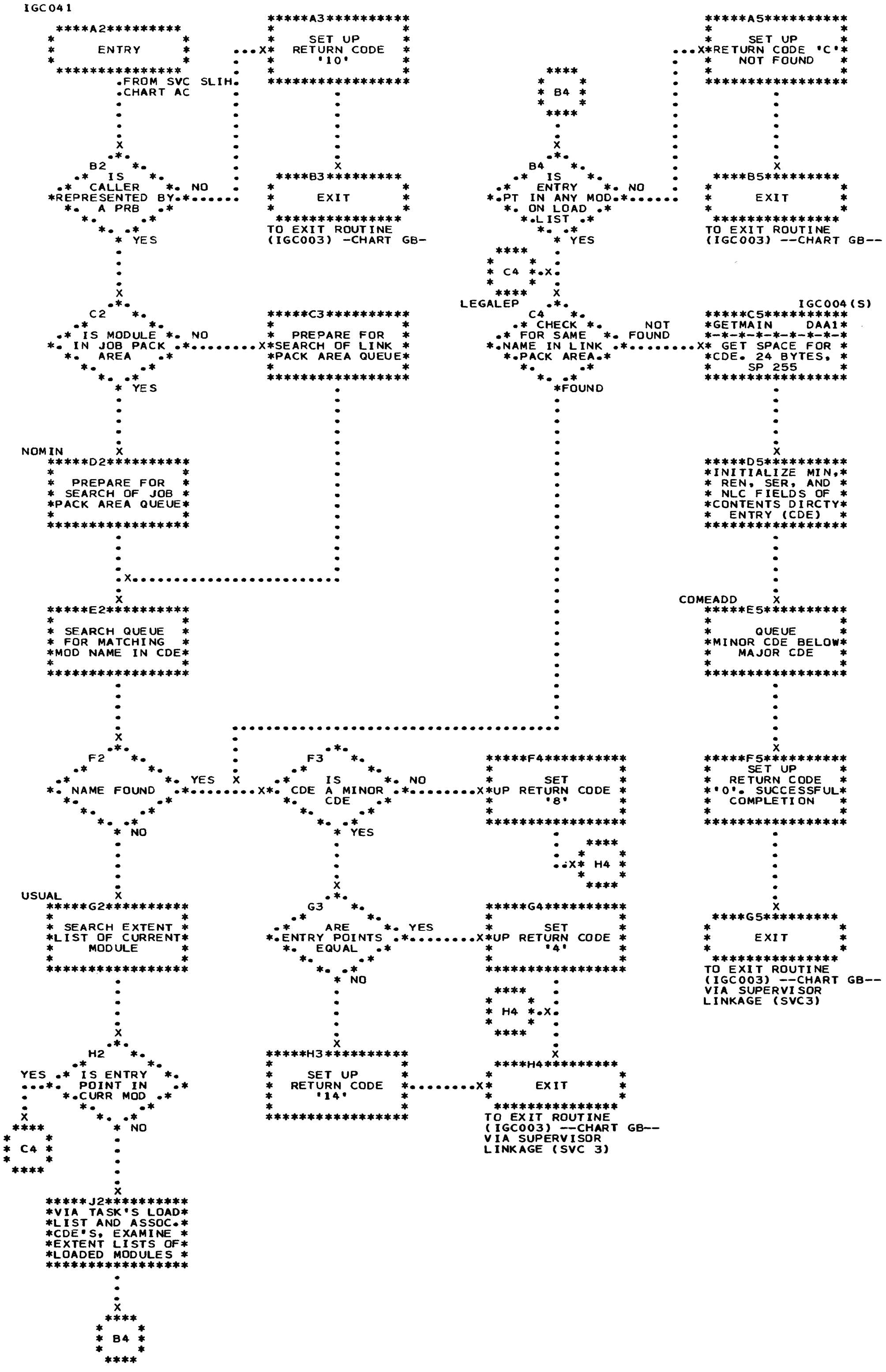
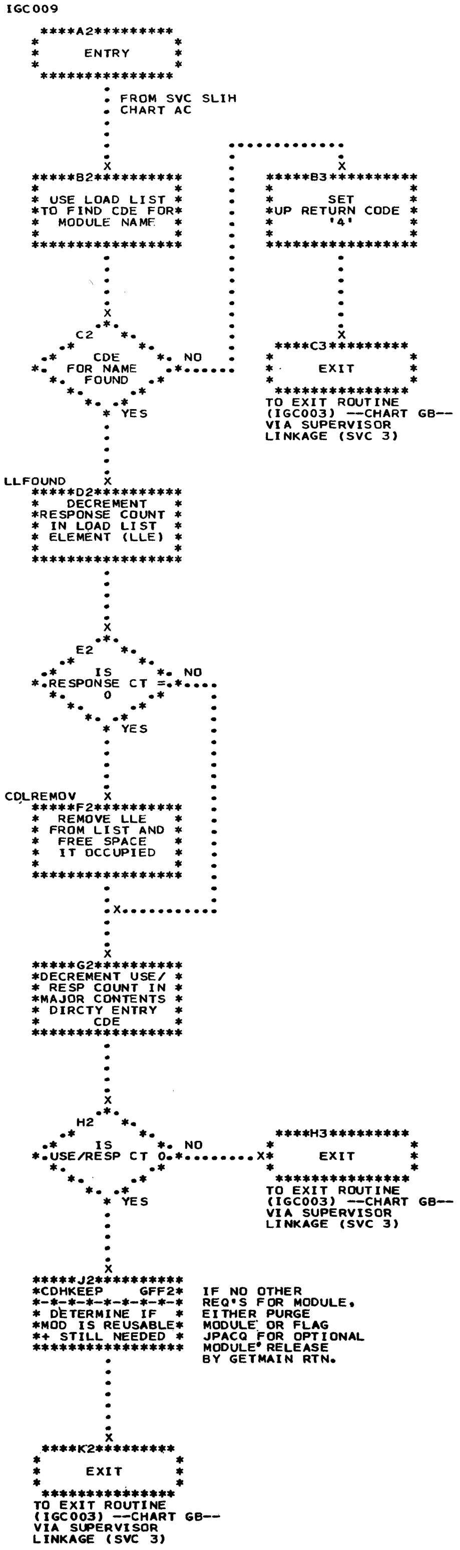
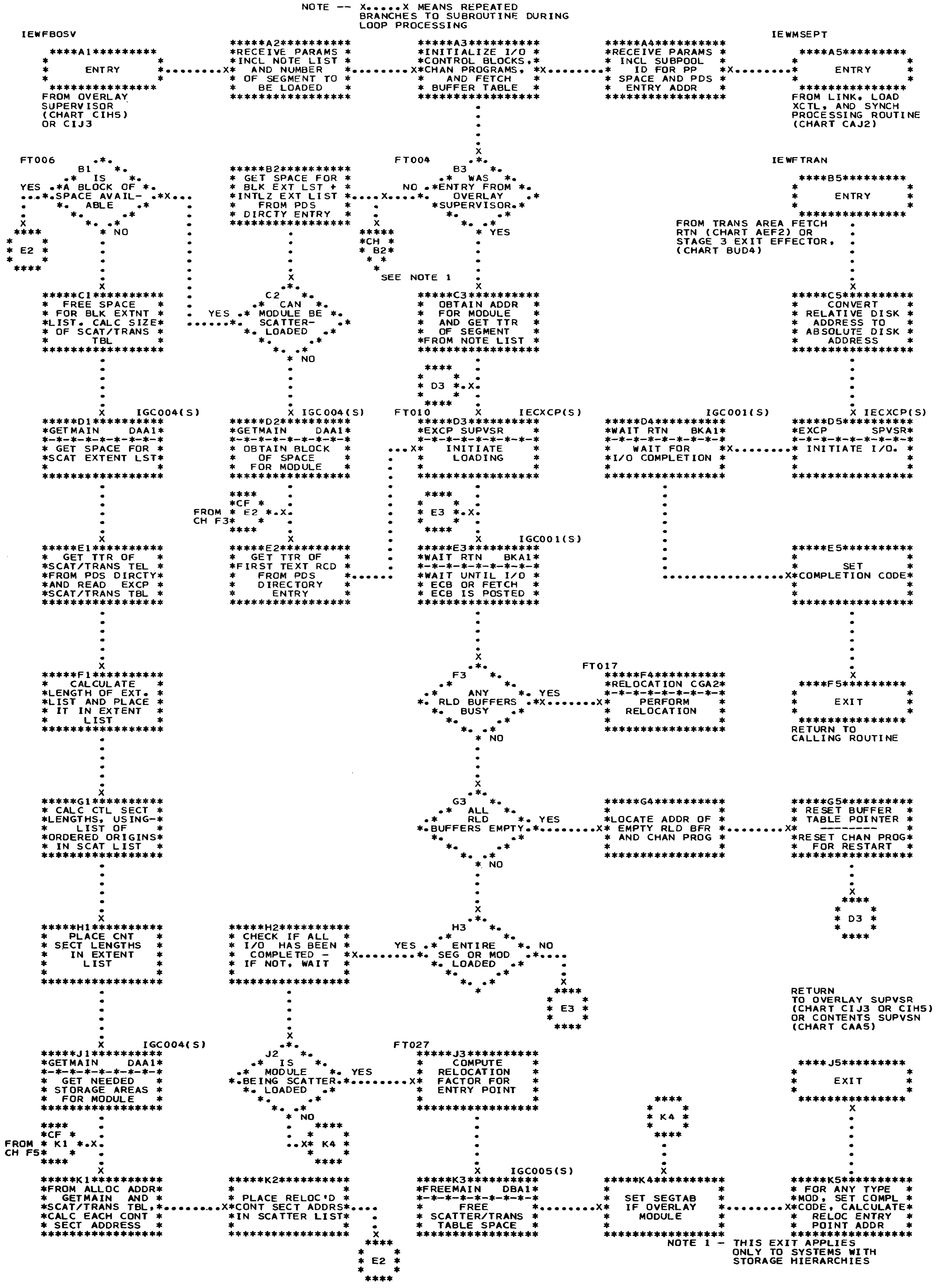

IGC041
A2 ENTRY
FROM SVC SLIH. CHART AC
A3 SET UP RETURN CODE '10'
B2 IS CALLER REPRESENTED BY A PRB
NO
YES
B3 EXIT
TO EXIT ROUTINE (IGC003) -CHART GB-
C2 IS MODULE IN JOB PACK AREA
NO
YES
C3 PREPARE FOR SEARCH OF LINK PACK AREA QUEUE
NOMIN
D2 PREPARE FOR SEARCH OF JOB PACK AREA QUEUE
E2 SEARCH QUEUE FOR MATCHING MOD NAME IN CDE
F2 NAME FOUND
YES
NO
F3 IS CDE A MINOR CDE
NO
YES
USUAL
G2 SEARCH EXTENT LIST OF CURRENT MODULE
G3 ARE ENTRY POINTS EQUAL
YES
NO
H2 IS ENTRY POINT IN CURR MOD
YES
NO
C4
H3 SET UP RETURN CODE '14'
J2 VIA TASK'S LOAD LIST AND ASSOC. CDE'S, EXAMINE EXTENT LISTS OF LOADED MODULES
B4
B4
B4 IS ENTRY PT IN ANY MOD. ON LOAD LIST
NO
YES
C4
LEGALEP
C4 CHECK FOR SAME NAME IN LINK PACK AREA
NOT FOUND
FOUND
F4 SET UP RETURN CODE '8'
H4
G4 SET UP RETURN CODE '4'
H4
H4 EXIT
TO EXIT ROUTINE (IGC003) --CHART GB-- VIA SUPERVISOR LINKAGE (SVC 3)
A5 SET UP RETURN CODE 'C' NOT FOUND
B5 EXIT
TO EXIT ROUTINE (IGC003) --CHART GB--
IGC004(S)
C5 GETMAIN DAA1
GET SPACE FOR CDE. 24 BYTES. SP 255
D5 INITIALIZE MIN, REN, SER, AND NLC FIELDS OF CONTENTS DIRCTY ENTRY (CDE)
COMEADD
E5 QUEUE MINOR CDE BELOW MAJOR CDE
F5 SET UP RETURN CODE '0'. SUCCESSFUL COMPLETION
G5 EXIT
TO EXIT ROUTINE (IGC003) --CHART GB-- VIA SUPERVISOR LINKAGE (SVC3)

Chart CE. Delete Routine

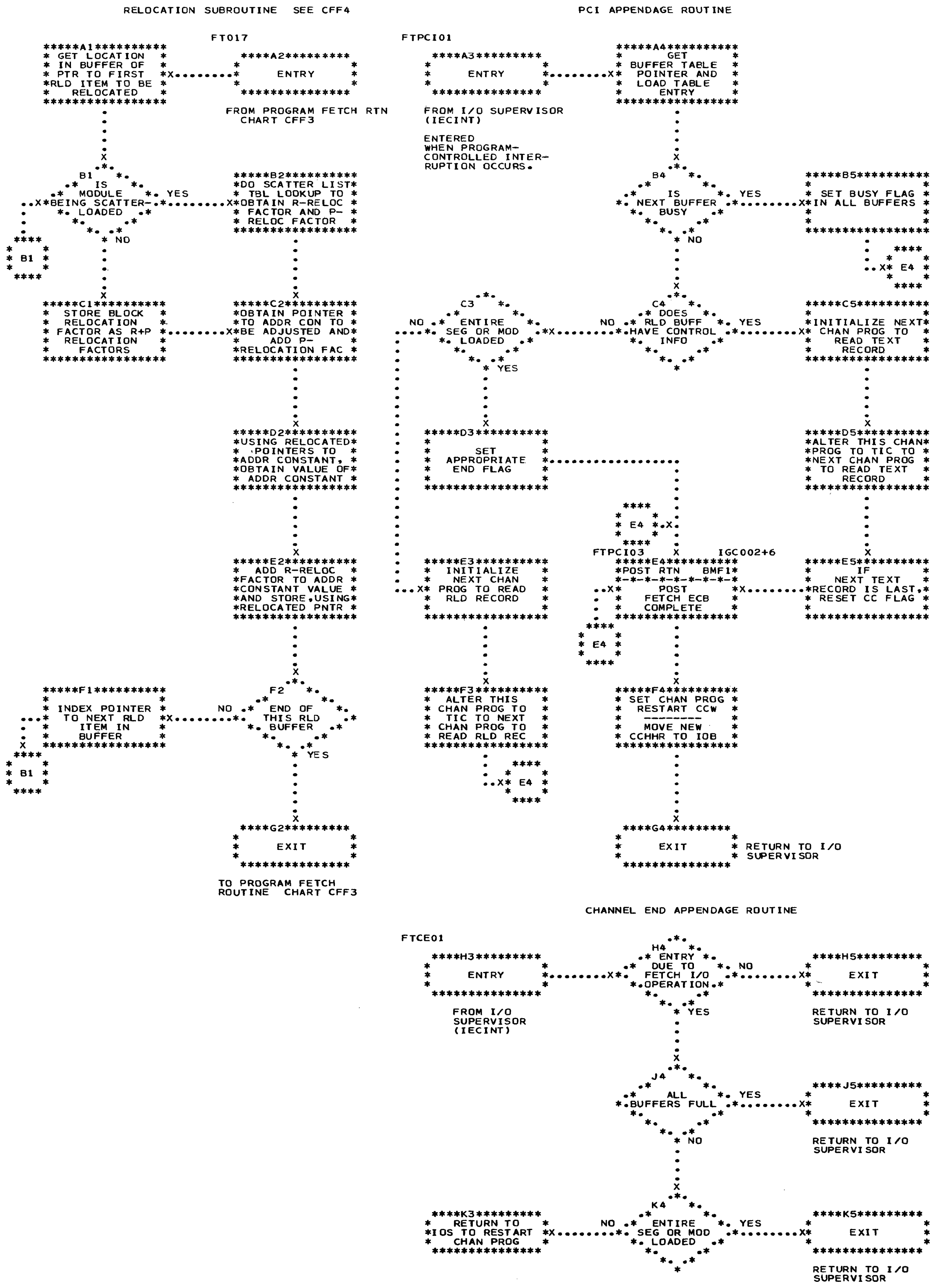

IGC009
A2
ENTRY
FROM SVC SLIH CHART AC
B2
USE LOAD LIST TO FIND CDE FOR MODULE NAME
B3
SET UP RETURN CODE '4'
C2
CDE FOR NAME FOUND
NO
YES
C3
EXIT
TO EXIT ROUTINE (IGC003) --CHART GB-- VIA SUPERVISOR LINKAGE (SVC 3)
LLFOUND
D2
DECREMENT RESPONSE COUNT IN LOAD LIST ELEMENT (LLE)
E2
IS RESPONSE CT = 0
NO
YES
CDLREMOV
F2
REMOVE LLE FROM LIST AND FREE SPACE IT OCCUPIED
G2
DECREMENT USE/ RESP COUNT IN MAJOR CONTENTS DIRCTY ENTRY CDE
H2
IS USE/RESP CT 0
NO
YES
H3
EXIT
TO EXIT ROUTINE (IGC003) --CHART GB-- VIA SUPERVISOR LINKAGE (SVC 3)
J2
CDHKEEP GFF2
DETERMINE IF MOD IS REUSABLE + STILL NEEDED
IF NO OTHER REQ'S FOR MODULE, EITHER PURGE MODULE OR FLAG JPACQ FOR OPTIONAL MODULE' RELEASE BY GETMAIN RTN.
K2
EXIT
TO EXIT ROUTINE (IGC003) --CHART GB-- VIA SUPERVISOR LINKAGE (SVC 3)

• Chart CF. Program Fetch Routine (Part 1 of 2)



NOTE -- X.....X MEANS REPEATED BRANCHES TO SUBROUTINE DURING LOOP PROCESSING
IEWFBOSV
A1 ENTRY
FROM OVERLAY SUPERVISOR (CHART CIH5) OR CIJ3
A2 RECEIVE PARAMS INCL NOTE LIST AND NUMBER OF SEGMENT TO BE LOADED
A3 INITIALIZE I/O CONTROL BLOCKS, CHAN PROGRAMS, AND FETCH BUFFER TABLE
A4 RECEIVE PARAMS INCL SUBPOOL ID FOR PP SPACE AND PDS ENTRY ADDR
IEWMSEPT
A5 ENTRY
FROM LINK, LOAD XCTL, AND SYNCH PROCESSING ROUTINE (CHART CAJ2)
FT006
B1 IS A BLOCK OF SPACE AVAILABLE
YES
NO
E2
B2 GET SPACE FOR BLK EXT LST + INTLZ EXT LIST FROM PDS DIRCTY ENTRY
FT004
B3 WAS ENTRY FROM OVERLAY SUPERVISOR
NO
YES
CH B2
SEE NOTE 1
IEWFTRAN
B5 ENTRY
FROM TRANS AREA FETCH RTN (CHART AEF2) OR STAGE 3 EXIT EFFECTOR, (CHART BUD4)
C1 FREE SPACE FOR BLK EXTNT LIST, CALC SIZE OF SCAT/TRANS TBL
C2 CAN MODULE BE SCATTER-LOADED
YES
NO
C3 OBTAIN ADDR FOR MODULE AND GET TTR OF SEGMENT FROM NOTE LIST
C5 CONVERT RELATIVE DISK ADDRESS TO ABSOLUTE DISK ADDRESS
D3
IGC004(S)
D1 GETMAIN DAA1 GET SPACE FOR SCAT EXTENT LST
IGC004(S)
D2 GETMAIN DAA1 OBTAIN BLOCK OF SPACE FOR MODULE
FT010
IECXCP(S)
D3 EXCP SUPVSR INITIATE LOADING
IGC001(S)
D4 WAIT RTN BKA1 WAIT FOR I/O COMPLETION
IECXCP(S)
D5 EXCP SPVSR INITIATE I/O.
CF E2
FROM CH F3
E3
IGC001(S)
E1 GET TTR OF SCAT/TRANS TBL FROM PDS DIRCTY AND READ EXCP SCAT/TRANS TBL
E2 GET TTR OF FIRST TEXT RCD FROM PDS DIRECTORY ENTRY
E3 WAIT RTN BKA1 WAIT UNTIL I/O ECB OR FETCH ECB IS POSTED
E5 SET COMPLETION CODE
F1 CALCULATE LENGTH OF EXT. LIST AND PLACE IT IN EXTENT LIST
F3 ANY RLD BUFFERS BUSY
YES
NO
FT017
F4 RELOCATION CGA2 PERFORM RELOCATION
F5 EXIT
RETURN TO CALLING ROUTINE
G1 CALC CTL SECT LENGTHS, USING LIST OF ORDERED ORIGINS IN SCAT LIST
G3 ALL RLD BUFFERS EMPTY
YES
NO
G4 LOCATE ADDR OF EMPTY RLD BFR AND CHAN PROG
G5 RESET BUFFER TABLE POINTER -------- RESET CHAN PROG FOR RESTART
D3
H1 PLACE CNT SECT LENGTHS IN EXTENT LIST
H2 CHECK IF ALL I/O HAS BEEN COMPLETED - IF NOT, WAIT
H3 ENTIRE SEG OR MOD LOADED
YES
NO
E3
RETURN TO OVERLAY SUPVSR (CHART CIJ3 OR CIH5) OR CONTENTS SUPVSN (CHART CAA5)
IGC004(S)
J1 GETMAIN DAA1 GET NEEDED STORAGE AREAS FOR MODULE
J2 IS MODULE BEING SCATTER-LOADED
YES
NO
FT027
J3 COMPUTE RELOCATION FACTOR FOR ENTRY POINT
J5 EXIT
CF K1
FROM CH F5
K4
K4
IGC005(S)
K1 FROM ALLOC ADDR GETMAIN AND SCAT/TRANS TBL, CALC EACH CONT SECT ADDRESS
K2 PLACE RELOC'D CONT SECT ADDRS IN SCATTER LIST
E2
K3 FREEMAIN DBA1 FREE SCATTER/TRANS TABLE SPACE
K4 SET SEGTAB IF OVERLAY MODULE
K5 FOR ANY TYPE MOD, SET COMPL CODE, CALCULATE RELOC ENTRY POINT ADDR
NOTE 1 - THIS EXIT APPLIES ONLY TO SYSTEMS WITH STORAGE HIERARCHIES

Chart CG. Program Fetch Routine (Part 2 of 2)


RELOCATION SUBROUTINE SEE CFF4
PCI APPENDAGE ROUTINE
A1 GET LOCATION IN BUFFER OF PTR TO FIRST RLD ITEM TO BE RELOCATED
FT017
A2 ENTRY
FROM PROGRAM FETCH RTN CHART CFF3
FTPCI01
A3 ENTRY
FROM I/O SUPERVISOR (IECINT)
ENTERED WHEN PROGRAM-CONTROLLED INTERRUPTION OCCURS.
A4 GET BUFFER TABLE POINTER AND LOAD TABLE ENTRY
B1 IS MODULE BEING SCATTER-LOADED
YES
NO
B1
B2 DO SCATTER LIST TBL LOOKUP TO OBTAIN R-RELOC FACTOR AND P-RELOC FACTOR
B4 IS NEXT BUFFER BUSY
B5 SET BUSY FLAG IN ALL BUFFERS
E4
C1 STORE BLOCK RELOCATION FACTOR AS R+P RELOCATION FACTORS
C2 OBTAIN POINTER TO ADDR CON TO BE ADJUSTED AND ADD P-RELOCATION FAC
C3 ENTIRE SEG OR MOD LOADED
C4 DOES RLD BUFF HAVE CONTROL INFO
C5 INITIALIZE NEXT CHAN PROG TO READ TEXT RECORD
D2 USING RELOCATED POINTERS TO ADDR CONSTANT, OBTAIN VALUE OF ADDR CONSTANT
D3 SET APPROPRIATE END FLAG
D5 ALTER THIS CHAN PROG TO TIC TO NEXT CHAN PROG TO READ TEXT RECORD
E2 ADD R-RELOC FACTOR TO ADDR CONSTANT VALUE AND STORE, USING RELOCATED PNTR
E3 INITIALIZE NEXT CHAN PROG TO READ RLD RECORD
FTPCI03
IGC002+6
E4 POST RTN BMF1 POST FETCH ECB COMPLETE
E5 IF NEXT TEXT RECORD IS LAST, RESET CC FLAG
F1 INDEX POINTER TO NEXT RLD ITEM IN BUFFER
F2 END OF THIS RLD BUFFER
F3 ALTER THIS CHAN PROG TO TIC TO NEXT CHAN PROG TO READ RLD REC
F4 SET CHAN PROG RESTART CCW MOVE NEW CCHHR TO IOB
G2 EXIT
TO PROGRAM FETCH ROUTINE CHART CFF3
G4 EXIT
RETURN TO I/O SUPERVISOR
CHANNEL END APPENDAGE ROUTINE
FTCE01
H3 ENTRY
FROM I/O SUPERVISOR (IECINT)
H4 ENTRY DUE TO FETCH I/O OPERATION
H5 EXIT
RETURN TO I/O SUPERVISOR
J4 ALL BUFFERS FULL
J5 EXIT
RETURN TO I/O SUPERVISOR
K3 RETURN TO IOS TO RESTART CHAN PROG
K4 ENTIRE SEG OR MOD LOADED
K5 EXIT
RETURN TO I/O SUPERVISOR

• Chart CH. Program Fetch Routine (Main Storage Hierarchies)

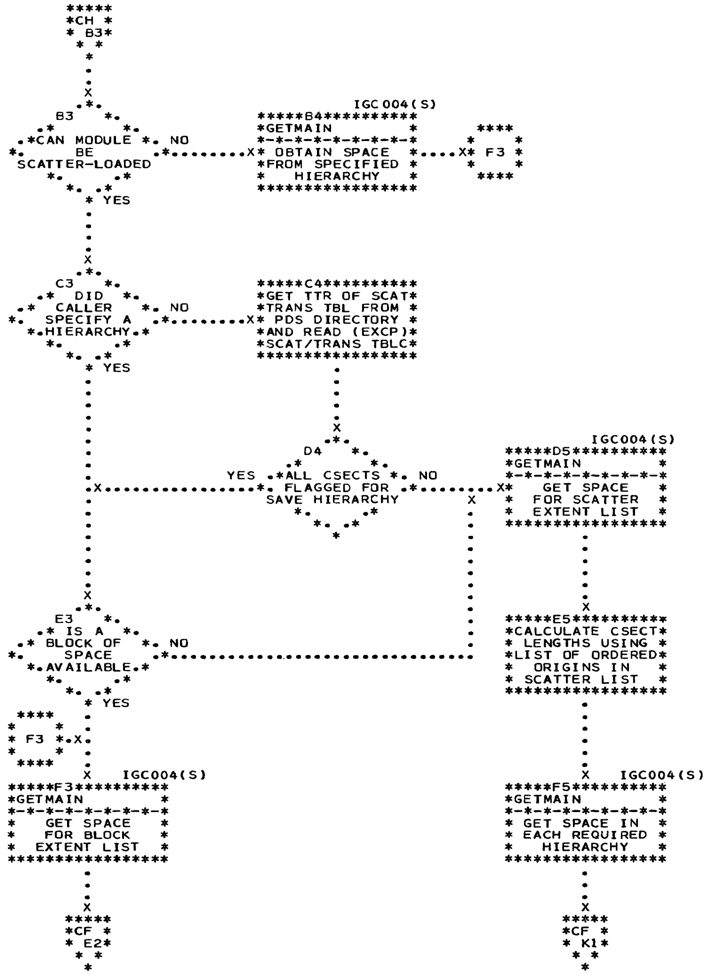
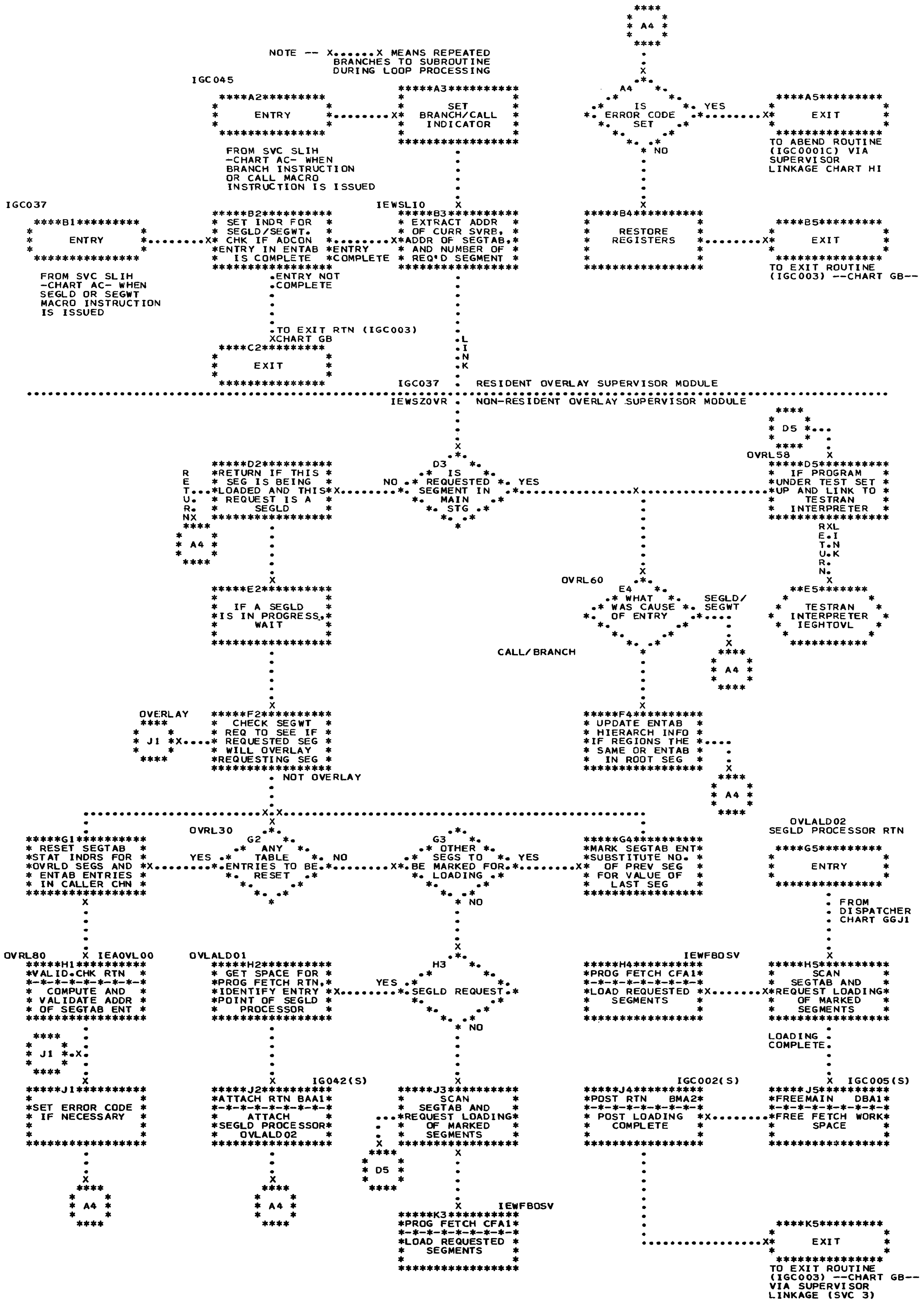
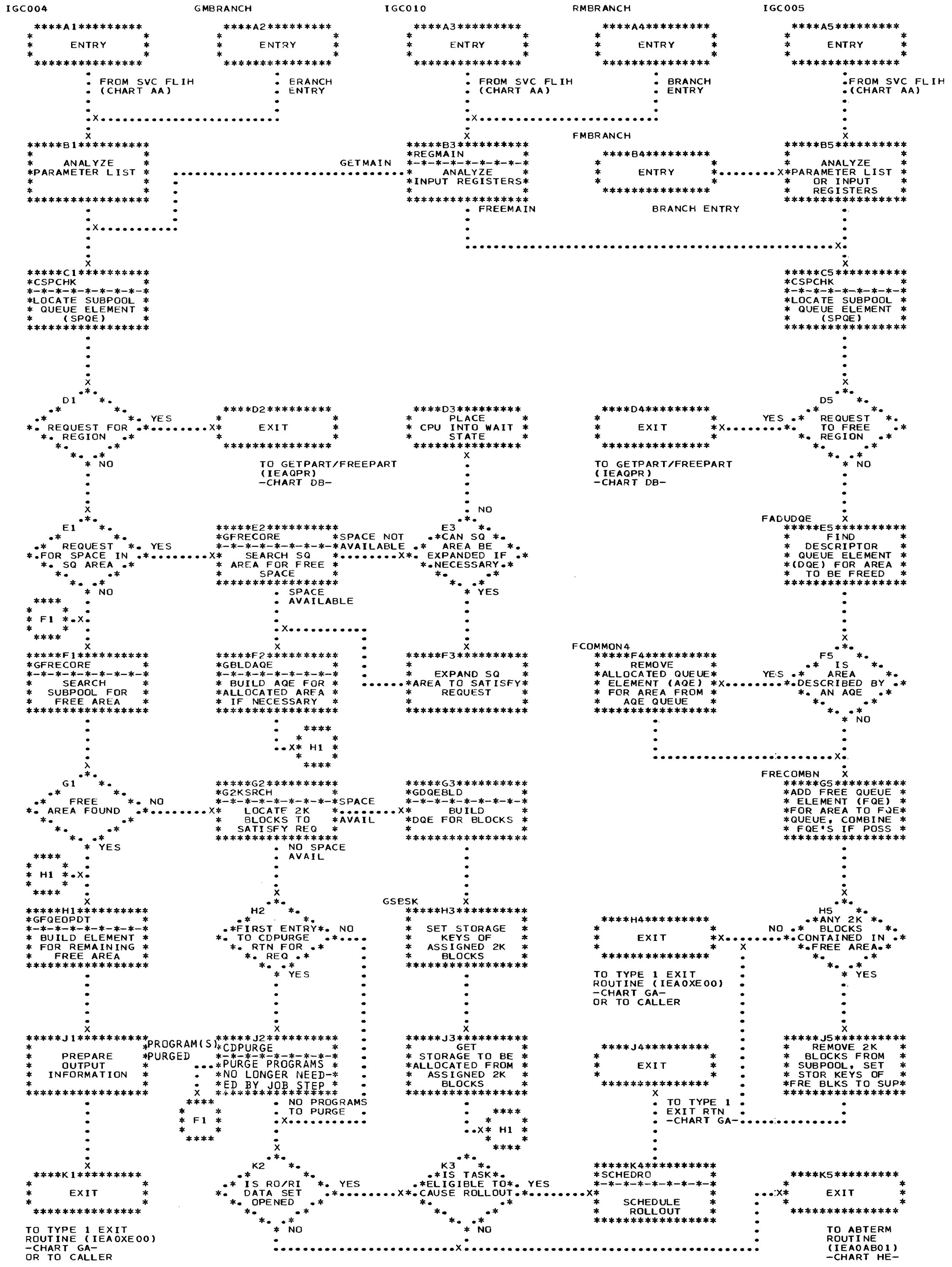

CH
B3
B3
CAN MODULE BE SCATTER-LOADED
NO
YES
IGC004(S)
B4
GETMAIN
OBTAIN SPACE FROM SPECIFIED HIERARCHY
F3
C3
DID CALLER SPECIFY A HIERARCHY
NO
YES
C4
GET TTR OF SCAT TRANS TBL FROM PDS DIRECTORY AND READ (EXCP) SCAT/TRANS TBLC
D4
ALL CSECTS FLAGGED FOR SAVE HIERARCHY
YES
NO
IGC004(S)
D5
GETMAIN
GET SPACE FOR SCATTER EXTENT LIST
E5
CALCULATE CSECT LENGTHS USING LIST OF ORDERED ORIGINS IN SCATTER LIST
E3
IS A BLOCK OF SPACE AVAILABLE
NO
YES
F3
IGC004(S)
F3
GETMAIN
GET SPACE FOR BLOCK EXTENT LIST
IGC004(S)
F5
GETMAIN
GET SPACE IN EACH REQUIRED HIERARCHY
CF
E2
CF
K1

Chart CI. Overlay Supervisor


A4
NOTE -- X......X MEANS REPEATED BRANCHES TO SUBROUTINE DURING LOOP PROCESSING
IGC045
A2 ENTRY
A3 SET BRANCH/CALL INDICATOR
A4 IS ERROR CODE SET
YES
NO
A5 EXIT
TO ABEND ROUTINE (IGC0001C) VIA SUPERVISOR LINKAGE CHART HI
FROM SVC SLIH -CHART AC- WHEN BRANCH INSTRUCTION OR CALL MACRO INSTRUCTION IS ISSUED
IGC037
B1 ENTRY
FROM SVC SLIH -CHART AC- WHEN SEGLD OR SEGWT MACRO INSTRUCTION IS ISSUED
B2 SET INDR FOR SEGLD/SEGWT. CHK IF ADCON ENTRY IN ENTAB IS COMPLETE
ENTRY COMPLETE
ENTRY NOT COMPLETE
IEWSLIO
B3 EXTRACT ADDR OF CURR SVRB, ADDR OF SEGTAB, AND NUMBER OF REQ'D SEGMENT
B4 RESTORE REGISTERS
B5 EXIT
TO EXIT ROUTINE (IGC003) --CHART GB--
TO EXIT RTN (IGC003) CHART GB
C2 EXIT
LINK
IGC037 RESIDENT OVERLAY SUPERVISOR MODULE
IEWSZOVR NON-RESIDENT OVERLAY SUPERVISOR MODULE
D5
OVRL58
D5 IF PROGRAM UNDER TEST SET UP AND LINK TO TESTRAN INTERPRETER
RXL E.I T.N U.K R. N.
E5 TESTRAN INTERPRETER IEGHTOVL
D2 RETURN IF THIS SEG IS BEING LOADED AND THIS REQUEST IS A SEGLD
RETURN
D3 IS REQUESTED SEGMENT IN MAIN STG
NO
YES
A4
E2 IF A SEGLD IS IN PROGRESS, WAIT
OVRL60
E4 WHAT WAS CAUSE OF ENTRY
SEGLD/ SEGWT
CALL/BRANCH
A4
OVERLAY
J1
F2 CHECK SEGWT REQ TO SEE IF REQUESTED SEG WILL OVERLAY REQUESTING SEG
NOT OVERLAY
F4 UPDATE ENTAB HIERARCH INFO IF REGIONS THE SAME OR ENTAB IN ROOT SEG
A4
OVRL30
G1 RESET SEGTAB STAT INDRS FOR OVRLD SEGS AND ENTAB ENTRIES IN CALLER CHN
G2 ANY TABLE ENTRIES TO BE RESET
YES
NO
G3 OTHER SEGS TO BE MARKED FOR LOADING
YES
NO
G4 MARK SEGTAB ENT SUBSTITUTE NO. OF PREV SEG FOR VALUE OF LAST SEG
OVLALD02 SEGLD PROCESSOR RTN
G5 ENTRY
FROM DISPATCHER CHART GGJ1
OVRL80
IEAOVL00
H1 VALID.CHK RTN COMPUTE AND VALIDATE ADDR OF SEGTAB ENT
OVLALD01
H2 GET SPACE FOR PROG FETCH RTN, IDENTIFY ENTRY POINT OF SEGLD PROCESSOR
H3 SEGLD REQUEST
YES
NO
IEWFBOSV
H4 PROG FETCH CFA1 LOAD REQUESTED SEGMENTS
H5 SCAN SEGTAB AND REQUEST LOADING OF MARKED SEGMENTS
LOADING COMPLETE
J1
J1 SET ERROR CODE IF NECESSARY
IG042(S)
J2 ATTACH RTN BAA1 ATTACH SEGLD PROCESSOR OVLALD02
J3 SCAN SEGTAB AND REQUEST LOADING OF MARKED SEGMENTS
D5
IGC002(S)
J4 POST RTN BMA2 POST LOADING COMPLETE
IGC005(S)
J5 FREEMAIN DBA1 FREE FETCH WORK SPACE
A4
A4
IEWFBOSV
K3 PROG FETCH CFA1 LOAD REQUESTED SEGMENTS
K5 EXIT
TO EXIT ROUTINE (IGC003) --CHART GB-- VIA SUPERVISOR LINKAGE (SVC 3)

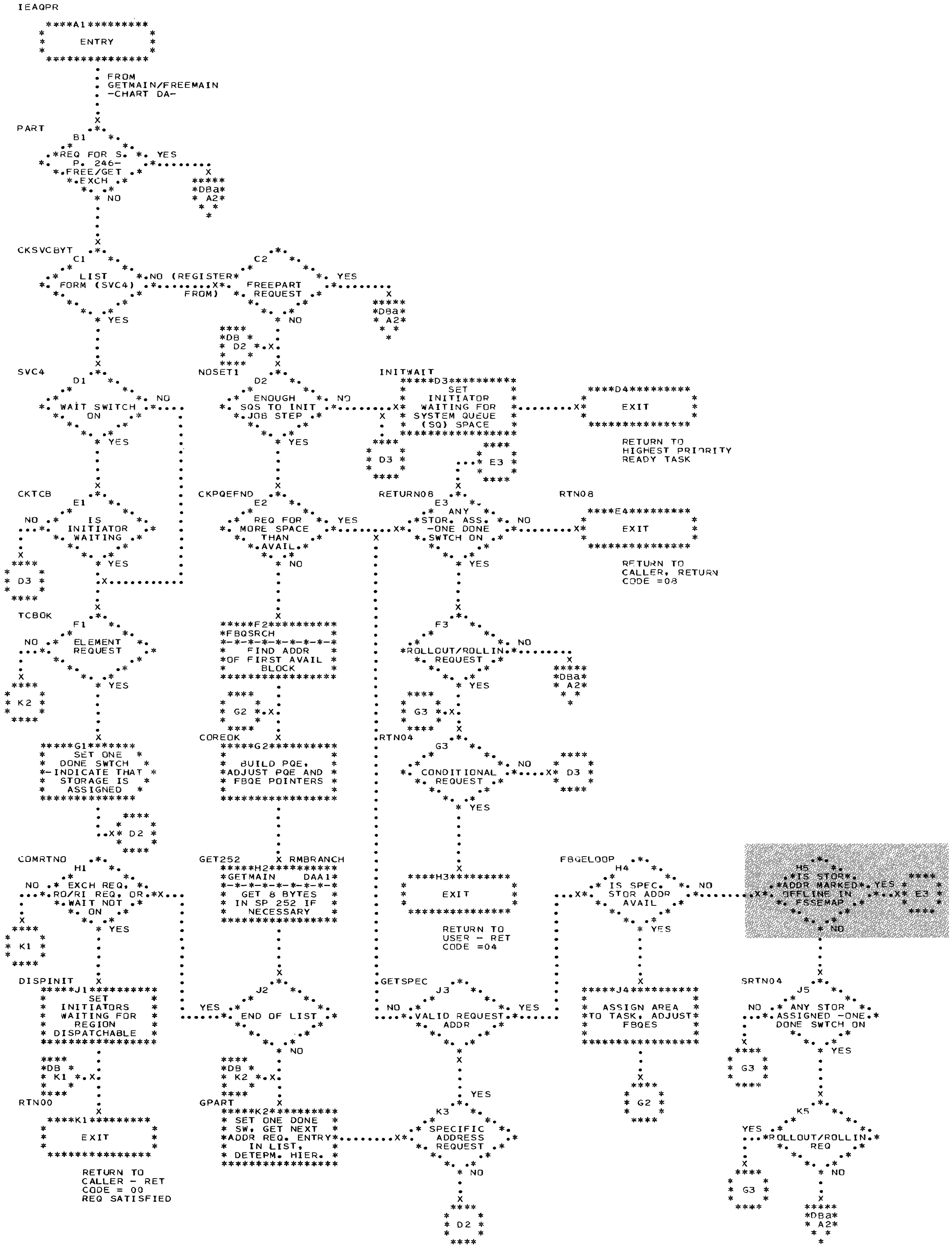
• Chart DA. GETMAIN/FREEMAIN Routine



IGC004
A1 ENTRY
FROM SVC FLIH (CHART AA)
GMBRANCH
A2 ENTRY
BRANCH ENTRY
IGC010
A3 ENTRY
FROM SVC FLIH (CHART AA)
RMBRANCH
A4 ENTRY
BRANCH ENTRY
IGC005
A5 ENTRY
FROM SVC FLIH (CHART AA)
B1 ANALYZE PARAMETER LIST
GETMAIN
B3 REGMAIN ANALYZE INPUT REGISTERS
FREEMAIN
FMBRANCH
B4 ENTRY
BRANCH ENTRY
B5 ANALYZE PARAMETER LIST OR INPUT REGISTERS
C1 CSPCHK LOCATE SUBPOOL QUEUE ELEMENT (SPQE)
C5 CSPCHK LOCATE SUBPOOL QUEUE ELEMENT (SPQE)
D1 REQUEST FOR REGION
YES
NO
D2 EXIT
TO GETPART/FREEPART (IEAQPR) -CHART DB-
D3 PLACE CPU INTO WAIT STATE
D4 EXIT
TO GETPART/FREEPART (IEAQPR) -CHART DB-
D5 REQUEST TO FREE REGION
YES
NO
E1 REQUEST FOR SPACE IN SQ AREA
YES
NO
E2 GFRECORE SEARCH SQ AREA FOR FREE SPACE
SPACE NOT AVAILABLE
SPACE AVAILABLE
E3 CAN SQ AREA BE EXPANDED IF NECESSARY
NO
YES
FADUDQE
E5 FIND DESCRIPTOR QUEUE ELEMENT (DQE) FOR AREA TO BE FREED
F1
F1 GFRECORE SEARCH SUBPOOL FOR FREE AREA
F2 GBLDAQE BUILD AQE FOR ALLOCATED AREA IF NECESSARY
H1
F3 EXPAND SQ AREA TO SATISFY REQUEST
FCOMMON4
F4 REMOVE ALLOCATED QUEUE ELEMENT (AQE) FOR AREA FROM AQE QUEUE
F5 IS AREA DESCRIBED BY AN AQE
YES
NO
G1 FREE AREA FOUND
NO
YES
G2 G2KSRCH LOCATE 2K BLOCKS TO SATISFY REQ
SPACE AVAIL
NO SPACE AVAIL
G3 GDQEBLD BUILD DQE FOR BLOCKS
FRECOMBN
G5 ADD FREE QUEUE ELEMENT (FQE) FOR AREA TO FQE QUEUE, COMBINE FQE'S IF POSS
H1
H1 GFQEOPDT BUILD ELEMENT FOR REMAINING FREE AREA
H2 FIRST ENTRY TO CDPURGE RTN FOR REQ
NO
YES
GSESK
H3 SET STORAGE KEYS OF ASSIGNED 2K BLOCKS
H4 EXIT
TO TYPE 1 EXIT ROUTINE (IEA0XE00) -CHART GA- OR TO CALLER
H5 ANY 2K BLOCKS CONTAINED IN FREE AREA
NO
YES
J1 PREPARE OUTPUT INFORMATION
PROGRAM(S) PURGED
J2 CDPURGE PURGE PROGRAMS NO LONGER NEEDED BY JOB STEP
F1
NO PROGRAMS TO PURGE
J3 GET STORAGE TO BE ALLOCATED FROM ASSIGNED 2K BLOCKS
H1
J4 EXIT
TO TYPE 1 EXIT RTN -CHART GA-
J5 REMOVE 2K BLOCKS FROM SUBPOOL, SET STOR KEYS OF FRE BLKS TO SUP
K1 EXIT
TO TYPE 1 EXIT ROUTINE (IEA0XE00) -CHART GA- OR TO CALLER
K2 IS RO/RI DATA SET OPENED
YES
NO
K3 IS TASK ELIGIBLE TO CAUSE ROLLOUT
YES
NO
K4 SCHEDRO SCHEDULE ROLLOUT
K5 EXIT
TO ABTERM ROUTINE (IEA0AB01) -CHART HE-

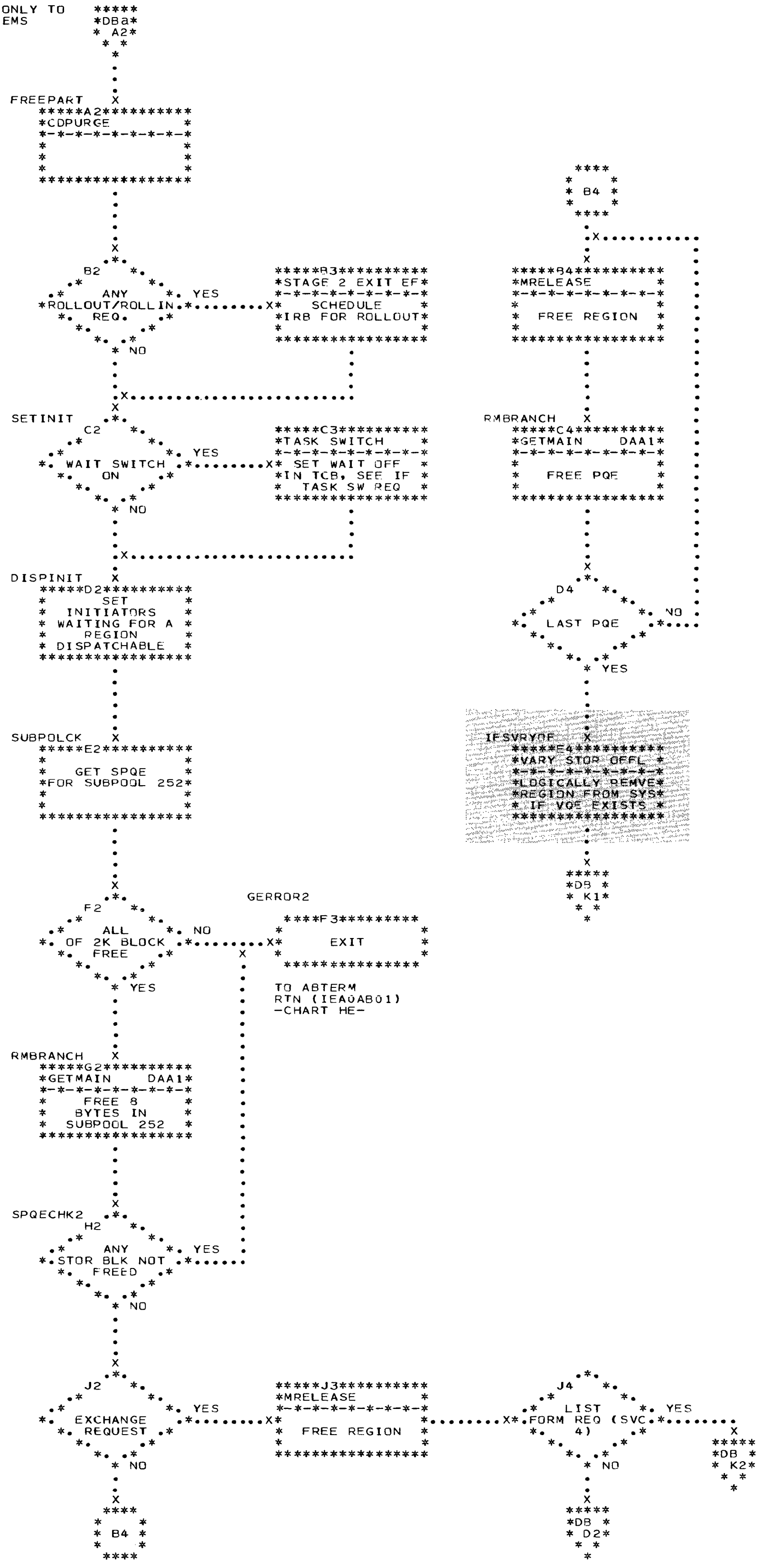
• Chart DB. GETPART/FREEPART Routine (Part 1 of 2)

NOTE - SHADED AREA APPLIES ONLY
TO MULTIPROCESSING SYSTEMS

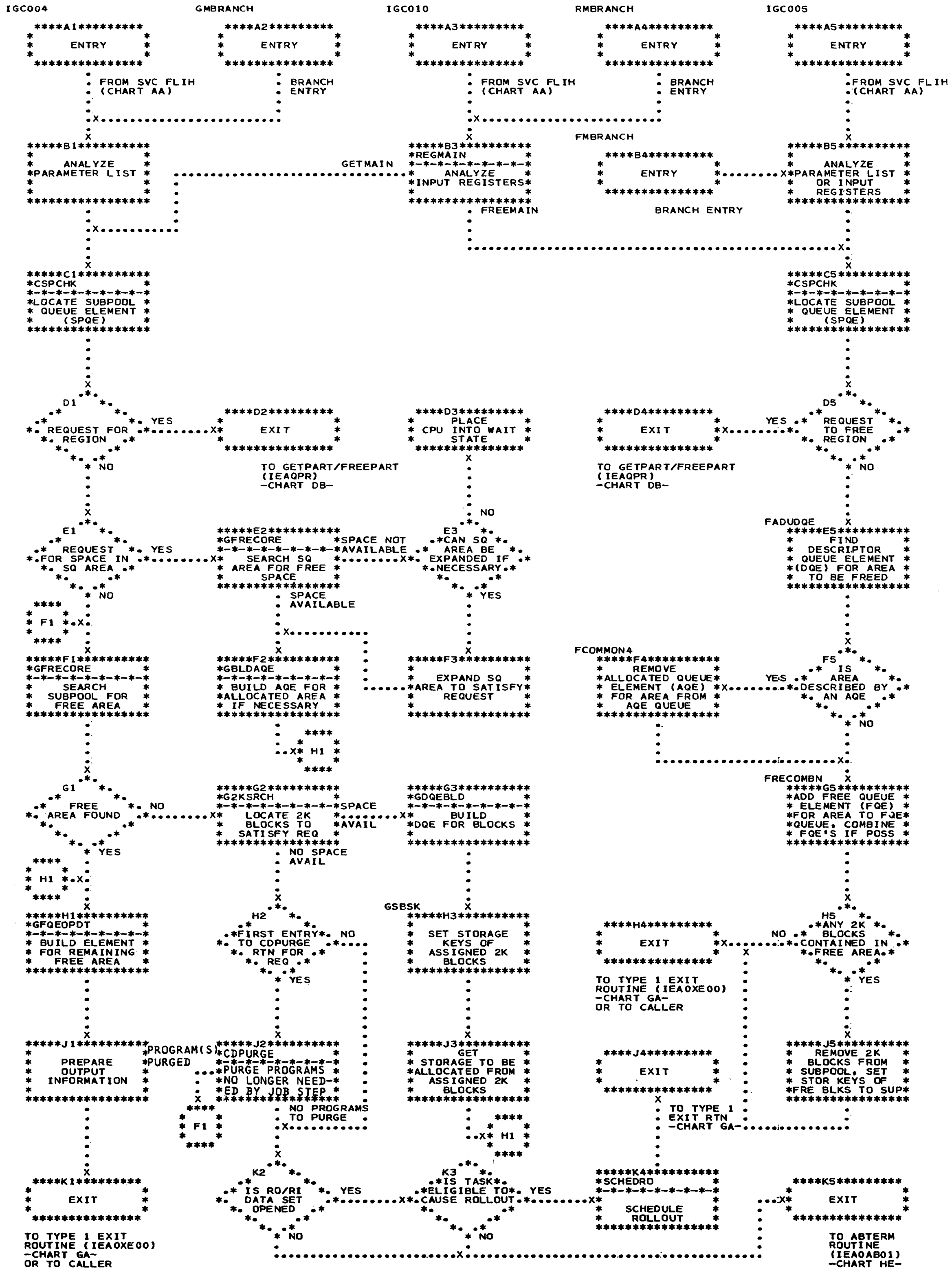

IEAQPR
A1
ENTRY
FROM GETMAIN/FREEMAIN -CHART DA-
PART
B1
REQ FOR S. P. 246- FREE/GET EXCH
YES
NO
DBa A2
CKSVCBYT
C1
LIST FORM (SVC4)
NO (REGISTER FROM)
C2
FREEPART REQUEST
YES
NO
DBa A2
YES
DB D2
SVC4
D1
WAIT SWITCH ON
NO
YES
NOSET1
D2
ENOUGH SQS TO INIT JOB STEP
NO
YES
INITWAIT
D3
SET INITIATOR WAITING FOR SYSTEM QUEUE (SQ) SPACE
D3
E3
D4
EXIT
RETURN TO HIGHEST PRIORITY READY TASK
CKTCB
E1
IS INITIATOR WAITING
NO
YES
D3
CKPQEFND
E2
REQ FOR MORE SPACE THAN AVAIL.
YES
NO
RETURN08
E3
ANY STOR. ASS. -ONE DONE SWTCH ON
NO
YES
RTN08
E4
EXIT
RETURN TO CALLER. RETURN CODE =08
TCBOK
F1
ELEMENT REQUEST
NO
YES
K2
F2
FBQSRCH
FIND ADDR OF FIRST AVAIL BLOCK
G2
F3
ROLLOUT/ROLLIN REQUEST
NO
YES
DBa A2
G3
G1
SET ONE DONE SWTCH -INDICATE THAT STORAGE IS ASSIGNED
D2
COREOK
G2
BUILD PQE. ADJUST PQE AND FBQE POINTERS
RTN04
G3
CONDITIONAL REQUEST
NO
D3
YES
COMRTNO
H1
EXCH REQ. RO/RI REQ. OR WAIT NOT ON
NO
YES
K1
GET252
RMBRANCH
H2
GETMAIN DAA1
GET 8 BYTES IN SP 252 IF NECESSARY
H3
EXIT
RETURN TO USER - RET CODE =04
FBQELOOP
H4
IS SPEC. STOR ADDR AVAIL
NO
YES
H5
IS STOR ADDR MARKED OFFLINE IN FSSEMAP
YES
E3
NO
DISPINIT
J1
SET INITIATORS WAITING FOR REGION DISPATCHABLE
DB K1
RTN00
K1
EXIT
RETURN TO CALLER - RET CODE = 00 REQ SATISFIED
J2
YES
END OF LIST
NO
DB K2
GPART
K2
SET ONE DONE SW. GET NEXT ADDR REQ. ENTRY IN LIST. DETERM. HIER.
GETSPEC
J3
NO
VALID REQUEST ADDR
YES
YES
K3
SPECIFIC ADDRESS REQUEST
NO
D2
J4
ASSIGN AREA TO TASK. ADJUST FBQES
G2
SRTN04
J5
NO
ANY STOR ASSIGNED -ONE DONE SWTCH ON
YES
G3
K5
YES
ROLLOUT/ROLLIN REQ
NO
G3
DBa A2

• Chart DBa. GETPART/FREEPART Routine (Part 2 of 2)

NOTE - SHADED AREA APPLIES ONLY TO MULTIPROCESSING SYSTEMS



• Chart DA. GETMAIN/FREEMAIN Routine


IGC004
A1 ENTRY
FROM SVC FLIH (CHART AA)
GMBRANCH
A2 ENTRY
BRANCH ENTRY
IGC010
A3 ENTRY
FROM SVC FLIH (CHART AA)
RMBRANCH
A4 ENTRY
BRANCH ENTRY
IGC005
A5 ENTRY
FROM SVC FLIH (CHART AA)
B1 ANALYZE PARAMETER LIST
GETMAIN
B3 REGMAIN ANALYZE INPUT REGISTERS
FREEMAIN
FMBRANCH
B4 ENTRY
BRANCH ENTRY
B5 ANALYZE PARAMETER LIST OR INPUT REGISTERS
C1 CSPCHK LOCATE SUBPOOL QUEUE ELEMENT (SPQE)
C5 CSPCHK LOCATE SUBPOOL QUEUE ELEMENT (SPQE)
D1 REQUEST FOR REGION
YES
NO
D2 EXIT
TO GETPART/FREEPART (IEAQPR) -CHART DB-
D3 PLACE CPU INTO WAIT STATE
D4 EXIT
TO GETPART/FREEPART (IEAQPR) -CHART DB-
D5 REQUEST TO FREE REGION
YES
NO
E1 REQUEST FOR SPACE IN SQ AREA
YES
NO
E2 GFRECORE SEARCH SQ AREA FOR FREE SPACE
SPACE NOT AVAILABLE
SPACE AVAILABLE
E3 CAN SQ AREA BE EXPANDED IF NECESSARY
NO
YES
FADUDQE
E5 FIND DESCRIPTOR QUEUE ELEMENT (DQE) FOR AREA TO BE FREED
F1
F1 GFRECORE SEARCH SUBPOOL FOR FREE AREA
F2 GBLDAQE BUILD AQE FOR ALLOCATED AREA IF NECESSARY
F3 EXPAND SQ AREA TO SATISFY REQUEST
FCOMMON4
F4 REMOVE ALLOCATED QUEUE ELEMENT (AQE) FOR AREA FROM AQE QUEUE
F5 IS AREA DESCRIBED BY AN AQE
YES
NO
H1
G1 FREE AREA FOUND
NO
YES
G2 G2KSRCH LOCATE 2K BLOCKS TO SATISFY REQ
SPACE AVAIL
NO SPACE AVAIL
G3 GDQEBLD BUILD DQE FOR BLOCKS
FRECOMBN
G5 ADD FREE QUEUE ELEMENT (FQE) FOR AREA TO FQE QUEUE, COMBINE FQE'S IF POSS
H1
H1 GFQEOPDT BUILD ELEMENT FOR REMAINING FREE AREA
H2 FIRST ENTRY TO CDPURGE RTN FOR REQ
NO
YES
GSBSK
H3 SET STORAGE KEYS OF ASSIGNED 2K BLOCKS
H4 EXIT
TO TYPE 1 EXIT ROUTINE (IEA0XE00) -CHART GA- OR TO CALLER
H5 ANY 2K BLOCKS CONTAINED IN FREE AREA
NO
YES
J1 PREPARE OUTPUT INFORMATION
PROGRAM(S) PURGED
J2 CDPURGE PURGE PROGRAMS NO LONGER NEEDED BY JOB STEP
F1
NO PROGRAMS TO PURGE
J3 GET STORAGE TO BE ALLOCATED FROM ASSIGNED 2K BLOCKS
H1
J4 EXIT
TO TYPE 1 EXIT RTN -CHART GA-
J5 REMOVE 2K BLOCKS FROM SUBPOOL, SET STOR KEYS OF FRE BLKS TO SUP
K1 EXIT
TO TYPE 1 EXIT ROUTINE (IEA0XE00) -CHART GA- OR TO CALLER
K2 IS RO/RI DATA SET OPENED
YES
NO
K3 IS TASK ELIGIBLE TO CAUSE ROLLOUT
YES
NO
K4 SCHEDRO SCHEDULE ROLLOUT
K5 EXIT
TO ABTERM ROUTINE (IEA0AB01) -CHART HE-

• Chart DB. GETPART/FREEPART Routine

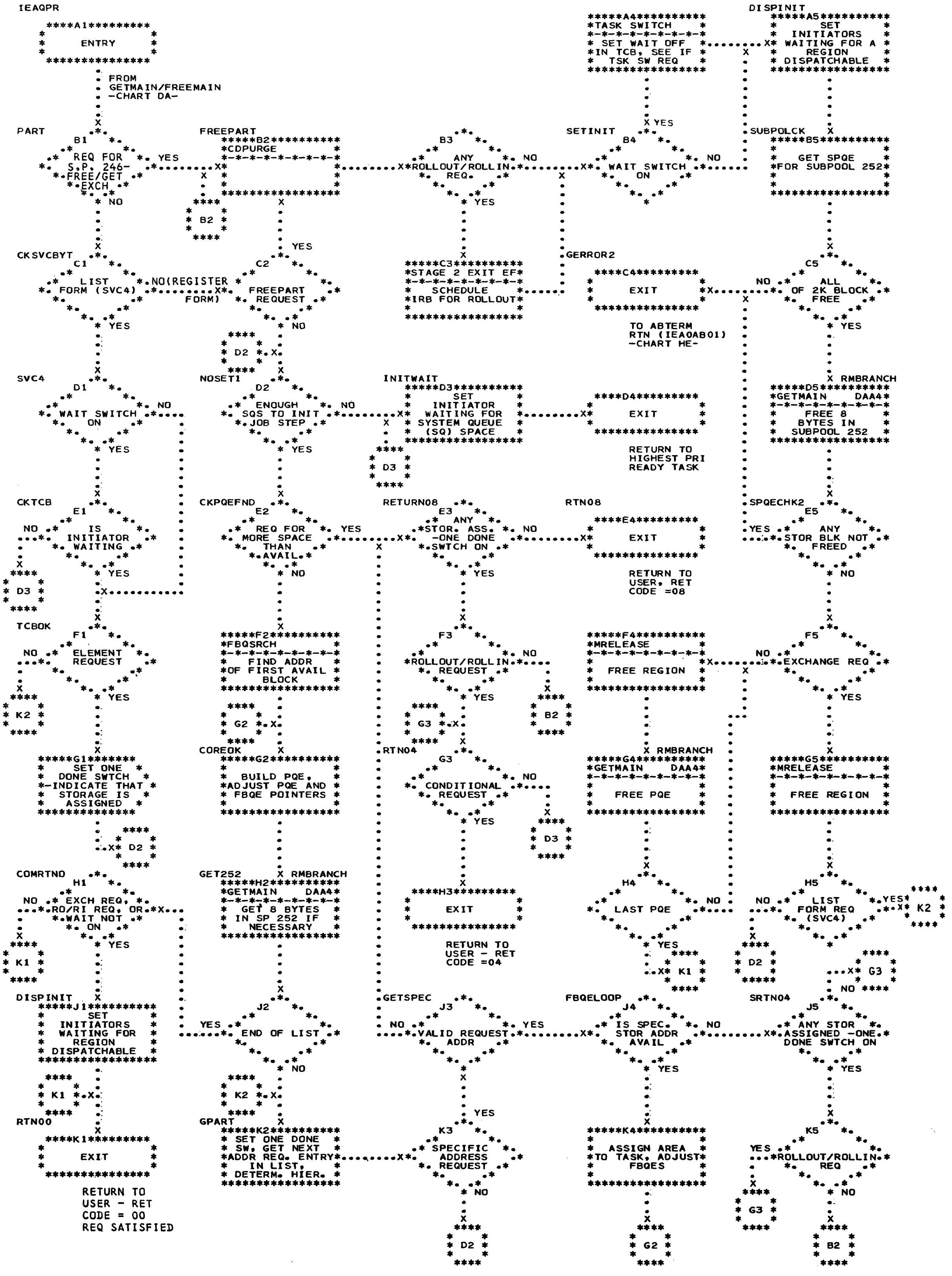


Chart DC. Rollout Criterion Routine

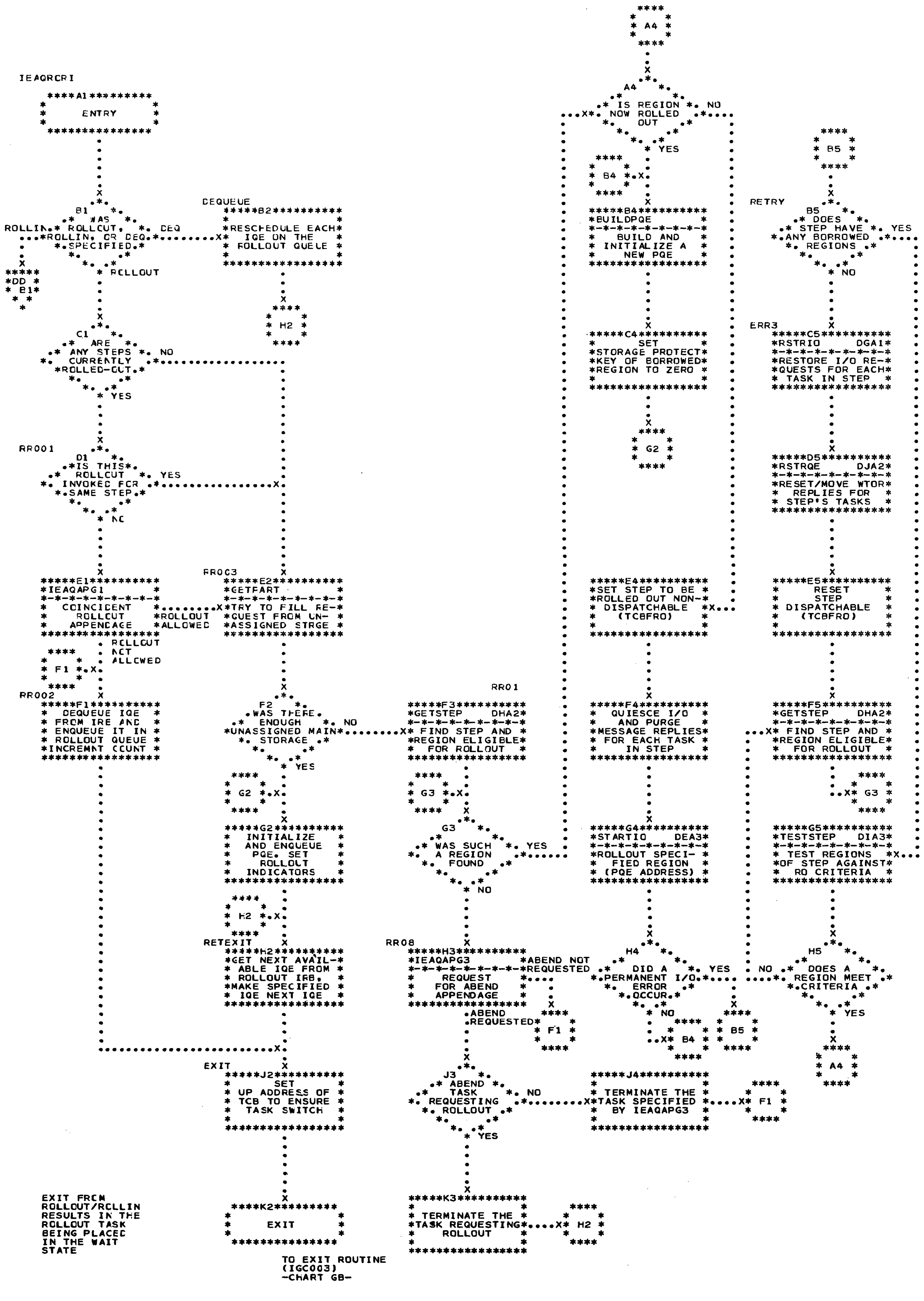

IEAQRCRI
A1
ENTRY
B1
WAS ROLLOUT, ROLLIN, OR DEQ SPECIFIED
ROLLIN
DEQ
ROLLOUT
DD B1
DEQUEUE
B2
RESCHEDULE EACH IQE ON THE ROLLOUT QUEUE
H2
C1
ARE ANY STEPS CURRENTLY ROLLED-OUT
NO
YES
RR001
D1
IS THIS ROLLOUT INVOKED FOR SAME STEP
YES
NO
E1
IEAQAPG1
COINCIDENT ROLLOUT APPENDAGE
ROLLOUT ALLOWED
ROLLOUT NOT ALLOWED
F1
RR003
E2
GETPART
TRY TO FILL RE-QUEST FROM UN-ASSIGNED STRGE
RR002
F1
DEQUEUE IQE FROM IRB AND ENQUEUE IT IN ROLLOUT QUEUE INCREMNT COUNT
F2
WAS THERE ENOUGH UNASSIGNED MAIN STORAGE
NO
YES
G2
G2
INITIALIZE AND ENQUEUE PQE. SET ROLLOUT INDICATORS
H2
RETEXIT
H2
GET NEXT AVAIL-ABLE IQE FROM ROLLOUT IRB. MAKE SPECIFIED IQE NEXT IQE
EXIT
J2
SET UP ADDRESS OF TCB TO ENSURE TASK SWITCH
EXIT FROM ROLLOUT/ROLLIN RESULTS IN THE ROLLOUT TASK BEING PLACED IN THE WAIT STATE
K2
EXIT
TO EXIT ROUTINE (IGC003) -CHART GB-
RR01
F3
GETSTEP DHA2
FIND STEP AND REGION ELIGIBLE FOR ROLLOUT
G3
G3
WAS SUCH A REGION FOUND
YES
NO
RR08
H3
IEAQAPG3
REQUEST FOR ABEND APPENDAGE
ABEND NOT REQUESTED
ABEND REQUESTED
F1
J3
ABEND TASK REQUESTING ROLLOUT
NO
YES
K3
TERMINATE THE TASK REQUESTING ROLLOUT
H2
A4
A4
IS REGION NOW ROLLED OUT
NO
YES
B4
B4
BUILDPQE
BUILD AND INITIALIZE A NEW PQE
C4
SET STORAGE PROTECT KEY OF BORROWED REGION TO ZERO
G2
E4
SET STEP TO BE ROLLED OUT NON-DISPATCHABLE (TCBFRO)
F4
QUIESCE I/O AND PURGE MESSAGE REPLIES FOR EACH TASK IN STEP
G4
STARTIO DEA3
ROLLOUT SPECI-FIED REGION (PQE ADDRESS)
H4
DID A PERMANENT I/O ERROR OCCUR
YES
NO
B4
B5
J4
TERMINATE THE TASK SPECIFIED BY IEAQAPG3
F1
B5
RETRY
B5
DOES STEP HAVE ANY BORROWED REGIONS
YES
NO
ERR3
C5
RSTRIO DGA1
RESTORE I/O RE-QUESTS FOR EACH TASK IN STEP
D5
RSTRQE DJA2
RESET/MOVE WTOR REPLIES FOR STEP'S TASKS
E5
RESET STEP DISPATCHABLE (TCBFRO)
F5
GETSTEP DHA2
FIND STEP AND REGION ELIGIBLE FOR ROLLOUT
G3
G5
TESTSTEP DIA3
TEST REGIONS OF STEP AGAINST RO CRITERIA
H5
DOES A REGION MEET CRITERIA
NO
YES
A4

Chart DD. Rollin Criterion Routine

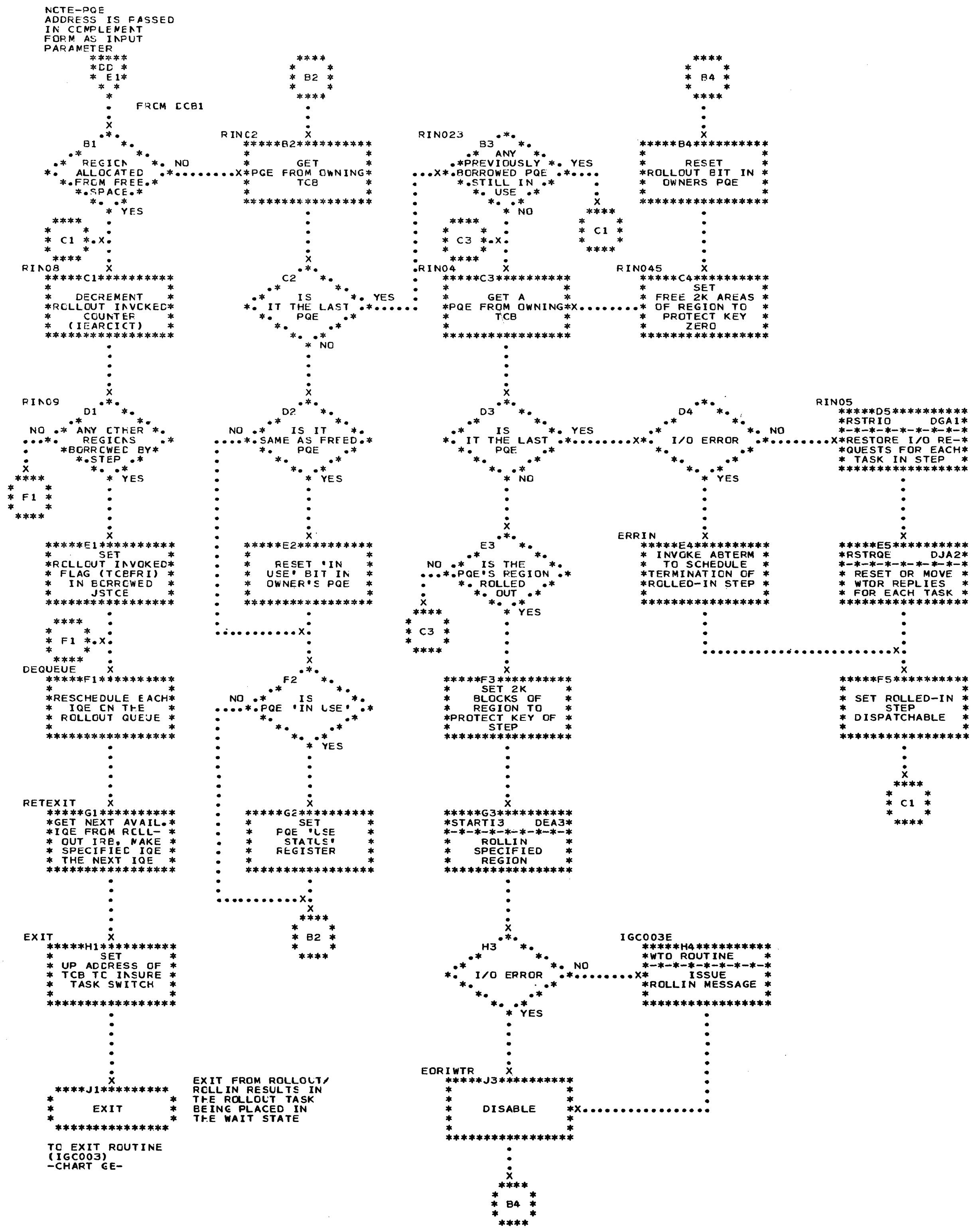

NCTE-PQE ADDRESS IS PASSED IN COMPLEMENT FORM AS INPUT PARAMETER
DD E1
FROM CCB1
B1 REGION ALLOCATED FROM FREE SPACE
NO
YES
C1
RIN08
C1 DECREMENT ROLLOUT INVOKED COUNTER (IEARCICT)
RIN09
D1 ANY OTHER REGIONS BORROWED BY STEP
NO
YES
F1
E1 SET ROLLOUT INVOKED FLAG (TCBFRI) IN BORROWED JSTCE
F1
DEQUEUE
F1 RESCHEDULE EACH IQE ON THE ROLLOUT QUEUE
RETEXIT
G1 GET NEXT AVAIL. IQE FROM ROLL-OUT IRB. MAKE SPECIFIED IQE THE NEXT IQE
EXIT
H1 SET UP ADDRESS OF TCB TO INSURE TASK SWITCH
J1 EXIT
EXIT FROM ROLLOUT/ ROLLIN RESULTS IN THE ROLLOUT TASK BEING PLACED IN THE WAIT STATE
TO EXIT ROUTINE (IGC003) -CHART GE-
B2
RIN02
B2 GET PQE FROM OWNING TCB
C2 IS IT THE LAST PQE
YES
NO
D2 IS IT SAME AS FREED PQE
NO
YES
E2 RESET 'IN USE' BIT IN OWNER'S PQE
F2 IS PQE 'IN USE'
NO
YES
G2 SET PQE 'USE STATUS' REGISTER
B2
RIN023
B3 ANY PREVIOUSLY BORROWED PQE STILL IN USE
YES
NO
C3
C1
RIN04
C3 GET A PQE FROM OWNING TCB
D3 IS IT THE LAST PQE
YES
NO
E3 IS THE PQE'S REGION ROLLED OUT
NO
YES
C3
F3 SET 2K BLOCKS OF REGION TO PROTECT KEY OF STEP
G3 STARTI3 DEA3 ROLLIN SPECIFIED REGION
H3 I/O ERROR
NO
YES
EORIWTR
J3 DISABLE
B4
B4
B4 RESET ROLLOUT BIT IN OWNERS PQE
RIN045
C4 SET FREE 2K AREAS OF REGION TO PROTECT KEY ZERO
D4 I/O ERROR
NO
YES
ERRIN
E4 INVOKE ABTERM TO SCHEDULE TERMINATION OF ROLLED-IN STEP
IGC003E
H4 WTO ROUTINE ISSUE ROLLIN MESSAGE
RIN05
D5 RSTRIO DGA1 RESTORE I/O RE-QUESTS FOR EACH TASK IN STEP
E5 RSTRQE DJA2 RESET OR MOVE WTOR REPLIES FOR EACH TASK
F5 SET ROLLED-IN STEP DISPATCHABLE
C1

Chart DE. Rollout/Rollin I/O Routine

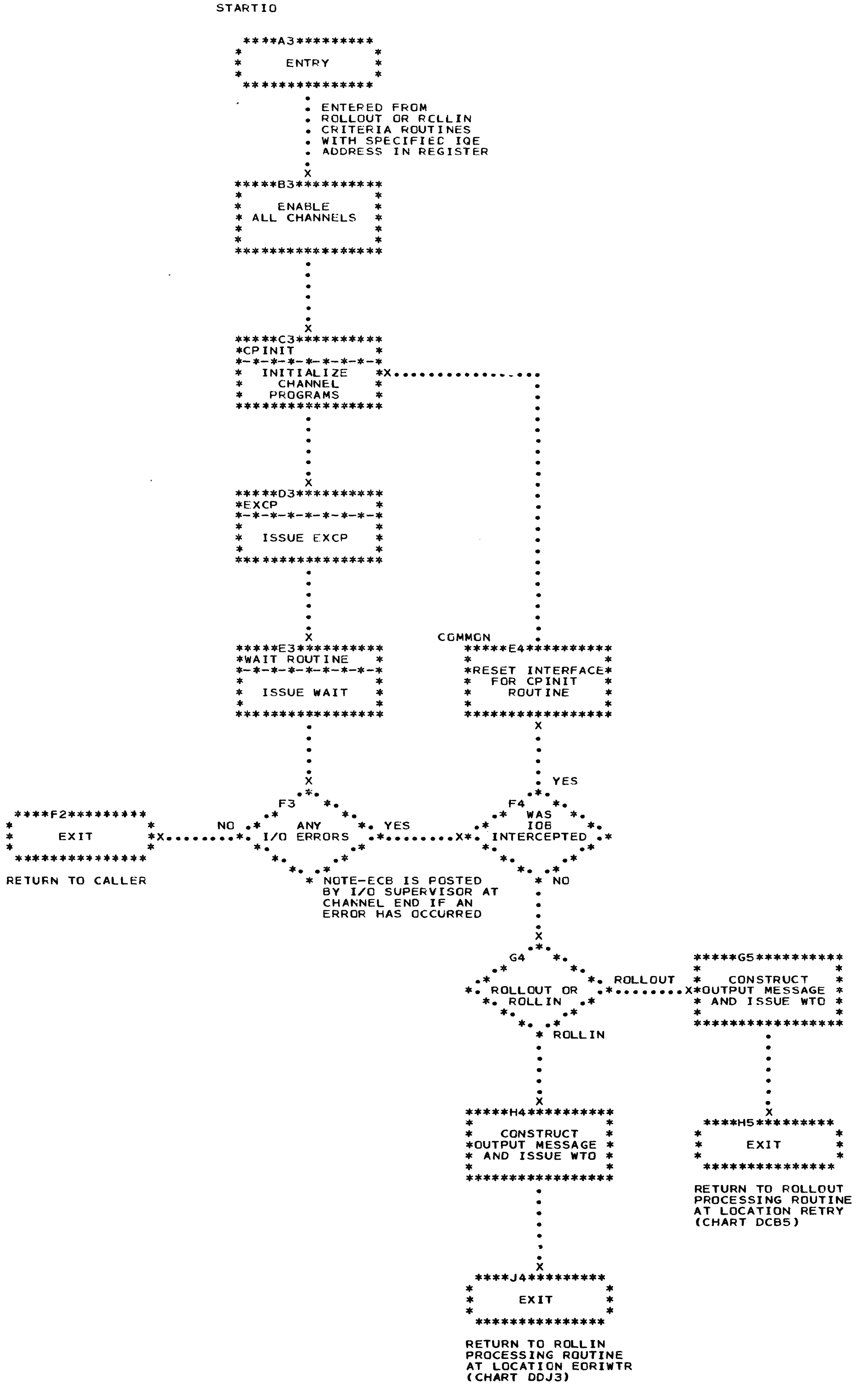

STARTIO
A3
ENTRY
ENTERED FROM ROLLOUT OR RCLLIN CRITERIA ROUTINES WITH SPECIFIEC IQE ADDRESS IN REGISTER
B3
ENABLE ALL CHANNELS
C3
CPINIT
INITIALIZE CHANNEL PROGRAMS
D3
EXCP
ISSUE EXCP
E3
WAIT ROUTINE
ISSUE WAIT
COMMON
E4
RESET INTERFACE FOR CPINIT ROUTINE
F2
EXIT
RETURN TO CALLER
NO
F3
ANY I/O ERRORS
YES
NOTE-ECB IS POSTED BY I/O SUPERVISOR AT CHANNEL END IF AN ERROR HAS OCCURRED
F4
WAS IOB INTERCEPTED
YES
NO
G4
ROLLOUT OR ROLLIN
ROLLOUT
ROLLIN
G5
CONSTRUCT OUTPUT MESSAGE AND ISSUE WTO
H4
CONSTRUCT OUTPUT MESSAGE AND ISSUE WTO
H5
EXIT
RETURN TO ROLLOUT PROCESSING ROUTINE AT LOCATION RETRY (CHART DCB5)
J4
EXIT
RETURN TO ROLLIN PROCESSING ROUTINE AT LOCATION EORIWTR (CHART DDJ3)

Chart DF. SVC Purge Interface

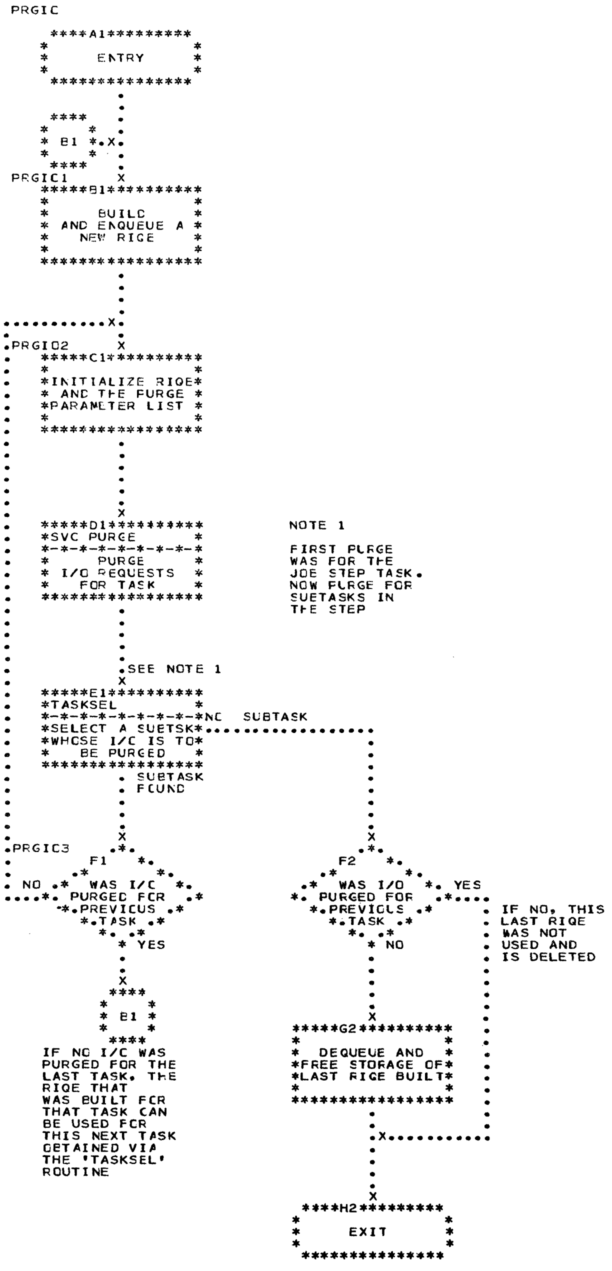

PRGIC
A1
ENTRY
B1
PRGIC1
B1
BUILC AND ENQUEUE A NEW RIGE
PRGIO2
C1
INITIALIZE RIQE AND THE PURGE PARAMETER LIST
D1
SVC PURGE
PURGE I/O REQUESTS FOR TASK
NOTE 1
FIRST PURGE WAS FOR THE JOB STEP TASK. NOW PURGE FOR SUBTASKS IN THE STEP
SEE NOTE 1
E1
TASKSEL
SELECT A SUBTSK WHOSE I/O IS TO BE PURGED
NO SUBTASK
SUBTASK FOUND
PRGIC3
F1
WAS I/O PURGED FOR PREVIOUS TASK
NO
YES
B1
IF NO I/O WAS PURGED FOR THE LAST TASK. THE RIQE THAT WAS BUILT FOR THAT TASK CAN BE USED FOR THIS NEXT TASK OBTAINED VIA THE 'TASKSEL' ROUTINE
F2
WAS I/O PURGED FOR PREVIOUS TASK
YES
NO
IF NO, THIS LAST RIQE WAS NOT USED AND IS DELETED
G2
DEQUEUE AND FREE STORAGE OF LAST RIQE BUILT
H2
EXIT

Chart DG. SVC Restore Interface

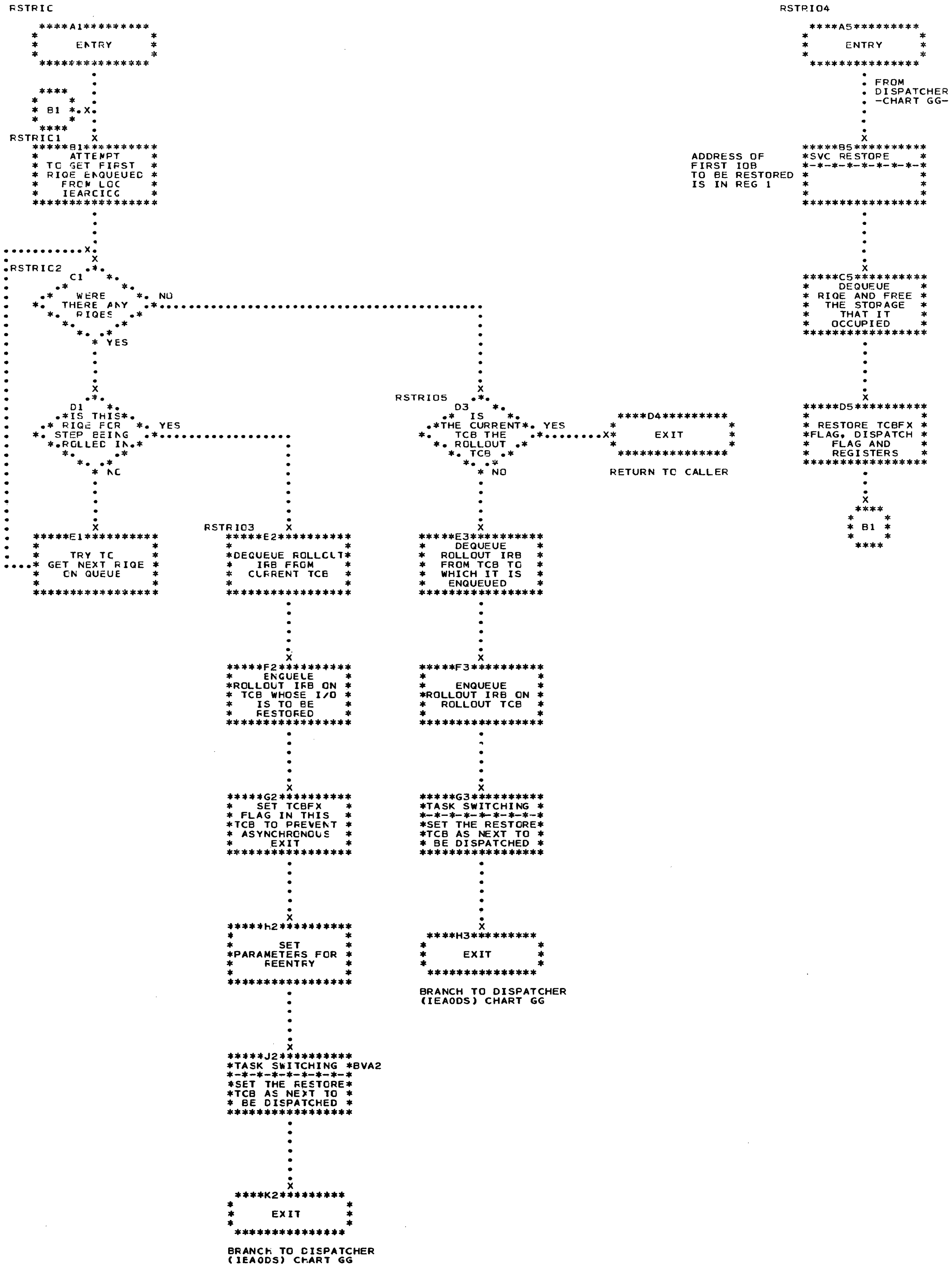

RSTRIC
A1
ENTRY
B1
RSTRIC1
B1
ATTEMPT TO GET FIRST RIQE ENQUEUED FROM LOC IEARCICC
RSTRIC2
C1
WERE THERE ANY RIQES
NO
YES
D1
IS THIS RIQE FOR STEP BEING ROLLED IN
YES
NO
E1
TRY TO GET NEXT RIQE ON QUEUE
RSTRIO3
E2
DEQUEUE ROLLOUT IRB FROM CURRENT TCB
F2
ENQUEUE ROLLOUT IRB ON TCB WHOSE I/O IS TO BE RESTORED
G2
SET TCBFX FLAG IN THIS TCB TO PREVENT ASYNCHRONOUS EXIT
H2
SET PARAMETERS FOR REENTRY
J2
TASK SWITCHING
BVA2
SET THE RESTORE TCB AS NEXT TO BE DISPATCHED
K2
EXIT
BRANCH TO DISPATCHER (IEAODS) CHART GG
RSTRIO5
D3
IS THE CURRENT TCB THE ROLLOUT TCB
YES
NO
D4
EXIT
RETURN TO CALLER
E3
DEQUEUE ROLLOUT IRB FROM TCB TO WHICH IT IS ENQUEUED
F3
ENQUEUE ROLLOUT IRB ON ROLLOUT TCB
G3
TASK SWITCHING
SET THE RESTORE TCB AS NEXT TO BE DISPATCHED
H3
EXIT
BRANCH TO DISPATCHER (IEAODS) CHART GG
RSTRIO4
A5
ENTRY
FROM DISPATCHER -CHART GG-
ADDRESS OF FIRST IOB TO BE RESTORED IS IN REG 1
B5
SVC RESTORE
C5
DEQUEUE RIQE AND FREE THE STORAGE THAT IT OCCUPIED
D5
RESTORE TCBFX FLAG, DISPATCH FLAG AND REGISTERS
B1

Chart DH. Rollout/Rollin GETSTEP Routine

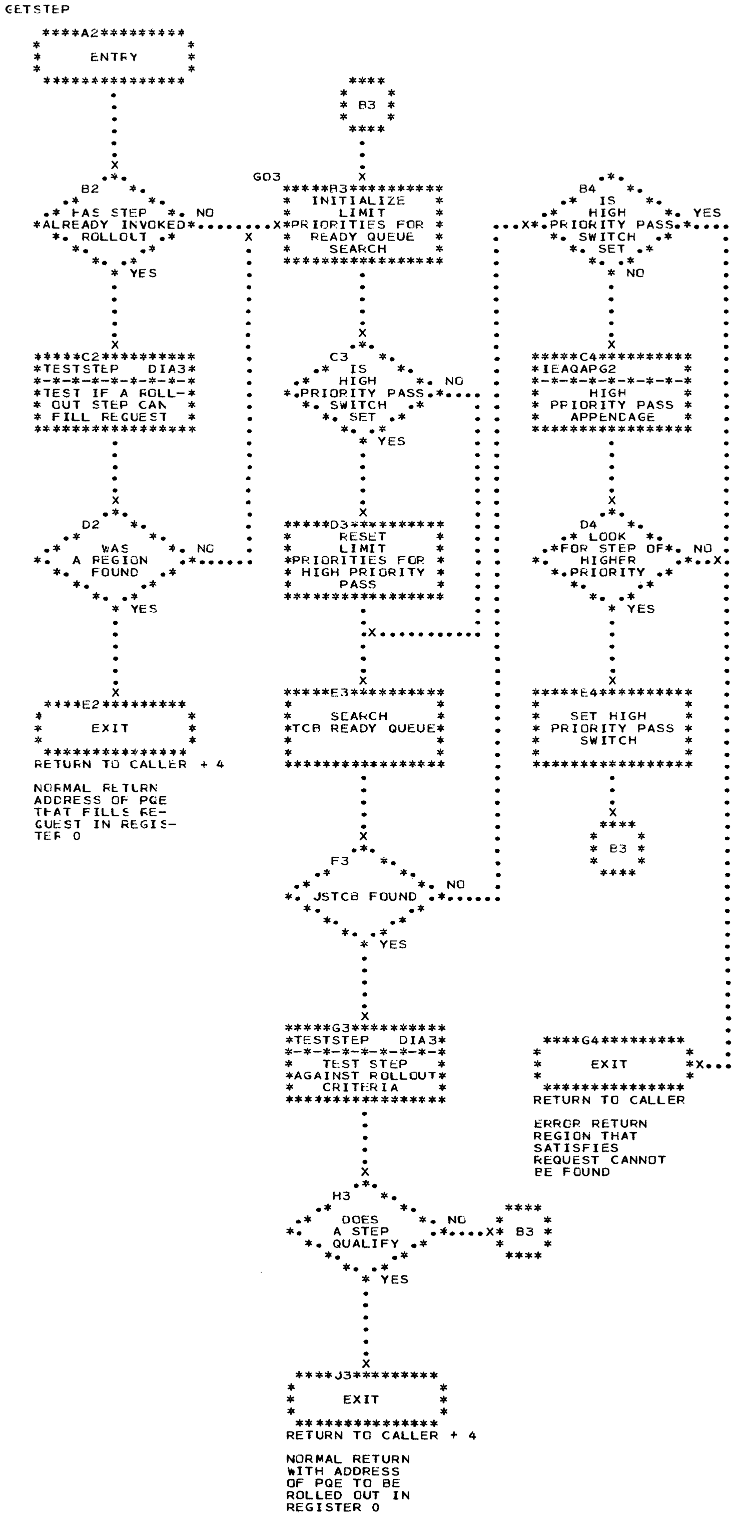

GETSTEP
A2
ENTRY
B3
B2
HAS STEP ALREADY INVOKED ROLLOUT
NO
YES
G03
B3
INITIALIZE LIMIT PRIORITIES FOR READY QUEUE SEARCH
B4
IS HIGH PRIORITY PASS SWITCH SET
YES
NO
C2
TESTSTEP DIA3
TEST IF A ROLL-OUT STEP CAN FILL REQUEST
C3
IS HIGH PRIORITY PASS SWITCH SET
NO
YES
C4
IEAQAPG2
HIGH PRIORITY PASS APPENDAGE
D2
WAS A REGION FOUND
NO
YES
D3
RESET LIMIT PRIORITIES FOR HIGH PRIORITY PASS
D4
LOOK FOR STEP OF HIGHER PRIORITY
NO
YES
E2
EXIT
RETURN TO CALLER + 4
NORMAL RETURN ADDRESS OF PQE THAT FILLS RE-QUEST IN REGIS-TER 0
E3
SEARCH TCB READY QUEUE
E4
SET HIGH PRIORITY PASS SWITCH
B3
F3
JSTCB FOUND
NO
YES
G3
TESTSTEP DIA3
TEST STEP AGAINST ROLLOUT CRITERIA
G4
EXIT
RETURN TO CALLER
ERROR RETURN REGION THAT SATISFIES REQUEST CANNOT BE FOUND
H3
DOES A STEP QUALIFY
NO
B3
YES
J3
EXIT
RETURN TO CALLER + 4
NORMAL RETURN WITH ADDRESS OF PQE TO BE ROLLED OUT IN REGISTER 0

Chart DI. Rollout/Rollin TESTSTEP Routine

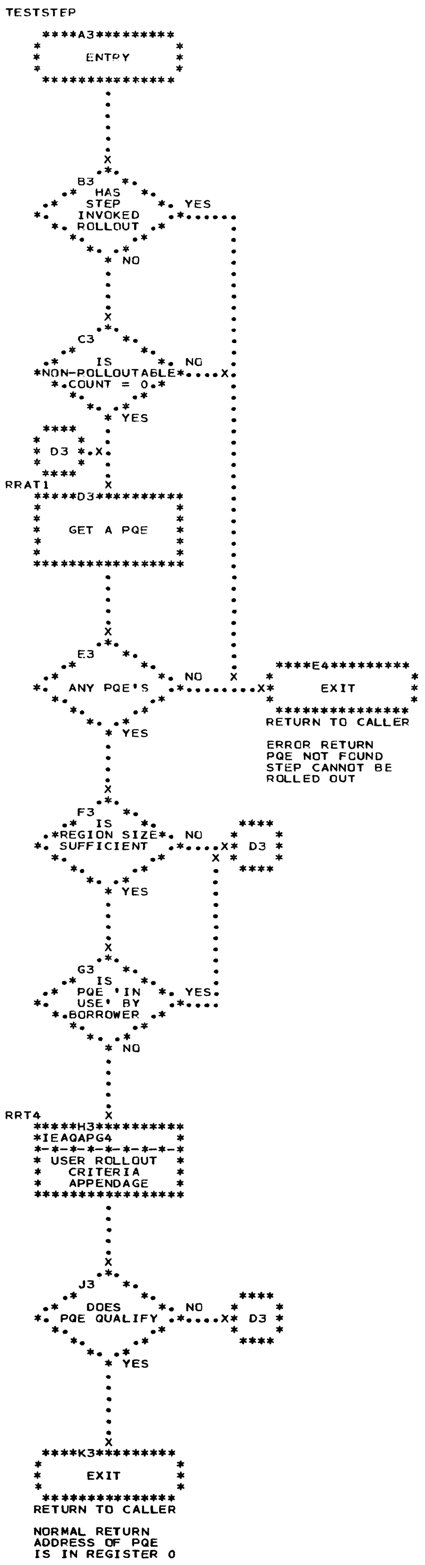


Chart DJ. Rollout/Rollin Reply Restore Routine

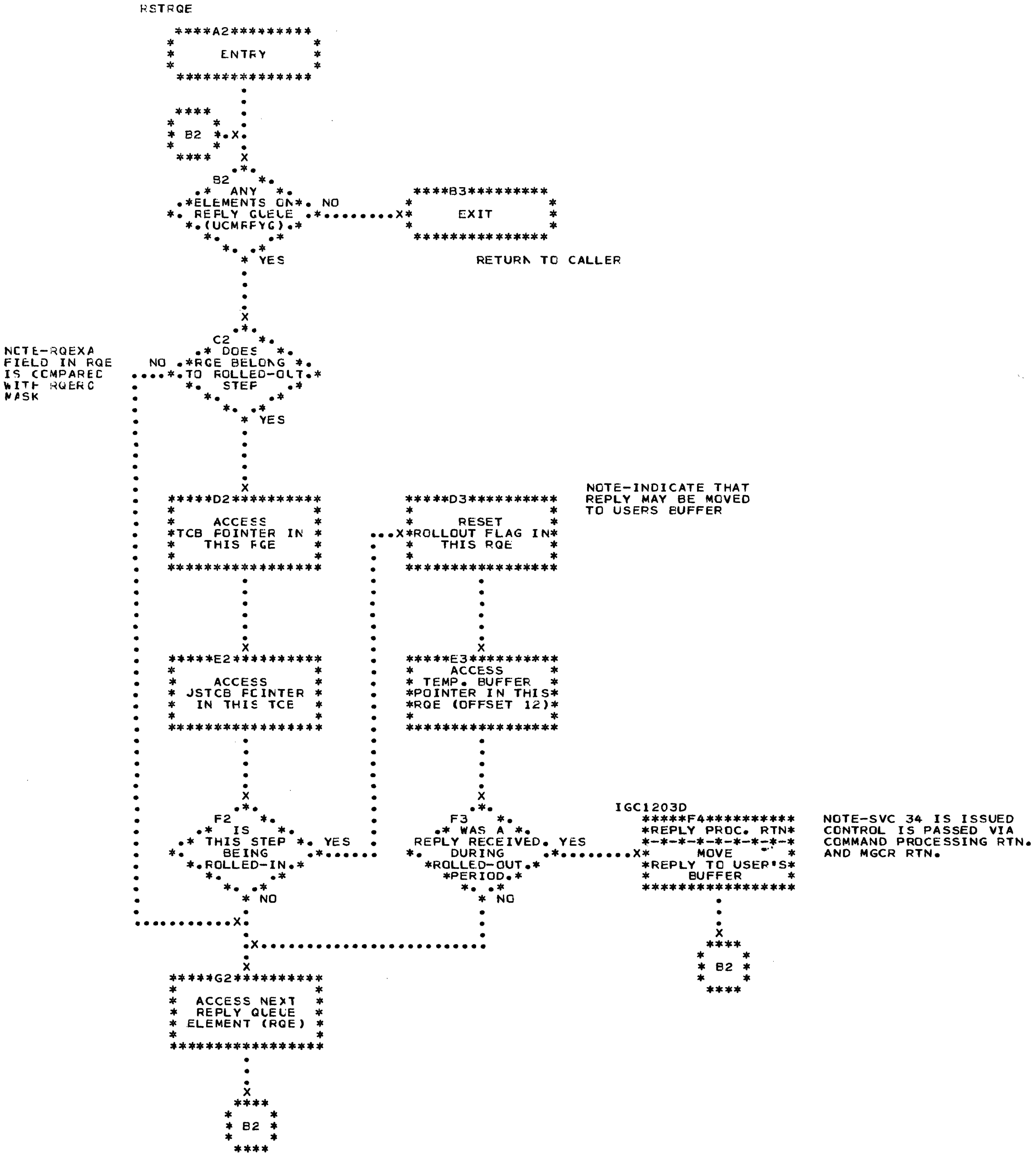

RSTRQE
A2
ENTRY
B2
B2
ANY ELEMENTS ON REPLY QUEUE (UCMRPYQ)
NO
YES
B3
EXIT
RETURN TO CALLER
C2
DOES RQE BELONG TO ROLLED-OUT STEP
NOTE-RQEXA FIELD IN RQE IS COMPARED WITH RQERC MASK
NO
YES
D2
ACCESS TCB POINTER IN THIS RQE
D3
RESET ROLLOUT FLAG IN THIS RQE
NOTE-INDICATE THAT REPLY MAY BE MOVED TO USERS BUFFER
E2
ACCESS JSTCB POINTER IN THIS TCB
E3
ACCESS TEMP. BUFFER POINTER IN THIS RQE (OFFSET 12)
F2
IS THIS STEP BEING ROLLED-IN
YES
NO
F3
WAS A REPLY RECEIVED DURING ROLLED-OUT PERIOD
YES
NO
IGC1203D
F4
REPLY PROC. RTN
MOVE REPLY TO USER'S BUFFER
NOTE-SVC 34 IS ISSUED CONTROL IS PASSED VIA COMMAND PROCESSING RTN. AND MGCR RTN.
B2
G2
ACCESS NEXT REPLY QUEUE ELEMENT (RQE)
B2

Chart EA. Time Routine

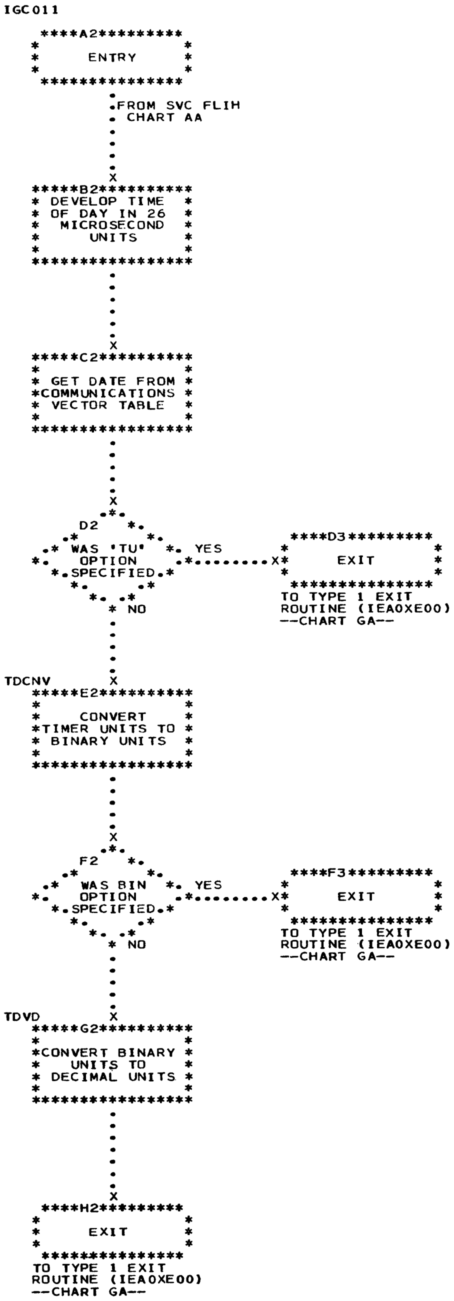
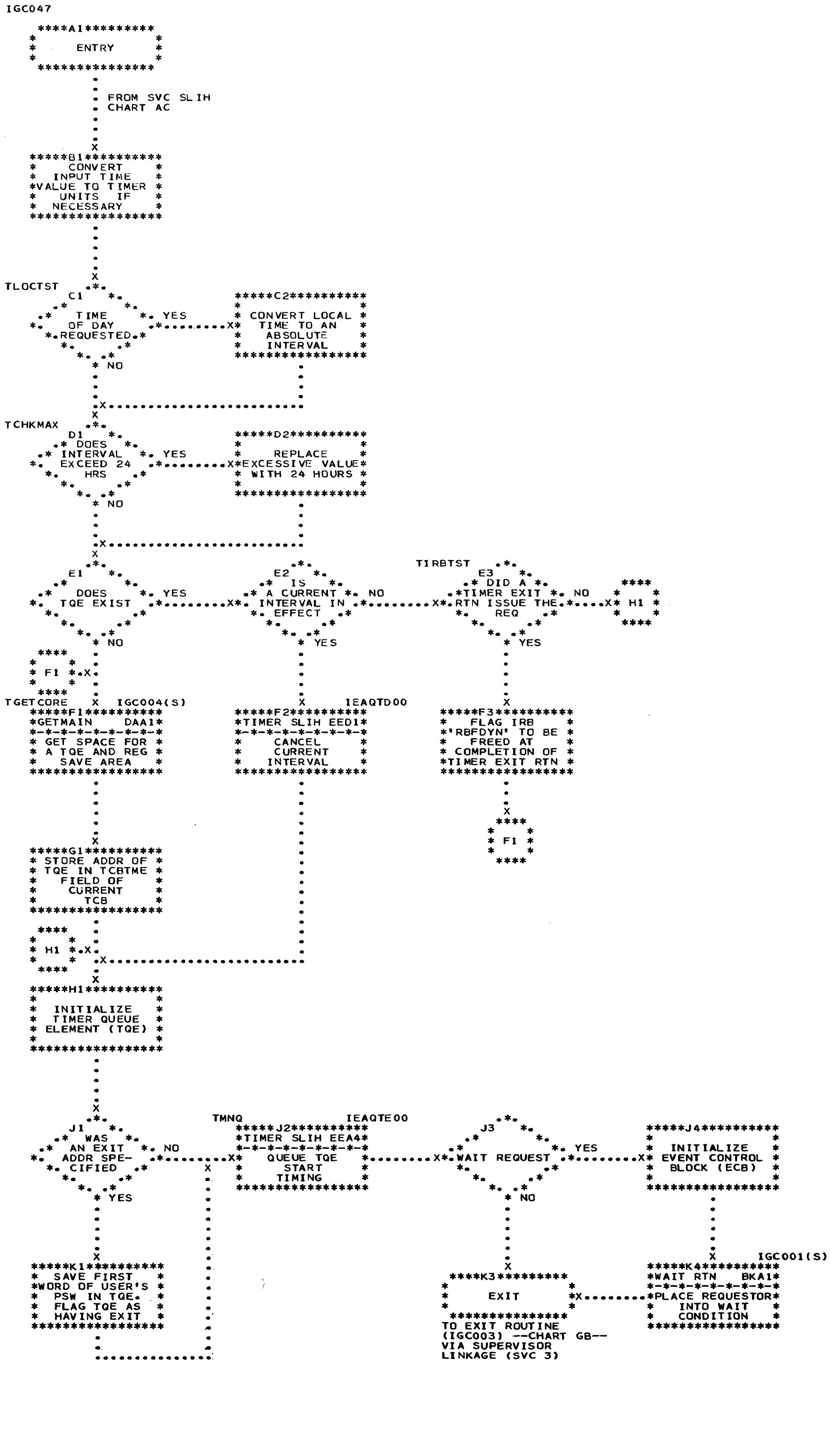
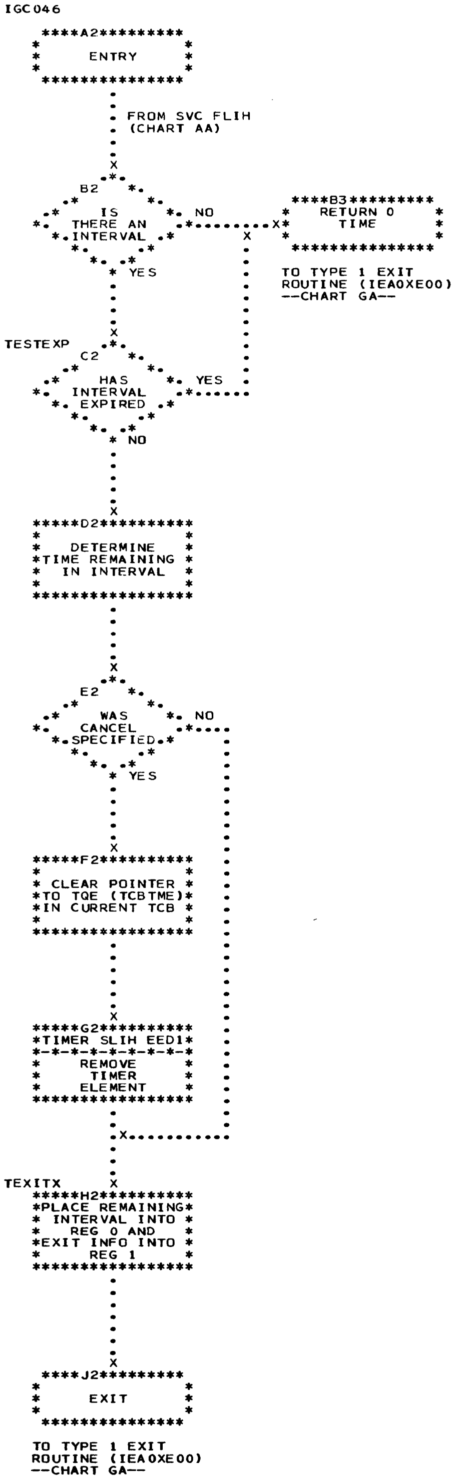

IGC011
A2
ENTRY
FROM SVC FLIH CHART AA
B2
DEVELOP TIME OF DAY IN 26 MICROSECOND UNITS
C2
GET DATE FROM COMMUNICATIONS VECTOR TABLE
D2
WAS 'TU' OPTION SPECIFIED
YES
NO
D3
EXIT
TO TYPE 1 EXIT ROUTINE (IEA0XE00) --CHART GA--
TDCNV
E2
CONVERT TIMER UNITS TO BINARY UNITS
F2
WAS BIN OPTION SPECIFIED
YES
NO
F3
EXIT
TO TYPE 1 EXIT ROUTINE (IEA0XE00) --CHART GA--
TDVD
G2
CONVERT BINARY UNITS TO DECIMAL UNITS
H2
EXIT
TO TYPE 1 EXIT ROUTINE (IEA0XE00) --CHART GA--

Chart EB. STIMER Routine


IGC047
A1 ENTRY
FROM SVC SLIH CHART AC
B1 CONVERT INPUT TIME VALUE TO TIMER UNITS IF NECESSARY
TLOCTST
C1 TIME OF DAY REQUESTED
YES
C2 CONVERT LOCAL TIME TO AN ABSOLUTE INTERVAL
NO
TCHKMAX
D1 DOES INTERVAL EXCEED 24 HRS
YES
D2 REPLACE EXCESSIVE VALUE WITH 24 HOURS
NO
E1 DOES TQE EXIST
YES
E2 IS A CURRENT INTERVAL IN EFFECT
NO
TIRBTST
E3 DID A TIMER EXIT RTN ISSUE THE REQ
NO
H1
NO
F1
TGETCORE
IGC004(S)
F1 GETMAIN DAA1 GET SPACE FOR A TQE AND REG SAVE AREA
YES
IEAQTD00
F2 TIMER SLIH EED1 CANCEL CURRENT INTERVAL
YES
F3 FLAG IRB 'RBFDYN' TO BE FREED AT COMPLETION OF TIMER EXIT RTN
F1
G1 STORE ADDR OF TQE IN TCBTME FIELD OF CURRENT TCB
H1
H1 INITIALIZE TIMER QUEUE ELEMENT (TQE)
J1 WAS AN EXIT ADDR SPECIFIED
NO
TMNQ
IEAQTE00
J2 TIMER SLIH EEA4 QUEUE TQE START TIMING
J3 WAIT REQUEST
YES
J4 INITIALIZE EVENT CONTROL BLOCK (ECB)
YES
K1 SAVE FIRST WORD OF USER'S PSW IN TQE. FLAG TQE AS HAVING EXIT
NO
K3 EXIT
TO EXIT ROUTINE (IGC003) --CHART GB-- VIA SUPERVISOR LINKAGE (SVC 3)
IGC001(S)
K4 WAIT RTN BKA1 PLACE REQUESTOR INTO WAIT CONDITION

• Chart EC. TTIMER Routine


IGC046
A2 ENTRY
FROM SVC FLIH (CHART AA)
B2 IS THERE AN INTERVAL
NO
B3 RETURN 0 TIME
TO TYPE 1 EXIT ROUTINE (IEA0XE00) --CHART GA--
YES
TESTEXP
C2 HAS INTERVAL EXPIRED
YES
NO
D2 DETERMINE TIME REMAINING IN INTERVAL
E2 WAS CANCEL SPECIFIED
NO
YES
F2 CLEAR POINTER TO TQE (TCBTME) IN CURRENT TCB
G2 TIMER SLIH EED1
REMOVE TIMER ELEMENT
TEXITX
H2 PLACE REMAINING INTERVAL INTO REG 0 AND EXIT INFO INTO REG 1
J2 EXIT
TO TYPE 1 EXIT ROUTINE (IEA0XE00) --CHART GA--

• Chart ED. Timer Second-Level Interruption Handler (Part 1 of 2)

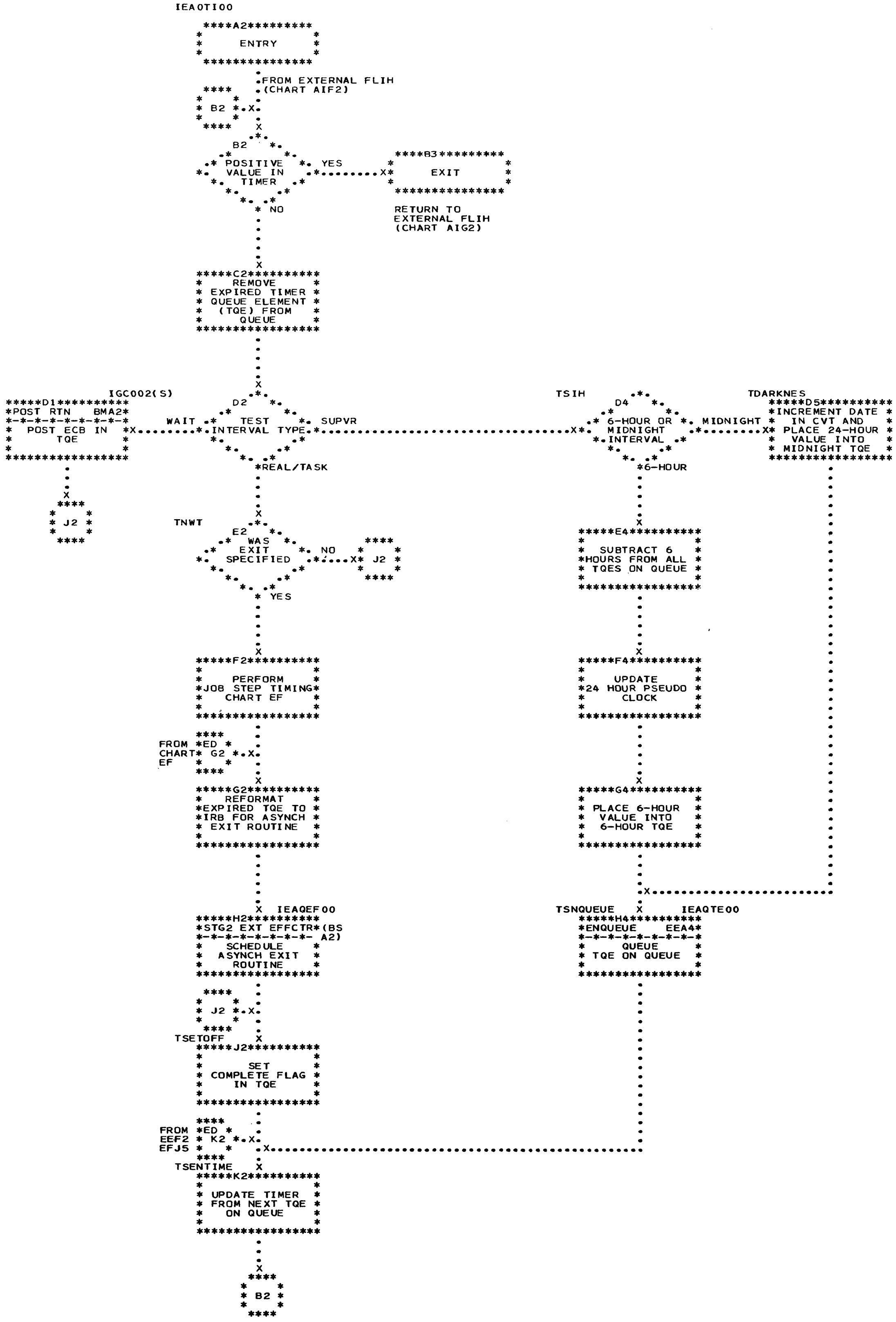

IEAOTIOO
A2
ENTRY
FROM EXTERNAL FLIH (CHART AIF2)
B2
B2
POSITIVE VALUE IN TIMER
YES
NO
B3
EXIT
RETURN TO EXTERNAL FLIH (CHART AIG2)
C2
REMOVE EXPIRED TIMER QUEUE ELEMENT (TQE) FROM QUEUE
IGC002(S)
D1
POST RTN BMA2
POST ECB IN TQE
J2
WAIT
D2
TEST INTERVAL TYPE
SUPVR
REAL/TASK
TNWT
E2
WAS EXIT SPECIFIED
NO
J2
YES
F2
PERFORM JOB STEP TIMING CHART EF
FROM CHART EF
ED G2
G2
REFORMAT EXPIRED TQE TO IRB FOR ASYNCH EXIT ROUTINE
IEAQEF00
H2
STG2 EXT EFFCTR (BS A2)
SCHEDULE ASYNCH EXIT ROUTINE
J2
TSETOFF
J2
SET COMPLETE FLAG IN TQE
FROM EEF2 EFJ5
ED K2
TSENTIME
K2
UPDATE TIMER FROM NEXT TQE ON QUEUE
B2
TSIH
D4
6-HOUR OR MIDNIGHT INTERVAL
MIDNIGHT
6-HOUR
TDARKNES
D5
INCREMENT DATE IN CVT AND PLACE 24-HOUR VALUE INTO MIDNIGHT TQE
E4
SUBTRACT 6 HOURS FROM ALL TQES ON QUEUE
F4
UPDATE 24 HOUR PSEUDO CLOCK
G4
PLACE 6-HOUR VALUE INTO 6-HOUR TQE
TSNQUEUE
IEAQTE00
H4
ENQUEUE EEA4
QUEUE TQE ON QUEUE

Chart EE. Timer Second-Level Interruption Handler (Part 2 of 2)-Dequeue and Enqueue Subroutines

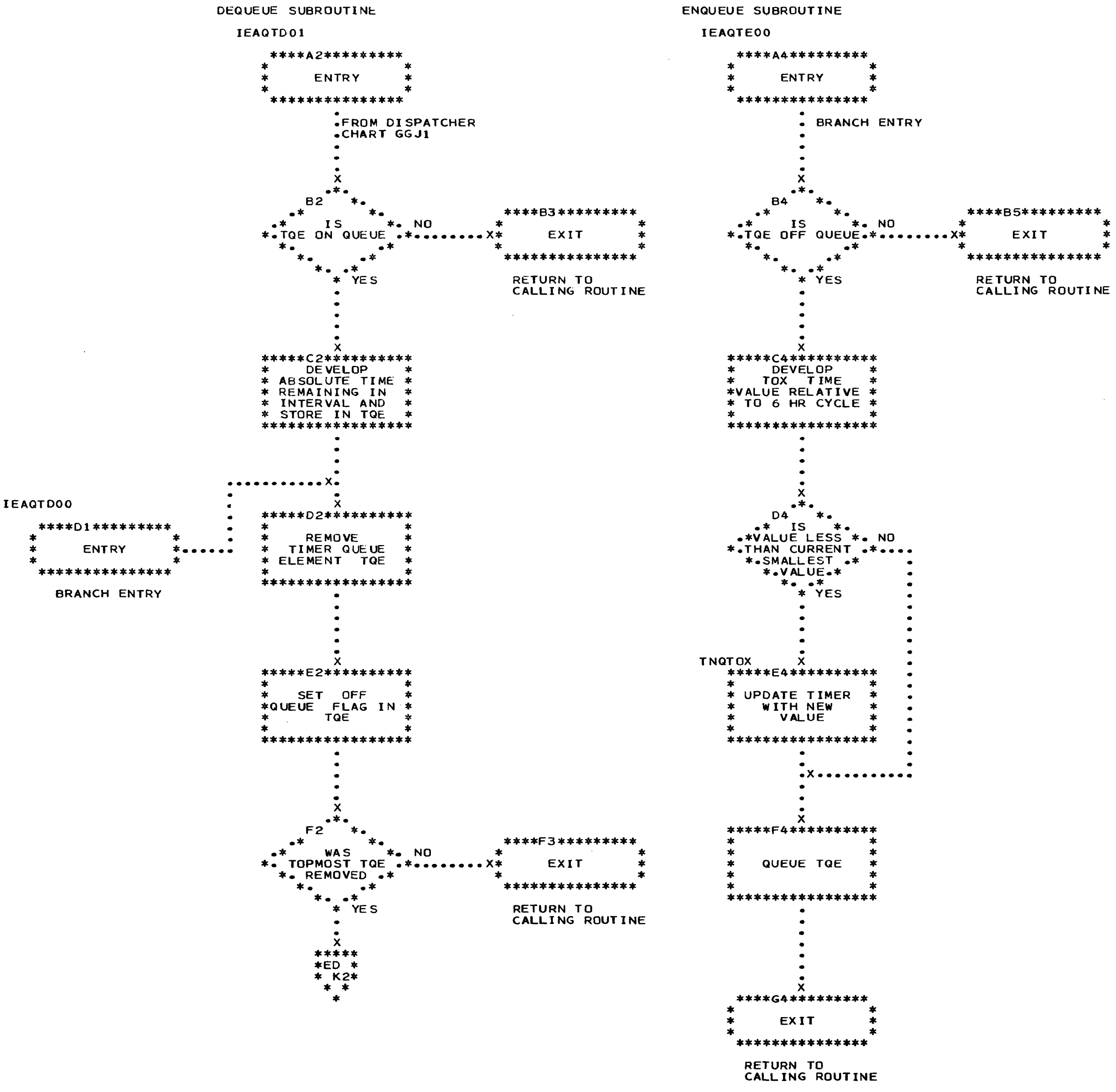
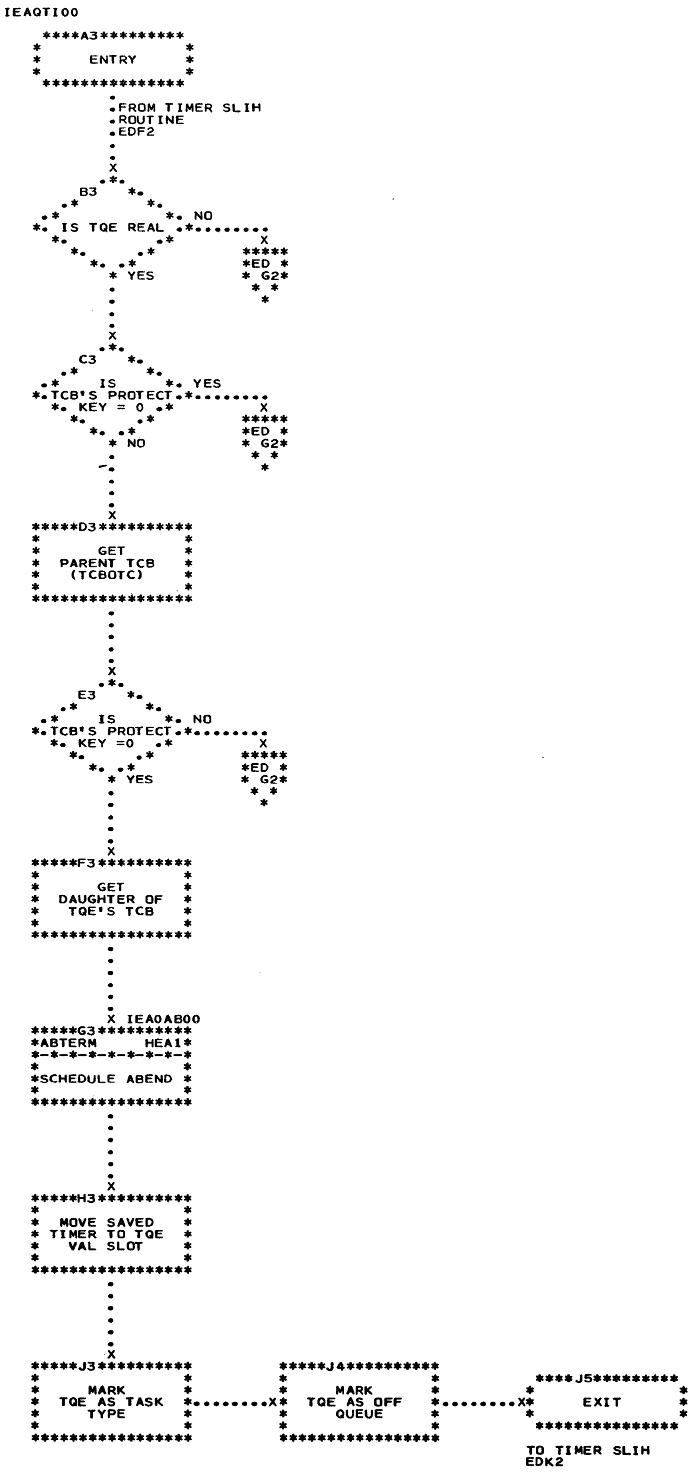
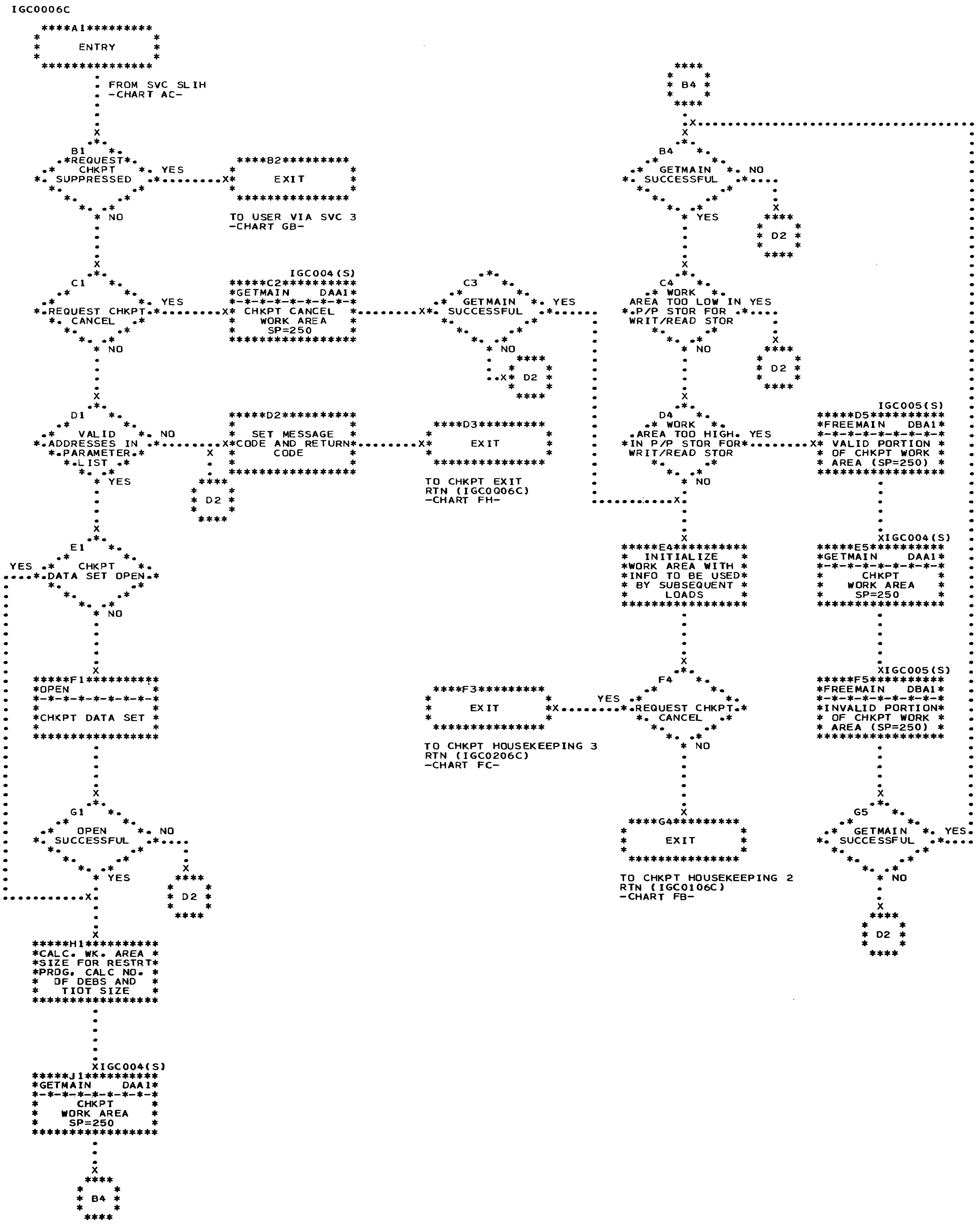

DEQUEUE SUBROUTINE
IEAQTD01
A2
ENTRY
FROM DISPATCHER CHART GGJ1
B2
IS TQE ON QUEUE
NO
B3
EXIT
RETURN TO CALLING ROUTINE
YES
C2
DEVELOP ABSOLUTE TIME REMAINING IN INTERVAL AND STORE IN TQE
IEAQTD00
D1
ENTRY
BRANCH ENTRY
D2
REMOVE TIMER QUEUE ELEMENT TQE
E2
SET OFF QUEUE FLAG IN TQE
F2
WAS TOPMOST TQE REMOVED
NO
F3
EXIT
RETURN TO CALLING ROUTINE
YES
ED K2
ENQUEUE SUBROUTINE
IEAQTE00
A4
ENTRY
BRANCH ENTRY
B4
IS TQE OFF QUEUE
NO
B5
EXIT
RETURN TO CALLING ROUTINE
YES
C4
DEVELOP TOX TIME VALUE RELATIVE TO 6 HR CYCLE
D4
IS VALUE LESS THAN CURRENT SMALLEST VALUE
NO
YES
TNQTOX
E4
UPDATE TIMER WITH NEW VALUE
F4
QUEUE TQE
G4
EXIT
RETURN TO CALLING ROUTINE

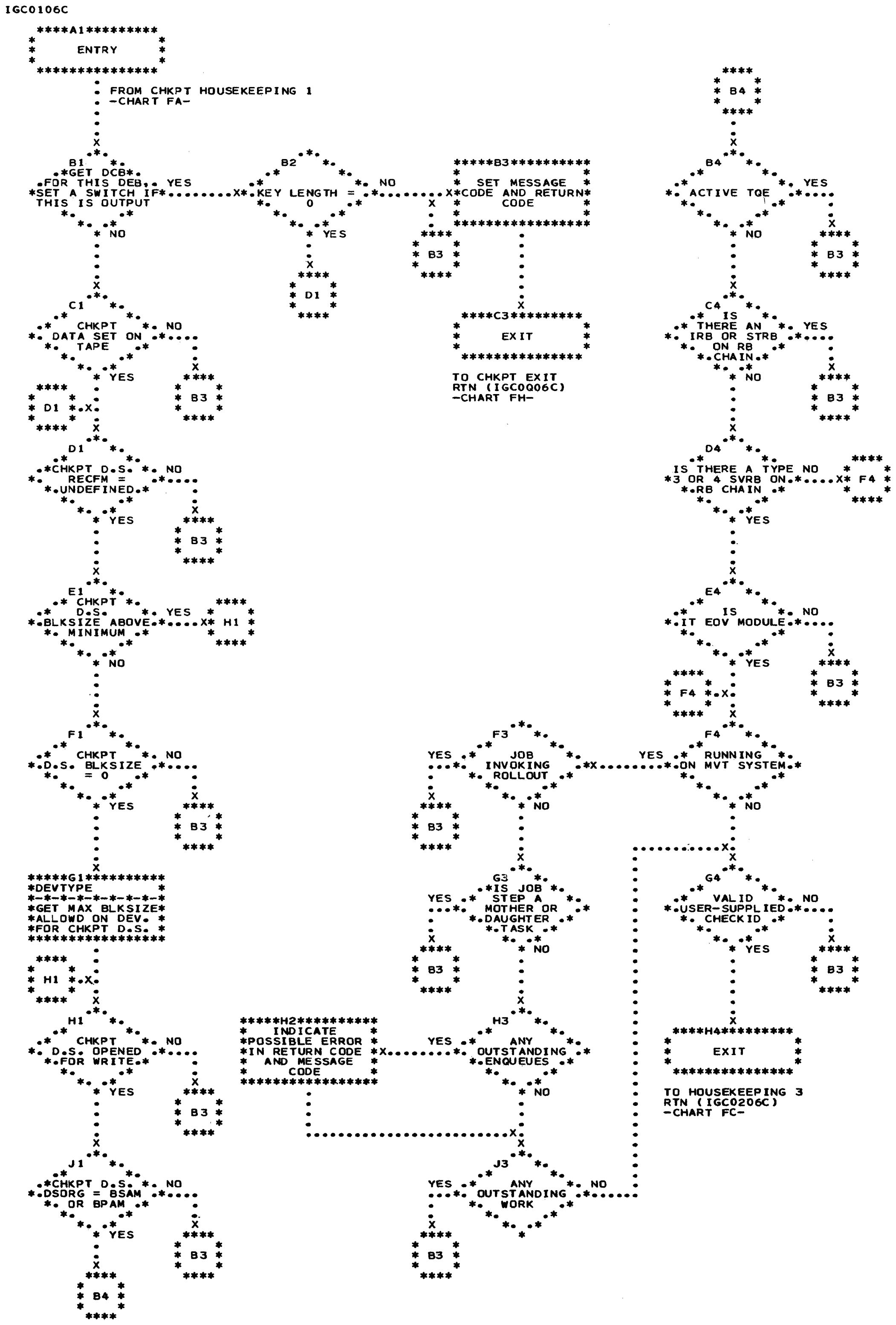
Chart EF. Timer Second-Level Interruption Handler (Job Step Timing)



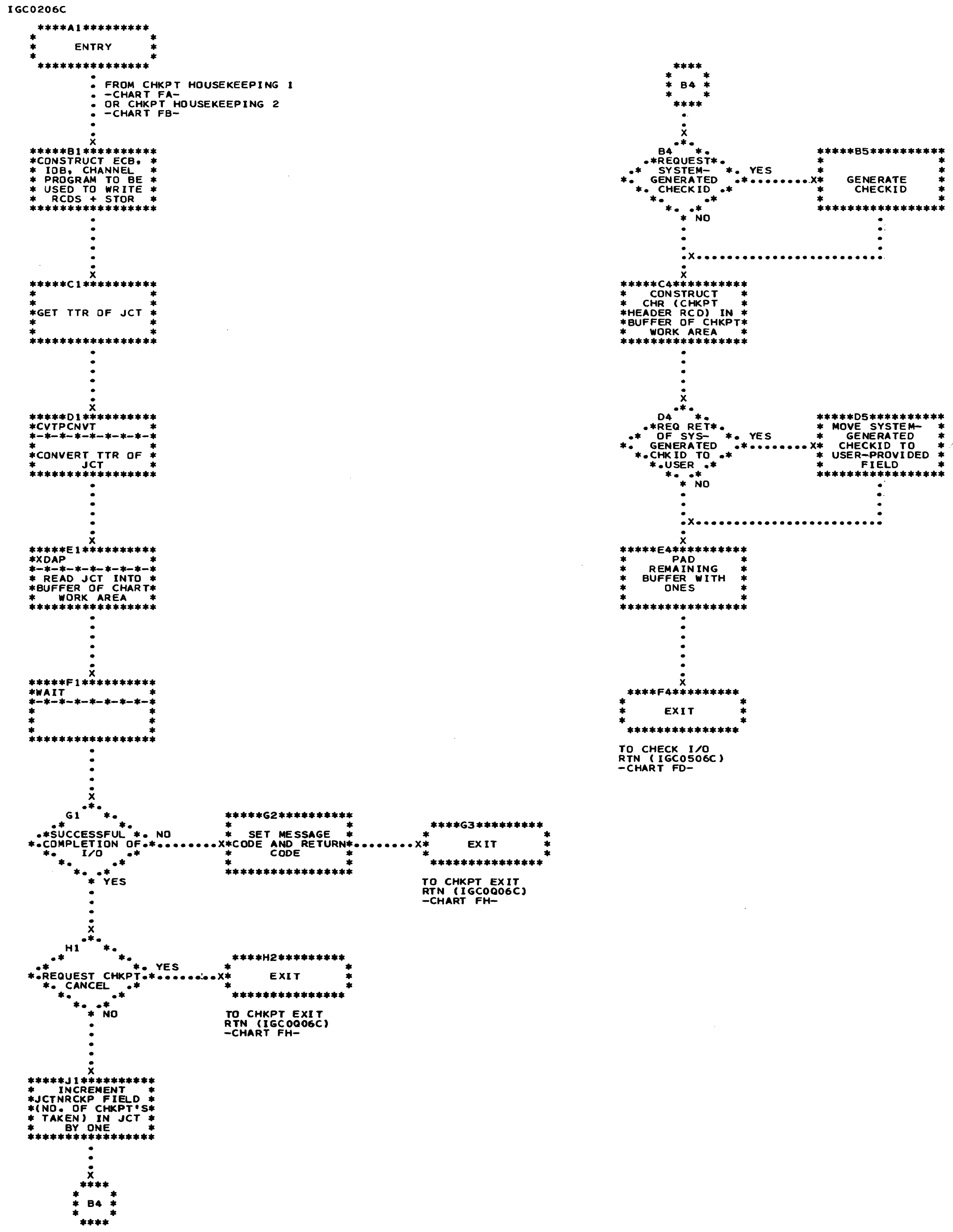
• Chart FA. Checkpoint Housekeeping 1 Routine


IGC0006C
A1
ENTRY
FROM SVC SLIH -CHART AC-
B1
REQUEST CHKPT SUPPRESSED
YES
NO
B2
EXIT
TO USER VIA SVC 3 -CHART GB-
C1
REQUEST CHKPT CANCEL
YES
NO
IGC004(S)
C2
GETMAIN DAA1
CHKPT CANCEL WORK AREA SP=250
C3
GETMAIN SUCCESSFUL
YES
NO
D2
D1
VALID ADDRESSES IN PARAMETER LIST
NO
YES
D2
SET MESSAGE CODE AND RETURN CODE
D2
D3
EXIT
TO CHKPT EXIT RTN (IGC0Q06C) -CHART FH-
E1
CHKPT DATA SET OPEN
YES
NO
F1
OPEN
CHKPT DATA SET
G1
OPEN SUCCESSFUL
NO
YES
D2
H1
CALC. WK. AREA SIZE FOR RESTRT PRDG, CALC NO. OF DEBS AND TIOT SIZE
IGC004(S)
J1
GETMAIN DAA1
CHKPT WORK AREA SP=250
B4
B4
B4
GETMAIN SUCCESSFUL
NO
YES
D2
C4
WORK AREA TOO LOW IN P/P STOR FOR WRIT/READ STOR
YES
NO
D2
D4
WORK AREA TOO HIGH IN P/P STOR FOR WRIT/READ STOR
YES
NO
IGC005(S)
D5
FREEMAIN DBA1
VALID PORTION OF CHKPT WORK AREA (SP=250)
E4
INITIALIZE WORK AREA WITH INFO TO BE USED BY SUBSEQUENT LOADS
IGC004(S)
E5
GETMAIN DAA1
CHKPT WORK AREA SP=250
F3
EXIT
TO CHKPT HOUSEKEEPING 3 RTN (IGC0206C) -CHART FC-
F4
REQUEST CHKPT CANCEL
YES
NO
IGC005(S)
F5
FREEMAIN DBA1
INVALID PORTION OF CHKPT WORK AREA (SP=250)
G4
EXIT
TO CHKPT HOUSEKEEPING 2 RTN (IGC0106C) -CHART FB-
G5
GETMAIN SUCCESSFUL
YES
NO
D2

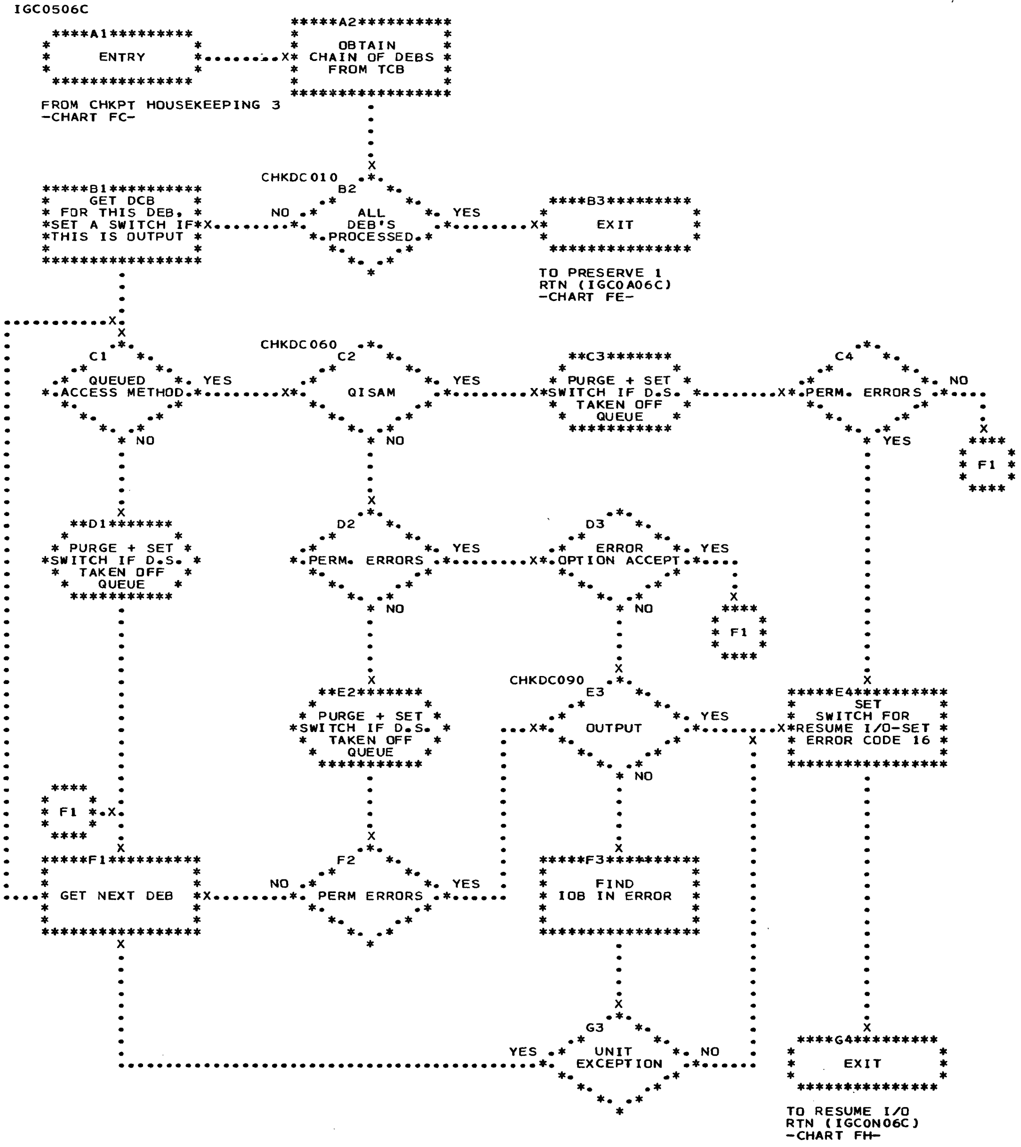
• **Chart FB. Checkpoint Housekeeping 2 Routine**


IGC0106C
A1
ENTRY
FROM CHKPT HOUSEKEEPING 1 -CHART FA-
B1
GET DCB. FOR THIS DEB, SET A SWITCH IF THIS IS OUTPUT
YES
NO
B2
KEY LENGTH = 0
NO
YES
D1
B3
SET MESSAGE CODE AND RETURN CODE
B3
C3
EXIT
TO CHKPT EXIT RTN (IGC0Q06C) -CHART FH-
B4
B4
ACTIVE TQE
YES
NO
B3
C1
CHKPT DATA SET ON TAPE
NO
YES
B3
D1
C4
IS THERE AN IRB OR STRB ON RB CHAIN
YES
NO
B3
D1
CHKPT D.S. RECFM = UNDEFINED
NO
YES
B3
D4
IS THERE A TYPE 3 OR 4 SVRB ON RB CHAIN
NO
F4
YES
E1
CHKPT D.S. BLKSIZE ABOVE MINIMUM
YES
H1
NO
E4
IS IT EOV MODULE
NO
YES
F4
B3
F1
CHKPT D.S. BLKSIZE = 0
NO
YES
B3
F3
JOB INVOKING ROLLOUT
YES
NO
B3
F4
RUNNING ON MVT SYSTEM
YES
NO
G1
DEVTYPE
GET MAX BLKSIZE ALLOWD ON DEV. FOR CHKPT D.S.
G3
IS JOB STEP A MOTHER OR DAUGHTER TASK
YES
NO
B3
G4
VALID USER-SUPPLIED CHECKID
NO
YES
B3
H1
H1
CHKPT D.S. OPENED FOR WRITE
NO
YES
B3
H2
INDICATE POSSIBLE ERROR IN RETURN CODE AND MESSAGE CODE
H3
ANY OUTSTANDING ENQUEUES
YES
NO
H4
EXIT
TO HOUSEKEEPING 3 RTN (IGC0206C) -CHART FC-
J1
CHKPT D.S. DSORG = BSAM OR BPAM
NO
YES
B3
B4
J3
ANY OUTSTANDING WORK
YES
NO
B3

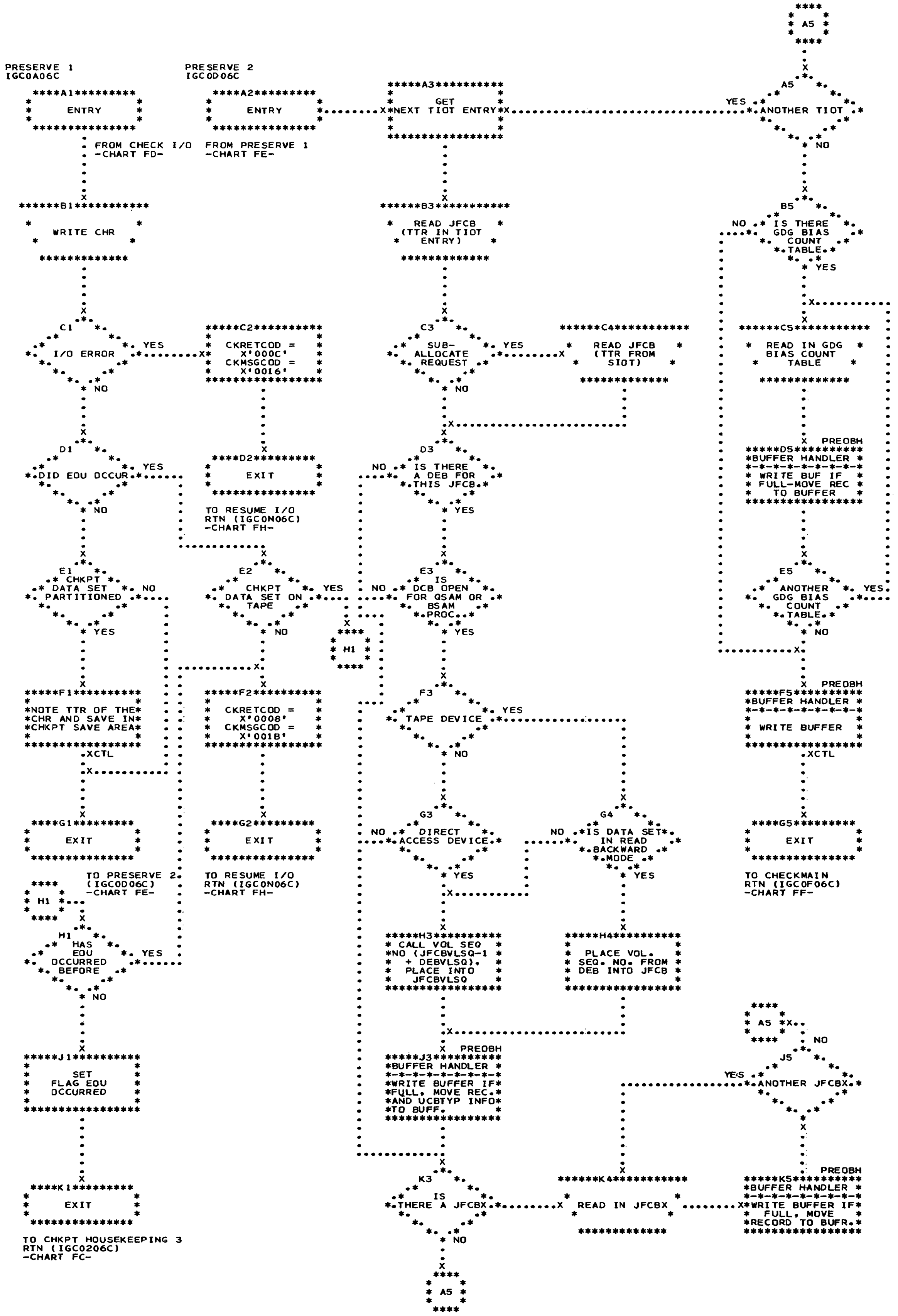
• Chart FC. Checkpoint Housekeeping 3 Routine


IGC0206C
A1
ENTRY
FROM CHKPT HOUSEKEEPING 1 -CHART FA- OR CHKPT HOUSEKEEPING 2 -CHART FB-
B1
CONSTRUCT ECB, IOB, CHANNEL PROGRAM TO BE USED TO WRITE RCDS + STOR
C1
GET TTR OF JCT
D1
CVTPCNVT
CONVERT TTR OF JCT
E1
XDAP
READ JCT INTO BUFFER OF CHART WORK AREA
F1
WAIT
G1
SUCCESSFUL COMPLETION OF I/O
NO
YES
G2
SET MESSAGE CODE AND RETURN CODE
G3
EXIT
TO CHKPT EXIT RTN (IGC0Q06C) -CHART FH-
H1
REQUEST CHKPT CANCEL
YES
NO
H2
EXIT
TO CHKPT EXIT RTN (IGC0Q06C) -CHART FH-
J1
INCREMENT JCTNRCKP FIELD (NO. OF CHKPT'S TAKEN) IN JCT BY ONE
B4
B4
B4
REQUEST SYSTEM-GENERATED CHECKID
YES
NO
B5
GENERATE CHECKID
C4
CONSTRUCT CHR (CHKPT HEADER RCD) IN BUFFER OF CHKPT WORK AREA
D4
REQ RET OF SYS-GENERATED CHKID TO USER
YES
NO
D5
MOVE SYSTEM-GENERATED CHECKID TO USER-PROVIDED FIELD
E4
PAD REMAINING BUFFER WITH ONES
F4
EXIT
TO CHECK I/O RTN (IGC0506C) -CHART FD-

• Chart FD. Check I/O Routine

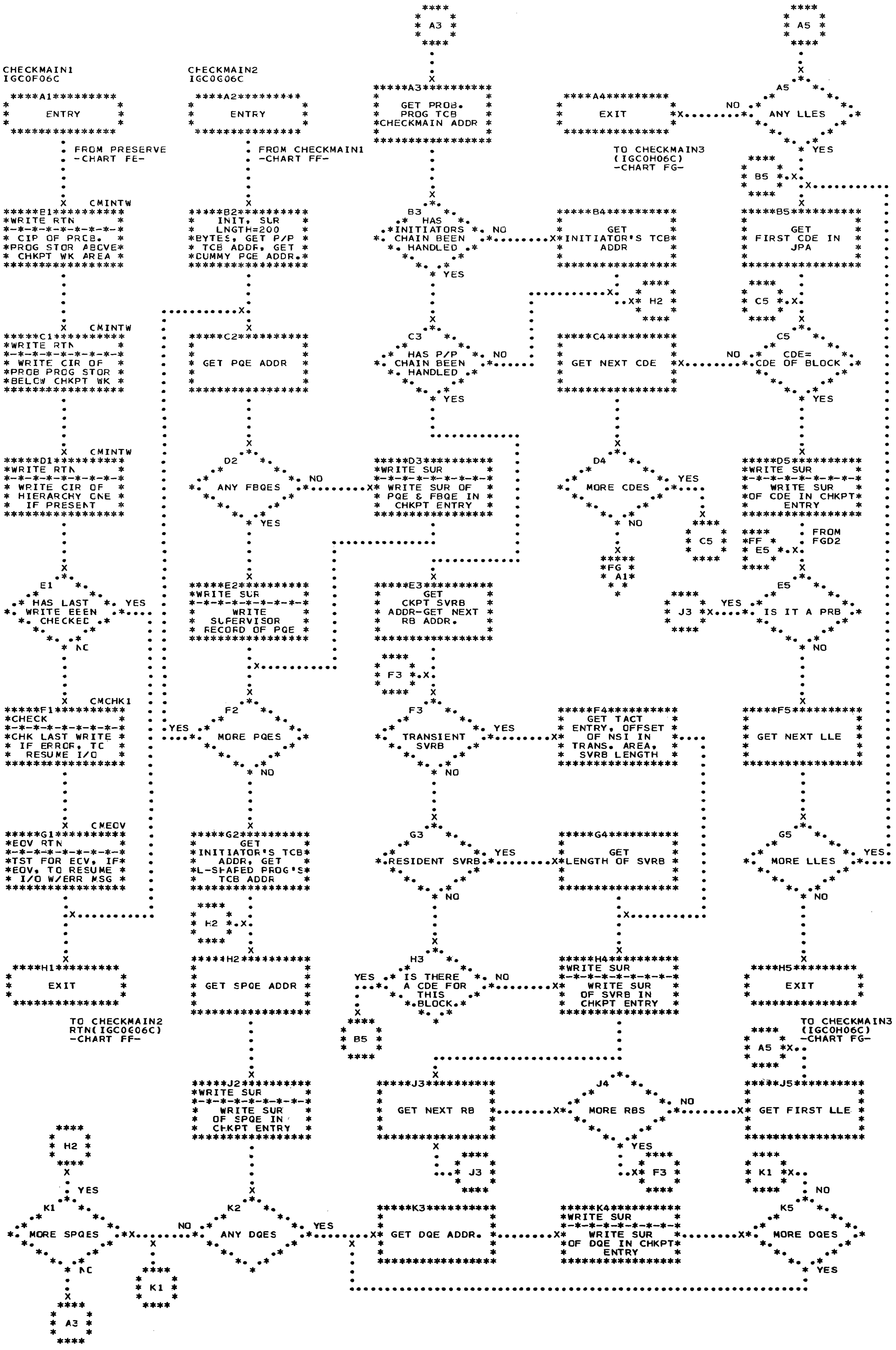

IGC0506C
A1
ENTRY
FROM CHKPT HOUSEKEEPING 3 -CHART FC-
A2
OBTAIN CHAIN OF DEBS FROM TCB
CHKDC010
B2
ALL DEB'S PROCESSED
YES
NO
B3
EXIT
TO PRESERVE 1 RTN (IGC0A06C) -CHART FE-
B1
GET DCB FOR THIS DEB. SET A SWITCH IF THIS IS OUTPUT
C1
QUEUED ACCESS METHOD
YES
NO
CHKDC060
C2
QISAM
YES
NO
C3
PURGE + SET SWITCH IF D.S. TAKEN OFF QUEUE
C4
PERM. ERRORS
NO
YES
F1
D1
PURGE + SET SWITCH IF D.S. TAKEN OFF QUEUE
D2
PERM. ERRORS
YES
NO
D3
ERROR OPTION ACCEPT
YES
NO
F1
E2
PURGE + SET SWITCH IF D.S. TAKEN OFF QUEUE
CHKDC090
E3
OUTPUT
YES
NO
E4
SET SWITCH FOR RESUME I/O-SET ERROR CODE 16
F1
F1
GET NEXT DEB
F2
PERM ERRORS
NO
YES
F3
FIND IOB IN ERROR
G3
UNIT EXCEPTION
YES
NO
G4
EXIT
TO RESUME I/O RTN (IGCON06C) -CHART FH-

• Chart FE. Preserve 1 and 2 Routines

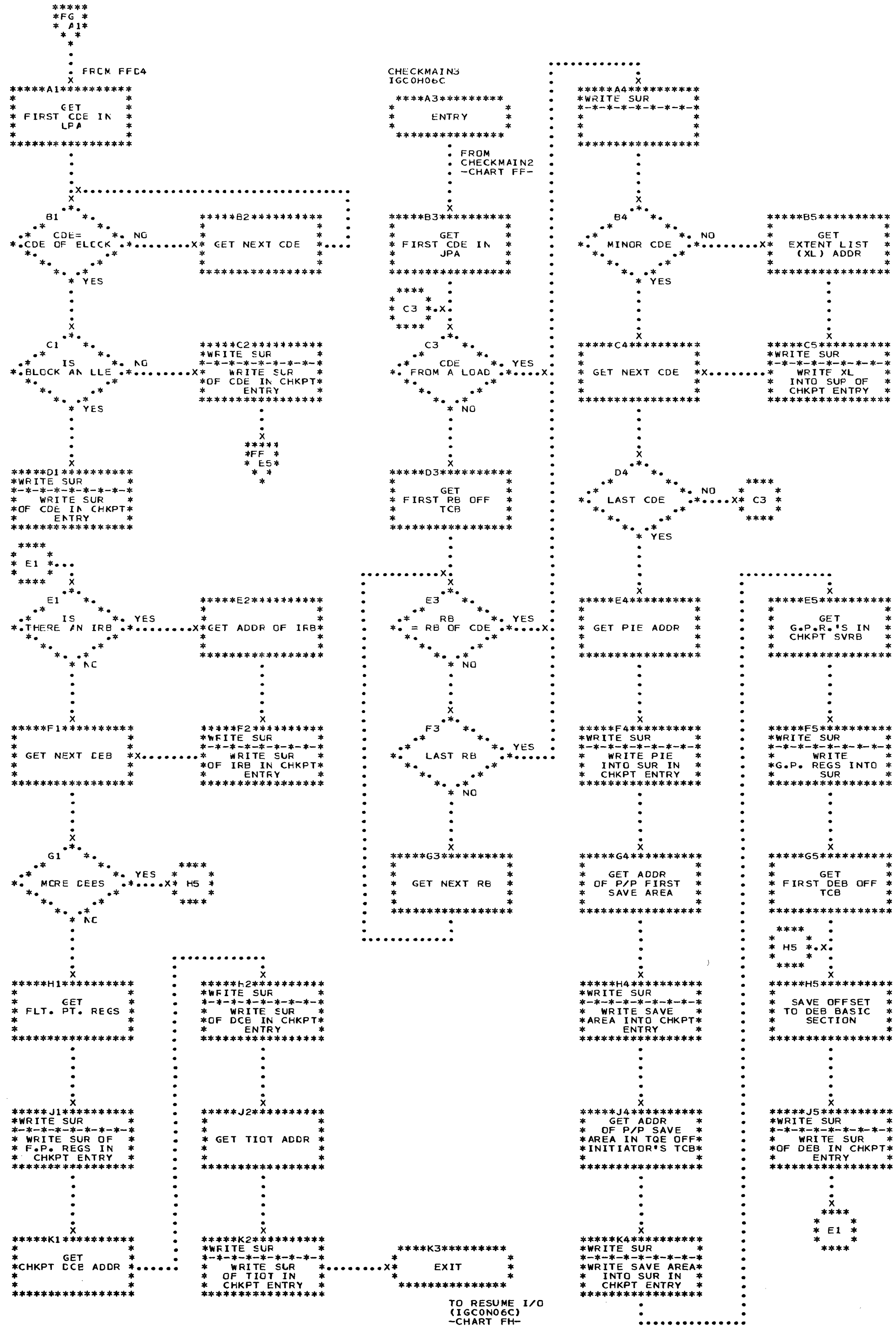


A5
PRESERVE 1
IGC0A06C
PRESERVE 2
IGC0D06C
A1
ENTRY
A2
ENTRY
A3
GET NEXT TIOT ENTRY
A5
YES
ANOTHER TIOT
NO
FROM CHECK I/O -CHART FD-
FROM PRESERVE 1 -CHART FE-
B1
WRITE CHR
B3
READ JFCB (TTR IN TIOT ENTRY)
B5
NO
IS THERE GDG BIAS COUNT TABLE
YES
C1
I/O ERROR
YES
NO
C2
CKRETCOD = X'000C' CKMSGCOD = X'0016'
C3
SUB-ALLOCATE REQUEST
YES
NO
C4
READ JFCB (TTR FROM SIOT)
C5
READ IN GDG BIAS COUNT TABLE
D1
DID EOU OCCUR
YES
NO
D2
EXIT
TO RESUME I/O RTN (IGC0N06C) -CHART FH-
D3
NO
IS THERE A DEB FOR THIS JFCB
YES
PREOBH
D5
BUFFER HANDLER
WRITE BUF IF FULL-MOVE REC TO BUFFER
E1
CHKPT DATA SET PARTITIONED
NO
YES
E2
CHKPT DATA SET ON TAPE
YES
NO
E3
NO
IS DCB OPEN FOR QSAM OR BSAM PROC.
YES
E5
ANOTHER GDG BIAS COUNT TABLE
YES
NO
H1
F1
NOTE TTR OF THE CHR AND SAVE IN CHKPT SAVE AREA
XCTL
F2
CKRETCOD = X'0008' CKMSGCOD = X'001B'
F3
TAPE DEVICE
YES
NO
PREOBH
F5
BUFFER HANDLER
WRITE BUFFER
XCTL
G1
EXIT
TO PRESERVE 2 (IGC0D06C) -CHART FE-
G2
EXIT
TO RESUME I/O RTN (IGC0N06C) -CHART FH-
G3
NO
DIRECT ACCESS DEVICE
YES
G4
NO
IS DATA SET IN READ BACKWARD MODE
YES
G5
EXIT
TO CHECKMAIN RTN (IGC0F06C) -CHART FF-
H1
H1
HAS EOU OCCURRED BEFORE
YES
NO
H3
CALL VOL SEQ NO (JFCBVLSQ-1 + DEBVLSQ), PLACE INTO JFCBVLSQ
H4
PLACE VOL. SEQ. NO. FROM DEB INTO JFCB
A5
NO
J5
YES
ANOTHER JFCBX
J1
SET FLAG EOU OCCURRED
PREOBH
J3
BUFFER HANDLER
WRITE BUFFER IF FULL. MOVE REC. AND UCBTYP INFO TO BUFF.
K1
EXIT
TO CHKPT HOUSEKEEPING 3 RTN (IGC0206C) -CHART FC-
K3
IS THERE A JFCBX
NO
K4
READ IN JFCBX
PREOBH
K5
BUFFER HANDLER
WRITE BUFFER IF FULL. MOVE RECORD TO BUFR.
A5

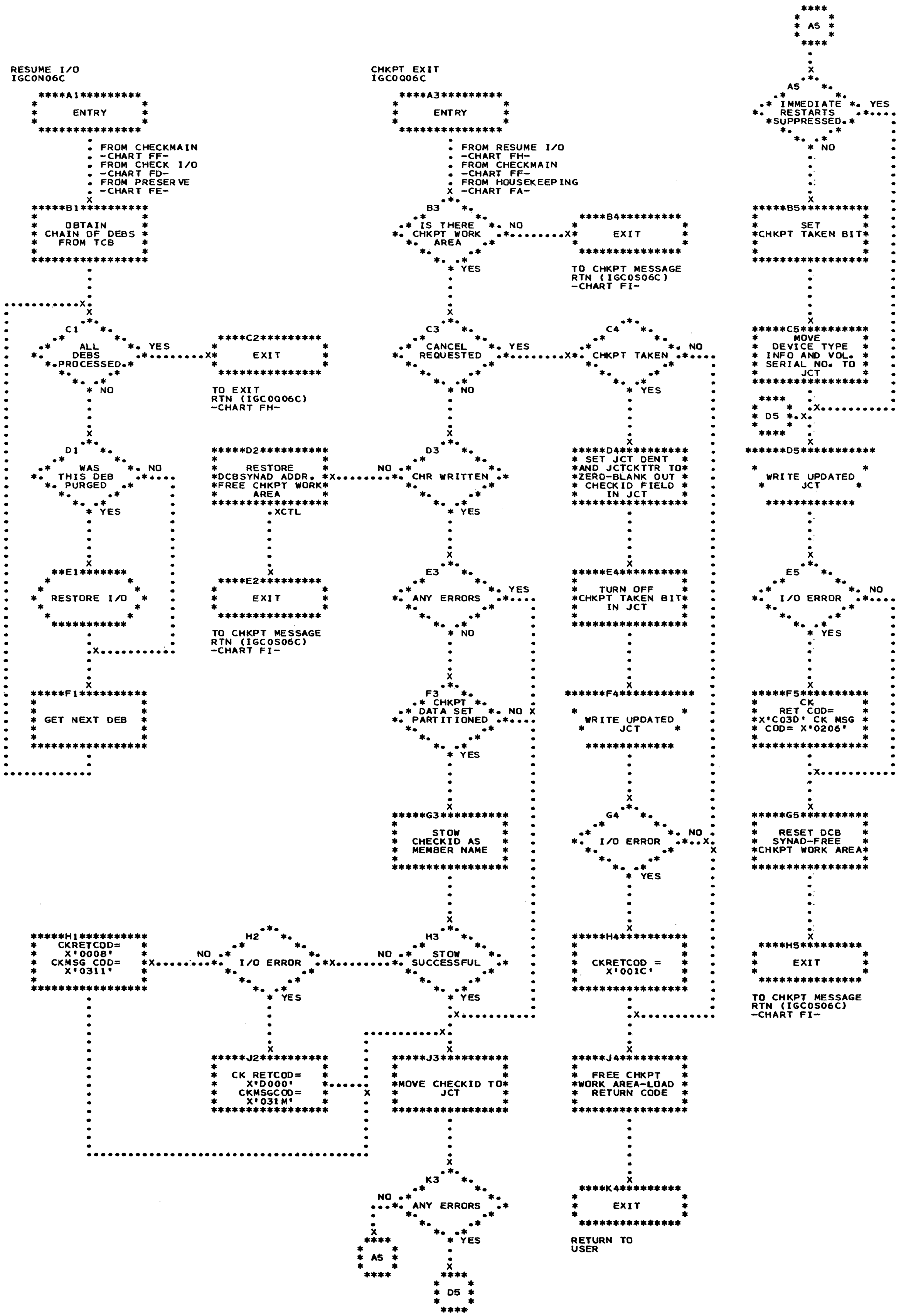
• Chart FF. Checkmain 1 and 2 Routines


A3
A5
CHECKMAIN1
IGC0F06C
A1
ENTRY
FROM PRESERVE -CHART FE-
CHECKMAIN2
IGC0G06C
A2
ENTRY
FROM CHECKMAIN1 -CHART FF-
A3
GET PROB. PROG TCB *CHECKMAIN ADDR
A4
EXIT
TO CHECKMAIN3 (IGC0H06C) -CHART FG-
A5
NO
ANY LLES
YES
B5
CMINTW
B1
WRITE RTN
CIP OF PROB. PROG STOR ABCVE CHKPT WK AREA
B2
INIT. SUR LNGTH=200 BYTES. GET P/P TCB ADDR. GET DUMMY PQE ADDR.
B3
HAS INITIATORS CHAIN BEEN HANDLED
NO
YES
B4
GET INITIATOR'S TCB ADDR
B5
GET FIRST CDE IN JPA
H2
C5
CMINTW
C1
WRITE RTN
WRITE CIR OF PROB PROG STOR BELOW CHKPT WK
C2
GET PQE ADDR
C3
HAS P/P CHAIN BEEN HANDLED
NO
YES
C4
GET NEXT CDE
C5
NO
CDE= CDE OF BLOCK
YES
CMINTW
D1
WRITE RTN
WRITE CIR OF HIERARCHY ONE IF PRESENT
D2
ANY FBQES
NO
YES
D3
WRITE SUR
WRITE SUR OF PQE & FBQE IN CHKPT ENTRY
D4
MORE CDES
YES
NO
C5
FG A1
D5
WRITE SUR
WRITE SUR OF CDE IN CHKPT ENTRY
FF E5
FROM FGD2
E1
HAS LAST WRITE BEEN CHECKED
YES
NO
E2
WRITE SUR
WRITE SUPERVISOR RECORD OF PQE
E3
GET CKPT SVRB ADDR-GET NEXT RB ADDR.
J3
YES
E5
IS IT A PRB
NO
F3
CMCHK1
F1
CHECK
CHK LAST WRITE IF ERROR, TO RESUME I/O
F2
YES
MORE PQES
NO
F3
TRANSIENT SVRB
YES
NO
F4
GET TACT ENTRY, OFFSET OF NSI IN TRANS. AREA, SVRB LENGTH
F5
GET NEXT LLE
CMEOV
G1
EOV RTN
TST FOR EOV, IF EOV, TO RESUME I/O W/ERR MSG
G2
GET INITIATOR'S TCB ADDR. GET L-SHAPED PROG'S TCB ADDR
G3
RESIDENT SVRB
YES
NO
G4
GET LENGTH OF SVRB
G5
MORE LLES
YES
NO
H2
H1
EXIT
TO CHECKMAIN2 RTN(IGC0G06C) -CHART FF-
H2
GET SPQE ADDR
H3
YES
IS THERE A CDE FOR THIS BLOCK
NO
B5
H4
WRITE SUR
WRITE SUR OF SVRB IN CHKPT ENTRY
H5
EXIT
TO CHECKMAIN3 (IGC0H06C) -CHART FG-
A5
J2
WRITE SUR
WRITE SUR OF SPQE IN CHKPT ENTRY
J3
GET NEXT RB
J4
MORE RBS
NO
YES
J5
GET FIRST LLE
H2
J3
F3
K1
YES
K1
MORE SPQES
NO
NC
A3
K1
K2
ANY DQES
YES
K3
GET DQE ADDR.
K4
WRITE SUR
WRITE SUR OF DQE IN CHKPT ENTRY
K5
NO
MORE DQES
YES

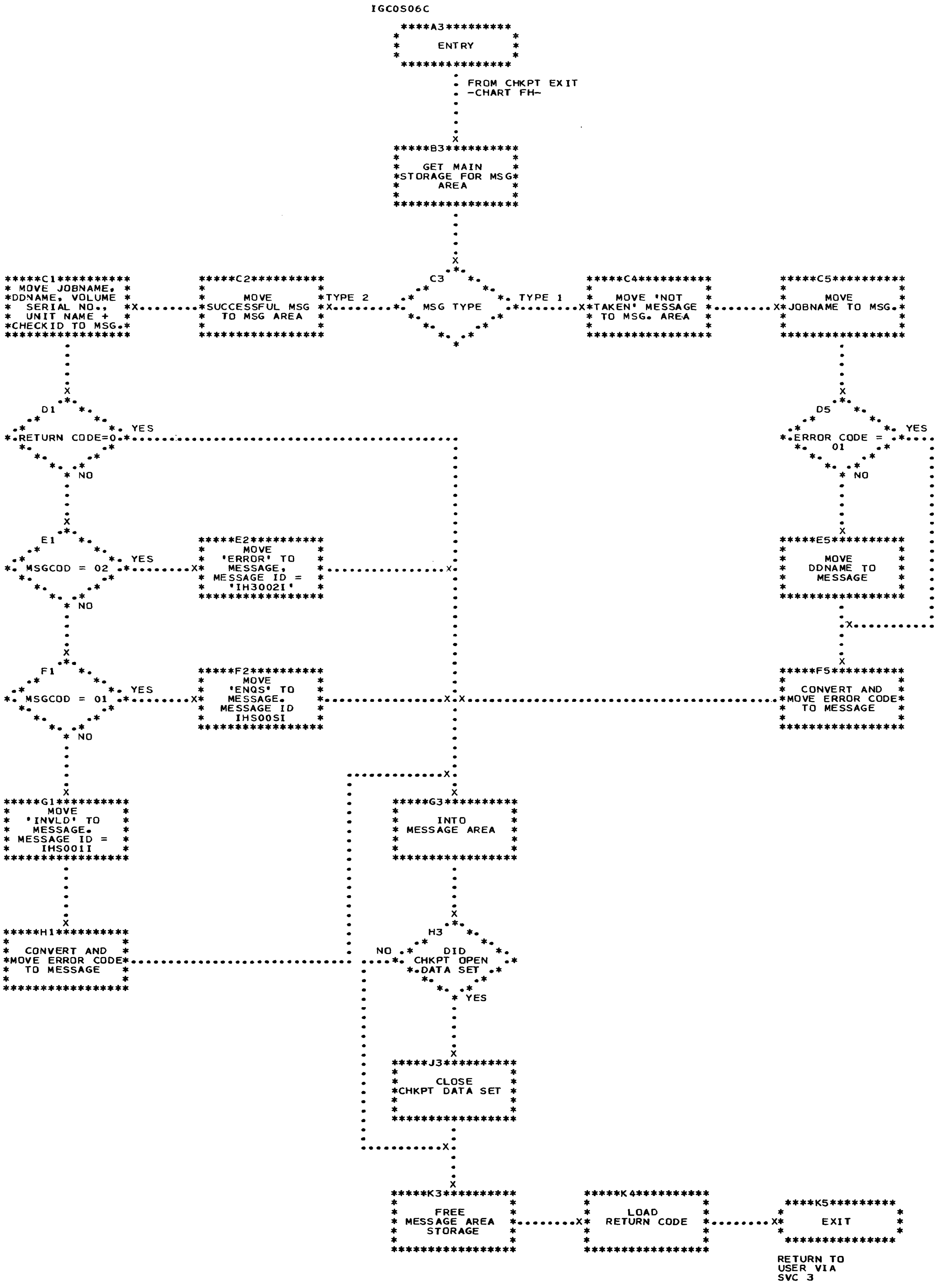
• Chart FG. Checkmain 3 Routine


FG A1
FROM FFC4
A1 GET FIRST CDE IN LPA
B1 CDE= CDE OF BLOCK
NO
B2 GET NEXT CDE
YES
C1 IS BLOCK AN LLE
NO
C2 WRITE SUR WRITE SUR OF CDE IN CHKPT ENTRY
FF E5
YES
D1 WRITE SUR WRITE SUR OF CDE IN CHKPT ENTRY
E1
E1 IS THERE AN IRB
YES
E2 GET ADDR OF IRB
NO
F1 GET NEXT DEB
F2 WRITE SUR WRITE SUR OF IRB IN CHKPT ENTRY
G1 MORE DEBS
YES
H5
NO
H1 GET FLT. PT. REGS
J1 WRITE SUR WRITE SUR OF F.P. REGS IN CHKPT ENTRY
K1 GET CHKPT DCB ADDR
H2 WRITE SUR WRITE SUR OF DCB IN CHKPT ENTRY
J2 GET TIOT ADDR
K2 WRITE SUR WRITE SUR OF TIOT IN CHKPT ENTRY
CHECKMAIN3
IGC0H06C
A3 ENTRY
FROM CHECKMAIN2 -CHART FF-
B3 GET FIRST CDE IN JPA
C3
C3 CDE FROM A LOAD
YES
NO
D3 GET FIRST RB OFF TCB
E3 RB = RB OF CDE
YES
NO
F3 LAST RB
YES
NO
G3 GET NEXT RB
K3 EXIT
TO RESUME I/O (IGC0N06C) -CHART FH-
A4 WRITE SUR
B4 MINOR CDE
NO
B5 GET EXTENT LIST (XL) ADDR
YES
C4 GET NEXT CDE
C5 WRITE SUR WRITE XL INTO SUP OF CHKPT ENTRY
D4 LAST CDE
NO
C3
YES
E4 GET PIE ADDR
F4 WRITE SUR WRITE PIE INTO SUR IN CHKPT ENTRY
G4 GET ADDR OF P/P FIRST SAVE AREA
H4 WRITE SUR WRITE SAVE AREA INTO CHKPT ENTRY
J4 GET ADDR OF P/P SAVE AREA IN TQE OFF INITIATOR'S TCB
K4 WRITE SUR WRITE SAVE AREA INTO SUR IN CHKPT ENTRY
E5 GET G.P.R.'S IN CHKPT SVRB
F5 WRITE SUR WRITE G.P. REGS INTO SUR
G5 GET FIRST DEB OFF TCB
H5
H5 SAVE OFFSET TO DEB BASIC SECTION
J5 WRITE SUR WRITE SUR OF DEB IN CHKPT ENTRY
E1

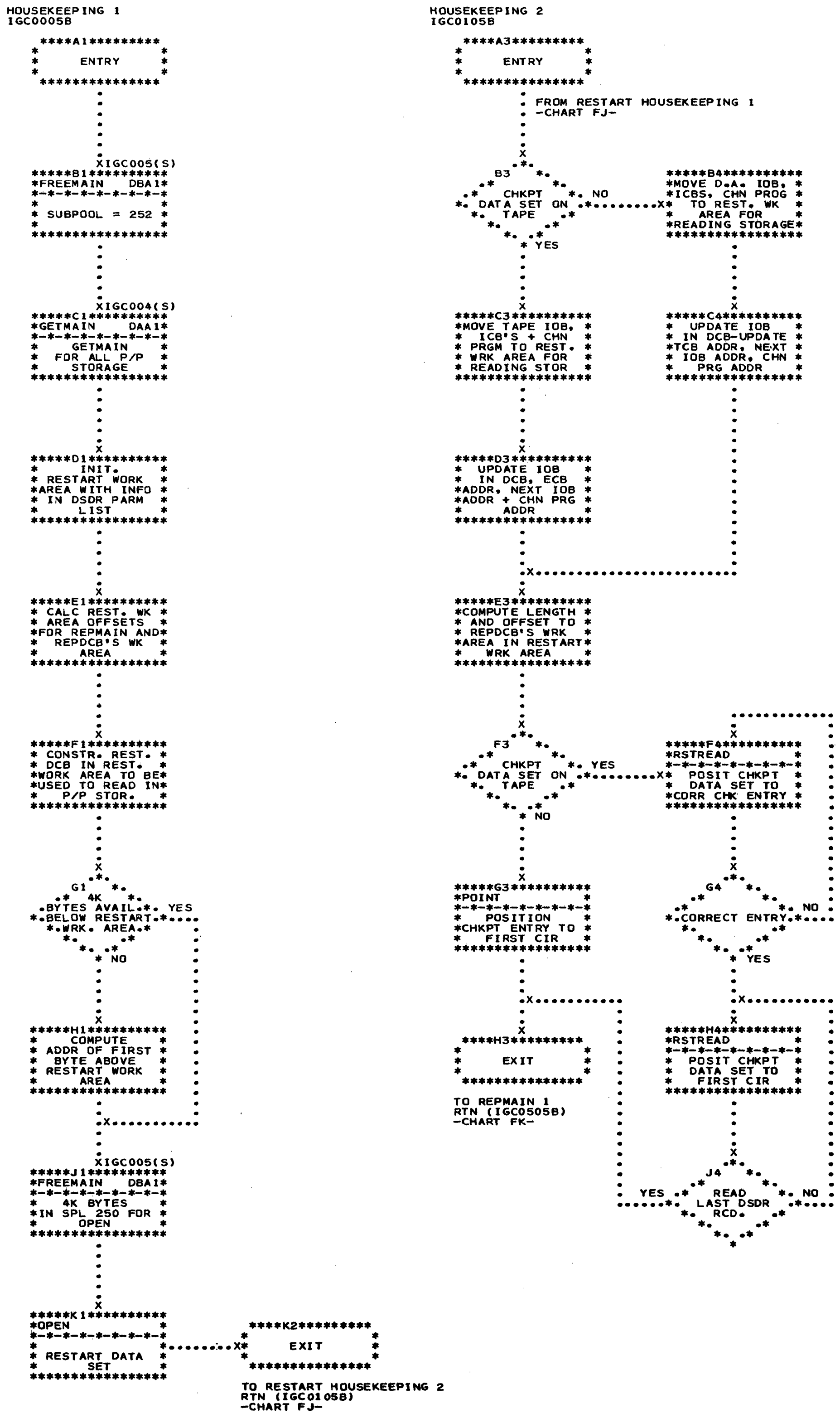
• Chart FH. Resume I/O and Checkpoint Exit Routines


RESUME I/O
IGC0N06C
A1
ENTRY
FROM CHECKMAIN -CHART FF-
FROM CHECK I/O -CHART FD-
FROM PRESERVE -CHART FE-
B1
OBTAIN CHAIN OF DEBS FROM TCB
C1
ALL DEBS PROCESSED
YES
NO
C2
EXIT
TO EXIT RTN (IGC0Q06C) -CHART FH-
D1
WAS THIS DEB PURGED
NO
YES
E1
RESTORE I/O
F1
GET NEXT DEB
H1
CKRETCOD= X'0008' CKMSG COD= X'0311'
CHKPT EXIT
IGC0Q06C
A3
ENTRY
FROM RESUME I/O -CHART FH-
FROM CHECKMAIN -CHART FF-
FROM HOUSEKEEPING -CHART FA-
B3
IS THERE CHKPT WORK AREA
NO
YES
B4
EXIT
TO CHKPT MESSAGE RTN (IGC0S06C) -CHART FI-
C3
CANCEL REQUESTED
YES
NO
C4
CHKPT TAKEN
NO
YES
D2
RESTORE DCBSYNAD ADDR. FREE CHKPT WORK AREA
XCTL
E2
EXIT
TO CHKPT MESSAGE RTN (IGC0S06C) -CHART FI-
D3
CHR WRITTEN
NO
YES
D4
SET JCT DENT AND JCTCKTTR TO ZERO-BLANK OUT CHECKID FIELD IN JCT
E3
ANY ERRORS
YES
NO
E4
TURN OFF CHKPT TAKEN BIT IN JCT
F3
CHKPT DATA SET PARTITIONED
NO
YES
F4
WRITE UPDATED JCT
G3
STOW CHECKID AS MEMBER NAME
G4
I/O ERROR
NO
YES
H2
I/O ERROR
NO
YES
H3
STOW SUCCESSFUL
NO
YES
H4
CKRETCOD = X'001C'
J2
CK RETCOD= X'D000' CKMSGCOD= X'031M'
J3
MOVE CHECKID TO JCT
J4
FREE CHKPT WORK AREA-LOAD RETURN CODE
K3
ANY ERRORS
NO
YES
A5
D5
K4
EXIT
RETURN TO USER
A5
A5
IMMEDIATE RESTARTS SUPPRESSED
YES
NO
B5
SET CHKPT TAKEN BIT
C5
MOVE DEVICE TYPE INFO AND VOL. SERIAL NO. TO JCT
D5
D5
WRITE UPDATED JCT
E5
I/O ERROR
NO
YES
F5
CK RET COD= X'C03D' CK MSG COD= X'0206'
G5
RESET DCB SYNAD-FREE CHKPT WORK AREA
H5
EXIT
TO CHKPT MESSAGE RTN (IGC0S06C) -CHART FI-

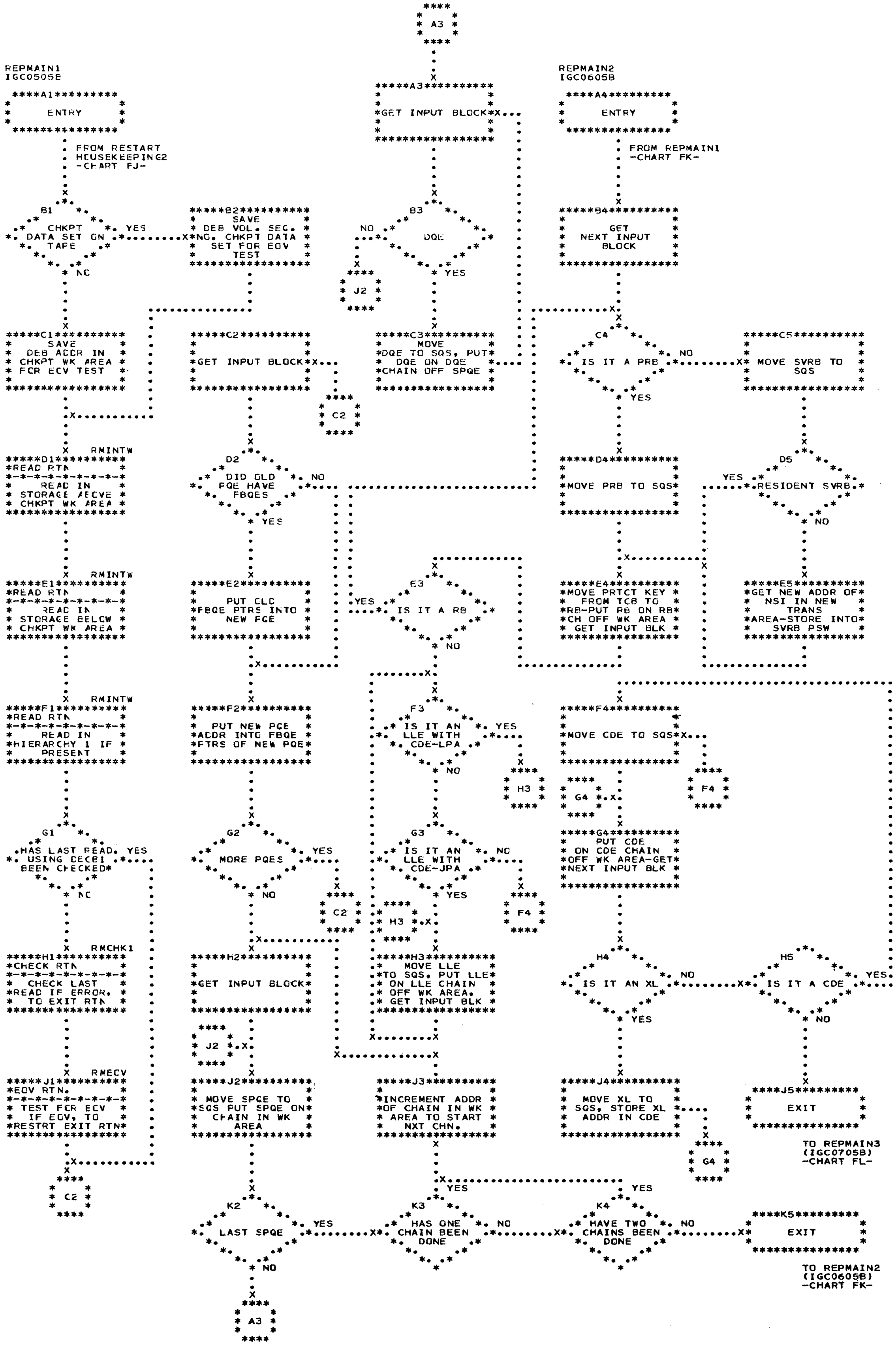
• Chart FI. Checkpoint Message Module


IGC0S06C
A3
ENTRY
FROM CHKPT EXIT -CHART FH-
B3
GET MAIN STORAGE FOR MSG AREA
C1
MOVE JOBNAME, DDNAME, VOLUME SERIAL NO., UNIT NAME + CHECKID TO MSG.
C2
MOVE SUCCESSFUL MSG TO MSG AREA
TYPE 2
C3
MSG TYPE
TYPE 1
C4
MOVE 'NOT TAKEN' MESSAGE TO MSG. AREA
C5
MOVE JOBNAME TO MSG.
D1
RETURN CODE=0
YES
NO
D5
ERROR CODE = 01
YES
NO
E1
MSGCOD = 02
YES
NO
E2
MOVE 'ERROR' TO MESSAGE, MESSAGE ID = 'IH3002I'
E5
MOVE DDNAME TO MESSAGE
F1
MSGCOD = 01
YES
NO
F2
MOVE 'ENQS' TO MESSAGE. MESSAGE ID IHS00SI
F5
CONVERT AND MOVE ERROR CODE TO MESSAGE
G1
MOVE 'INVLD' TO MESSAGE. MESSAGE ID = IHS001I
G3
INTO MESSAGE AREA
H1
CONVERT AND MOVE ERROR CODE TO MESSAGE
H3
NO
DID CHKPT OPEN DATA SET
YES
J3
CLOSE CHKPT DATA SET
K3
FREE MESSAGE AREA STORAGE
K4
LOAD RETURN CODE
K5
EXIT
RETURN TO USER VIA SVC 3

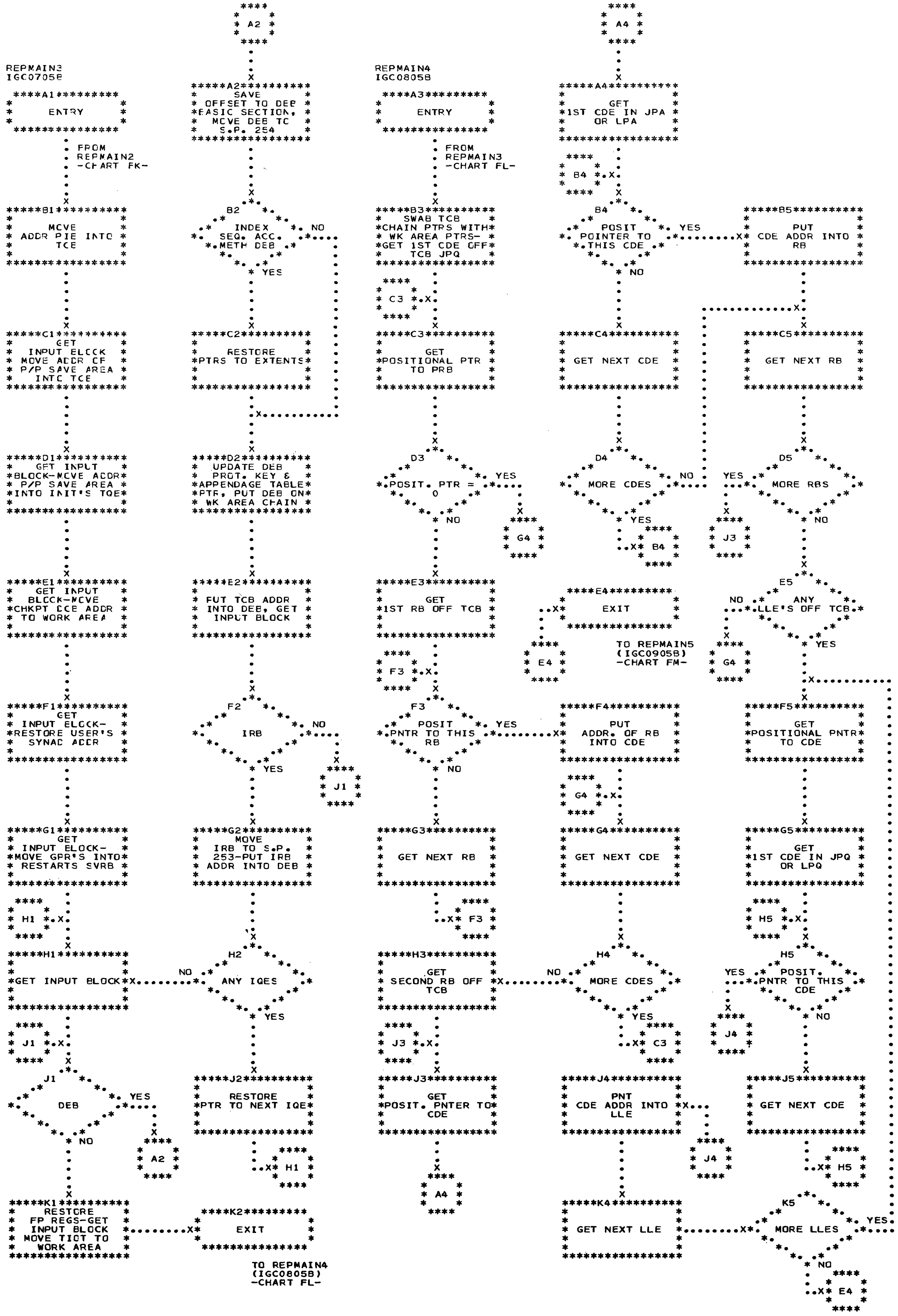
• Chart FJ. Restart Housekeeping 1 and 2 Routines


HOUSEKEEPING 1
IGC0005B
A1
ENTRY
XIGC005(S)
B1
FREEMAIN DBA1
SUBPOOL = 252
XIGC004(S)
C1
GETMAIN DAA1
GETMAIN FOR ALL P/P STORAGE
D1
INIT. RESTART WORK AREA WITH INFO IN DSDR PARM LIST
E1
CALC REST. WK AREA OFFSETS FOR REPMAIN AND REPDCB'S WK AREA
F1
CONSTR. REST. DCB IN REST. WORK AREA TO BE USED TO READ IN P/P STOR.
G1
4K BYTES AVAIL. BELOW RESTART WRK. AREA
YES
NO
H1
COMPUTE ADDR OF FIRST BYTE ABOVE RESTART WORK AREA
XIGC005(S)
J1
FREEMAIN DBA1
4K BYTES IN SPL 250 FOR OPEN
K1
OPEN
RESTART DATA SET
K2
EXIT
TO RESTART HOUSEKEEPING 2 RTN (IGC0105B) -CHART FJ-
HOUSEKEEPING 2
IGC0105B
A3
ENTRY
FROM RESTART HOUSEKEEPING 1 -CHART FJ-
B3
CHKPT DATA SET ON TAPE
NO
YES
B4
MOVE D.A. IOB, ICBS, CHN PROG TO REST. WK AREA FOR READING STORAGE
C3
MOVE TAPE IOB, ICB'S + CHN PRGM TO REST. WRK AREA FOR READING STOR
C4
UPDATE IOB IN DCB-UPDATE TCB ADDR, NEXT IOB ADDR, CHN PRG ADDR
D3
UPDATE IOB IN DCB, ECB ADDR, NEXT IOB ADDR + CHN PRG ADDR
E3
COMPUTE LENGTH AND OFFSET TO REPDCB'S WRK AREA IN RESTART WRK AREA
F3
CHKPT DATA SET ON TAPE
YES
NO
F4
RSTREAD
POSIT CHKPT DATA SET TO CORR CHK ENTRY
G3
POINT
POSITION CHKPT ENTRY TO FIRST CIR
G4
CORRECT ENTRY
NO
YES
H3
EXIT
TO REPMAIN 1 RTN (IGC0505B) -CHART FK-
H4
RSTREAD
POSIT CHKPT DATA SET TO FIRST CIR
J4
READ LAST DSDR RCD.
YES
NO

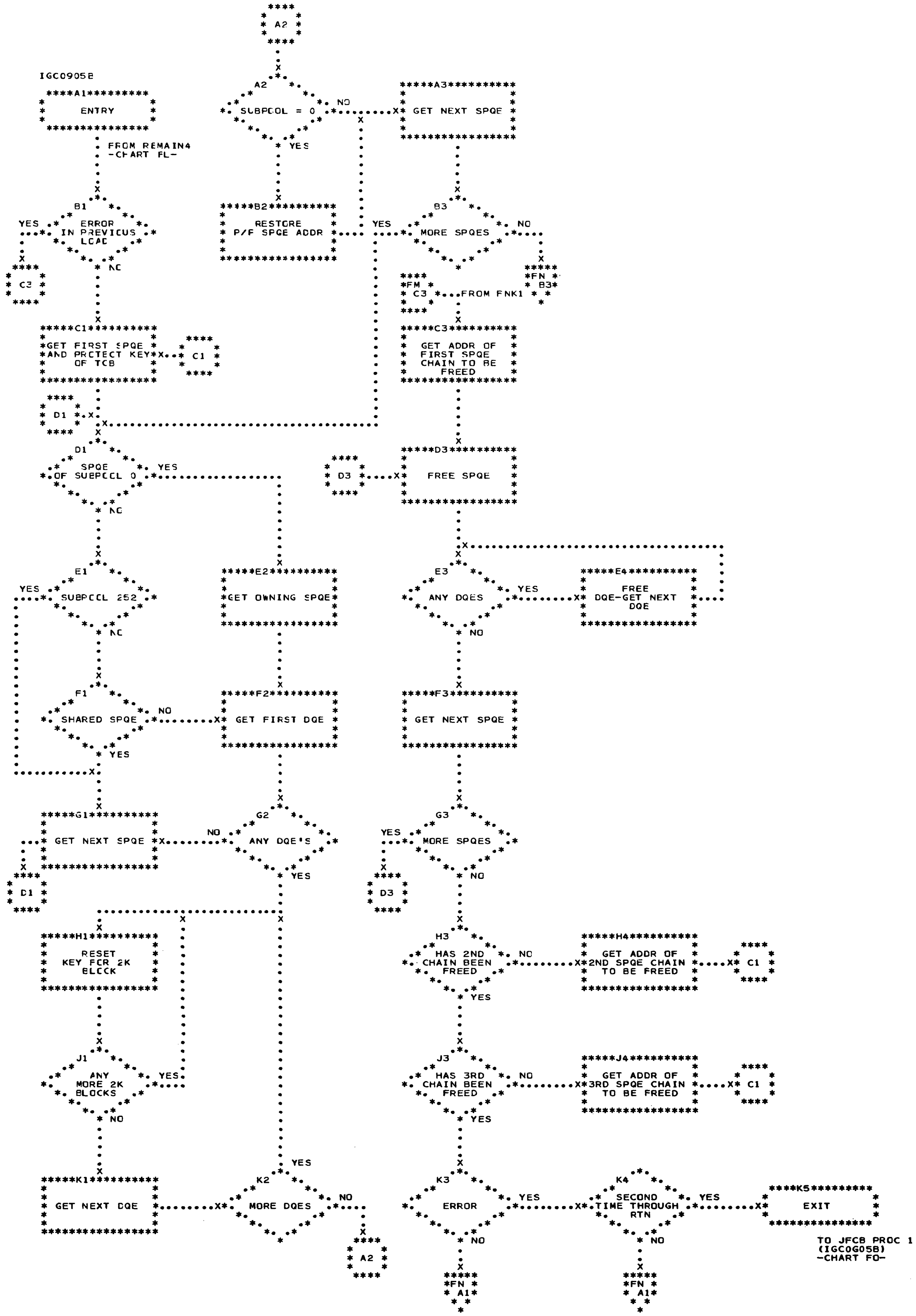
• Chart FK. Repmain 1 and 2 Routines


A3
REPMAIN1
IGC0505B
A1
ENTRY
FROM RESTART HOUSEKEEPING2 -CHART FJ-
B1
CHKPT DATA SET ON TAPE
YES
NO
B2
SAVE DEB VOL. SEC. NO. CHKPT DATA SET FOR EOV TEST
C1
SAVE DEB ADDR IN CHKPT WK AREA FOR EOV TEST
RMINTW
D1
READ RTN
READ IN STORAGE ABOVE CHKPT WK AREA
RMINTW
E1
READ RTN
READ IN STORAGE BELOW CHKPT WK AREA
RMINTW
F1
READ RTN
READ IN HIERARCHY 1 IF PRESENT
G1
HAS LAST READ. USING DECB1 BEEN CHECKED
YES
NO
RMCHK1
H1
CHECK RTN
CHECK LAST READ IF ERROR, TO EXIT RTN
RMECV
J1
EOV RTN.
TEST FOR EOV IF EOV, TO RESTRT EXIT RTN
C2
A3
GET INPUT BLOCK
B3
DQE
NO
YES
J2
C2
GET INPUT BLOCK
C2
C3
MOVE DQE TO SQS, PUT DQE ON DQE CHAIN OFF SPQE
D2
DID OLD FQE HAVE FBQES
NO
YES
E2
PUT OLD FBQE PTRS INTO NEW FQE
E3
IS IT A RB
YES
NO
F2
PUT NEW PQE ADDR INTO FBQE PTRS OF NEW PQE
F3
IS IT AN LLE WITH CDE-LPA
YES
NO
H3
G2
MORE PQES
YES
NO
C2
G3
IS IT AN LLE WITH CDE-JPA
NO
YES
F4
H3
H2
GET INPUT BLOCK
J2
H3
MOVE LLE TO SQS, PUT LLE ON LLE CHAIN OFF WK AREA, GET INPUT BLK
J2
MOVE SPQE TO SQS PUT SPQE ON CHAIN IN WK AREA
J3
INCREMENT ADDR OF CHAIN IN WK AREA TO START NXT CHN.
K2
LAST SPQE
YES
NO
A3
K3
HAS ONE CHAIN BEEN DONE
YES
NO
REPMAIN2
IGC0605B
A4
ENTRY
FROM REPMAIN1 -CHART FK-
B4
GET NEXT INPUT BLOCK
C4
IS IT A PRB
NO
YES
C5
MOVE SVRB TO SQS
D4
MOVE PRB TO SQS
D5
RESIDENT SVRB
YES
NO
E4
MOVE PRTCT KEY FROM TCB TO RB-PUT RB ON RB CH OFF WK AREA GET INPUT BLK
E5
GET NEW ADDR OF NSI IN NEW TRANS AREA-STORE INTO SVRB PSW
F4
MOVE CDE TO SQS
F4
G4
G4
PUT CDE ON CDE CHAIN OFF WK AREA-GET NEXT INPUT BLK
H4
IS IT AN XL
NO
YES
H5
IS IT A CDE
YES
NO
J4
MOVE XL TO SQS, STORE XL ADDR IN CDE
G4
J5
EXIT
TO REPMAIN3 (IGC0705B) -CHART FL-
K4
HAVE TWO CHAINS BEEN DONE
YES
NO
K5
EXIT
TO REPMAIN2 (IGC0605B) -CHART FK-

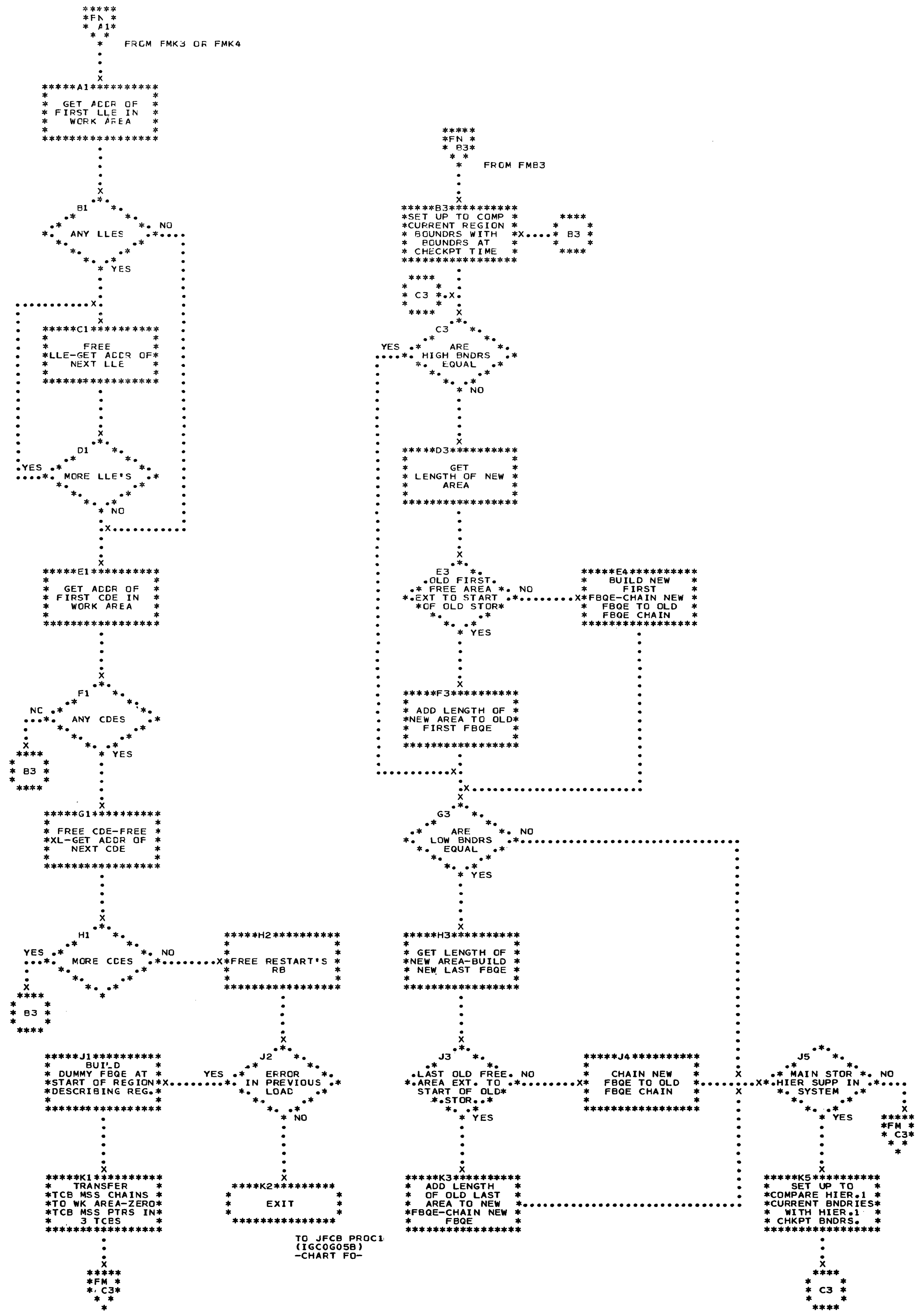
• Chart FL. Repmain 3 and 4 Routines


REPMAIN3
IGC0705B
A1 ENTRY
FROM REPMAIN2 -CHART FK-
B1 MOVE ADDR PIE INTO TCB
C1 GET INPUT BLOCK MOVE ADDR OF P/P SAVE AREA INTO TCB
D1 GET INPUT BLOCK-MOVE ADDR P/P SAVE AREA INTO INIT'S TCB
E1 GET INPUT BLOCK-MOVE CHKPT DCE ADDR TO WORK AREA
F1 GET INPUT BLOCK-RESTORE USER'S SYNAD ADDR
G1 GET INPUT BLOCK-MOVE GPR'S INTO RESTARTS SVRB
H1 GET INPUT BLOCK
J1 DEB
YES
NO
A2
K1 RESTORE FP REGS-GET INPUT BLOCK MOVE TIOT TO WORK AREA
A2
A2 SAVE OFFSET TO DEB BASIC SECTION, MOVE DEB TO S.P. 254
B2 INDEX SEQ. ACC. METH DEB
NO
YES
C2 RESTORE PTRS TO EXTENTS
D2 UPDATE DEB PROT. KEY & APPENDAGE TABLE PTR, PUT DEB ON WK AREA CHAIN
E2 PUT TCB ADDR INTO DEB, GET INPUT BLOCK
F2 IRB
NO
J1
YES
G2 MOVE IRB TO S.P. 253-PUT IRB ADDR INTO DEB
H2 ANY IQES
NO
YES
J2 RESTORE PTR TO NEXT IQE
H1
K2 EXIT
TO REPMAIN4 (IGC0805B) -CHART FL-
REPMAIN4
IGC0805B
A3 ENTRY
FROM REPMAIN3 -CHART FL-
B3 SWAB TCB CHAIN PTRS WITH WK AREA PTRS-GET 1ST CDE OFF TCB JPQ
C3
C3 GET POSITIONAL PTR TO PRB
D3 POSIT. PTR = 0
YES
G4
NO
E3 GET 1ST RB OFF TCB
F3
F3 POSIT PNTR TO THIS RB
YES
NO
G3 GET NEXT RB
F3
H3 GET SECOND RB OFF TCB
J3
J3 GET POSIT. PNTER TO CDE
A4
A4
A4 GET 1ST CDE IN JPA OR LPA
B4
B4 POSIT POINTER TO THIS CDE
YES
NO
C4 GET NEXT CDE
D4 MORE CDES
NO
YES
B4
E4 EXIT
E4
TO REPMAIN5 (IGC0905B) -CHART FM-
F4 PUT ADDR. OF RB INTO CDE
G4
G4 GET NEXT CDE
H4 MORE CDES
NO
YES
C3
J4 PNT CDE ADDR INTO LLE
J4
K4 GET NEXT LLE
B5 PUT CDE ADDR INTO RB
C5 GET NEXT RB
D5 MORE RBS
YES
J3
NO
E5 ANY LLE'S OFF TCB
NO
G4
YES
F5 GET POSITIONAL PNTR TO CDE
G5 GET 1ST CDE IN JPQ OR LPQ
H5
H5 POSIT. PNTR TO THIS CDE
YES
J4
NO
J5 GET NEXT CDE
H5
K5 MORE LLES
YES
NO
E4

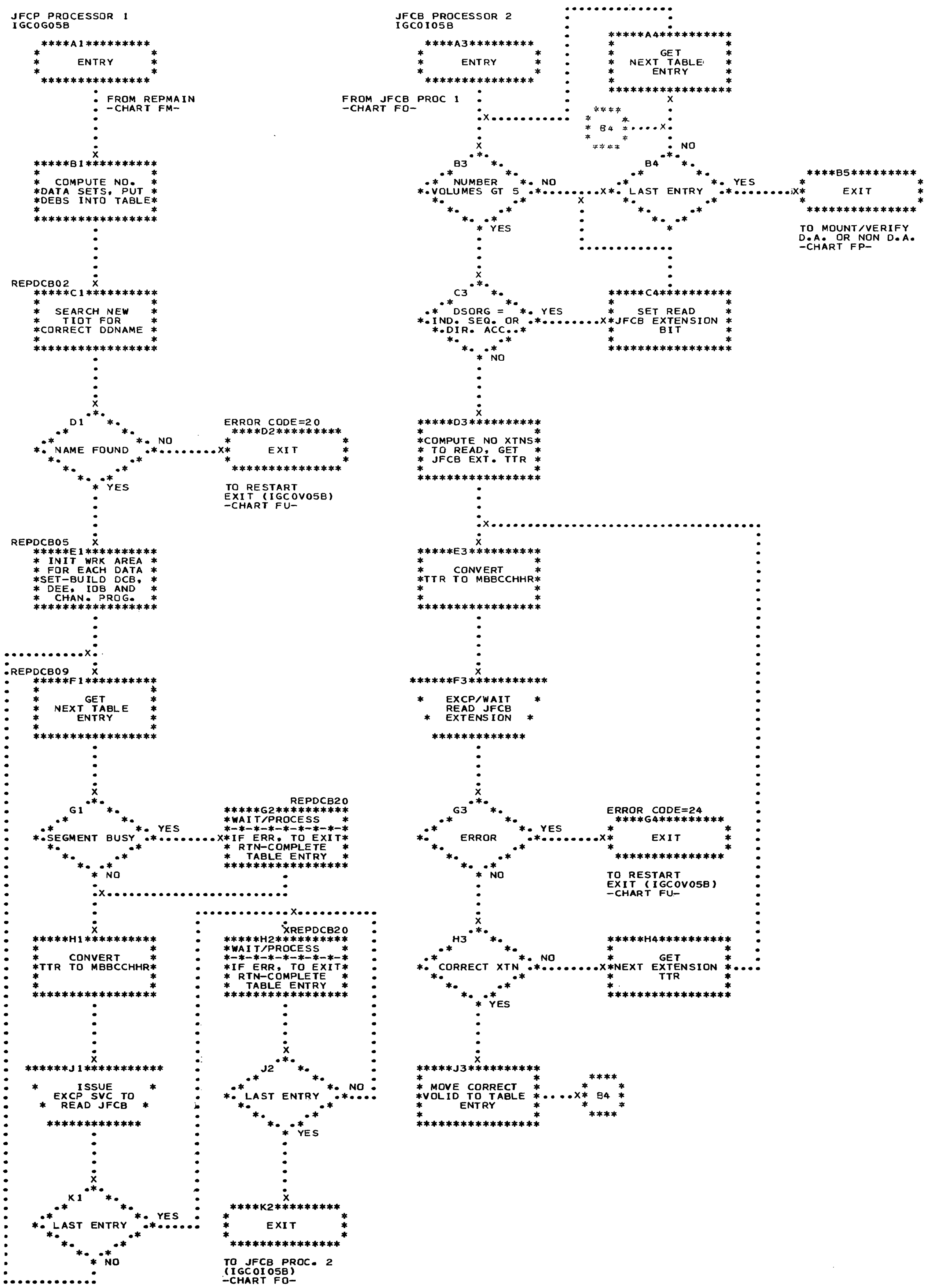
• Chart FM. Repmain 5 Routine (Part 1 of 2)



• Chart FN. Repmain 5 Routine (Part 2 of 2)

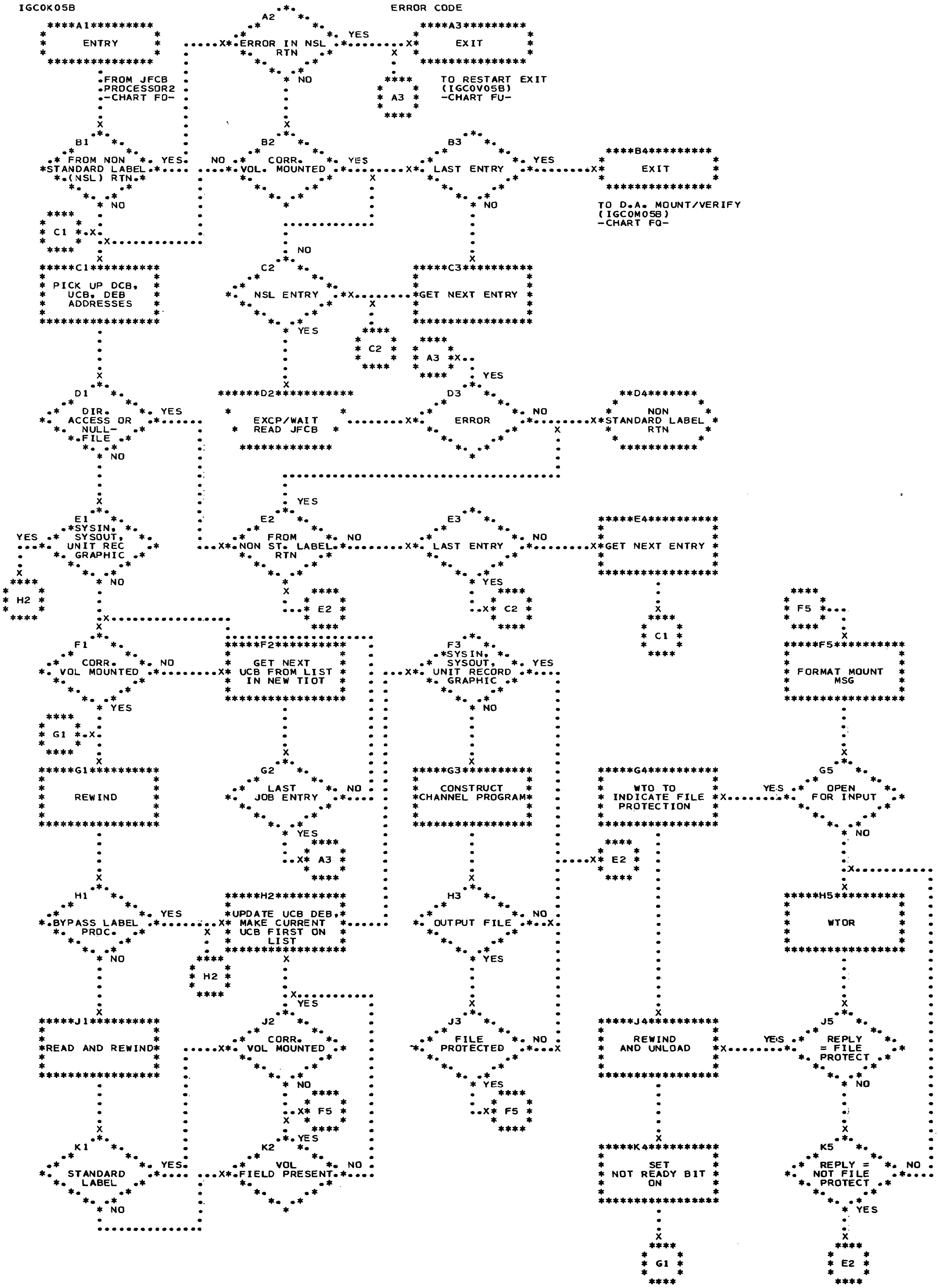

FN A1
FROM FMK3 OR FMK4
A1 GET ADDR OF FIRST LLE IN WORK AREA
B1 ANY LLES
NO
YES
C1 FREE LLE-GET ADDR OF NEXT LLE
D1 MORE LLE'S
YES
NO
E1 GET ADDR OF FIRST CDE IN WORK AREA
F1 ANY CDES
NO
B3
YES
G1 FREE CDE-FREE XL-GET ADDR OF NEXT CDE
H1 MORE CDES
YES
B3
NO
H2 FREE RESTART'S RB
J1 BUILD DUMMY FBQE AT START OF REGION DESCRIBING REG.
J2 ERROR IN PREVIOUS LOAD
YES
NO
K1 TRANSFER TCB MSS CHAINS TO WK AREA-ZERO TCB MSS PTRS IN 3 TCBS
FM C3
K2 EXIT
TO JFCB PROC1 (IGC0G05B) -CHART FO-
FN B3
FROM FMB3
B3 SET UP TO COMP CURRENT REGION BOUNDRS WITH BOUNDRS AT CHECKPT TIME
B3
C3
C3 ARE HIGH BNDRS EQUAL
YES
NO
D3 GET LENGTH OF NEW AREA
E3 OLD FIRST. FREE AREA EXT TO START OF OLD STOR
NO
YES
E4 BUILD NEW FIRST FBQE-CHAIN NEW FBQE TO OLD FBQE CHAIN
F3 ADD LENGTH OF NEW AREA TO OLD FIRST FBQE
G3 ARE LOW BNDRS EQUAL
NO
YES
H3 GET LENGTH OF NEW AREA-BUILD NEW LAST FBQE
J3 LAST OLD FREE. AREA EXT. TO START OF OLD STOR.
NO
YES
J4 CHAIN NEW FBQE TO OLD FBQE CHAIN
K3 ADD LENGTH OF OLD LAST AREA TO NEW FBQE-CHAIN NEW FBQE
J5 MAIN STOR HIER SUPP IN SYSTEM
NO
FM C3
YES
K5 SET UP TO COMPARE HIER.1 CURRENT BNDRIES WITH HIER.1 CHKPT BNDRS.
C3

• Chart FO. JFCB Processor 1 and 2 Routines

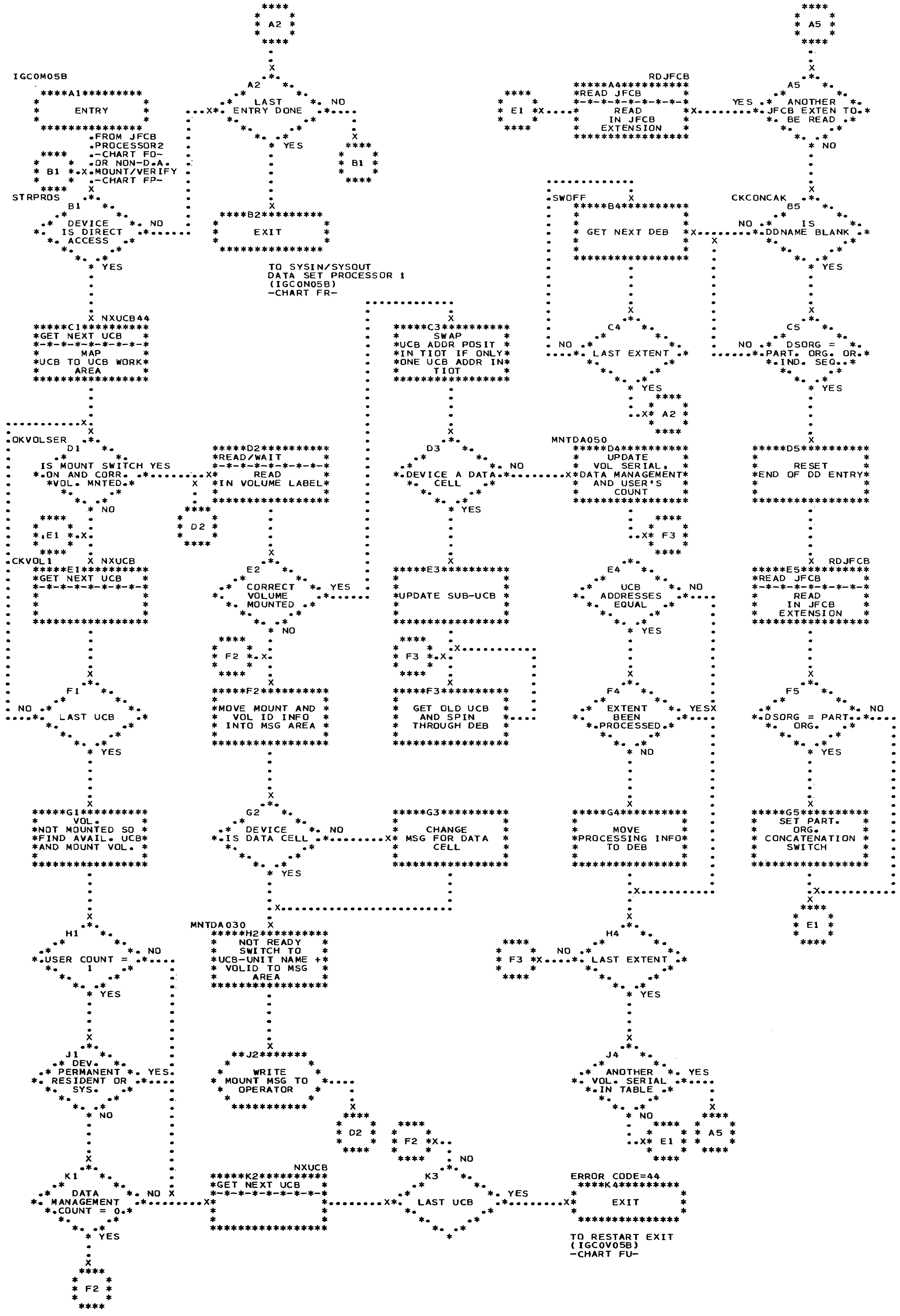


JFCP PROCESSOR 1
IGC0G05B
A1
ENTRY
FROM REPMAIN -CHART FM-
B1
COMPUTE NO. DATA SETS, PUT DEBS INTO TABLE
REPDCB02
C1
SEARCH NEW TIOT FOR CORRECT DDNAME
D1
NAME FOUND
NO
YES
ERROR CODE=20
D2
EXIT
TO RESTART EXIT (IGC0V05B) -CHART FU-
REPDCB05
E1
INIT WRK AREA FOR EACH DATA SET-BUILD DCB, DEE, IOB AND CHAN. PROG.
REPDCB09
F1
GET NEXT TABLE ENTRY
G1
SEGMENT BUSY
YES
NO
REPDCB20
G2
WAIT/PROCESS
IF ERR, TO EXIT RTN-COMPLETE TABLE ENTRY
H1
CONVERT TTR TO MBBCCHHR
REPDCB20
H2
WAIT/PROCESS
IF ERR, TO EXIT RTN-COMPLETE TABLE ENTRY
J1
ISSUE EXCP SVC TO READ JFCB
J2
LAST ENTRY
NO
YES
K1
LAST ENTRY
YES
NO
K2
EXIT
TO JFCB PROC. 2 (IGC0I05B) -CHART FO-
JFCB PROCESSOR 2
IGC0I05B
A3
ENTRY
FROM JFCB PROC 1 -CHART FO-
A4
GET NEXT TABLE ENTRY
B4
B3
NUMBER VOLUMES GT 5
NO
YES
B4
LAST ENTRY
YES
NO
B5
EXIT
TO MOUNT/VERIFY D.A. OR NON D.A. -CHART FP-
C3
DSORG = IND. SEQ. OR DIR. ACC.
YES
NO
C4
SET READ JFCB EXTENSION BIT
D3
COMPUTE NO XTNS TO READ, GET JFCB EXT. TTR
E3
CONVERT TTR TO MBBCCHHR
F3
EXCP/WAIT READ JFCB EXTENSION
G3
ERROR
YES
NO
ERROR CODE=24
G4
EXIT
TO RESTART EXIT (IGC0V05B) -CHART FU-
H3
CORRECT XTN
NO
YES
H4
GET NEXT EXTENSION TTR
J3
MOVE CORRECT VOLID TO TABLE ENTRY
B4

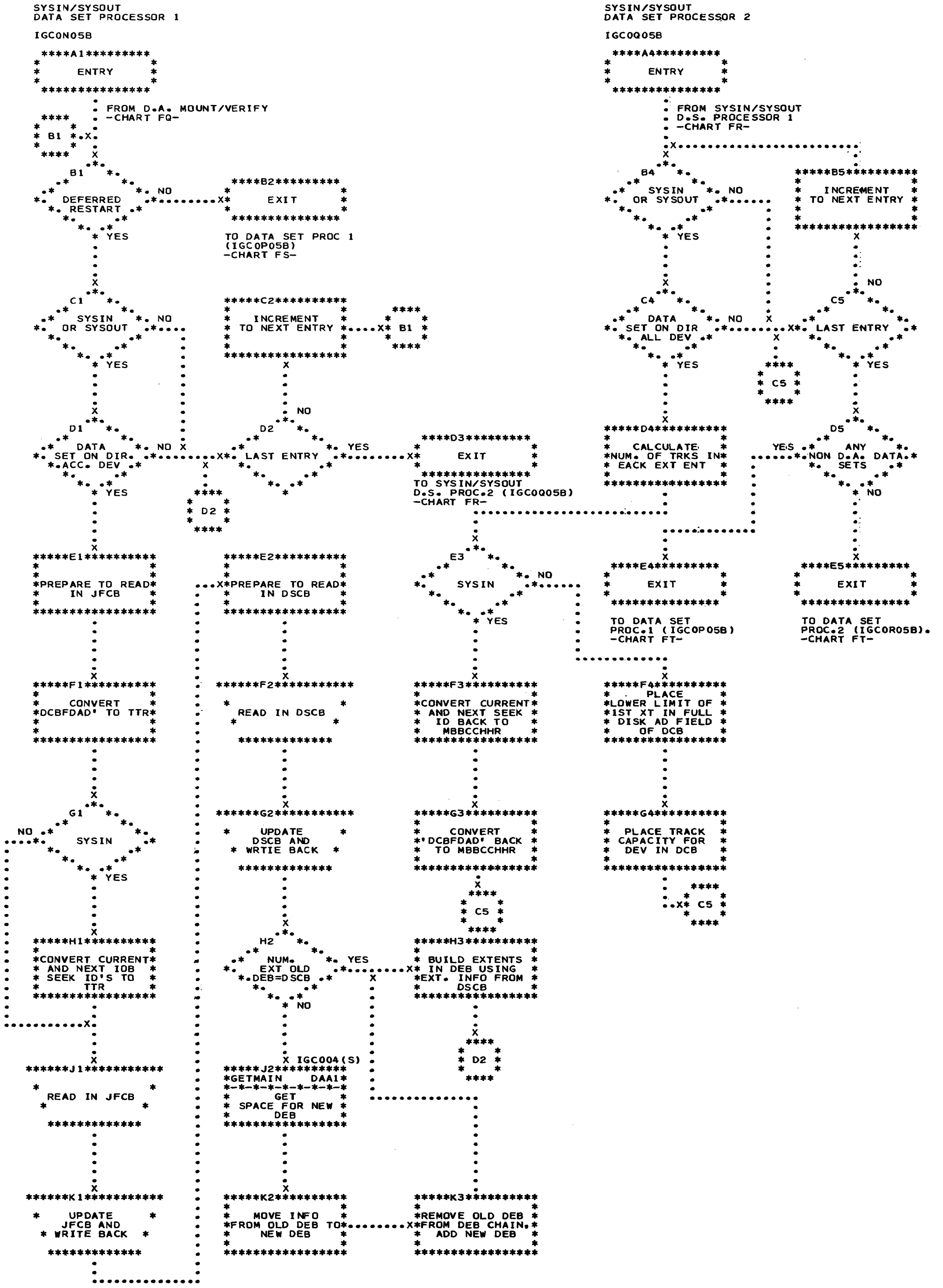
• Chart FP. Mount/Verify 1 (Non Direct-Access) Routine



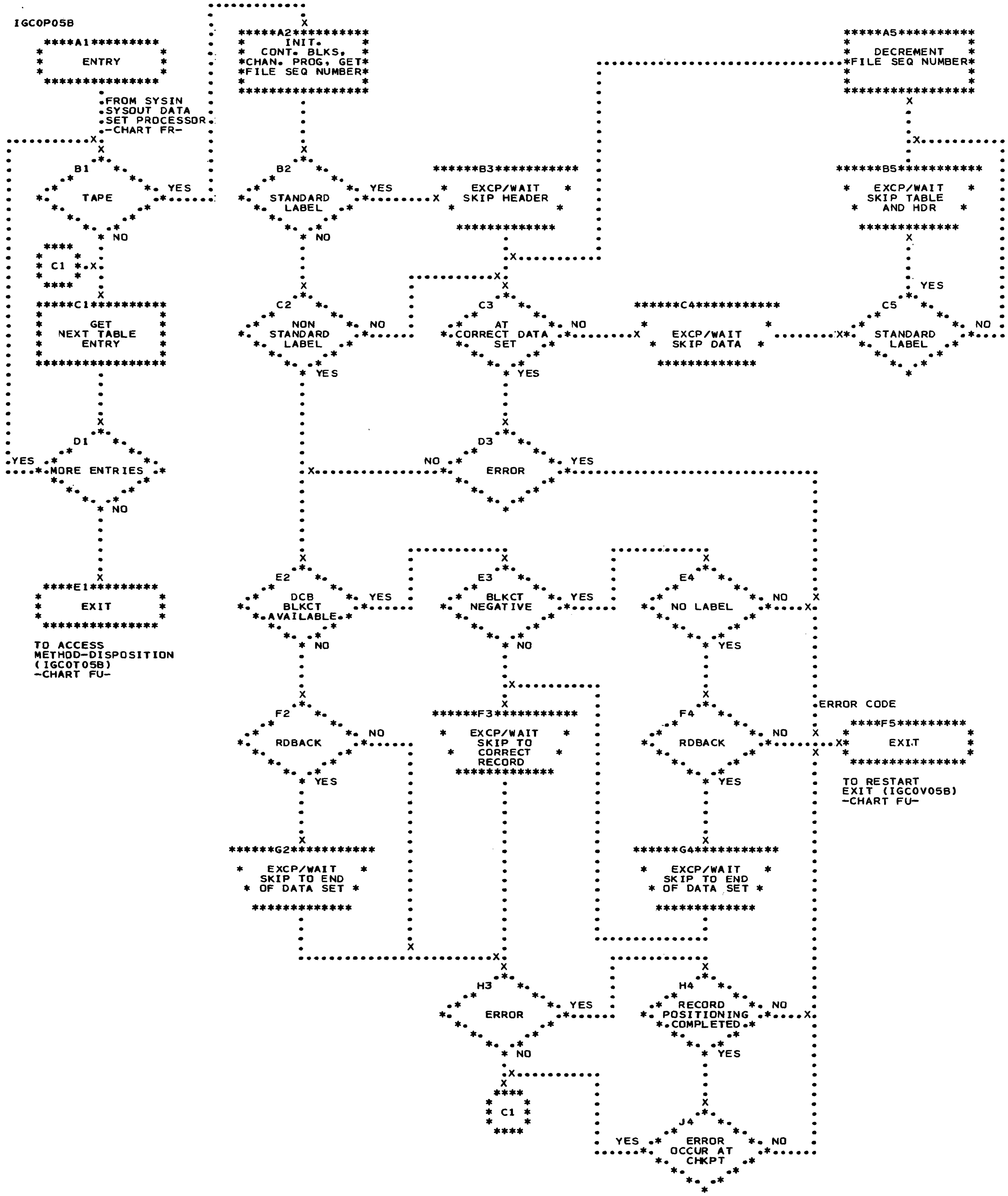
- Chart FQ. Mount/Verify 2 (Direct-Access) Routine



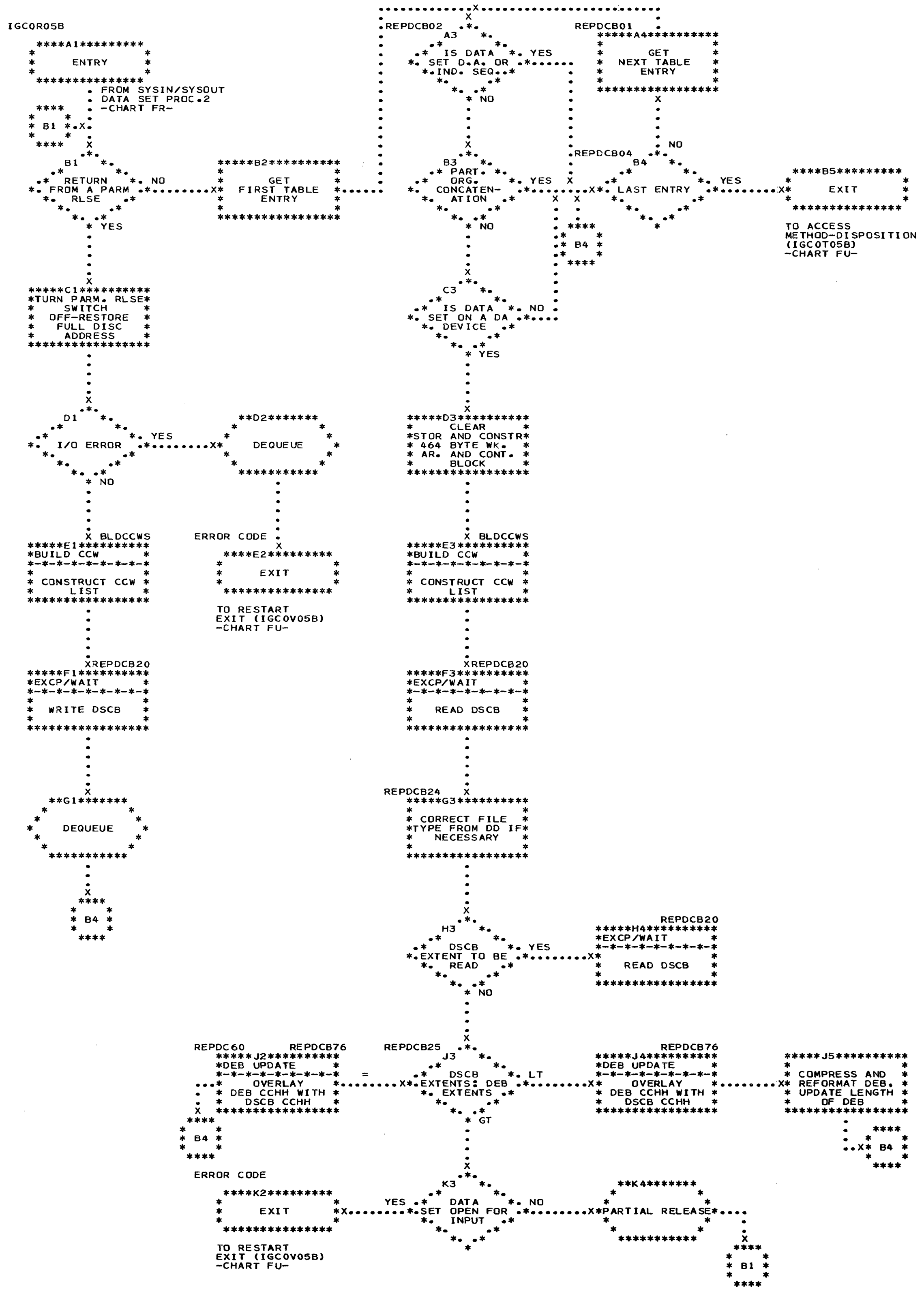
• Chart FR. Sysin/Sysout Data Set Processor 1 and 2 Routines


SYSIN/SYSOUT DATA SET PROCESSOR 1
IGCON05B
A1 ENTRY
FROM D.A. MOUNT/VERIFY -CHART FQ-
B1
B1 DEFERRED RESTART
NO
YES
B2 EXIT
TO DATA SET PROC 1 (IGCOP05B) -CHART FS-
C1 SYSIN OR SYSOUT
NO
YES
C2 INCREMENT TO NEXT ENTRY
B1
D1 DATA SET ON DIR. ACC. DEV
NO
YES
D2 LAST ENTRY
NO
YES
D2
D3 EXIT
TO SYSIN/SYSOUT D.S. PROC.2 (IGCOQ05B) -CHART FR-
E1 PREPARE TO READ IN JFCB
E2 PREPARE TO READ IN DSCB
E3 SYSIN
NO
YES
F1 CONVERT 'DCBFDAD' TO TTR
F2 READ IN DSCB
F3 CONVERT CURRENT AND NEXT SEEK ID BACK TO MBBCCHHR
G1 SYSIN
NO
YES
G2 UPDATE DSCB AND WRTIE BACK
G3 CONVERT 'DCBFDAD' BACK TO MBBCCHHR
C5
H1 CONVERT CURRENT AND NEXT IOB SEEK ID'S TO TTR
H2 NUM. EXT OLD DEB=DSCB
YES
NO
H3 BUILD EXTENTS IN DEB USING EXT. INFO FROM DSCB
D2
IGC004(S)
J1 READ IN JFCB
J2 GETMAIN DAA1 GET SPACE FOR NEW DEB
K1 UPDATE JFCB AND WRITE BACK
K2 MOVE INFO FROM OLD DEB TO NEW DEB
K3 REMOVE OLD DEB FROM DEB CHAIN, ADD NEW DEB
SYSIN/SYSOUT DATA SET PROCESSOR 2
IGCOQ05B
A4 ENTRY
FROM SYSIN/SYSOUT D.S. PROCESSOR 1 -CHART FR-
B4 SYSIN OR SYSOUT
NO
YES
B5 INCREMENT TO NEXT ENTRY
C4 DATA SET ON DIR ALL DEV
NO
YES
C5
C5 LAST ENTRY
NO
YES
D4 CALCULATE NUM. OF TRKS IN EACK EXT ENT
D5 ANY NON D.A. DATA SETS
YES
NO
E4 EXIT
TO DATA SET PROC.1 (IGCOP05B) -CHART FT-
E5 EXIT
TO DATA SET PROC.2 (IGCOR05B). -CHART FT-
F4 PLACE LOWER LIMIT OF 1ST XT IN FULL DISK AD FIELD OF DCB
G4 PLACE TRACK CAPACITY FOR DEV IN DCB
C5

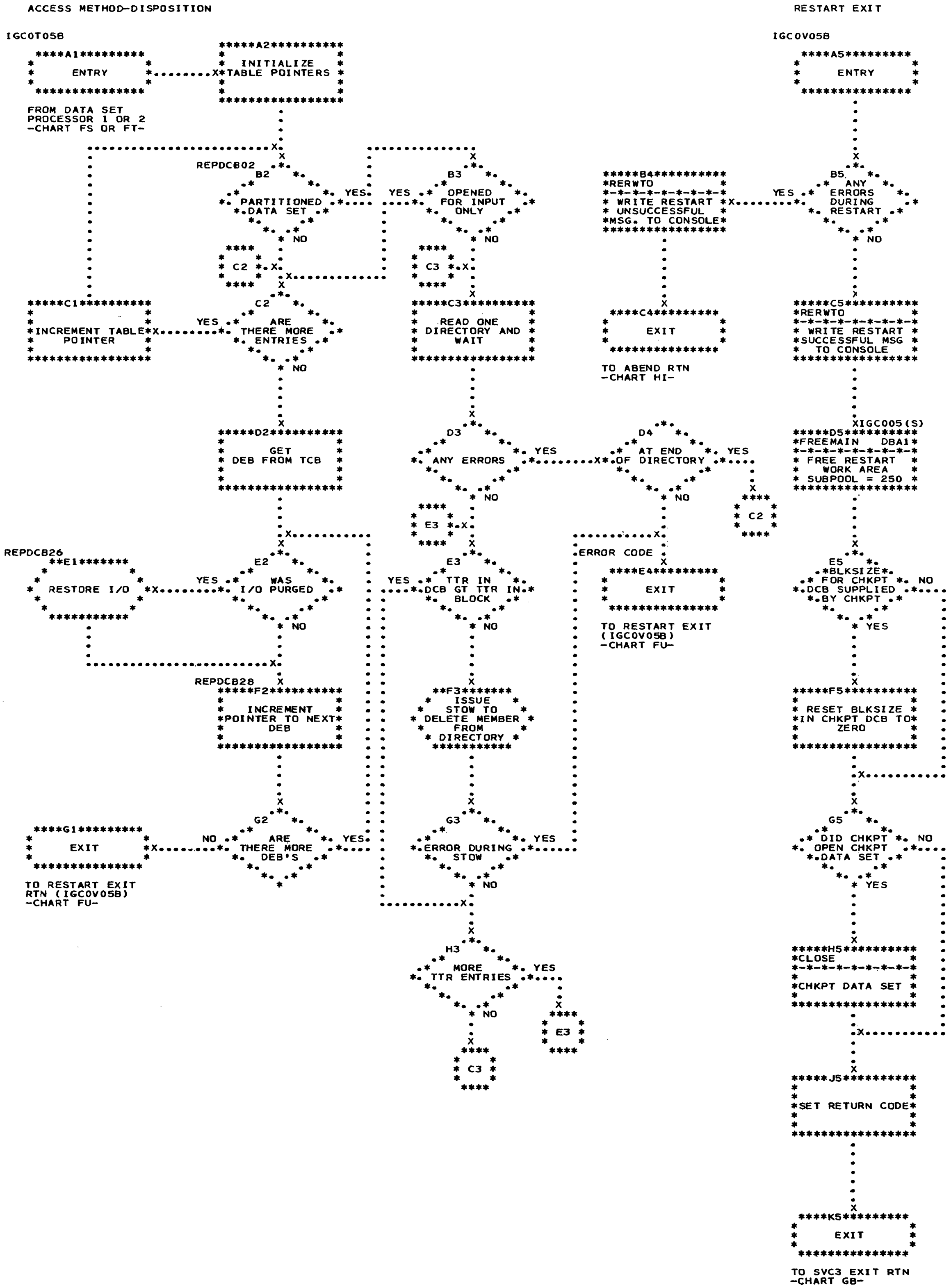
• Chart FS. Data Set Processor 1 (Non Direct-Access) Routine



• Chart FT. Data Set Processor 2 (Direct-Access) Routine

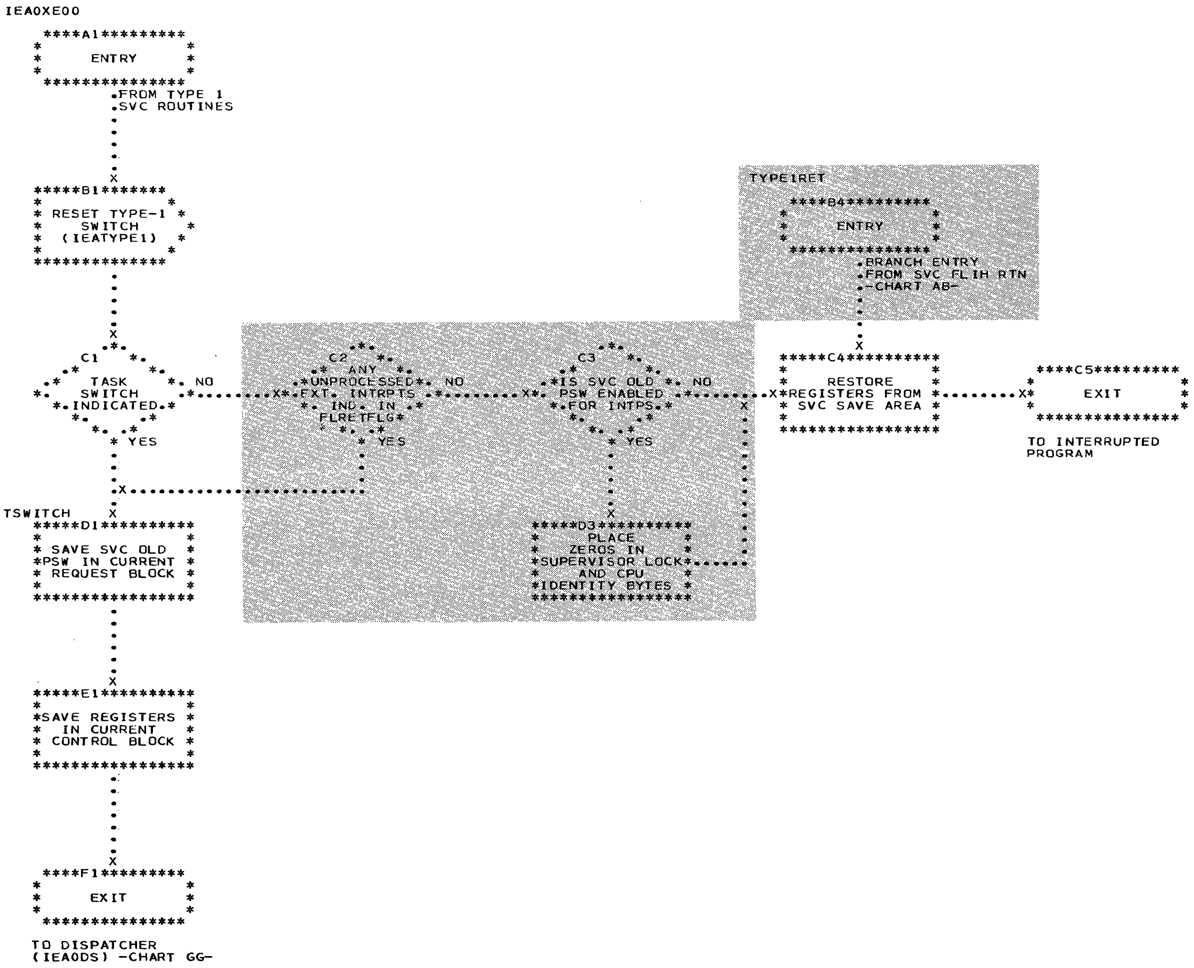

IGCOR05B
A1 ENTRY
FROM SYSIN/SYSOUT DATA SET PROC.2 -CHART FR-
B1
B1 RETURN FROM A PARM RLSE
NO
YES
B2 GET FIRST TABLE ENTRY
C1 TURN PARM. RLSE SWITCH OFF-RESTORE FULL DISC ADDRESS
D1 I/O ERROR
YES
NO
D2 DEQUEUE
ERROR CODE
E2 EXIT
TO RESTART EXIT (IGCOV05B) -CHART FU-
BLDCCWS
E1 BUILD CCW
CONSTRUCT CCW LIST
REPDCB20
F1 EXCP/WAIT
WRITE DSCB
G1 DEQUEUE
B4
REPDCB02
A3 IS DATA SET D.A. OR IND. SEQ.
YES
NO
REPDCB01
A4 GET NEXT TABLE ENTRY
B3 PART. ORG. CONCATEN-ATION
YES
NO
REPDCB04
B4 LAST ENTRY
YES
NO
B5 EXIT
TO ACCESS METHOD-DISPOSITION (IGCOT05B) -CHART FU-
B4
C3 IS DATA SET ON A DA DEVICE
NO
YES
D3 CLEAR STOR AND CONSTR 464 BYTE WK. AR. AND CONT. BLOCK
BLDCCWS
E3 BUILD CCW
CONSTRUCT CCW LIST
REPDCB20
F3 EXCP/WAIT
READ DSCB
REPDCB24
G3 CORRECT FILE TYPE FROM DD IF NECESSARY
H3 DSCB EXTENT TO BE READ
YES
NO
REPDCB20
H4 EXCP/WAIT
READ DSCB
REPDC60
REPDCB76
J2 DEB UPDATE
OVERLAY DEB CCHH WITH DSCB CCHH
B4
=
REPDCB25
J3 DSCB EXTENTS: DEB EXTENTS
LT
GT
REPDCB76
J4 DEB UPDATE
OVERLAY DEB CCHH WITH DSCB CCHH
J5 COMPRESS AND REFORMAT DEB. UPDATE LENGTH OF DEB
B4
ERROR CODE
K2 EXIT
TO RESTART EXIT (IGCOV05B) -CHART FU-
K3 DATA SET OPEN FOR INPUT
YES
NO
K4 PARTIAL RELEASE
B1

• Chart FU. Access Method-Disposition and Restart Exit Routines


ACCESS METHOD-DISPOSITION
RESTART EXIT
IGC0T05B
A1 ENTRY
FROM DATA SET PROCESSOR 1 OR 2 -CHART FS OR FT-
A2 INITIALIZE TABLE POINTERS
REPDCB02
B2 PARTITIONED DATA SET
YES
NO
B3 OPENED FOR INPUT ONLY
YES
NO
C2
C3
C1 INCREMENT TABLE POINTER
C2 ARE THERE MORE ENTRIES
YES
NO
C3 READ ONE DIRECTORY AND WAIT
D2 GET DEB FROM TCB
D3 ANY ERRORS
YES
NO
E3
D4 AT END OF DIRECTORY
YES
NO
C2
ERROR CODE
E4 EXIT
TO RESTART EXIT (IGC0V05B) -CHART FU-
REPDCB26
E1 RESTORE I/O
E2 WAS I/O PURGED
YES
NO
E3 TTR IN DCB GT TTR IN BLOCK
YES
NO
REPDCB28
F2 INCREMENT POINTER TO NEXT DEB
F3 ISSUE STOW TO DELETE MEMBER FROM DIRECTORY
G1 EXIT
TO RESTART EXIT RTN (IGC0V05B) -CHART FU-
G2 ARE THERE MORE DEB'S
NO
YES
G3 ERROR DURING STOW
YES
NO
H3 MORE TTR ENTRIES
YES
NO
E3
C3
IGC0V05B
A5 ENTRY
B4 RERWTO WRITE RESTART UNSUCCESSFUL MSG. TO CONSOLE
B5 ANY ERRORS DURING RESTART
YES
NO
C4 EXIT
TO ABEND RTN -CHART HI-
C5 RERWTO WRITE RESTART SUCCESSFUL MSG TO CONSOLE
IGC005(S)
D5 FREEMAIN DBA1 FREE RESTART WORK AREA SUBPOOL = 250
E5 BLKSIZE FOR CHKPT DCB SUPPLIED BY CHKPT
NO
YES
F5 RESET BLKSIZE IN CHKPT DCB TO ZERO
G5 DID CHKPT OPEN CHKPT DATA SET
NO
YES
H5 CLOSE CHKPT DATA SET
J5 SET RETURN CODE
K5 EXIT
TO SVC3 EXIT RTN -CHART GB-

• Chart GA. Type-1 Exit Routine

NOTE – SHADED AREA APPLIES ONLY TO MULTIPROCESSING SYSTEMS


IEA0XE00
A1
ENTRY
FROM TYPE 1 SVC ROUTINES
B1
RESET TYPE-1 SWITCH (IEATYPE1)
C1
TASK SWITCH INDICATED
NO
YES
C2
ANY UNPROCESSED EXT. INTRPTS IND. IN FLRETFLG
NO
YES
C3
IS SVC OLD PSW ENABLED FOR INTPS.
NO
YES
TYPE1RET
B4
ENTRY
BRANCH ENTRY FROM SVC FLIH RTN -CHART AB-
C4
RESTORE REGISTERS FROM SVC SAVE AREA
C5
EXIT
TO INTERRUPTED PROGRAM
D3
PLACE ZEROS IN SUPERVISOR LOCK AND CPU IDENTITY BYTES
TSWITCH
D1
SAVE SVC OLD PSW IN CURRENT REQUEST BLOCK
E1
SAVE REGISTERS IN CURRENT CONTROL BLOCK
F1
EXIT
TO DISPATCHER (IEA0DS) -CHART GG-

• Chart GB. Exit Routine (Part 1 of 2)

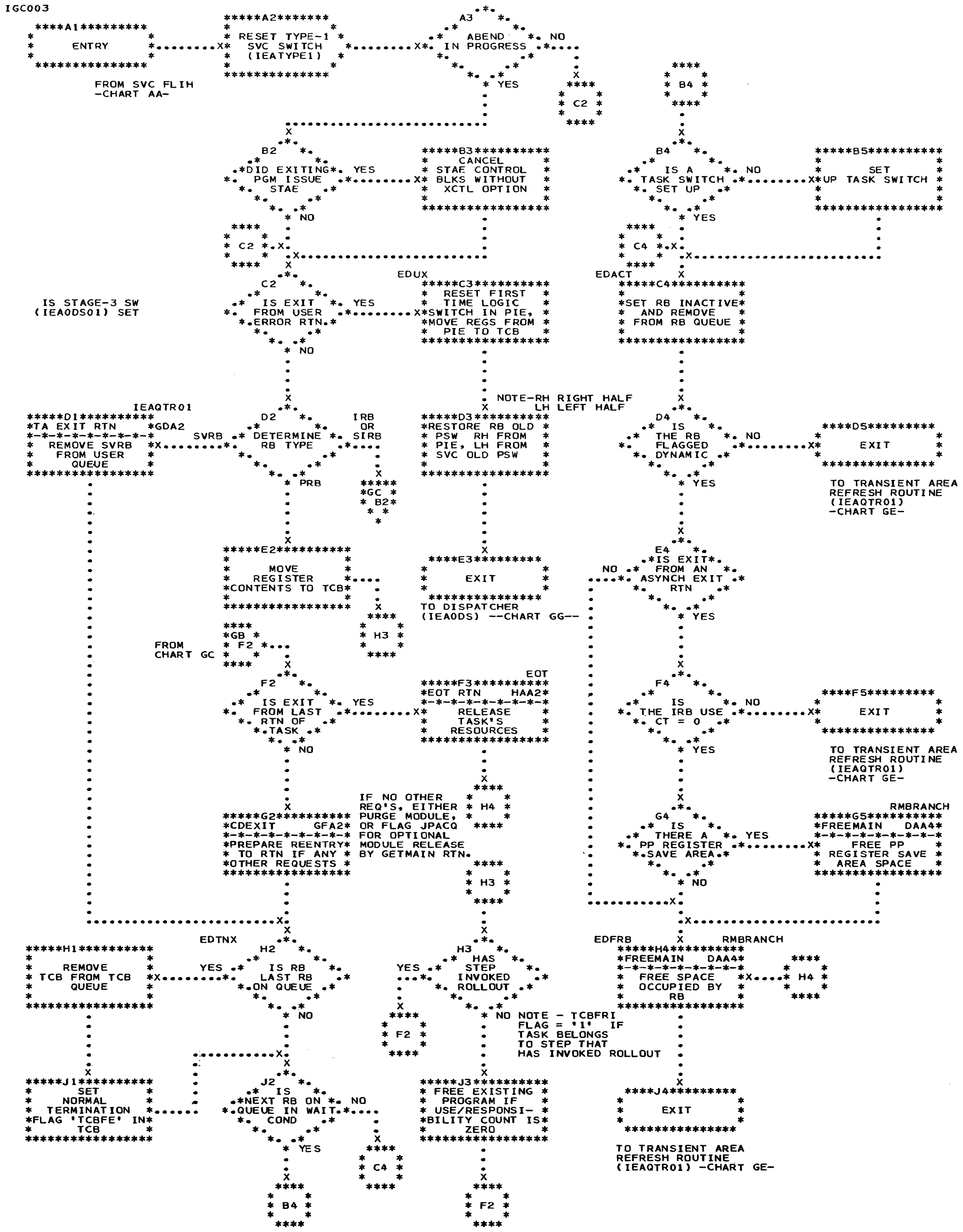

IGC003
A1
ENTRY
FROM SVC FLIH -CHART AA-
A2
RESET TYPE-1 SVC SWITCH (IEATYPE1)
A3
ABEND IN PROGRESS
NO
YES
C2
B4
B2
DID EXITING PGM ISSUE STAE
YES
NO
B3
CANCEL STAE CONTROL BLKS WITHOUT XCTL OPTION
B4
IS A TASK SWITCH SET UP
NO
YES
B5
SET UP TASK SWITCH
C2
C4
EDUX
EDACT
IS STAGE-3 SW (IEAODSO1) SET
C2
IS EXIT FROM USER ERROR RTN.
YES
NO
C3
RESET FIRST TIME LOGIC SWITCH IN PIE. MOVE REGS FROM PIE TO TCB
C4
SET RB INACTIVE AND REMOVE FROM RB QUEUE
NOTE-RH RIGHT HALF
LH LEFT HALF
IEAQTR01
D1
TA EXIT RTN GDA2
REMOVE SVRB FROM USER QUEUE
SVRB
D2
DETERMINE RB TYPE
IRB OR SIRB
PRB
GC B2
D3
RESTORE RB OLD PSW RH FROM PIE, LH FROM SVC OLD PSW
D4
IS THE RB FLAGGED DYNAMIC
NO
YES
D5
EXIT
TO TRANSIENT AREA REFRESH ROUTINE (IEAQTR01) -CHART GE-
E2
MOVE REGISTER CONTENTS TO TCB
H3
E3
EXIT
TO DISPATCHER (IEAODS) --CHART GG--
E4
IS EXIT FROM AN ASYNCH EXIT RTN
NO
YES
GB F2
FROM CHART GC
EOT
F2
IS EXIT FROM LAST RTN OF TASK
YES
NO
F3
EOT RTN HAA2
RELEASE TASK'S RESOURCES
F4
IS THE IRB USE CT = 0
NO
YES
F5
EXIT
TO TRANSIENT AREA REFRESH ROUTINE (IEAQTR01) -CHART GE-
H4
IF NO OTHER REQ'S, EITHER PURGE MODULE, OR FLAG JPACQ FOR OPTIONAL MODULE RELEASE BY GETMAIN RTN.
G2
CDEXIT GFA2
PREPARE REENTRY TO RTN IF ANY OTHER REQUESTS
H3
RMBRANCH
G4
IS THERE A PP REGISTER SAVE AREA
YES
NO
G5
FREEMAIN DAA4
FREE PP REGISTER SAVE AREA SPACE
EDTNX
H1
REMOVE TCB FROM TCB QUEUE
H2
IS RB LAST RB ON QUEUE
YES
NO
H3
HAS STEP INVOKED ROLLOUT
YES
NO
F2
NOTE - TCBFRI FLAG = '1' IF TASK BELONGS TO STEP THAT HAS INVOKED ROLLOUT
EDFRB
RMBRANCH
H4
FREEMAIN DAA4
FREE SPACE OCCUPIED BY RB
H4
J1
SET NORMAL TERMINATION FLAG 'TCBFE' IN TCB
J2
IS NEXT RB ON QUEUE IN WAIT COND
NO
YES
C4
B4
J3
FREE EXISTING PROGRAM IF USE/RESPONSIBILITY COUNT IS ZERO
F2
J4
EXIT
TO TRANSIENT AREA REFRESH ROUTINE (IEAQTR01) -CHART GE-

Chart GC. Exit Routine (Part 2 of 2)

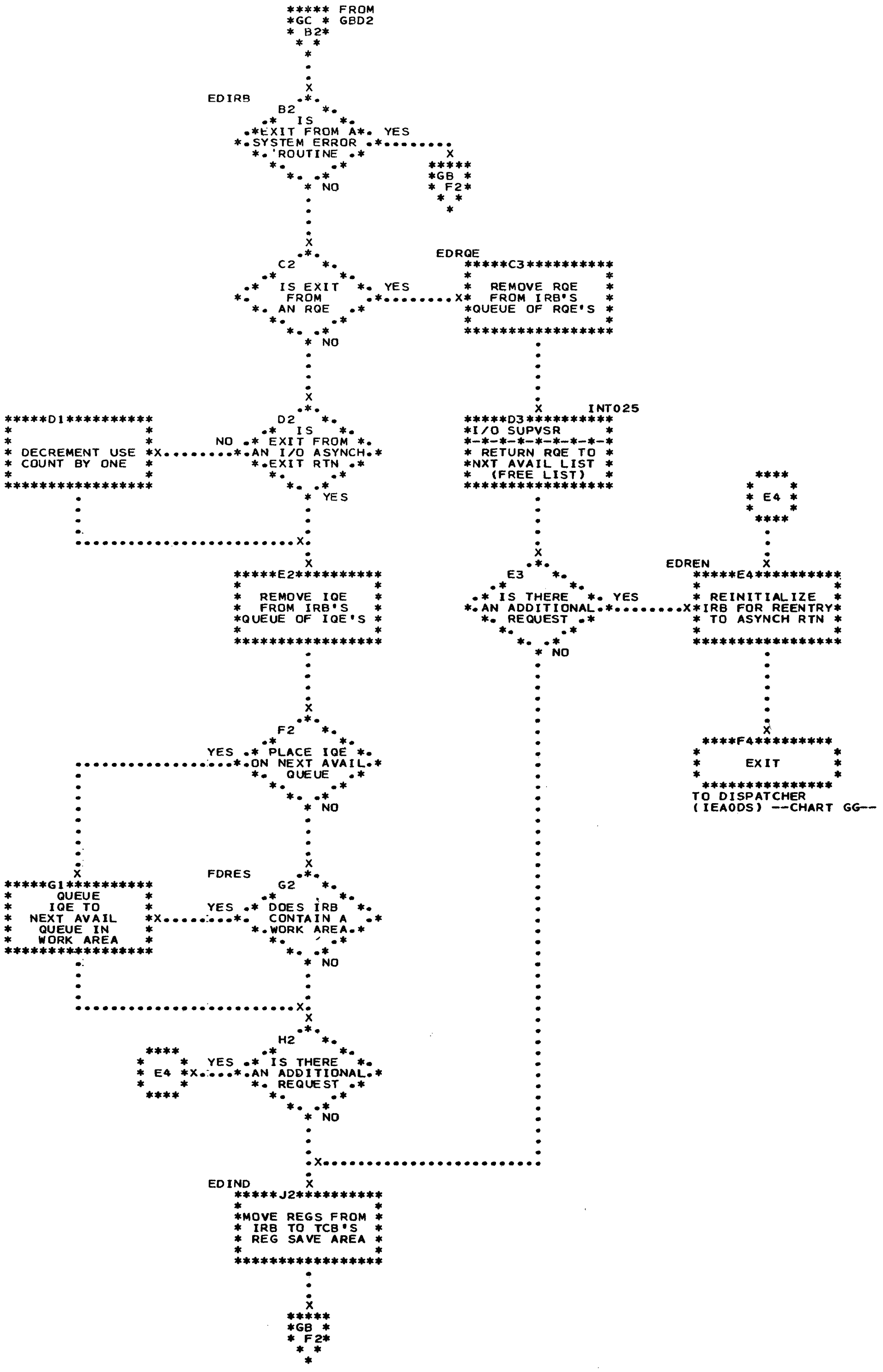
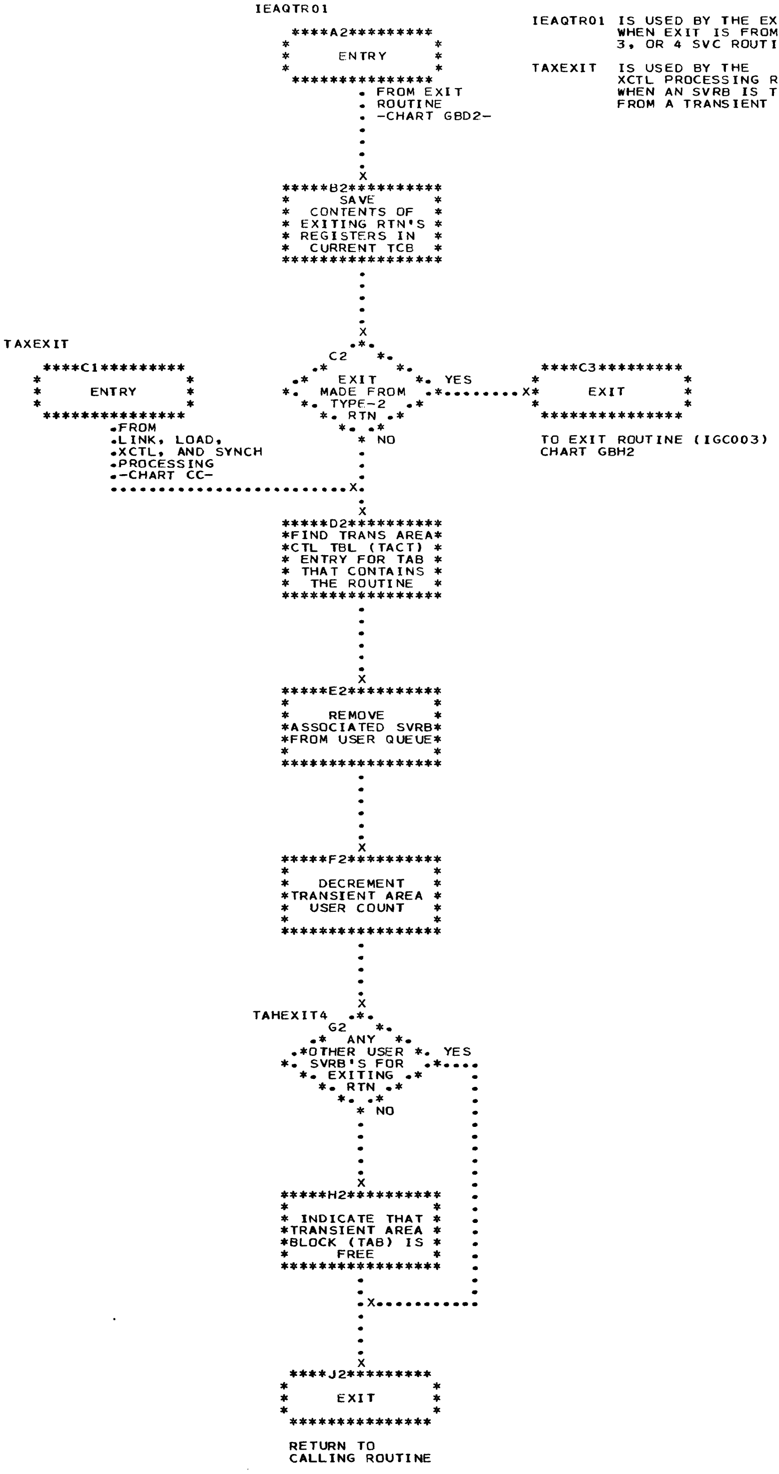

FROM GBD2
GC B2
EDIRB
B2
IS EXIT FROM A SYSTEM ERROR ROUTINE
YES
GB F2
NO
C2
IS EXIT FROM AN RQE
YES
EDRQE
C3
REMOVE RQE FROM IRB'S QUEUE OF RQE'S
NO
D1
DECREMENT USE COUNT BY ONE
D2
IS EXIT FROM AN I/O ASYNCH EXIT RTN
NO
YES
INT025
D3
I/O SUPVSR
RETURN RQE TO NXT AVAIL LIST (FREE LIST)
E4
E2
REMOVE IQE FROM IRB'S QUEUE OF IQE'S
E3
IS THERE AN ADDITIONAL REQUEST
YES
NO
EDREN
E4
REINITIALIZE IRB FOR REENTRY TO ASYNCH RTN
F2
PLACE IQE ON NEXT AVAIL QUEUE
YES
NO
F4
EXIT
TO DISPATCHER (IEAODS) --CHART GG--
G1
QUEUE IQE TO NEXT AVAIL QUEUE IN WORK AREA
FDRES
G2
DOES IRB CONTAIN A WORK AREA
YES
NO
H2
IS THERE AN ADDITIONAL REQUEST
YES
E4
NO
EDIND
J2
MOVE REGS FROM IRB TO TCB'S REG SAVE AREA
GB F2

Chart GD. Transient Area Exit Routine


IEAQTR01
A2
ENTRY
FROM EXIT ROUTINE -CHART GBD2-
B2
SAVE CONTENTS OF EXITING RTN'S REGISTERS IN CURRENT TCB
TAXEXIT
C1
ENTRY
FROM LINK, LOAD, XCTL, AND SYNCH PROCESSING -CHART CC-
C2
EXIT MADE FROM TYPE-2 RTN
YES
NO
C3
EXIT
TO EXIT ROUTINE (IGC003) CHART GBH2
D2
FIND TRANS AREA CTL TBL (TACT) ENTRY FOR TAB THAT CONTAINS THE ROUTINE
E2
REMOVE ASSOCIATED SVRB FROM USER QUEUE
F2
DECREMENT TRANSIENT AREA USER COUNT
TAHEXIT4
G2
ANY OTHER USER SVRB'S FOR EXITING RTN
YES
NO
H2
INDICATE THAT TRANSIENT AREA BLOCK (TAB) IS FREE
J2
EXIT
RETURN TO CALLING ROUTINE

IEAQTR01 IS USED BY THE EXIT ROUTINE WHEN EXIT IS FROM A TYPE 2, 3, OR 4 SVC ROUTINE.

TAXEXIT IS USED BY THE XCTL PROCESSING ROUTINE WHEN AN SVRB IS TO BE REMOVED FROM A TRANSIENT AREA QUEUE.

Chart GE. Transient Area Refresh Routine

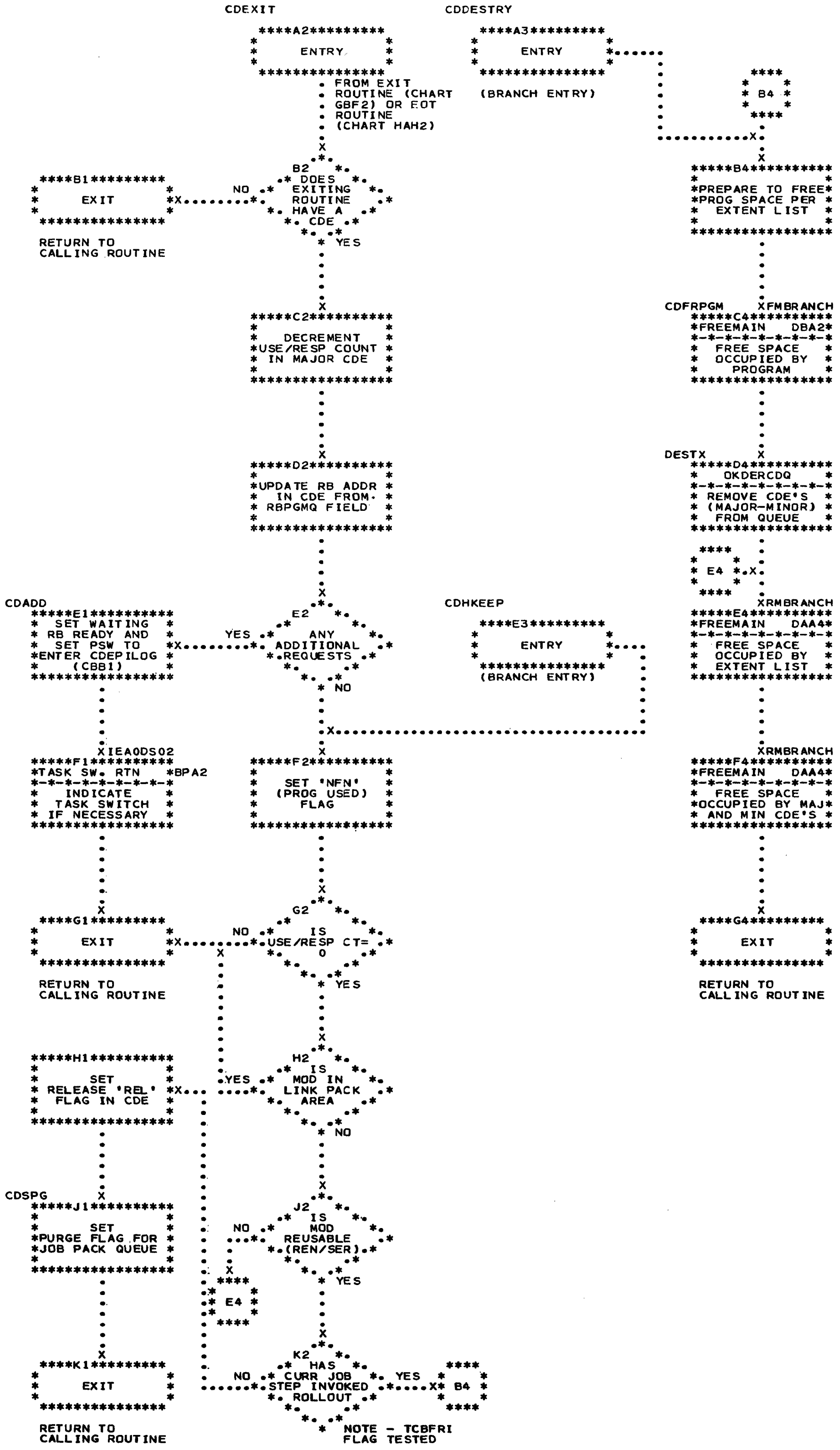

CDEXIT
A2 ENTRY
FROM EXIT ROUTINE (CHART GBF2) OR EOT ROUTINE (CHART HAH2)
CDDESTRY
A3 ENTRY
(BRANCH ENTRY)
B4
B1 EXIT
RETURN TO CALLING ROUTINE
B2 DOES EXITING ROUTINE HAVE A CDE
NO
YES
B4 PREPARE TO FREE PROG SPACE PER EXTENT LIST
C2 DECREMENT USE/RESP COUNT IN MAJOR CDE
CDFRPGM
XFMBRANCH
C4 FREEMAIN DBA2
FREE SPACE OCCUPIED BY PROGRAM
D2 UPDATE RB ADDR IN CDE FROM RBPGMQ FIELD
DESTX
D4 OKDERCDQ
REMOVE CDE'S (MAJOR-MINOR) FROM QUEUE
E4
CDADD
E1 SET WAITING RB READY AND SET PSW TO ENTER CDEPILOG (CBB1)
E2 ANY ADDITIONAL REQUESTS
YES
NO
CDHKEEP
E3 ENTRY
(BRANCH ENTRY)
XRMBRANCH
E4 FREEMAIN DAA4
FREE SPACE OCCUPIED BY EXTENT LIST
XIEAODS02
F1 TASK SW. RTN BPA2
INDICATE TASK SWITCH IF NECESSARY
F2 SET 'NFN' (PROG USED) FLAG
XRMBRANCH
F4 FREEMAIN DAA4
FREE SPACE OCCUPIED BY MAJ AND MIN CDE'S
G1 EXIT
RETURN TO CALLING ROUTINE
G2 IS USE/RESP CT= 0
NO
YES
G4 EXIT
RETURN TO CALLING ROUTINE
H1 SET RELEASE 'REL' FLAG IN CDE
H2 IS MOD IN LINK PACK AREA
YES
NO
CDSPG
J1 SET PURGE FLAG FOR JOB PACK QUEUE
J2 IS MOD REUSABLE (REN/SER)
NO
YES
E4
K1 EXIT
RETURN TO CALLING ROUTINE
K2 HAS CURR JOB STEP INVOKED ROLLOUT
NO
YES
B4
NOTE - TCBFRI FLAG TESTED

Chart GF. CDEXIT Routine

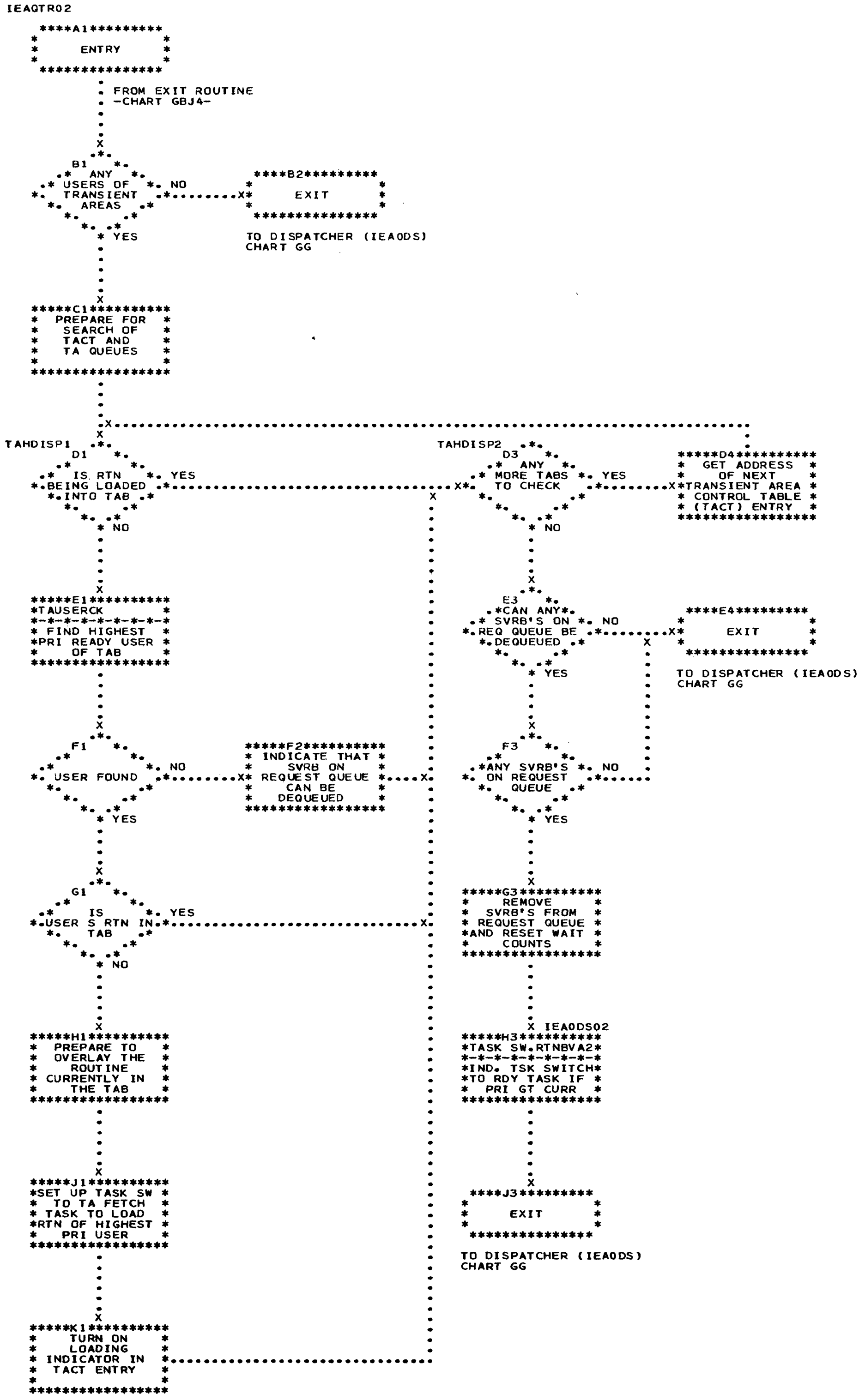

IEAQTR02
A1
ENTRY
FROM EXIT ROUTINE -CHART GBJ4-
B1
ANY USERS OF TRANSIENT AREAS
NO
YES
B2
EXIT
TO DISPATCHER (IEAODS) CHART GG
C1
PREPARE FOR SEARCH OF TACT AND TA QUEUES
TAHDISP1
D1
IS RTN BEING LOADED INTO TAB
YES
NO
TAHDISP2
D3
ANY MORE TABS TO CHECK
YES
NO
D4
GET ADDRESS OF NEXT TRANSIENT AREA CONTROL TABLE (TACT) ENTRY
E1
TAUSERCK
FIND HIGHEST PRI READY USER OF TAB
E3
CAN ANY SVRB'S ON REQ QUEUE BE DEQUEUED
NO
YES
E4
EXIT
TO DISPATCHER (IEAODS) CHART GG
F1
USER FOUND
NO
YES
F2
INDICATE THAT SVRB ON REQUEST QUEUE CAN BE DEQUEUED
F3
ANY SVRB'S ON REQUEST QUEUE
NO
YES
G1
IS USER S RTN IN TAB
YES
NO
G3
REMOVE SVRB'S FROM REQUEST QUEUE AND RESET WAIT COUNTS
IEAODS02
H3
TASK SW.RTNBVA2
IND. TSK SWITCH TO RDY TASK IF PRI GT CURR
H1
PREPARE TO OVERLAY THE ROUTINE CURRENTLY IN THE TAB
J1
SET UP TASK SW TO TA FETCH TASK TO LOAD RTN OF HIGHEST PRI USER
J3
EXIT
TO DISPATCHER (IEAODS) CHART GG
K1
TURN ON LOADING INDICATOR IN TACT ENTRY

Chart GG. Dispatcher (Uniprocessing System)

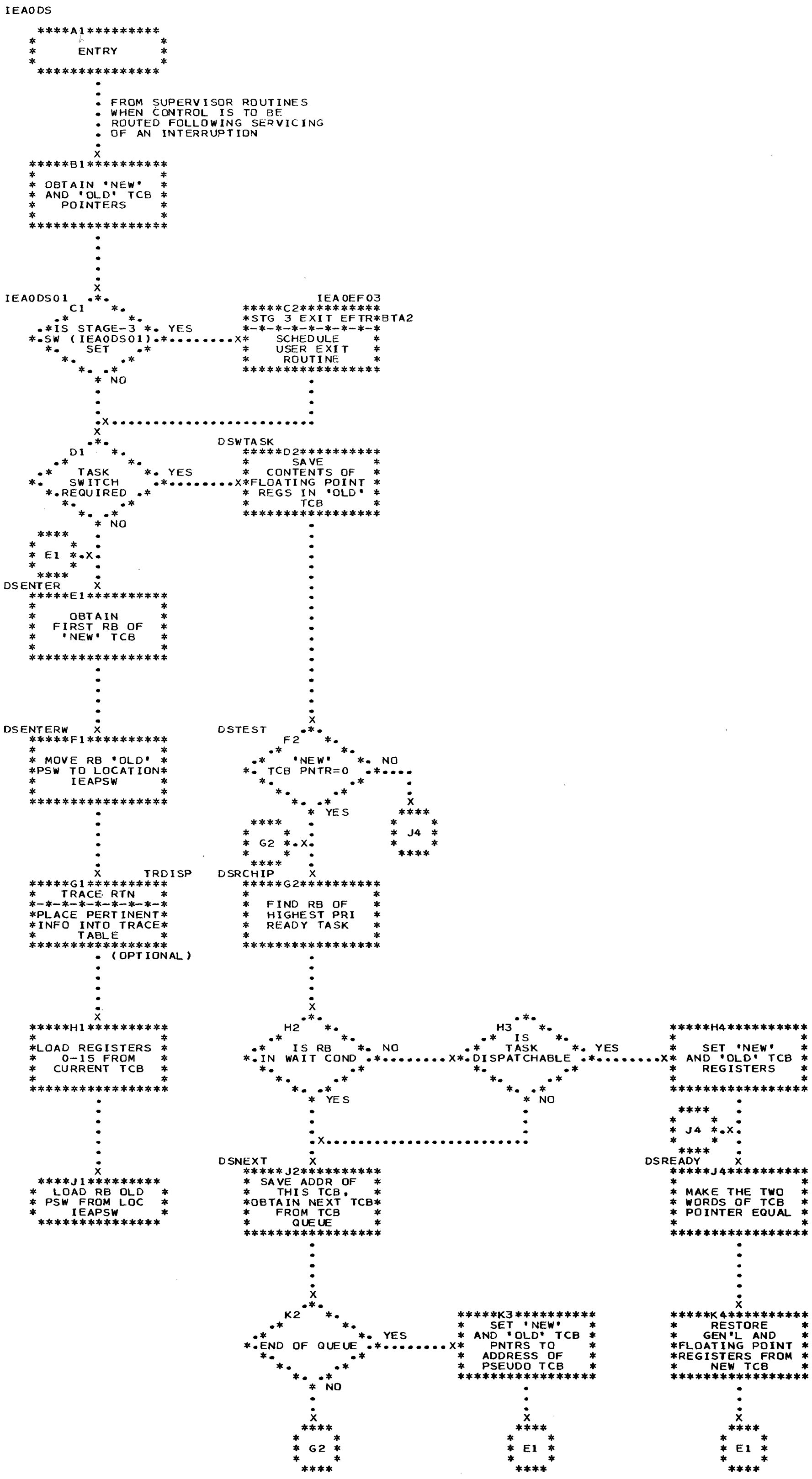

IEAODS
A1
ENTRY
FROM SUPERVISOR ROUTINES WHEN CONTROL IS TO BE ROUTED FOLLOWING SERVICING OF AN INTERRUPTION
B1
OBTAIN 'NEW' AND 'OLD' TCB POINTERS
IEAODS01
C1
IS STAGE-3 SW (IEAODS01) SET
YES
NO
IEAOEF03
C2
STG 3 EXIT EFTR*BTA2
SCHEDULE USER EXIT ROUTINE
D1
TASK SWITCH REQUIRED
YES
NO
DSWTASK
D2
SAVE CONTENTS OF FLOATING POINT REGS IN 'OLD' TCB
E1
DSENTER
E1
OBTAIN FIRST RB OF 'NEW' TCB
DSENTERW
F1
MOVE RB 'OLD' PSW TO LOCATION IEAPSW
DSTEST
F2
'NEW' TCB PNTR=0
NO
YES
J4
G2
TRDISP
G1
TRACE RTN
PLACE PERTINENT INFO INTO TRACE TABLE
(OPTIONAL)
DSRCHIP
G2
FIND RB OF HIGHEST PRI READY TASK
H1
LOAD REGISTERS 0-15 FROM CURRENT TCB
H2
IS RB IN WAIT COND
NO
YES
H3
IS TASK DISPATCHABLE
YES
NO
H4
SET 'NEW' AND 'OLD' TCB REGISTERS
J4
J1
LOAD RB OLD PSW FROM LOC IEAPSW
DSNEXT
J2
SAVE ADDR OF THIS TCB, OBTAIN NEXT TCB FROM TCB QUEUE
DSREADY
J4
MAKE THE TWO WORDS OF TCB POINTER EQUAL
K2
END OF QUEUE
YES
NO
K3
SET 'NEW' AND 'OLD' TCB PNTRS TO ADDRESS OF PSEUDO TCB
K4
RESTORE GEN'L AND FLOATING POINT REGISTERS FROM NEW TCB
G2
E1
E1

Chart GH. Dispatcher (Uniprocessing System With Job Step and Task Timing; Part 1 of 2)

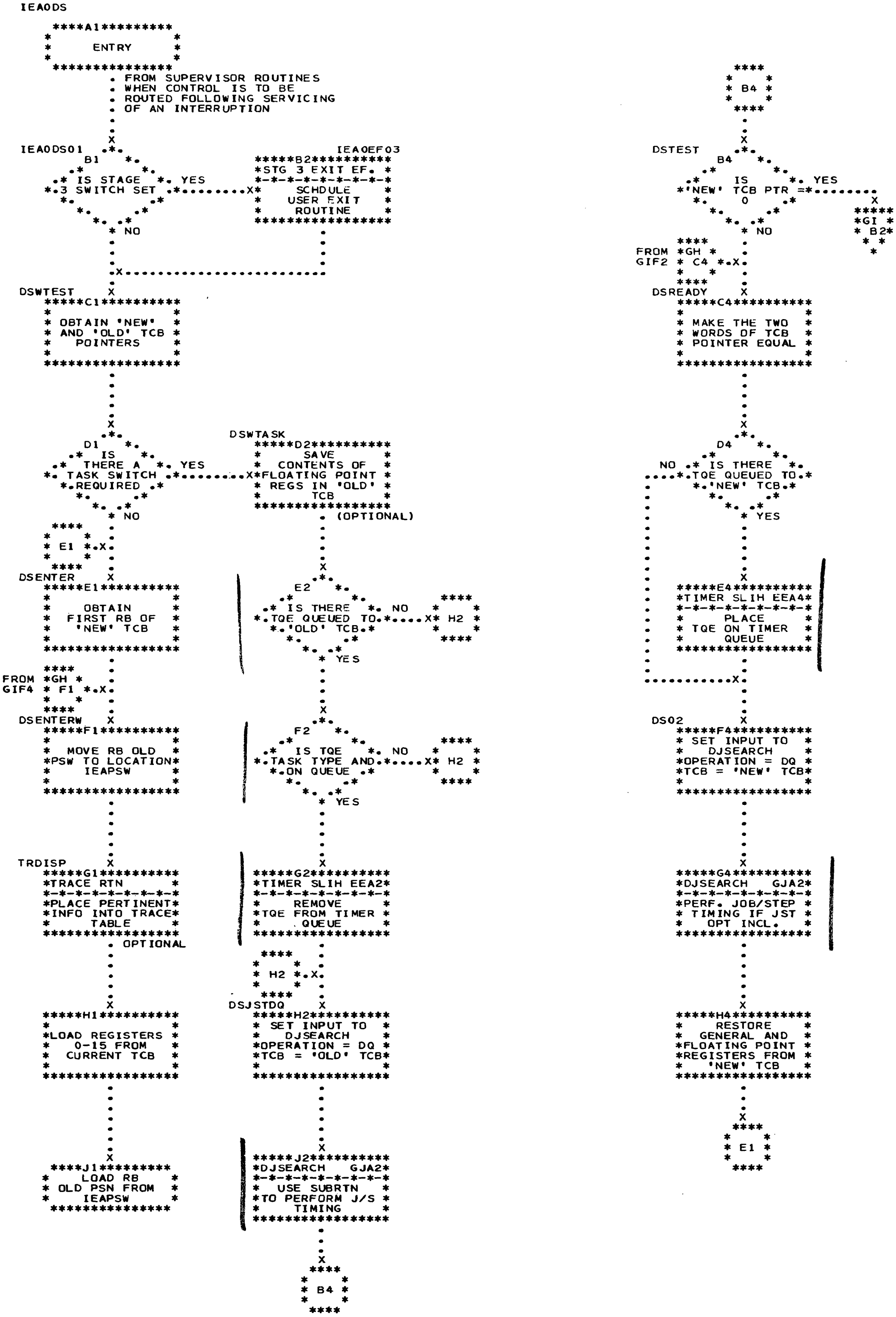

IEAODS
A1
ENTRY
FROM SUPERVISOR ROUTINES WHEN CONTROL IS TO BE ROUTED FOLLOWING SERVICING OF AN INTERRUPTION
IEAODSO1
B1
IS STAGE 3 SWITCH SET
YES
NO
IEAOEF03
B2
STG 3 EXIT EF.
SCHDULE USER EXIT ROUTINE
DSWTEST
C1
OBTAIN 'NEW' AND 'OLD' TCB POINTERS
D1
IS THERE A TASK SWITCH REQUIRED
YES
NO
DSWTASK
D2
SAVE CONTENTS OF FLOATING POINT REGS IN 'OLD' TCB
(OPTIONAL)
E1
DSENTER
E1
OBTAIN FIRST RB OF 'NEW' TCB
FROM GH GIF4
F1
DSENTERW
F1
MOVE RB OLD PSW TO LOCATION IEAPSW
TRDISP
G1
TRACE RTN
PLACE PERTINENT INFO INTO TRACE TABLE
OPTIONAL
H1
LOAD REGISTERS 0-15 FROM CURRENT TCB
J1
LOAD RB OLD PSN FROM IEAPSW
E2
IS THERE TQE QUEUED TO 'OLD' TCB
NO
H2
YES
F2
IS TQE TASK TYPE AND ON QUEUE
NO
H2
YES
G2
TIMER SLIH EEA2
REMOVE TQE FROM TIMER QUEUE
H2
DSJSTDQ
H2
SET INPUT TO DJSEARCH OPERATION = DQ TCB = 'OLD' TCB
J2
DJSEARCH GJA2
USE SUBRTN TO PERFORM J/S TIMING
B4
B4
DSTEST
B4
IS 'NEW' TCB PTR = 0
YES
GI B2
NO
FROM GH GIF2
C4
DSREADY
C4
MAKE THE TWO WORDS OF TCB POINTER EQUAL
D4
IS THERE TQE QUEUED TO 'NEW' TCB
NO
YES
E4
TIMER SLIH EEA4
PLACE TQE ON TIMER QUEUE
DS02
F4
SET INPUT TO DJSEARCH OPERATION = DQ TCB = 'NEW' TCB
G4
DJSEARCH GJA2
PERF. JOB/STEP TIMING IF JST OPT INCL.
H4
RESTORE GENERAL AND FLOATING POINT REGISTERS FROM 'NEW' TCB
E1

Chart GI. Dispatcher (Uniprocessing System With Job Step and Task Timing; Part 2 of 2)

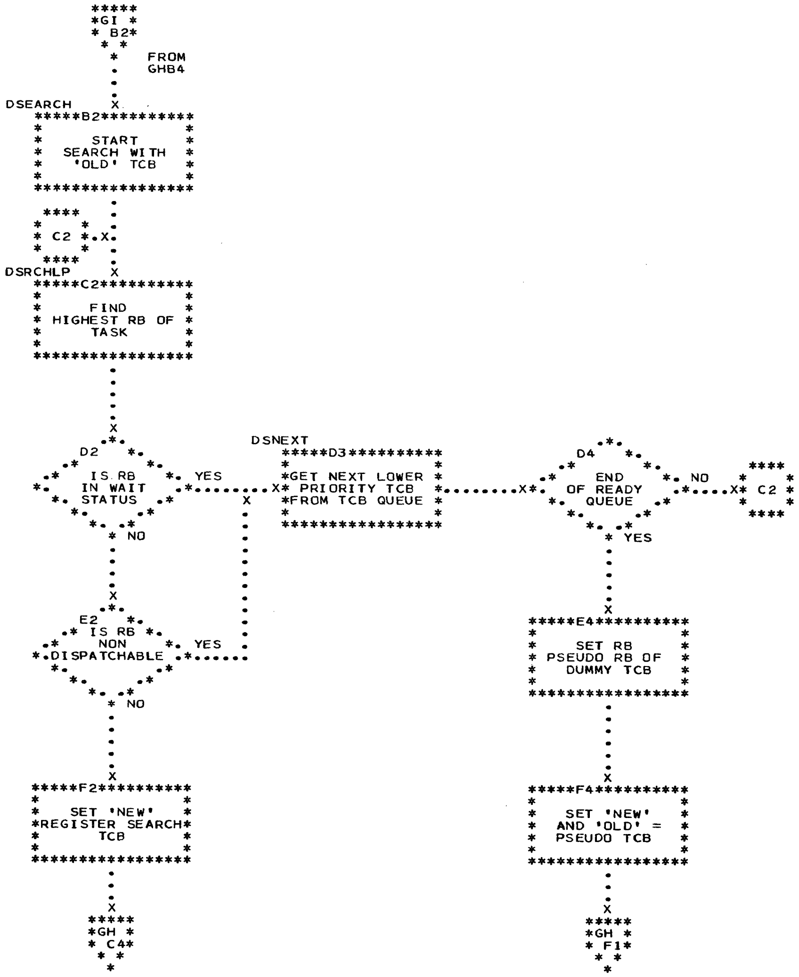


Chart GJ. DJSEARCH Subroutine (Uniprocessing System)

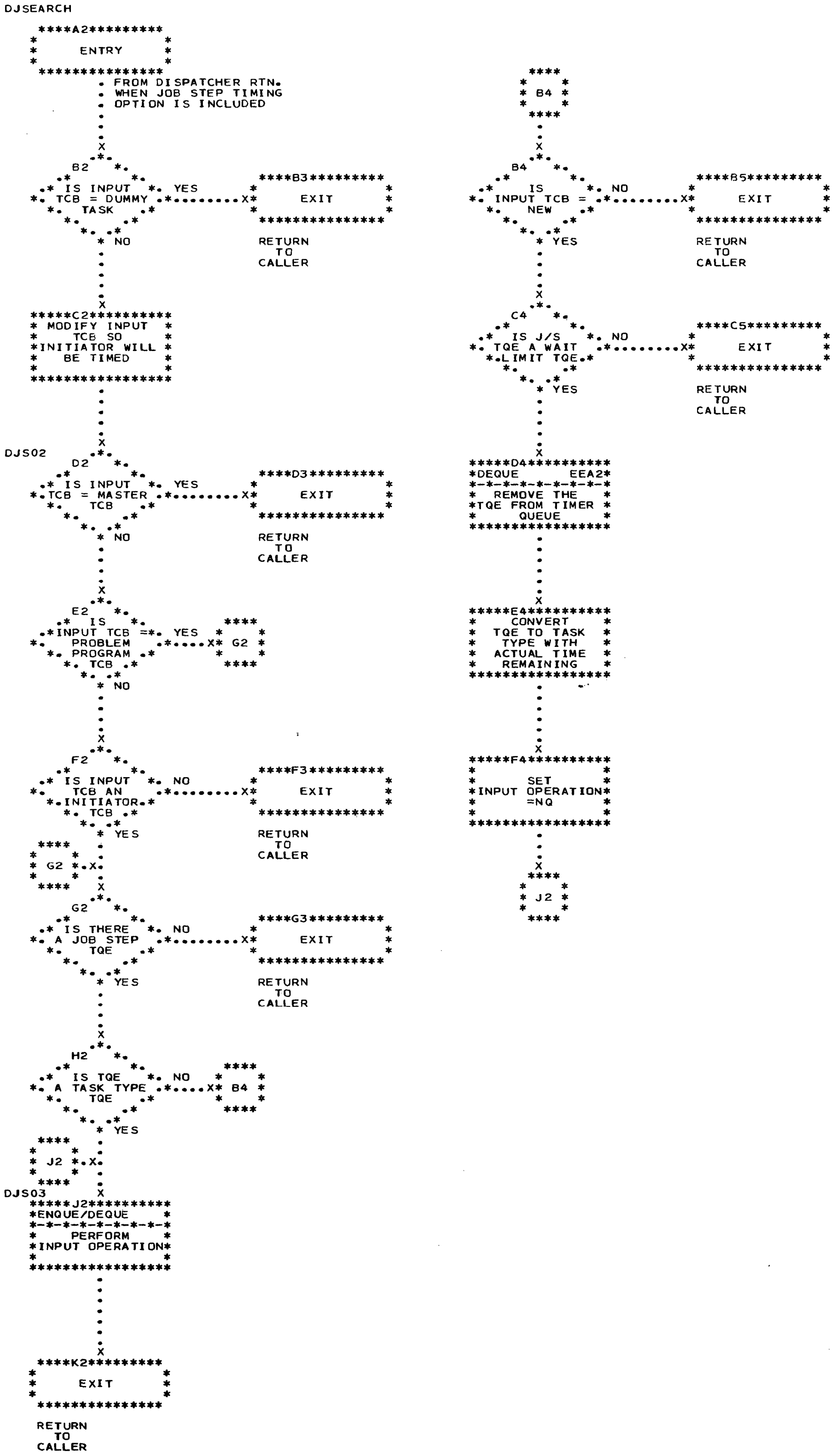
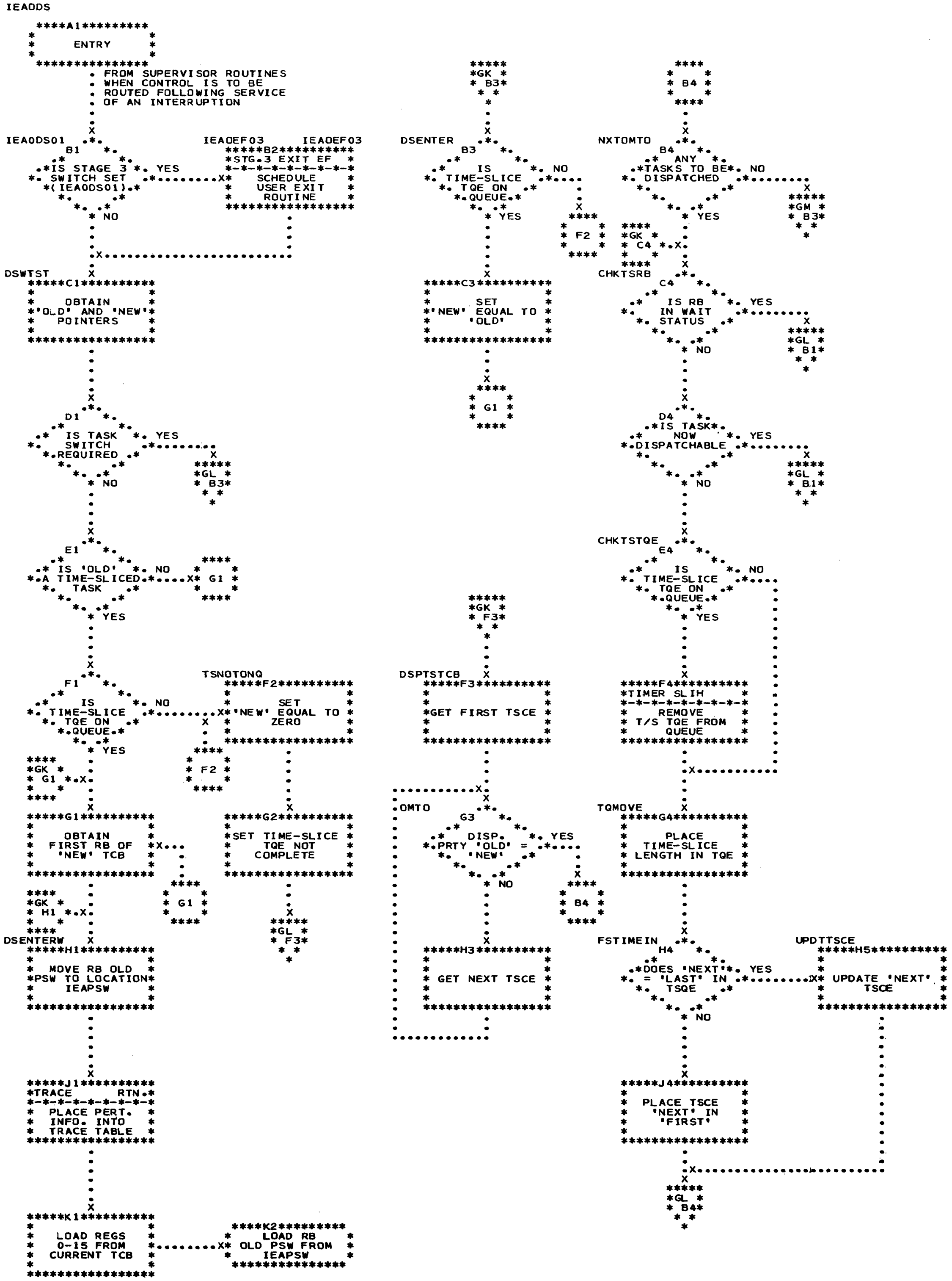


Chart GK. Dispatcher (Uniprocessing System With Time-Slicing; Part 1 of 3)



IEAODS
A1
ENTRY
FROM SUPERVISOR ROUTINES WHEN CONTROL IS TO BE ROUTED FOLLOWING SERVICE OF AN INTERRUPTION
IEAODS01
B1
IS STAGE 3 SWITCH SET (IEAODS01)
YES
NO
IEAOEF03
IEAOEF03
B2
STG.3 EXIT EF
SCHEDULE USER EXIT ROUTINE
DSWTST
C1
OBTAIN 'OLD' AND 'NEW' POINTERS
D1
IS TASK SWITCH REQUIRED
YES
NO
GL B3
E1
IS 'OLD' A TIME-SLICED TASK
NO
YES
G1
F1
IS TIME-SLICE TQE ON QUEUE
NO
YES
TSNOTONQ
F2
SET 'NEW' EQUAL TO ZERO
GK G1
F2
G1
OBTAIN FIRST RB OF 'NEW' TCB
G2
SET TIME-SLICE TQE NOT COMPLETE
G1
GL F3
GK H1
DSENTERW
H1
MOVE RB OLD PSW TO LOCATION IEAPSW
J1
TRACE RTN.
PLACE PERT. INFO. INTO TRACE TABLE
K1
LOAD REGS 0-15 FROM CURRENT TCB
K2
LOAD RB OLD PSW FROM IEAPSW
GK B3
DSENTER
B3
IS TIME-SLICE TQE ON QUEUE
NO
YES
F2
GK C4
C3
SET 'NEW' EQUAL TO 'OLD'
G1
GK F3
DSPTSTCB
F3
GET FIRST TSCE
OMTO
G3
DISP. PRTY 'OLD' = 'NEW'
YES
NO
B4
H3
GET NEXT TSCE
B4
NXTOMTO
B4
ANY TASKS TO BE DISPATCHED
NO
YES
GM B3
CHKTSRB
C4
IS RB IN WAIT STATUS
YES
NO
GL B1
D4
IS TASK NOW DISPATCHABLE
YES
NO
GL B1
CHKTSTQE
E4
IS TIME-SLICE TQE ON QUEUE
NO
YES
F4
TIMER SLIH
REMOVE T/S TQE FROM QUEUE
TQMOVE
G4
PLACE TIME-SLICE LENGTH IN TQE
FSTIMEIN
H4
DOES 'NEXT' = 'LAST' IN TSQE
YES
NO
UPDTTSCE
H5
UPDATE 'NEXT' TSCE
J4
PLACE TSCE 'NEXT' IN 'FIRST'
GL B4

Chart GL. Dispatcher (Uniprocessing System With Time-Slicing; Part 2 of 3)

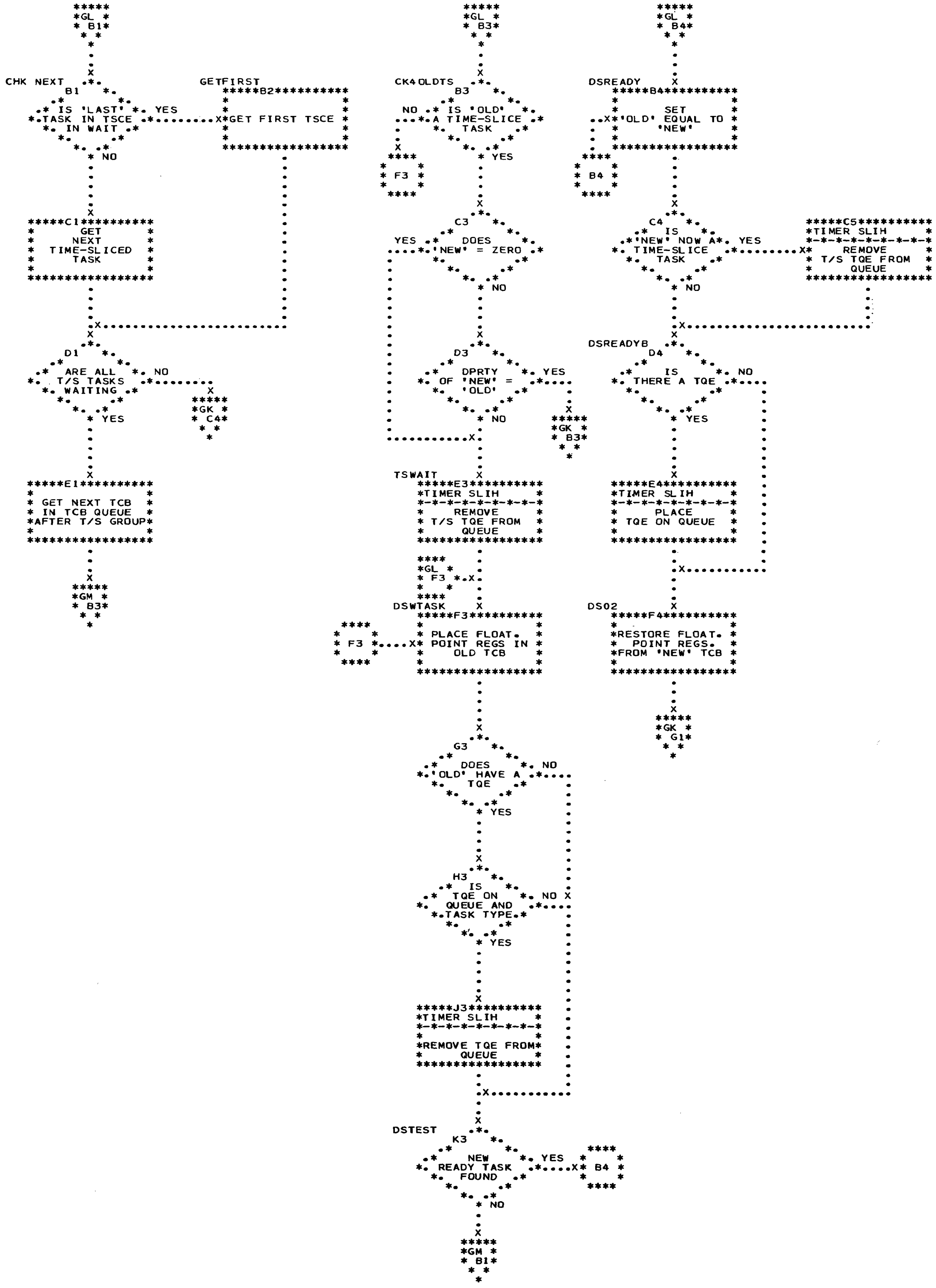
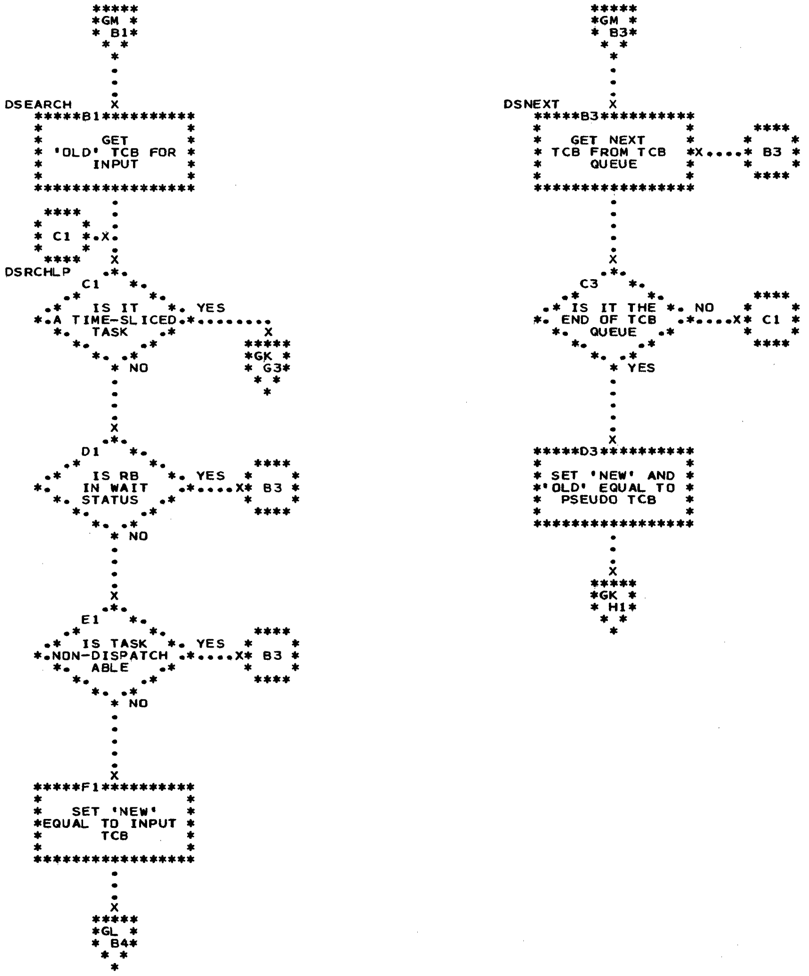
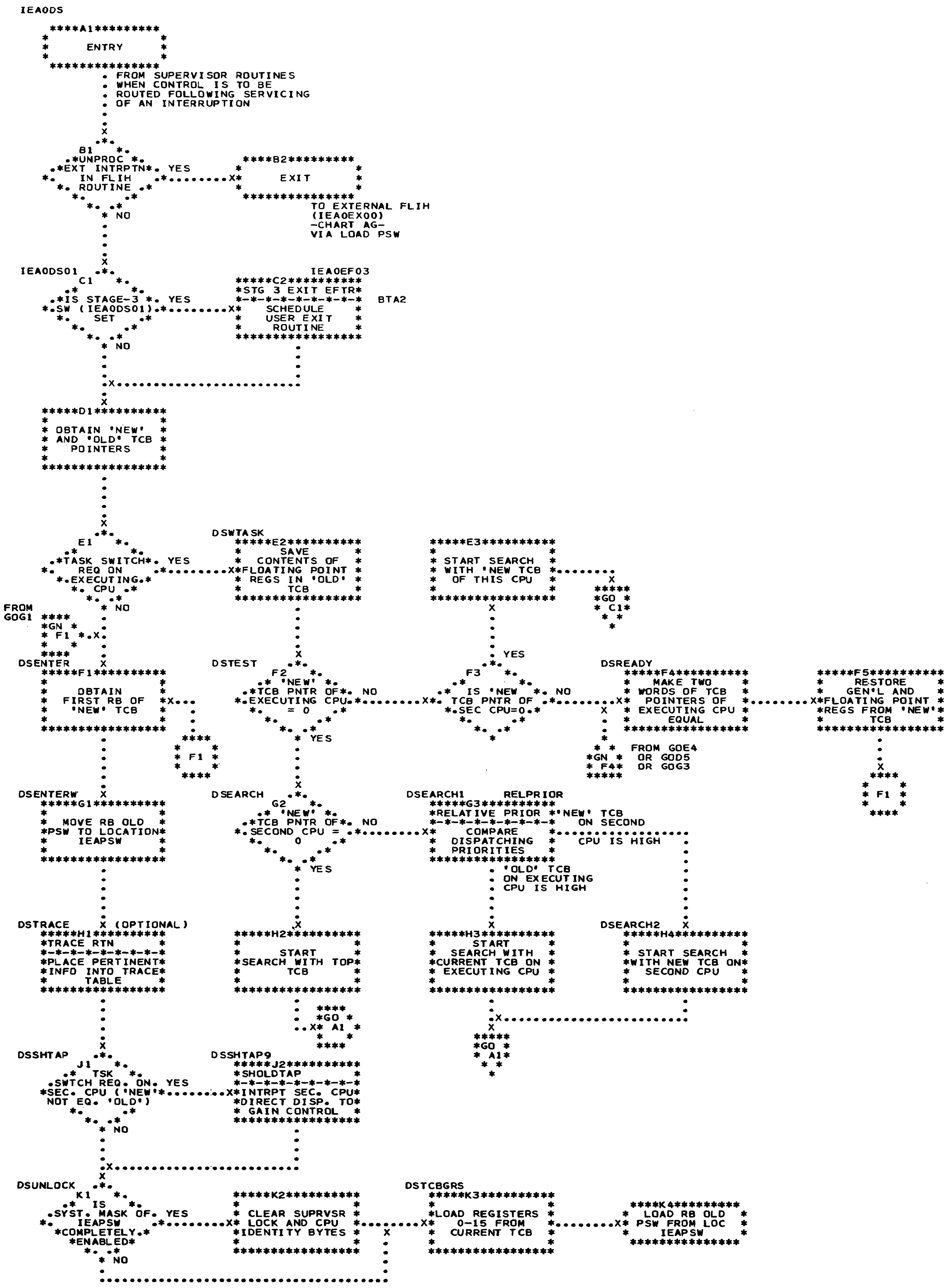


Chart GM. Dispatcher (Unprocessing System With Time-Slicing; Part 3 of 3)

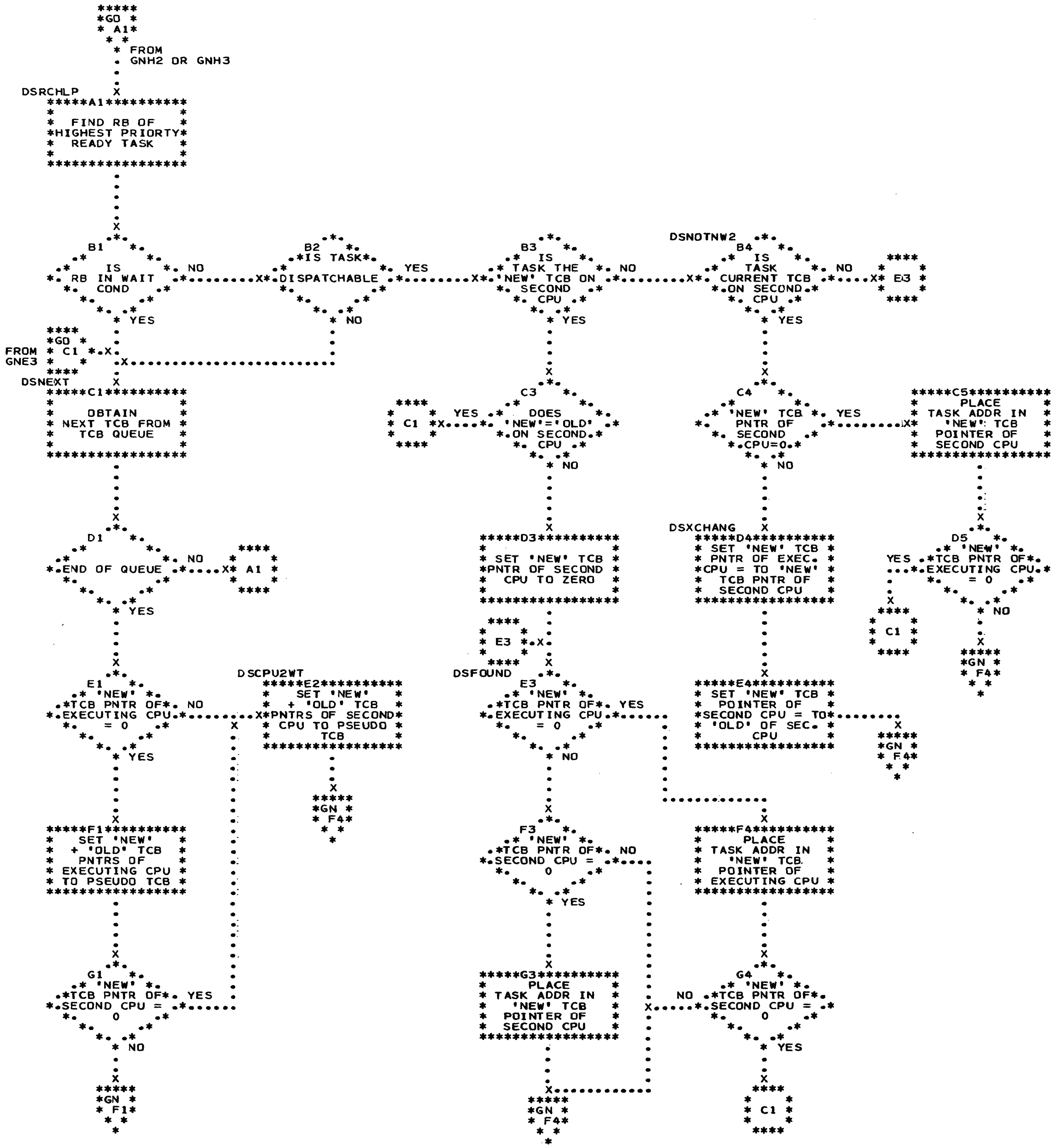


• Chart GN. Dispatcher (Multiprocessing System; Part 1 of 2)



IEA0DS
A1
ENTRY
FROM SUPERVISOR ROUTINES WHEN CONTROL IS TO BE ROUTED FOLLOWING SERVICING OF AN INTERRUPTION
B1
UNPROC EXT INTRPTN IN FLIH ROUTINE
YES
NO
B2
EXIT
TO EXTERNAL FLIH (IEA0EX00) -CHART AG- VIA LOAD PSW
IEA0DS01
C1
IS STAGE-3 SW (IEA0DS01) SET
YES
NO
IEA0EF03
C2
STG 3 EXIT EFTR
SCHEDULE USER EXIT ROUTINE
BTA2
D1
OBTAIN 'NEW' AND 'OLD' TCB POINTERS
E1
TASK SWITCH REQ ON EXECUTING CPU
YES
NO
DSWTASK
E2
SAVE CONTENTS OF FLOATING POINT REGS IN 'OLD' TCB
E3
START SEARCH WITH 'NEW TCB OF THIS CPU
GO C1
FROM GOG1
GN F1
DSENTER
F1
OBTAIN FIRST RB OF 'NEW' TCB
F1
DSTEST
F2
'NEW' TCB PNTR OF EXECUTING CPU = 0
NO
YES
F3
IS 'NEW TCB PNTR OF SEC CPU=0
YES
NO
DSREADY
F4
MAKE TWO WORDS OF TCB POINTERS OF EXECUTING CPU EQUAL
GN F4
FROM GOE4 OR GOD5 OR GOG3
F5
RESTORE GEN'L AND FLOATING POINT REGS FROM 'NEW' TCB
F1
DSENTERW
G1
MOVE RB OLD PSW TO LOCATION IEAPSW
DSEARCH
G2
'NEW' TCB PNTR OF SECOND CPU = 0
NO
YES
DSEARCH1
RELPRIOR
G3
RELATIVE PRIOR COMPARE DISPATCHING PRIORITIES
'NEW' TCB ON SECOND CPU IS HIGH
'OLD' TCB ON EXECUTING CPU IS HIGH
DSTRACE
(OPTIONAL)
H1
TRACE RTN
PLACE PERTINENT INFO INTO TRACE TABLE
H2
START SEARCH WITH TOP TCB
GO A1
H3
START SEARCH WITH CURRENT TCB ON EXECUTING CPU
DSEARCH2
H4
START SEARCH WITH NEW TCB ON SECOND CPU
GO A1
DSSHTAP
J1
TSK SWTCH REQ. ON SEC. CPU ('NEW' NOT EQ. 'OLD')
YES
NO
DSSHTAP9
J2
SHOLDTAP
INTRPT SEC. CPU DIRECT DISP. TO GAIN CONTROL
DSUNLOCK
K1
IS SYST. MASK OF IEAPSW COMPLETELY ENABLED
YES
NO
K2
CLEAR SUPRVSR LOCK AND CPU IDENTITY BYTES
DSTCBGRS
K3
LOAD REGISTERS 0-15 FROM CURRENT TCB
K4
LOAD RB OLD PSW FROM LOC IEAPSW

• Chart GO. Dispatcher (Multiprocessing System; Part 2 of 2)


GO A1
FROM GNH2 OR GNH3
DSRCHLP
A1
FIND RB OF HIGHEST PRIORTY READY TASK
B1
IS RB IN WAIT COND
NO
YES
B2
IS TASK DISPATCHABLE
YES
NO
B3
IS TASK THE 'NEW' TCB ON SECOND CPU
NO
YES
DSNOTNW2
B4
IS TASK CURRENT TCB ON SECOND CPU
NO
YES
E3
GO C1
FROM GNE3
DSNEXT
C1
OBTAIN NEXT TCB FROM TCB QUEUE
C1
YES
C3
DOES 'NEW'='OLD' ON SECOND CPU
NO
C4
'NEW' TCB PNTR OF SECOND CPU=0
YES
NO
C5
PLACE TASK ADDR IN 'NEW' TCB POINTER OF SECOND CPU
D1
END OF QUEUE
NO
A1
YES
D3
SET 'NEW' TCB PNTR OF SECOND CPU TO ZERO
DSXCHANG
D4
SET 'NEW' TCB PNTR OF EXEC. CPU = TO 'NEW' TCB PNTR OF SECOND CPU
D5
YES
'NEW' TCB PNTR OF EXECUTING CPU = 0
NO
C1
GN F4
E3
E1
'NEW' TCB PNTR OF EXECUTING CPU = 0
NO
YES
DSCPU2WT
E2
SET 'NEW' + 'OLD' TCB PNTRS OF SECOND CPU TO PSEUDO TCB
GN F4
DSFOUND
E3
'NEW' TCB PNTR OF EXECUTING CPU = 0
YES
NO
E4
SET 'NEW' TCB POINTER OF SECOND CPU = TO 'OLD' OF SEC. CPU
GN F4
F1
SET 'NEW' + 'OLD' TCB PNTRS OF EXECUTING CPU TO PSEUDO TCB
F3
'NEW' TCB PNTR OF SECOND CPU = 0
NO
YES
F4
PLACE TASK ADDR IN 'NEW' TCB POINTER OF EXECUTING CPU
G1
'NEW' TCB PNTR OF SECOND CPU = 0
YES
NO
GN F1
G3
PLACE TASK ADDR IN 'NEW' TCB POINTER OF SECOND CPU
GN F4
G4
NO
'NEW' TCB PNTR OF SECOND CPU = 0
YES
C1

• Chart GP. DJSEARCH Subroutine (Multiprocessing System)

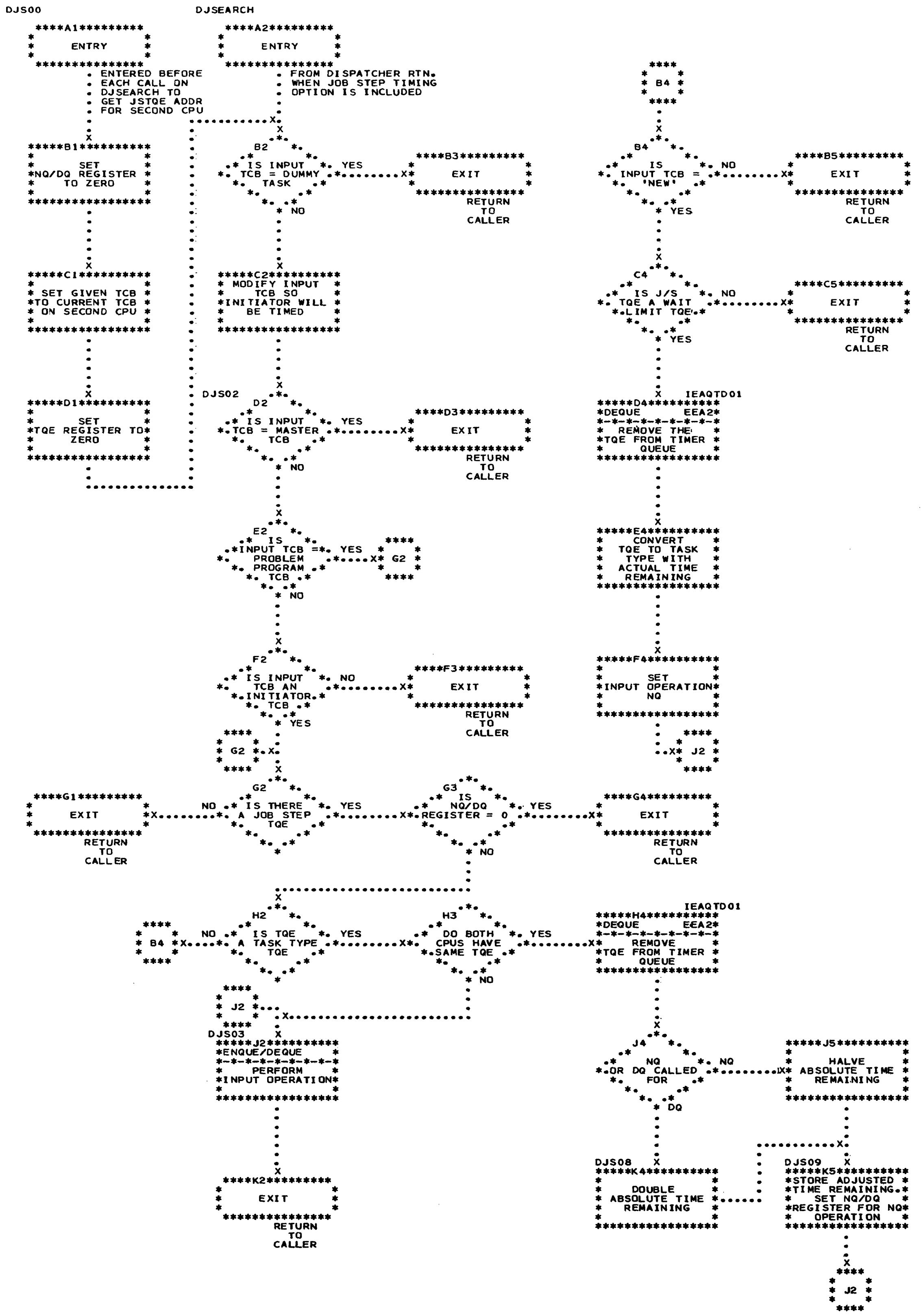

DJS00
A1 ENTRY
ENTERED BEFORE EACH CALL ON DJSEARCH TO GET JSTQE ADDR FOR SECOND CPU
DJSEARCH
A2 ENTRY
FROM DISPATCHER RTN. WHEN JOB STEP TIMING OPTION IS INCLUDED
B1 SET NQ/DQ REGISTER TO ZERO
B2 IS INPUT TCB = DUMMY TASK
YES
NO
B3 EXIT
RETURN TO CALLER
C1 SET GIVEN TCB TO CURRENT TCB ON SECOND CPU
C2 MODIFY INPUT TCB SO INITIATOR WILL BE TIMED
D1 SET TQE REGISTER TO ZERO
DJS02
D2 IS INPUT TCB = MASTER TCB
D3 EXIT
RETURN TO CALLER
E2 IS INPUT TCB = PROBLEM PROGRAM TCB
G2
F2 IS INPUT TCB AN INITIATOR TCB
F3 EXIT
RETURN TO CALLER
G1 EXIT
RETURN TO CALLER
G2 IS THERE A JOB STEP TQE
G3 IS NQ/DQ REGISTER = 0
G4 EXIT
RETURN TO CALLER
B4
H2 IS TQE A TASK TYPE TQE
H3 DO BOTH CPUS HAVE SAME TQE
IEAQTD01
H4 DEQUE EEA2 REMOVE TQE FROM TIMER QUEUE
J2
DJS03
J2 ENQUE/DEQUE PERFORM INPUT OPERATION
K2 EXIT
RETURN TO CALLER
B4
B4 IS INPUT TCB = 'NEW'
B5 EXIT
RETURN TO CALLER
C4 IS J/S TQE A WAIT LIMIT TQE
C5 EXIT
RETURN TO CALLER
IEAQTD01
D4 DEQUE EEA2 REMOVE THE TQE FROM TIMER QUEUE
E4 CONVERT TQE TO TASK TYPE WITH ACTUAL TIME REMAINING
F4 SET INPUT OPERATION NQ
J2
J4 NQ OR DQ CALLED FOR
NQ
DQ
J5 HALVE ABSOLUTE TIME REMAINING
DJS08
K4 DOUBLE ABSOLUTE TIME REMAINING
DJS09
K5 STORE ADJUSTED TIME REMAINING. SET NQ/DQ REGISTER FOR NQ OPERATION
J2

Chart HA. EOT Routine

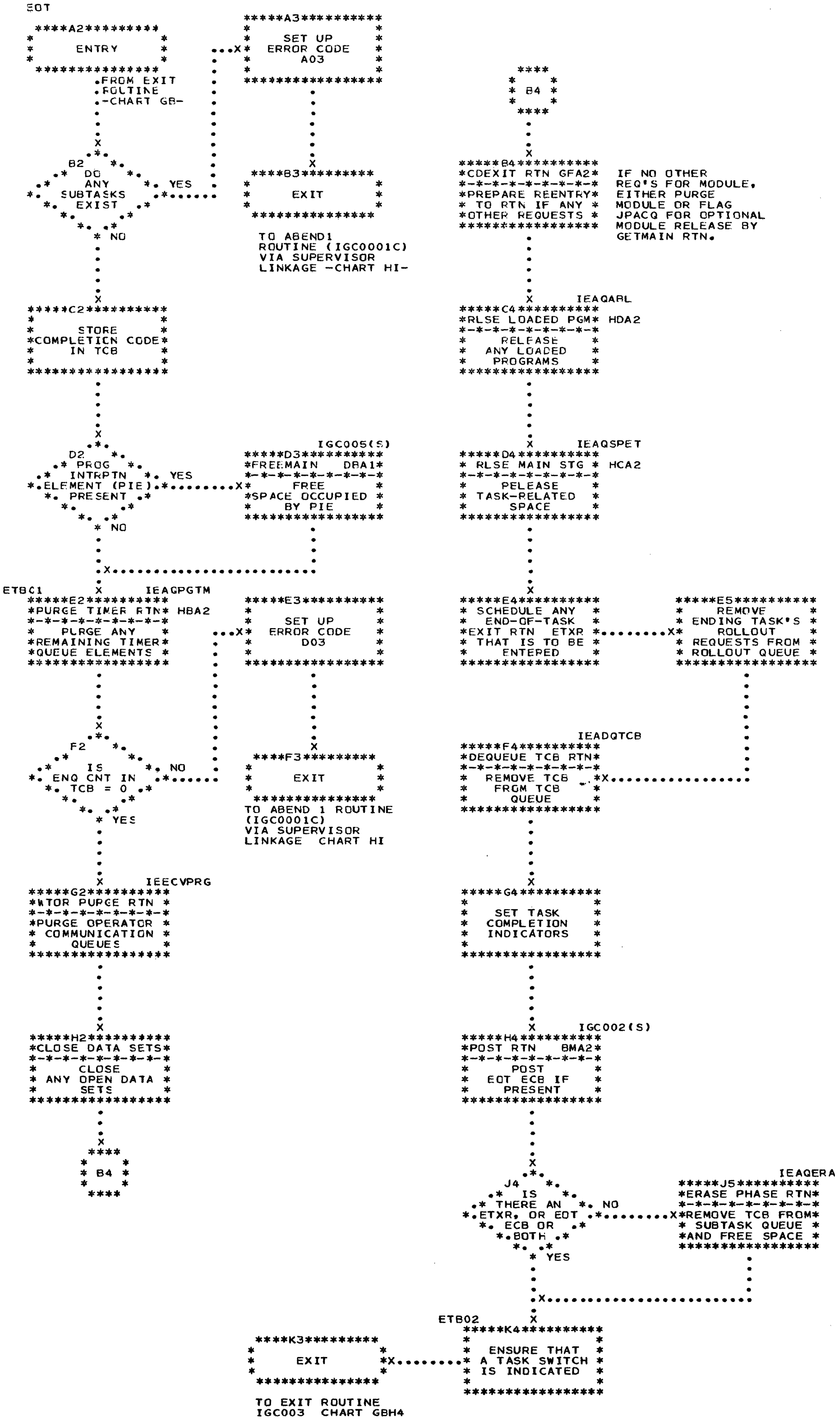

EOT
A2
ENTRY
FROM EXIT FOLTINE -CHART GB-
B2
DO ANY SUBTASKS EXIST
YES
NO
A3
SET UP ERROR CODE A03
B3
EXIT
TO ABEND1 ROUTINE (IGC0001C) VIA SUPERVISOR LINKAGE -CHART HI-
C2
STORE COMPLETICN CODE IN TCB
D2
PROG INTRPTN ELEMENT (PIE) PRESENT
YES
NO
IGC005(S)
D3
FREEMAIN DBA1
FREE SPACE OCCUPIED BY PIE
ETBC1
IEAGPGTM
E2
PURGE TIMER RTN HBA2
PURGE ANY REMAINING TIMER QUEUE ELEMENTS
E3
SET UP ERROR CODE D03
F2
IS ENQ CNT IN TCB = 0
NO
YES
F3
EXIT
TO ABEND 1 ROUTINE (IGC0001C) VIA SUPERVISOR LINKAGE CHART HI
IEECVPRG
G2
WTOR PURGE RTN
PURGE OPERATOR COMMUNICATION QUEUES
H2
CLOSE DATA SETS
CLOSE ANY OPEN DATA SETS
B4
B4
B4
CDEXIT RTN GFA2
PREPARE REENTRY TO RTN IF ANY OTHER REQUESTS
IF NO OTHER REQ'S FOR MODULE, EITHER PURGE MODULE OR FLAG JPACQ FOR OPTIONAL MODULE RELEASE BY GETMAIN RTN.
IEAQARL
C4
RLSE LOADED PGM HDA2
RELEASE ANY LOADED PROGRAMS
IEAQSPET
D4
RLSE MAIN STG HCA2
PELEASE TASK-RELATED SPACE
E4
SCHEDULE ANY END-OF-TASK EXIT RTN ETXR THAT IS TO BE ENTEPED
E5
REMOVE ENDING TASK'S ROLLOUT REQUESTS FROM ROLLOUT QUEUE
IEADQTCB
F4
DEQUEUE TCB RTN
REMOVE TCB FROM TCB QUEUE
G4
SET TASK COMPLETION INDICATORS
IGC002(S)
H4
POST RTN BMA2
POST EOT ECB IF PRESENT
J4
IS THERE AN ETXR, OR EOT ECB OR BOTH
NO
YES
IEAQERA
J5
ERASE PHASE RTN
REMOVE TCB FROM SUBTASK QUEUE AND FREE SPACE
ETB02
K4
ENSURE THAT A TASK SWITCH IS INDICATED
K3
EXIT
TO EXIT ROUTINE IGC003 CHART GBH4

Chart HB. Purge Timer Routine

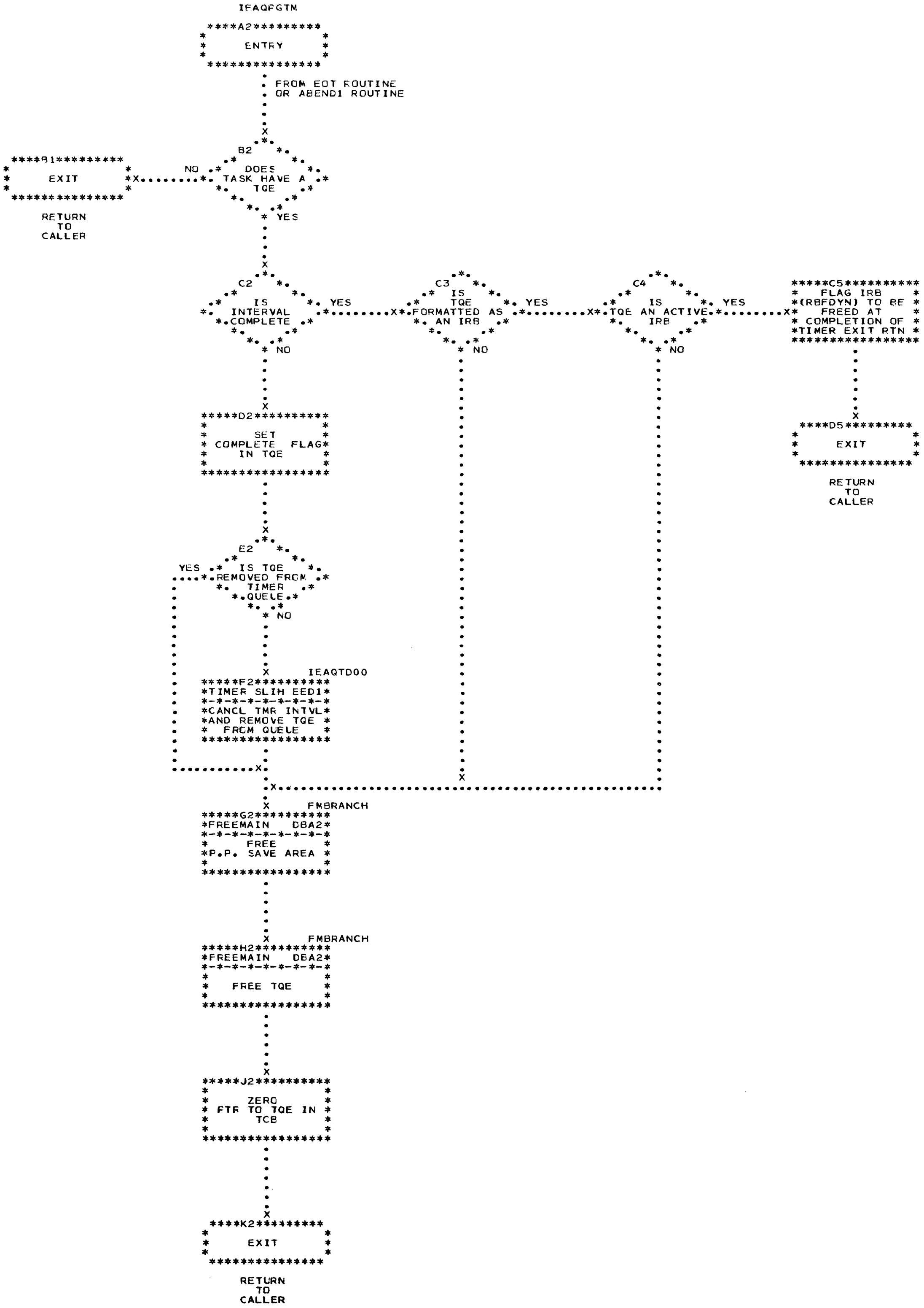


Chart HC. Release Main Storage Routine

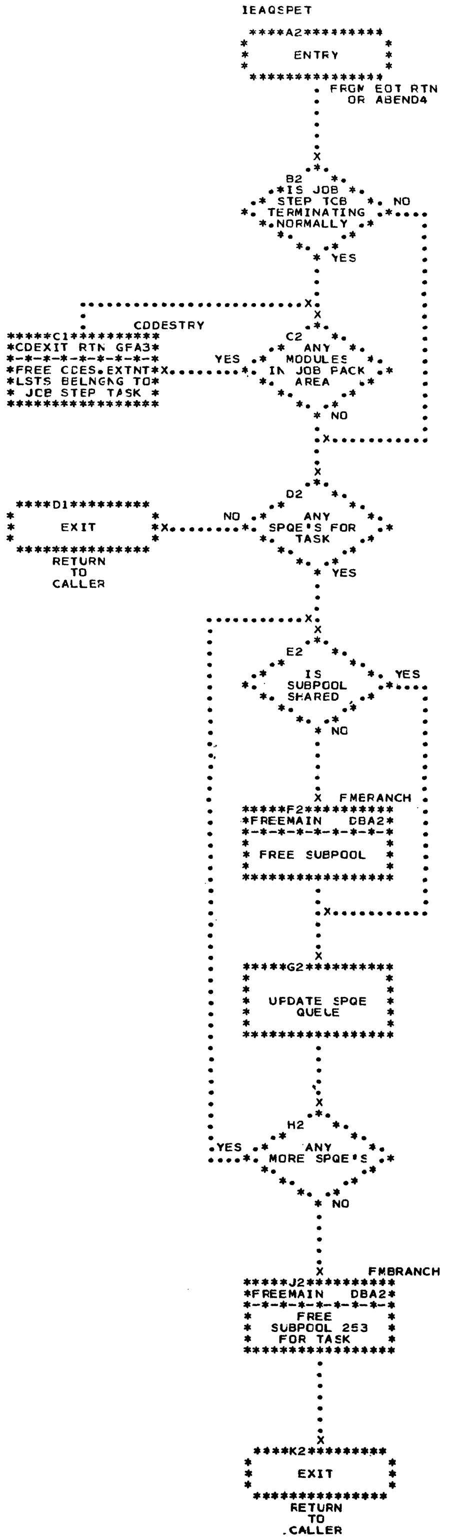


Chart HD. Release Loaded Programs Routine

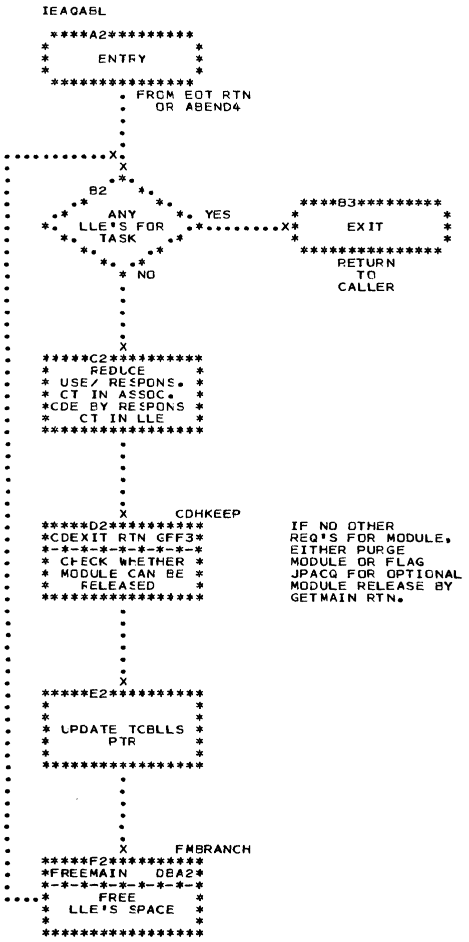

IEAQABL
A2
ENTRY
FROM EOT RTN OR ABEND4
B2
ANY LLE'S FOR TASK
YES
NO
B3
EXIT
RETURN TO CALLER
C2
REDUCE USE/ RESPONS. CT IN ASSOC. CDE BY RESPONS CT IN LLE
CDHKEEP
D2
CDEXIT RTN GFF3
CHECK WHETHER MODULE CAN BE RELEASED
IF NO OTHER REQ'S FOR MODULE, EITHER PURGE MODULE OR FLAG JPACQ FOR OPTIONAL MODULE RELEASE BY GETMAIN RTN.
E2
UPDATE TCBLLS PTR
FMBRANCH
F2
FREEMAIN DBA2
FREE LLE'S SPACE

Chart HE. ABTERM Routine (Part 1 of 2)

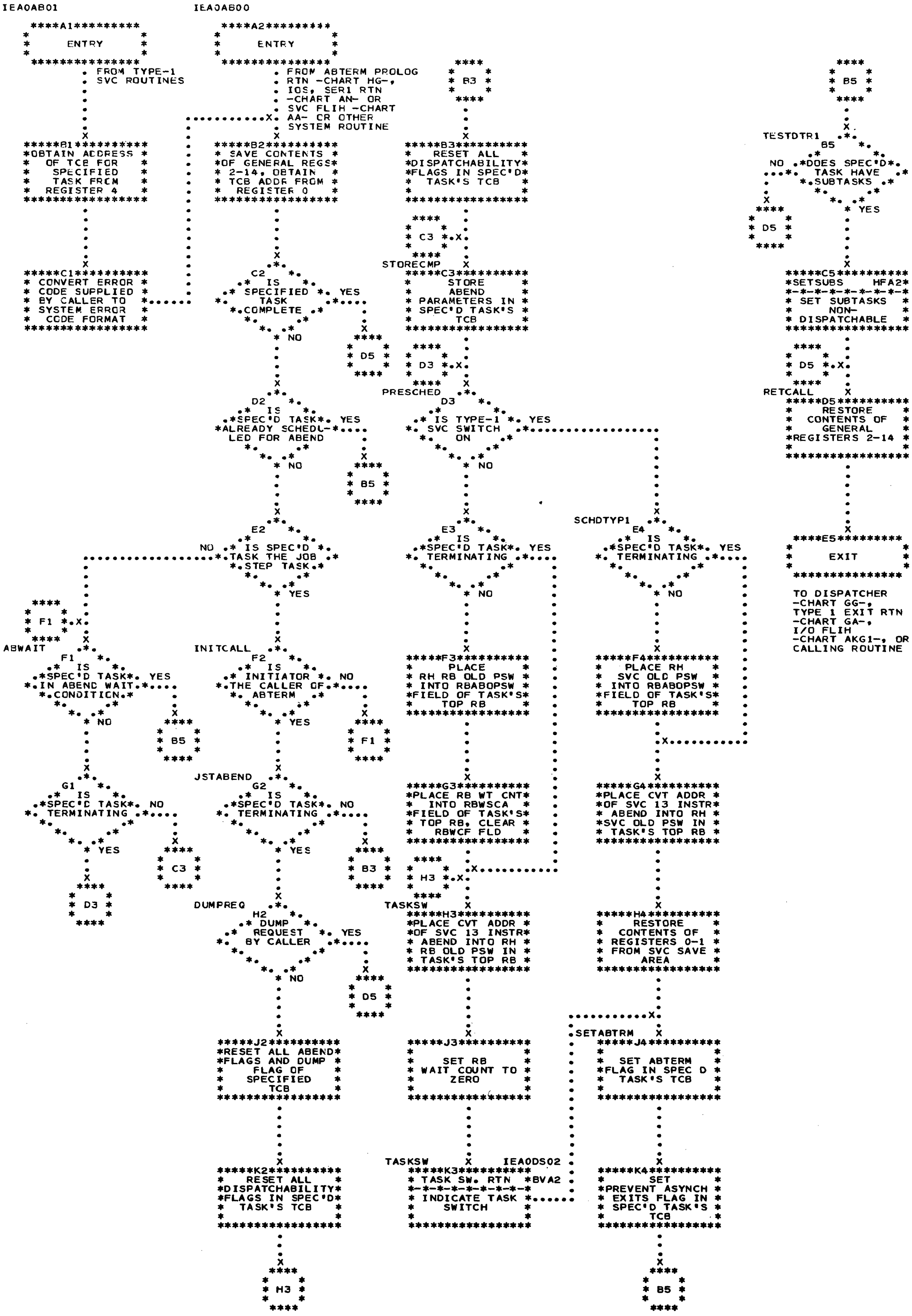

IEAOAB01
A1
ENTRY
FROM TYPE-1 SVC ROUTINES
IEAOAB00
A2
ENTRY
FROM ABTERM PROLOG RTN -CHART HG-, IOS, SERI RTN -CHART AN- OR SVC FLIH -CHART AA- OR OTHER SYSTEM ROUTINE
B3
B5
B1
OBTAIN ADDRESS OF TCB FOR SPECIFIED TASK FROM REGISTER 4
B2
SAVE CONTENTS OF GENERAL REGS 2-14, OBTAIN TCB ADDR FROM REGISTER 0
B3
RESET ALL DISPATCHABILITY FLAGS IN SPEC'D TASK'S TCB
TESTDTR1
B5
DOES SPEC'D TASK HAVE SUBTASKS
NO
YES
D5
C1
CONVERT ERROR CODE SUPPLIED BY CALLER TO SYSTEM ERROR CODE FORMAT
C2
IS SPECIFIED TASK COMPLETE
YES
NO
D5
C3
STORECMP
C3
STORE ABEND PARAMETERS IN SPEC'D TASK'S TCB
C5
SETSUBS HFA2
SET SUBTASKS NON-DISPATCHABLE
D3
D5
PRESCHED
RETCALL
D2
IS SPEC'D TASK ALREADY SCHEDULED FOR ABEND
YES
NO
B5
D3
IS TYPE-1 SVC SWITCH ON
YES
NO
D5
RESTORE CONTENTS OF GENERAL REGISTERS 2-14
E2
IS SPEC'D TASK THE JOB STEP TASK
NO
YES
E3
IS SPEC'D TASK TERMINATING
YES
NO
SCHDTYP1
E4
IS SPEC'D TASK TERMINATING
YES
NO
E5
EXIT
TO DISPATCHER -CHART GG-, TYPE 1 EXIT RTN -CHART GA-, I/O FLIH -CHART AKG1-, OR CALLING ROUTINE
F1
ABWAIT
F1
IS SPEC'D TASK IN ABEND WAIT CONDITION
YES
NO
B5
INITCALL
F2
IS INITIATOR THE CALLER OF ABTERM
NO
YES
F1
F3
PLACE RH RB OLD PSW INTO RBABOPSW FIELD OF TASK'S TOP RB
F4
PLACE RH SVC OLD PSW INTO RBABOPSW FIELD OF TASK'S TOP RB
G1
IS SPEC'D TASK TERMINATING
NO
YES
C3
D3
JSTABEND
G2
IS SPEC'D TASK TERMINATING
NO
YES
B3
G3
PLACE RB WT CNT INTO RBWSCA FIELD OF TASK'S TOP RB, CLEAR RBWCF FLD
G4
PLACE CVT ADDR OF SVC 13 INSTR ABEND INTO RH SVC OLD PSW IN TASK'S TOP RB
H3
TASKSW
DUMPREQ
H2
DUMP REQUEST BY CALLER
YES
NO
D5
H3
PLACE CVT ADDR OF SVC 13 INSTR ABEND INTO RH RB OLD PSW IN TASK'S TOP RB
H4
RESTORE CONTENTS OF REGISTERS 0-1 FROM SVC SAVE AREA
SETABTRM
J2
RESET ALL ABEND FLAGS AND DUMP FLAG OF SPECIFIED TCB
J3
SET RB WAIT COUNT TO ZERO
J4
SET ABTERM FLAG IN SPEC D TASK'S TCB
TASKSW
IEAODS02
K3
TASK SW. RTN BVA2
INDICATE TASK SWITCH
K2
RESET ALL DISPATCHABILITY FLAGS IN SPEC'D TASK'S TCB
K4
SET PREVENT ASYNCH EXITS FLAG IN SPEC'D TASK'S TCB
H3
B5

Chart HF. ABTERM Routine (Part 2 of 2) -- SETSUBS Subroutine

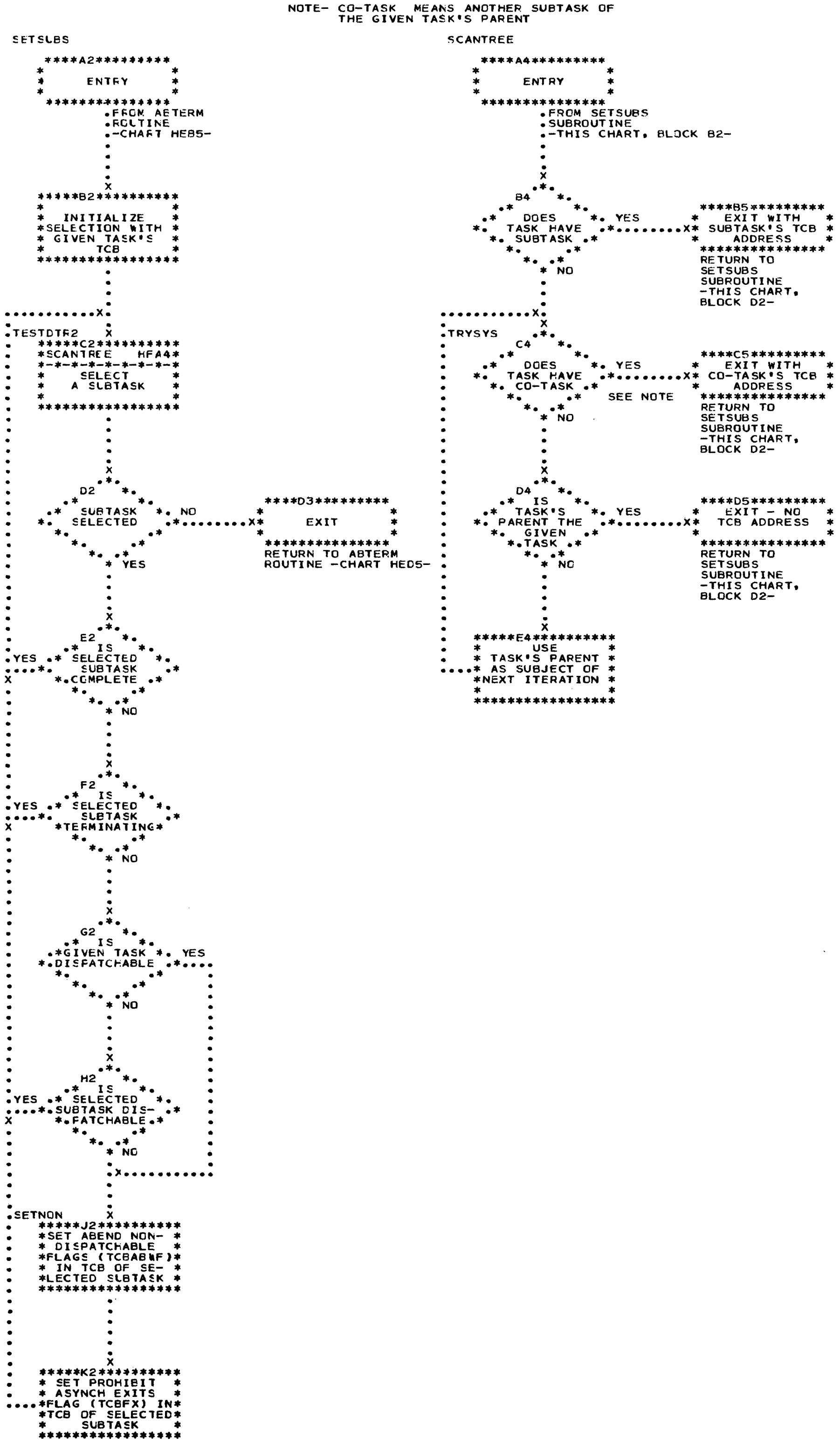
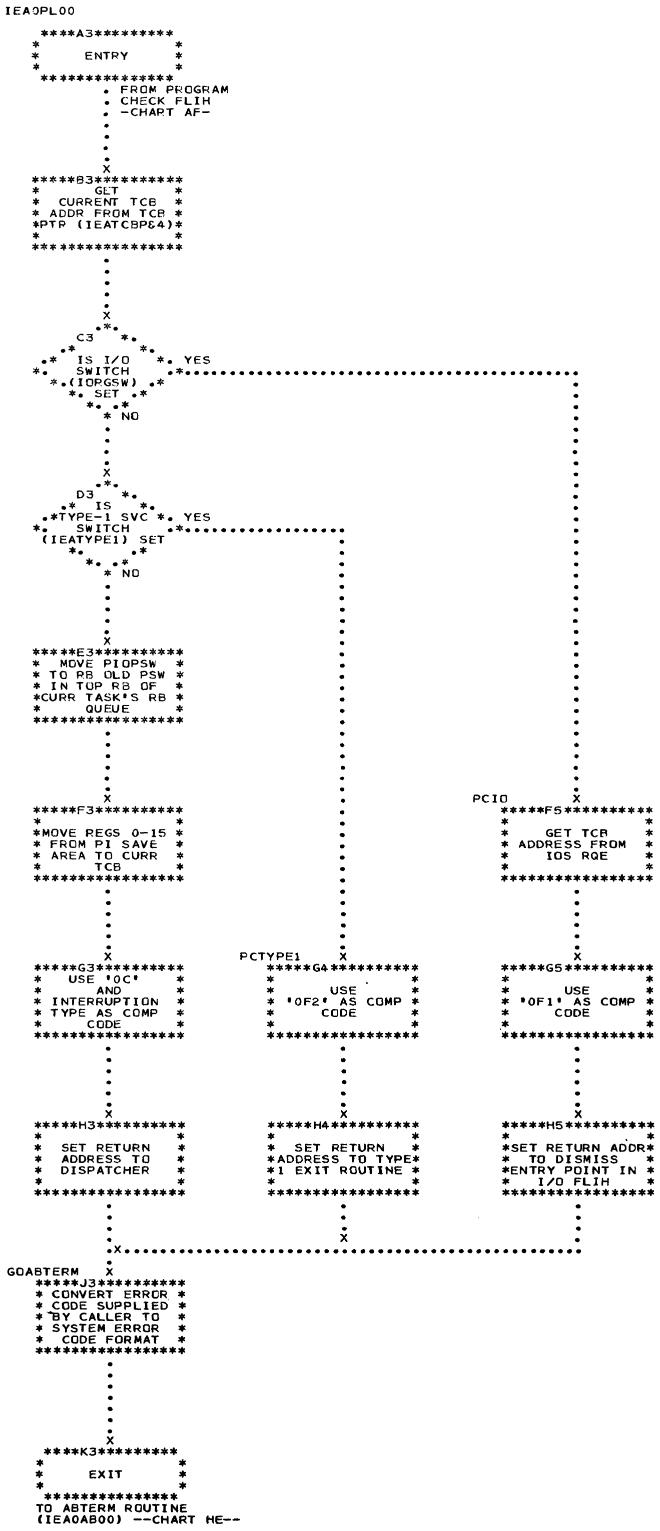
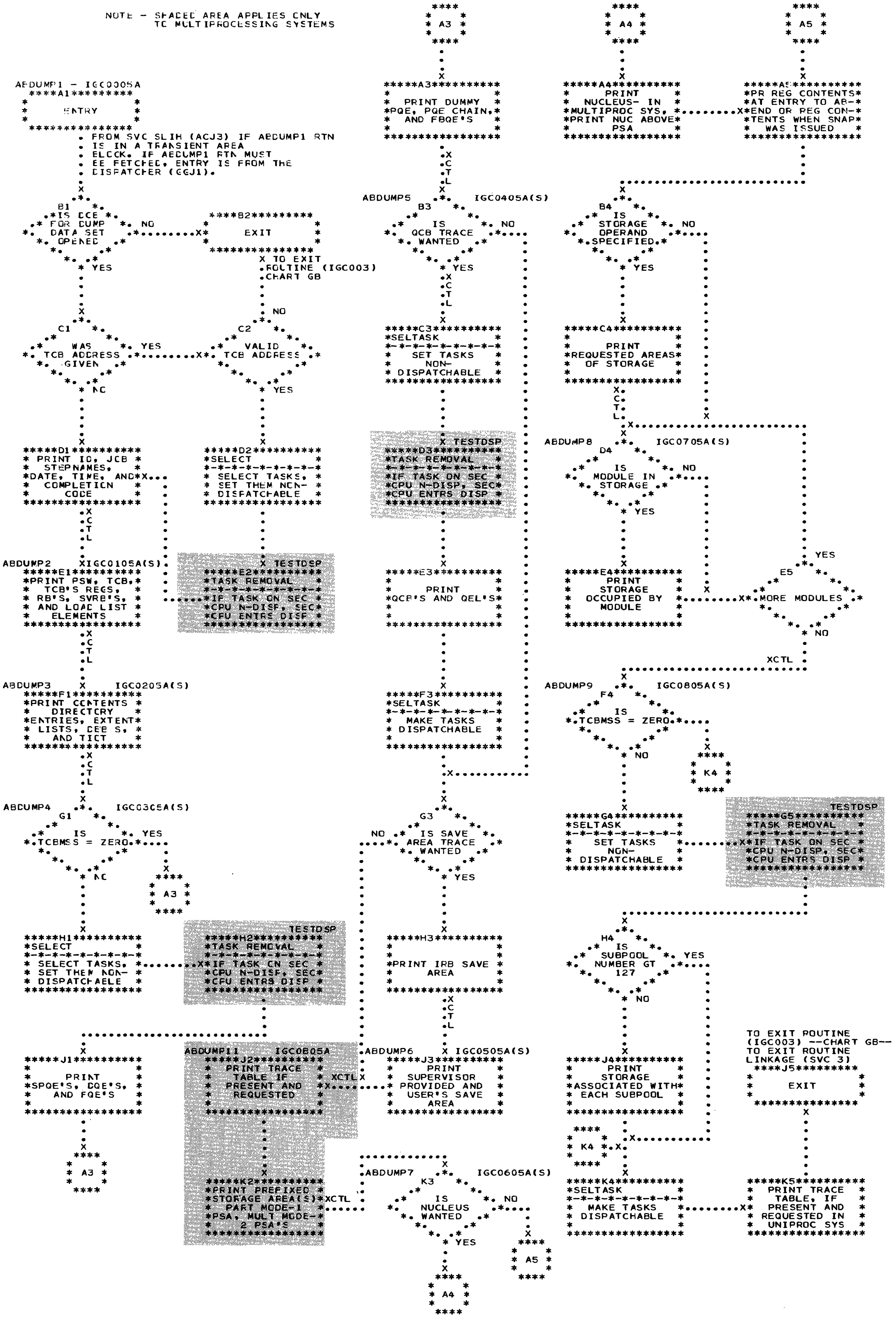

NOTE- CO-TASK MEANS ANOTHER SUBTASK OF THE GIVEN TASK'S PARENT
SETSUBS
A2
ENTRY
FROM ABTERM ROUTINE -CHART HEB5-
B2
INITIALIZE SELECTION WITH GIVEN TASK'S TCB
TESTDTR2
C2
SCANTREE HFA4
SELECT A SUBTASK
D2
SUBTASK SELECTED
NO
YES
D3
EXIT
RETURN TO ABTERM ROUTINE -CHART HED5-
E2
IS SELECTED SUBTASK COMPLETE
YES
NO
F2
IS SELECTED SUBTASK TERMINATING
YES
NO
G2
IS GIVEN TASK DISPATCHABLE
YES
NO
H2
IS SELECTED SUBTASK DIS-FATCHABLE
YES
NO
SETNON
J2
SET ABEND NON-DISPATCHABLE FLAGS (TCBABNF) IN TCB OF SE-LECTED SUBTASK
K2
SET PROHIBIT ASYNCH EXITS FLAG (TCBFX) IN TCB OF SELECTED SUBTASK
SCANTREE
A4
ENTRY
FROM SETSUBS SUBROUTINE -THIS CHART, BLOCK B2-
B4
DOES TASK HAVE SUBTASK
YES
NO
B5
EXIT WITH SUBTASK'S TCB ADDRESS
RETURN TO SETSUBS SUBROUTINE -THIS CHART, BLOCK D2-
TRYSYS
C4
DOES TASK HAVE CO-TASK
YES
SEE NOTE
NO
C5
EXIT WITH CO-TASK'S TCB ADDRESS
RETURN TO SETSUBS SUBROUTINE -THIS CHART, BLOCK D2-
D4
IS TASK'S PARENT THE GIVEN TASK
YES
NO
D5
EXIT - NO TCB ADDRESS
RETURN TO SETSUBS SUBROUTINE -THIS CHART, BLOCK D2-
E4
USE TASK'S PARENT AS SUBJECT OF NEXT ITERATION

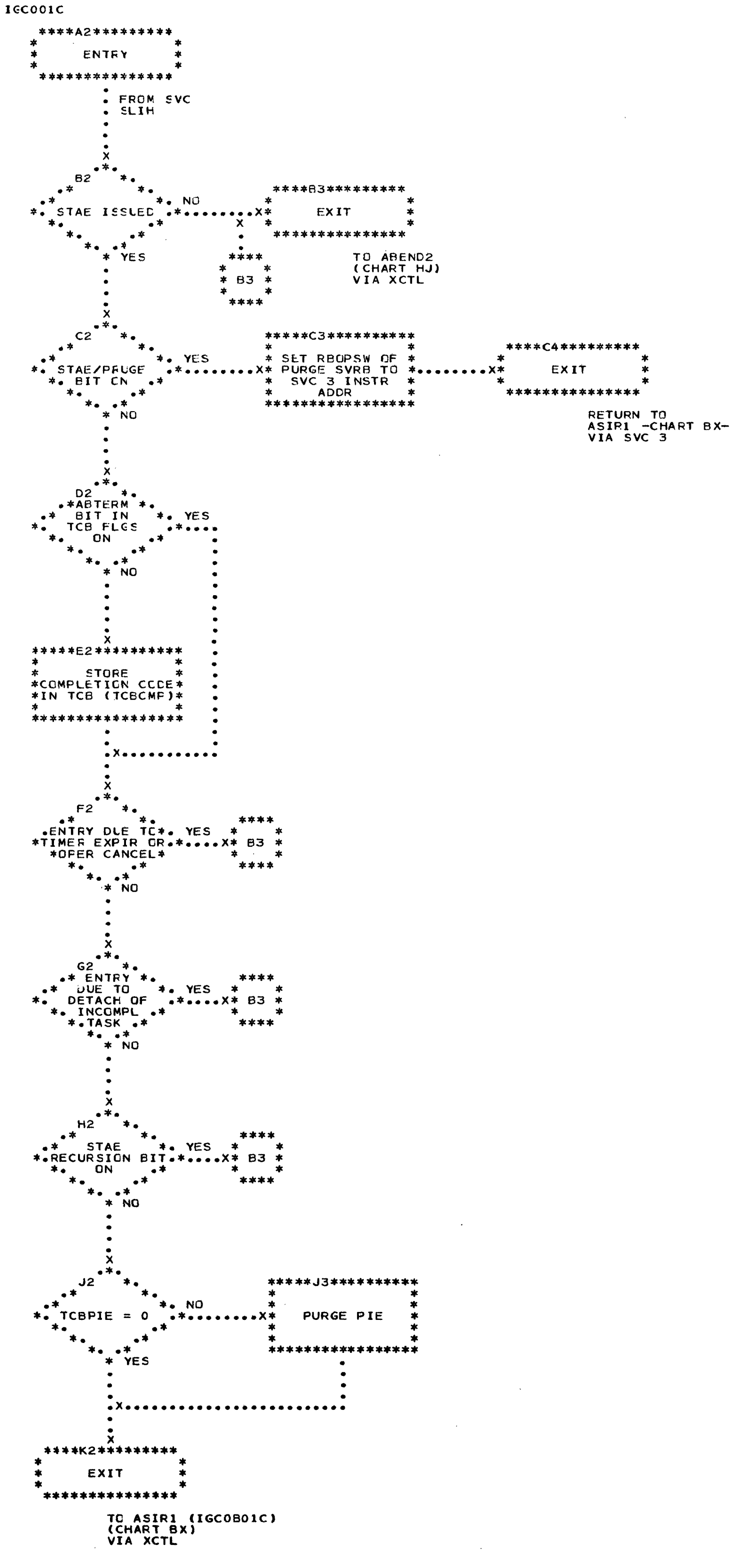
Chart HG. ABTERM Prologue Routine


IEAOPL00
A3
ENTRY
FROM PROGRAM CHECK FLIH -CHART AF-
B3
GET CURRENT TCB ADDR FROM TCB PTR (IEATCBP&4)
C3
IS I/O SWITCH (IORGSW) SET
YES
NO
D3
IS TYPE-1 SVC SWITCH (IEATYPE1) SET
YES
NO
E3
MOVE PIOPSW TO RB OLD PSW IN TOP RB OF CURR TASK'S RB QUEUE
F3
MOVE REGS 0-15 FROM PI SAVE AREA TO CURR TCB
PCIO
F5
GET TCB ADDRESS FROM IOS RQE
G3
USE '0C' AND INTERRUPTION TYPE AS COMP CODE
PCTYPE1
G4
USE '0F2' AS COMP CODE
G5
USE '0F1' AS COMP CODE
H3
SET RETURN ADDRESS TO DISPATCHER
H4
SET RETURN ADDRESS TO TYPE 1 EXIT ROUTINE
H5
SET RETURN ADDR TO DISMISS ENTRY POINT IN I/O FLIH
GOABTERM
J3
CONVERT ERROR CODE SUPPLIED BY CALLER TO SYSTEM ERROR CODE FORMAT
K3
EXIT
TO ABTERM ROUTINE (IEA0AB00) --CHART HE--

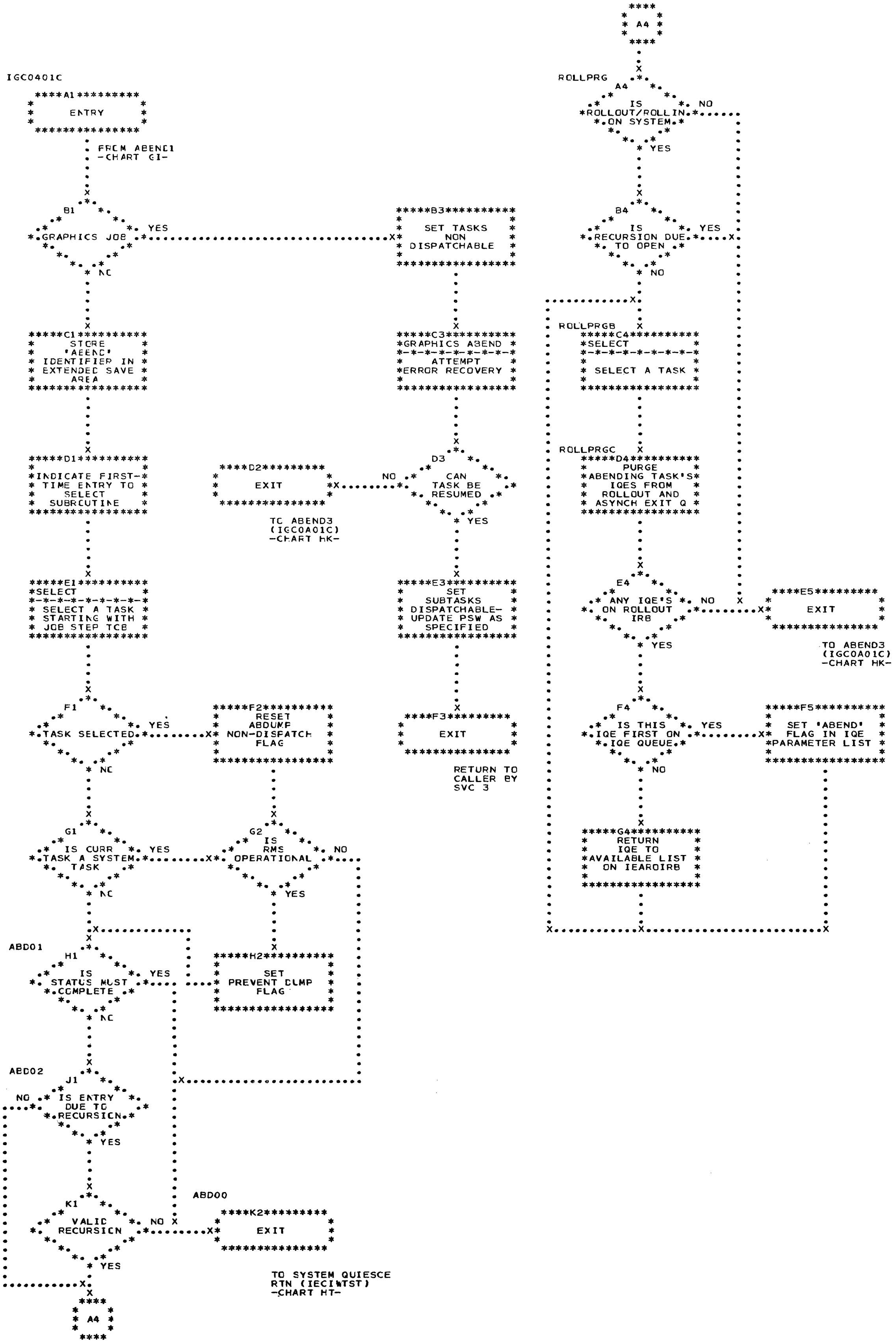
• Chart HH. ABDUMP Routine


NOTE - SFACEC AREA APPLIES CNLY TC MULTIFROCESSING SYSTEMS
AFDUMP1 - IGC0005A
A1
ENTRY
FROM SVC SLIH (ACJ3) IF AECUMP1 RTN IS IN A TRANSIENT AREA ELCCK. IF AECUMP1 RTN MUST EE FETCHEC, ENTRY IS FROM THE DISPATCHER (GGJ1).
B1
IS DCE FOR DUMP DATA SET OPENED
NO
YES
B2
EXIT
TO EXIT ROUTINE (IGC003) CHART GB
C1
WAS TCB ADDRESS GIVEN
YES
NC
C2
VALID TCB ADDRESS
NO
YES
D1
PRINT ID, JCB STEPNAMES, DATE, TIME, AND COMPLETION CODE
XCTL
D2
SELECT
SELECT TASKS, SET THEM NCN-DISPATCHABLE
ABDUMP2
IGC0105A(S)
E1
PRINT PSW, TCB, TCB'S REGS, RB'S, SVRB'S, AND LOAD LIST ELEMENTS
TESTDSP
E2
TASK REMOVAL
IF TASK ON SEC CPU N-DISP, SEC CPU ENTRS DISP
XCTL
ABDUMP3
IGC0205A(S)
F1
PRINT CONTENTS DIRECTORY ENTRIES, EXTENT LISTS, DEB S, AND TICT
XCTL
ABDUMP4
IGC0305A(S)
G1
IS TCBMSS = ZERO
YES
NC
A3
H1
SELECT
SELECT TASKS, SET THEM NON-DISPATCHAELE
TESTDSP
H2
TASK REMOVAL
IF TASK ON SEC CPU N-DISP, SEC CPU ENTRS DISP
J1
PRINT SPQE'S, DQE'S, AND FQE'S
A3
A3
A3
PRINT DUMMY PQE, PQE CHAIN, AND FBQE'S
XCTL
ABDUMP5
IGC0405A(S)
B3
IS QCB TRACE WANTED
NO
YES
XCTL
C3
SELTASK
SET TASKS NON-DISPATCHABLE
TESTDSP
D3
TASK REMOVAL
IF TASK ON SEC CPU N-DISP, SEC CPU ENTRS DISP
E3
PRINT QCB'S AND QEL'S
F3
SELTASK
MAKE TASKS DISPATCHABLE
G3
IS SAVE AREA TRACE WANTED
NO
YES
H3
PRINT IRB SAVE AREA
XCTL
ABDUMP11
IGC0805A
J2
PRINT TRACE TABLE IF PRESENT AND REQUESTED
XCTL
ABDUMP6
IGC0505A(S)
J3
PRINT SUPERVISOR PROVIDED AND USER'S SAVE AREA
K2
PRINT PREFIXED STORAGE AREA(S) PART MODE-1 PSA, MULT MODE-2 PSA'S
XCTL
ABDUMP7
IGC0605A(S)
K3
IS NUCLEUS WANTED
NO
YES
A4
A5
A4
A4
PRINT NUCLEUS- IN MULTIPROC SYS, PRINT NUC ABOVE PSA
A5
A5
PR REG CONTENTS AT ENTRY TO AB-END OR REG CON-TENTS WHEN SNAP WAS ISSUED
B4
IS STORAGE OPERAND SPECIFIED
NO
YES
C4
PRINT REQUESTED AREAS OF STORAGE
XCTL
ABDUMP8
IGC0705A(S)
D4
IS MODULE IN STORAGE
NO
YES
E4
PRINT STORAGE OCCUPIED BY MODULE
E5
MORE MODULES
YES
NO
XCTL
ABDUMP9
IGC0805A(S)
F4
IS TCBMSS = ZERO
NO
K4
G4
SELTASK
SET TASKS NON-DISPATCHABLE
TESTDSP
G5
TASK REMOVAL
IF TASK ON SEC CPU N-DISP, SEC CPU ENTRS DISP
H4
IS SUBPOOL NUMBER GT 127
YES
NO
J4
PRINT STORAGE ASSOCIATED WITH EACH SUBPOOL
K4
K4
SELTASK
MAKE TASKS DISPATCHABLE
TO EXIT ROUTINE (IGC003) --CHART GB-- TO EXIT ROUTINE LINKAGE (SVC 3)
J5
EXIT
K5
PRINT TRACE TABLE, IF PRESENT AND REQUESTED IN UNIPROC SYS

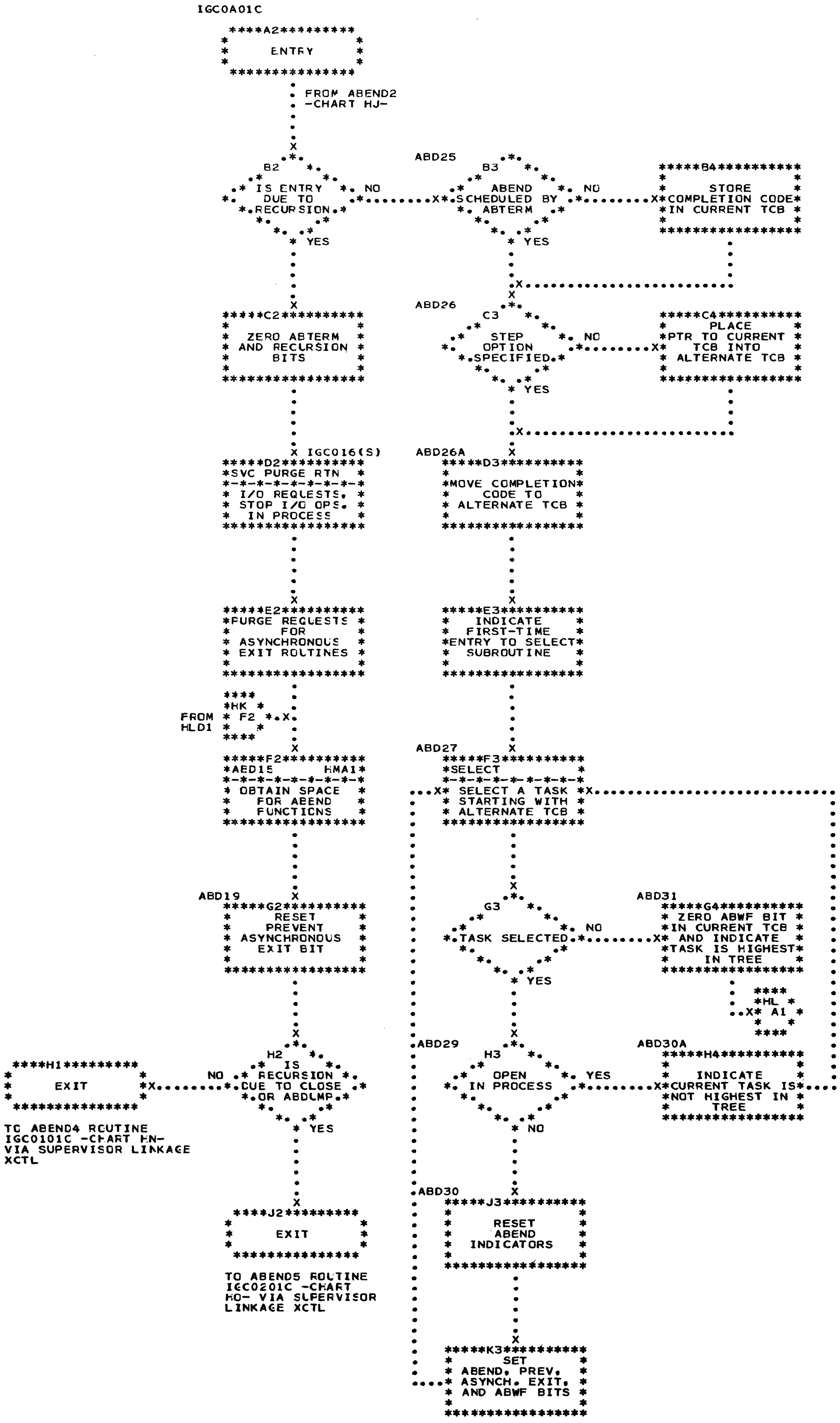
• Chart HI. ABEND1


IGC001C
A2
ENTRY
FROM SVC SLIH
B2
STAE ISSUED
NO
YES
B3
EXIT
TO ABEND2 (CHART HJ) VIA XCTL
B3
C2
STAE/PURGE BIT ON
YES
NO
C3
SET RBOPSW OF PURGE SVRB TO SVC 3 INSTR ADDR
C4
EXIT
RETURN TO ASIR1 -CHART BX- VIA SVC 3
D2
ABTERM BIT IN TCB FLGS ON
YES
NO
E2
STORE COMPLETION CODE IN TCB (TCBCMP)
F2
ENTRY DUE TO TIMER EXPIR OR OPER CANCEL
YES
B3
NO
G2
ENTRY DUE TO DETACH OF INCOMPL TASK
YES
B3
NO
H2
STAE RECURSION BIT ON
YES
B3
NO
J2
TCBPIE = 0
NO
J3
PURGE PIE
YES
K2
EXIT
TO ASIR1 (IGC0B01C) (CHART BX) VIA XCTL

• Chart HJ. ABEND2

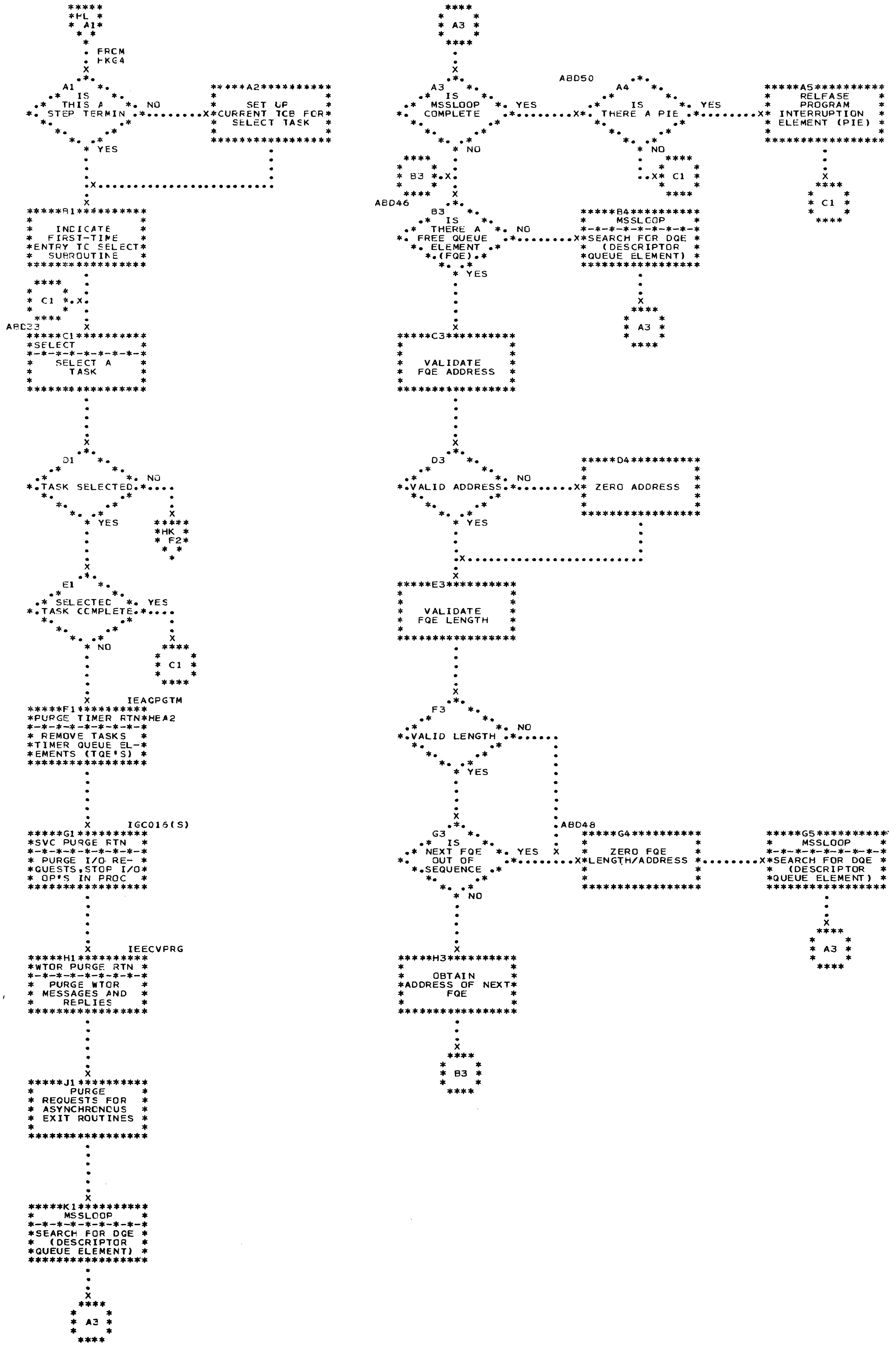

IGC0401C
A1
ENTRY
FROM ABEND1 -CHART GI-
B1
GRAPHICS JOB
YES
NO
B3
SET TASKS NON DISPATCHABLE
C1
STORE 'ABEND' IDENTIFIER IN EXTENDED SAVE AREA
C3
GRAPHICS ABEND
ATTEMPT ERROR RECOVERY
D1
INDICATE FIRST-TIME ENTRY TO SELECT SUBROUTINE
D2
EXIT
TO ABEND3 (IGC0A01C) -CHART HK-
D3
CAN TASK BE RESUMED
NO
YES
E1
SELECT
SELECT A TASK STARTING WITH JOB STEP TCB
E3
SET SUBTASKS DISPATCHABLE- UPDATE PSW AS SPECIFIED
F1
TASK SELECTED
YES
NO
F2
RESET ABDUMP NON-DISPATCH FLAG
F3
EXIT
RETURN TO CALLER BY SVC 3
G1
IS CURR TASK A SYSTEM TASK
YES
NO
G2
IS RMS OPERATIONAL
NO
YES
ABD01
H1
IS STATUS MUST COMPLETE
YES
NO
H2
SET PREVENT DUMP FLAG
ABD02
J1
IS ENTRY DUE TO RECURSION
NO
YES
K1
VALID RECURSION
NO
YES
ABD00
K2
EXIT
TO SYSTEM QUIESCE RTN (IECIWTST) -CHART HT-
A4
A4
ROLLPRG
A4
IS ROLLOUT/ROLLIN ON SYSTEM
NO
YES
B4
IS RECURSION DUE TO OPEN
YES
NO
ROLLPRGB
C4
SELECT
SELECT A TASK
ROLLPRGC
D4
PURGE ABENDING TASK'S IQES FROM ROLLOUT AND ASYNCH EXIT Q
E4
ANY IQE'S ON ROLLOUT IRB
NO
YES
E5
EXIT
TO ABEND3 (IGC0A01C) -CHART HK-
F4
IS THIS IQE FIRST ON IQE QUEUE
YES
NO
F5
SET 'ABEND' FLAG IN IQE PARAMETER LIST
G4
RETURN IQE TO AVAILABLE LIST ON IEAROIRB

• Chart HK. ABEND3 (Part 1 of 3)

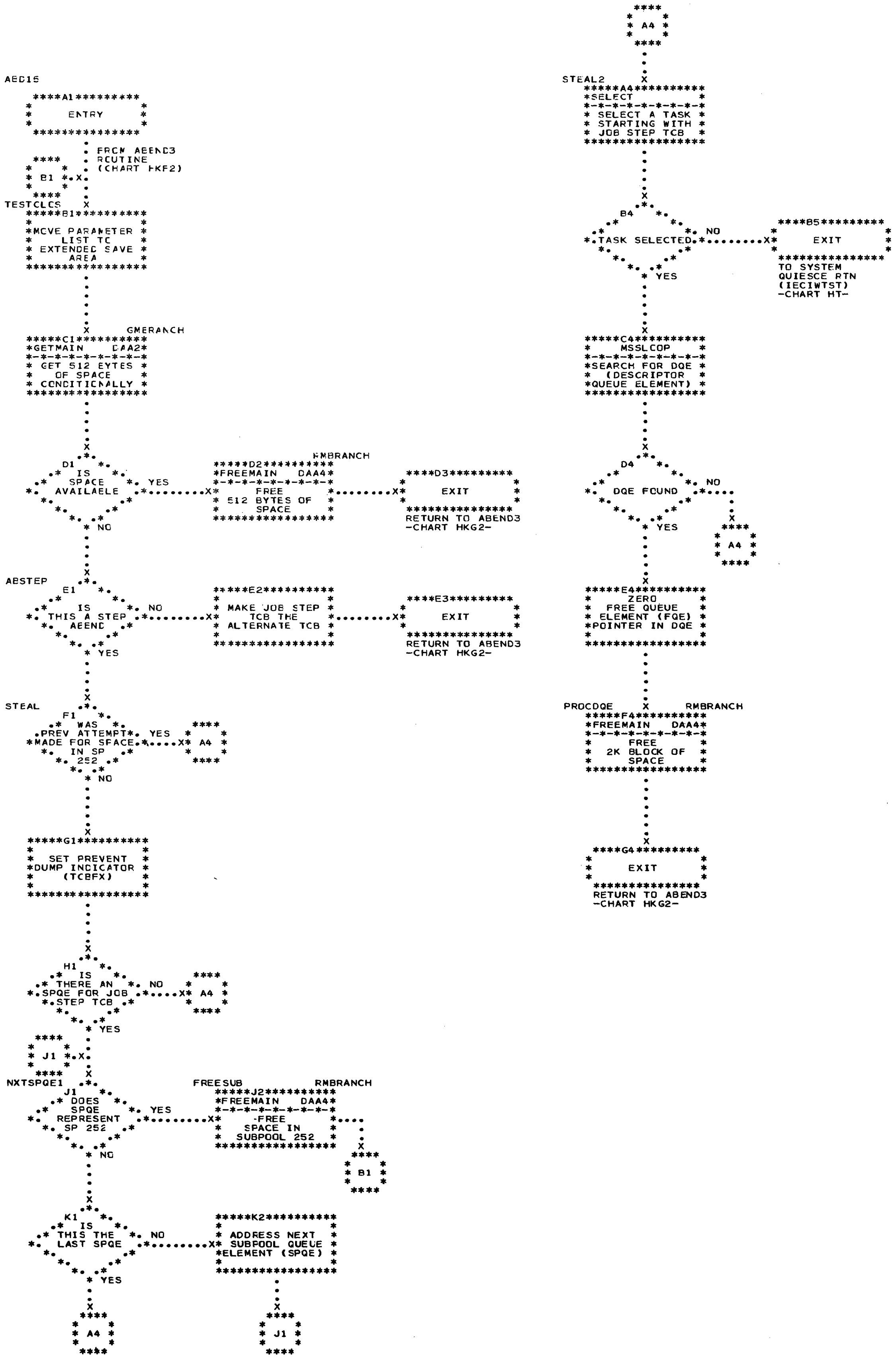


IGC0A01C
A2
ENTRY
FROM ABEND2 -CHART HJ-
B2
IS ENTRY DUE TO RECURSION
NO
YES
ABD25
B3
ABEND SCHEDULED BY ABTERM
NO
YES
B4
STORE COMPLETION CODE IN CURRENT TCB
C2
ZERO ABTERM AND RECURSION BITS
ABD26
C3
STEP OPTION SPECIFIED
NO
YES
C4
PLACE PTR TO CURRENT TCB INTO ALTERNATE TCB
IGC016(S)
D2
SVC PURGE RTN
I/O REQUESTS, STOP I/O OPS. IN PROCESS
ABD26A
D3
MOVE COMPLETION CODE TO ALTERNATE TCB
E2
PURGE REQUESTS FOR ASYNCHRONOUS EXIT ROUTINES
E3
INDICATE FIRST-TIME ENTRY TO SELECT SUBROUTINE
FROM HLD1
HK F2
F2
ABD15 HMA1
OBTAIN SPACE FOR ABEND FUNCTIONS
ABD27
F3
SELECT
SELECT A TASK STARTING WITH ALTERNATE TCB
ABD19
G2
RESET PREVENT ASYNCHRONOUS EXIT BIT
G3
TASK SELECTED
NO
YES
ABD31
G4
ZERO ABWF BIT IN CURRENT TCB AND INDICATE TASK IS HIGHEST IN TREE
HL A1
H1
EXIT
TO ABEND4 ROUTINE IGC0101C -CHART HN- VIA SUPERVISOR LINKAGE XCTL
H2
IS RECURSION DUE TO CLOSE OR ABDUMP
NO
YES
ABD29
H3
OPEN IN PROCESS
YES
NO
ABD30A
H4
INDICATE CURRENT TASK IS NOT HIGHEST IN TREE
J2
EXIT
TO ABEND5 ROUTINE IGC0201C -CHART HO- VIA SUPERVISOR LINKAGE XCTL
ABD30
J3
RESET ABEND INDICATORS
K3
SET ABEND, PREV. ASYNCH. EXIT, AND ABWF BITS

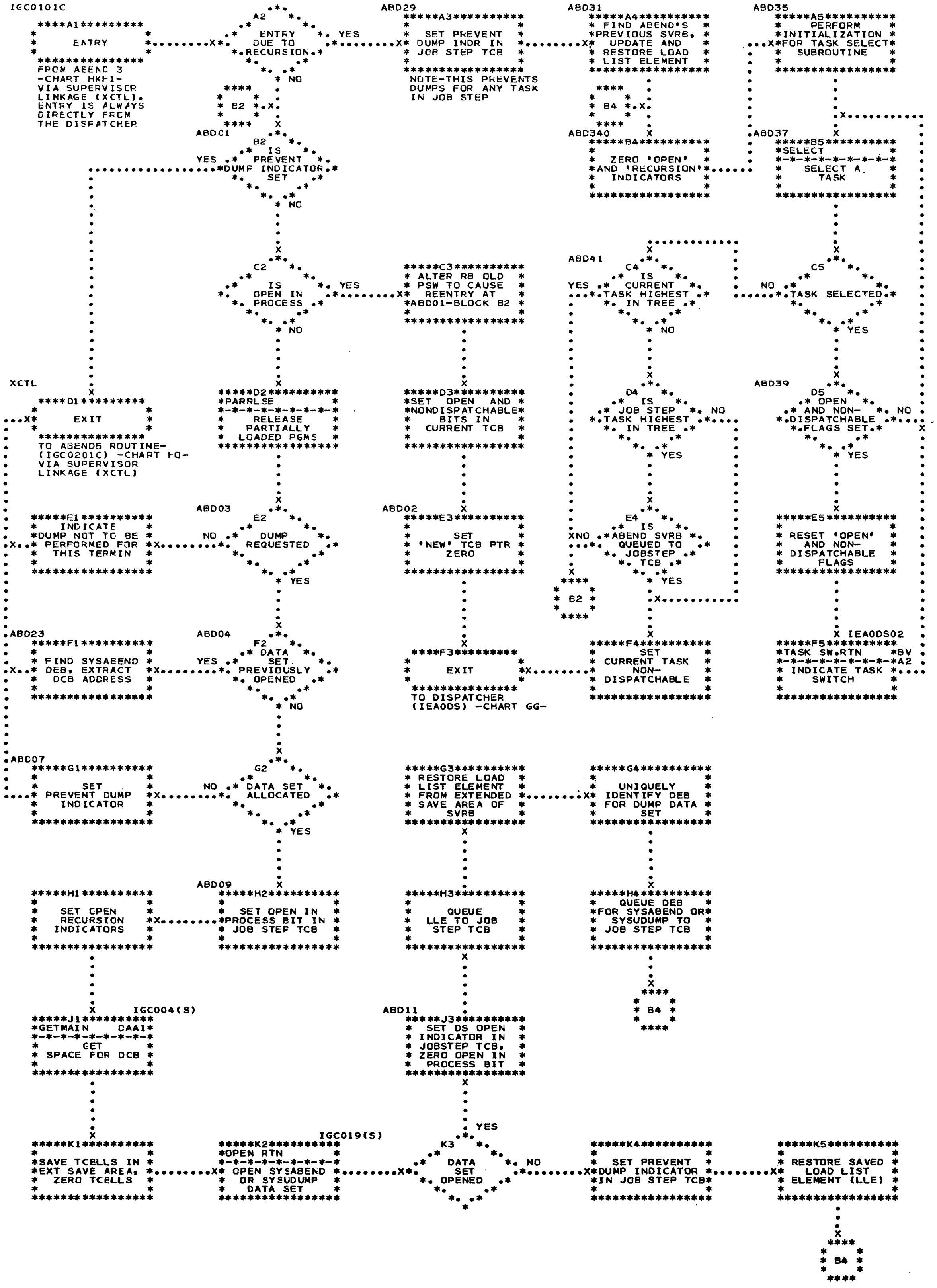
• Chart HL. ABEND3 (Part 2 of 3)


HL A1
FROM HKG4
A1 IS THIS A STEP TERMIN
NO
YES
A2 SET UP CURRENT TCB FOR SELECT TASK
B1 INDICATE FIRST-TIME ENTRY TO SELECT SUBROUTINE
C1
ABD33
C1 SELECT
SELECT A TASK
D1 TASK SELECTED
NO
YES
HK F2
E1 SELECTED TASK COMPLETE
YES
NO
C1
IEAGPGTM
F1 PURGE TIMER RTN HEA2
REMOVE TASKS TIMER QUEUE ELEMENTS (TQE'S)
IGC016(S)
G1 SVC PURGE RTN
PURGE I/O REQUESTS, STOP I/O OP'S IN PROC
IEECVPRG
H1 WTOR PURGE RTN
PURGE WTOR MESSAGES AND REPLIES
J1 PURGE REQUESTS FOR ASYNCHRONOUS EXIT ROUTINES
K1 MSSLOOP
SEARCH FOR DQE (DESCRIPTOR QUEUE ELEMENT)
A3
A3
A3 IS MSSLOOP COMPLETE
YES
NO
ABD50
A4 IS THERE A PIE
YES
NO
C1
A5 RELEASE PROGRAM INTERRUPTION ELEMENT (PIE)
C1
B3
ABD46
B3 IS THERE A FREE QUEUE ELEMENT (FQE)
NO
YES
B4 MSSLOOP
SEARCH FOR DQE (DESCRIPTOR QUEUE ELEMENT)
A3
C3 VALIDATE FQE ADDRESS
D3 VALID ADDRESS
NO
YES
D4 ZERO ADDRESS
E3 VALIDATE FQE LENGTH
F3 VALID LENGTH
NO
YES
ABD48
G3 IS NEXT FQE OUT OF SEQUENCE
YES
NO
G4 ZERO FQE LENGTH/ADDRESS
G5 MSSLOOP
SEARCH FOR DQE (DESCRIPTOR QUEUE ELEMENT)
A3
H3 OBTAIN ADDRESS OF NEXT FQE
B3

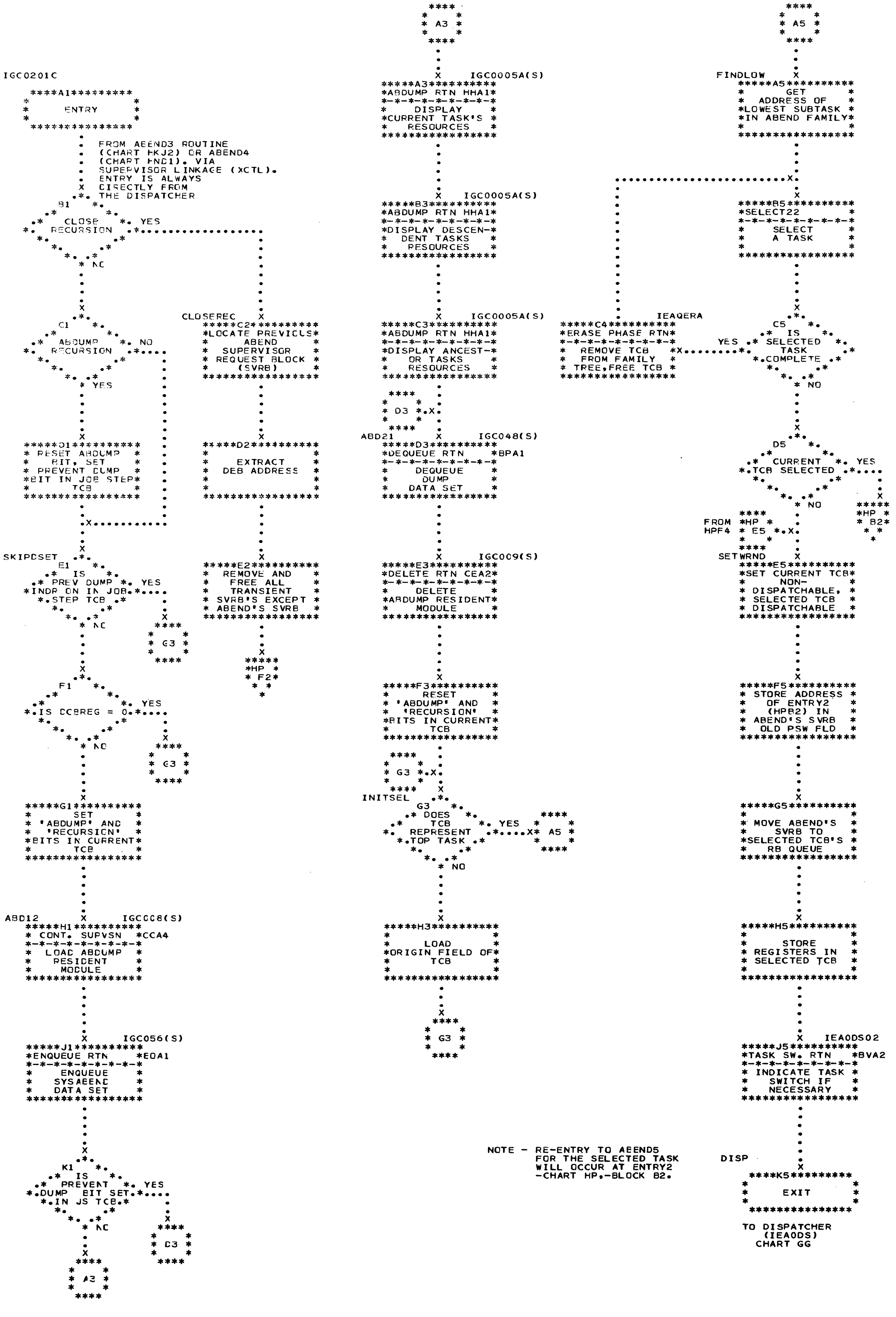
• Chart HM. ABEND3 (Part 3 of 3) -- Steal Core Subroutine


AEC15
A1
ENTRY
FROM ABEND3 ROUTINE (CHART HKF2)
B1
TESTCLCS
B1
MOVE PARAMETER LIST TO EXTENDED SAVE AREA
GMBRANCH
C1
GETMAIN DAA2
GET 512 BYTES OF SPACE CONDITIONALLY
D1
IS SPACE AVAILABLE
YES
NO
RMBRANCH
D2
FREEMAIN DAA4
FREE 512 BYTES OF SPACE
D3
EXIT
RETURN TO ABEND3 -CHART HKG2-
ABSTEP
E1
IS THIS A STEP ABEND
NO
YES
E2
MAKE JOB STEP TCB THE ALTERNATE TCB
E3
EXIT
RETURN TO ABEND3 -CHART HKG2-
STEAL
F1
WAS PREV ATTEMPT MADE FOR SPACE IN SP 252
YES
A4
NO
G1
SET PREVENT DUMP INDICATOR (TCBFX)
H1
IS THERE AN SPQE FOR JOB STEP TCB
NO
A4
YES
J1
NXTSPQE1
J1
DOES SPQE REPRESENT SP 252
YES
NO
FREESUB
RMBRANCH
J2
FREEMAIN DAA4
-FREE SPACE IN SUBPOOL 252
B1
K1
IS THIS THE LAST SPQE
NO
YES
A4
K2
ADDRESS NEXT SUBPOOL QUEUE ELEMENT (SPQE)
J1
A4
STEAL2
A4
SELECT
SELECT A TASK STARTING WITH JOB STEP TCB
B4
TASK SELECTED
NO
YES
B5
EXIT
TO SYSTEM QUIESCE RTN (IECIWTST) -CHART HT-
C4
MSSLCOP
SEARCH FOR DQE (DESCRIPTOR QUEUE ELEMENT)
D4
DQE FOUND
NO
A4
YES
E4
ZERO FREE QUEUE ELEMENT (FQE) POINTER IN DQE
PROCDQE
RMBRANCH
F4
FREEMAIN DAA4
FREE 2K BLOCK OF SPACE
G4
EXIT
RETURN TO ABEND3 -CHART HKG2-

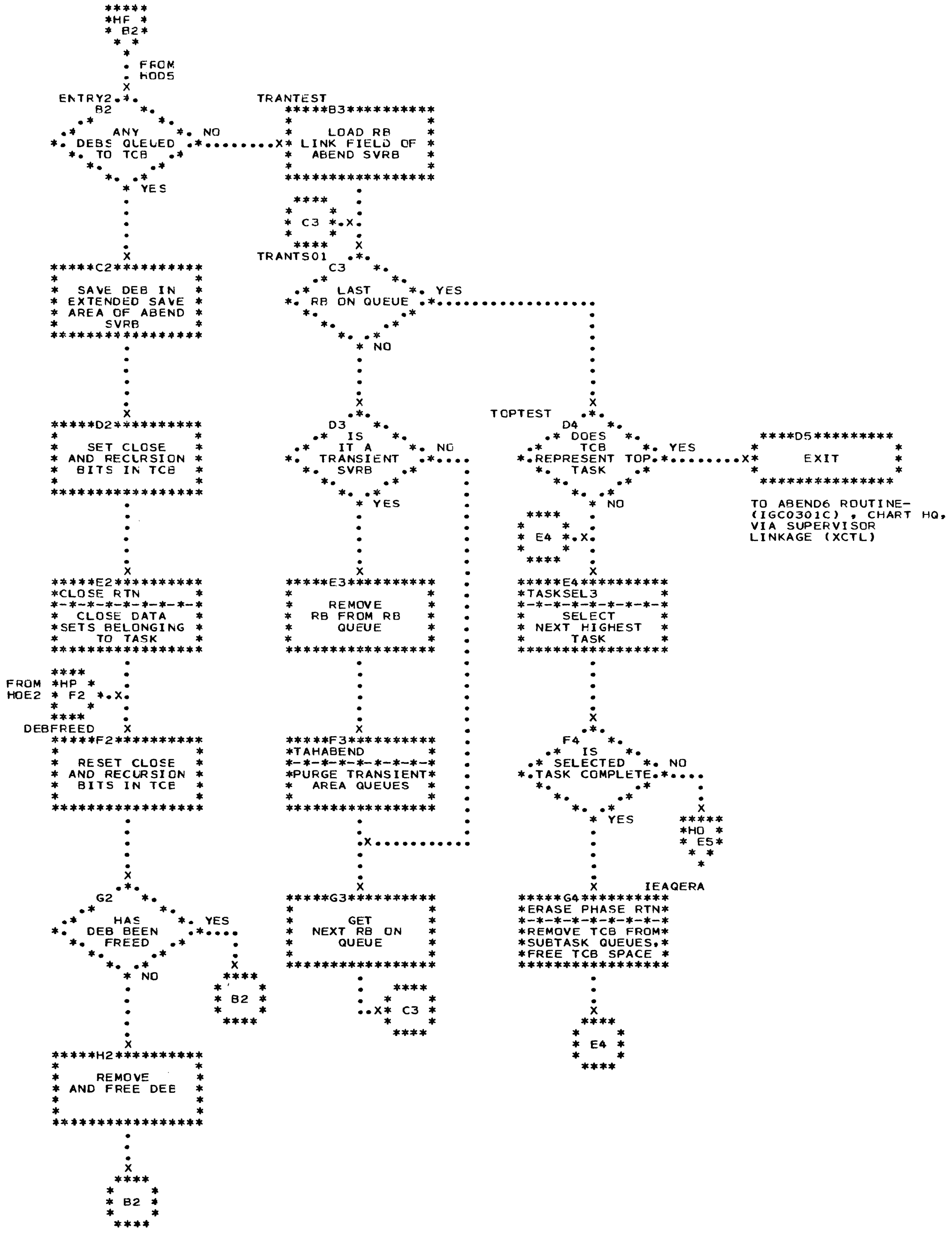
• Chart HN. ABEND4


IGC0101C
A1
ENTRY
FROM ABEND 3 -CHART HKH1- VIA SUPERVISOR LINKAGE (XCTL). ENTRY IS ALWAYS DIRECTLY FROM THE DISPATCHER
A2
ENTRY DUE TO RECURSION
YES
NO
ABD29
A3
SET PREVENT DUMP INDR IN JOB STEP TCB
NOTE-THIS PREVENTS DUMPS FOR ANY TASK IN JOB STEP
ABD31
A4
FIND ABEND'S PREVIOUS SVRB, UPDATE AND RESTORE LOAD LIST ELEMENT
ABD35
A5
PERFORM INITIALIZATION FOR TASK SELECT SUBROUTINE
B2
ABDC1
B2
IS PREVENT DUMP INDICATOR SET
YES
NO
B4
ABD340
B4
ZERO 'OPEN' AND 'RECURSION' INDICATORS
ABD37
B5
SELECT
SELECT A TASK
C2
IS OPEN IN PROCESS
YES
NO
C3
ALTER RB OLD PSW TO CAUSE REENTRY AT ABD01-BLOCK B2
ABD41
C4
IS CURRENT TASK HIGHEST IN TREE
YES
NO
C5
TASK SELECTED
NO
YES
XCTL
D1
EXIT
TO ABEND5 ROUTINE- (IGC0201C) -CHART HO- VIA SUPERVISOR LINKAGE (XCTL)
D2
PARRLSE
RELEASE PARTIALLY LOADED PGMS
D3
SET OPEN AND NONDISPATCHABLE BITS IN CURRENT TCB
D4
IS JOB STEP TASK HIGHEST IN TREE
NO
YES
ABD39
D5
OPEN AND NON-DISPATCHABLE FLAGS SET
NO
YES
E1
INDICATE DUMP NOT TO BE PERFORMED FOR THIS TERMIN
ABD03
E2
DUMP REQUESTED
NO
YES
ABD02
E3
SET 'NEW' TCB PTR ZERO
E4
IS ABEND SVRB QUEUED TO JOBSTEP TCB
NO
YES
B2
E5
RESET 'OPEN' AND NON-DISPATCHABLE FLAGS
ABD23
F1
FIND SYSABEND DEB, EXTRACT DCB ADDRESS
ABD04
F2
DATA SET PREVIOUSLY OPENED
YES
NO
F3
EXIT
TO DISPATCHER (IEAODS) -CHART GG-
F4
SET CURRENT TASK NON-DISPATCHABLE
IEAODS02
F5
TASK SW.RTN BV A2
INDICATE TASK SWITCH
ABD07
G1
SET PREVENT DUMP INDICATOR
G2
DATA SET ALLOCATED
NO
YES
G3
RESTORE LOAD LIST ELEMENT FROM EXTENDED SAVE AREA OF SVRB
G4
UNIQUELY IDENTIFY DEB FOR DUMP DATA SET
H1
SET OPEN RECURSION INDICATORS
ABD09
H2
SET OPEN IN PROCESS BIT IN JOB STEP TCB
H3
QUEUE LLE TO JOB STEP TCB
H4
QUEUE DEB FOR SYSABEND OR SYSUDUMP TO JOB STEP TCB
B4
IGC004(S)
J1
GETMAIN CAA1
GET SPACE FOR DCB
ABD11
J3
SET DS OPEN INDICATOR IN JOBSTEP TCB, ZERO OPEN IN PROCESS BIT
YES
K1
SAVE TCELLS IN EXT SAVE AREA, ZERO TCELLS
IGC019(S)
K2
OPEN RTN
OPEN SYSABEND OR SYSUDUMP DATA SET
K3
DATA SET OPENED
NO
K4
SET PREVENT DUMP INDICATOR IN JOB STEP TCB
K5
RESTORE SAVED LOAD LIST ELEMENT (LLE)
B4

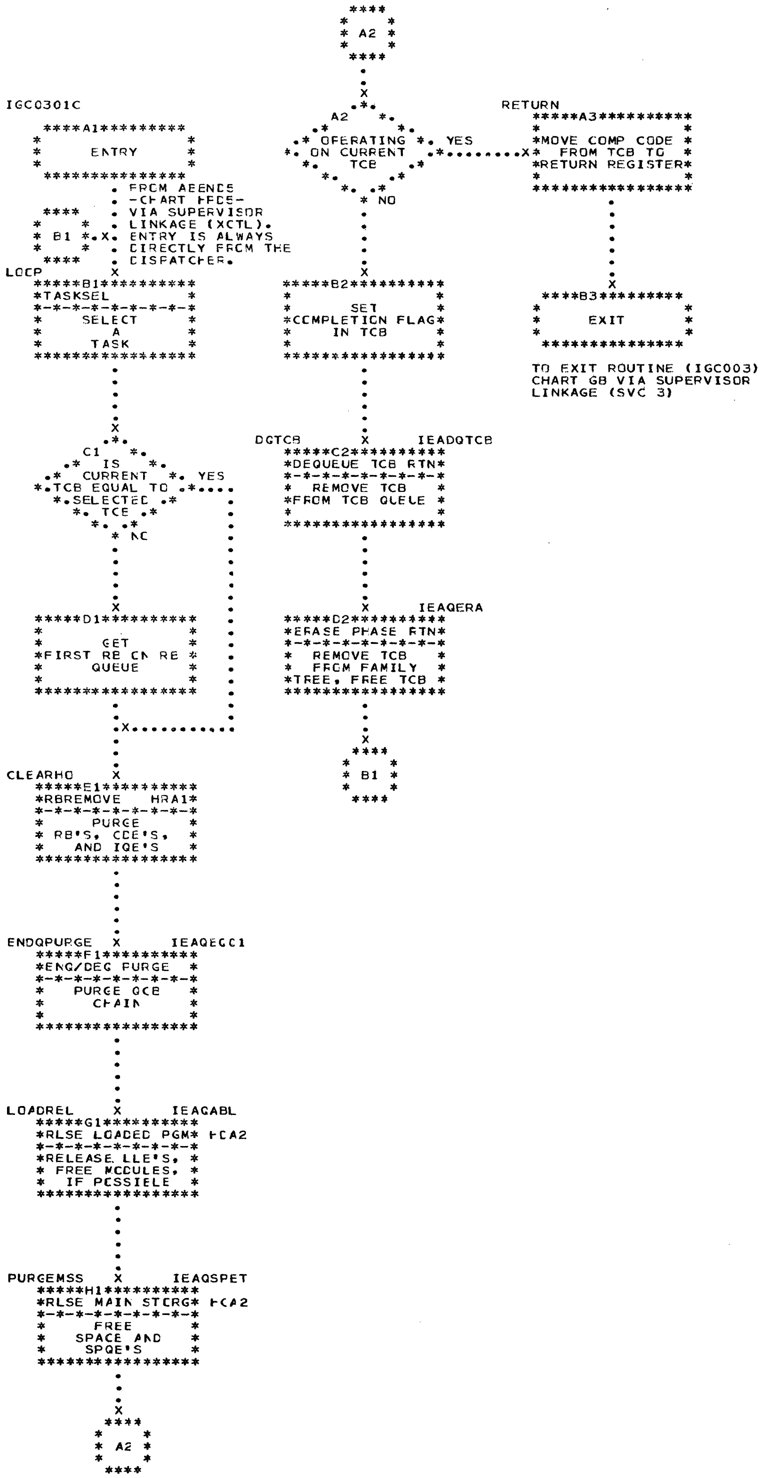
• Chart HO. ABEND5 (Part 1 of 2)


IGC0201C
A1 ENTRY
FROM AEEND3 ROUTINE (CHART FKJ2) OR ABEND4 (CHART FND1), VIA SUPERVISOR LINKAGE (XCTL). ENTRY IS ALWAYS DIRECTLY FROM THE DISPATCHER
B1 CLOSE RECURSION
YES
NO
C1 ABDUMP RECURSION
NO
YES
D1 RESET ABDUMP BIT. SET PREVENT DUMP BIT IN JOB STEP TCB
SKIPDSET
E1 IS PREV DUMP INDR ON IN JOB-STEP TCB
YES
NO
G3
F1 IS DCBREG = 0.
YES
NO
G3
G1 SET 'ABDUMP' AND 'RECURSION' BITS IN CURRENT TCB
ABD12 IGCCC8(S)
H1 CONT. SUPVSN *CCA4 LOAD ABDUMP RESIDENT MODULE
IGC056(S)
J1 ENQUEUE RTN *EOA1 ENQUEUE SYSABEND DATA SET
K1 IS PREVENT DUMP BIT SET IN JS TCB
YES
NO
D3
A3
CLOSEREC
C2 LOCATE PREVIOUS ABEND SUPERVISOR REQUEST BLOCK (SVRB)
D2 EXTRACT DEB ADDRESS
E2 REMOVE AND FREE ALL TRANSIENT SVRB'S EXCEPT ABEND'S SVRB
HP F2
A3
IGC0005A(S)
A3 ABDUMP RTN HHA1 DISPLAY CURRENT TASK'S RESOURCES
IGC0005A(S)
B3 ABDUMP RTN HHA1 DISPLAY DESCENDENT TASKS RESOURCES
IGC0005A(S)
C3 ABDUMP RTN HHA1 DISPLAY ANCESTOR TASKS RESOURCES
D3
ABD21 IGC048(S)
D3 DEQUEUE RTN *BPA1 DEQUEUE DUMP DATA SET
IGC009(S)
E3 DELETE RTN CEA2 DELETE ABDUMP RESIDENT MODULE
F3 RESET 'ABDUMP' AND 'RECURSION' BITS IN CURRENT TCB
G3
INITSEL
G3 DOES TCB REPRESENT TOP TASK
YES
A5
NO
H3 LOAD ORIGIN FIELD OF TCB
G3
IEAQERA
C4 ERASE PHASE RTN REMOVE TCB FROM FAMILY TREE, FREE TCB
A5
FINDLOW
A5 GET ADDRESS OF LOWEST SUBTASK IN ABEND FAMILY
B5 SELECT22 SELECT A TASK
C5 IS SELECTED TASK COMPLETE
YES
NO
D5 CURRENT TCB SELECTED
YES
NO
HP B2
FROM HPF4
HP E5
SETWRND
E5 SET CURRENT TCB NON-DISPATCHABLE, SELECTED TCB DISPATCHABLE
F5 STORE ADDRESS OF ENTRY2 (HPB2) IN ABEND'S SVRB OLD PSW FLD
G5 MOVE ABEND'S SVRB TO SELECTED TCB'S RB QUEUE
H5 STORE REGISTERS IN SELECTED TCB
IEAODS02
J5 TASK SW. RTN *BVA2 INDICATE TASK SWITCH IF NECESSARY
DISP
K5 EXIT
TO DISPATCHER (IEAODS) CHART GG
NOTE - RE-ENTRY TO AEEND5 FOR THE SELECTED TASK WILL OCCUR AT ENTRY2 -CHART HP,-BLOCK B2.

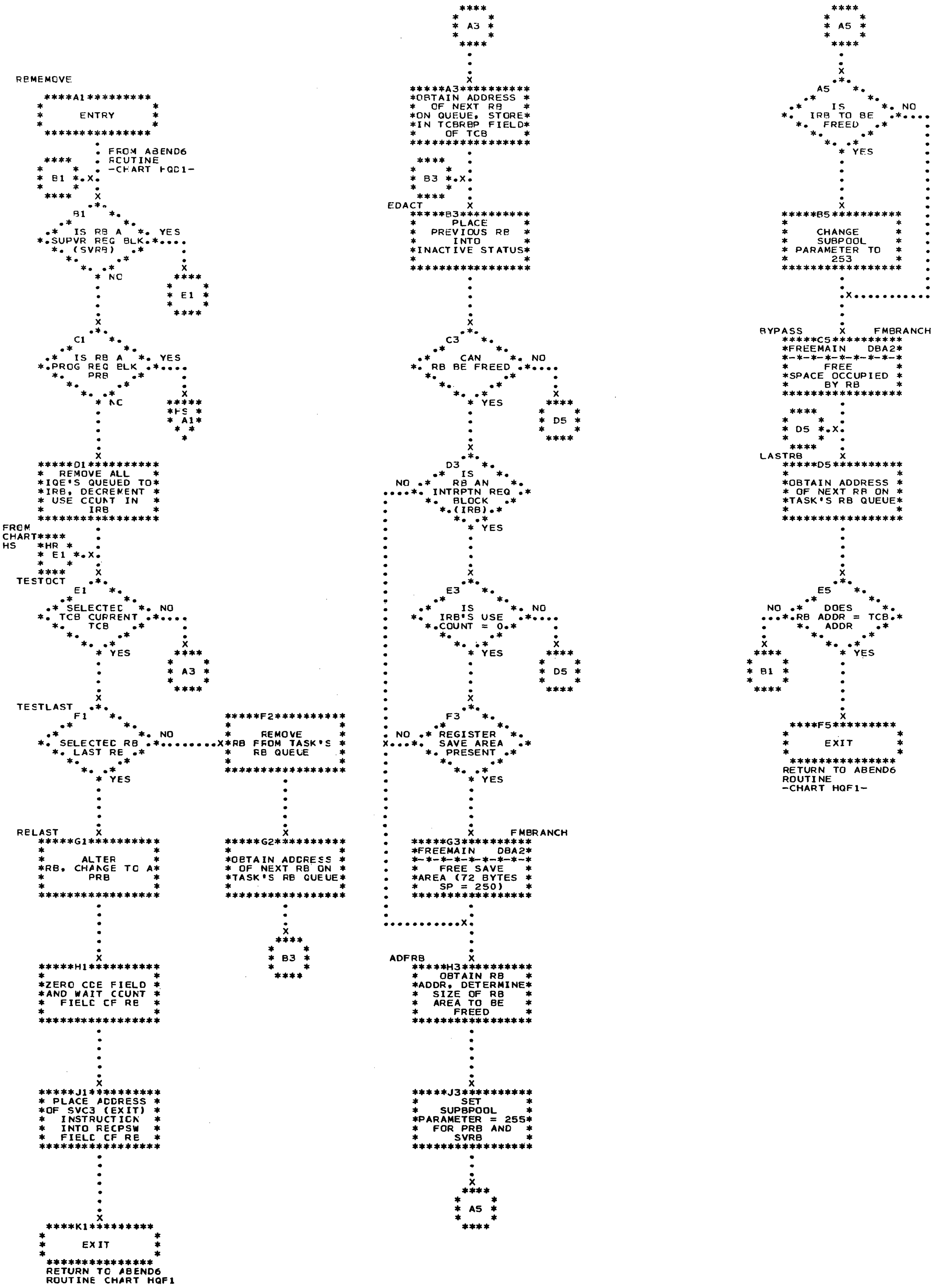
• **Chart HP. ABEND5 (Part 2 of 2)**


HP B2
FROM HOD5
ENTRY2 B2
ANY DEBS QUEUED TO TCB
NO
YES
TRANTEST B3
LOAD RB LINK FIELD OF ABEND SVRB
C3
TRANTS01 C3
LAST RB ON QUEUE
YES
NO
C2
SAVE DEB IN EXTENDED SAVE AREA OF ABEND SVRB
D2
SET CLOSE AND RECURSION BITS IN TCB
D3
IS IT A TRANSIENT SVRB
NO
YES
TOPTEST D4
DOES TCB REPRESENT TOP TASK
YES
NO
D5
EXIT
TO ABEND6 ROUTINE-(IGC0301C) , CHART HQ, VIA SUPERVISOR LINKAGE (XCTL)
E2
CLOSE RTN
CLOSE DATA SETS BELONGING TO TASK
E3
REMOVE RB FROM RB QUEUE
E4
E4
TASKSEL3
SELECT NEXT HIGHEST TASK
FROM HOE2
HP F2
DEBFREED
F2
RESET CLOSE AND RECURSION BITS IN TCB
F3
TAHABEND
PURGE TRANSIENT AREA QUEUES
F4
IS SELECTED TASK COMPLETE
NO
YES
HO E5
G2
HAS DEB BEEN FREED
YES
NO
B2
G3
GET NEXT RB ON QUEUE
C3
IEAQERA
G4
ERASE PHASE RTN
REMOVE TCB FROM SUBTASK QUEUES. FREE TCB SPACE
E4
H2
REMOVE AND FREE DEB
B2

• Chart HQ. ABEND6

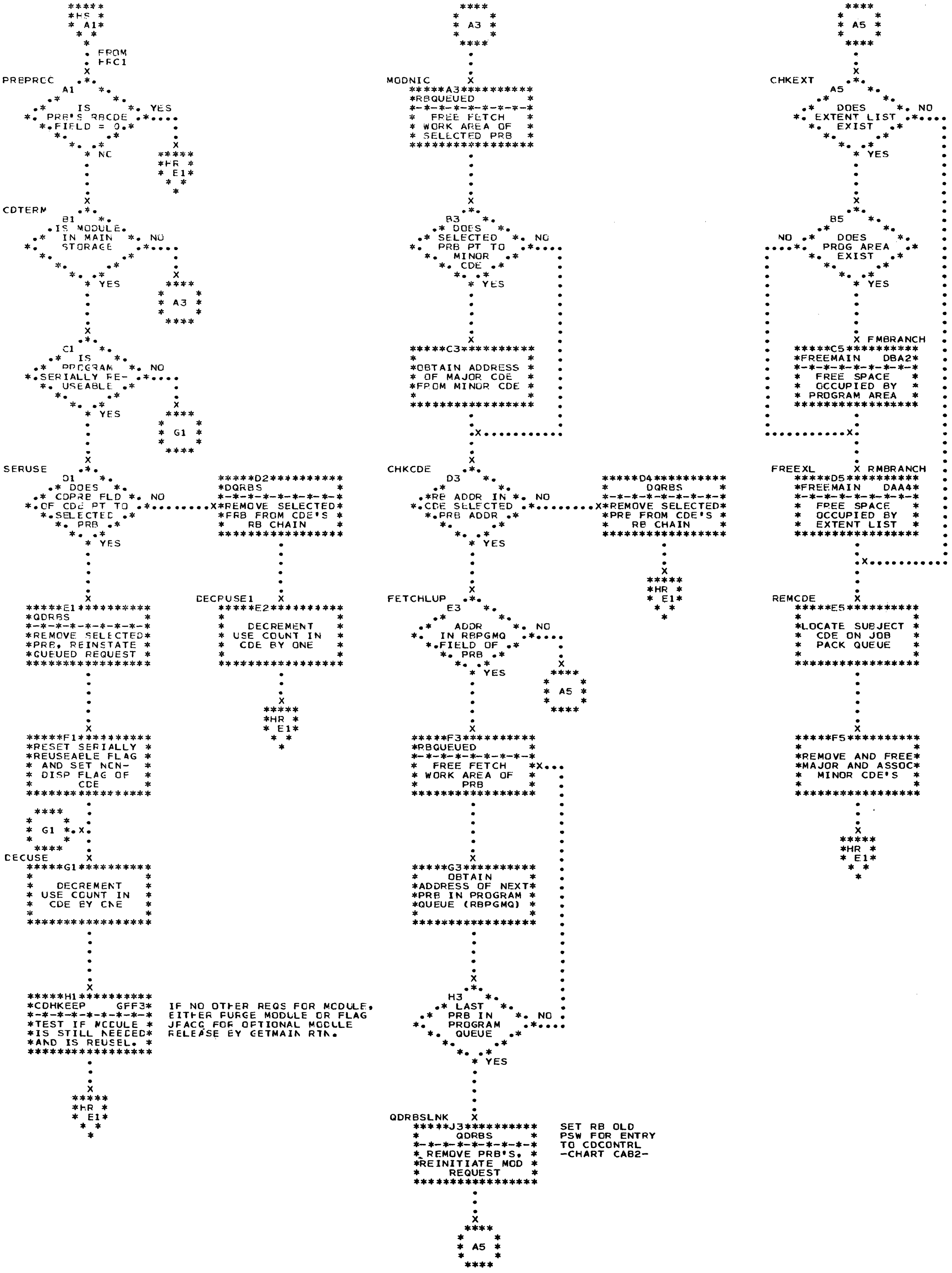

A2
IGC0301C
A1
ENTRY
FRCM AEENCS -CFART FFC5- VIA SUPERVISOR LINKAGE (XCTL). ENTRY IS ALWAYS CIRECTLY FRCM THE CISPATCFER.
B1
LOCP
B1
TASKSEL
SELECT A TASK
C1
IS CURRENT TCB EQUAL TO SELECTEC TCE
YES
NC
D1
GET FIRST RE CN RE QUEUE
CLEARHO
E1
RBREMOVE HRA1
PURGE RB'S, CDE'S, AND IQE'S
ENDQPURGE
IEAQEGC1
F1
ENG/DEG FURGE
PURGE QCE CFAIN
LOADREL
IEAGABL
G1
RLSE LOADED PGM
FCA2
RELEASE LLE'S, FREE MCDULES, IF PCSSIELE
PURGEMSS
IEAQSPET
H1
RLSE MAIN STCRG
FCA2
FREE SPACE AND SPQE'S
A2
A2
OPERATING ON CURRENT TCB
YES
NO
B2
SET CCMPLETION FLAG IN TCB
DGTCB
IEADQTCB
C2
DEQUEUE TCB RTN
REMOVE TCB FROM TCB QUEUE
IEAQERA
D2
EFASE PHASE FTN
REMOVE TCB FROM FAMILY TREE, FREE TCB
B1
RETURN
A3
MOVE COMP CODE FROM TCB TO RETURN REGISTER
B3
EXIT
TO EXIT ROUTINE (IGC003) CHART GB VIA SUPERVISOR LINKAGE (SVC 3)

• Chart HR. ABEND6-RBREMOVE Subroutine (Part 1 of 2)



RBMEMOVE
A1
ENTRY
FROM ABEND6 RCUTINE -CHART HQD1-
B1
B1
IS RB A SUPVR REG BLK (SVRB)
YES
NO
E1
C1
IS RB A PROG REQ BLK PRB
YES
NC
HS A1
D1
REMOVE ALL IQE'S QUEUED TO IRB, DECREMENT USE CCUNT IN IRB
FROM CHART HS
HR E1
TESTOCT
E1
SELECTEC TCB CURRENT TCB
NO
YES
A3
TESTLAST
F1
SELECTEC RB LAST RE
NO
YES
F2
REMOVE RB FROM TASK'S RB QUEUE
G2
OBTAIN ADDRESS OF NEXT RB ON TASK'S RB QUEUE
B3
RELAST
G1
ALTER RB, CHANGE TO A PRB
H1
ZERO CDE FIELD AND WAIT CCUNT FIELC CF RB
J1
PLACE ADDRESS OF SVC3 (EXIT) INSTRUCTICN INTO RECPSW FIELC CF RB
K1
EXIT
RETURN TO ABEND6 ROUTINE CHART HQF1
A3
A3
OBTAIN ADDRESS OF NEXT RB ON QUEUE, STORE IN TCBRBP FIELD OF TCB
B3
EDACT
B3
PLACE PREVIOUS RB INTO INACTIVE STATUS
C3
CAN RB BE FREED
NO
D5
YES
D3
IS RB AN INTRPTN REQ BLOCK (IRB)
NO
YES
E3
IS IRB'S USE COUNT = 0
NO
D5
YES
F3
REGISTER SAVE AREA PRESENT
NO
YES
FMBRANCH
G3
FREEMAIN DBA2
FREE SAVE AREA (72 BYTES SP = 250)
ADFRB
H3
OBTAIN RB ADDR, DETERMINE SIZE OF RB AREA TO BE FREED
J3
SET SUPBPOOL PARAMETER = 255 FOR PRB AND SVRB
A5
A5
A5
IS IRB TO BE FREED
NO
YES
B5
CHANGE SUBPOOL PARAMETER TO 253
BYPASS
FMBRANCH
C5
FREEMAIN DBA2
FREE SPACE OCCUPIED BY RB
D5
LASTRB
D5
OBTAIN ADDRESS OF NEXT RB ON TASK'S RB QUEUE
E5
DOES RB ADDR = TCB ADDR
NO
B1
YES
F5
EXIT
RETURN TO ABEND6 ROUTINE -CHART HQF1-

• Chart HS. ABEND6-RBREMOVE Subroutine (Part 2 of 2)


HS A1
FROM FFC1
PREPROC A1
IS PRB'S RBCDE FIELD = 0
YES
NO
HR E1
CDTERM B1
IS MODULE IN MAIN STORAGE
NO
YES
A3
C1
IS PROGRAM SERIALLY RE-USEABLE
NO
YES
G1
SERUSE D1
DOES CDPRB FLD OF CDE PT TO SELECTED PRB
NO
D2
DQRBS
REMOVE SELECTED PRB FROM CDE'S RB CHAIN
YES
E1
QDRBS
REMOVE SELECTED PRB, REINSTATE QUEUED REQUEST
DECPUSE1 E2
DECREMENT USE COUNT IN CDE BY ONE
HR E1
F1
RESET SERIALLY REUSEABLE FLAG AND SET NON-DISP FLAG OF CDE
G1
DECUSE G1
DECREMENT USE COUNT IN CDE BY ONE
H1
CDHKEEP GFF3
TEST IF MODULE IS STILL NEEDED AND IS REUSEL.
IF NO OTHER REQS FOR MODULE, EITHER PURGE MODULE OR FLAG JFACG FOR OPTIONAL MODULE RELEASE BY GETMAIN RTN.
HR E1
A3
MODNIC A3
RBQUEUED
FREE FETCH WORK AREA OF SELECTED PRB
B3
DOES SELECTED PRB PT TO MINOR CDE
NO
YES
C3
OBTAIN ADDRESS OF MAJOR CDE FROM MINOR CDE
CHKCDE D3
RB ADDR IN CDE SELECTED PRB ADDR
NO
D4
DQRBS
REMOVE SELECTED PRB FROM CDE'S RB CHAIN
HR E1
YES
FETCHLUP E3
ADDR IN RBPGMQ FIELD OF PRB
NO
A5
YES
F3
RBQUEUED
FREE FETCH WORK AREA OF PRB
G3
OBTAIN ADDRESS OF NEXT PRB IN PROGRAM QUEUE (RBPGMQ)
H3
LAST PRB IN PROGRAM QUEUE
NO
YES
QDRBSLNK J3
QDRBS
REMOVE PRB'S, REINITIATE MOD REQUEST
SET RB OLD PSW FOR ENTRY TO CDCONTRL -CHART CAB2-
A5
A5
CHKEXT A5
DOES EXTENT LIST EXIST
NO
YES
B5
DOES PROG AREA EXIST
NO
YES
FMBRANCH C5
FREEMAIN DBA2
FREE SPACE OCCUPIED BY PROGRAM AREA
FREEXL D5 RMBRANCH
FREEMAIN DAA4
FREE SPACE OCCUPIED BY EXTENT LIST
REMCDE E5
LOCATE SUBJECT CDE ON JOB PACK QUEUE
F5
REMOVE AND FREE MAJOR AND ASSOC MINOR CDE'S
HR E1

• Chart HT. System Quiesce Routine

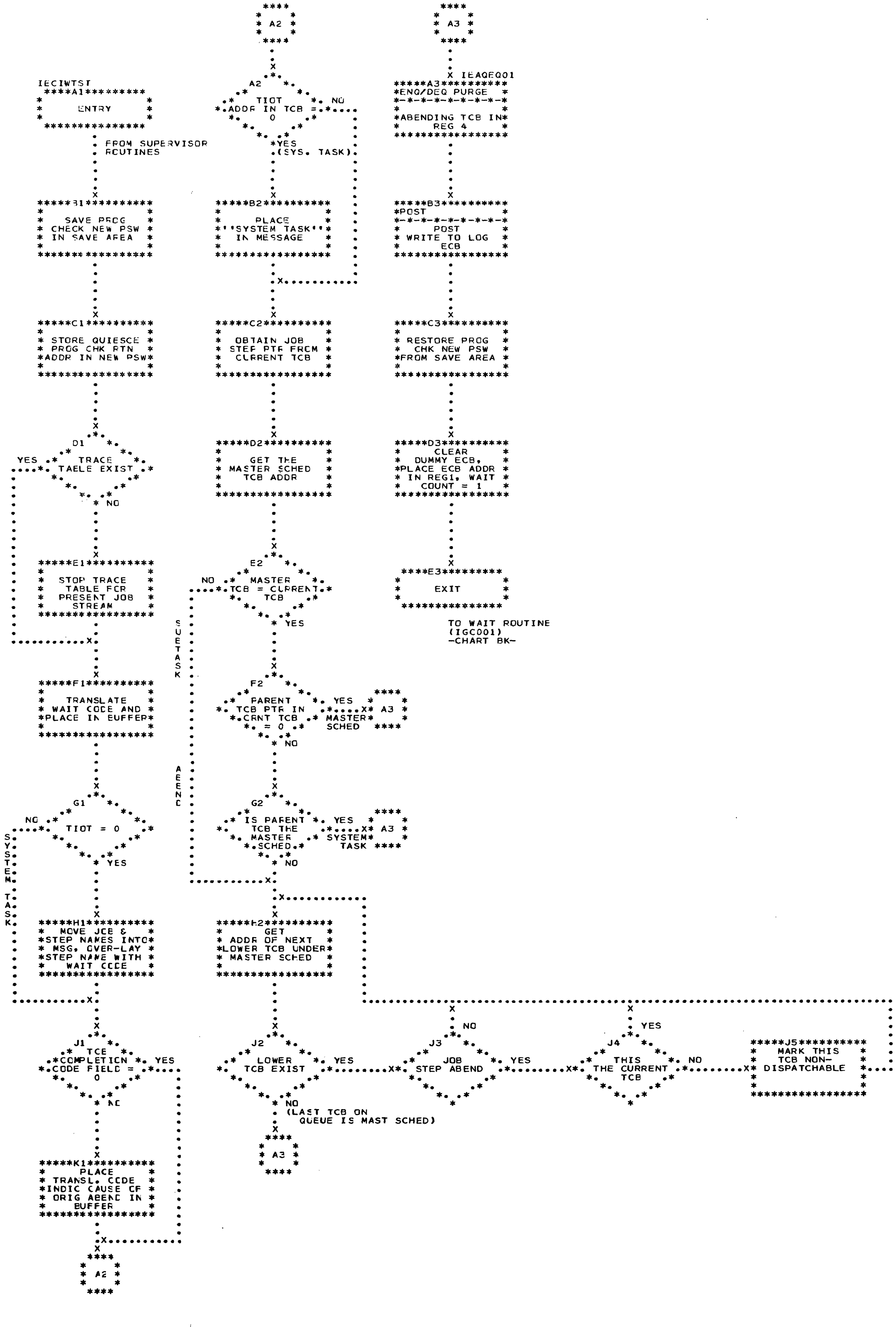

A2
A3
IECIWTST
A1
ENTRY
FROM SUPERVISOR ROUTINES
A2
TIOT ADDR IN TCB = 0
NO
YES (SYS. TASK)
IEAQEQ01
A3
ENQ/DEQ PURGE
ABENDING TCB IN REG 4
B1
SAVE PROG CHECK NEW PSW IN SAVE AREA
B2
PLACE ''SYSTEM TASK'' IN MESSAGE
B3
POST
POST WRITE TO LOG ECB
C1
STORE QUIESCE PROG CHK RTN ADDR IN NEW PSW
C2
OBTAIN JOB STEP PTR FRCM CLRRENT TCB
C3
RESTORE PROG CHK NEW PSW FROM SAVE AREA
D1
TRACE TAELE EXIST
YES
NO
D2
GET THE MASTER SCHED TCB ADDR
D3
CLEAR DUMMY ECB, PLACE ECB ADDR IN REG1, WAIT COUNT = 1
E1
STOP TRACE TABLE FCR PRESENT JOB STREAM
E2
MASTER TCB = CLRRENT TCB
NO
YES
E3
EXIT
TO WAIT ROUTINE (IGC001) -CHART BK-
SUBTASK
F1
TRANSLATE WAIT CODE AND PLACE IN BUFFER
F2
PARENT TCB PTR IN CRNT TCB = 0
YES
A3
MASTER SCHED
NO
ABEND
G1
TIOT = 0
NO
YES
SYSTEM TASK
G2
IS PARENT TCB THE MASTER SCHED
YES
A3
SYSTEM TASK
NO
H1
MOVE JOB & STEP NAMES INTO MSG. OVER-LAY STEP NAME WITH WAIT CODE
H2
GET ADDR OF NEXT LOWER TCB UNDER MASTER SCHED
J1
TCB COMPLETION CODE FIELD = 0
YES
NO
J2
LOWER TCB EXIST
YES
NO (LAST TCB ON QUEUE IS MAST SCHED)
A3
J3
JOB STEP ABEND
NO
YES
J4
THIS THE CURRENT TCB
YES
NO
J5
MARK THIS TCB NON-DISPATCHABLE
K1
PLACE TRANSL. CODE INDIC CAUSE OF ORIG ABEND IN BUFFER
A2

Chart IA. 2250 Processor Routine (Part 1 of 4)

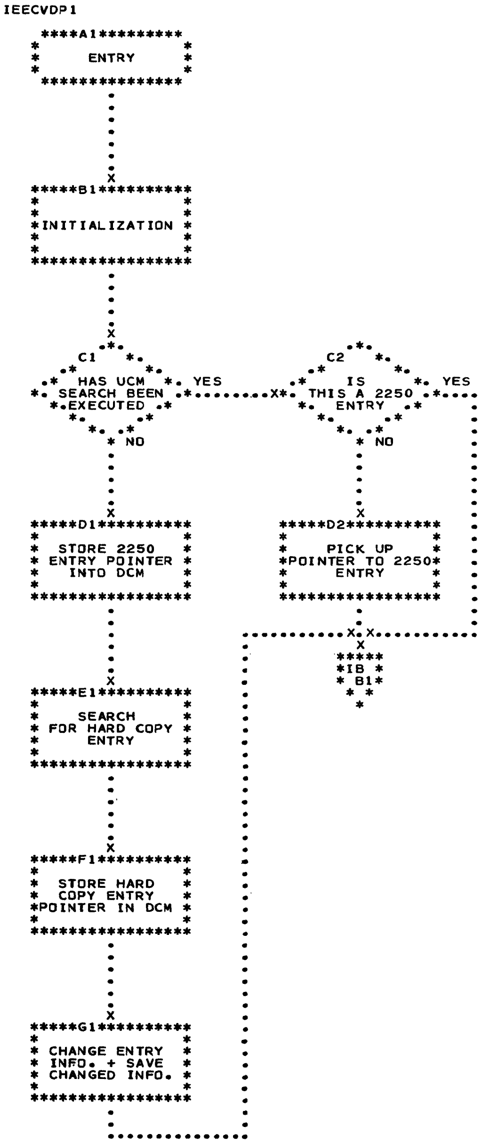

IEECVDP1
A1
ENTRY
B1
INITIALIZATION
C1
HAS UCM SEARCH BEEN EXECUTED
YES
NO
C2
IS THIS A 2250 ENTRY
YES
NO
D1
STORE 2250 ENTRY POINTER INTO DCM
D2
PICK UP POINTER TO 2250 ENTRY
IB B1
E1
SEARCH FOR HARD COPY ENTRY
F1
STORE HARD COPY ENTRY POINTER IN DCM
G1
CHANGE ENTRY INFO. + SAVE CHANGED INFO.

Chart IB. 2250 Processor Routine (Part 2 of 4)

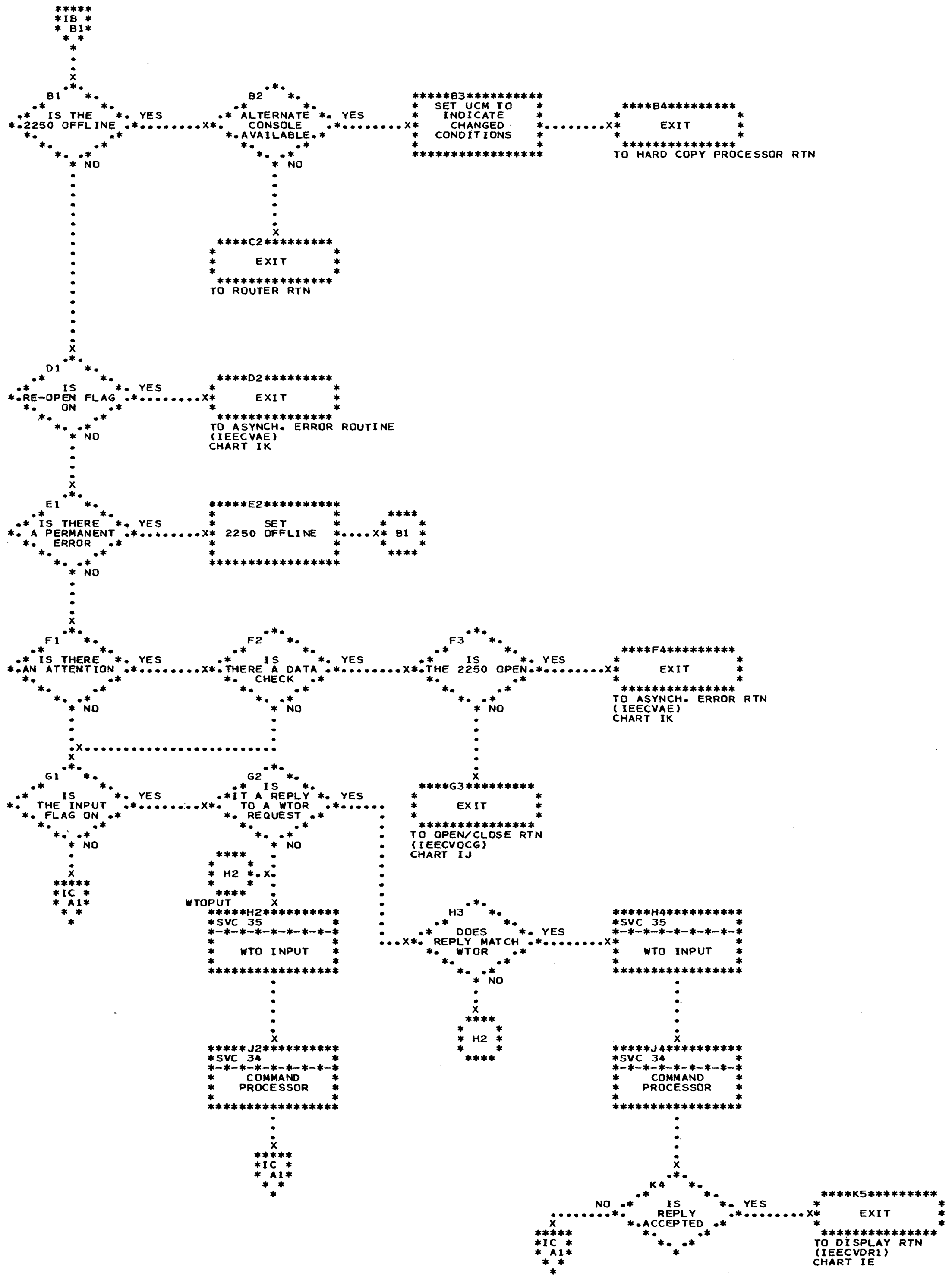
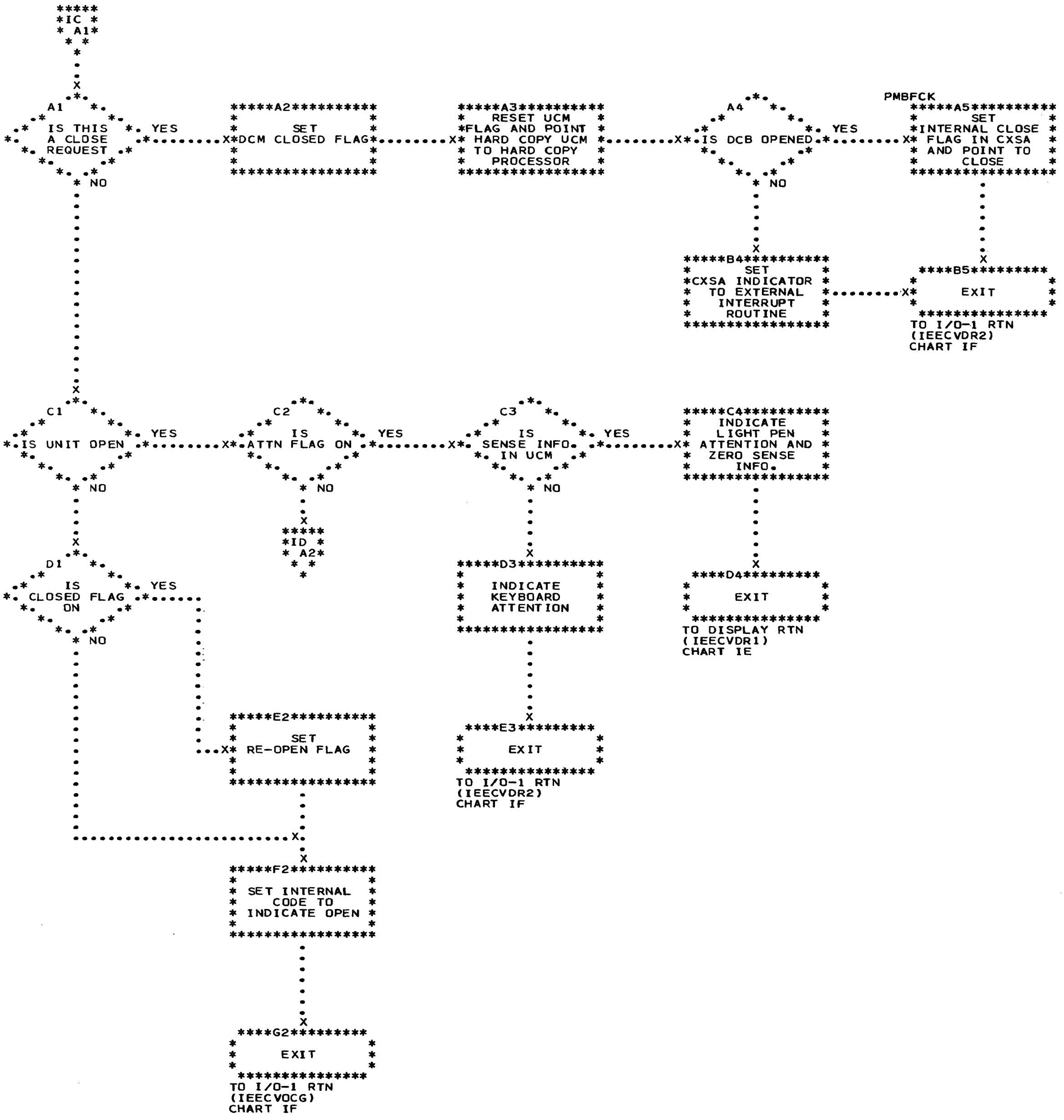


Chart IC. 2250 Processor Routine (Part 3 of 4)


IC A1
A1 IS THIS A CLOSE REQUEST
YES
NO
A2 SET DCM CLOSED FLAG
A3 RESET UCM FLAG AND POINT HARD COPY UCM TO HARD COPY PROCESSOR
A4 IS DCB OPENED
YES
NO
PMBFCK
A5 SET INTERNAL CLOSE FLAG IN CXSA AND POINT TO CLOSE
B4 SET CXSA INDICATOR TO EXTERNAL INTERRUPT ROUTINE
B5 EXIT
TO I/O-1 RTN (IEECVDR2) CHART IF
C1 IS UNIT OPEN
YES
NO
C2 IS ATTN FLAG ON
YES
NO
ID A2
C3 IS SENSE INFO. IN UCM
YES
NO
C4 INDICATE LIGHT PEN ATTENTION AND ZERO SENSE INFO.
D1 IS CLOSED FLAG ON
YES
NO
D3 INDICATE KEYBOARD ATTENTION
D4 EXIT
TO DISPLAY RTN (IEECVDR1) CHART IE
E2 SET RE-OPEN FLAG
E3 EXIT
TO I/O-1 RTN (IEECVDR2) CHART IF
F2 SET INTERNAL CODE TO INDICATE OPEN
G2 EXIT
TO I/O-1 RTN (IEECVOCG) CHART IF

• Chart ID. 2250 Processor Routine (Part 4 of 4)

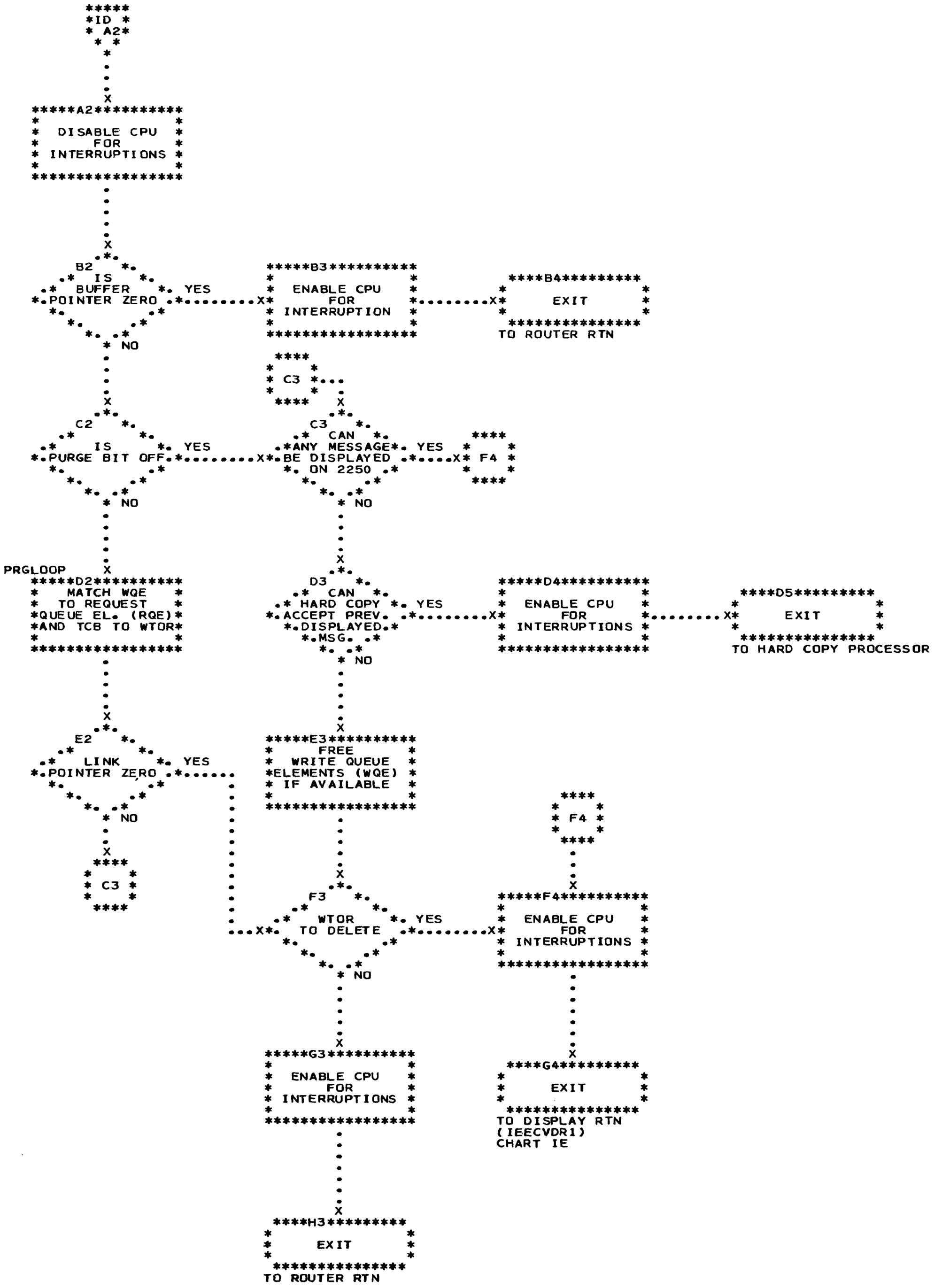

ID A2
A2 DISABLE CPU FOR INTERRUPTIONS
B2 IS BUFFER POINTER ZERO
YES
B3 ENABLE CPU FOR INTERRUPTION
B4 EXIT
TO ROUTER RTN
NO
C3
C2 IS PURGE BIT OFF
YES
C3 CAN ANY MESSAGE BE DISPLAYED ON 2250
YES
F4
NO
NO
PRGLOOP
D2 MATCH WQE TO REQUEST QUEUE EL. (RQE) AND TCB TO WTOR
D3 CAN HARD COPY ACCEPT PREV. DISPLAYED MSG.
YES
D4 ENABLE CPU FOR INTERRUPTIONS
D5 EXIT
TO HARD COPY PROCESSOR
NO
E2 LINK POINTER ZERO
YES
E3 FREE WRITE QUEUE ELEMENTS (WQE) IF AVAILABLE
NO
C3
F4
F3 WTOR TO DELETE
YES
F4 ENABLE CPU FOR INTERRUPTIONS
NO
G3 ENABLE CPU FOR INTERRUPTIONS
G4 EXIT
TO DISPLAY RTN (IEECVDR1) CHART IE
H3 EXIT
TO ROUTER RTN

Chart IE. Display Routine

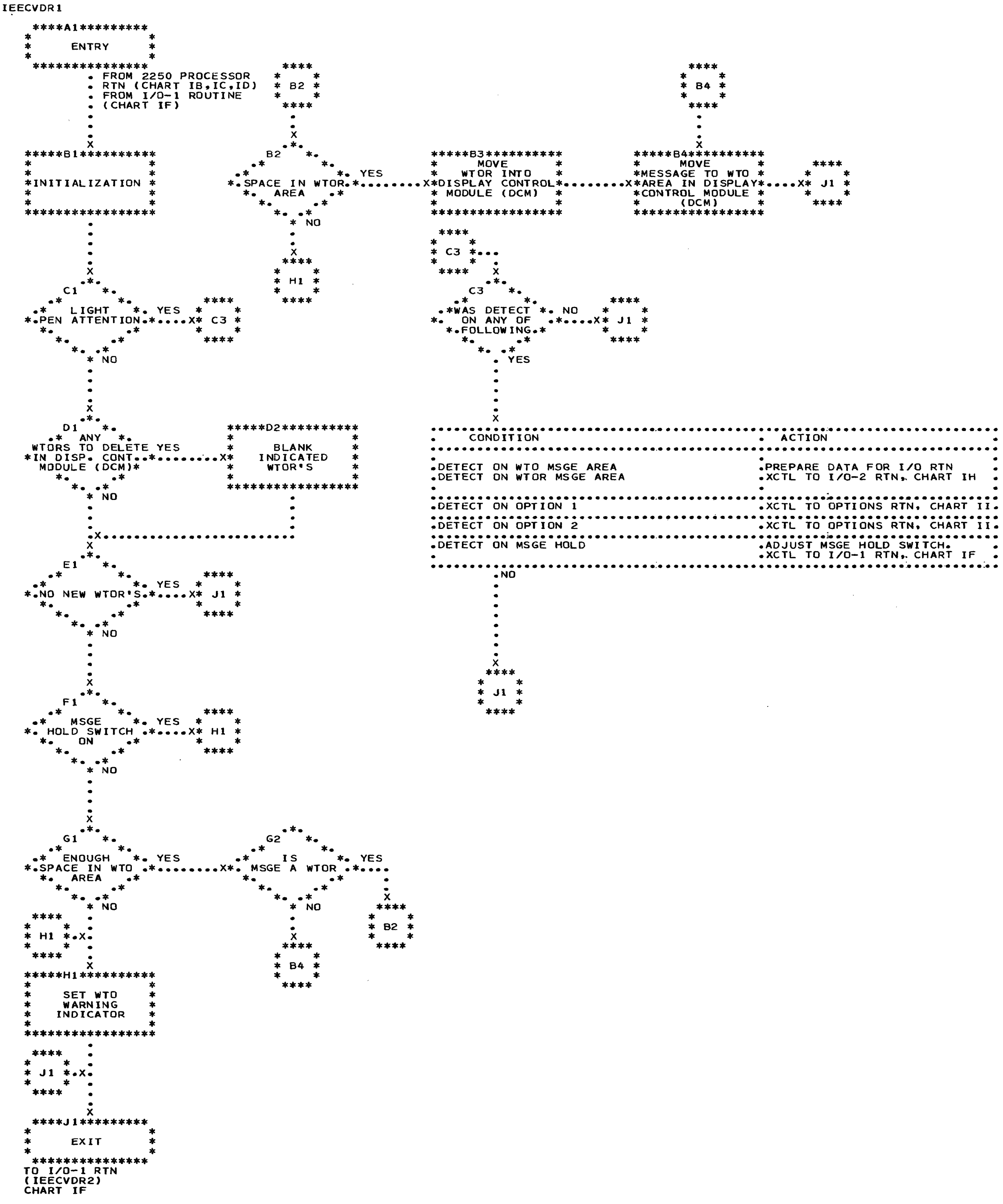
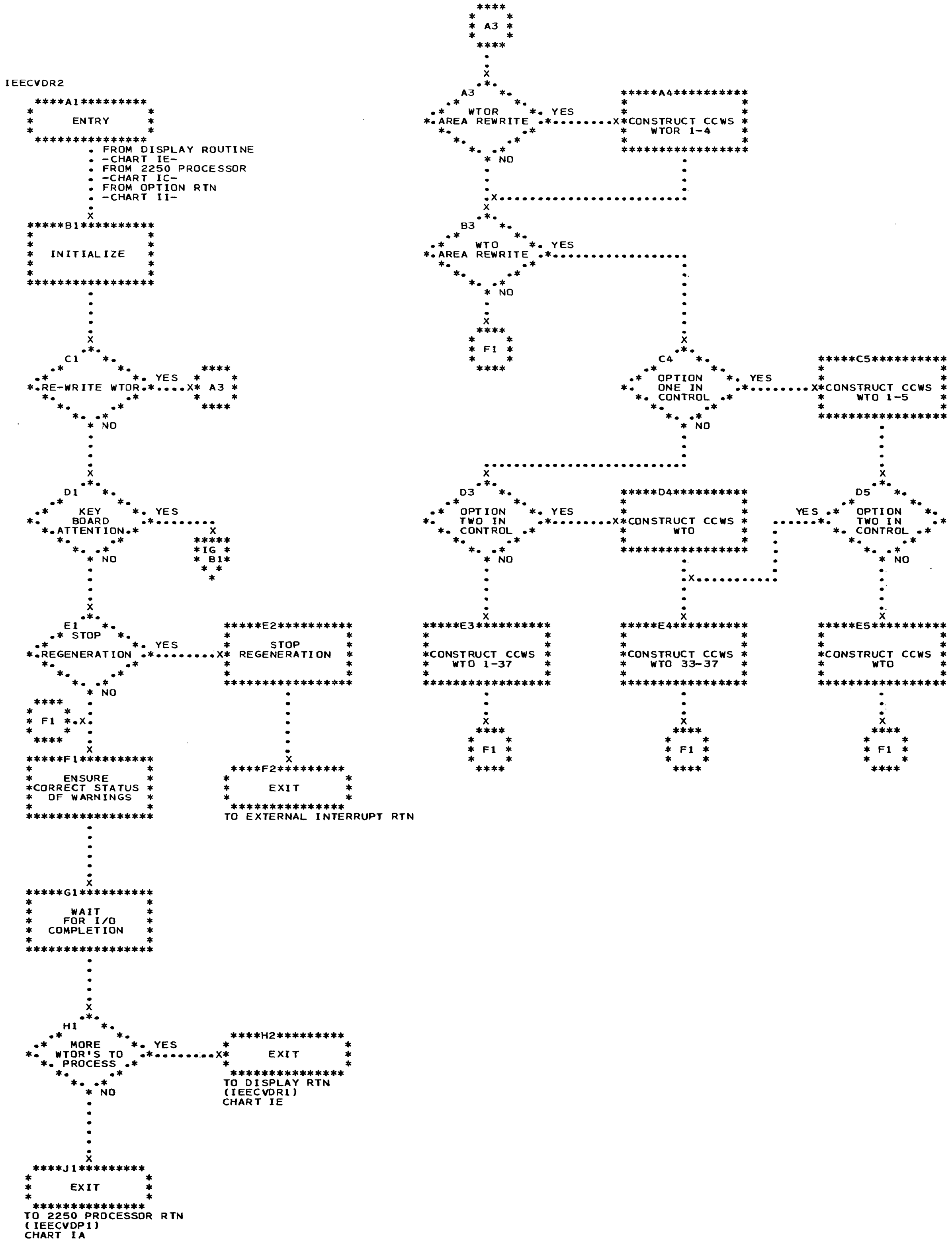


Chart IF. I/O-1 Routine (Part 1 of 2)



• Chart IG. I/O-1 Routine (Part 2 of 2)

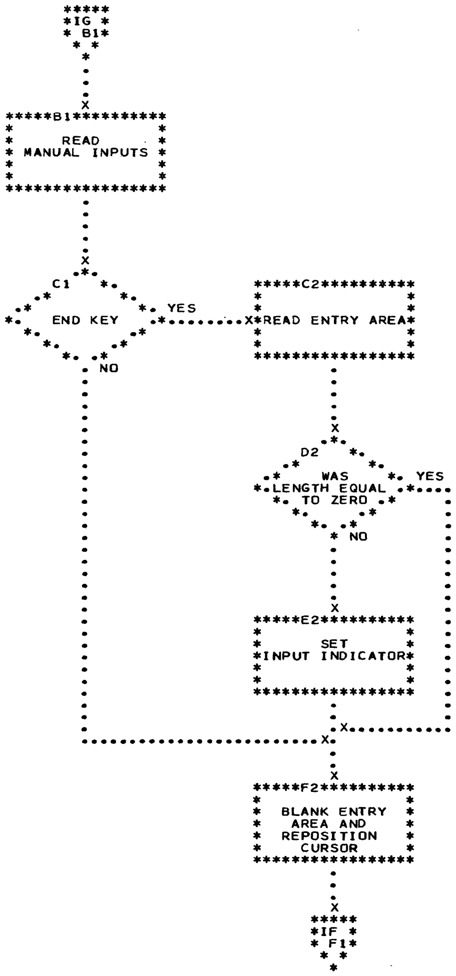
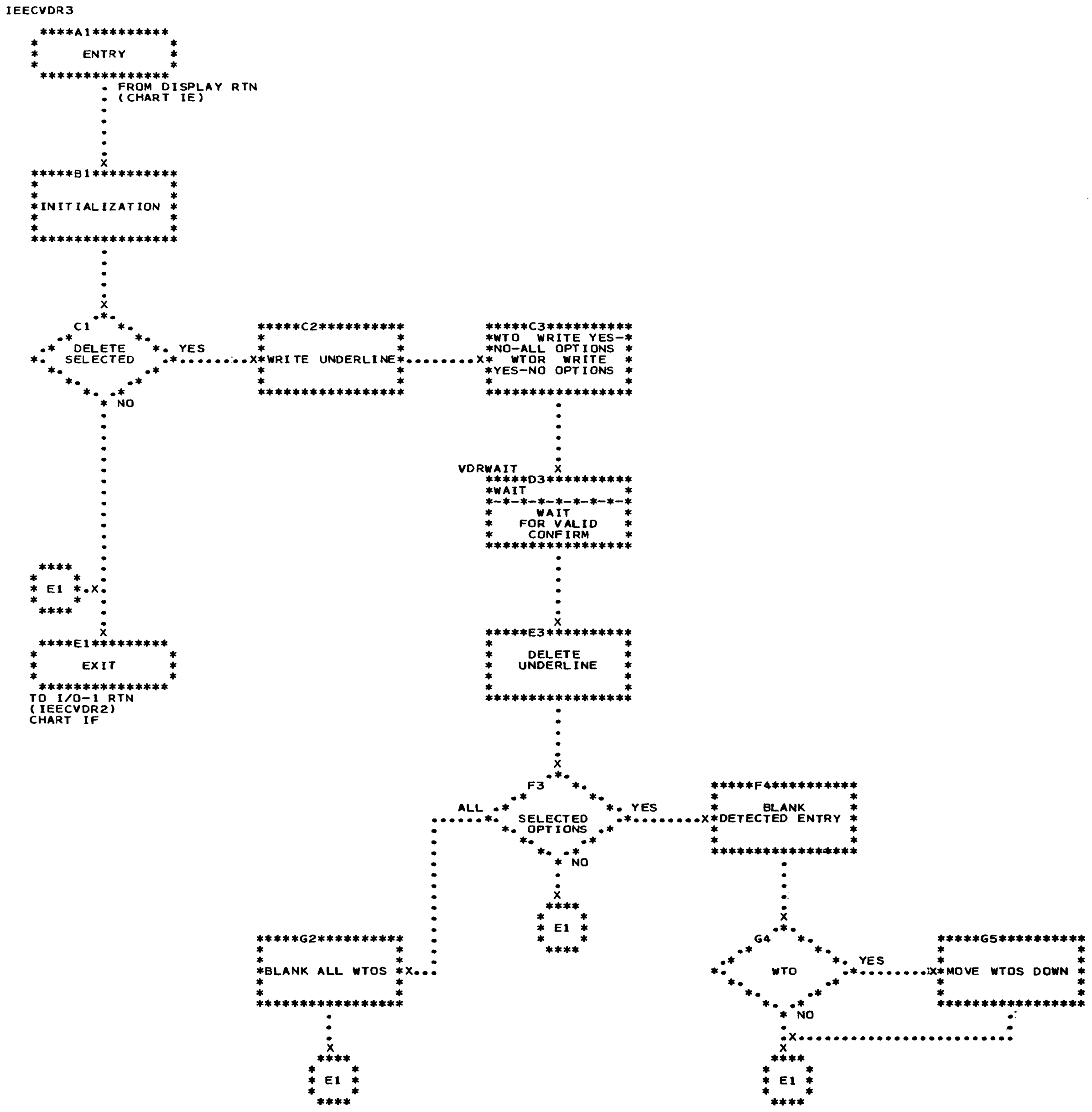


Chart IH. I/O-2 Routine



• Chart II. Option Routine

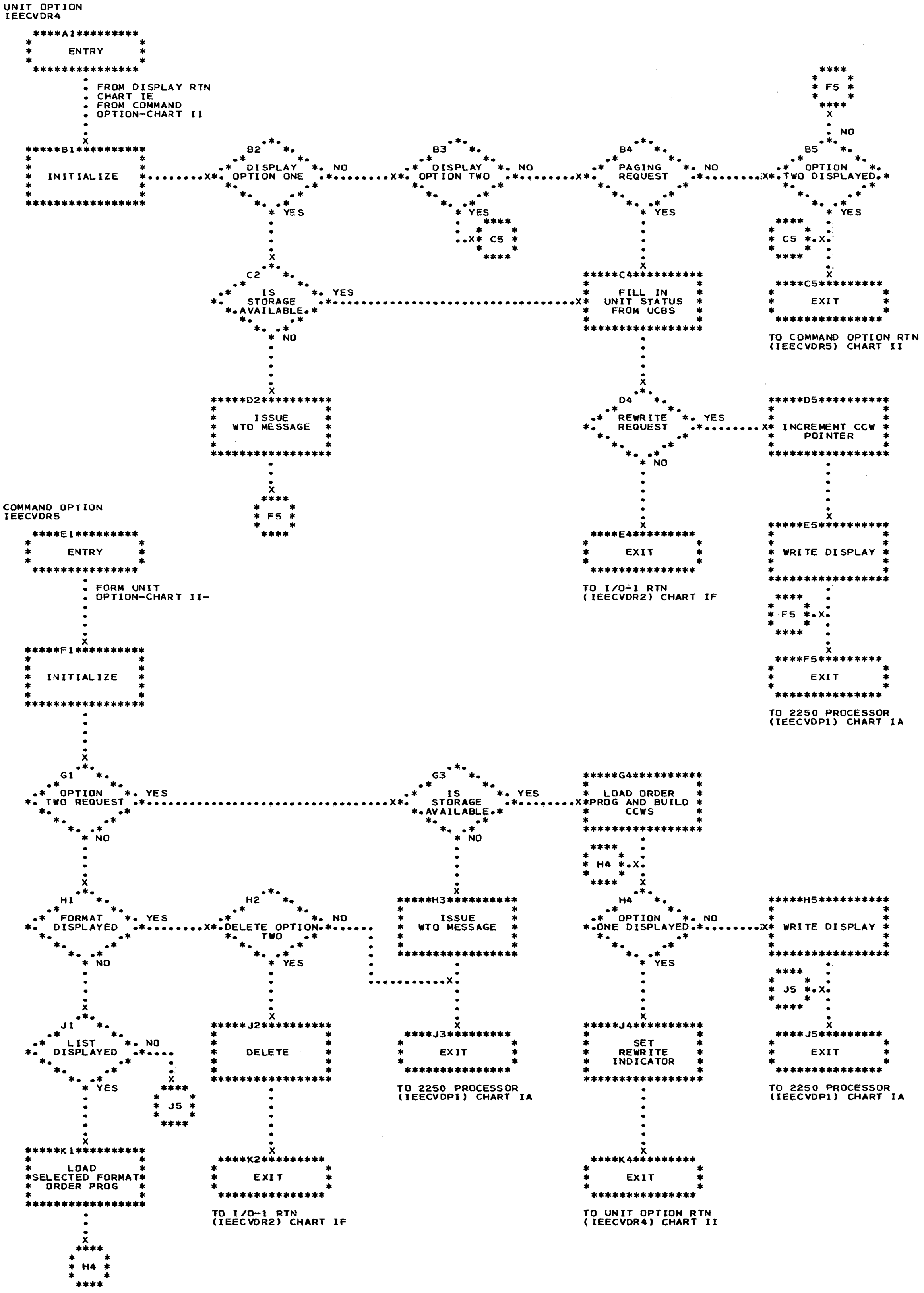


Chart IJ. Open/Close Routine

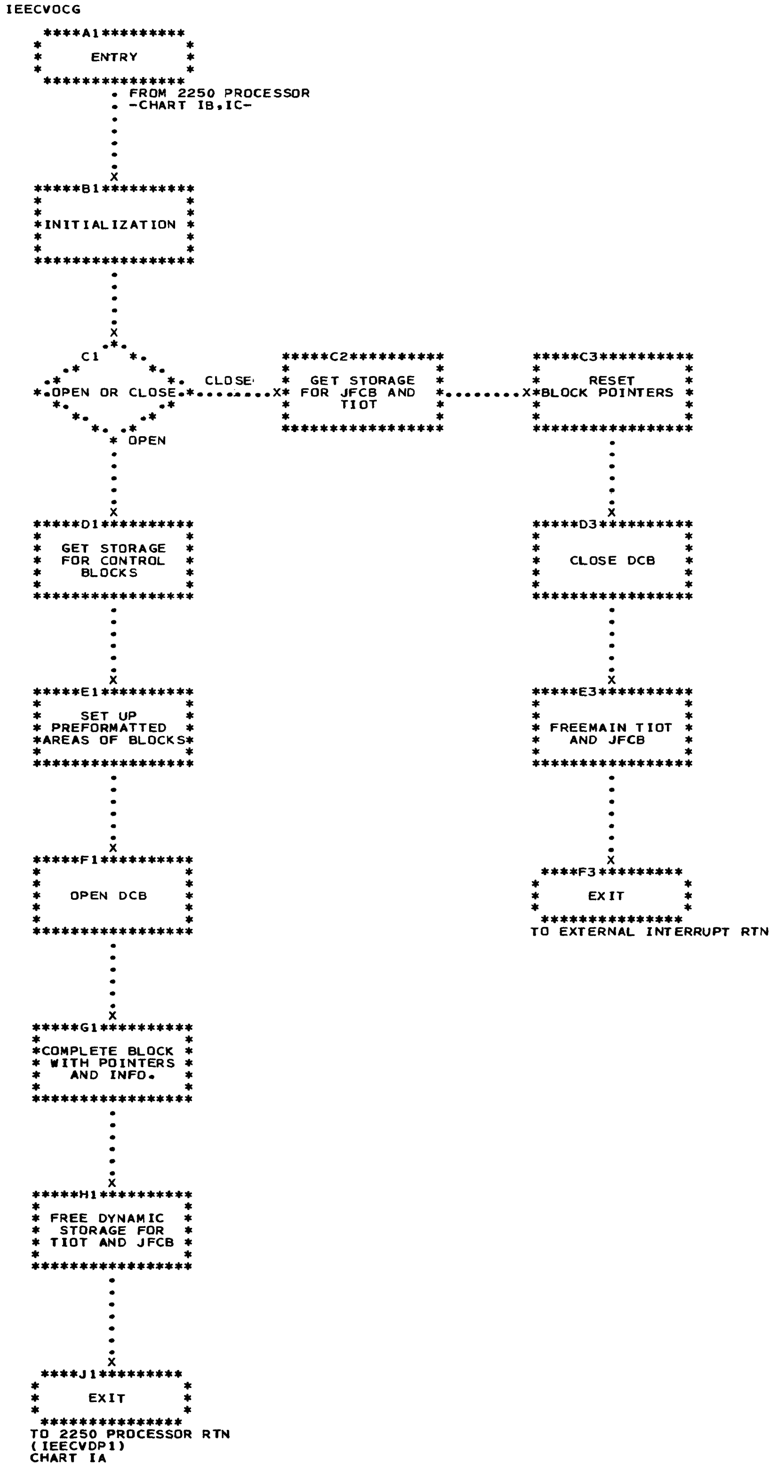
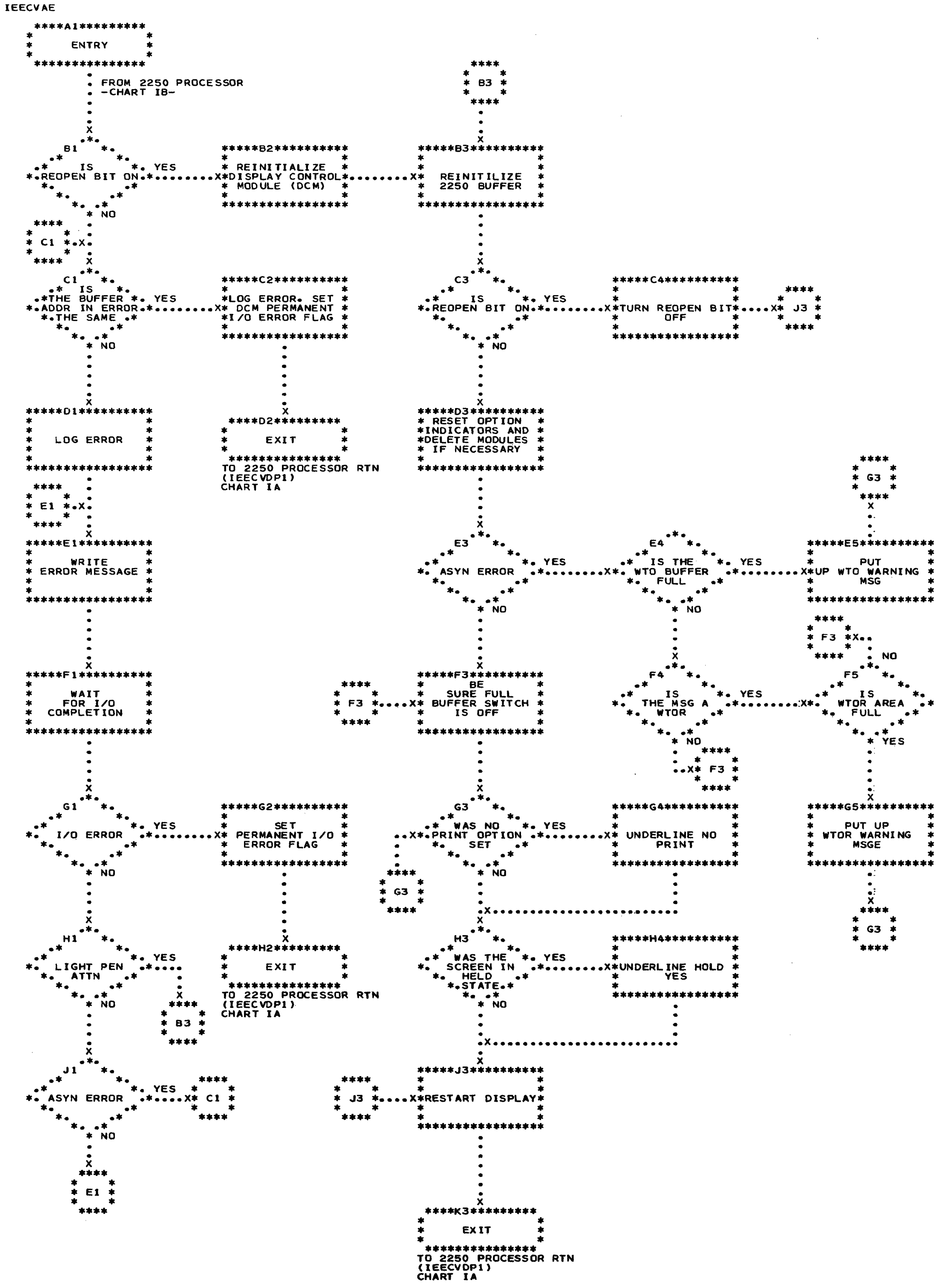
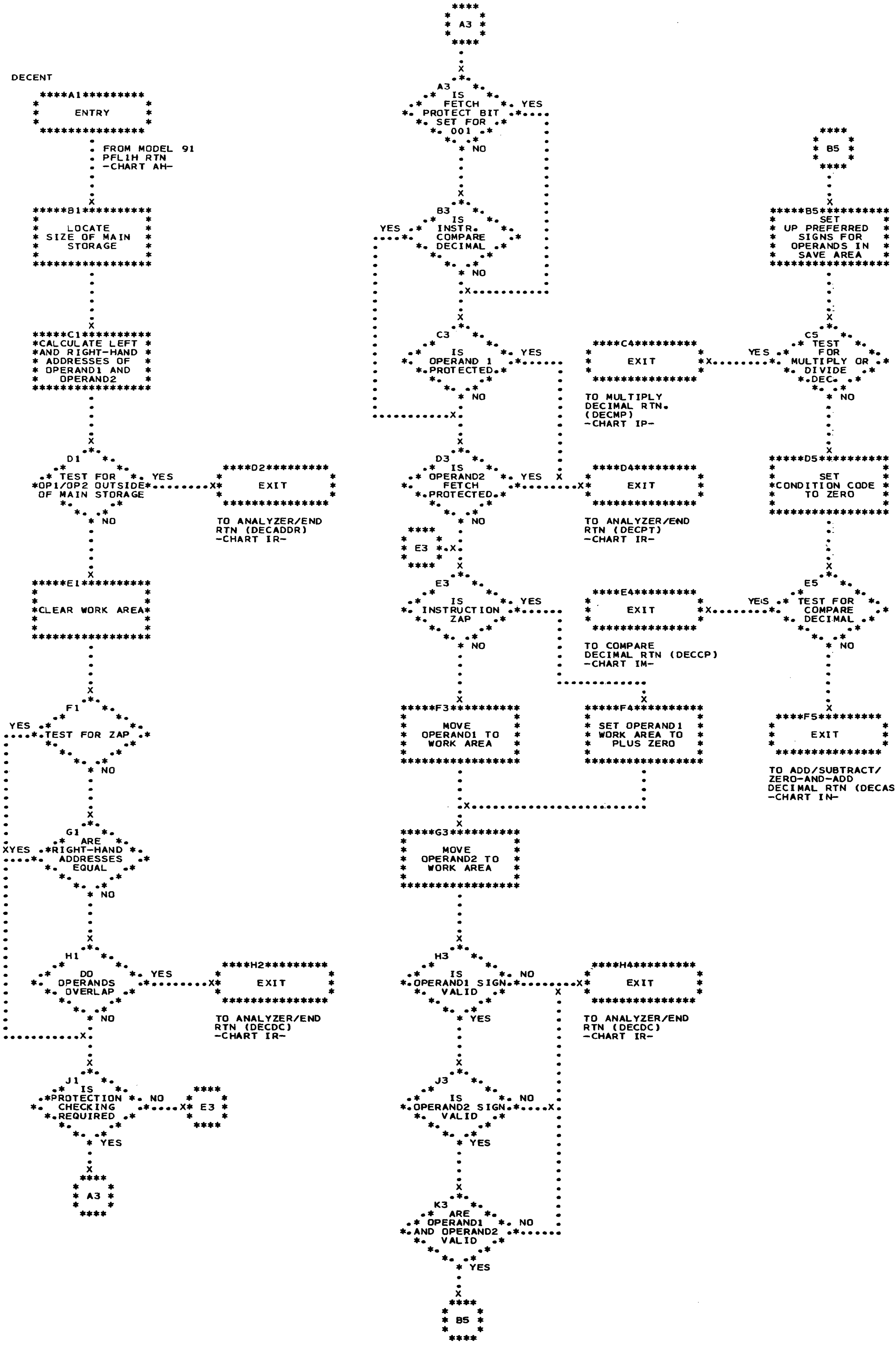


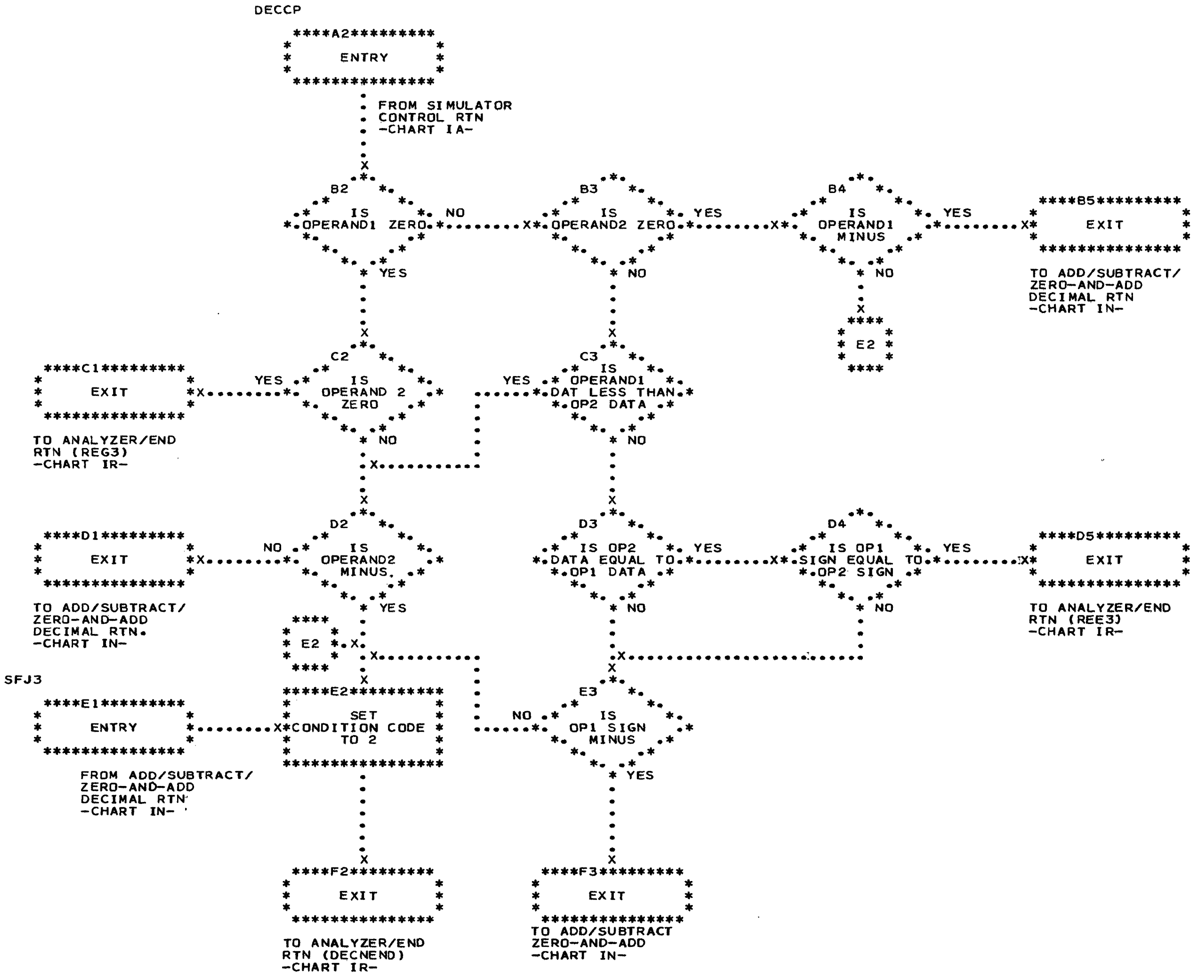
Chart IK. Asynchronous Error Routine


IEECVAE
A1 ENTRY
FROM 2250 PROCESSOR -CHART IB-
B1 IS REOPEN BIT ON
YES
NO
B2 REINITIALIZE DISPLAY CONTROL MODULE (DCM)
B3
B3 REINITILIZE 2250 BUFFER
C1
C1 IS THE BUFFER ADDR IN ERROR THE SAME
YES
NO
C2 LOG ERROR. SET DCM PERMANENT I/O ERROR FLAG
C3 IS REOPEN BIT ON
YES
NO
C4 TURN REOPEN BIT OFF
J3
D1 LOG ERROR
D2 EXIT
TO 2250 PROCESSOR RTN (IEECVDP1) CHART IA
D3 RESET OPTION INDICATORS AND DELETE MODULES IF NECESSARY
G3
E1
E1 WRITE ERROR MESSAGE
E3 ASYN ERROR
YES
NO
E4 IS THE WTO BUFFER FULL
YES
NO
E5 PUT UP WTO WARNING MSG
F3
NO
F1 WAIT FOR I/O COMPLETION
F3
F3 BE SURE FULL BUFFER SWITCH IS OFF
F4 IS THE MSG A WTOR
YES
NO
F3
F5 IS WTOR AREA FULL
YES
G1 I/O ERROR
YES
NO
G2 SET PERMANENT I/O ERROR FLAG
G3 WAS NO PRINT OPTION SET
YES
NO
G3
G4 UNDERLINE NO PRINT
G5 PUT UP WTOR WARNING MSGE
G3
H1 LIGHT PEN ATTN
YES
NO
B3
H2 EXIT
TO 2250 PROCESSOR RTN (IEECVDP1) CHART IA
H3 WAS THE SCREEN IN HELD STATE
YES
NO
H4 UNDERLINE HOLD YES
J1 ASYN ERROR
YES
NO
C1
E1
J3
J3 RESTART DISPLAY
K3 EXIT
TO 2250 PROCESSOR RTN (IEECVDP1) CHART IA

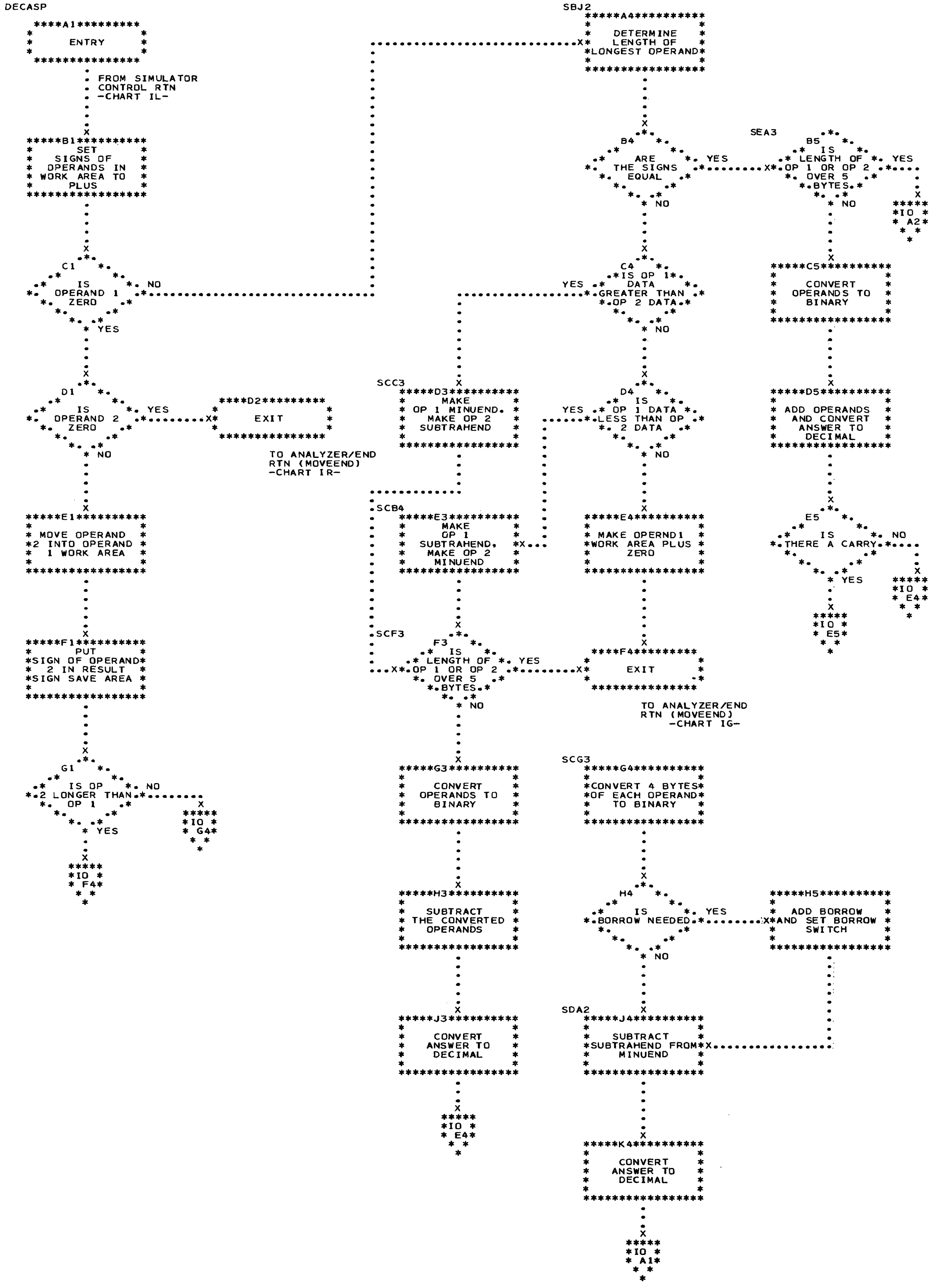
• Chart IL. Simulator Control Routine (Model 91)


A3
DECENT
A1
ENTRY
FROM MODEL 91 PFL1H RTN -CHART AH-
B1
LOCATE SIZE OF MAIN STORAGE
C1
CALCULATE LEFT AND RIGHT-HAND ADDRESSES OF OPERAND1 AND OPERAND2
D1
TEST FOR OP1/OP2 OUTSIDE OF MAIN STORAGE
YES
NO
D2
EXIT
TO ANALYZER/END RTN (DECADDR) -CHART IR-
E1
CLEAR WORK AREA
F1
TEST FOR ZAP
YES
NO
G1
ARE RIGHT-HAND ADDRESSES EQUAL
YES
NO
H1
DO OPERANDS OVERLAP
YES
NO
H2
EXIT
TO ANALYZER/END RTN (DECDC) -CHART IR-
J1
IS PROTECTION CHECKING REQUIRED
NO
E3
YES
A3
A3
IS FETCH PROTECT BIT SET FOR 001
YES
NO
B3
IS INSTR. COMPARE DECIMAL
YES
NO
C3
IS OPERAND 1 PROTECTED
YES
NO
C4
EXIT
TO MULTIPLY DECIMAL RTN. (DECMP) -CHART IP-
D3
IS OPERAND2 FETCH PROTECTED
YES
NO
D4
EXIT
TO ANALYZER/END RTN (DECPT) -CHART IR-
E3
E3
IS INSTRUCTION ZAP
YES
NO
E4
EXIT
TO COMPARE DECIMAL RTN (DECCP) -CHART IM-
F3
MOVE OPERAND1 TO WORK AREA
F4
SET OPERAND1 WORK AREA TO PLUS ZERO
G3
MOVE OPERAND2 TO WORK AREA
H3
IS OPERAND1 SIGN VALID
NO
YES
H4
EXIT
TO ANALYZER/END RTN (DECDC) -CHART IR-
J3
IS OPERAND2 SIGN VALID
NO
YES
K3
ARE OPERAND1 AND OPERAND2 VALID
NO
YES
B5
B5
B5
SET UP PREFERRED SIGNS FOR OPERANDS IN SAVE AREA
C5
TEST FOR MULTIPLY OR DIVIDE DEC.
YES
NO
D5
SET CONDITION CODE TO ZERO
E5
TEST FOR COMPARE DECIMAL
YES
NO
F5
EXIT
TO ADD/SUBTRACT/ ZERO-AND-ADD DECIMAL RTN (DECASP) -CHART IN-

• Chart IM. Compare Decimal Routine (Model 91)

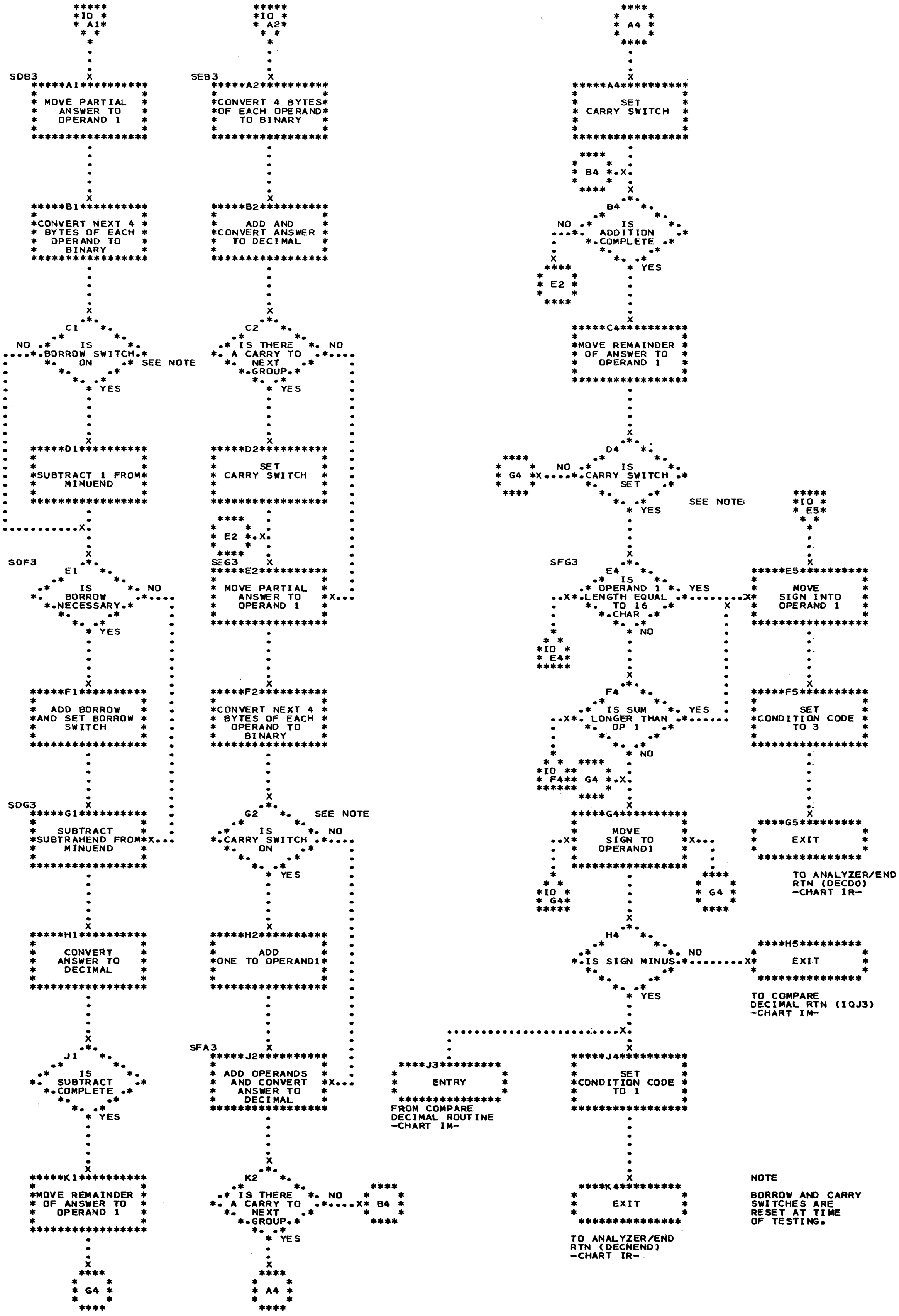

DECCP
A2
ENTRY
FROM SIMULATOR CONTROL RTN -CHART IA-
B2
IS OPERAND1 ZERO
NO
YES
B3
IS OPERAND2 ZERO
YES
NO
B4
IS OPERAND1 MINUS
YES
NO
B5
EXIT
TO ADD/SUBTRACT/ ZERO-AND-ADD DECIMAL RTN -CHART IN-
E2
C1
EXIT
TO ANALYZER/END RTN (REG3) -CHART IR-
C2
IS OPERAND 2 ZERO
YES
NO
C3
IS OPERAND1 DAT LESS THAN OP2 DATA
YES
NO
D1
EXIT
TO ADD/SUBTRACT/ ZERO-AND-ADD DECIMAL RTN. -CHART IN-
D2
IS OPERAND2 MINUS
NO
YES
E2
D3
IS OP2 DATA EQUAL TO OP1 DATA
YES
NO
D4
IS OP1 SIGN EQUAL TO OP2 SIGN
YES
NO
D5
EXIT
TO ANALYZER/END RTN (REE3) -CHART IR-
SFJ3
E1
ENTRY
FROM ADD/SUBTRACT/ ZERO-AND-ADD DECIMAL RTN -CHART IN-
E2
SET CONDITION CODE TO 2
E3
IS OP1 SIGN MINUS
NO
YES
F2
EXIT
TO ANALYZER/END RTN (DECNEND) -CHART IR-
F3
EXIT
TO ADD/SUBTRACT ZERO-AND-ADD -CHART IN-

• Chart IN. Add/Subtract/Zero-and-Add Decimal Routine (Model 91; Part 1 of 2)

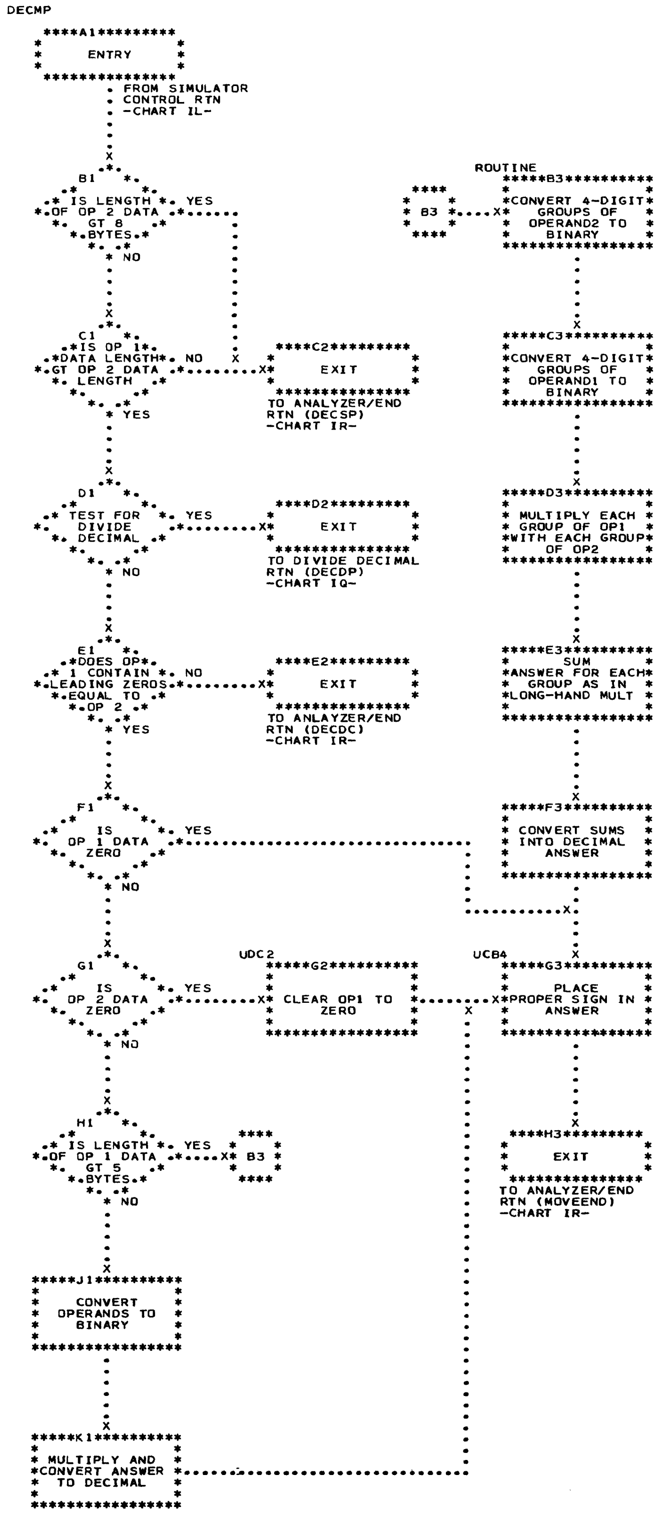


DECASP
A1
ENTRY
FROM SIMULATOR CONTROL RTN -CHART IL-
B1
SET SIGNS OF OPERANDS IN WORK AREA TO PLUS
C1
IS OPERAND 1 ZERO
NO
YES
D1
IS OPERAND 2 ZERO
YES
NO
D2
EXIT
TO ANALYZER/END RTN (MOVEEND) -CHART IR-
E1
MOVE OPERAND 2 INTO OPERAND 1 WORK AREA
F1
PUT SIGN OF OPERAND 2 IN RESULT SIGN SAVE AREA
G1
IS OP 2 LONGER THAN OP 1
NO
YES
IO G4
IO F4
SBJ2
A4
DETERMINE LENGTH OF LONGEST OPERAND
B4
ARE THE SIGNS EQUAL
YES
NO
SEA3
B5
IS LENGTH OF OP 1 OR OP 2 OVER 5 BYTES
YES
NO
IO A2
C4
IS OP 1 DATA GREATER THAN OP 2 DATA
YES
NO
C5
CONVERT OPERANDS TO BINARY
SCC3
D3
MAKE OP 1 MINUEND. MAKE OP 2 SUBTRAHEND
D4
IS OP 1 DATA LESS THAN OP 2 DATA
YES
NO
D5
ADD OPERANDS AND CONVERT ANSWER TO DECIMAL
SCB4
E3
MAKE OP 1 SUBTRAHEND. MAKE OP 2 MINUEND
E4
MAKE OPERND1 WORK AREA PLUS ZERO
E5
IS THERE A CARRY
NO
YES
IO E4
IO E5
SCF3
F3
IS LENGTH OF OP 1 OR OP 2 OVER 5 BYTES
YES
NO
F4
EXIT
TO ANALYZER/END RTN (MOVEEND) -CHART IG-
G3
CONVERT OPERANDS TO BINARY
SCG3
G4
CONVERT 4 BYTES OF EACH OPERAND TO BINARY
H3
SUBTRACT THE CONVERTED OPERANDS
H4
IS BORROW NEEDED
YES
NO
H5
ADD BORROW AND SET BORROW SWITCH
J3
CONVERT ANSWER TO DECIMAL
IO E4
SDA2
J4
SUBTRACT SUBTRAHEND FROM MINUEND
K4
CONVERT ANSWER TO DECIMAL
IO A1

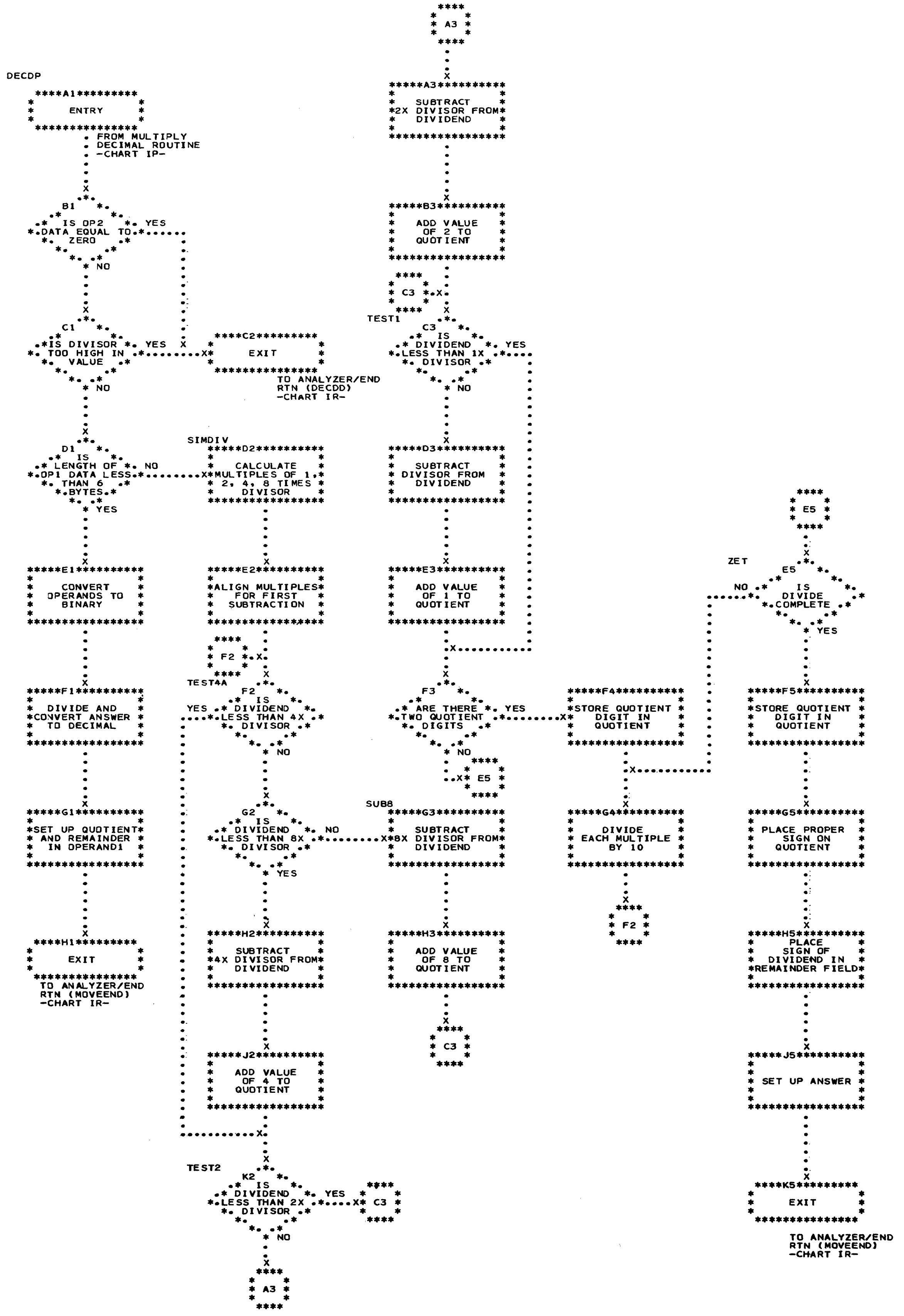
• Chart IO. Add/Subtract/Zero-and-Add Decimal Routine (Model 91; Part 2 of 2)


IO A1
SDB3
A1 MOVE PARTIAL ANSWER TO OPERAND 1
B1 CONVERT NEXT 4 BYTES OF EACH OPERAND TO BINARY
C1 IS BORROW SWITCH ON
NO
YES
SEE NOTE
D1 SUBTRACT 1 FROM MINUEND
SDF3
E1 IS BORROW NECESSARY
NO
YES
F1 ADD BORROW AND SET BORROW SWITCH
SDG3
G1 SUBTRACT SUBTRAHEND FROM MINUEND
H1 CONVERT ANSWER TO DECIMAL
J1 IS SUBTRACT COMPLETE
YES
K1 MOVE REMAINDER OF ANSWER TO OPERAND 1
G4
IO A2
SEB3
A2 CONVERT 4 BYTES OF EACH OPERAND TO BINARY
B2 ADD AND CONVERT ANSWER TO DECIMAL
C2 IS THERE A CARRY TO NEXT GROUP
NO
YES
D2 SET CARRY SWITCH
E2
SEG3
E2 MOVE PARTIAL ANSWER TO OPERAND 1
F2 CONVERT NEXT 4 BYTES OF EACH OPERAND TO BINARY
G2 CARRY SWITCH ON
SEE NOTE
NO
YES
H2 ADD ONE TO OPERAND1
SFA3
J2 ADD OPERANDS AND CONVERT ANSWER TO DECIMAL
K2 IS THERE A CARRY TO NEXT GROUP
NO
YES
B4
A4
J3 ENTRY
FROM COMPARE DECIMAL ROUTINE -CHART IM-
A4
A4 SET CARRY SWITCH
B4
B4 IS ADDITION COMPLETE
NO
YES
E2
C4 MOVE REMAINDER OF ANSWER TO OPERAND 1
D4 IS CARRY SWITCH SET
G4
NO
YES
SEE NOTE
SFG3
E4 IS OPERAND 1 LENGTH EQUAL TO 16 CHAR
YES
NO
IO E4
F4 IS SUM LONGER THAN OP 1
YES
NO
IO F4
G4
G4 MOVE SIGN TO OPERAND1
IO G4
G4
H4 IS SIGN MINUS
NO
YES
J4 SET CONDITION CODE TO 1
K4 EXIT
TO ANALYZER/END RTN (DECNEND) -CHART IR-
IO E5
E5 MOVE SIGN INTO OPERAND 1
F5 SET CONDITION CODE TO 3
G5 EXIT
TO ANALYZER/END RTN (DECDO) -CHART IR-
H5 EXIT
TO COMPARE DECIMAL RTN (IQJ3) -CHART IM-
NOTE
BORROW AND CARRY SWITCHES ARE RESET AT TIME OF TESTING.

• Chart IP. Multiply Decimal Routine (Model 91)

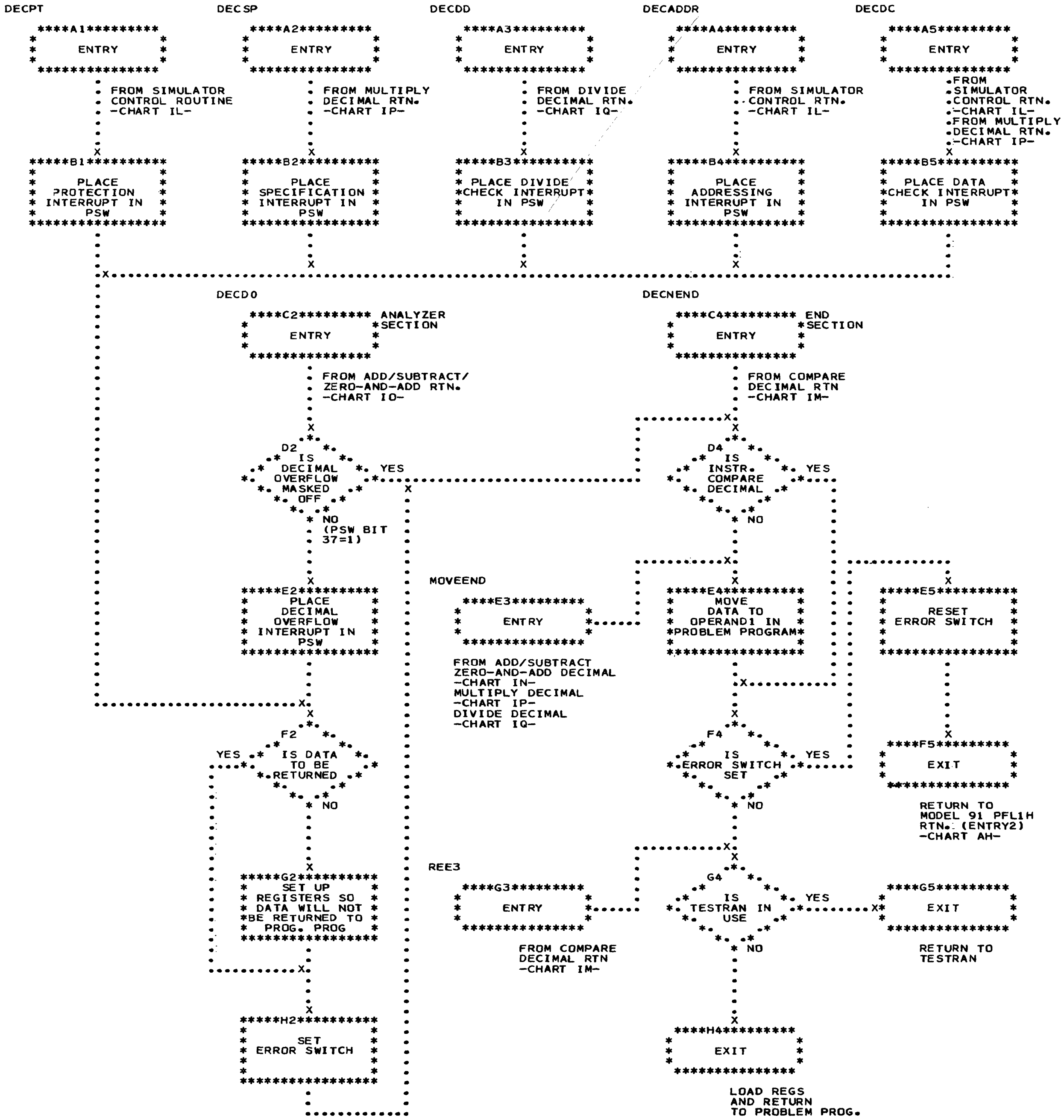

DECMP
A1 ENTRY
FROM SIMULATOR CONTROL RTN -CHART IL-
B1 IS LENGTH OF OP 2 DATA GT 8 BYTES
YES
NO
C1 IS OP 1 DATA LENGTH GT OP 2 DATA LENGTH
NO
YES
C2 EXIT
TO ANALYZER/END RTN (DECSP) -CHART IR-
D1 TEST FOR DIVIDE DECIMAL
YES
NO
D2 EXIT
TO DIVIDE DECIMAL RTN (DECDP) -CHART IQ-
E1 DOES OP 1 CONTAIN LEADING ZEROS EQUAL TO OP 2
NO
YES
E2 EXIT
TO ANLAYZER/END RTN (DECDC) -CHART IR-
F1 IS OP 1 DATA ZERO
YES
NO
G1 IS OP 2 DATA ZERO
YES
NO
UDC2
G2 CLEAR OP1 TO ZERO
H1 IS LENGTH OF OP 1 DATA GT 5 BYTES
YES
B3
NO
J1 CONVERT OPERANDS TO BINARY
K1 MULTIPLY AND CONVERT ANSWER TO DECIMAL
B3
ROUTINE
B3 CONVERT 4-DIGIT GROUPS OF OPERAND2 TO BINARY
C3 CONVERT 4-DIGIT GROUPS OF OPERAND1 TO BINARY
D3 MULTIPLY EACH GROUP OF OP1 WITH EACH GROUP OF OP2
E3 SUM ANSWER FOR EACH GROUP AS IN LONG-HAND MULT
F3 CONVERT SUMS INTO DECIMAL ANSWER
UCB4
G3 PLACE PROPER SIGN IN ANSWER
H3 EXIT
TO ANALYZER/END RTN (MOVEEND) -CHART IR-

• Chart IQ. Divide Decimal Routine (Model 91)



A3
DECDP
A1
ENTRY
FROM MULTIPLY DECIMAL ROUTINE -CHART IP-
B1
IS OP2 DATA EQUAL TO ZERO
YES
NO
C1
IS DIVISOR TOO HIGH IN VALUE
YES
NO
C2
EXIT
TO ANALYZER/END RTN (DECDD) -CHART IR-
D1
IS LENGTH OF OP1 DATA LESS THAN 6 BYTES
NO
YES
SIMDIV
D2
CALCULATE MULTIPLES OF 1, 2, 4, 8 TIMES DIVISOR
E1
CONVERT OPERANDS TO BINARY
E2
ALIGN MULTIPLES FOR FIRST SUBTRACTION
F2
TEST4A
F1
DIVIDE AND CONVERT ANSWER TO DECIMAL
F2
IS DIVIDEND LESS THAN 4X DIVISOR
YES
NO
G1
SET UP QUOTIENT AND REMAINDER IN OPERAND1
G2
IS DIVIDEND LESS THAN 8X DIVISOR
NO
YES
H1
EXIT
TO ANALYZER/END RTN (MOVEEND) -CHART IR-
H2
SUBTRACT 4X DIVISOR FROM DIVIDEND
J2
ADD VALUE OF 4 TO QUOTIENT
TEST2
K2
IS DIVIDEND LESS THAN 2X DIVISOR
YES
C3
NO
A3
A3
SUBTRACT 2X DIVISOR FROM DIVIDEND
B3
ADD VALUE OF 2 TO QUOTIENT
C3
TEST1
C3
IS DIVIDEND LESS THAN 1X DIVISOR
YES
NO
D3
SUBTRACT DIVISOR FROM DIVIDEND
E3
ADD VALUE OF 1 TO QUOTIENT
F3
ARE THERE TWO QUOTIENT DIGITS
YES
NO
E5
SUB8
G3
SUBTRACT 8X DIVISOR FROM DIVIDEND
H3
ADD VALUE OF 8 TO QUOTIENT
C3
F4
STORE QUOTIENT DIGIT IN QUOTIENT
G4
DIVIDE EACH MULTIPLE BY 10
F2
E5
ZET
E5
IS DIVIDE COMPLETE
NO
YES
F5
STORE QUOTIENT DIGIT IN QUOTIENT
G5
PLACE PROPER SIGN ON QUOTIENT
H5
PLACE SIGN OF DIVIDEND IN REMAINDER FIELD
J5
SET UP ANSWER
K5
EXIT
TO ANALYZER/END RTN (MOVEEND) -CHART IR-

• Chart IR. Analyzer/End Routine (Model 91)


DECPT
A1 ENTRY
FROM SIMULATOR CONTROL ROUTINE -CHART IL-
B1 PLACE PROTECTION INTERRUPT IN PSW
DECSP
A2 ENTRY
FROM MULTIPLY DECIMAL RTN. -CHART IP-
B2 PLACE SPECIFICATION INTERRUPT IN PSW
DECDD
A3 ENTRY
FROM DIVIDE DECIMAL RTN. -CHART IQ-
B3 PLACE DIVIDE CHECK INTERRUPT IN PSW
DECADDR
A4 ENTRY
FROM SIMULATOR CONTROL RTN. -CHART IL-
B4 PLACE ADDRESSING INTERRUPT IN PSW
DECDC
A5 ENTRY
FROM SIMULATOR CONTROL RTN. -CHART IL- FROM MULTIPLY DECIMAL RTN. -CHART IP-
B5 PLACE DATA CHECK INTERRUPT IN PSW
DECDO
C2 ENTRY
ANALYZER SECTION
FROM ADD/SUBTRACT/ ZERO-AND-ADD RTN. -CHART IO-
D2 IS DECIMAL OVERFLOW MASKED OFF
YES
NO (PSW BIT 37=1)
E2 PLACE DECIMAL OVERFLOW INTERRUPT IN PSW
F2 IS DATA TO BE RETURNED
YES
NO
G2 SET UP REGISTERS SO DATA WILL NOT BE RETURNED TO PROG. PROG
H2 SET ERROR SWITCH
MOVEEND
E3 ENTRY
FROM ADD/SUBTRACT ZERO-AND-ADD DECIMAL -CHART IN- MULTIPLY DECIMAL -CHART IP- DIVIDE DECIMAL -CHART IQ-
REE3
G3 ENTRY
FROM COMPARE DECIMAL RTN -CHART IM-
DECNEND
C4 ENTRY
END SECTION
FROM COMPARE DECIMAL RTN -CHART IM-
D4 IS INSTR. COMPARE DECIMAL
YES
NO
E4 MOVE DATA TO OPERAND1 IN PROBLEM PROGRAM
F4 IS ERROR SWITCH SET
YES
NO
G4 IS TESTRAN IN USE
YES
NO
H4 EXIT
LOAD REGS AND RETURN TO PROBLEM PROG.
E5 RESET ERROR SWITCH
F5 EXIT
RETURN TO MODEL 91 PFL1H RTN. (ENTRY2) -CHART AH-
G5 EXIT
RETURN TO TESTRAN

SECTION 14: PROGRAM ORGANIZATION

This section is designed help the reader to understand the relationships among supervisor routines and to aid the reader in locating the routines in the program listings. It includes a module directory, a directory of entry point names and flowchart IDs, a table of routines invoked by SVC instructions, and synopses of supervisor routines.

MODULE DIRECTORY

The module directory contains structural information about each routine. The directory is arranged in alphameric order by entry point names. The directory should be used to locate modules and control sections for supervisor and related routines.

LEGEND:

LIBRARY CODES

LINK = SYS1.LINKLIB Data Set
NUC = SYS1.NUCLEUS Data Set
SVC = SYS1.SVCLIB Data Set

SECTION CODES

CCSL = Console Communications and System Log
CR = Checkpoint/Restart
CS = Contents Supervision
EP = Exiting Procedures
IH = Interruption Handling
MSS = Main Storage Supervision
SF = Special Features
TMS = Timer Supervision
TP = Termination Procedures
TS = Task Supervision

Note 1: e.p. = entry point

Note 2: Blank items in chart are not applicable.

•Table 14-1. Module Directory (Part 1 of 11) (See legend on previous page)

Entry Point Name	Name of Routine, Control Block, Table, Transient Area	Module Name	Control Section Name	PLM References		Lib.	If SVC Routine		
				Section	Chart ID		Type	Macro Instruction	SVC Instr.
CATNIP	Channel-Check Handler (CCH)	IEAQFX00	IEAQFX00	IH		NUC			
CDADVANS	Contents Supvsn. common subroutines (search phase)	IEAQLK00	IEAQLK00	CS	CA	NUC			
CDCONTRL also called IEAQCS02	Contents Supvsn. common subroutines (search phase)	IEAQLK00	IEAQLK00	CS	CA	NUC			
CDDESTRY	CDEXIT routine	IEAQED02	IGC003	EP,TP	GF	NUC			
CDEPILOG	Contents Supvsn. common subroutines (scheduling phase)	IEAQLK00	IEAQLK00	CS	CB	NUC			
CDEXIT	CDEXIT routine	IEAQED02	IGC003	EP	GF	NUC			
CDHKEEP	CDEXIT routine	IEAQED02	IGC003	EP,TP	GF	NUC			
EOT	End-of-Task (EOT) routine	IEAQED02	IGC003	TP	HA	NUC			
ERFETCH	Stage 3 Exit Effector	IEAQNU02	IEAQNU02	TS	BU	NUC			
FLASH	First CPU Signal Routine	IEAQFX00	IEAQFX00	TS		NUC			
FMBRANCH	FREEMAIN routine (branch e. p.)	IEAQGM00	IEAQGM00	MSS	DB	NUC	1		
FREEPART	FREEMAIN routine (branch e. p. for request to free a region)	IEAQPRT0	IEAQPRT0	MSS	DB	NUC	1		
FTCE01	Program Fetch Channel-End Appendage routine	IEWFETCH	IEWFETCH	CS	CG	NUC			
FTPCI01	Program Fetch PCI Appendage routine	IEWFETCH	IEWFETCH	CS	CG	NUC			
GETIQE	GETMAIN routine (branch e.p. to the GETIQE subroutine)	IEAQPRT0	IEAQPRT0	CS		NUC			
GETPART	GETMAIN routine (branch e. p. for request to allocate a region)	IEAQPRT0	IEAQPRT0	MSS	DA	NUC	1		
GMBRANCH	GETMAIN routine (branch e. p.)	IEAQGM00	IEAQGM00	MSS	DA	NUC	1		
IBMORG	SVC table (start of IBM-assigned SVC numbers)	IEAQBK00	IEAQBK00			NUC			
IEABEND	secondary communications vector table (used by the ABEND routine)	IEAQED02	IGC003			NUC			
IEACVT	communications vector table	IEAQBK00	IEAQBK00			NUC			
IEADQTCB	Dequeue TCB routine	IEAQED02	IGC003	TP	HA	NUC			
IEAERRTA	I/O block (IOB) for the I/O Supervisor transient area	IEAQBK00	IEAQBK00			NUC			
IEAERTCB	TCB for the system error task (associated with the I/O Supervisor transient area)	IEAQBK00	IEAQBK00	TS		NUC			
IEAERWA	I/O Supervisor transient area	IEAQBK00	IEAQBK00			NUC			
IEAMCH00	SERO routine (resident module)	IFBSR000	IFBSR000	IH	AM	NUC			

• Table 14-1. Module Directory (Part 2 of 11) (See legend before Part 1)

Entry Point Name	Name of Routine, Control Block, Table, Transient Area	Module Name	Control Section Name	PLM References		Lib.	If SVC Routine		
				Section	Chart ID		Type	Macro Instruction	SVC Instr.
IEAMCH00	SER1 routine for System/360, model 40	IFBSR340	IFBSR340	IH	AN	LINK			
	SER1 routine for System/360, model 50	IFBSR350	IFBSR350	IH	AN	LINK			
	SER1 routine for System/360, model 65	IFBSR365	IFBSR365	IH	AN	LINK			
	SER1 routine for System/360, model 75	IFBSR375	IFBSR375	IH	AN	LINK			
	SER1 routine for System/360, model 91	IFBSR395	IFBSR395	IH	AO	LINK			
IEAMSTCB	TCB for the Master Scheduler task	IEAQBK00	IEAQBK00			NUC			
IEANIP4	Nucleus Initialization Program	IEANUC0n	IEAANIP0			NUC			
IEAOPT01	Post routine (branch e. p. from the I/O Supervisor)	IEAQSY50	IGC001	TS	BM	NUC	1		
IEAOPT02	Post routine (branch e. p. from the I/O Supervisor and from supervisor routines)	IEAQSY50	IGC001	TS	BM	NUC	1		
IEAQABL	Release Loaded Programs routine	IEAQED02	IGC003	TP	HD	NUC			
IEAQCS01	Contents Supervsn. common sub-routines (e.p. for the ATTACH macro instruction)	IEAQLK00	IEAQLK00	CS	CA	NUC			
IEAQCS02 (also called CDCONTRL)	Contents Supervsn. common sub-routines (search phase)	IEAQLK00	IEAQLK00	CS	CA	NUC			
IEAQCS03 also called CDEPILOG	Contents Supervsn. common sub-routines (scheduling phase)	IEAQLK00	IEAQLK00	CS	CB	NUC			
IEAQEQ01	ENQ/DEQ Purge routine	IEAQENQ2	IGC048	TP		NUC			
IEAQERA	Erase Phase routine	IEAQED02	IGC003	TP		NUC			
IEAQEX00	External First-Level Interruption Handler	IEAQNU00	IEAQNU00	IH	AI-AJ	NUC			
IEAQI000	I/O First-Level Intrp. Handler	IEAQNU00	IEAQNU00	IH	AK-AL	NUC			
IEAQLPAQ	Link pack area queue	IEAQBK00	IEAQBK00			NUC			
IEAQPGTM	Purge Timer routine	IEAQED02	IGC003	TP	HB	NUC			
IEAQPK00	Program Check First-Level Interruption Handler	IEAQNU00	IEAQNU00	IH	AF-AH	NUC			
IEAQQCB0	Origin of QCB queues	IEAQENQ2	IGC048			NUC			
IEAQRORI	Rollout/Rollin module	IEAQRORI	IEAQRORI	MSS	DC-DJ	NUC			
IEAQSC00	SVC First-Level Interruption Handler	IEAQNU00	IEAQNU00	IH	AA-AB	NUC			
IEAQSPET	Release Main Storage routine	IEAQED02	IHC003	TP	HC	NUC			

• Table 14-1. Module Directory (Part 3 of 11) (See legend before Part 1)

Entry Point Name	Name of Routine, Control Block, Table, Transient Area	Module Name	Control Section Name	PLM References		Lib.	If SVC Routine		
				Section	Chart ID		Type	Macro Instruction	SVC Instr.
IEAQTAQ IEAQTAQ1	transient area control table (TACT)	IEAQBK00	IEAQBK00			NUC			
IEAQTD00	Timer Second-Level Interruption Handler (branch e.p.)	IEAQTI00	IEAQTI00	TMS	ED	NUC			
IEAQTD01	Timer Second-Level Interruption Handler (e.p. for Dispatcher)	IEAQTI00	IEAQTI00	TMS	EE	NUC			
IEAQTE00	Timer Second-Level Interruption Handler (branch e.p.)	IEAQTI00	IEAQTI00	TMS	EE	NUC			
IEAQTR00	SVC Second-Level Interruption Handler	IEAQTR33	IEAQTR00	IH	AC	NUC			
IEAQTR01	Transient Area Exit routine	IEAQTR33	IEAQTR00	EP	GD	NUC			
IEAQTR02	Transient Area Refresh routine	IEAQTR33	IEAQTR00	EP	GE	NUC			
IEAQTR03	Transient Area XCTL routine	IEAQTR33	IEAQTR00	CS	CC	NUC			
IEATCB1	transient area fetch TCBs	IEAQBK00	IEAQBK00			NUC			
IEATCB2		IEAQBK00	IEAQBK00			NUC			
IEATCBn		IEAQBK00	IEAQBK00			NUC			
IEATCBn+1		IEAQBK00	IEAQBK00			NUC			
IEATYPE1	Type-1 SVC Switch	IEAQNU00	IEAQNU00			NUC			
IEAXDS00	Decimal Simulator routine, (Model 91)	IEAXDS00	IEAXDS00	SF	IL-IR				
IEA0AB00 IEA0AB01	ABTERM routine	IEAQAB00	IEAQAB00	TP	HE-HF	NUC			
IEA0DS	Dispatcher	IEAQNU00	IEAQNU00	EP	GG-GP	NUC			
IEA0DS02	Task Switching routine	IEAQNU00	IEAQNU00	TS	BV-BW	NUC			
IEA0EF00	Stage 2 Exit Effector	IEAQNU00	IEAQNU00	TS	BS	NUC			
IEA0EF03	Stage 3 Exit Effector	IEAQNU00	IEAQNU00	TS	BT	NUC			
IEA0PL00	ABTERM Prologue routine	IEAQAB00	IEAQAB00	TP	HG	NUC			
IEA0TI00	Timer Second-Level Interruption Handler (e.p. for External First-Level Interruption Handler)	IEAQTI00	IEAQTI00	TMS	ED	NUC			
IEAOVL00	Validity Check routine	IEAQNU00	IEAQNU00	TS		NUC			
IEA0XE00	Type-1 Exit routine	IEAQNU00	IEAQNU00	EP	GA	NUC			
IECINT	I/O Interruption Supvsr. in the I/O Supervisor	IEAQFX00 (IEAQFX and IECIOS macros)	IEAQFX00	CS		NUC			
IECIWTST	System Quiesce routine	IEAGTWST	IEAF03BP	TP	HT	NUC			
IECPBLDL	BLDL routine	IECPFIND or IECPFND1 (depends on SYSGEN option)	IGC018			NUC	2	BLDL	SVC 18

• Table 14-1. Module Directory (Part 4 of 11) (See legend before Part 1)

Entry Point Name	Name of Routine, Control Block, Table, Transient Area	Module Name	Control Section Name	PLM References		Lib.	If SVC Routine		
				Section	Chart ID		Type	Macro Instruction	SVC Instr.
IECXCP	EXCP Supervsr. in the I/O Supervisor	IEAQFX00 (IEAQFX and IECIOS macros)	IEAQFX00	CS	CF	NUC	1	EXCP	SVC 0
IECXTLER	Stage 3 Exit Effector	IEAQNU00	IEAQNU00	TS	BU	NUC			
IEEBA1	Attention routine	IEECVCRA	IEEBA1	CCSL	(Fig. 7-1)	NUC			
IEEBC1PE	Communications Task External Interruption Handler routine	IEECVCRX	IEEBCIPE	CCSL	(Fig. 7-1)	NUC			
IEECVAE	Asynchronous Error routine	IGC3707B	IGC3707B	SF		SVC	4	XCTL	SVC 72
IEECVCTI	Communications Task Initialization routine	IEECVINT	IEECVCTI	CCSL	(Fig. 7-1)	NUC			
IEECVCTW	Communications Task Wait routine	IEECVCTB	IEECVCTW	CCSL	(Fig. 7-1, 7-2)	NUC			
IEECVAOP	Command Format Order Prog. RELEASE (A) (not executable)	IEECVAOP	IEECVAOP	SF		LINK			
IEECVBOP	Command Format Order Prog. BRDCST (RJE) (not executable)	IEECVBOP	IEECVBOP	SF		LINK			
IEECVCOP	Command Format Order Prog. CANCEL (C) (not executable)	IEECVCOP	IEECVCOP	SF		LINK			
IEECVDCM	Display Control module (not executable)	IEANUC01	IEANUC01	SF		NUC			
IEECVDOP	Command Format Order Prog. DISPLAY (D) (not executable)	IEECVDOP	IEECVDOP	SF		LINK			
IEECVDP1	2250 Processor routine	IGC3107B	IGC3107B	SF	IA	SVC	4	XCTL	SVC 72
IEECVDP2	2250 Processor routine	IGC3207B	IGC3207B	SF	IC	SVC	4	XCTL	SVC 72
IEECVDR1	Display routine	IGC3407B	IEECVDR1	SF	IE	SVC	4	XCTL	SVC 72
IEECVDR2	I/O (Part 1) routine	IGC3507B	IEECVDR2	SF	IF	SVC	4	XCTL	SVC 72
IEECVDR3	I/O (Part 2) routine	IGC3607B	IEECVDR3	SF	IH	SVC	4	XCTL	SVC 72
IEECVDR4	Options routine	IGC3807B	IEECVDR4	SF	II	SVC	4	XCTL	SVC 72
IEECVDR5	Options routine	IGC3907B	IEECVDR5	SF	II	SVC	4	XCTL	SVC 72
IEECVEOP	Command Format Order Prog. RESET (E) (not executable)	IEECVEOP	IEECVEOP	SF		LINK			
IEECVFOP	Command Format Order Prog. MODIFY (F) (not executable)	IEECVFOP	IEECVFOP	SF		LINK			
IEECVHOP	Command Format Order Prog. HOLD (H) (not executable)	IEECVHOP	IEECVHOP	SF		LINK			
IEECVIOP	Command Format Order Prog. MSG (RJE) (not executable)	IEECVIOP	IEECVIOP	SF		LINK			
IEECVJOP	Command Format Order Prog. CENOUT (RJE) (not executable)	IEECVJOP	IEECVJOP	SF		LINK			

• Table 14-1. Directory (Part 5 of 11) (See legend before Part 1)

Entry Point Name	Name of Routine, Control Block, Table, Transient Area	Module Name	Control Section Name	PLM References		Lib.	If SVC Routine		
				Section	Chart ID		Type	Macro Instruction	SVC Instr.
IEECVKOP	Command Format Order Prog. SHOW (RJE) (not executable)	IEECVKOP	IEECVKOP	SF		LINK			
IEECVLOP	Command Format Order Prog. LOG (L) (not executable)	IEECVLOP	IEECVLOP	SF		LINK			
IEECVMOP	Command Format Order Prog. MOUNT (M) (not executable)	IEECVMOP	IEECVMOP	SF		LINK			
IEECVNOP	Command Format Order Prog. START (RJE) (not executable)	IEECVNOP	IEECVNOP	SF		LINK			
IEECVOCG	Open/Close routine	IGC3I07B	IEECVOCG	SF	IJ	SVC	4	XCTL	XCTL
IEECVOOP	Command Format Order Prog. USERID (RJE) (not executable)	IEECVOOP	IEECVOOP	SF		LINK			
IEECVOP1	Unit Status Order Prog. Option 1 (not executable)	IEECVOP1	UNITDISP	SF		LINK			
IEECVOP2	Command Format Order Prog. Option 2 (not executable)	IEECUOP2	FORMAT	SF		LINK			
IEECVPOP	Command Format Order Prog. STOP (P) (not executable)	IEECVPOP	IEECVPOP	SF		LINK			
IEECVPRB	Communications Task request block (RB)	IEAQBK00	IEAQBK00			NUC			
IEECVPRG	WTOR Purge routine (also called the Reply Purge routine)	IEECVED2	IEECVPRG	TP		NUC			
IEECVQOP	Command Format Order Prog. STOP (RJE) (not executable)	IEECVQOP	IEECVQOP	SF		LINK			
IEECVROP	Command Format Order Prog. REPLY (R) (not executable)	IEECVROP	IEECVROP	SF		LINK			
IEECVSOP	Command Format Order Prog. START (S) (not executable)	IEECVSOP	IEECVSOP	SF		LINK			
IEECVTCB	Communications Task TCB	IEAQBK00	IEAQBK00	CCSL		NUC			
IEECVTOP	Command Format Order Prog. SET (T) (not executable)	IEECVTOP	IEECVTOP	SF		LINK			
IEECVUCM	Communications Task unit control tables	IEECVUCM	IEECVUCM	CCSL	(Fig. 7-1, 7-2)	NUC			
IEECVUOP	Command Format Order Prog. UNLOAD (U) (not executable)	IEECVUOP	IEECVUOP	SF		LINK			
IEECVVOP	Command Format Order Prog. VARY (V) (not executable)	IEECVVOP	IEECVVOP	SF		LINK			
IEECVWOP	Command Format Order Prog. WRITELOG (W) (not executable)	IEECVWOP	IEECVWOP	SF		LINK			
IEECVZOP	Command Format Order Prog. HALT (Z) (not executable)	IEECVZOP	IEECVZOP	SF		LINK			
IEEMSER	Master Scheduler resident table	IEEBASEC	IEEMSER			NUC			
IEEPLDSP	WRITELOG Get Region routine	IEEPLDSP	IEEPLDSP	CCSL		LINK			
IEEVIPL	Master Scheduler Initialization routine	IEECVIPL	IEEVIPL	CCSL	(Fig. 7-3)	LINK			

• Table 14-1. Module Directory (Part 6 of 11) (See legend before Part 1)

Entry Point Name	Name of Routine, Control Block, Table, Transient Area	Module Name	Control Section Name	PLM References		Lib.	If SVC Routine		
				Section	Chart ID		Type	Macro Instruction	SVC Instr.
IEEVLDSP	WRITELOG Dispatch routine	IEEVLDSP	IEEVLDSP	CCSL	(Fig. 7-3)	LINK			
IEEVLIN	WRITELOG Log Initialization routine	IEEVLIN	IEEVLIN1	CCSL	(Fig. 7-3)	LINK			
IEEVLOGJ	Job File Control Blocks (JFCBs) for log data sets	IEEVLOGJ	IEEVLOGJ	CCSL	(Fig. 7-3)	LINK			
IEEVLOPN	WRITELOG Open Device routine	IEEVLOPN	IEEVLOPN	CCSL	(Fig. 7-3)	LINK			
IEEVLOUT	WRITELOG Available Log Data Set routine	IEEVLOUT	IEEVLOUT	CCSL	(Fig. 7-3)	LINK			
IEEVLWTR	WRITELOG Log Write routine	IEEVLWTR	IEEVLWTR	CCSL	(Fig. 7-3)	LINK			
IEEVL03F	Write-to-Log routine	IGC0003F	IEEVL03F	CCSL	(Fig. 7-3)	SVC	3	WTL	SVC 36
IEEVRFRX	Communications Task Misc. Look-up Services routine	IEEVRFRX	IEEVRFRX	CCSL		LINK			
IEEVWAIT	WRITELOG Master Wait routine	IEEVWAIT	IEEVWAIT	CCSL	(Fig. 7-3)	LINK			
IEE1103D	Log and WRITELOG Post routine	IGC1103D	IEE1103D	CCSL	(Fig. 7-3)	SVC	4	MGCR	SVC 34
IEE1203D	Communications Task Reply Processor routine	IEE1203D	IEE1203D	CCSL		SVC			
IEGHTOVL	TESTRAN Interpreter	IEGTTRNO	IEGHTOVL	CS		LINK			
IEWFBOSV	Program Fetch routine (e.p. from the Overlay Supervisor)	IEWFETCH	IEWFETCH	CS	CF-CG	NUC			
IEWFTRAN	Program Fetch routine (e.p. from the TA Fetch routine)	IEWFETCH	IEWFETCH	CS	CF-CG	NUC			
IEWMSEPT	Program Fetch routine (e.p. from the common subroutines of Contents Supervision)	IEWFETCH	IEWFETCH	CS	CF-CG	NUC			
IEWSZOVR	Overlay Supervisor (non-resident module)	IEWSWOVR	IEWSWOVR	CS	CH	LINK			
IFBSER00	SER0 routine (System/360, model 40)	IFBSR040	IFBSR040	IH	AM	LINK			
IFBSER00	SER0 routine (System/360, model 50)	IFBSR050	IFBSR050	IH	AM	LINK			
IFBSER00	SER0 routine (System/360, model 65)	IFBSR065	IFBSR065	IH	AM	LINK			
IFBSER00	SER0 routine (System/360, model 75)	IFBSR075	IFBSR075	IH	AM	LINK			
IGCXL07B	Communications Task External Processor routine	IEECVCTX	IEECVCTX	CCSL	(Fig. 7-1)	SVC			
IGC0A05A	ABDUMP routine ("resident" module)	IEAQAD0A	IGC0A05A	TP	HH	SVC	4		

• Table 14-1. Module Directory (Part 7 of 11) (See legend before Part 1)

Entry Point Name	Name of Routine, Control Block, Table, Transient Area	Module Name	Control Section Name	PLM References		Lib.	If SVC Routine		
				Section	Chart ID		Type	Macro Instruction	SVC Instr.
IGC0I07B	Communications Task 1052 Console Open/Close routine	IEECVOCX	IEECVOC	CCSL	(Fig. 7-1, 7-2)	SVC			
IGC0001C	ABEND routine (ABEND1)	IEAQTM00	IGC0001C	TP	HI	SVC	4	ABEND	SVC 13
IGC0003E	Write-to-Operator routine	IEECVWTO	IGC0003E	CCSL	(Fig. 7-2)	SVC	3	WTO WTOR	SVC 35
IGC0005A	ABDUMP routine (ABDUMP1)	IEAQAD00	IGC0005A	TP	HH	SVC	4	SNAP	SVC 51
+ IGC00060	STAE Service routine	IEAAST00	IGC00060	TS	BX-B0	SVC	3	STAE	SVC 60
IGC0007B	Communications Task Router routine	IEECVCTR	IEECVCTR	CCSL	(Fig. 7-1, 7-2)	SVC	4	CHATR	SVC 72
IGC001	Wait routine	IEAQSY50	IGC001	TS	BK-BL	NUC	1	WAIT	SVC 1
IGC002	Post routine	IEAQSY50	IGC001	TS	BM-BN	NUC	1	POST	SVC 2
IGC002+6	Post routine (branch e.p. from supervisor routines)								
IGC003	Exit routine	IEAQED02	IGC003	EP	GB-GC	NUC	1		SVC 3
IGC004	GETMAIN routine (e.p. for S-form macro instruction)	IEAQGM00	IEAQGM00	MSS	DA	NUC	1	GETMAIN(S)	SVC 4
IGC005	FREEMAIN routine (e.p. for S-form macro instruction)	IEAQGM00	IEAQGM00	MSS	DB	NUC	1	FREEMAIN(S)	SVC 5
IGC006	Contents Supervision, common subroutines (e. p. for the LINK macro instruction)	IEAQLK00	IEAQLK00	CS	CA	NUC	2	LINK	SVC 6
IGC007	Contents Supervision, common subroutines (e.p. for the XCTL macro instruction)	IEAQLK00	IEAQLK00	CS	CC	NUC	2	XCTL	SVC 7
IGC008	Contents Supervision, common subroutines (e.p. for the LOAD macro instruction)	IEAQLK00	IEAQLK00	CS	CC	NUC	2	LOAD	SVC 8
IGC009	Delete routine	IEAQLK00	IEAQLK00	CS	CE	NUC	2	DELETE	SVC 9

IGC010	GETMAIN/FREEMAIN routines (e.p. for R-form macro instructions)	IEAQGM00	IEAQGM00	MSS	DA	NUC	1	GETMAIN(R)
								FREEMAIN(R)
									SVC 10

IGC011	Time routine	IEAQRT00	IGC011	TMS	EA	NUC	1	TIME	SVC 11
IGC012	Contents Supervision, common subroutines (e.p. for the SYNCH macro instruction)	IEAQLK00	IEAQLK00	CS	CB	NUC	2	SYNCH	SVC 12
IGC014	SPIE routine	IEAQTB00	IGC014	TS	BJ	NUC	2	SPIE	SVC 14
IGC016	SVC Purge routine	IECIPR16 macros	IGC016	MSS		NUC	2	PURGE	SVC 16
IGC037	Overlay Supervisor, resident module (e.p. for a SEGLD or SEGWT macro instruction)	IEWSUOVR	IGC037	CS	CI	NUC	2	SEGLD SEGWT	SVC 37
IGC040 IGC040+8	Extract routine Extract routine (branch e.p.)	IEAQTB00	IGC014	TS	BH	NUC	1	EXTRACT	SVC 40

• Table 14-1. Module Directory (Part 8 of 11) (See legend before Part 1)

Entry Point Name	Name of Routine, Control Block, Table, Transient Area	Module Name	Control Section Name	PLM References		Lib.	If SVC Routine		
				Section	Chart ID		Type	Macro Instruction	SVC Instr.
IGC041	Identify routine	IEAQID00	IGC041	CS	CD	NUC	2	IDENTIFY	SVC 41
IGC042	Attach routine	IEAQAT00	IGC042	TS	BA-BC	NUC	2	ATTACH	SVC 42
IGC043	Stage 1 Exit Effector	IEAQEF00	IGC043	TS	BR	NUC	2	CIRB	SVC 43
IGC044 IGC044 +12	CHAP routine CHAP routine (branch e.p.)	IEAQTB00	IGC044	TS	BE-BG	NUC	1	CHAP	SVC 44
IGC045	Overlay Supervisor, resident module (e.p. for branch instruction or CALL macro instruction)	IEWSUOVR	IGC037	CS	CI	NUC	2	CALL	SVC 45
IGC046	TTIMER routine	IEAQST00	IGC046	TMS	EC	NUC	1	TTIMER	SVC 46
IGC047	STIMER routine	IEAQST00	IGC046	TMS	EB	NUC	2	STIMER	SVC 47
IGC048	Dequeue routine	IEAQENQ2	IGC048	TS	BP-BQ	NUC	2	DEQ	SVC 48
IGC056	Enqueue routine	IEAQENQ2	IGC048	TS	BO	NUC	2	ENQ	SVC 56
IGC061	TESTRAN interpreter	IGC0006A	IEGHRSAV	CS	CA	SVC	3	TTSAV	SVC 61
IGC062	Detach routine	IEAQED02	IGC003	TS	BI	NUC	2	DETACH	SVC 62
IGC079	Set Status routine	IEAQSETS	IEAQSETS	MSS		NUC	1	STATUS	SVC 79
IGC0005B	Restart Housekeeping 1	IHJARS00	IGC0005B	CR	FJ	SVC	4	CHKPT	
IGC0006C	Checkpoint Housekeeping 1	IHFACP00	IGC0006C	CR	FA	SVC	4	CHKPT	63
IGC0101C	ABEND routine (ABEND4)	IEAQTM01	IGC0101C	TP	HN	SVC	4		
IGC0105A	ABDUMP routine (ABDUMP2)	IEAQAD01	IGC0105A	TP	HH	SVC	4		
IGC0105B	Restart Housekeeping 2	IHJARS01	IGC0105B	CR	FJ	SVC	4	CHKPT	52
IGC0106C	Checkpoint Housekeeping 2	IHJACP01	IGC0106C	CR	FB	SVC	4	CHKPT	63
IGC0107B	Communications Task 1052 Console Processor routine	IEECVPMX	IEECVPM	CCSL	(Fig. 7-1, 7-2)	SVC			
IGC0201C	ABEND routine (ABEND5)	IEAQTM02	IGC0201C	TP	HO-HP	SVC	4		
IGC0205A	ABDUMP routine (ABDUMP3)	IEAQAD02	IGC0205A	TP	HH	SVC	4		
IGC0206C	Checkpoint Housekeeping 3	IHJACP02	IGC0206C	CR	FC	SVC	4	CHKPT	63
IGC0301C	ABEND routine (ABEND6)	IEAQTM03	IGC0301C	TP	HQ	SVC	4		
IGC0305A	ABDUMP routine (ABDUMP4)	IEAQAD03	IGC0305A	TP	HH	SVC	4		
IGC0401C	ABEND routine (ABEND2)	IEAQTM04	IGC0401C	TP	HJ	SVC	4		
IGC0405A	ABDUMP routine (ABDUMP5)	IEAQAD04	IGC0405A	TP	HH	SVC	4		
IGC0505A	ABDUMP routine (ABDUMP6)	IEAQAD05	IGC0505A	TP	HH	SVC	4		
IGC0505B	Repmain 1	IHJQRS20	IGC0505B	CR	FK	SVC	4	CHKPT	52
IGC0506C	Check I/O routine	IHJACP10	IGC0506C	CR	FD	SVC	4	CHKPT	63
IGC0605A	ABDUMP routine (ABDUMP7)	IEAQAD06	IGC0605A	TP	HH	SVC	4		
IGC0605B	Repmain 2	IHJQRS21	IGC0605B	CR	FK	SVC	4	CHKPT	52

• Table 14-1. Module Directory (Part 9 of 11) (See legend before Part 1)

Entry Point Name	Name of Routine, Control Block, Table, Transient Area	Module Name	Control Section Name	PLM References		Lib.	If SVC Routine		
				Section	Chart ID		Type	Macro Instruction	SVC Instr.
IGC0705A	ABDUMP routine (ABDUMP8)	IEAQAD07	IGC0705A	TP	HH	SVC	4		
IGC0705B	Repmain 3	IHJQRS22	IGC0705B	CR	FL	SVC	4	CHKPT	52
IGC0805A	ABDUMP routine (ABDUMP9)	IEAQAD08	IGC0805A	TP	HH	SVC	4		
IGC0805B	Repmain 4	IHJQRS23	IGC0805B	CR	FL	SVC	4	CHKPT	52
IGC0905B	Repmain 5	IHJQRS24	IGC0905B	CR	FM,FN	SVC	4	CHKPT	52
IGC0A01C	ABEND routine (ABEND3)	IEAQTM0A	IGC0A01C	TP	HK-HM	SVC	4		
IGC0A06C	Preserve 1 routine	IHJACP20	IGC0A06C	CR	FE	SVC	4	CHKPT	63
IGC0B01C	ABEND/STAE Interface 1 routine	IEAQTM0B	IGC0B01C	TS	BY	SVC	4		
IGC0B05A	ABDUMP routine (ABDUMP 11)	IEAQAD0B	IGC0B05A	TP	HH	SVC	4		
IGC0C01C	ABEND/STAE Interface 2 routine	IEAQTM0C	IGC0C01C	TS	BZ	SVC	4		
IGC0D01C	ABEND/STAE Interface 3 routine	IEAQTM0D	IGC0D01C	TS	B0	SVC	4		
IGC0D06C	Preserve 2 routine	IHJACP25	IGC0D06C	CR	FE	SVC	4	CHKPT	63
IGC0E01C	ABEND/STAE Interface 4 routine	IEAQTM0E	IGC0E01C	TS	BO	SVC	4		
IGC0F06C	Checkmain 1	IHJQCP30	IGC0F06C	CR	FF	SVC	4	CHKPT	63
IGC0G05B	REP I/O-JFCB Processor 1	IHJARS40	IGC0G05B	CR	FO	SVC	4	CHKPT	52
IGC0G06C	Checkmain 2	IHJQCP31	IGC0G06C	CR	FF	SVC	4	CHKPT	63
IGC0H06C	Checkmain 3	IHJQCP32	IGC0H06C	CR	FG	SVC	4	CHKPT	63
IGC0I05B	REP I/O-JFCB Processor 2	IHJARS41	IGC0I05B	CR	FO	SVC	4	CHKPT	52
IGC0K05B	REP I/O-Mount/Verify routine 1	IHJARS43	IGC0K05B	CR	FP	SVC	4	CHKPT	52
IGC0M05B	REP I/O-Mount/Verify routine 2	IHJARS45	IGC0M05B	CR	FQ	SVC	4	CHKPT	52
IGC0N05B	REP I/O-SYSIN/SYSOUT Data Set Processor 1	IHJARS4D	IGC0N05B	CR	FR	SVC	4	CHKPT	52
IGC0N06C	Resume I/O routine	IHJACP40	IGC0N06C	CR	FH	SVC	4	CHKPT	63
IGC0P05B	REP I/O-Data Set Processor 1	IHJARS47	IGC0P05B	CR	FS	SVC	4	CHKPT	52
IGC0Q05B	REP I/O-SYSIN/SYSOUT Data Set Processor 2	IHJARS4E	IGC0Q05B	CR	FR	SVC	4	CHKPT	52
IGC0Q06C	Checkpoint Exit routine	IHJACP50	IGC0Q06C	CR	FI	SVC	4	CHKPT	63
IGC0R05B	REP I/O-Data Set Processor 2	IHJARS49	IGC0R05B	CR	FT	SVC	4	CHKPT	52
IGC0S06C	Checkpoint Message Module	IHJACP70	IGC0S06C	CR	FI	SVC	4	CHKPT	63
IGC0T05B	REP I/O-Access Method-Disposition routine	IHJARS4B	IGC0T05B	CR	FU	SVC	4	CHKPT	52
IGC0V05B	Restart Exit routine	IHJARS60	IGC0V05B	CR	FU	SVC	4	CHKPT	52

• Table 14-1. Module Directory (Part 10 of 11) (See legend before Part 1)

Entry Point Name	Name of Routine, Control Block, Table, Transient Area	Module Name	Control Section Name	PLM References		Lib.	If SVC Routine		
				Section	Chart ID		Type	Macro Instruction	SVC Instr.
IGC1I07B	Communications Task 2540 Console Open/Close routine	IEECVOCC	IEECVOC	CCSL	(Fig. 7-2)	SVC			
IGC1I07B	Communications Task 2540 Console Processor routine	IEECVPMC	IEECVPM	CCSL	(Fig. 7-1, 7-2)	SVC			
IGC2I07B	Communications Task 1443 Printer Open/Close routine	IEECVOCP	IEECVOC	CCSL CCSL	(Fig. 7-1, 7-2	SVC			
IGC2107B	Communications Task 1443 Printer Processor routine	IEECVPMP	IEECVPM	CCSL	(Fig. 7-1, 7-2)	SVC			
IGFASR0B	CPU Analysis module	IGFASR0B	IGFASR0B			SVC			
IGFASR0C	Instruction Retry Analysis module, phase 1	IGFASR0C	IGFASR0C			SVC			
IGFASR0D	Storage Protection Feature Test module	IGFASR0D	IGFASR0D			SVC			
IGFASR01	MCH Error Recorder module	IGFASR01	IGFASR01			SVC			
IGFASR00	System Analysis module	IGFASR00	IGFASR00			SVC			
IGFASR1A	Refresh Clear Channel module	IGFASR1A	IGFASR1A			SVC			
IGFASR1C	Instruction Retry Analysis module, phase 2	IGFASR1C	IGFASR1C			SVC			
IGFASR1D	Error Check Circuitry Verification module	IGFASR1D	IGFASR1D			SVC			
IGFASR10	Refresh Loader module	IGFASR10	IGFASR10			SVC			
IGFASR2C	Instruction Retry Execution module, phase 1	IGFASR2C	IGFASR2C			SVC			
IGFASR2D	Main Storage Scan module	IGFASR2D	IGFASR2D			SVC			
IGFASR20	PDAR Termination Analysis module	IGFASR20	IGFASR20			SVC			
IGFASR3C	Instruction Retry Execution module, phase 2	IGFASR3C	IGFASR3C			SVC			
IGFN0000	MCH Resident Nucleus module	IGFNUC00	IGFNUC00			NUC			
IGFN0001	Console/SYSRES Clear Channel routine	IGFASR0A	IGFASR0A			NUC SVC			
IGFN0002	MCH Termination routine	IGFASR0A	IGFASR0A			NUC SVC			
INT025A	routine in the I/O Supervisor that returns a request element to the free list	IEAQFX00 (IEAQFX and IECIOS macros)	IEAQFX00	EP	GC	NUC			
INTEXTRN	Second CPU Interruption Analysis routine	IEAQFX00	IEAQFX00	IH		NUC			
INTMLFAL	Second CPU Recovery Management System Interface routine	IEAQFX00	IEAQFX00	IH		NUC			

• Table 14-1. Module Directory (Part 11 of 11) (See legend before Part 1)

Entry Point Name	Name of Routine, Control Block, Table, Transient Area	Module Name	Control Section Name	PLM References		Lib.	If SVC Routine		
				Section	Chart ID		Type	Macro Instruction	SVC Instr.
IORGSW	I/O Switch (in I/O First-Level Interruption Handler)	IEAQNU02	IEAQNU02			NUC			
LINKDCB	data control block (DCB) for the SYS1.LINKLIB data set	IEAQBK00	IEAQBK00			NUC			
LINKDEB	data extent block (DEB) for the SYS1.LINKLIB data set	IEAQBK00	IEAQBK00			NUC			
OVLALD02	SEGLD Processor routine	IEWSWOVR	IEWSWOVR	CS	CI	LINK			
RMBRANCH	GETMAIN/FREEMAIN routines (branch e.p.)	IEAQGM00	IEAQGM00	MSS	DA	NUC	1		
SECMCI	SER0 routine System/360, model 40 System/360, model 50 System/360, model 65 System/360, model 75	 IFBSR040 IFBSR050 IFBSR065 IFBSR075	 IFBSR040 IFBSR050 IFBSR065 IFBSR075	 IH IH IH IH	 AM AM AM AM	 LINK LINK LINK LINK			
START	Initial Program Loading routine	IEAIPL00	IEAIPL						
SVCDCB	data control block (DCB) for the SYS1.SVCLIB data set	IEAQBK00	IEAQBK00			NUC			
SVCDEB	data extent block (DEB) for the SYS1.SVCLIB data set	IEAQBK00	IEAQBK00			NUC			
TABLDL TAHFETCH	Transient Area Fetch routine	IEAQTR33	IEAQTR00	IH	AE	NUC			
TAIOB1 TAIOB2 TAIOBn TAIOBn+1	Transient area I/O blocks (IOBs) and associated transient areas	IEAQBK00	IEAQBK00			NUC			
TASEARCH	Transient Area XCTL routine	IEAQTR33	IEAQTR00	CS	CC	NUC			
TATABCK	Transient Area Availability Check routine	IEAQTR33	IEAQTR00	IH	AD	NUC			
TAXEXIT	Transient Area Exit routine	IEAQTR33	IEAQTR00	EP	GD	NUC			
TAXRETRY	Transient Area XCTL routine	IEAQTR33	IEAQTR00	CS	CC	NUC			
TESTDSP	Task Removal routine	IEAQFX00	IEAQFX00	IH		NUC			
TRDISP	Trace routine (e.p. for the Dispatcher)	IEAQTRCE	IEAQTRCE		GG	NUC			
TREX	Trace routine (e.p. for Ext. FLIH)	IEAQTRCE	IEAQTRCE		AI	NUC			
TRIO	Trace routine (e.p. for I/O FLIH)	IEAQTRCE	IEAQTRCE		AK	NUC			
TRPI	Trace routine (e.p. for PC FLIH)	IEAQTRCE	IEAQTRCE		AF	NUC			
TRSVC	Trace routine (e.p. for SVC FLIH	IEAQTRCE	IEAQTRCE		AA	NUC			
USERORG	SVC table (start of user-assigned SVC numbers)	IEAQBK00	IEAQBK00	IH		NUC			

•Table 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 1 of 10)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
ABDUMP routine		
"resident" module	IGC0A05A	HH
ABDUMP1	IGC0005A	HH
ABDUMP2	IGC0105A	HH
ABDUMP3	IGC0205A	HH
ABDUMP4	IGC0305A	HH
ABDUMP5	IGC0405A	HH
ABDUMP6	IGC0505A	HH
ABDUMP7	IGC0605A	HH
ABDUMP8	IGC0705A	HH
ABDUMP9	IGC0805A	HH
ABDUMP11	IGC0B05A	HH
ABEND routine		
ABEND1	IGC0001C	HI
ABEND2	IGC0401C	HJ
ABEND3	IGC0A01C	HK-HM
ABEND4	IGC0101C	HN
ABEND5	IGC0201C	HO-HP
ABEND6	IGC0301C	HQ-HR
ABTERM routine	IEA0AB00 IEA0AB01	HE-HF HE-HF
ABTERM Prologue routine	IEA0PL00	HG
Attach routine	IGC042	BA-BC
Attention routine	IEEBA1	(Fig. 7-1)
BLDL routine	IECPBLDL	None
CDEXIT routine	CDDESTRY CDEXIT CDHKEEP	GF GF GF
Channel-Check Handler routine	CATNIP	None
CHAP routine	IGC044	BE-BF
	IGC044+12	
Checkpoint routine	IGC0006C	
Checkpoint Housekeeping 1 routine	IGC0006C	FA
Checkpoint Housekeeping 2 routine	IGC0106C	FB

•Table 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 2 of 10)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
Checkpoint Housekeeping 3 routine	IGC0206C	FC
Check I/O routine	IGC0506C	FD
Preserve 1 routine	IGC0A06C	FE
Preserve 2 routine	IGC0D06C	FE
Checkmain 1 routine	IGC0F06C	FF
Checkmain 2 routine	IGC0G06C	FF
Checkmain 3 routine	IGC0H06C	FG
Resume I/O routine	IGC0N06C	FH
Checkpoint Exit routine	IGC0Q06C	FI
Checkpoint Message Module	IGC0S06C	FI
Communications Task External Interruption Handler routine	IEEBC1PE	(Fig. 7-1)
Communications Task External Processor routine	IGCXL07B	(Fig. 7-1)
Communications Task Initialization routine	IEECVCTI	(Fig. 7-1)
Communications Task Misc. Lookup Services routine	IEEVRFRX	None
Communications Task Reply Processor routine	IGC1203D	None
Communications Task request block (RB)	IEECVPRB	None
Communications Task Router routine	IGC0007B	(Fig. 7-1, 7-2)
Communications Task TCB	IEECVTCB	None
Communications Task Unit Control Tables	IEECVUCM	None
Communications Task Wait routine	IEECVCTW	(Fig. 7-1, 7-2)
Communications Task 1052 Console Open/Close routine	IGC0I07B	(Fig. 7-1, 7-2)
Communications Task 1052 Console Processor routine	IGC0107B	(Fig. 7-1, 7-2)
Communications Task 1443 Printer Open/Close routine	IGC2I07B	(Fig. 7-1, 7-2)
Communications Task 1443 Printer Processor routine	IGC2107B	(Fig. 7-1, 7-2)
Communications Task 2540 Console Open/Close routine	IGC1I07B	(Fig. 7-1, 7-2)
Communications Task 2540 Console Processor routine	IGC1107B	(Fig. 7-1, 7-2)
Communications vector table (CVT)	IEACVT	None
Console Device Support routines		
Asynchronous Error routine	IEECVAE	IK

• Table 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 3 of 10)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
Command Format Order Programs (not executable)		
RELEASE (A)	IEECVAOP	None
BRDCST (RJE)	IEECVBOP	None
CANCEL (C)	IEECVCOP	None
DISPLAY (D)	IEECVDOP	None
RESET (E)	IEECVEOP	None
MODIFY (F)	IEECVFOP	None
HOLD (H)	IEECVHOP	None
MSG (RJE)	IEECVIOP	None
CENOUT (RJE)	IEECVJOP	None
SHOW (RJE)	IEECVKOP	None
LOG (L)	IEECVLOP	None
MOUNT (M)	IEECVMOP	None
START (RJE)	IEECVNOP	None
USERID (RJE)	IEECVOOP	None
STOP (P)	IEECVPOP	None
STOP (RJE)	IEECVQOP	None
REPLY (R)	IEECVROP	None
START (S)	IEECVSOP	None
SET (T)	IEECVTOP	None
UNLOAD (U)	IEECVUOP	None
VARY (V)	IEECVVOP	None
WRITELOG (W)	IEECVWOP	None
HALT (Z)	IEECVZOP	None
Command List Order Program Option 2 (not executable)	IEECVOP2	None
Display Control Module (not executable)	IEECVDCM	None
Display routine	IEECVDR1	IE
I/O (Part 1) routine	IEECVDR2	IF-IG
I/O (Part 2) routine	IEECVDR3	IH
Open/Close routine	IEECVOCG	IJ
Options routine	IEECVDR4	II

• Table 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 4 of 10)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
Options routine	IEECVDR5	II
Unit Status Order Program Option 1 (not executable)	IEECVOP1	None
2250 Processor routine	IEECVDP1	IA-ID
2250 Processor routine	IEECVDP2	
Console/SYSRES Clear Channel routine	IGFN0001	None
Contents Supervision Common Subroutines		
entry points for search phase	CDADVANS	CA
	CDCONTRL also called IEAQCS02	CA
entry point for scheduling phase	CDEPILOG also called IEAQCS03	CB
entry point for the ATTACH macro instruction	IEAQCS01	CA
entry point for the LINK macro instruction	IGC006	CA
entry point for the XCTL macro instruction	IGC007	CC
entry point for the LOAD macro instruction	IGC008	CC
entry point for the SYNCH macro instruction	IGC012	CB
CPU Analysis module	IGFASR0B	None
Data control block (DCB) for the SYS1.LINKLIB data set	LINKDCB	None
Data control block (DCB) for the SYS1.SVCLIB data set	SVCDCB	None
Decimal Simulator routines for Model 91		
Add/Subtract/Zero-and-Add Decimal routine	DECASP	IN,IO
Analyzer/End routine	DECD0 DECNEND	IR
Compare Decimal routine	DECCP	IM
Divide Decimal routine	DECDP	IQ
Multiply Decimal routine	DECMP	IP
Simulator Control routine	DECENT	IL
Delete routine	IGC009	CE
Dequeue routine	IGC048	BP-BQ
Dequeue TCB routine	IEAQDQTCB	HA,HO
Detach routine	IGC062	BI
Dispatcher	IEA0DS	GG

• Table 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 5 of 10)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
End-of-Task (EOT) routine	EOT	HA
Enqueue routine	IGC056	BO
ENQ/DEQ Purge routine	IEAQEQ01	None
Erase Phase routine	IEAQERA	HA
Error Check Circuitry Verfication module	IGFASR1D	None
EXCP Supervisor	IECXCP	None
Exit routine	IGC003	GB-GC
External First-Level Interruption Handler	IEAQEX00	AI-AJ
Extract routine	IGC040	BH
branch entry point	IGC040+8	BH
First CPU Signal routine	FLASH	None
FREEMAIN routine		
branch entry point	FMBRANCH	DA
branch entry point to free a region	FREEPART	DB
entry point for S-form FREEMAIN macro instruction	IGC005	DA
GETMAIN routine		
branch entry point to allocate a region	GETPART	DB
branch entry point	GMBRANCH	DA
entry point for S-form GETMAIN macro instruction	IGC004	DA
GETMAIN/FREEMAIN routines		
entry point for R-form GETMAIN or FREEMAIN macro instruction	IGC010	DA
branch entry point	RMBRANCH	DA
Identify routine	IGC041	CD
Initial Program Loading (IPL) routine	START	None
Instruction Retry Analysis module, phase 1	IGFASR0C	None
Instruction Retry Analysis module, phase 2	IGFASR1C	None
Instruction Retry Execution module, phase 1	IGFASR2C	None
Instruction Retry Execution module, phase 2	IGFASR3C	None
I/O block (IOB) for the I/O Supervisor transient area	IEAERRTA	None
I/O First-Level Interruption Handler	IEAQI000	AK-AL
I/O Interruption Supervisor	IECINT	None

Table 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 6 of 10)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
I/O Supervisor transient area	IEAERWA	None
I/O switch (in I/O FLIH)	IORGSW	None
Job File Control Blocks (JFCBs) for the log data sets	IEEVLOGJ	None
link pack area queue	IEAQLPAQ	None
Log and WRITELOG Post routine	IEE1103D	(Fig. 7-3)
Main Storage Scan module	IGFASR2D	None
Master Scheduler Initialization routine	IEEVIPL	(Fig. 7-3)
Master Scheduler resident table	IEEMSER	None
Master Scheduler TCB	IEAMSTCB	None
MCH Error Recorder module	IGFASR01	None
MCH Resident Nucleus module	IGFN0000	None
MCH Termination routine	IGFN0002	None
Nucleus Initialization Program (NIP)	IEANIP4	None
Overlay Supervisor		
nonresident module	IEWSZOVR	CI
resident module		
entry point for SEGLD or SEGWT macro instruction	IGC037	CI
entry point for a branch instruction or CALL macro instruction	IGC045	CI
PDAR Termination Analysis module	IGFASR20	None
Post routine		
branch entry point for I/O Supervisor routines	IEAOPT01	BM
branch entry point for I/O Supervisor routines and for supervisor routines	IEAOPT02	BM
entry point for the POST macro instruction	IGC002	BM
branch entry point for supervisor routines	IGC002+6	BM
Program Check First-Level Interruption Handler	IEAQPK00	AF-FH
Program Fetch routine		
entry point for the Overlay Supervisor (IEWSZOVR)	IEWFBOSV	CF
entry point for the Transient Area Fetch routine	IEWFTRAN	CF
entry point for Contents Supervison common subroutines	IEWMSEPT	CF
Program Fetch Channel-End Appendage routine	FTCE01	CG

• Table 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 7 of 10)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
Program Fetch PCI Appendage routine	FTPCI01	CG
Purge Timer routine	IEAQPGTM	HB
QCB queues, origin of	IEAQQCB0	None
Refresh Clear Channel module	IGFASR1A	None
Refresh Loader module	IGFASR10	None
Reply Purge routine (also called the WTOR Purge routine)	IEECVPRG	None
Release Loaded Programs routine	IEAQABL	HD
Release Main Storage routine	IEAQSPET	HC
Restart Routines	IGC0005B	
Restart Housekeeping 1	IGC0005B	FJ
Restart Housekeeping 2	IGC0105B	FJ
Repmain 1 routine	IGC0505B	FK
Repmain 2 routine	IGC0605B	FK
Repmain 3 routine	IGC0705B	FL
Repmain 4 routine	IGC0805B	FL
Repmain 5 routine	IGC0905B	FM,FN
REP I/O-JFCB Processor 1	IGC0G05B	FO
REP I/O-JFCB Processor 2	IGC0I05B	FO
REP I/O-Mount/Verify 1 (Non Direct-Access) routine	IGC0K05B	FP
REP I/O-Mount/Verify 2 (Direct Access) routine	IGC0M05B	FQ
REP I/O-SYSIN/SYSOUT Data Set Processor 1	IGC0N05B	FR
REP I/O-SYSIN/SYSOUT Data Set Processor 2	IGC0Q05B	FR
REP I/O-Data Set Processor 1	IGC0P05B	FS
REP I/O-Data Set Processor 2	IGC0R05B	FT
REP I/O-Access Method-Disposition routine	IGC0T05B	FU
Restart Exit routine	IGC0V05B	FU
Rollout/Rollin module	IEAQRORI	DC-DJ
Second CPU Interruption Analysis routine	INTEXTRN	None
Second CPU Recovery Management System Interface routine	INTMLFAL	None
secondary communications vector table	IEABEND	None
SEGLD Processor routine	OVLALD02	CI

• Table 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 8 of 10)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
SER0 routine		
resident module	IEAMCH00	AM
nonresident modules (for System/360 Models 40, 50, 65, 75)	IFBSER00 SECMCI	AM AM
SER1 routine (for System/360 Models 40, 50, 65, 75, 91)	IEAMCH00	AN,AO,AP
Set Status routine	IGC079	
SPIE routine	IGC014	BJ
STAE Service routine	IGC0060	BX
ABEND/STAE Interface 1 routine	IGC0B01C	BY
ABEND/STAE Interface 2 routine	IGC0C01C	BZ
ABEND/STAE Interface 3 routine	IGC0D01C	B0
ABEND/STAE Interface 4 routine	IGC0E01C	B0
Stage 1 Exit Effector	IGC043	BR
Stage 2 Exit Effector	IEA0EF00	BS
Stage 3 Exit Effector		
entry points for the Dispatcher	ERFETCH IEA0EF03	BU BT
entry point for an I/O error-handling routine	IECXTLER	BU
STIMER routine	IGC047	EB
Storage Protect Feature Test Module	IGFASR0D	None
SVC First-Level Interruption Handler	IEAQSC00	AA-AB
SVC Purge routine	IGC016	None
SVC Second-Level Interruption Handler	IEAQTR00	AC
SVC table		
start of IBM-assigned SVC numbers	IBMORG	None
start of user-assigned SVC numbers	USERORG	None
System Analysis module	IGFASR00	None
System error TCB (associated with I/O Supervisor transient area	IEAERTCB	None
System Quiesce routine	IECIWTST	HT
Task Removal routine	TESTDSP	None
Task Switching routine	IEA0DS02	BV-BW

Table 14-2. Directory of Entry Point Names and Flowchart Indentifications (Part 9 of 10)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
TESTRAN Interpreter		
entry point for SVC 61 instruction	IGC061	None
entry point for the Overlay Supervisor (IEWSZOVR)	IEGHTOVL	None
Time routine	IGC011	EA
Timer Second-Level Interruption Handler		
branch entry point	IEAQTD00	EE
entry point for the Dispatcher	IEAQTD01	EE
branch entry point	IEAQTE00	EE
entry point for the External First-Level Interruption Handler	IEA0TI00	ED
Trace routine		
entry point for the Dispatcher	TRDISP	None
entry point for the External First-Level Interruption Handler	TREX	None
entry point for the I/O First-Level Interruption Handler	TRIO	None
entry point for the Program Check First-Level Interruption Handler	TRPI	None
entry point for the SVC First-Level Interruption Handler	TRSVC	None
Transient Area Availability Check routine	TATABCK	AD
Transient area control table	IEAQTAQ IEAQTAQ1	None None
Transient Area Exit routine		
entry point for the Exit routine	IEAQTR01	GD
entry point for the common subroutines of Contents Supvsn.	TAXEXIT	GD
Transient Area Fetch routine		
entry point to perform BLDL and fetch	TABLDL	AE
entry point to perform fetch only	TAHFETCH	AE
Transient area fetch TCBs	IEATCB1 IEATCB2 IEATCBn IEATCBn+1	None None None None
Transient area I/O Blocks (IOBs) and associated transient areas	TAIOB1 TAIOB2 TAIOBn TAIOBn+1	None None None None
Transient Area Refresh routine	IEAQTR02	GE

Table 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 10 of 10)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
Transient Area XCTL routine	IEAQTR03 TASEARCH TAXRETRY	CC CC CC
TTIMER routine	IGC046	EC
Type-1 Exit routine	IEA0XE00	GA
Type-1 SVC switch (in SVC First-Level Interruption Handler)	IEATYPE1	None
Validity Check routine	IEA0VL00	None
Wait routine	IGC001	BK
Write-to-Log routine	IEEVL03F	(Fig. 7-3)
Write-to-Operator routine	IGC0003E	(Fig. 7-2)
WRITELOG Available Log Data Set routine	IEEVLOUT	(Fig. 7-3)
WRITELOG Dispatch routine	IEEVLDSP	(Fig. 7-3)
WRITELOG Get Region routine	IEEPLDSP	None
WRITELOG Log Initialization routine	IEEVLIN	(Fig. 7-3)
WRITELOG Log Writer routine	IEEVLWTR	(Fig. 7-3)
WRITELOG Master Wait routine	IEEVWAIT	(Fig. 7-3)
WRITELOG Open Device routine	IEEVLOPN	(Fig. 7-3)
WTOR Purge routine (also called the Reply Purge routine)	IEECVPRG	None

• Table 14-3. Table of Routines Invoked by SVC Instructions (Part 1 of 2)

SVC Instruction	Name of Routine
SVC 0	EXCP Supervisor (in the I/O Supervisor)
SVC 1	Wait routine
SVC 2	Post routine
SVC 3	Exit Routine
SVC 4	GETMAIN routine
SVC 5	FREEMAIN routine
SVC 6	Contents Supervision common subroutines (entry point for the LINK macro instruction)
SVC 7	Contents Supervision common subroutines (entry point for the XCTL macro instruction)
SVC 8	Contents Supervision common subroutines (entry point for the LINK macro instruction)
SVC 9	Delete routine
SVC 10	GETMAIN/FREEMAIN routines
SVC 11	Time routine
SVC 12	Contents Supervision common subroutines (entry point for the SYNCH macro instruction)
SVC 13	ABEND routine (ABEND1)
SVC 14	SPIE routine

SVC 16	SVC Purge routine

SVC 18	BLDL routine
SVC 19	Open routine
SVC 20	Close routine
SVC 34	Log and WRITELOG Post routine
SVC 35	Write-to-Operator routine
SVC 36	Write-to-Log routine
SVC 37	Overlay Supervisor resident module (entry point for SEGLD or SEGWT macro instructions)

SVC 40	Extract routine
SVC 41	Identify routine
SVC 42	Attach routine
SVC 43	Stage 1 Exit Effector
SVC 44	CHAP routine

• Table 14-3. Table of Routines Invoked by SVC Instructions (Part 2 of 2)

SVC Instruction	Name of Routine
SVC 45	Overlay Supervisor resident module (entry point for branch instruction or CALL macro instruction)
SVC 46	TTIMER routine
SVC 47	STIMER routine
SVC 48	DEQ routine

SVC 51	ABDUMP routine (ABDUMP1)
SVC 52	Restart routine

SVC 56	ENQ routine

SVC 60	STAE Service routine
SVC 61	TESTRAN Interpreter (entry point for TTSAV macro instruction)
SVC 62	Detach routine
SVC 63	Checkpoint routine

SVC 72	Communications Task Router routine
SVC 79	Set Status routine
Note: Only those routines that are used by the supervisor are included in this list.	

ROUTINE SYNOPSES

Each major routine used by the supervisor is briefly described.

ABDUMP routine: (Chart HH) Displays control blocks, programs, and dynamically acquired main storage belonging to a specific task, as specified by input parameters. Is invoked through a SNAP macro instruction either by the ABEND routine (ABEND3) during an abnormal termination, or by a user routine at any time.

ABEND routine: (Charts HI-HS) Invokes the ABEND/STAE interface routine if STAE processing is indicated. Frees the control blocks, main storage, and other resources used by a terminating task and its incomplete subtasks. Depending on the type of ABEND request, terminates either a specific task and its incomplete subtasks, or all tasks of a job step. At the caller's option (if possible), invokes the ABDUMP routine to display resources belonging to the terminating task, its direct ancestors, and its descendants. Branches to the System Quiesce routine if any of three conditions occurs: the task specified for termination is in "must complete" status, the task specified for termination is a system task, or the ABEND routine is reentered as an invalid recursion.

ABTERM routine: (Charts HE-HF) Schedules execution of the ABEND routine. Is used by system routines that wish to terminate a task other than their own. Also used by type-1 SVC routines, which are not permitted to issue an SVC instruction, and which therefore cannot directly invoke the ABEND routine.

ABTERM Prologue routine: (Chart HG) Performs housekeeping functions in preparation for entry to the ABTERM routine after a program interruption. Housekeeping includes obtaining the address of the TCB for the task to be terminated, and setting a completion code to indicate the cause of the program check.

Attach routine: (Charts BA-BC) Obtains storage space for a new TCB for the subtask to be attached. Places in the new TCB information needed to control the subtask. Allocates to the subtask subpools of main storage belonging to its parent or attaching task. Places the address of the new TCB on two lists: the subtask queue of its parent task, and the TCB queue used by the Dispatcher. Schedules supervisor linkage to the common subroutines of Contents Supervision. The subroutines will locate, fetch (if necessary), and schedule execution of the first program of the new subtask.

Attention routine: (Figure 7-1) Receives control from the I/O First-Level Interruption Handler after an operator-caused interruption (when the REQUEST key of a 1052 Printer-Keyboard or the START key of a card reader is pressed). Posts the appropriate ECB of the Communications Task.

BLDL routine: Causes member addresses and optional information from a partitioned data set directory to be placed in a specified list previously constructed in main storage.

CDEXIT routine: (Chart GF) Determines if there is an outstanding request for use of a recently completed module. If so, schedules reentry to the module for a waiting requestor. If there is no outstanding request for the module, the routine tests the module's attributes. If the module is in the link pack area, control is returned immediately to the caller. If the module is in the job step's region, and is either reenterable or reusable, the routine sets the "release" flag in the module's CDE and the "purge" flag for the job pack queue. (These flags are tested by the GETMAIN routine to determine which module's space may be freed, if needed space is otherwise unavailable.) If the module is neither serially reusable nor reenterable, CDEXIT (via its CDDESTRY subroutine) removes the module's CDE from the job pack queue, and frees the space occupied by the module, its extent list, and its CDEs (major and minor).

Channel-Check Handler (CCH): Available on configurations using the 2860/2870 channels (Model 65 and higher). Receives control from the I/O Supervisor via a branch. Performs two main functions. It places information in the error routine interface bytes so that the appropriate device error routine may retry the channel operation that was being performed when the channel check occurred. The CCH also records environment data regarding the channel failure. This data is later written to the SYS1.LOGREC data set by the I/O Supervisor. (For a full description of the Channel-Check Handler, refer to the publication IBM System/360 Operating System: Input/Output Supervisor, Program Logic Manual, Form Y28-6616.

CHAP routine: (Charts BE-FG) Changes the dispatching priority of a TCB by adding the specified value to the TCB's existing dispatching priority. Validates the new dispatching priority and corrects it if necessary.

Checkpoint routines: (Chart FA-FI) Intercepts a task's I/O requests, copies the task's main storage region into a user-supplied data set, and records the status of data sets, main storage and contents supervision control blocks, and other supervisor information necessary to restart the task's execution at a later time.

Communications Task External Interruption Handler routine: (Figure 7-1) Receives control from the External First-Level Interruption Handler after an operator-caused interruption (when the INTERRUPT key on the system control panel is pressed). Posts the appropriate ECB of the Communications Task.

Communications Task External Processor routine: (Figure 7-1) Switches control from the principal console device to the alternate console device, and vice versa.

Communications Task Initialization routine: (Figure 7-1) Initializes tables used for the Communications Task. Is executed when nucleus initialization is performed.

Communications Task Miscellaneous Look-Up Services routine: Provides Communications Task routines with the addresses of tables, or pointers to information in the tables. The tables include the communications vector table (CVT), TCBs, RBs, task I/O tables (TIOTs), and unit control blocks (UCBs).

Communications Task Reply Processor routine: Processes replies given by an operator in response to program messages written via the WTOR macro instruction. Moves each reply to a buffer and posts the appropriate ECB.

Communications Task Router routine: (Figure 7-1, 7-2) Selects the service to be performed after the posting of an ECB for the Communications Task. Routes control to the appropriate Communications Task processor routine.

Communications Task unit control tables: Describe characteristics of the I/O devices that perform the Communications Task. Also contain ECBs for the Communications Task.

Communications Task Wait routine: (Figure 7-1, 7-2) Waits for an ECB to be posted, then issues an SVC-72 instruction to cause entry to the Communications Task Router routine.

Communications Task Open/Close routines: (Figure 7-1, 7-2) Device-dependent routines that cause data sets for specific devices to be opened and closed. The devices are the 1052 Console, the 2540 Console, and the 1443 Printer.

Communications Task Processor routines: (Figure 7-1, 7-2) Device-dependent routines that direct activity on specific devices by initiating I/O activity, managing data buffers, and responding to I/O completion and error conditions. The devices are the 1052 Console, the 2540 Console, and the 1443 Printer.

Console Device Support routines (Figure 11-1, Charts IA-IK): Added to SVC 72 when the 2250 Model 1 Display Unit is used as a system operators console. These routines perform read and write operations to display system and problem program messages to the operator, cause a hard copy of the messages to be produced, receive commands issued by the operator, process light pen attentions, and display unit status and command formats.

Contents Supervision common subroutines: (Charts CA-CC) Locate, fetch, and schedule execution of a specified module. If the module is in main storage and is available for use, schedule its execution. If the module is not in main storage, or is non-reusable, locate the module. They search the specified private library, the link library, or the job library; then invoke the Program Fetch routine to load the module. Finally, they schedule execution of the module. If the module is being loaded or is serially reusable and is in use, they place the current SVRB in a wait condition, and queue it to a list of SVRBs waiting for the module.

Decimal Simulator routines: (Charts IL-IR) Perform decimal arithmetic instructions on the Model 91. After receiving control from the Program First-Level Interruption Handler, interpret the instruction, check it for validity, and perform operations that simulate the execution of the instruction.

Delete routine: (Chart CE) Locates the CDE for the specified module via a search of the task's load list. If there are no outstanding LOAD requests for the module, removes the module's load list element from the load list and frees the storage space it occupies. Tests the use/responsibility count in the module's CDE. If there are no outstanding requests for the module's use, branches to CDHKEEP in the CDEXIT routine to test the module's attributes. According to the attributes, CDHKEEP returns control immediately to the caller, or frees the module's storage areas, or sets "release" and "purge" flags for use by the GETMAIN routine (see CDEXIT routine).

Dequeue routine: (Chart BP) Updates the resource queues by removing and freeing the queue element that represents the request for the resource whose use is now complete. For the next requester represented on the QEL queue, reduces the wait count in its SVRB and tests if the requestor is ready. Determines if a readied requestor can replace the caller as the next-to-be dispatched routine. Makes this determination via a branch to the Task Switching routine. If no readied requestor's task is of higher priority than the caller's, returns control to the caller. Otherwise, returns control to the readied requestor, whose resource(s) are now available.

If the caller is a system routine and specifies the "reset must complete" operand, the current task, previously placed in "must complete" status, is released from that status.

Dequeue TCB routine: (Charts HA, HO) Is invoked by either the EOT routine or ABEND4 during a normal or abnormal termination. Removes a specified TCB from the TCB queue.

Detach routine: (Chart BI) Removes the specified TCB from the TCB queue, and frees the TCB's storage space and the space of any associated problem-program register save area. If the caller supplies an invalid TCB address, the routine branches to the ABTERM routine to schedule abnormal termination of the caller's task. If the specified task is incomplete, the routine branches to the ABTERM routine to schedule abnormal termination of the specified task.

Dispatcher: (Charts GG-GP) Determines the routine to be executed next, restores the contents of saved registers, and loads an old PSW to give control to the routine. As an optional feature, if a task switch is to occur, suspends timing of the previously current task and starts or resumes timing of the task to be given control.

End-of-Task (EOT) routine: (Chart HA) Frees the resources used in performing a successfully completed task. The resources (control blocks, main storage, data sets, modules) are released only if they are not needed by another task.

Enqueue routine: (Chart BO) Creates, if necessary, one or more queue control blocks (QCBs) to represent the requested resource(s), and places them on the resource queues. Depending on the RET parameter that the caller has specified, creates a queue element (QEL) to represent the request, and places it on a QEL queue. If the requested resource is not enqueued for another requestor, returns control to the current requestor, with or without a return code (depending on the RET parameter). If the requested resource is already enqueued for another requestor, either of two functions are performed, depending on the RET parameter: the current requestor is placed in a wait condition, pending the availability of the resource; or control is returned to the current requestor, with a return code that indicates that the resource is not available.

If the caller is a system routine and specifies the "set must complete" operand, the Enqueue routine places the current task in "must complete" status.

ENQ/DEQ Purge routine: Is invoked by ABEND4 to remove from the resource queues requests (QELs and possibly one or more QCBs) belonging to a terminating task. The purge is performed so that routines belonging to other tasks could gain access to the enqueued resource(s), if the task terminated before the DEQ routine could be executed.

Erase Phase routine (also called the Erase routine): (Chart HA) Is invoked by the EOT routine or ABEND4 during a normal or abnormal termination. Removes the specified TCB from its parent's subtask queue, and frees the space occupied by the TCB and any related problem-program register save area. (A similar function is performed by the Detach routine under different circumstances.)

EXCP Supervisor: Is a part of the I/O Supervisor. Given control by the SVC First-Level Interruption Handler, it starts execution of a channel program. It issues a Start I/O instruction, then a Stand-Alone Seek command. The Stand-Alone Seek command moves the access arm of the direct-access device to the seek address contained in the caller's IOB.

Exit routine: (Charts GB-GC) Processing depends on the type of RB associated with the exiting routine, as follows:

1. If the task's current RB is a PRB (see Chart FB), the general register contents are moved from lower main storage (location IEASCSAV) to the TCB. If the PRB is not the last RB on the RB queue, the routine branches to the CDEXIT routine to perform exit processing for the completed module. When CDEXIT returns control, function #5 is performed (see below). If the PRB is the last RB on the RB queue (queued directly from the TCB), the EOT routine is entered to normally terminate the task. When the EOT routine returns control, the Exit routine frees the RB's space and exits to the Transient Area Refresh routine.

2. If the task's current RB is an SVRB (see chart FB), the Exit routine branches to the Transient Area Exit routine to remove (if necessary) the SVRB from a transient area user queue. When control is returned, the register contents originally saved in the SVRB (registers 2-14) and register contents returned by the SVC routine (regs 0, 1, 15) are in the TCB's save area.

Function #5 is then performed (see below).

3. If the task's current RB is an IRB (see Charts FB-FC), the "top" IQE or RQE on the IRB's list of elements is returned to a next-available list. If there is another IQE or RQE queued to the IRB, the routine reinitializes the IRB for reentry to the asynchronous exit routine, and branches to the Dispatcher. But if there is no other IQE or RQE queued to the IRB, the routine moves register contents from the IRB's register save area to the TCB's register save area. Function #5 is then performed (see below).

4. If the task's current RB is an SIRB, it is removed from the system error TCB, the SIRB's "active" bit is reset, and the Transient Area Refresh routine is entered.

5. If the next RB on the task's RB queue is waiting, the routine indicates to the Dispatcher the need for a task switch by placing zero in the "new" TCB pointer. The routine clears the RB's "active" flag and removes the RB from the task's RB queue. If the RB is not a permanent system RB nor an IRB that is still needed,[1] the RB's storage space is freed. Also freed is space used for a related problem-program register save area. The Exit routine then enters the Transient Area Refresh routine.

External First-Level Interruption Handler: (Charts AI-AJ) Saves the caller's register contents in the TCB and the external old PSW in the current RB. Branches to the Trace routine to store information in trace table. Determines whether the interruption was caused by the operator or the timer, by examining the interruption code in the external old PSW. Depending on the cause of the interruption, gives control to either the Timer Second-Level Interruption Handler or the Console Switch routine.

If the multiprocessing feature was selected, after saving the old PSW, determines if a FLIH routine, other than External FLIH, was interrupted. If it was, saves the interruption code and returns control. Otherwise, processes the interruption after setting the supervisor lock byte. Also, determines if the interruption was caused by the second CPU, and, if it was, passes control to the routine indicated in the STMASK byte of the second CPU.

Extract routine: (Chart BH) Moves the contents of selected TCB fields to a specified area of main storage.

FREEMAIN routine: (Chart DB) Frees specified allocated main storage. If request is to free a region, the job pack queue is purged. In this case tasks that are waiting for allocation of a region are made ready and task switch is indicated.

If the multiprocessing feature was selected, branches to the Vary Storage Offline (IFSVRYOF) subroutine to process VQEs which apply to the freed area of main storage.

GETMAIN routine: (Chart DA) Allocates main storage space and builds main storage control blocks, if needed. If the request is for system queue area and there is no available space in this area, expands the supervisor queue area, if possible. If request is not for system queue area, and free space is not available, makes space available by purging those modules in the region's job pack area whose CDEs have "release" flag set. These modules have no outstanding requests for their use and have been so flagged by the CDEXIT routine. If sufficient module space cannot be made available, branches to the ABTERM routine to schedule the abnormal termination of the caller's task.

Identify routine: (Chart CD) Creates a minor CDE to represent the specified embedded entry point to a load module. Queues the minor CDE to the module's major CDE on the appropriate CDE queue.

Initial Program Loading (IPL) routine: Clears main storage and machine registers to correct parity. Sets the storage protection key of main storage to the supervisor protection key. Locates the nucleus data set on the system residence device. Loads into main storage the nucleus and the Nucleus Initialization Program (NIP). Gives control to the NIP.

I/O First-Level Interruption Handler: (Charts AK-AL) Sets the I/O switch (IORGSW) to indicate that an I/O interruption has occurred. Saves current register contents in the current TCB. Saves I/O old PSW in the current RB. Branches to the Trace routine to store pertinent information in the trace table. Branches to the I/O Interruption Supervisor to process the interruption. When control is returned, clears the I/O switch (IORGSW) and enters the dispatcher.

----

[1] An IRB may be retained for use with the same end-of-task exit routine (ETXR) for another task. In this case, its RBUSE count is not zero.

I/O Interruption Supervisor: Is a part of the I/O Supervisor. Given control by the I/O First-Level Interruption Handler (I/O FLIH), it services the I/O interruption. It then returns control to the I/O FLIH.

I/O Supervisor transient area: The area of main storage in which the system error task loads a system I/O error-handling routine.

Job file control blocks (JFCBs) for log data sets: Contain descriptive information about the primary and alternate system log data sets. They are constructed and written on auxiliary storage by job management routines. Each JFCB is loaded in main storage when the DCB with the same ddname is opened.

Link pack area queue (also called the link pack area control queue or LPACQ): Con tains CDEs for modules stored in the link pack area of main storage. The link pack queue and the job pack queue together are called the contents directory.

Log and WRITELOG Post routine: (Figure 7-3) Posts the ECB representing the appropriate command. For log commands also issues a WTL macro instruction.

Machine-Check Handler for Model 65 (MCH/65): Is available only with System/360 Model 65. Receives control via hardware loading of the machine check new PSW. This program consists of a resident routine, and transient modules which reside on the SYS1. SVCLIB data set. It attempts to recover from a machine check interruption.

It first determines if the instruction that was being executed when the machine check occurred can be retried. If instruction retry is possible, the Machine-Check Handler (MCH) attempts reexecution of the interrupted instruction. If, however, instruction retry is not possible, the MCH tries to repair program damage. The program damage may be associated with either a defective storage protection feature (SPF) key or a defective main storage location. The MCH may correct a defective SPF key by issuing a Set Storage Key (SSK) instruction. A main storage location can sometimes be corrected by reloading (refreshing) the module that was being executed when the machine check occurred.

If program damage is repaired, the MCH attempts to retry the interrupted instruction. If the retry is successful, the MCH has recovered completely from the machine check interruption.

If program damage cannot be repaired or instruction retry is unsuccessful, the MCH can either continue partial system operation or place the CPU in the wait state. The choice depends on the type of task that was current at the time of the machine interruption, the number of tasks that are affected, and the extent of the program damage. If limited system operation is possible, the MCH either abnormally terminates the current job step or sets the current task nondispatchable. If even limited system operation is not possible, because a critical system task is permanently damaged, the MCH issues an error message and places the CPU in the wait state.[1]

For a full description of the Machine-Check Handler for Model 65 (MCH/65), refer to the publication IBM System/360 Operating System: Machine-Check Handler, Program Logic Manual, Form Y27-7155.

Master Scheduler Initialization routine: (Figure 7-3) Is performed during nucleus initialization. Places appropriate unit control block name in the unit control table. Constructs an initial list of ECBs for the communications task. Determines which consoles are active. Gives control to the communications task Wait routine.

Master Scheduler resident table: Contains switches and pointers that are used by the Master Scheduler during nucleus initialization.

Nucleus Initialization Program (NIP): Initializes the resident part of the control program and prepares main storage for control program operation. Receives control from the IPL routine via a Load PSW instruction. Initializes nucleus tables, performs general system initialization, and sets up divisions of main storage.

Open routine: A data management routine that completes the specified data control block and prepares the associated data set for processing. Analyses input labels and creates output labels.

Overlay Supervisor (nonresident module): (Chart CI) Directs loading of the specified overlay segment and any segments in its path that are not in main storage. When loading is complete and, the caller has issued a CALL macro instruction or a branch instruction, alters the entry tables of the loaded segments. The alteration permits future branches to the same points in the loaded segments without help from the Overlay Supervisor.

----

[1] The operator may then load the SEREP program in order to format and print diagnostic information from the CPU logout area.

Overlay Supervisor (resident module): (Chart CI) Obtains the address of the segment table for the overlay module. Ensures that the appropriate entry table contains the specified entry point name. Causes supervisor linkage to the nonresident module of the Overlay Supervisor. (The nonresident module was loaded by the common subroutines of Contents Supervision when the root segment of the overlay module was requested.)

Post routine: (Charts BM-BN) Places the caller's post code into the specified ECB; sets the completion bit and clears the wait bit in the ECB. Also decreases by one the RB wait count for the waiting routine. If the new RB wait count is greater than zero, prepares for return of control to the caller. If the new RB wait count is zero, branches to the Task Switching, then prepares for return of control to the caller or the newly readied routine.

Program Check First-Level Interruption Handler: (Charts AF-AH) Saves register contents in program check register save area in lower main storage. If the trace option exists in the system, branches to the Trace routine to store information in the trace table. If the interrupted routine was operating in supervisor state, gives control to the ABTERM Prologue routine. If the interrupted routine was operating in problem-program state, determines if the address of a program interruption element (PIE) is in the current TCB. If a PIE address is not in the TCB, branches to the ABTERM Prologue routine. Otherwise, stores the program old PSW and registers 2-14 in the PIE. If a program interruption control area (PICA) is not in effect or is being used for a previous program interruption, the routine branches to the ABTERM Prologue routine. Otherwise, places the entry point address of the interrupted routine in the program old PSW and branches to a user-written error-handling routine.

If the multiprocessing feature was selected, determines if the interruption was caused by an SSM instruction. If it was, sets the supervisor lock byte if complete enablement is not indicated, and returns control to the interrupted routine. Before processing other types of program interruptions, sets the supervisor lock byte.

Program Fetch routine: (Chart CF-CH) Obtains needed storage space, initializes tables and an extent list, initiates I/O operations, and loads the specified module or overlay segment into main storage. Performs any needed relocation of address constants. Computes the module's relocated entry point address and returns it to the caller. Also returns the address of the module's extent list.

Program Fetch Channel-End Appendage routine: (Chart CG) Determines if all buffers are full and whether the entire module or overlay segment has been loaded. Receives control from and returns control to the I/O Interruption Supervisor.

Program Fetch PCI Appendage routine: (Chart CG) After each PCI interruption, it tests a record in the current RLD buffer. When necessary, it causes a channel-program switch between two-record mode and single-record mode. Such switch is necessary if an RLD or control record does not follow a text record on auxiliary storage. When the last record is being read, posts a fetch ECB. Receives control from and returns control to the I/O Supervisor.

Purge Timer routine: (Chart HB) Is invoked by the EOT routine or ABEND4 during a normal or abnormal termination. Tests a timer queue element (TQE), if one belongs to the terminating task. If the TQE is not on the timer queue, issues a FREEMAIN macro instruction to free the space the TCQ occupies. If, however, the TQE is on the timer queue, the routine branches to the Timer Second-Level Interruption Handler (IEAQTD00) to cancel the interval request and remove the TQE from the timer queue.

Reply Purge routine: Refer to the WTOR Purge routine.

Release Loaded Programs Routine: (Chart HD) Is invoked by the EOT routine or ABEND4 during a normal or abnormal termination. Frees load list elements for the terminating task and reduces the use/responsibility count in each related CDE. Branches to CDHKEEP in the CDEXIT routine to test the reduced use/responsibility count and perform, if necessary, further module cleanup.

Release Main Storage routine: (Chart HC) Is invoked by the EOT routine or ABEND4 during a normal or abnormal termination. Releases main storage exclusively allocated to the terminating task. The task's subpool queue elements (SPQEs) are used to free unshared subpools. The SPQEs are removed from the task's main storage queues and their space is freed. In addition, if the job step task is being terminated, the routine branches to CDDESTRY in the CDEXIT routine. CDDESTRY frees main storage occupied by each module in the job pack area, its extent list, and its CDEs (major and minor).

Restart routine: (Charts FJ-FU) Reads and interprets records from a checkpoint entry to restore a previously executed task to its main storage region, open and reposition its data sets, and restore task control blocks and queues so that it may be restarted within a job step.

Rollout/Rollin module: (Charts DC-DJ) Schedules rollout when an unconditional GETMAIN cannot be satisfied with space from the job step's region; schedules rollin when all space in a borrowed region is freed.

SEGLD Processor routine: (Chart CI) Is a part of the Overlay Supervisor nonresident module. Is attached to operate as a subtask. Scans the segment table. Invokes the Program Fetch routine to load each indicated segment of an overlay module. Posts an ECB for the Overlay Supervisor when all indicated segments have been loaded.

SER0 routine (resident module): (Chart AM) Model-independent part of the SER0 routine. Is entered after a machine check interruption. Saves register contents and other indicative information, halts I/O activity, and causes entry to the appropriate model-dependent part of the routine.

SER0 routine (nonresident module): (Chart AM) Model-dependent modules, one of which is used with the corresponding model of IBM System/360. Collects information about the status of the system at the time of the interruption, writes the information onto the SYS1.LOGREC data set, and places the CPU into a wait state.

SER1 routine: (Charts AN-AP) Model-dependent modules, one of which is used with the corresponding model of IBM System/360. Collects information about the status of the machine at the time of the interruption and writes the information onto the SYS1.LOGREC data set. Then, either causes an abnormal termination of the job step, or places the CPU into the wait state, depending on the severity of the error condition.

Set Status routine: Sets all tasks of the specified job step nondispatchable by setting the TCBPRO flags in their TCBs.

SHOLDTAP routine: Issues a write direct instruction which initiates an external interruption on the second CPU in a multiprocessing system. The STMASK byte indicates the routine that gains control on the second CPU as a result of the interruption.

SPIE routine: (Chart BJ) Places into the caller's TCB an indirect pointer to the specified user error-handling routine. Either locates an existing program interruption element (PIE) or creates a new one, and places its address in the caller's TCB. Then places in the PIE the address of the associated program interruption control area (PICA). The PICA contains the address of the user error-handling routine.

STAE Service routine: (Charts BW-BO) Creates a STAE control block which contains the address of a user-written STAE exit routine and parameter list. When an ABEND is scheduled for a task that has issued STAE, the ABEND routine invokes the ABEND/STAE interface routine, which purges the task's I/O, schedules the STAE exit routine, and returns control to the user at the STAE exit routine address. Upon completion of the STAE exit routine, the ABEND/STAE interface routine either returns control to ABEND to terminate the task or schedules the user-written STAE retry routine.

Stage 1 Exit Effector: (Chart BR) Creates and initializes an interruption request block (IRB) to schedule and control execution of a user exit routine.

Stage 2 Exit Effector: (Chart BS) Starts the scheduling of entry to a user exit routine by placing the specified queue element (IQE or RQE) on the appropriate asynchronous exit queue.

Stage 3 Exit Effector: (Charts BT-BU) Completes the scheduling of a user exit routine. It does this by transferring an IQE or RQE from an asynchronous exit queue to the queue belonging to the appropriate IRB or SIRB. Queues the IRB to the appropriate TCB. Queues the SIRB to the system error TCB. Contains an error fetch sequence (similar to the TA Fetch routine) that causes a needed but unavailable system I/O error-handling routine to be loaded. The error-handling routine is loaded in the I/O Supervisor transient area.

STIMER routine: (Chart EB) Builds and places on the timer queue the elements that represent specified time intervals.

SVC First-Level Interruption Handler: (Charts AA-AB) Saves the caller's register contents. Determines from the SVC table the type of SVC routine to be given control. If a type-1 routine, gives control to the routine. If a type-2, 3, or 4 routine, gives control to the SVC Second-Level Interruption Handler.

SVC Purge routine: Is part of the I/O Supervisor. Is invoked by ABEND1 during an abnormal termination. Removes from system queues the request elements (RQEs) that represent I/O requests issued for the terminating task. Issues a Halt I/O instruction to stop the task's I/O operations.

SVC Second-Level Interruption Handler: (Chart AC) is entered from the SVC First-Level Interruption Handler. Constructs a supervisor request block (SVRB) from previously allocated space and initializes the SVRB. Moves the caller's register contents from lower main storage to the SVRB. Queues the SVRB to the TCB for the caller's task. If a resident (type-2) SVC routine is needed, branches directly to the routine.

If a nonresident (type 3 or 4) SVC routine is needed, determines if the routine is already in a transient area block (TAB). If the routine is in a TAB, places the SVRB on a user queue and branches to the TAB. If the routine is not in a TAB, examines the transient area control table and the user queues to find an available TAB. If it finds an available TAB, places those SVRBs "using" the TAB into a wait condition, places the new SVRB on the user queue for the TAB, makes the new SVRB wait, readies a transient area fetch task to load the SVC routine into the TAB, invokes the Task Switching routine, and branches to the Dispatcher. If, however, an available TAB cannot be found, places the new SVRB into a wait condition, indicates the need for a task switch, and branches to the Dispatcher.

System Quiesce routine: (Chart HT) Is entered from the ABEND routine when severe error conditions are detected. Abnormally terminates the failing task and sets all related tasks nondispatchable. Issues a message to the operator indicating that a CPU wait state has been averted and instructing him to allow the system to quiesce. Returns control to the supervisor thus enabling other tasks that have been scheduled previously to continue processing.

System Error TCB: The system TCB under whose control system I/O error-handling routines are loaded into the I/O Supervisor transient area and then executed.

Task Removal routine: Determines if the current task on the second CPU in a multiprocessing system has been set nondispatchable. If it has, causes the dispatcher to gain control on the second CPU and dispatch a new task.

Task Switching routine: (Charts BV,BW) Determines if a newly readied task, which may be of higher dispatching priority than the current task, should be dispatched in place of the current task. Compares the dispatching priority of the specified ready task with that of the next-to-be-dispatched task. (The address of the TCB for the next-to-be-dispatched task is stored in the "new" TCB pointer, IEATCBP.) If the specified task's priority is higher, places its TCB address into the "new" TCB pointer. If the specified task's priority is lower, makes no change. If the task priorities are equal, places in the "new" TCB pointer the address of the TCB positioned higher on the TCB queue.

In a multiprocessing system, determines if the newly readied task should be dispatched in place of the current task on either CPU. Determines which of the two next-to-be dispatched tasks has the lower dispatching priority, and compares the lower task with the newly readied task. If the newly readied task has a higher priority, places its TCB address into the "new" TCB pointer.

TESTRAN Interpreter: Is the part of the control program that interprets requests for test services. Is invoked by either the common subroutines of Contents Supervision or the Overlay Supervisor if the loaded module or segment is being tested.

Time routine: (Chart EA) Determines the current date and time of day and returns both values to the caller. Places the time of day into register 0 and the date into register 1.

Timer Second-Level Interruption Handler: (Charts ED-EE) Is entered from the External First-Level Interruption Handler after a timer-caused interruption. Determines what action to take by removing and examining the topmost timer queue element (TQE) on the timer queue. May prepare entry to a user-written routine or posts a specified ECB. Resets the interval timer, using the value contained in the new top TQE.

Trace routine: Builds the trace table, a system option. The trace table describes conditions at each SVC interruption, external interruption, program interruption, and at each issuance of a Start I/O instruction, and each execution of the Dispatcher. The Trace routine is invoked by the SVC First-Level Interruption Handler (SVC FLIH), the I/O FLIH, the External FLIH, the Program Check FLIH, and the Dispatcher.

Transient Area Availability Check routine: (Chart AD) Is invoked by the SVC Second-Level Interruption Handler. Examines the transient area control table and the user queues to locate a transient area block that may be overlaid by a SVC routine.

Transient Area Exit routine: (Chart GD) Is invoked by the Exit routine or by the common subroutines of Contents Supervision. Prepares for return of control to the caller of a type-2, 3, or 4 SVC routine. Moves saved register contents from the exiting routine's SVRB to the TCB for the caller's task. For an exiting nonresident routine (type 3 or 4), removes the SVRB from its transient area user queue.

Transient Area Fetch routine: (Chart AE) Is entered when the SVC Second-Level Interruption Handler, or the Transient Area XCTL routine, or the Transient Area Refresh routine determines that a nonresident SVC routine must be loaded. Locates the needed routine, and uses the Program Fetch routine to load the needed routine into the available transient area block. Is controlled by a high-priority system TCB, called a transient area fetch TCB.

Transient Area Refresh routine: (Chart GE) Determines if an SVC routine that occupied a transient area block but was overlaid should be reinstated. It so, schedules reloading of and entry to the routine.

Transient Area XCTL routine: (Chart CC) Prepares for entry to another module of a multi-module (type-4) SVC routine. Rein itializes the appropriate SVRB. If the needed module is in a transient area block, schedules entry to it. Otherwise, locates (if possible) an available transient area block and schedules loading of and entry to the module. If a transient area block is not available, places the SVC routine's SVRB in the wait condition, queues the SVRB to a queue of waiting SVRBs, and indicates the need for a task switch.

TTIMER routine: (Chart EC) Determines and places into register 0 the time remaining in a previously requested time interval. Optionally cancels a previously requested interval.

Type-1 Exit routine: (Chart GA) Routes control to the interrupted routine or to the Dispatcher. Restores saved register contents and returns control to the interrupted routine, if the need for a task switch is not indicated. (A task switch is not indicated if the addresses in the two TCB pointers, IEATCBP and IEATCBP+4, are equal.) If the need for a task switch is indicated, moves saved register contents to the current TCB, and gives control to the Dispatcher to perform the task switch.

Validity Check routine: Validates user-supplied addresses. Checks addresses for full-word boundary alignment, determines ifthe addresses lie within the limits of main storage, and tests if the addresses specify storage areas whose storage protection keys match the protection key in the caller's TCB.

Vary Storage Offline Subroutine (IFSVRYOF): Processes requests to remove an area from available main storage in a multiprocessing system. Alters the FBQE(s) and marks the area unavailable in the FSSEMAP.

Wait routine: (Charts BK-BL) Determines if any of the specified events have occurred. If all have occurred, prepares for return of control to the caller. If all the specified events have not occurred, makes the caller wait by placing the appropriate wait count into the caller's RB. Then indicates the need for a task switch.

Write-to-Log routine: (Figure 7-3) Schedules servicing of a request to write a message onto the system log. Places the message into a log element and adds the element to a chain of log elements. Posts the system log ECB to signify receipt of the message.

Write-to-Operator routine: (Figure 7-2) Prepares buffers and posts the communications task ECB.

WRITELOG Available Log Data Set routine: (Figure 7-3) Sets a bit in the log control area to indicate availability of either the primary or alternate system log data set.

WRITELOG Dispatch routine: (Figure 7-3) Initializes a job file control block (JFCB) and a data set block (DSB) for the specified primary or alternate log data set. Places both the JFCB and DSB on the job queue.

WRITELOG Get Region routine: Obtains the region to be used for the log dispatcher task.

WRITELOG Log Initialization routine: (Figure 7-3) Searches the catalog to locate the two log data sets. Creates a data control block (DCB) for and opens the primary log data set. Initializes the log control area.

WRITELOG Log Writer routine: (Figure 7-3) Writes messages onto the system log data set. In response to a WRITELOG command, causes the appropriate log data set to be transferred to an output device by a system output writer.

WRITELOG Master Wait routine: (Figure 7-3) Passes control to the Log Writer routine when the system log ECB is posted.

WRITELOG Open Device routine: (Figure 7-3) Opens the specified system output-writer data set.

WTOR Purge routine (also called Reply Purge routine): Is invoked by the EOT routine or ABEND1 during a normal or abnormal termination. Disposes of outstanding messages and replies to messages by removing elements from the buffer queue and the reply queue.

- ABDUMP nondispatchability flag
  - clearing the 203
- ABDUMP parameter list
  - format of 295
- ABDUMP "resident" module
  - description of 194
  - entry point name of 471
  - module name for 465
- ABDUMP routine
  - description of 193
  - flowchart of 427
  - synopsis of 485
- ABDUMP1
  - description of 194
  - entry point name of 471
  - module name for 466
- ABDUMP11
  - description of 199
  - entry point name of 471
  - module name for 468
- ABDUMP2
  - description of 196
  - entry point name of 471
  - module name for 467
- ABDUMP3
  - description of 196
  - entry point name of 471
  - module name for 467
- ABDUMP4
  - description of 196
  - entry point name of 471
  - module name for 467
- ABDUMP5
  - description of 197
  - entry point name of 471
  - module name for 467
- ABDUMP6
  - description of 198
  - entry point name of 471
  - module name for 467
- ABDUMP7
  - description of 199
  - entry point name of 471
  - module name for 467
- ABDUMP8
  - description of 200
  - entry point name of 471
  - module name for 468
- ABDUMP9
  - description of 200
  - entry point name of 471
  - module name for 468
- ABEND routine
  - description of 201
  - synopsis of 483
- ABEND1
  - description of 203
  - entry point name of 471
  - flowchart(s) of 428
  - module name for 466
- ABEND 2
  - description of 203
  - entry point name of 471
  - flowchart(s) of 429
  - module name for 467
- ABEND3
  - description of 205
  - entry point name of 471
  - flowchart(s) of 430-432
  - module name for 468
- ABEND4
  - description of 209
  - entry point name of 471
  - flowchart(s) of 433
  - module name for 467
- ABEND5
  - description of 214
  - entry point name of 471
  - flowchart(s) of 434-435
  - module name for 467
- ABEND6
  - description of 220
  - entry point name of 471
  - flowchart(s) of 436-438
  - module name for 467
- ABEND/STAE Interface routines
  - description of 76-80
  - entry point names of 478
  - flowcharts of 355-357
  - module name(s) for 468
- Abnormal dump
  - (see dumping selected areas of main storage)
- Abnormal termination 185
- ABTERM flag
  - meaning of 205
- ABTERM Prologue routine
  - description of 192
  - entry point name(s) of 471
  - flowchart(s) of 426
  - module name for 462
  - synopsis of 483
- ABTERM routine
  - description of 186
  - entry point name(s) of 471
  - flowchart(s) of 424-425
  - module name for 462
  - synopsis of 483
- Access - Method Disposition routines
  - description of 168
  - entry point name of 477
  - flowchart of 403
  - module name for 468
- AEQA
  - (see RQE queue)
- AEQJ
  - (see IQE list)
- Alias processing
  - description of 90
- Allocated queue element (AQE)
  - construction of 139
  - format of 286
  - normal release of 145
- AQE

(see allocated queue element)
ASIR
(see ABEND/STAE Interface routines)
Asynchronous exit queues
description of 70
Asynchronous exit routine
(see user exit routine or task asynchronous exit routes)
Attach routine
description of 39
entry point name(s) of 471
flowchart of 331-334
module name for 467
synopsis of 483
Attention routine
entry point name of 471
flowchart of 156
module name for 463
synopsis of 483
AUTOPRG
(see Autopurge routine)
Awaiting
an available transient area block
(see deferring a request for a transient SVC routine)
an event
(see causing a program to wait for one or more events)
availability of an enqueued resource
(see requesting one or more resources via an ENQ macro instruction)
availability of a serially reusable module
(see deferring the request for an unavailable module)

BLDL routine
entry point name of 471
function of when used by the common subroutines of Contents Supervision 87
module name for 462
preparation for use of during transient area XCTL processing 96
synopsis of 483
Block extent list and note list
format of 276

Causing a program to wait for one or more events
description of 53
CDABDEL routine
(see Release Loaded Programs routine)
CDADVANS 92
CDALLOC subroutine
function of 88
CDATTR field
use of 88,91,271
CDATTR2 field
use of 91,271
CDCONTRL
use of 91
CDDESTRY routine
function of 173
CDE
(see contents directory entry)
CDEEPADR field
meaning of 38
CDENIC
meaning of 89,271
CDENTPT field
meaning of 271
CDEPILOG subroutine
function of 91
CDEPRGNM field
meaning of 38,271
CDEXIT routine
description of 171
entry point name(s) of 471
flowchart of 409
module name for 460
synopsis of 483
CDHKEEP
use of 173
CDLLSRCH 87
CDMOPUP subroutine
function of 91
CDPURGE routine
function of
CDQUECTL subroutine
function of 90
CDRBP field
meaning of 271
CDSEARCH subroutine
function of 87
CDSETUP subroutine
function of 88
CDXLMJP field
meaning of 271
Channel-Check Handler
entry point name(s) of 471
module name(s) for 460
synopsis of 483
Channel error
recovery options for 34
CHAP routine
description of 45
entry point name(s) of 471
flowchart of 335-337
module name for 467
synopsis of 483
CHATR macro instruction
use of 155
Check I/O routine
description of 162
entry point name of 472
flowchart of 386
module name for 467
Checkmain routines
description of 163
entry point name(s) of 472
flowcharts of 388-389
module name for 468
Checkpoint entry
description of 159
Checkpoint Exit routine
description of 163
entry point name of 472
flowchart of 390
module name for 468
Checkpoint Header Record (CHR)
construction of 161
format of 162
Checkpoint Housekeeping routines
description of 160-161
entry point names of 471
flowcharts of 383-385

- module names for 467

Checkpoint message module
- description of 164
- entry point name of 472
- flowchart of 391
- module name for 468

CIRB
- macro instruction
  - function of 72
- routine (see Stage 1 Exit Effector)

Cleanup procedure (ABDUMP)
- function of 201

Closing data sets
- for abnormally terminating tasks 217
- for normally terminating tasks 182

Communications task
- External Interruption Handler routine
  - entry point name(s) of 472
  - flowchart of 156
  - module name for 463
  - synopsis of 484
- External Processor routine
  - entry point name(s) of 472
  - flowchart of 156
  - module name for 465
  - synopsis of 484
- Initialization routine
  - entry point name(s) of 472
  - flowchart of 156
  - module name for 463
  - synopsis of 484
- Miscellaneous Lookup Services routine
  - entry point name(s) of 472
  - module name for 465
  - synopsis of 484
- Open/Close routines
  - entry point names of 472
  - flowcharts of 156
  - module name(s) for 464
  - synopsis .of 484
- Processor routines
  - entry point names of 472
  - flowcharts of 156
  - module name(s) for 469
  - synopsis of 484
- Reply Processor routine
  - entry point name(s) of 472
  - module name for 465
  - synopsis of 484
- Request block
  - entry point name of 472
  - module name for 465
- Router routine
  - entry point name(s) of 472
  - flowcharts of 156
  - module name for 466
  - synopsis of 484
- task control block
  - entry point name of 472
  - module name for 465
- unit control tables (UCBs)
  - entry point names of 472
  - module name for 465
  - synopsis of 484
- Wait routine
  - charts of 156
  - entry point name(s) of 472
  - module name for 463
  - synopsis of 484

Communications vector table (CVT)
- definition of 29,246
- entry point name of 472
- format of 246
- module name for 460

Console Alarm routine
- (see Ring Bell routine)

Contents directory
- definition of 16

Contents directory entry (CDE)
- abnormal release of 223-224
- construction of 88
- definition of 85
- format of 271
- normal release of 183

Contents Supervision, common subroutines of
- description of 87
- entry point names of 472
- flowchart(s) of 358-360
- module name for 466
- synopsis of 484

Control record
- format of 280

Control and relocation dictionary record
- format of 282

CSPCHK subroutine
- function of 119,140

CVT
- (see communications vector table)

Data set descriptor records
- format of 168

Data set directory entry
- as used by common subroutines of Contents Supervision 87
- as used by the Program Fetch routine 110
- as used by the Stage 3 Exit Effector 74
- format of 272

Data Set Processor routines
- description of 167
- entry point names of 477
- flowcharts of 401-402
- module name for 468

DCB parameter for LINK, LOAD, or XCTL processing
- use of 88

DCM
- (see display control module)

DD statement for dump data set
- testing for issuance of 211

DE parameter
- use of 96

DEB queue
- use of to close data sets
  - during abnormal termination 218
  - during normal termination 182

Decimal Simulator routines for Model 91
- Add, Subtract, Zero - and - Add
  - description of 236
  - entry point name of 474
  - flowchart of 453
- Analyzer and End
  - description of 241
  - entry point name of 474
  - flowchart of 457
- Compare Decimal

      description of 241
      entry point name of 474
      flowchart of 452
   Divide Decimal 2
      description of 239
      entry point name of 474
      flowchart of 456
   Multiply Decimal
      description of 237
      entry point name of 474
      flowchart of 455
   Simulator Control
      description of 233
      entry point name of 474
      flowchart of 451
Deferring a request for a transient SVC routine 28
Deferring the request for an unavailable module 86
Delete routine
   description of 99
   entry point name(s) of 474
   flowchart of 362
   module name for 466
   synopsis of 484
Dequeue routine
   description of 66
   entry point name(s) of 474
   flowchart of 346-347
   module name for 468
   synopsis of 484
Dequeue TCB routine
   function of
synopsis of 485
Descriptor queue element (DQE)
   construction of 122
   definition of 122
   format of 285
   normal release of 140
Detach routine
   description of 49
   entry point name(s) of 474
   flowchart of 339
   module name for 467
   synopsis of 485
Dispatcher
   description of 175
   entry point name(s) of 474
   flowcharts of 410-419
   module name for 462
   synopsis of 485
Dispatching priority
   changing of 45
   definition of 45
   use of by the Dispatcher 176
Display Control Module (DCM)
   description of 229
   format of 297
DJSEARCH Subroutine
   description of 179-180
   flowchart of 413,419
DPQE
   (see dummy partition queue element)
DQE
   (see descriptor queue element)
DQLOAD subroutine
   function of 91
DQRBS subroutine
   function of 222
DQTCB
   (see Dequeue TCB routine, function of)
Dummy partition queue element (DPQE)
   construction of 120
   definition of 120
   format of 289
Dummy request block for the system error task 73
Dump, sample 302
Dump data set
   determining whether to open 211
   ensuring that the dump data set remains open for the duration of the job step step 212
   indicating whether the dump data set has been opened 213
   preparing to open 212
Dumping
   contents directory entries 196
   data extent blocks (DEBs) 196
   dynamically acquired storage 201
   extent lists 196
   load modules represented by CDEs 200
   main storage queue elements 196
   nucleus of main storage 199
   old PSW 196
   problem-program register save areas 199
   QCBs, QELs, and save areas belonging to IRBs 197
   register contents 200
   request blocks 196
   storage acquired for the task 201
   task I/O table (TIOT) 196
   task's load list 196
   TCB 196
   trace table 199,200
Dumps
   (see dumping, and sample dump)

ECB
   (see event control block)
End-of-Task (EOT) routine
   description of 181
   entry point name(s) of 475
   flowchart of 420
   module name for 460
   synopsis of 485
End-of-Task Exit routine (ETXR)
   scheduling of 183-184
ENQ routine
   (see Enqueue routine)
ENQ/DEQ Purge routine
   description of 224
   entry point name of 475
   synopsis of 485
Enqueue routine
   description of 61
   entry point name(s) of 475
   flowchart of 345
   module name for 467
   synopsis of 485
ENTAB
   (see entry table)
Entry points
   directory of 471
   embedded, informing the supervisor of of 97

Entry table (ENTAB)
definition of 100
format of 284
ENTRY2
use of 220
EOM indicator
meaning of 113
EOS indicator
meaning of 113
Erase Phase routine
entry point name of 475
function of 184
module name for 461
synopsis of 485
ERFETCH
(see error fetch sequence)
Error fetch sequence
function of 74
ETXR operand
effect when ATTACH routine is executed
executed 40
ETXR scheduling
(see End-of-Task Exit routine (EXTR), scheduling of)
Event control block
format of 265
(also see posting an event control block)
EXCP Supervisor
entry point name of 475
function of 111
module name for 463
synopsis of 485
Exit Effector
(see Stage 1 Exit Effector, Stage 2 Exit Effector, and Stage 3 Exit Effector)
Exit routine
description of 169
entry point name(s) of 475
flowchart of 405-406
module name for 466
synopsis of 485
Extent list
abnormal release of 224,225
construction of 109-110
definition of 108
normal release of 173
(also see scatter extent list, format of; and block extent list and note list, format of)
External First-Level Interruption Handler
description of 31-33
entry point name(s) of 475
flowcharts of 323,324
module name for 461
synopsis of 486
External FLIH
(see External First-Level Interruption Handler)
External interruptions
description of 31-33
Extract routine
description of 47
entry point name(s) of 475
flowchart of 338
module name for 466
synopsis of 486

Fail soft storage element map (FSSEMAP)
format of 301

FBQE
(see free block queue element)
First CPU Signal routine
description of 84
entry point name of 475
module name for 460
First-time logic switch
(see program interruption element)
FMBRANCH
meaning of 460
FQE
(see free queue element)
Free block queue element (FBQE)
construction of 120,140
definition of 120
format of 289
Free queue element (FQE)
construction of 122
format of 286
function of 122
FREEMAIN macro instruction
types of SVC instructions for 118
FREEMAIN routine
description of 139
entry point name(s) of 475
flowchart of 368
module name for 466
synopsis of 486
FSSEMAP
(See fail soft storage element map)

GETMAIN macro instruction
list structure of 122
types of SVC instructions for 118
GETMAIN routine
description of 119
entry point name(s) of 475
flowchart of 367
module name for 466
synopsis of 486
GOVRFLB
format of 287
meaning of 139

I/O block
for the I/O supervisor transient area
entry point name of 475
module name for 460
for the SVC transient areas
entry point names of 475
module name for 470
I/O First-Level Interruption Handler
description of 33
entry point name(s) of 475
flowchart of 325,326
module name for 461
synopsis of 486
I/O FLIH
(see I/O First-Level Interruption Handler)
I/O Interruption Supervisor
entry point name of 475
module name for 462
synopsis of 487
I/O interruptions 33

I/O requests and I/O operations in process
   purging of 207
I/O supervisor transient area
   entry point name of 475
   I/O block
      entry point name of 475
   module name for 460
   synopsis of 487
I/O switch (IORGSW)
   function of 33
Identify routine
   description of 97
   entry point name(s) of 475
   flowchart of 361
   module name for 467
   synopsis of 487
IEADQTCB
   (see Dequeue TCB routine, function of)
IEAHEAD
   meaning of 46
IEAQABL
   (see Release Loaded Programs routine)
IEAQERA
   (see Erase Phase routine)
IEAQSPET
   (see Release Main Storage routine)
IEAQTAQ
   (see transient area control table)
IEASCSAV
   meaning of 22
IEATCBP
   meaning of 176
IEATYPE1
   use of during ABTERM processing 193
IEEVLIN routine
   function of 157
   (see also WRITELOG Log Initialization routine)
IEEVLOPN routine
   function of 157
   (see also WRITELOG Open Device routine)
IEEVWAIT routine
   function of 157
   (see also WRITELOG Master Wait routine)
Informing the supervisor of an embedded module entry point 97
Initial Program Loading (IPL) routine
   entry point name of 475
   module name for 470
   synopsis of 487
Interruption handling 21
   (see also SVC interruption handling, program interruptions, external interruptions, I/O interruptions, and machine interruptions)
Interruption queue element
   construction of 41
   definition of 70
   format of 269
   normal release of 75
   queuing of 72
Interruption request block
   abnormal release of 222
   construction of 72
   definition of 16
   format of 256
   normal release of 174
IOB
   (see I/O block)
IOBSEEK field
   use of 113
IORGSW
   (see I/O switch)
IQE
   (see interruption queue element)
IQE list (AEQJ) 72
   (see also interruption queue element, construction of)
IRB
   (see interruption request block)

JFCB Processor routines
   description of 166
   entry point names of 477
   flowchart of 397
   module name for 468
Job file control blocks for log data sets
   (see log data sets, job file control blocks (JFCBs) for)
Job pack area control queue
   definition of 16
   meaning of 87
   use of 87
Job pack area queue
   (see job pack area control queue)
Job Step Timing
   in Dispatcher routine
      description of 179,180
      flowchart of 411-413
   in Post routine
      description of 55
      flowchart of 344
   in Wait routine
      description of 55
      flowchart of 342
   in Timer Second-Level Interruption Handler
      description of 150
      flowchart of 382
JPACQ
   (see job pack area control queue)
LCS (2361 Core Storage)
   See Main Storage Hierarchy Support

Limit priority
   use of 45
Link pack area
   definition of 16
Link pack area control queue
   definition of 16
   meaning of 88
   search of 88
Link pack area queue
   (see link pack area control queue)
LLCDPTR field
   meaning of 272
LLCOUNT field
   meaning of 272
Load list
   definition of 16,87
   purging of 225
Load list element
   abnormal release of 225
   construction of 82
   definition of 82
   format of 272

normal release of 99
Log and WRITELOG Post routine
entry point name of 476
module name for 465
synopsis of 487
Log command
use of 157
Log data sets
job file control blocks (JFCBs) for
entry point name of 476
module name for 465
LPACQ
(see link pack area control queue)

Machine-Check Handler
entry point name(s) of 476
module name(s) for 469
synopsis of 487
Machine interruptions
recovery options for 33
Main storage
dumping of allocated 201
purging of 225
Main Storage Hierarchy Support feature
contents supervision service routines with 86
description of 17,19
GETMAIN routine with 110,120
loading of overlay module with 100
Major CDE
(see contents directory entry)
Major QCB
(see queue control block)
Master Scheduler Initialization routine
entry point name of 476
module name for 464
synopsis of 487
Master scheduler resident table
entry point name of 476
module name for 464
synopsis of 487
Master scheduler task control block (TCB)
entry point name of 476
module name for 333
Minor CDE
(see contents directory entry)
Minor QCB
(see queue control block)
Model 91
Decimal Simulator routines (See Decimal Simulator routines)
Program check First-Level Interruption Handler routine for
description of 31
flowchart of 322
SER1 routine for
description of 36-37
flowchart of 329-330
module name for 462
Modules
purging of during abnormal termination 222
Mount/Verify routines
description of 166
entry point names of 477
flowcharts of 398,399
module name for 468
MPCVT
(See multiprocessing communications vector table)
MSSLOOP
function of 207
Multiprocessing communications vector table (MPCVT)
format of 300
Multiprocessing feature
ABDUMP routine with 194
description of 18
Dispatcher with 175,176,177
External FLIH routine with
description of 32
flowchart of 324
FREEMAIN routine with 140,141
I/O FLIH routine with
description of 33
flowchart of 326
Job Step Timing with
description of 179,180
flowchart of 419
Machine - check recovery with 34
Program Interruption FLIH routine with
description of 30-31
flowchart of 321
Set Status routine with 82
Stage 3 Exit Effector with 74
SVC FLIH routine with
description of 22-23
flowchart of 316
Task Switching routine with 81
Type 1 Exit routine with 169
Must-complete status
clearing of 68
meaning of TCB flags for 65
setting of 64

NFN flag
meaning of 271
setting of 90
NIC flag
meaning of 271
NIP
(see nucleus initialization program)
NLR flag
meaning of 271
Nondispatchability flags, TCB
meaning of 189
Nondispatchable
(see nondispatchability flags)
Nonreusable module
purging of module after its use is complete 173
Normal termination
description of 181
Nucleus initialization program (NIP)
entry point name of 476
module name for 461
synopsis of 487

Opening the dump data set 212
Operator communications queues
purging of 207
OPSW
(see RB old PSW, SVC old PSW, external interruptions)
ORDERCDQ routine
function of 173

Overlay supervisor
   description of 101
   entry point names of 476
   flowchart of 366
   loading of nonresident module 89
   module names for 465,466
   synopsis of 487
   types of processing 104

Parameter list element (ENQ/DEQ routines)
   format of 266
PARRLSE routine
   function of 211
Partially loaded modules
   release of 211
Partition queue element
   construction of 119
   definition of 17
   format of 286
Partitioned data set directory entry
   (see data set directory entry)
PC FLIH
   (see Program-Check First-Level
    Interruption Handler
PDS directory entry
   (see data set directory entry)
PICA
   (see program interruption control area)
PIE
   (see program interruption element)
Post routine
   description of 55
   entry point name(s) of 476
   flowchart of 343
   module name for 466
   synopsis of 488
Posting an event control block
   general
      (see Post routine)
   posting the I/O supervisor event control
    block 108
   posting the parent task's event control
    block 184
   posting the program fetch event control
    block 108
PQE
   (see partition queue element)
PRB
   (see program request block)
PRBPROC subroutine
   function of 222
Preserve routines
   description of 162
   entry point names of 472
   flowchart of 387
   module name for 468
Program Check First-Level Interruption
 Handler
   description of 29-31
Program Fetch buffer table
   description of 108
   format of 279
Program Fetch Channel-End Appendage routine
   description of 113
   entry point name of 476
   flowchart of 364
   module name for 364
   synopsis of 488
Program Fetch PCI Appendage routine
   description of 113
   entry point name of 477
   flowchart of 364
   module name for 460
   synopsis of 488
Program Fetch routine
   description of 105
   entry point names of 476
   flowchart of 363-364
   module name for 465
   synopsis of 488
Program Fetch work area
   description of 106,107
   format of 278
   initialization of 107,108
Program interruption control area (PICA)
   construction of 51
   format of 263
Program interruption element (PIE)
   construction of 52
   format of 263
   normal release of 182
   purging of 207
Program interruptions 29
Program request block (PRB)
   abnormal release of 224
   construction of 91
   definition of 16
   format of 258
   normal release of 174
Purge Timer routine
   entry point of 477
   flowchart of 421
   function of 207
   module name for 461
   synopsis of 488

QCB
   (see queue control block)
QCB queues
   abnormal removal of elements from 224
   construction of 60
   illustration of 60
   normal removal of element from 67
   origin of 461
QEL
   (see queue element (QEL) for serializing
    the use of a resource)
QEL queues
   abnormal removal of elements from 224
   construction of 61
   description of 58
   illustration of 60
   normal removal of element from 67
Qname
   meaning of 58
Queue control block (QCB)
   abnormal release of 224
   construction of 61
   definition of 58
   formats of 267
   normal release of 67
Queue element (QEL) for serializing the use
 of a resource
   abnormal release of 224
   construction of 61
   definition of 60

format of 268
normal release of 67

RB
(see request block)
RB fields
meaning of
(see interruption request block, format of; program request block, format of; supervisor request block, format of; system interruption request block, format of)
RB flags
meaning of 258
RB old PSW
meaning of 21
saving of 21
RBQUEUED subroutine
function of 223
RBREMOVE subroutine
function of 222
Recovery options for machine checks and channel errors 33-34
Recursion, ABEND
invalid 204
meaning of 204
valid 205
Refreshing a transient area block
(see Transient Area Refresh routine)
Region of main storage
allocation of 119
Relative Priority routine
description of 83
Release Loaded Programs routine
description of 183
entry point name of 477
flowchart of 423
module name for 461
synopsis of 488
Release Main Storage routine
description of 183
entry point name of 477
flowchart of 422
module name for 461
synopsis of 488
Releasing main storage
at end of task 183
during abnormal termination 225
in response to a FREEMAIN macro instruction 139
Relocation list dictionary record
format of 281
use of 114
RELPRIOR routine
(See Relative Priority routine)
REN flag
meaning of 271
Reply Purge routine
(see WTOR Purge routine)
Reply queue element
abnormal release of 207
definition of 135
format of 290
Repmain routines
description of 165
entry point names of 477
flowcharts of 393-394
module name for 467
Request block
definition of types 16
dummy
(see dummy request block for the system error task)
fields
meaning of
(see interruption request block, format of; program request block, format of; supervisor request block, format of; system interruption request block, format of)
flags
description of 258
formats of (see interruption request block, format of; program request block, format of; supervisor request block, format of; system interruption request block, format of)
normal release of 174
purging of 221-224
Request queue element (RQE)
abnormal return to free list 207
definition of 70
format of 290
normal return to free list 74
queuing of 72
Requesting one or more resources via an ENQ macro instruction 61
Requests for user exit routines
purging of 207
Reserve macro instruction
as used in ABEND6 routine 224
as used in Dequeue routine 70
as used in Enqueue routine 66
"Reset must complete" function
description of 68
Resource queues for ENQ/DEQ requests 58
Responsibility count (LLCOUNT)
format of 272
use of 92,99,225
(see also load list element)
Restart Exit routine
description of 168
entry point name of 477
flowchart of 403
module name for 468
Restart Housekeeping routines
description of 165
entry point names of 477
flowcharts for 392
module name of 467
Resume I/O routine
description of 163
entry point name of 472
flowchart for 390
module name of 468
RET parameter
effect of during DEQ processing 69
effect of during ENQ processing 61-62
RIQE
(see Rollout I/O queue element)
RLD buffer
description of 108
RLD record
format of 281
Rname

   meaning of 58
ROLL parameter 42
Rollout I/O Queue Element (RIQE)
   construction of 133
   definition of 133
   format of 290
   normal release of 143
Rollout/Rollin feature
   description of 18
Rollout/Rollin module
   description of 127
   entry point name of 477
   flowcharts of 369-376
   module name for 461
   scheduling of 124
   synopsis of 489
RQE
   (see request queue element)
RQE queue (AEQA)
   placing element on 72
   removing element from 74
   (also see request queue element,
    abnormal return to free list)

Sample dump
   format of 302
SCANTREE subroutine
   function of 188
Scatter extent list
   format of 275
Scatter/translation record
   format of 277
SCB
   (see STAE control block)
Secondary communications vector table
   format of 142
Second CPU Recovery Management System
 Interface routine
   description of 32
   entry point name of 477
   module name for 469
SEGLD macro instruction
   linkage to the Overlay Supervisor
    for 101
SEGLD Processor routine
   description of 103
   entry point name of 477
   flowchart of 366
   module name for 470
   synopsis of 489
Segment table
   definition of 100
   format of 283
SEGTAB
   (see segment table)
SEGWT macro instruction
   linkage to the Overlay Supervisor
    for 101
SER flag
   meaning of 271
SER routines
   (see SER0 routine and SER1 routine)
Serializing the use of a resource 58
SETSUBS subroutine
   function of 188
SER0 routine
   description of 35
   entry point names of 478
   flowchart of 327
   module names for 460,465
   synopsis of 489
SER1 routine
   description of 36
   entry point name of 478
   flowcharts of 328-330
   module name for 461
   synopsis of 489
"Set must complete" function
   description of 64
Set Status routine
   description of 82
   entry point name of 478
   module name for 467
   synopsis of 489
Severe error condition
   recognizing a 204
Shared Direct Access Device feature
   Dequeue routine with
      description of 70
      flowchart of 347
   Enqueue routine with
      description of 66
      flowchart of 345
   description of 18
SHOLDTAP routine
   description of 84
   synopsis of 489
SHPC
   (see six-hour pseudo clock, use of)
SIRB
   (see system interruption request block)
Six-hour pseudo clock
   use of 148
SPEOT routine
   (see Release Main Storage routine)
SPIE routine
   description of 51
   entry point name(s) of 478
   flowchart of 340
   module name for 466
   synopsis of 489
SPLS parameter (ABDUMP)
   use of 201
SPQE
   (see subpool queue element)
STAE
   macro instruction 51,76
   Service routine
      description of 76
      entry point name of 478
      flowchart of 354
      module name for 466
      synopsis of 489
STAE control block (SCB)
   format of 264
Stage 1 Exit Effector
   description of 72
   entry point name(s) of 478
   flowchart of 348
   module name for 467
   synopsis of 489
Stage 2 Exit Effector
   description of 72
   entry point name(s) of 478
   flowchart of 349
   module name for 462

synopsis of 489
Stage 3 Exit Effector
description of 72
entry point name(s) of 478
flowchart of 350-351
module name for 462
synopsis of 489
STATUS macro instruction
analysis of parameters 82
Steal Core subroutine
function of 208-209
STIMER routine
description of 148
entry point name(s) of 478
flowchart of 378
module name for 467
synopsis of 489
Subpool
indicating shared ownership of 43
relationship with types of GETMAIN requests 120
Subpool End-of-Task routine
(see Release Main Storage routine)
Subpool queue element (SPQE)
abnormal release of 225
construction of 122
definition of 122
format of 285
normal release of at end of task 183
Subtask
attaching of 39
detaching of 49
queue
meaning of 43
selection of during abnormal termination 173,188,221
Supervisor request block
abnormal release of 220,221
construction of 23
definition of 16
format of 254-255
normal release of 171
SVC First-Level Interruption Handler
description of 21-23
SVC FLIH
(see SVC First-Level Interruption Handler)
SVC interruption handling
for nonresident (transient) SVC routine routine 25
for type-1 SVC routine 21-23
for resident SVC routine requiring an SVRB 23
SVC old PSW
loading of
saving of 21-22
SVC purge parameter list
definition of 134
format of 291
SVC Purge routine
entry point name of 478
function of 207
module name for 466
synopsis of 489
SVC Second-Level Interruption Handler
description of 23
entry point name(s) of 478
flowchart of 317
module name for 462
synopsis of 490
SVC SLIH
(see SVC Second-Level Interruption Handler)
SVC table
format of 245
function of 23
SVRB (see supervisor request block)
SYNCH processing
description of
(see Contents Supervision, common subroutines of, description of)
entry point name for 478
flowchart of 358-360
module name for 466
SYSABEND data set
(see dump data set)
System error task
meaning of 73
System error TCB
entry point name of 478
module name for 461
use of 73
System error transient area
(see I/O supervisor transient area)
System interruption request block
definition of 16,73
format of 257
initialization of 73
System log
definition of 157
System Quiesce routine
description of 226
entry point name of 478
flowchart of 439
module name for 462
synopsis of 490
SYSUDUMP data set
(see dump data set)
SYS1.LINKLIB data set
data control block (DCB)
entry point name for 478
module name for 476
data extent block (DEB)
entry point name for 478
module name for 476
SYS1.LOGREC data set
use of 34
SYS1.SVCLIB data set
data control block (DCB)
entry point name of 478
module name for 470
data extent block (DEB)
entry point name of 478
module name for 470

TAB
(see transient area block)
TABLDL
(see Transient Area Fetch routine)
TACT
(see transient area control table)
TACTNEXT
use of 27
TAHABEND subroutine
function of 220
TAHEXIT subroutine

(see Transient Area Exit routine)
TAHFETCH
(see Transient Area Fetch routine)
TARESTRT
(see SVC Second-Level Interruption Handler)
Task asynchronous exit routine
scheduling of 51
Task control block
abnormal release of 226
construction of 39
definition of 14
flags
description of 251
format of 250
normal release of 184
queuing of 43
Task Removal routine
description of 84
entry point name of 478
module name for 470
synopsis of 490
Task Switching routine
description of 80
entry point name of 478
flowcharts of 452,453
module name for 462
synopsis of 490
Task timing
description of 178
flowchart of 411-412
TASKSEL subroutine
function of 217
TAXEXIT
(see Transient Area Exit routine)
TAXRETRY
(see Transient Area XCTL routine)
TCB
(see task control block)
TCB fields
formats of 250
meaning of 250-252
TCB flags
meaning of 251-252
TESTDSP routine
(see Task Removal routine)
TESTOCT
use of 224
TESTRAN interpreter
entry point name(s) of 479
module names for 465,467
use by the common subroutines of Contents Supervision 89
use by the Model 91 Program Interruption Handler 31
use by the Overlay Supervisor 366
Time of expiration
determining 149
Time routine
description of 148
entry point name(s) of 479
flowchart of 377
module name for 466
synopsis of 490
Time-slice control element
pointers of 46
creation of 15
format of 296
Time-slicing feature
Attach routine with
description of 42
flowchart of 334
Chap routine with
description of 45
flowchart of 336-337
description of 18
Dispatcher routine with
description of 178
flowchart of 414-416
EOT routine with
description of 184
Timer interruption handling
description of 150
Timer queue
positioning of elements on 149
purging elements from 207
Timer queue element
abnormal release of 207
construction of 150
definition of 149
format of 292
normal removal from timer queue 150
Timer Second-Level Interruption Handler
description of 150
entry point names of 479
flowchart of 380-382
module name for 462
synopsis of 490
Timer SLIH
(see the Timer Second-Level Interruption Handler)
Timing
job step 178
task 178
TOX
(see time of expiration, determining)
TQE
(see timer queue element)
Trace routine
entry point names of 479
function of 195
module name for 470
synopsis of 490
Trace table
format of 260,261
function of 195
Transient Area Availability Check routine
description of 25
entry point name(s) of 479
flowchart of 318
module name for 470
synopsis of 490
Transient area block
function of 25
Transient area control table (TACT)
definition of 25
format of 262
Transient Area Exit routine
description of 171
entry point name(s) of 479
flowchart of 407
module name for 470
synopsis of 490
Transient Area Fetch routine
description of 28
entry point name(s) of 479

flowchart of 319
module name for 470
synopsis of 491
Transient area fetch task
purpose of 28
Transient area fetch TCBs
entry point name(s) of 479
function of 28
module name for 462
Transient area handler
flowchart of 317
functions of 25-29
Transient area I/O blocks (IOBs) and associated transient area blocks
entry point name(s) of 479
module name for 470
Transient Area Refresh routine
description of 174
entry point name(s) of 479
flowchart of 408
module name for 462
synopsis of 491
Transient area user count
address of 294
definition of 96
Transient Area XCTL routine
description of 92-97
entry point name(s) of 480
flowchart of 358
module name for 462
synopsis of 491
Tree of terminating tasks
illustration of 188
TSCE
(see time-slice control element)
TTIMER routine
description of 152
entry point name(s) of 480
flowchart of 379
module name for 467
synopsis of 491
Twenty-four hour pseudo clock (T4PC)
use of 148
2250 System Operator's Console
description of 18
2250 System Operator's Console Support routines
control flow of 230
description of 229
entry points of 472-473
flowcharts of 440-450
module names for 463,465
Type-1 Exit routine
description of 169
entry point name(s) of 480
flowchart of 404
module name for 462
synopsis of 491
Type-1 SVC switch (IEATYPE1)
entry point name of 480
function of during ABTERM processing 190
module name for 462
T4PC
(see twenty-four hour pseudo clock, use of)
Use/responsibility count
meaning of 99
(see also CDUSE)
User Exit routine
scheduling of 70

Validity Check routine
description of 81
entry point name(s) of 480
module name for 462
synopsis of 491
Vary Storage Offline routine
description of 140
synopsis of 491
Vary queue element (VQE)
format of 300
VQE
(see vary queue element)

Wait routine
description of 53
entry point name of 480
flowchart of 341-342
module name for 466
synopsis of 491
Write-to-Log routine
entry point name of 480
module name for 465
synopsis of 491
Write-to-Operator routine
entry point name of 480
module name for 466
synopsis of 491
WRITELOG Available Log Data Set routine
entry point name of 480
module name for 465
synopsis of 491
WRITELOG command
use of 157
WRITELOG Dispatch routine
entry point name of 480
module name for 465
synopsis of 491
WRITELOG Get Region routine
entry point name of 480
module name for 464
synopsis of 491
WRITELOG Log Initialization routine
entry point name of 480
module name for 465
synopsis of 491
WRITELOG Log Writer routine
entry point name of 480
module name for 465
synopsis of 491
WRITELOG Master Wait routine
entry point name of 480
module name for 465
synopsis of 491
WRITELOG Open Device routine
entry point name of 480
module name for 465
synopsis of 491
WTL routine
function of 157
(see also Write-to-Log routine)
WTO routine
(see Write-to-Operator routine)

WTOR macro instruction
   use of 156
WTOR Purge routine
   entry point name of 480
   function as used by ABEND routine 207
   function as used by EOT routine 182
   module name for 464
   synopsis of 491

XCTL processing
   description of 92
   entry point name of 480
   flowchart of 358-360
   module name for 466

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

Printed in U.S.A. Y28-6659-3

READER'S COMMENT FORM

IBM System/360 Operating System
MVT Supervisor

Y28-6659-3

Please check or fill in the items below, adding explanations and other comments in the space provided.

Which of the following terms best describes your job?

□ Programmer
□ Manager
□ Operator
□ Instructor
□ Systems Analyst
□ Engineer
□ Mathematician
□ Student/Trainee
□ Customer Engineer
□ Systems Engineer
□ Sales Representative
□ Other (explain) ____________

Does your installation subscribe to the SRL Revision Service? □ Yes □ No

How did you use this publication?

□ As an introduction
□ As a reference manual
□ As a text (student)
□ As a text (instructor)
□ For another purpose (explain) ______________________________

Did you find the material easy to read and understand? □ Yes □ No (explain below)

Did you find the material organized for convenient use? □ Yes □ No (explain below)

Specific criticisms (explain below)

Clarifications on pages ______________________________

Additions on pages ______________________________

Deletions on pages ______________________________

Errors on pages ______________________________

Explanations and other comments:

Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

Y28-6659-3

YOUR COMMENTS PLEASE . . .

This manual is one of a series which serves as reference sources for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

FOLD FOLD

FOLD FOLD

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

T, Fox

IBM Technical Newsletter

File Number	S360-36
Re: Form No.	Y28-6659-3
This Newsletter No.	Y27-7190
Date	April 28, 1969
Previous Newsletter Nos.	None

MVT SUPERVISOR PLM

This Technical Newsletter, a part of a special distribution supporting Advanced Checkpoint/Restart under Model 65 Multiprocessing, provides replacement pages for the publication IBM System/360 Operating System: MVT Supervisor Program Logic Manual, Form Y28-6659-3.

Pages to be replaced or added are:

121-122
122.1-122.2 (added)
367-368
368.1-368.2 (added)

Text changes are indicated by a vertical line to the left of the affected text; changes to illustrations are indicated by a bullet (•) to the left of the caption.

Summary of Amendments

This Technical Newsletter includes information for Advanced Checkpoint/Restart under Model 65 Multiprocessing.

Please file this cover letter at the back of the manual to provide a record of changes.

Restricted Distribution

IBM Corporation, Programming Publications, Dept. 637, Neighborhood Road, Kingston, N.Y. 12401

PRINTED IN U.S.A.