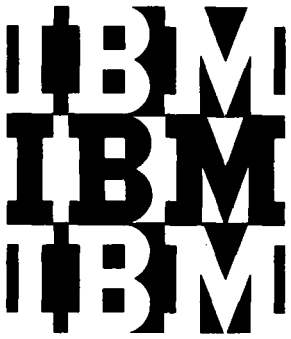


File No. S360-36
GY28-6659-6

Program Logic

OS Release 21

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:

- 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 personnel 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.

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

Seventh Edition (March, 1972)

This edition corresponds to Release 21 of System/360 Operating System. Major changes are listed in the Summary of Major Changes; the sections affected by the new items should be reviewed in their entirety.

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 chain.

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

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

PREFACE

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.

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 Guide it is called "system queue area."

PREREQUISITE PUBLICATIONS

IBM System/360 Operating System: MVT Guide, GC28-6720

Supervisor Services and Macro Instructions, GC28-6646

Data Management Macro Instructions, GC26-3794

Messages and Codes, GC28-6631 (useful for the formats and meanings of messages and codes)

PUBLICATIONS TO WHICH THE TEXT REFERS

IBM System/360 Principles of Operation, GA22-6821 (referred to in "Interruption Handling")

IBM System/360 Operating System: Job Control Language Reference, GC28-6704 (referred to in "Checkpoint/Restart" and "Abnormal Termination")

Linkage Editor, Program Logic Manual, GY28-6610 (referred to in "Contents Supervision")

Linkage Editor and Loader, GC28-6538 (referred to in "Contents Supervision")

MVT Job Management, Program Logic Manual, GY28-6660 (referred to in "Task Supervision" and "Checkpoint/Restart")

I/O Supervisor, Program Logic Manual, GY28-6616 (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 Handling")

Machine-Check Handler for the Model 65 Program Logic Manual, GY27-7155 (referred to in "Interruption Handling")

Machine-Check Handler for the Model 85 Program Logic Manual, GY27-7184 (referred to in "Interruption Handling")

Machine-Check Handler for the System/370 Models 135 and 145, GY27-7237 (referred to in "Interruption Handling")

Machine-Check Handler for the System/370 Models 155 and 165, GY27-7198 (referred to in "Interruption Handling")

Online Test Executive Program, Program Logic Manual, GY28-6651 (referred to in "Termination Procedures")

Service Aids Logic, Program Logic Manual, GY28-6721 (referred to in "Interruption Handling" and "Special Features")

System Generation, GC28-6554 (referred to in "Interruption Handling")

OTHER PUBLICATIONS

TSO Control Program, GY27-7199 (describes the logic of the Time Sharing Option which is accessed by the MVT Supervisor).

SUMMARY OF MAJOR CHANGES -- RELEASE 21

Item	Description	Sections Affected
Generalized Trace Facility	New facility that monitors and records system events.	1, 2, 10, 11, 12, 13, 14
Extended SVC Router	New routines that provide linkage to Supervisor Service routines.	2, 13, 14
System/370 Model 195	This CPU performs processing identical to the System/360 Model 195.	2, 11, 13, 14
Machine Check Record	New record containing information collected by SER0 and SER1.	2, 12
IFCEREP0	A service aid that formats and writes SYS1.LOGREC records.	2
Shared DASD	The restriction on Shared DASD Support by the Model 65 Multiprocessing System has been removed.	1
Inter-Partition Post	The Post routine checks for Post parameters in the user parameter list.	3, 13
Multiple-Line Write to Operator routines	New routines to process multiple-line WTOs in systems both with and without MCS.	7, 12, 13, 14
Graphic Console Initialization Module	New routine that initializes display console configuration.	7, 13, 14
Console Support routines	New routines to process multiple-line WTOs.	7, 13, 14
DIDOCS routines	New routines that provide an interface between DIDOCS and MCS, and process console requests.	7, 12, 13, 14
Restart routines	New routines to process TCAM, SYSIN, ISAM, BDAM, and DOS data sets.	8, 13, 14
ABDUMP routine	New modules to display trace data.	10, 13, 14
Abnormal Termination routine (ABEND)	The ABEND modules have been reorganized according to functional processing.	10, 12, 13, 14
Damage Assessment routine (DAR)	The DAR modules have been reorganized according to functional processing.	10, 12, 13, 14
ABEND/STAE Interface routine (ASIR)	The ASIR modules have been reorganized according to functional processing.	10, 12, 13, 14
Diagnostic Aids in ABEND Processing	Debugging aids.	Appendix A

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 MVT Guide.

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

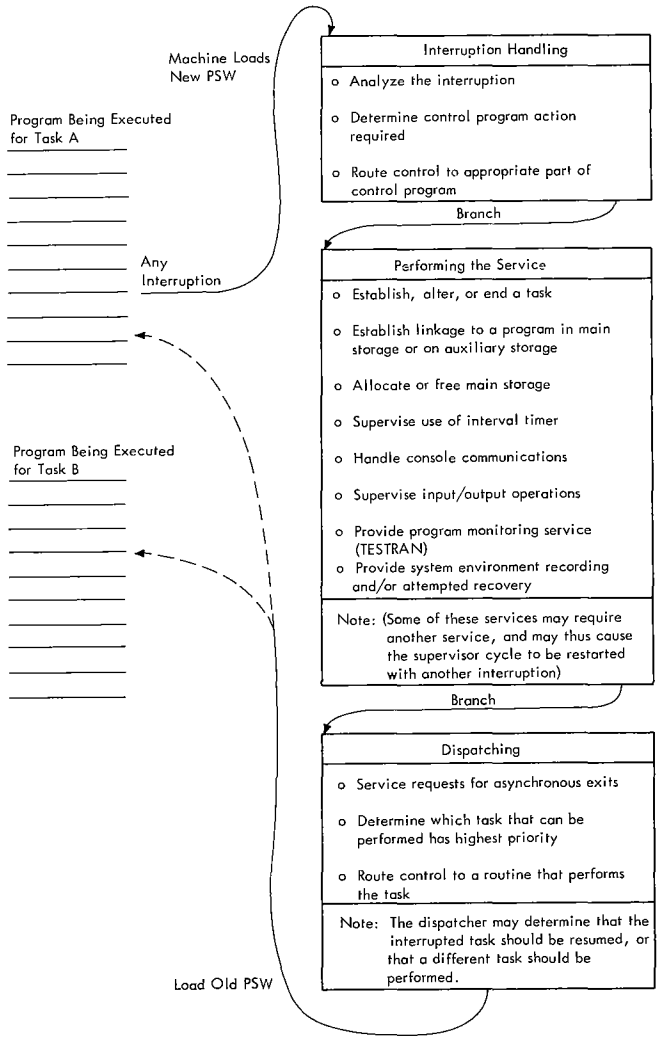


Figure 1-1. Overall Flow of the Supervisor

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 the system log and hard copy log. The supervisor provides the means for the operator to directly communicate with the system, and for a program to write a message to the operator or programmer. It also supports the system log, which records statistical information about system usage. When using the Multiple Console Support (MCS) option, a hard copy log can be specified to record operator commands, system messages and commands, and application program messages. The hard copy log can be either a console, or the system log.

- 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 Machine-Check Handler (MCH), optional programming support for Model 65 (MCH/65) and standard programming support for Model 65 Multiprocessor, the Model 85 (MCH/85), and the System/370 Models 145 (MCH/145), 155 (MCH/155), and 165 (MCH/165), performs the following functions: (1) records environmental data, and (2) attempts to analyze the malfunction and restore the system to normal operation.

  The Channel-Check Handler (CCH) provides programming support for models using the 2860/2870/2880 and System/370 Models 145 or 155 integrated channels. CCH aids the device-dependent error recovery procedures in recovering from channel errors by providing them with channel logout information. CCH also builds inboard record entries to be written on SYS1.LOGREC.

  Alternate Path Retry (APR) allows an I/O operation that has developed an error on one channel to be retried on another channel (if another channel is assigned to the device performing the I/O operation). APR also provides the capability to VARY a path to a device online or offline. The selective retry function of APR is optional for MVT and standard for M65MP. The VARY PATH function of APR is standard for both MVT and M65MP.

  Dynamic Device Reconfiguration (DDR), optional programming support that is not model-dependent but is automatically included for M65MP, allows a demountable volume to be moved from one device to another, and repositioned if necessary, without abnormally terminating the affected job or reperforming IPL. A request to move a volume may be initiated by either the operator or the system, for SYSRES or non-SYSRES devices.

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 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

---

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

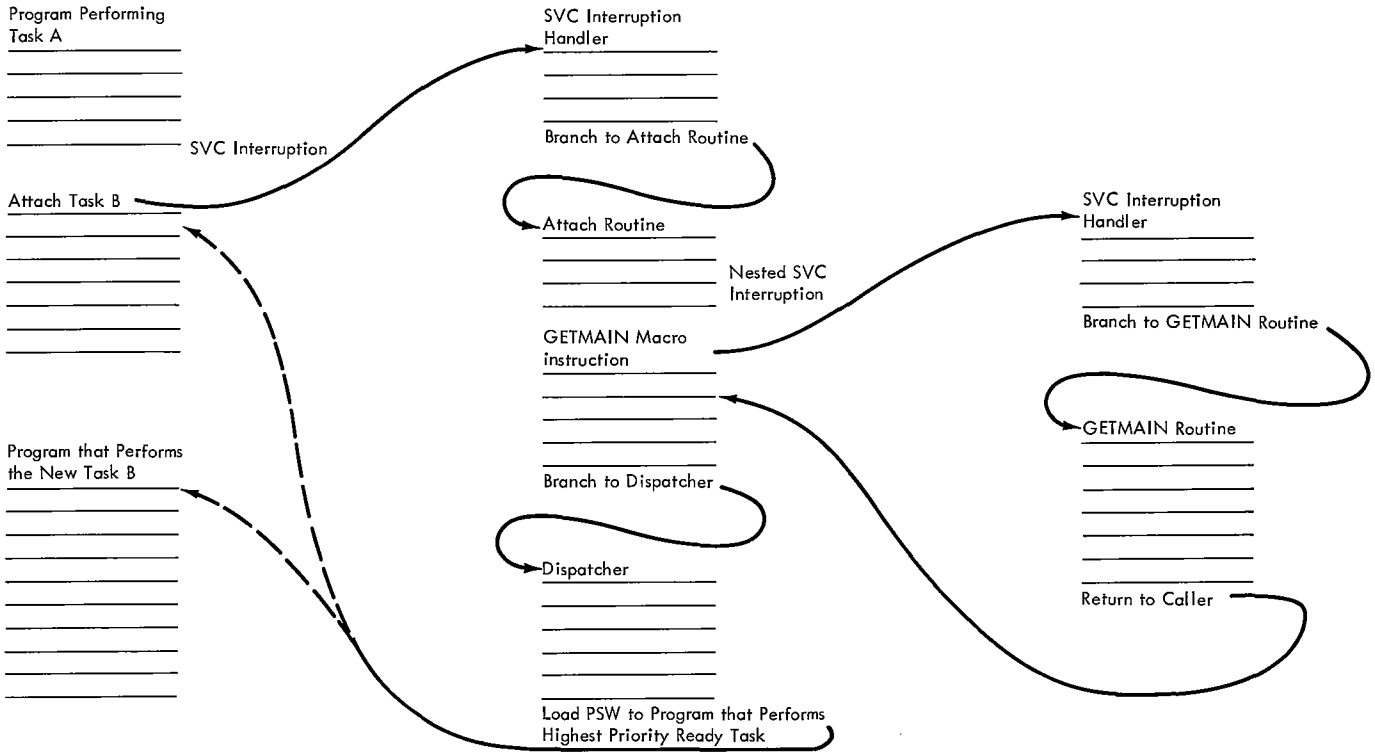


Figure 1-2. Flow of Control After the ATTACH Macro Instruction Is Issued

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.

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. In systems containing IBM 2361 Core Storage and Main Storage Hierarchy Support, a secondary link pack area may be constructed in hierarchy 1.

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 is 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 (excluding the Model 65 Multiprocessing 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 auto-

---

[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 requester is a supervisor routine.

matically 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

The console communications service routines handle messages from a program to the operator or programmer, as well as messages from the operator to the system. The SVC routines WTO and WTOR (Write to Operator and Write to Operator with Reply) process the output messages from the system to the operator or to the programmer. Input comes from an external interruption, caused by operator intervention at the console.

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

Multiple Console Support (MCS), a system generation option, allows the selective routing of messages (WTO and WTOR) to one or more consoles, using routing codes assigned to each message and each console. When a graphic console or more than one non-graphic console is included with MCS in the system, a hard copy log is mandatory. The system log or any non-graphic device may be specified as the hard copy 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 records 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, ASIR, DAR, 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 dispatcher. 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 inthe 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.

Time Sharing Option

The Time Sharing Option (TSO) adds general purpose time sharing to the facilities currently available with the MVT control program by enabling users at remote terminals to execute programs concurrently and to interact with those programs during execution. The installation can dedicate its system to time-sharing operations or it can run concurrent time-sharing and batch-processing operations. Including TSO in the control program does not restrict or limit batch operations.

The Time Sharing Option consists of a control program (containing IBM-supplied service routines and command processors) and a number of IBM Program Products (available from IBM for a license fee). The TSO control program, an extension of the MVT control program, accomplishes the following functions:

- Identifies and verifies the time-sharing user.
- Defines the user job.
- Assigns the user job an amount of execution time (as determined by the installation-selected scheduling algorithm).
- Ensures a specified percentage of execution time for batch processing operations (when required).
- Transfers the user job between main storage and a direct access device (swapping).
- Logically reconnects the user job to the operating system when the job is swapped into main storage (called "restoring" the job); logically disconnects the user job from the operating system before swapping out the user job (called "quiescing" the job).
- Establishes and maintains communications between the terminal user, his programs, and the TSO control program.
- Handles attention interruptions from the terminal user.
- Provides an optional set of SMF records for the TSO system, job and data management information, and optional system control task exits to installation-supplied routines.
- Provides an optional record of the information that is passed to the Time Sharing Driver via the Time Sharing Interface Program from such TSO system routines as the Region Control Task, the Time Sharing Control Task, and the Time Sharing Dispatcher.

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.

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, that is, 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).

MVT with Model 65 Multiprocessing

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

MVT with Model 65 multiprocessing can operate in two modes: multisystem mode and partitioned mode. In multisystem mode, all tasks are run under one control program. Both CPUs have access to all main storage and all I/O devices, except those devices that are supported asymmetrically. Each CPU has its own 4K-byte prefixed storage area (PSA) and can interrupt the other CPU through a direct hardware connection.

In partitioned mode, the two CPUs operate as two independent systems. Each CPU must have its own copy of the MVT with Model 65 multiprocessing control program, main storage units, and I/O devices.

Main Storage Hierarchy Support

Main storage may be expanded by including IBM 2361 Core Storage in the system (excluding the Model 65 Multiprocessing 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.

System Management Facilities

System Management Facilities (SMF) is an optional feature of the control program. This feature can be selected at system generation. System Management Facilities gather and record information used to evaluate system, data set, and direct access volume usage. SMF functions are performed by job management, input/output support, DADSM, and supervisor routines. The supervisor performs the following SMF functions:

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

For a detailed description of the implementation of these functions, see Section 11, "Special Features."

Tracing Facilities

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

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

Generalized Trace Facility: The Generalized Trace Facility (GTF) is invoked as a system task when the operator issues the START command. When GTF is started, the operator selects specific events to be traced and may select to record the trace data either in main storage or on an external device. When the internal storage option is selected, the recorded data is comparable to that provided by the optional Trace Table facility. When the external device storage option is selected, the recorded data is more comprehensive. When GTF is active, the optional Trace Table facility, if present, is disabled. When the trace records are maintained in main storage, ABDUMP provides formatted trace data to abnormally terminated users when a SYSABEND DD statement has been included. When trace records are stored on an external device, a trace EDIT function of IMDPRDMP can be used to provide the output of selected data.

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 straightforward. 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.

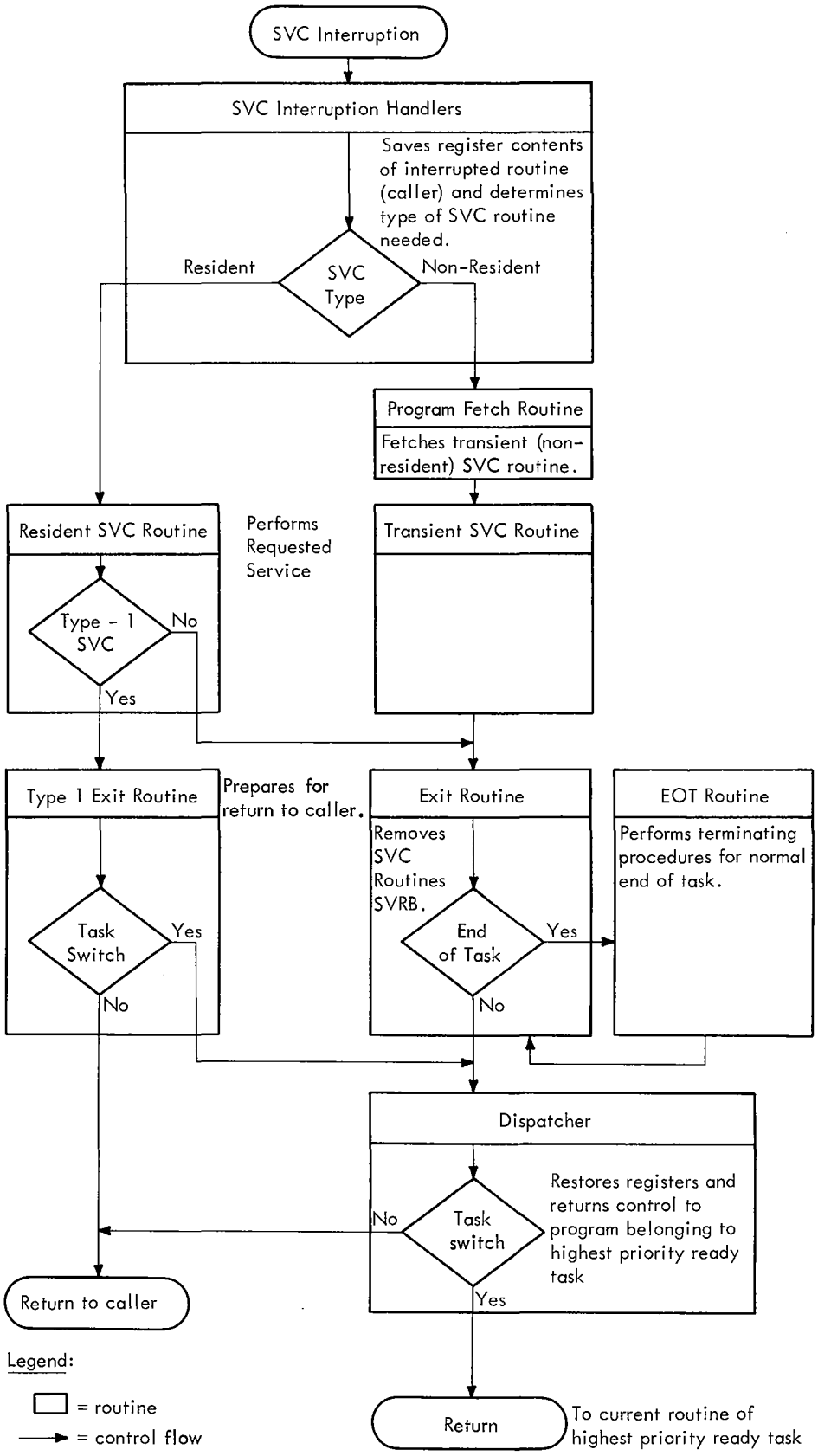


Figure 1-3. Processing After an SVC Interruption

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 determines whether the interruption is a monitor call request, or a program check. On monitor call requests, control is returned directly to the user's program after GTF processing; on program checks, 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.

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.
- Requests the Generalized Trace Facility (GTF), if GTF is active, or the optional Trace Table facility (a system generation option) to record the interruption.
- In MVT with Model 65 multiprocessing, 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, (SVC FLIH), and the SVC Second-Level Interruption Handler, (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 to a routine of another task that issues an SVC instruction.

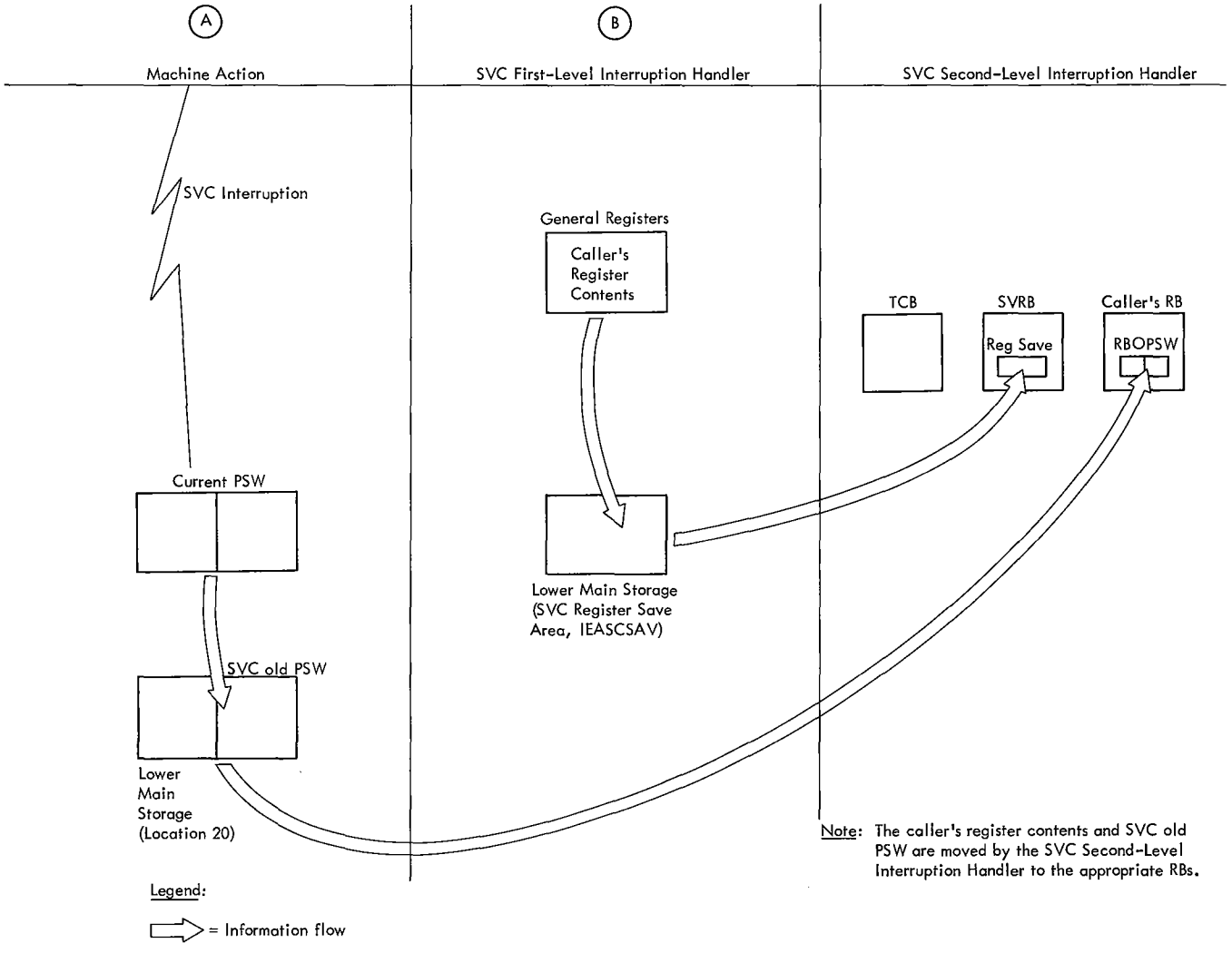


Figure 2-1. Status Saving by the SVC Interruption Handlers

Ensuring That Both CPUs in a Model 65 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 affinity bytes in the multiprocessing CVT. If the lock byte is not set, neither CPU is performing a disabled supervisor routine. When one CPU obtains possession of disabled Supervisor code, the SVC FLIH routine sets the supervisor lock byte and places the CPUID in the CPU affinity byte. If one CPU tries to get possession of disabled Supervisor code, but finds the supervisor lock byte is set, the CPU tests the CPU affinity byte to determine which CPU set the byte. If set by the testing CPU, the SVC routine proceeds; if not, the SVC old PSW is set (that is, the SVC old instruction address is decremented by two or four) to reissue the SVC* instruction.

If the SVC old PSW is enabled for external interruptions, the SVC* instruction is reissued via the Type-1 SVC Exit routine, which restores registers and loads the SVC old PSW. The SVC* instruction is thus reexecuted until the CPU in possession of the lock byte releases it.

If the SVC old PSW is disabled for external interruptions, the CPU that is executing disabled Supervisor code branches to the External FLIH routine so that external signals (such as a malfunction alert) from the other CPU can be received. The SVC* 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 the byte is unlocked. Between repeated tests of the lock byte, the CPU that is testing the byte is enabled for external interruptions. After the supervisor lock byte has been unlocked 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, the SVC* instruction is reissued.

---

*Either an SVC instruction or an EXECUTE instruction that executes an SVC 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.

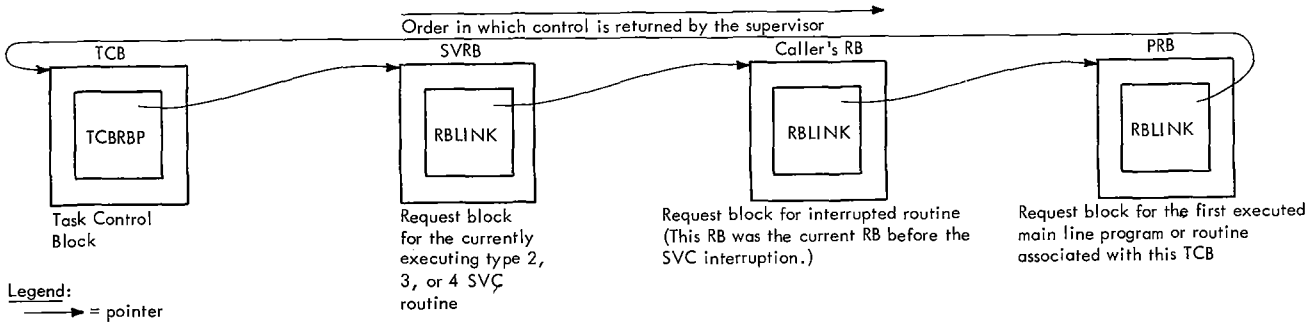


Figure 2-2. A Request Block Queue

Constructing, Initializing, and Queueing 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, 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* in the TCB points to the SVRB which is the "current" or head RB on the queue. The other pointer* 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.

The TCBATT flag in the current TCB is set to indicate that a system routine is executing and requires that this task not be interrupted by an attention exit for a time sharing task or by the STATUS SVC routine.

---

*In this chapter certain terms are accompanied by superscripts. The terms represent fields of control blocks that are defined in Figure 2-3.

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.

Figure 2-3. Control Block Fields Used by the SVC Second-Level Interruption Handler

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 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 a type 3 or type 4 SVC routine has been made resident, its entry in the SVC table is the same as that of a type 2 routine, and is accessed in the same manner. If the test of the SVC table entry reveals that the needed routine is resident in the nucleus, or is a type 3 or the first load of a type 4 routine preloaded into the link pack area, 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.

Extended SVC Router (ESR)

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

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

ESR Processing for Type 3 and 4 Supervisor Service Routines: The Type 3 and Type 4 ESR Supervisor Service Router is invoked via issuance of SVC 109 (Entry Point IGC109). The ESR code in register 15 is checked to insure that it is not greater than the highest assigned number. If the ESR code is not valid, the invoker is abnormally terminated. Otherwise, the linkage to the appropriate supervisor service routine is accomplished by an XCTL to that routine. The XCTL parameters are developed by converting the ESR code passed in register 15 to a 3-character EBCDIC number which is appended to a prefix (IGX00) to form an ESR name. For example, if the binary value in register 15 is 89, the resultant ESR name would be IGX00089. This ESR name represents the first load of an SVC and is used as input to the XCTL SVC. Control is now transferred to the module represented by the ESR name via an XCTL. Any subsequent loads of the Type 4 routine are named by incrementing the third and fourth characters of the original name. For example, several loads would be named IGX00089, IGX01089, and IGX02089.

The Transient Area Handler

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 2-4.) During its search, the transient area handler bypasses any TACT entry that indicates its transient area block is being loaded.

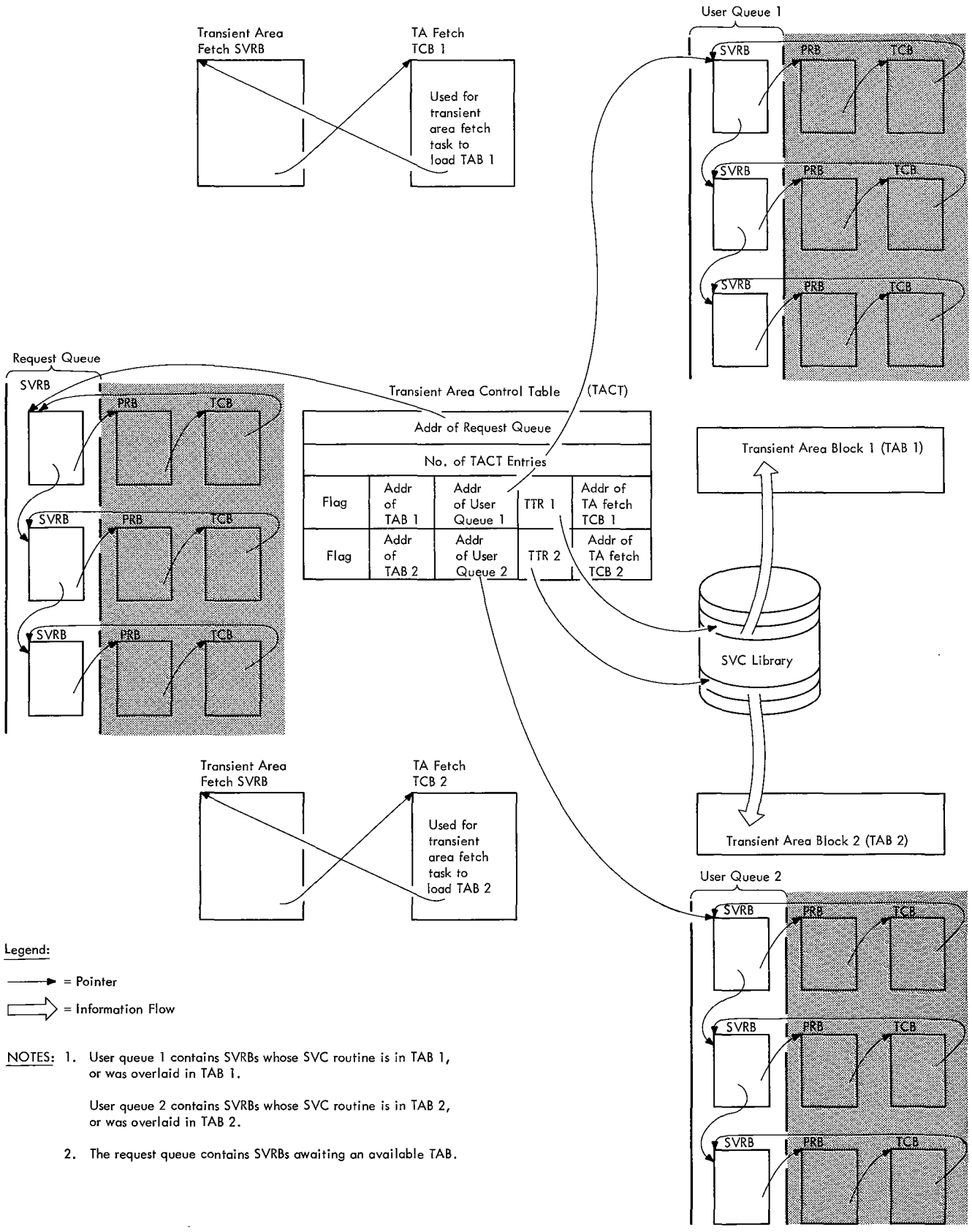
The TACT (see Section 12, "Control Blocks and Tables") contains one entry for each transient area block. Each entry contains four words: the address of the transient area block, the address of a "user queue" of SVRBs representing nonresident routines currently sharing a transient area, the relative track 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).

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 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



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 2-4. The Transient Area Queues

"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, and 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 (hereafter called the TA 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 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 the BLDL or FETCH routine encounters a permanent I/O error, the TA FETCH routine recycles the BLDL or FETCH operation up to five times. The total count of recycles of BLDL and FETCH operations for one use of the TA FETCH routine cannot exceed five. The recycle count is kept in the high order byte of the FETCH TCB address field of the TACT entry corresponding to the Transient Area Block; it is re-initialized after each use of the TA FETCH routine. If the permanent I/O error persists after five recycles of the BLDL or FETCH operation, the TA FETCH routine passes control to the Dynamic Device Reconfiguration SYSRES Effector, if DDR SYSRES support is in the system. DDR SYSRES returns control to the TA FETCH routine with a return code of 0 or 4. If the return code is 0, the BLDL or FETCH operation is recycled again once. If the return code is 4, the task which requested the SVC routine is scheduled for ABEND with a completion code of C06. If the renewed attempt to recycle the BLDL or FETCH operation results in a permanent error, DDR SYSRES is not invoked again. Instead, permanent error processing takes place; the task which requested the SVC routine is scheduled for ABEND with a completion code of C06.

If the time sharing option is operational in the the system, a TSEVENT macro instruction with the DONTSWAP operand is issued prior to FETCH or BLDL and FETCH processing. This indicates to the time sharing driver that a time sharing task cannot be swapped out of this time sharing region until FETCH or BLDL and FETCH processing is complete. When FETCH or BLDL and FETCH processing is complete (either successful or unsuccessful), a TSEVENT macro instruction with the OKSWAP operand is issued to permit the time sharing driver to resume swapping as necessary.

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 non-resident 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 communications 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 9, "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, loss of significance and monitoring. With the exception of monitoring, these events may be masked out.

The program interruption handler is automatically given control after any program interruption. An immediate branch is made to a Monitor Call Interrupt Handler routine to determine if the interruption was from either a System/370 or a System/360 simulated Monitor Call instruction. If it is, the Generalized Trace Facility (GTF) routines are used to record the event and control is returned directly to the user's program. Otherwise, the interruption is a valid program check and GTF is used to record the interruption and return control to the Program Check FLIH to continue pro-

---

*The queuing field is RBSVTQN.

cessing. Upon return, it tests whether the interruption occurred in the supervisor or in user code, by examining the program 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 Model 65 Multiprocessing System is in multisystem mode, the SSM instruction causes a program interruption. The system mask to be set is examined. If complete enablement is indicated, the supervisor lock and CPU affinity bytes in the multiprocessing CVT are reset to zero only if originally set by the CPU that is executing the Program First-Level Interruption Handler. The interruption is recorded by GTF (if active), before control is returned to the interrupted program with the system mask enabled.

If complete enablement is not indicated, the supervisor lock and CPU affinity bytes are tested to determine which CPU is executing disabled Supervisor code. If neither CPU is performing a disabled routine, the testing CPU sets the supervisor lock byte, and that CPU's ID is placed in the CPU affinity byte. The interruption is then recorded by GTF (if active), and control is passed to the interrupted routine with the system mask set as indicated. If the other CPU (not the testing CPU) is performing a disabled routine (supervisor lock byte set by other CPU), the program interruption old PSW is set (address is decremented) for reexecution of the SSM* 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* 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 (such as a malfunction alert) from the other CPU can be received. The SSM* instruction will be reissued when the calling task is 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 OPSW to reissue the SSM* 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 the byte is unlocked 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 unlocked 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* instruction.

---

*Either an SSM instruction or an EXECUTE instruction that executes an SSM 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 Supervisor lock and CPU affinity bytes in the multiprocessing CVT. If the lock byte is not set, the program FLIH routine sets the lock byte, places the CPUID in the CPU affinity byte, and the program FLIH routine proceeds as in MVT. If the lock byte was set by the testing CPU, the program FLIH routine proceeds. If the lock byte was set by the other CPU, it is tested until reset. Between repeated tests of the lock byte, the testing CPU is enabled for external interruptions by loading an enabled PSW so that the CPU can respond if the second CPU experiences a machine check 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.

MODELS 91 AND 195 PROGRAM INTERRUPTION HANDLER

For System/360 Models 91 and 195, and System/370 Model 195, 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 System/360 Models 91 and 195, and System/370 Model 195, 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.**

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 (for example, to a user exit, to the TESTRAN interpreter, or to a system task-terminating routine).

Entry from the TESTRAN Interpreter

On System/360 Models 91 and 195, and System/370 Model 195, 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 interpreter 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 Model 65

**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.

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. If the System Management Facility is included in the system, control is then passed to the SMF Wait Time Collection routine (described in Section 11). This routine returns control to the External FLIH routine.

The External FLIH routine saves the external old PSW in the current RB, and determines the cause of the interruption from the old PSW. If GTF is active, or if the Trace Table facility is present, the interruption is recorded by the tracing facilities. Control is then 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.

MVT WITH MODEL 65 MULTIPROCESSING

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 by the CPU on which a machine check occurs to the other CPU) 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 PSA of the receiving CPU to determine whether both CPUs are malfunctioning. If so, the receiving 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 malfunctioning CPU, causing it to execute the Recovery Management Support (RMS) routines. Before the External Start is issued, the supervisor lock byte is set for the malfunctioning CPU so that supervisor routines may be performed on that CPU. While that CPU performs the RMS routines, a completion flag (set when the malfunctioning CPU completes RMS) is tested. If the completion flag has not been set after a period of testing, the CPU waiting for the CPU to complete RMS enters the wait state. If the malfunctioning CPU is not in the wait state (that is, RMS was completed successfully), 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 either CPU requests a Halt I/O for I/O that was started on the other CPU, an external interruption is issued to the CPU on which the I/O was begun (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 the Halt I/O request flag in STMASK of the CPU on which the Halt I/O was requested is turned off. Control is 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 MVT with Model 65 multiprocessing 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 Dis-

patcher 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 affinity byte, and the interruption is recorded by the Generalized Trace Facility (GTF). If the lock has been set by the CPU that is executing FLIH, the FLIH routine continues. If set by the other CPU, the lock byte is tested until it is unlocked. Before each test, the testing CPU is enabled for external interruptions, and a bit in FLRETFLG is set to indicate that the interruption occurred during the External FLIH routine.

In multisystem mode, the 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.

If the System Management Facility (SMF) is included in the system, the input/output FLIH routine passes control to the SMF Wait Time Collection routine (described in Section 11). This routine returns control to the Input/Output FLIH routine.

In a Model 65 Multiprocessing System operating in multisystem mode, the supervisor lock and CPU affinity bytes are tested before the interruption is processed. If the lock byte is not already set, it is set by the CPU that is executing FLIH. This CPU also sets the CPU affinity byte. Interruption processing continues. If the lock has been set by the other CPU, it is repeatedly tested until it is unlocked. 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 multisystem mode, 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

There are three machine-check recovery programs and one channel error recovery program in OS/360 and OS/370. The availability of each is dependent upon the CPU and the other recovery management program(s) selected. Figure 2-5 indicates the availability of each of these programs.

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

[1]MCH cannot be used with SER0 or SER1.
[2]SER1 only.

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

Figure 2-5. Availability of Machine-Check Programs

The following two input/output recovery programs are not model dependent:

- The Alternate Path Retry (APR) option,* which consists of two functions: selective retry, which is optional for MVT and standard for M65MP; and VARY PATH, which is standard for both MVT and M65MP.
- The Dynamic Device Reconfiguration (DDR) option, which is not model-dependent but is automatically included for M65MP.

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.**
- 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.**
- 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.**
- If the Machine-Check Handler 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.**

SER0, SER1, and the Machine-Check Handler format the information they collect in machine check records which they then record on the SYS1.LOGREC data set. A description of the format and contents of the machine-check record for SER0 and SER1 is offered in Section 12.

If the Model 65 Multiprocessing System is operating in multisystem mode, a machine check causes one CPU to send a malfunction alert signal to the other CPU. The sending CPU enters the wait state via the machine-check new PSW. When the malfunction alert signal is received by the other CPU, the Second CPU Recovery Management System Interface routine (see "External Interruptions") gets control on the receiving CPU and issues an External Start to the malfunctioning CPU. The External Start causes the malfunctioning CPU to execute the RMS routines.

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:

---

*While it is not model-dependent, APR only performs its function usefully in a system with alternate paths and with the Channel-Check Handler.

**The operator may then load the System Environment Recording, Editing, and Printing (SEREP) program 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 no recovery program has been selected, the I/O Supervisor loads the machine check new PSW, and the CPU enters the wait state.*

- 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.**

- If the Channel-Check Handler for models using the 2860, 2870, 2880, or System/370 Models 145 or 155 integrated 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 provides an analysis of the channel logout information in the error recovery procedure interface block (ERPIB) to aid the appropriate device-dependent error recovery procedure in setting up for a retry of the failing operation by the I/O Supervisor. CCH also records environmental data about the channel error in a channel error inboard record entry. This record entry is later written onto SYS1.LOGREC by the outboard recorder routine (OBR) of the I/O Supervisor. An operator awareness message is issued each time a channel error is recorded. (For a full description of the Channel-Check Handler, refer to the Input/Output Supervisor PLM.

When a permanent I/O error occurs, OBR passes control to Dynamic Device Reconfiguration, if DDR is in the system. When a permanent I/O error occurs on a system fetch operation, TA FETCH, the error fetch sequence, or the DASD ERP passes control to DDR SYSRES, if DDR SYSRES is in the system. Both DDR and DDR SYSRES enable the operator to swap volumes on devices on which permanent I/O errors have occurred. (The operator may also request a swap at any time with the SWAP command.)

Alternate Path Retry also aids in the recovery from I/O errors, indirectly, by allowing an I/O operation that has developed an error on one channel to be retried on another channel. APR also allows the operator to VARY a path to a device online or offline.

---

*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.

For a full description of Dynamic Device Reconfiguration and Alternate Path Retry, refer to the Input/Output Supervisor PLM.

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 System Generation.

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 a machine check 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 is 90% full, a message is issued to the operator. The operator should then run the environment recording edit and print (IFCEREP0) service aid. IFCEREP0 formats the SYS1.LOGREC records, and then writes the records onto printer, tape, or disk, according to user specifications. IFCEREP0 is described in the Service Aids Logic PLM. If the operator delays in recording the contents of the SYS1.LOGREC data set, it will eventually become full; when it does, a message will be issued to the operator. He must then run IFCEREP0 immediately, or system performance may be degraded.

SER0

SER0 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 SER0 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 non-resident portion of SER0 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 SER0 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 machine check 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.

If the routine does not write an error record, it issues a message identifying the error.

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, 95, and 195.*

---

*The 195 applies to both System/360 and System/370 models.

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. An additional machine-check occurs 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.

Models 91, 95, and 195 can be interrupted by a special machine check called an external machine check. The SER1 routine for these models 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.

System Environment Recording with Multiple Console Support

The SER routines use WTO (SVC 35) macro instructions and SIO instructions to write messages to operator consoles. When operating with MCS, a WTO macro instruction (SER1 only) contains a routing code of 1 in the expansion, and the message is routed to the master console.

When using an SIO instruction, the master console is located using the communication task control tables. When the master console is a composite, a list of CCWs is constructed to write a message to the console. When the master console is a 1052 or a 2250, a CCW to ring the console alarm is added to the list. If the master console is not a 1052, 2250, 2740, or a composite console, no message is written.

After the SIO instruction has been issued, the device for the hard copy log is located. If it is the SYSLOG or a 2250, no message is written. If it is a 1052 or a 2740, the message is written but the alarm is not rung. For either case, the appropriate WAIT code is loaded into register 0 and an LPSW instruction puts the system into a wait state.

MACHINE-CHECK HANDLER (MCH)

This program is optional for the System/360 Model 65 and standard for the Model 65 Multiprocessor, the Model 85, and for all models of System/370. The Machine-Check Handler consists of a resident routine which always remains in main storage and of transient modules which reside in the SYS1.SVCLIB data set. The program attempts to recover from the cause of the machine-check interruption and record as much information as possible about the malfunction.

For the Model 65, MCH first determines if the instruction that was being executed when the machine check occurred can be retried, and, if this is possible, retries it. This step is handled by machine recovery facilities in the System/360 Model 85 and the System/370 Models 145, 155 and 165.

If, however, instruction retry is not possible, 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. In some models, the Machine-Check Handler is able to correct defective SPF keys or damaged code in main storage.

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. However, in the Model 65 Multiprocessing System, the current task is not set nondispatchable in the event of a solid storage failure. Instead, the Storage Reconfiguration facility schedules a selective ABEND* for that task, and logically removes the failing storage from the system. If possible, system operation is resumed.

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 to format and print diagnostic information from the CPU logout area.

For a complete description of the Machine-Check Handler program consult the manual appropriate for the model being considered.

- Machine-Check Handler for the Model 65 PLM.
- Machine-Check Handler for the Model 85 PLM.
- Machine-Check Handler for the System/370 Models 135 and 145.
- Machine-Check Handler for the System/370 Models 155 and 165 PLM.

---

*A selective ABEND may result in unclosed data sets.

SECTION 3: TASK SUPERVISION

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 whether the parent task or the new subtask will receive control next from the Dispatcher. In a Model 65 Multiprocessing System, the new subtask may receive control on either 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 Specify Task Abnormal Exit (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 the section "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 216 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 232 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 issuing a CIRB (Construct IRB) macro instruction with the WKAREA=29 parameter, which requests the CIRB routine to obtain a work area of 29 doublewords in addition to space for an IRB. The additional space is used for the TCB and the IQE. 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.

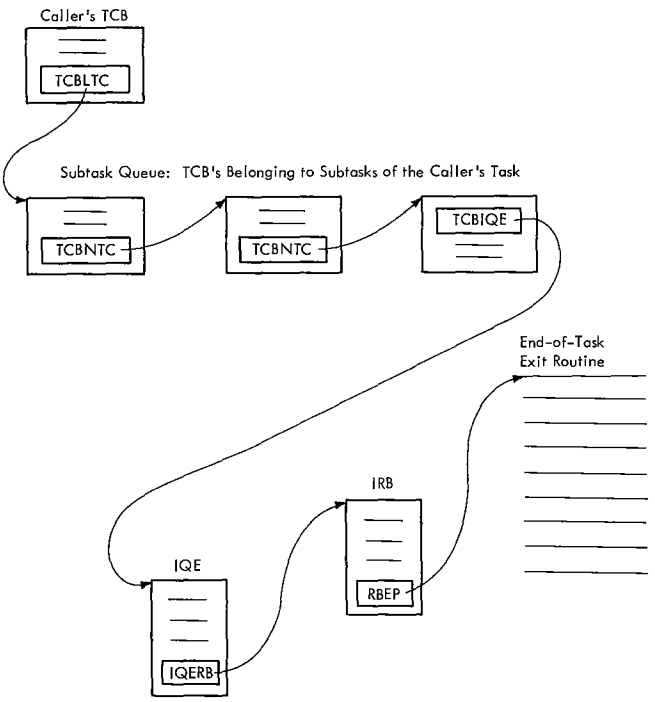


Figure 3-1. Queue Relationships among a TCB, IQE, IRB, and End-Of-Task Exit Routine

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 Figure 3-2.

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.

Note: If the ATTACH macro instruction was issued with the STAI operand, a subtask ABEND intercept (STAI) SCB is created, and any STAI SCBs queued to the TCB of the requesting program are propagated to the new subtask TCB.

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.

The Attach routine does not propagate the JOBLIB field if TASKLIB=dcb address was specified. Instead, the TASKLIB dcb address is placed in the JOBLIB field.

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

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

Figure 3-2. Initialization of the Interruption Queue Element

all TCBs within a job step. These fields include 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). If the System Management Facility is included in the system, a pointer to the timing control table (TCBTCT) is also propagated.

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); the TCBFRA flag; the initialization of the TCBJSTCB field; the initialization of the TCBJPQ field; the initialization of the protect key field (TCBPKF), if needed; the initialization of the save area pointer (TCBFSA), if needed; the initialization of the pointer to the SPQE chain (TCBMSS), if needed; and the address of a job step control block (JSCB).

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 macro instruction was issued by a time sharing task, the real limit and dispatching priorities and the time sharing limit and dispatching priorities are propagated to their respective fields in the subtask TCB. The time sharing priorities will be adjusted by the input parameters, if any, before the resulting values are stored in the subtask TCB.

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.)

Programs operating with a PSW protect key of zero may specify whether the job step pointer is to be propagated and whether the job pack queue is to be given to the subtask. The Attach routine tests the input parameters to determine the following: if the TCBJSTCB pointer should be copied from the task's TCB or made to point to the attached TCB; if the TCBJSCB field should be copied from the requesting task's TCB or made to point to a new JSCB whose address is given as input; and if the job pack queue is to be given to the subtask.

If the program operating with a PSW protect key of zero requests that the job pack queue be given to the subtask, the job pack queue cannot contain CDEs with use counts greater than zero -- for these represent active programs. If such CDEs are found, the Attach routine abnormally terminates the caller's task by issuing an ABEND macro instruction with a "32A" error code. Otherwise, the TCBJPQ field is copied from the attaching task's TCB to the attached task's TCB, and then zeroed in the attaching task's TCB. Also, the chain of SPQEs on the attaching task's TCB is searched for subpools 251 and 252. If an SPQE is found for either or both of these subpools, it is dequeued from the attaching task's TCB and queued on the attached task's TCB.

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 onthe 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 (that is, 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. If subpool zero is specified with the "give" or "share" parameters, the specification is ignored.

The Attach routine tests the SZERO parameter to determine whether subpool zero is to be shared. If not, the Attach routine has completed main storage allocation for the subtask and proceeds to its next function. However, if subpool zero is to be shared, the Attach routine searches the subpool queue of the parent task and creates the needed subpool queue elements, which are then chained from the parent and/or subtask TCB.

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).

If the ATTACH macro instruction was issued by a time sharing task, the new subtask TCB is placed on the queue using the time sharing dispatching priority. The Attach routine uses the time sharing job block extension (TJBX) to locate the user's subgroup on the queue. The Attach routine also determines whether there are enough entries in the quiesce parameter list for all active TCBs. If not, the current QPL is freed and a GETMAIN macro instruction is issued to obtain area for a new QPL that is twice as large as the QPL that was freed.

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 Model 65 Multiprocessing System, the parent and subtask TCBs may both have higher dispatching priorities than the current TCB on one 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 CPU on which the task switch must be made 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 dequeues its SVRB from the caller's TCB and queues it as the current RB for the new subtask. If a save area is not to be provided for the subtask, the Attach routine alters the old PSW in the SVRB so that it points to the 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. If a save area is to be provided for the subtask, the Attach routine alters the old PSW in the SVRB so that it points to a return point in the Attach routine where a save area is obtained when the newly created subtask becomes active.

By manipulating the register save areas, the Attach routine has established environments for the two TCBs. Also, the TCB and RB queues have been modified so that both TCBs 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.

When a save area is to be provided, the Attach routine goes to the Dispatcher only once; that is, before the subtask's Attach routine gets the save area. Then, instead of delaying the task by going to the Dispatcher again, the Attach routine branches directly to the special entry point in the Link function (IEAQCS01).

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, and the subtask has not been terminated, 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 3-3 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.

If the issuer is a time sharing task, the time sharing dispatching and limit priorities in the TCBs are used instead of the real dispatching and limit priorities. Only tasks within a user's subgroup, as defined by the time sharing job block extension, can be affected by CHAP. The changed TCB is placed on the TCB queue immediately preceding the TCB with the next lower dispatching priority.

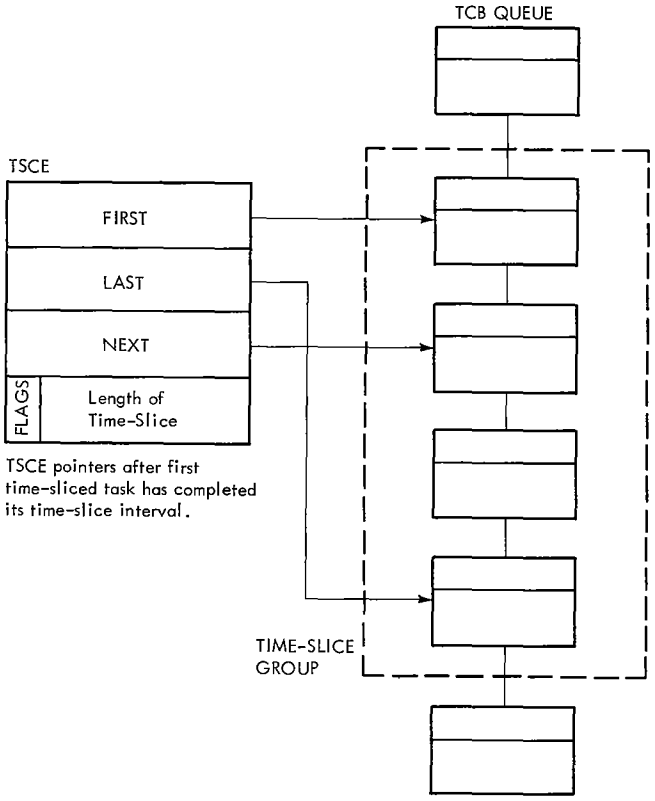


Figure 3-3. TSCE Pointers

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 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.

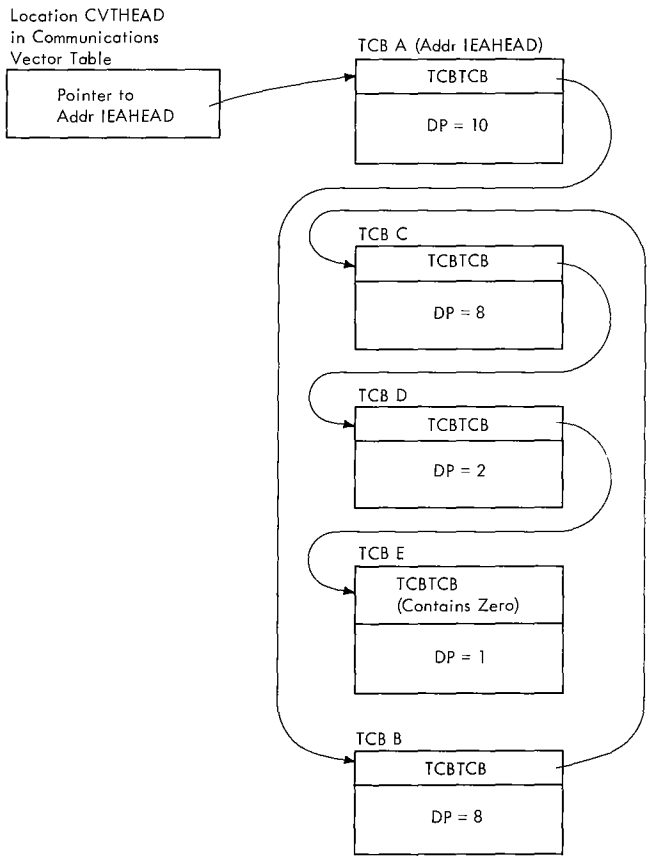
Having changed the dispatching priority of the TCB, the CHAP routine must next realign the TCB queue so that it is ordered from high to low dispatching priority. This queue is sometimes used by the Dispatcher during the exiting procedure to determine the highest priority ready task whose current routine 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 3-4.) The address IEAHEAD is contained in a field (CVTHEAD) of the communications vector table. This table, also called the CVT, contains pointers to major control blocks used by the control program.

Note on Figure 3-4 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.)

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



Legend: DP = dispatching priority value

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

Figure 3-4. The Task Control Block Queue

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 or a subsidiary control block the information contained in eleven 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 eleven fields or from all of the eleven 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 Supervisor Services and Macro Instructions, under the heading of "Extract."

Besides extracting information from a specified TCB or a subsidiary control block, 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 list is the user-specified table in which the Extract routine places the requested TCB information. One test determines if the list address lies on a fullword boundary, as required. Another test checks whether the list address lies within the boundaries of main storage. The remaining test determines if the list address specifies a storage area whose storage protection key matches the protection key in the caller's TCB. If any of these tests 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 Supervisor Services and Macro Instructions). 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.

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), FRS (floating-point register save area), the AETX (end-of-task exit routine), and COMM (CSCB communications list), PSB (protected storage control block), and the TJID (TSO terminal job identifier). 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, AETX, 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.) For the next two parameters, the returned values (addresses of the communications list and the protected storage control block) are also obtained indirectly. The TCB points to the JSCB, which contains the address of the CSCB communications list and the PSB. The TJID field contains the TSO terminal job identifier value.

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 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.

If the specified subtask has not completed processing, Detach determines if the requester has specified STAE=YES as an operand of the DETACH macro instruction. This will allow the STAE exit routine to be entered during subsequent ABEND processing. If STAE=YES was specified and the subtask is not already being terminated by ABEND, Detach determines if an IQE is pointed to by the subtask TCB. If so, the IQE is freed and, if the use count in the IRB is not greater than 1, the IRB is also freed. If the use count is greater than 1, it is decremented but the IRB is not freed. For all cases where STAE=YES and the subtask is not being terminated by ABEND, Detach loads the error code 0033E000, resets the stop/start nondispatchability flag, and, if no secondary nondispatchability flags are set, clears the primary nondispatchability flag. Detach branches to ABTERM to schedule abnormal termination of the subtask and then returns control to the caller.

If the requester had specified STAE=NO, or if STAE=YES was specified but the subtask is already being terminated by ABEND, Detach loads the error code 0013E000. If the subtask is not being terminated by ABEND, Detach resets the stop/start nondispatchability flag and branches to ABTERM to schedule abnormal termination of the subtask.

The Detach routine next 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[2] 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 release of the Detach routine from its wait condition via 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

---

[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.

---

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

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 resumes normal processing by removing the subtask TCB from the subtask queue originating in the TCB of the requesting task.

If, however, the subtask has an ECB, Detach branches to the entry point in the Post routine that provides validity checking for the ECB, and, if the ECB is valid, posts the ECB. Upon return from the Post routine, Detach resumes processing by removing the subtask TCB from the subtask queue.

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 asynchronous 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 Handler 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 requester 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 requester so that it may be placed into execution by the Dispatcher.

By serializing the use of resources, the ENQ and DEQ routines permit requesters 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 requester, 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 requester. The requester is placed in a wait condition, pending the availability of the resource, or control is returned to the requester 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 requester and tests it for readiness. If the requester is now ready, the DEQ routine determines if the requester 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 requester'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 supervi-

sor's Exit routine dequeues the IRB from its TCB, and if the IRB was dynamically acquired, frees the storage space it occupies.

An asynchronous (user) exit routine can also be specified to receive control when a task is scheduled for ABEND processing; however, the processing to handle this exit routine is considerably different. 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. (See Section 10, "Termination Procedures," for a detailed description of the STAE service routine and the ABEND/STAE Interface routines.)

SPECIFYING A PROGRAM INTERRUPTION EXIT ROUTINE

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

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 SPIE 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

---

[1]Space is allocated in subpool zero.

program-check routine address is then 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.

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 non-reentrant) 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 examines the search bit in the RB of the waiting program. If the search bit (RBECBWT) is set, the Post routine clears the wait bit in each of the ECBs not yet posted and also clears 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 Services 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.

If a validity error is detected, the Wait routine tests if a communications ECB (which is in storage with a protection key of zero) was specified. If yes, normal Wait processing continues. If a communications ECB was not specified, the Wait routine exits to the ABTERM routine.

The routine next tests bits in each specified ECB. (See Section 12 for 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 (that is, 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 caller's RB. This RB address is needed by the Post routine after an awaited event has occurred, when it wishes to adjust the waitcount 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; that is, 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. The Wait routine also turns on the "Wait pending" flag (RBWAITP) in the current RB. 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. If the System Management Facility is included in the system, the Wait routine checks the initiator TCB for the address of a timing control table (TCT). If there is a TCT, the job-step wait time limit, not the 30-minute wait limit, is placed in the TQE. The job-step wait time limit appears in the TCTWLMT field of the TCT. If there is no TCT, the 30 minute value is used.

The TQE is then enqueued on the timer queue by the Enqueue TQE routine (entry point IEAQTE00). The job-step timing algorithm necessitates job-step TQE manipulation.

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, 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 (that is, 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 (that is, 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 does 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.

If the TJID is specified for a time sharing task and the TCBINCOR flag in the terminal job block indicates that the task is not in main storage, an Inter-partition Post block is created to record the requested post. The Post routine then issues a TSEVENT macro instruction specifying that the task whose ECB is to be posted be brought into main storage (swapped-in).

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. If a validity check of the ECB address is bypassed, a check is made to determine if this is an Inter-Partition Post. If it is, Post parameters are obtained from the user parameter list. If it is not, 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 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 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 asynchronous 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 (that is, 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, the Post routine turns off the "Wait pending" flag (RBWAITP) and 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 (for example, 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 theExit 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 is honored strictly on a task-priority basis. Figure 3-5 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, for example, the name of a data set. A major QCB thus represents a set of resources.

Each major QCB points to a minor QCB, which represents a particular resource within the set of resources, for example, a specific record within a data set. 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 moves the next QEL to the top of the queue. The task may be dispatched when all of its QELs are at the top of their respective queues, or in a group of shared requests at the top of the queue.

Figure 3-6 illustrates the resource queues. In Figure 3-6, 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

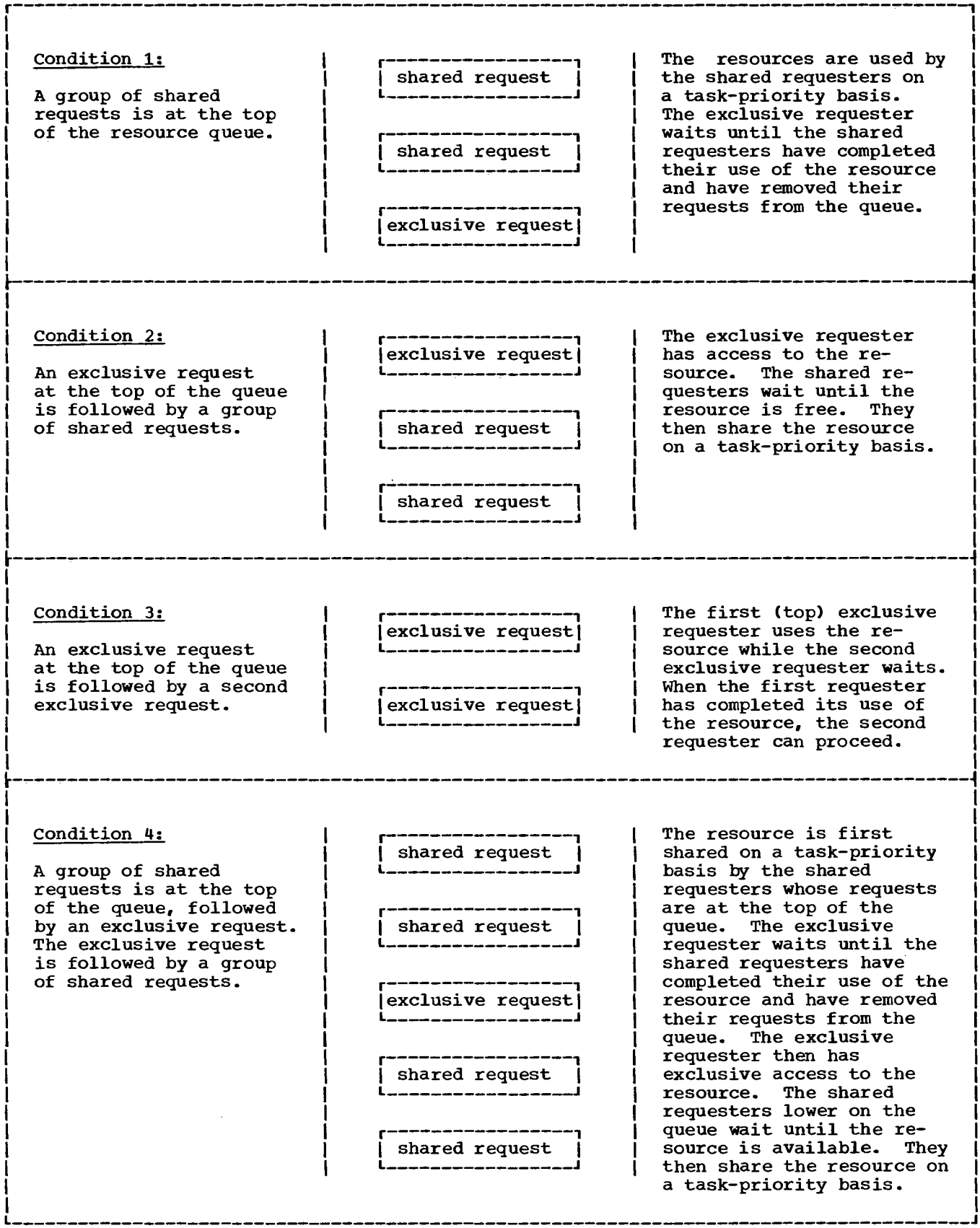
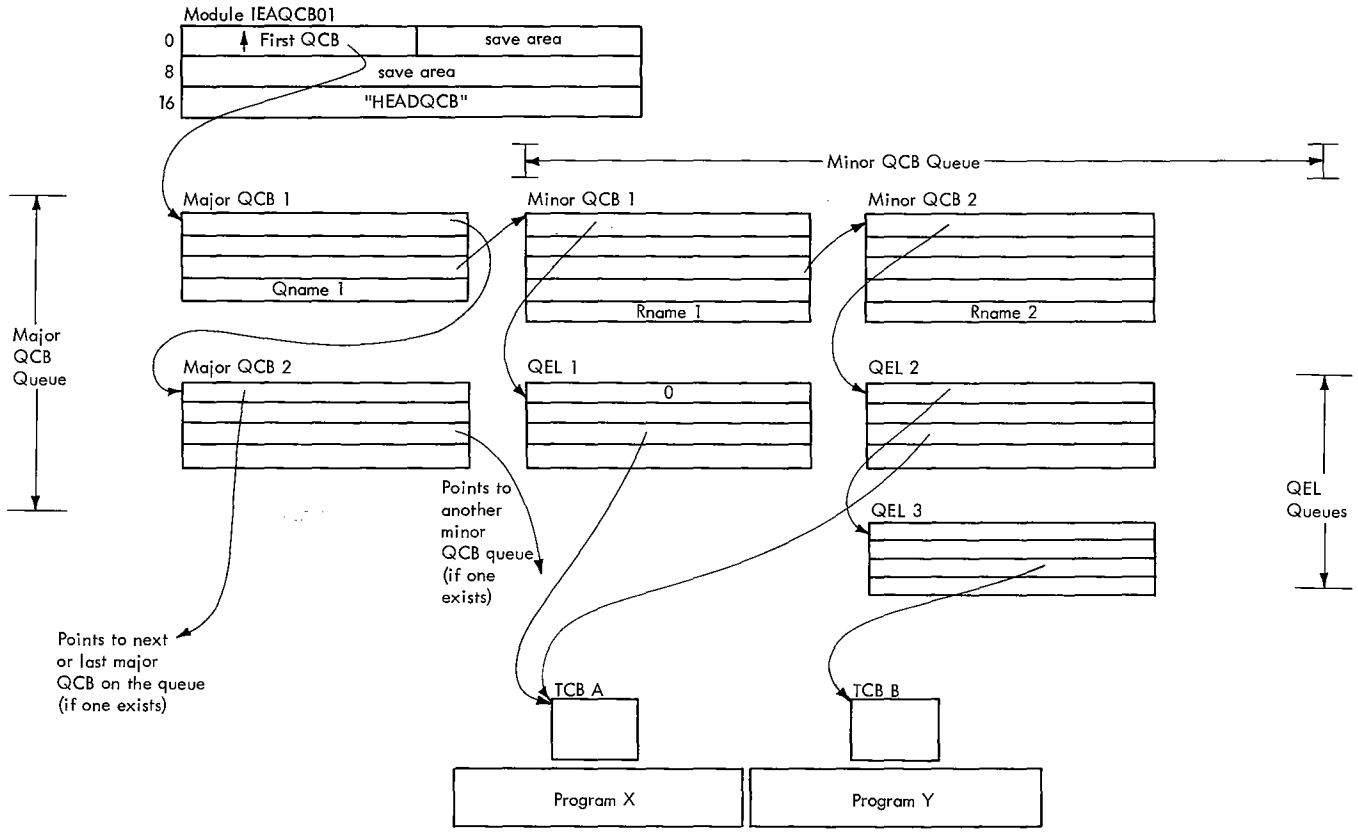


Figure 3-5. The Handling of Shared and Exclusive Requests



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 3-6. The Resource Queues

by major QCB 1 and minor QCB 2. Since the resources 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. 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 requester, with or without a return code that

indicates the availability of the resource, depending on the RET parameter.

- If the requested resource is not available, either of two functions are performed, depending on the RET parameter:
  - The requester is placed in a wait condition, pending the availability of the resource, or
  - Control is returned to the requester 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.

When the time sharing option is included in the system, step enqueue for a time sharing task causes the TJID to be placed in the minor QCB, in the first byte of the first QEL, and in the first byte of the previous minor QCB field.

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 Figure 3-7 for the various results.)

PROCESSING IF THE REQUESTED RESOURCE IS IN USE: If another requester 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.) Figure 3-8 lists the different forms of resultant processing.

Note in Figure 3-8 that a QEL is constructed and placed on a QEL queue if the requester wants access to the resource and is willing to wait for it. The requester'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 or CHNG never causes creation of a QEL, only the generation of a return code (see Figure 3-9) indicating whether the resource is available. If RET is USE, a QEL is created only if the requester can have immediate access to the resource (Part 2 of Figure 3-8).

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
CHNG	Requests a change of attribute from shared to exclusive on a resource for which the caller is enqueued. Never constructs control blocks.	no	8	Caller not enqueued for resource
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	

Figure 3-7. Processing if a Requested Resource Is not 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 requester 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 (that is, 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 requester only after the resource becomes available.

Figure 3-8. Processing if a Requested Resource Is in Use

Code	Meaning			
	RET = TEST	RET = USE	RET = HAVE	RET = CHNG
0	The resource is immediately available.	Control of the resource has been assigned to the active task.		Control of the resource has been assigned to the active task. Either the user was already enqueued for the resource with the exclusive attribute or the requested change from shared to exclusive was honored.
4	The Resource is not immediately available.		N/A	Requested change in attribute cannot be honored at this time.
8	A previous request for control of the same resource has been made for the same task.			The requesting task was not enqueued for the resource when the attribute change from shared to exclusive was requested.

Figure 3-9. Return Codes for the ENQ Routine

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 requester and those represented by the "shared" previous QELs on the queue may share the resource on a task-priority basis. Thus, a requester need not have its QEL at the top of the "shared" group of QELs. Any requester represented in the shared group may be executed if other requesters 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 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 Communications 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 "non-rolloutable 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.

If a time sharing task has requested "set must complete" processing, a TSEVENT macro instruction with the REQSTMC operand is issued to indicate to the time sharing driver that ENQ processing is not to be interrupted.

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. (If RET=CHNG, an error condition does not exist since the caller must be previously enqueued in order to request a change in attribute.) These error conditions 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, or a routine invoked by ABEND, 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 requester 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.

An "enqueue count" is maintained in the requester'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 requester'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 terminates the task by setting it and its related tasks nondispatchable. A message is issued 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 ABEND1 in "Termination Procedures.")

The ENQ routine makes several checks to determine if the requester's task should be set in "must complete" status (see Figure 3-10). The ENQ routine tests whether the following requirements have been met:

- The requester is a system routine, as indicated by a zero protection key in the requester'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.)

---

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

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.

Figure 3-10. TCB Flags that are Set if a Task Is in "Must Complete" Status

The processing varies, depending on the outcome of the tests. If all requirements have been met, 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 requester'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 requester'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 requester's TCB. One flag, when set, prevents the Stage 3 Exit Effector from scheduling user exit routines for the requester'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 (that is, 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 requester is not a system routine, it sets up an error code (338) and invokes the ABEND routine to abnormally terminate the requester'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, the ENQ routine points the SVRB old PSW to a restart point in the "set must complete" coding. When the requester's task is redispatched, after the resource becomes available, the restarted ENQ routine will set the requester'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.

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 issuing the ABEND macro instruction, causes supervisor linkage to

the ABEND routine to abnormally terminate the caller's task. If the RET option is CHNG, an error condition does not exist since the caller must be enqueued for the resource if he is requesting a change in attribute. If the RET option is neither NONE nor CHNG, 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 ninth 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 ABEND9" 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 reservecount 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 SVRB 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 requester represented on the QEL queue, reduction of the wait count in its SVRB, and testing if the requester is ready to resume execution.
- Determining if a readied requester 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 requester's task is of higher priority than the caller's. If a readied requester's task is of higher priority than the caller's, control is returned to the requester instead of to the caller.

Updating the Resource Queues: 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 Requester Should be Readied: After the old top QEL is dequeued, the DEQ routine determines if the next waiting requester, represented by the new top QEL, should be readied. The decision is based on the type of new top QEL, shared or exclusive, and on the type of dequeued QEL. According to the result of the decision, the SVRB wait count for the waiting requester 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 3-11.

If the new top QEL is associated with a time sharing task that is not in main storage (swapped-out), a TSEVENT macro instruction with the USERRDY operand is issued to indicate to the time sharing driver that the time sharing task is to be brought into main storage (swapped-in).

Determining if a Readied Requester 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 requester can be dispatched. If the requester's task is of higher dispatching priority than the caller's, the requester 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 requester's TCB has a higher priority than the caller's, the Task Switching routine indicates this fact to the Dispatcher by placing the requester's TCB address in the "new" TCB pointer, IEATCBP.

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

Figure 3-11. Determining if the Next Waiting Requester Should be Readied

Returning Control: The DEQ routine returns control to the caller or a readied requester, 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 requester, 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 requester 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:

- 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 "nonrolloutable 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.

If "reset must complete" processing has been performed during DEQ, a TSEVENT macro instruction with the RELMC operand is issued to indicate to the time sharing driver that this processing has been performed.

The DEQ routine clears the "must complete" nondispatchability flag (see Figure 3-10) 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 Figure 3-10). 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 Figure 3-12.

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.

Figure 3-12. Error Conditions when Use of a Resource is Signaled Complete

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 Figure 3-12.

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.

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 ABEND13 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 ABEND13 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 (for example, tape label checking or I/O error checking). The scheduling of user exit routines (sometimes called asynchronous exit routines) is handled by several supervisor routines: the Stage 1 Exit Effector, the Stage 2 Exit Effector, the Stage 3 Exit Effector, and the Exit routine. Note that these routines do not schedule the execution of user program check routines. ABEND processing that results from a program interruption can be intercepted by a SPIE macro instruction or, in the absence of SPIE, by a STAE macro instruction. See "Specifying a Program Interruption Exit Routine" for a description of the SPIE routine. See "Specifying a Task Asynchronous Exit Routine" in Section 10 for a description of the STAE routine.

As shown in Figure 3-13, 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 (for example, 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

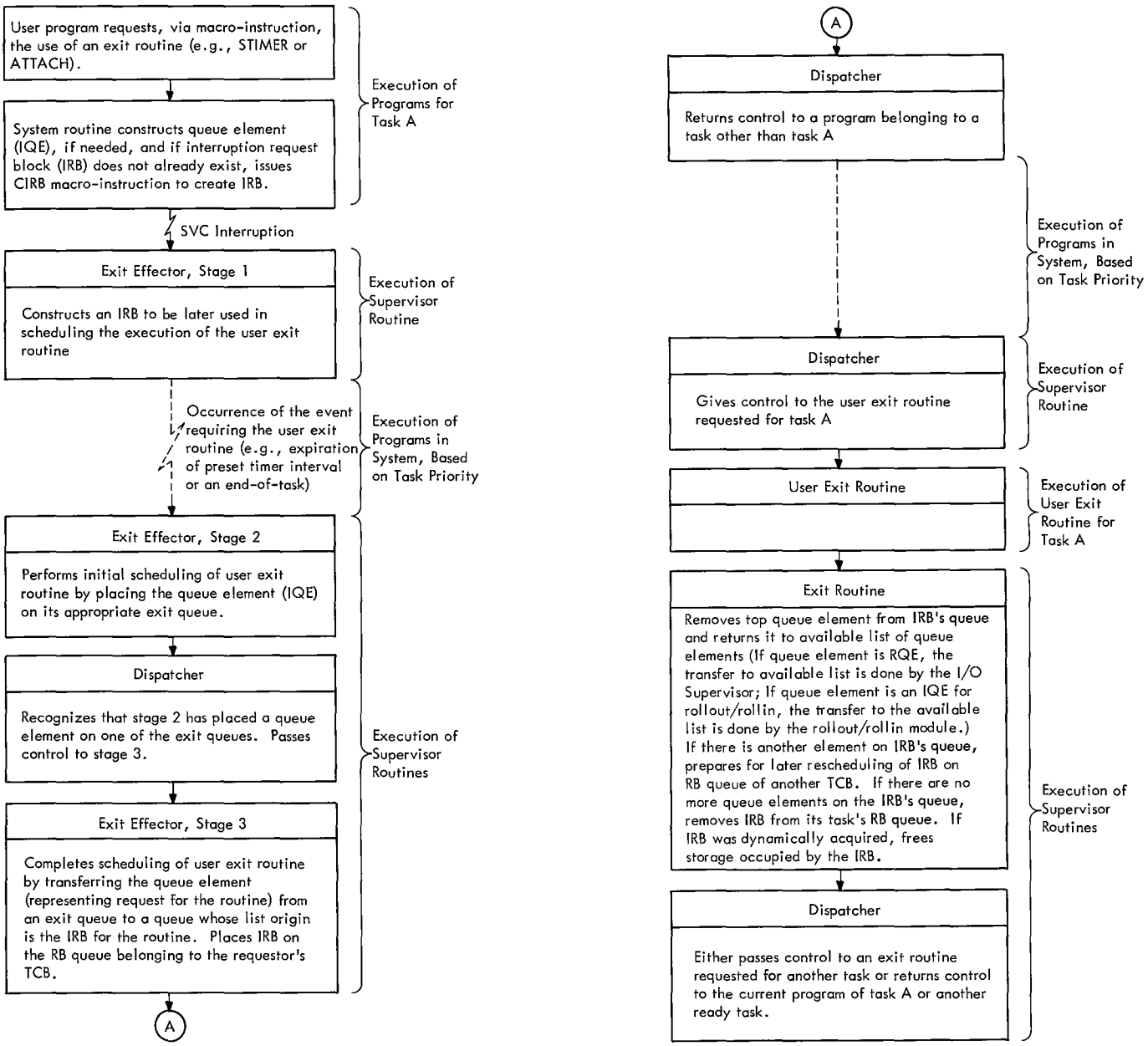


Figure 3-13. Scheduling of Asynchronous Exit Routines

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 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 attaching programs' 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, that is, 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, 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 represents requests for the use of an error routine, Stage 3 performs special processing for these RQEs. Other RQEs, 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," that is, 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 MVT with Model 65 multiprocessing, 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.

TSO Processing

If an IQE exists, the Stage 3 Exit Effector determines the type of IRB associated with it. If the IRB is not an inactive attention IRB, processing continues as described above. If it is an inactive attention IRB, the TCB associated with this IRB and all lower tasks in this user's task structure are checked to determine whether attention exits and asynchronous exits are permitted. If not, processing continues as described above. If these exits are permitted, the IQE is scheduled by queueing the IQE to the IRB in first-in first-out order.

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 compares the SIRB name field, which contains the name of the previously executed error routine, to the name field of the currently required error routine. If the names match, the error fetch sequence links to the previously determined entry point for the error routine.

If the names do not match and if the resident error recovery procedure option was selected during system generation, the name of the required error routine is compared with the name of the error routine last fetched into the I/O supervisor error transient area. If these names match, the error fetch sequence links to the start of the I/O supervisor transient area. If the names do not match, the chain of CDEs that describe the error routines made resident during nucleus initialization is searched. If the name field in one of the CDEs on this chain matches the name of the required error routine, the error fetch sequence obtains the entry point address for the resident error routine and links to this address.

If the resident error recovery option was not selected during system generation or if the required error routine is not resident, the error fetch sequence invokes the BLDL routine to get data set directory information in preparation for fetching the I/O error module.

If the BLDL routine encounters a permanent I/O error, the error fetch sequence recycles the BLDL operation up to five times. The total count of recycles of BLDL and FETCH operations (see below) for one use of the error fetch sequence cannot exceed five. The recycle count is kept in the error fetch sequence's work area; it is re-initialized after each use of the error fetch sequence. If the permanent I/O error persists after five recycles of the BLDL operation, the error fetch sequence passes control to the Dynamic Device Reconfigura-

tion SYSRES Effector, if DDR SYSRES support is in the system. DDR SYSRES returns control to the error fetch sequence with a return code of 0 or 4. If the return code is 0, the BLDL operation is recycled again once. If the return code is 4, the error fetch sequence sets up an error code (806) and branches to the ABTERM routine. If the renewed attempt to recycle the BLDL operation results in a permanent error, DDR SYSRES is not invoked again. Instead, permanent error processing takes place; 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 links to the entry point address of the I/O supervisor transient area. If, however, the FETCH routine encounters a permanent I/O error, processing continues as for BLDL (see above), except that the error fetch sequence posts a completion code of B06 when control must be passed to ABEND.

The Exit Routine

This discussion of the Exit routine includes only that part of its processing that affects the scheduling of user (asynchronous) exit routines. Other aspects of its processing are described 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 follows.

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 reen-

try 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 user 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.

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 of higher 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 (for example, 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 references this TCB pointer to determine the task and routine it should dispatch.

A detailed discussion of the operation of the Task Switching routine follows.

Upon branch entry from the calling supervisor routine, the Task Switching routine 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 dispatches 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, dispatches 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 MVT with Model 65 multiprocessing, 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 searches 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.

---

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

The Task Switching routine then returns control to the calling supervisor routine. Later the Dispatcher decides 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

Routines with a protect key of zero 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.

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.)

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 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 previously 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 Figure 3-10 in "Serializing the Use of a Resource.")

In MVT with Model 65 multiprocessing, 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 rou-

tine 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 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 jobstep'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 Figure 3-14.)

In MVT with Model 65 multiprocessing, 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.

When a user issues a STATUS macro instruction with the START or STOP operand, the Set Status routine determines if the specified subtask of the current task or all subtasks of the current task are to be modified. When START is specified, the stop/start count is decremented in the subtask TCB(s) and the nondispatchability flags are cleared if the count is zero. When STOP is specified, the stop/start count is incremented in the subtask TCB(s) and the nondispatchability flags are set. Any option other than START or STOP must be specified by a caller running with a protection key of zero. A task is set nondispatchable only if no system routines are executing for it as indicated by the TCBATT flag. If a system routine is executing for the task, the TCBSTPPR flag is set to indicate that this task should be made nondispatchable when it no longer has a system routine executing.

[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.

DETERMINING THE RELATIVE DISPATCHING PRIORITIES OF TASKS

The Relative Priority subroutine (entry point RELPRIOR) is used by the Task Switching and Dispatcher routines in MVT with Model 65 multiprocessing 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.

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 has the higher dispatching priority.

TESTING THE DISPATCHABILITY OF TASKS

The Task Removal subroutine (entry point TESTDSP) ensures, in MVT with Model 65 multiprocessing, 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 OF THE MODEL 65 MULTIPROCESSING SYSTEM

The First CPU Signal and SHOLDTAP subroutines are used by supervisor routines in MVT with Model 65 multiprocessing to cause one CPU to externally interrupt the other CPU. As a result of the external interruption, a routine specified in the word STMASK receives control on the interrupted CPU (see description of External FLIH routine). The word STMASK is located in the PSA, and the bit designating the routine to receive control on the interrupted 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 other CPU.

The First CPU Signal routine (entry point FLASH) is used when the interrupted 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 immediately requested service has been performed.

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.
3-6			Reserved.
7		32.6	This task is nondispatchable while VARY or QUIESCE processing is being performed in a Model 65 Multiprocessing System.
8	TCBONDSP	32.7	This task is nondispatchable while the dump data set is being opened for another task in the same job step.
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.

Figure 3-14. Mask Bit Numbers Used in the STATUS Macro Instruction

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 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 requester. 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 requester. 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 requester'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 fullword 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), if they are not already in storage. These routines are needed for later linkage. 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.

TSO Processing: If the time sharing link pack area modules have been loaded by the nucleus initialization program, minor CDEs must be queued off the major CDEs; the major CDEs are obtained from SQA.

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. This section describes each major function 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 13.

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

---

[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.

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 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.)

Note: If the time sharing option is included in the system, CDSEARCH searches the time sharing link pack area for a time sharing module entry point name.

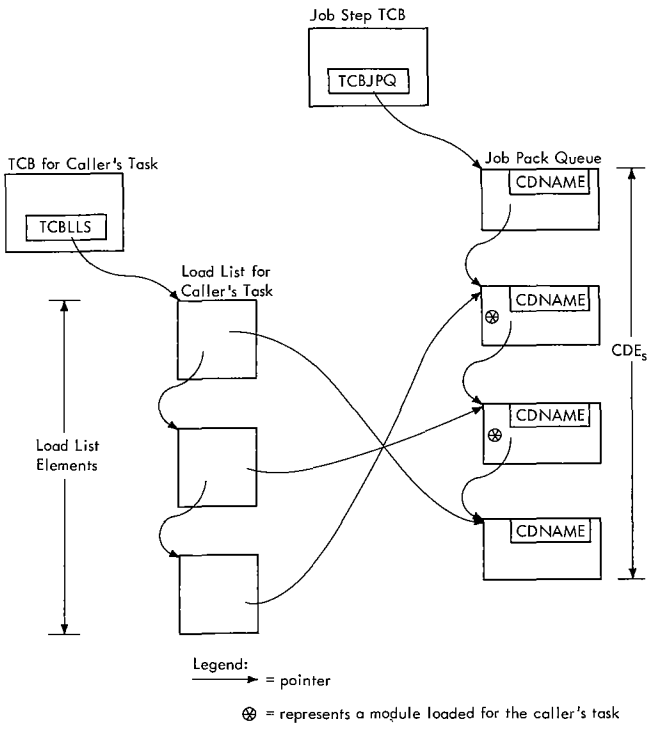


Figure 4-1. Subroutine CDSEARCH Uses the Load List and the Job Pack Queue in its Search for the Module's Name

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 task library, if one is present, is searched. If the module is not found or if a task library is not specified, each higher task's (parent tasks in subtask's hierarchical structure) task library is searched until the job-step task is reached, at which point the job library, if one is present, for the job-step task 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. In systems containing IBM 2361 Core Storage and Main Storage Hierarchy Support, a secondary link pack area may be constructed in hierarchy 1. 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 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.

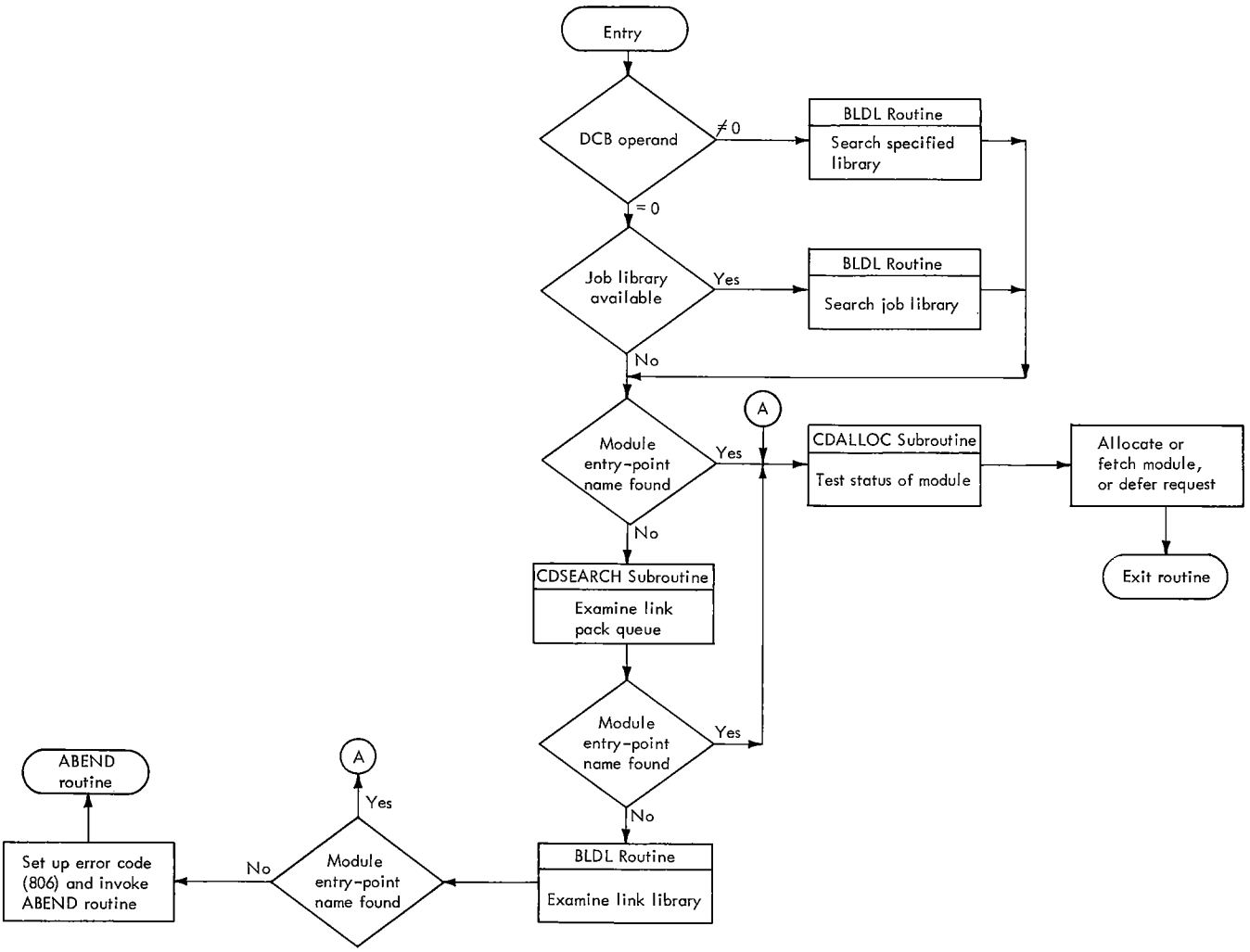


Figure 4-2. Further Search by the Common Subroutines of Contents Supervision if the Module's CDE Is not in the Job Pack Queue

(The 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 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 requesters; 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. 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 on the 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.

Note: If a build entry for a time sharing task is marked as an alias, the time sharing link pack area is searched for the entry.

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 input includes 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

---

[1]The Machine-Check Handler is optional system generation programming support for the System/360 Model 65 (MCH/65), and standard programming support for the Model 65 Multiprocessor, the Model 85 (MCH/85) and System/370. Refer to Section 2, "Interruption Handling."

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 requester'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 requester 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 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 occurs, 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"). The current task gains initial control of the module.

PERFORMANCE OF A TASK SWITCH: If the preparation for restart has altered the TCB pointer, a future branch to the Dispatcher causes 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, occurs during the execution of the scheduling subroutine CDEPILOG.

CDEPILOG (entry point IEAQCS03) tests if an available module should be allocated to the current requester, 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 requester 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 requester. 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, for example, 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.

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 requester'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 is removed and freed, leaving the PRB as the current RB for the requester's task. After queuing and initializing the newly created PRB, which controls the module's execution, CDEPILOG passes control to the module or to a restarted requester, 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 requester. 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 requester 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 Requester Is a User Program or a User Exit Routine

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

If the requester is a user program, operating under control of a program request block (PRB), the requester'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 requester is a user exit routine, operating under the control of an IRB, the requester'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 requester'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 requester's IRB (see Figure 4-3, part E1).

Processing if the Requester Is an SVC Routine

If the requester 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 requester'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 requester is a resident or nonresident SVC routine. It tests the status bit (RBFNSVRB) in the requester's SVRB.

UPDATING THE TRANSIENT AREA QUEUE: If the requester is nonresident, the TAXEXIT subroutine removes the requester'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 requester's SVRB from the user queue is necessary because control is not returned to the requester. 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 requester'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 requester's SVRB is flagged as "resident" (the RBFNSVRB bit in the RBSTAB field is cleared). The requester'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.

User Program Issues XCTL Request

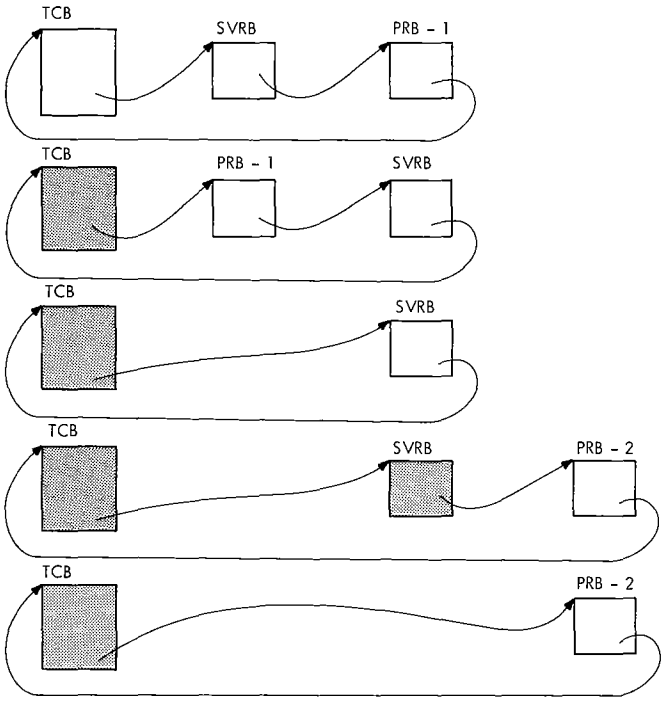
A. Condition of RB queue before XCTL processing.

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

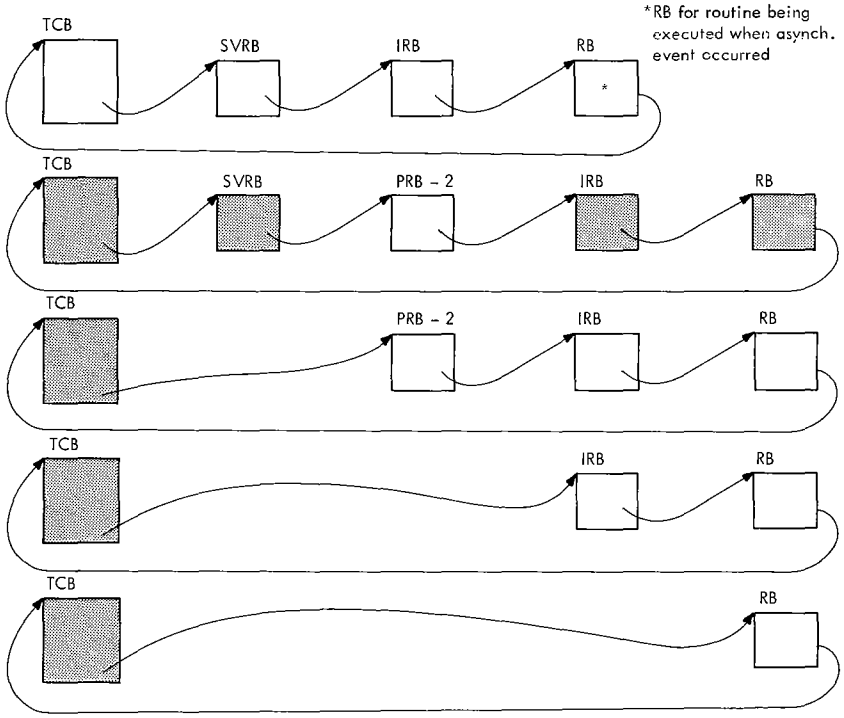
A1. Condition of RB queue before XCTL processing.

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

C1. SVRB is 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.

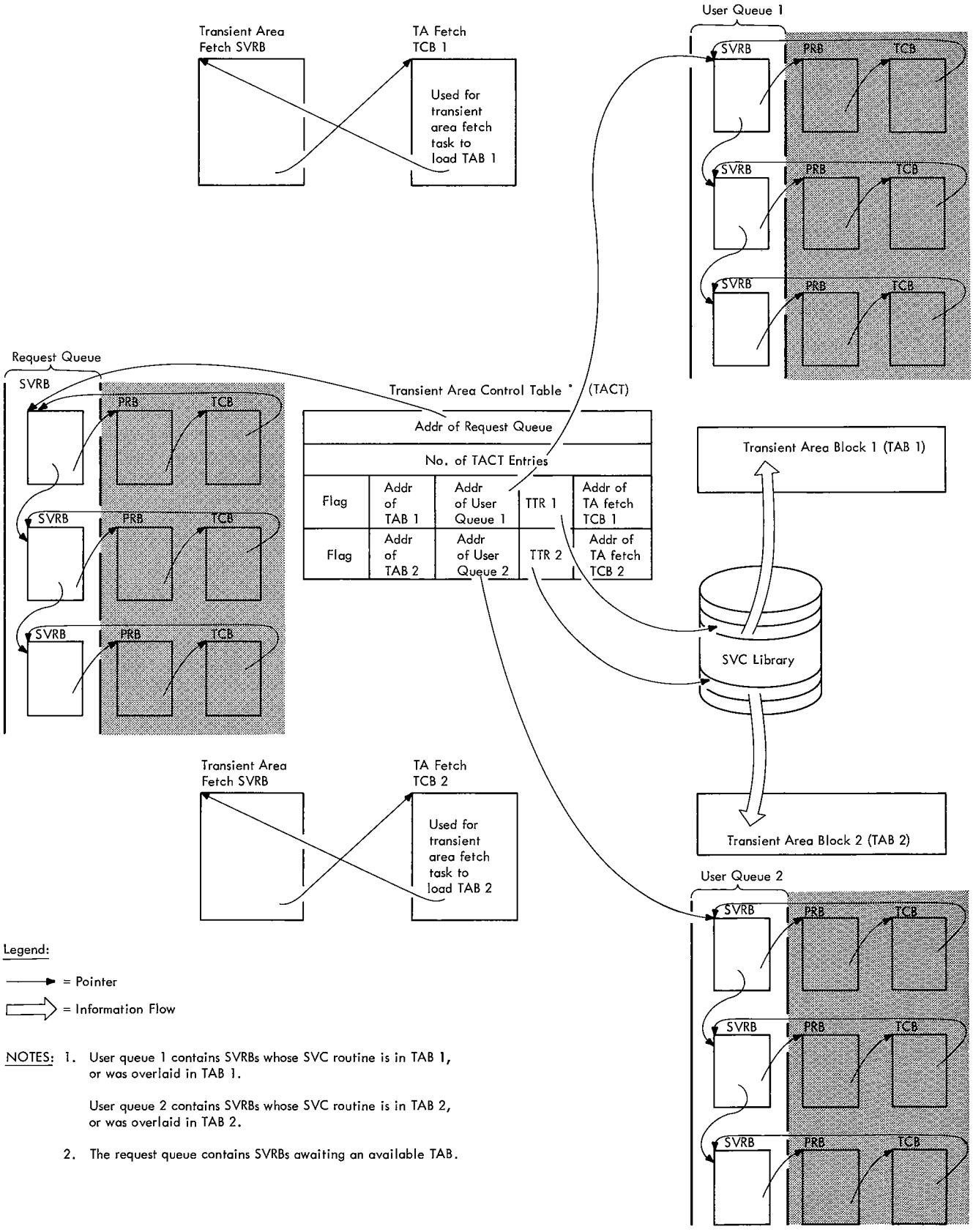
E1. Caller's IRB is removed by the Exit routine after the Dispatcher tries to restart the user exit routine.



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



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

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 requester'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 removes 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 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 steres data in the requester'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 requester'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 requester.

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 requester'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 is 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 (that is, 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 is 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 is entered when the Transient Area Fetch routine waits for I/O completion and the requester'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 of X'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.

The Identify routine prepares to search both the link pack area queue and 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 searches the CDEs of the queues 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 of X'08', 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 of X'04' 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, of X'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 rou-

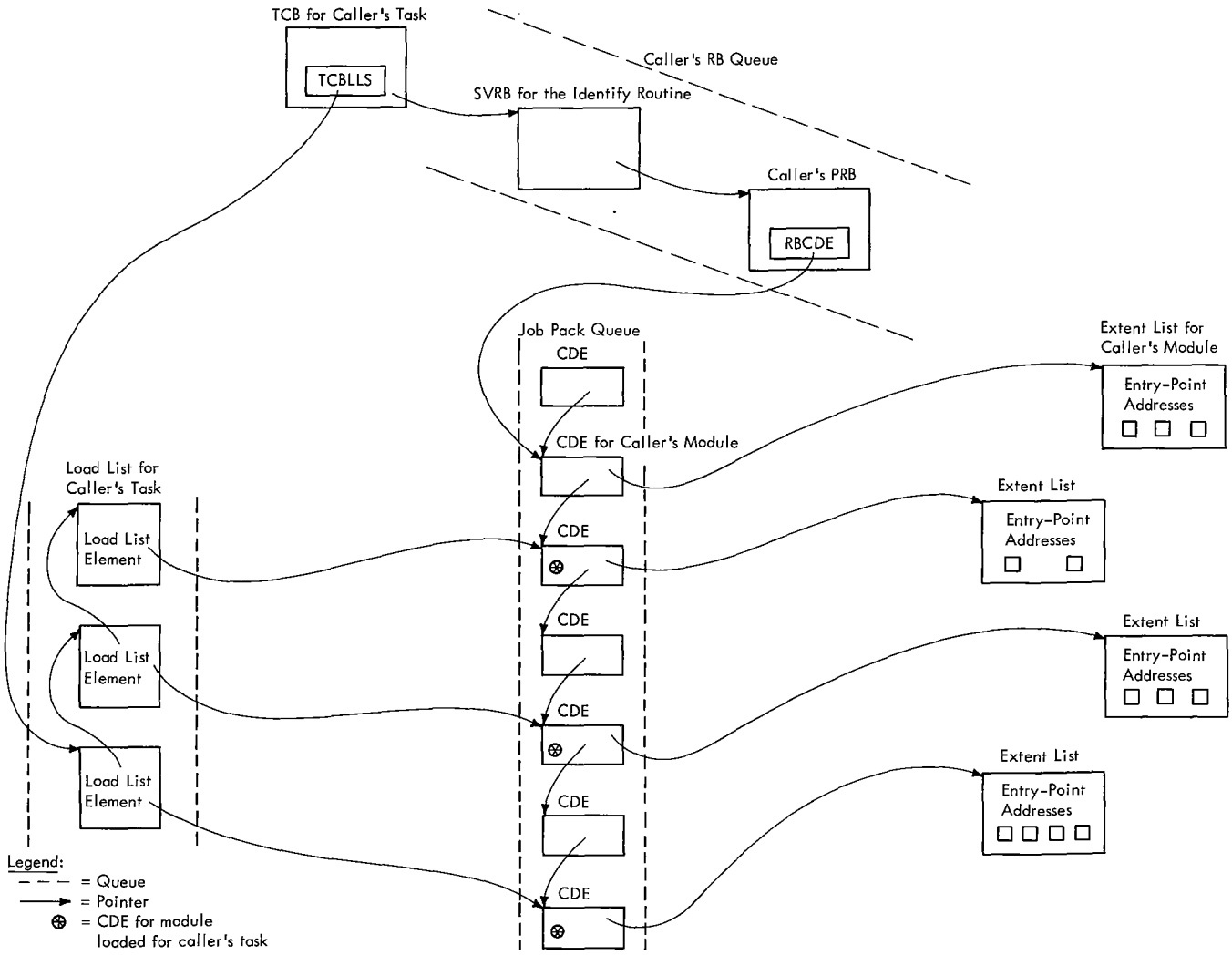


Figure 4-5. Finding an Extent List by Searching the Job Pack Queue or the Load List

tine 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 address is not found, the Identify routine indicates this result by a return code of X'0C'. 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 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 of X'00' 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 OF A MODULE LOADED DIRECTLY INTO MAIN STORAGE

The Identify SVC routine may be used to create a major CDE and extent list for a module brought directly into main storage by the loader. This allows the supervisor to identify the module.

The caller provides the Identify routine with the address and lengths of the module's extents, the entry point, and module name. If the entry point is not already in the link pack or the caller's job pack queue, and the entry point is within one of the specified extents, the Identify routine creates a major CDE and queues it on the user's job pack area control queue. It flags the CDE as "not loadable only", in subpool zero, and "with no backup copy," and builds the extent list. The routine y", then moves the extent list into subpool 255.

Upon entry (at location IGC041) the Identify routine determines whether the caller is a valid user program 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 of X'10', and returns control to the caller via the Exit routine and the Dispatcher. If the test of RB type indicates that the caller is valid, the Identify routine checks the caller's parameter list. If the parameter list is not on a doubleword boundary, or if the first byte of the parameter list is not zero, the Identify routine sets up a return code of X'18' and returns control to the caller via the Exit routine and the Dispatcher. Otherwise, the Identify routine checks the extent list length (EXLLNTH) and extent length addresses. If the extent list length is not positive or not a multiple of eight, or if the extent addresses are not on doubleword boundaries, the Identify routine returns control to the caller via the Exit routine and the Dispatcher with a return code of X'1C'. If the extents are on doubleword boundaries, the routine checks that they are in subpool zero. If they are not, Identify returns control to the caller with a return code of X'20'.

If the extents are in subpool zero, the Identify routine searches the job pack area queue and the link 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 first searches the CDEs of the job pack area queue for a match between the input entry name, supplied in the parameter list for 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, and 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 of X'04' 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 of X'14', and returns control to the caller via the Exit routine and the Dispatcher.

If, during its search of either of the CDE queues, the Identify routine does not find a CDE containing the specified entry-point name, it examines the extent list specified in the parameter list supplied by the calling routine to determine if the disired 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.

If an extent containing the specified entry-point address is not found, the Identify routine indicates this with a return code of X'0C'. It then returns control to the caller, via the Exit routine and the Dispatcher. If the entry-point address is found, the Identify routine creates and initializes a major CDE and extent list.

Identify issues a GETMAIN macro instruction to obtain space for the new CDE and extent list (from subpool 255, supervisor queue area). The routine then initializes the subfields of the CDE (SPZ, XLE, and NLR) to indicate that the CDE represents a major 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 caller's job pack area control queue. Then, setting up a return code of X'00' to indicate successful completion of the 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 similarly decreases 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 turns off bits in the CDATTR field to indicate that the module is neither reentrant nor reusable and then branches 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.

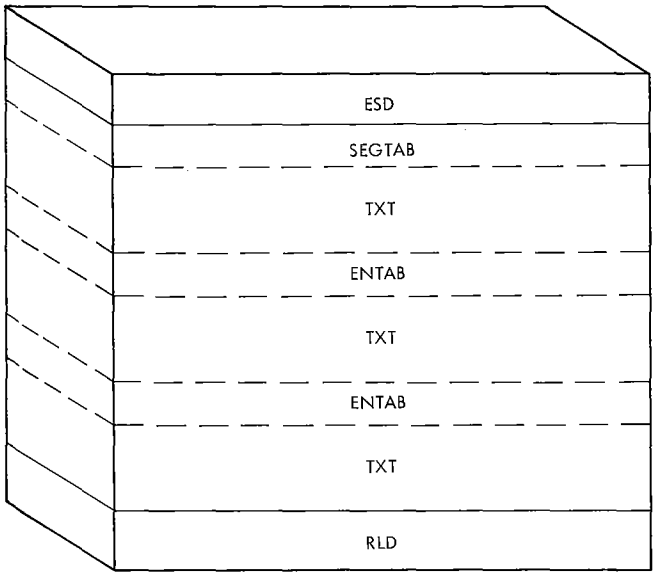


Figure 4-6. Organization of an Overlay Module

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.

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 gives control to an instruction in the associated ENTAB entry. Instructions in the ENTAB 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 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).

Linkage to the Overlay Supervisor

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

DIRECT SUPERVISOR LINKAGE: When the 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."

SUPERVISOR LINKAGE VIA THE CALLER'S ENTRY TABLE: 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."

Types of Processing

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 Figure 4-9.

Determining the Segments that Must Be Loaded

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 4-10 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 Linkage Editor publication for further information.)

During the scan of the segment table, the entry for the requested segment is located and its status indicators are examined. The resultant processing is tabulated in Figure 4-11.

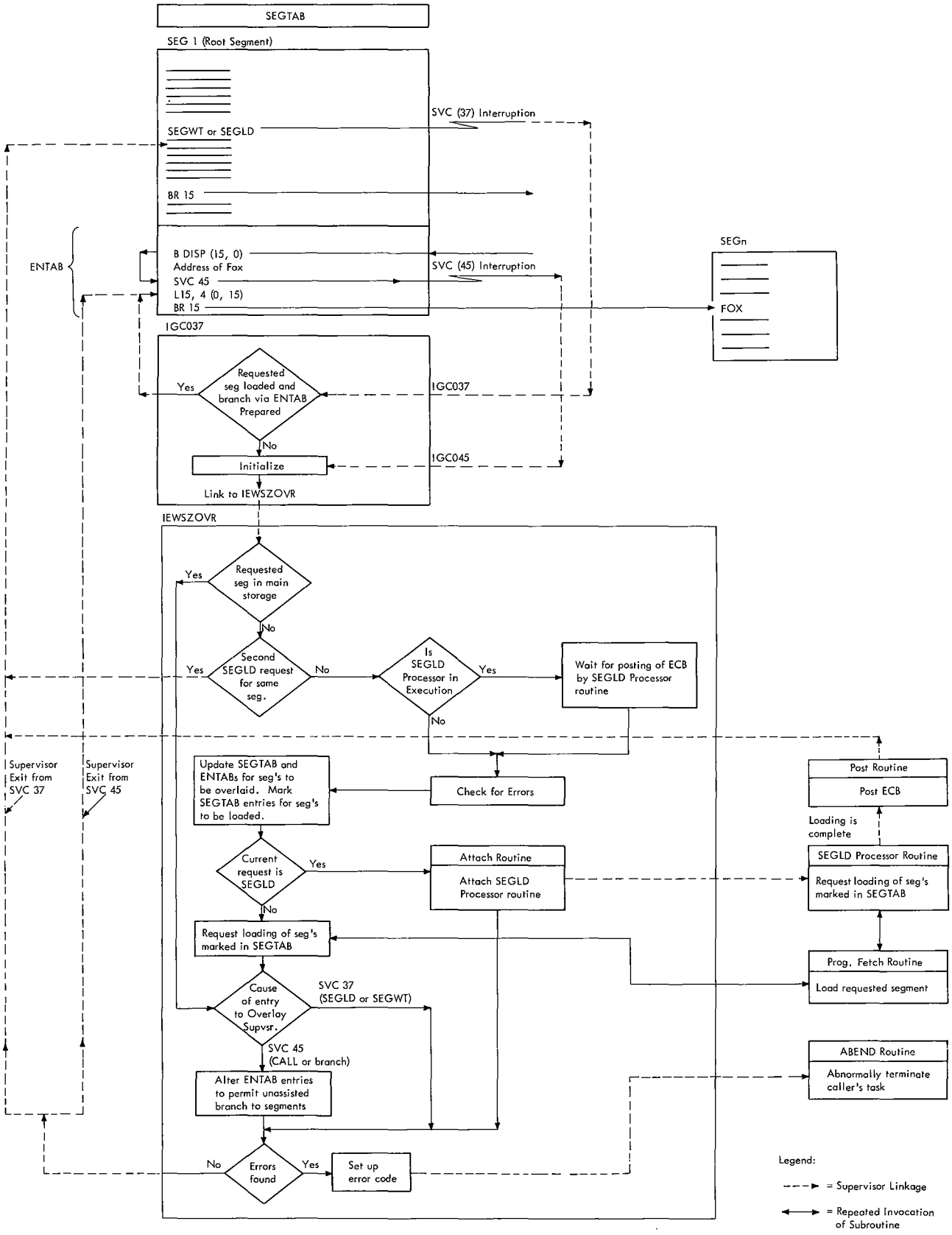


Figure 4-7. Functional Flow of Overlay Supervision

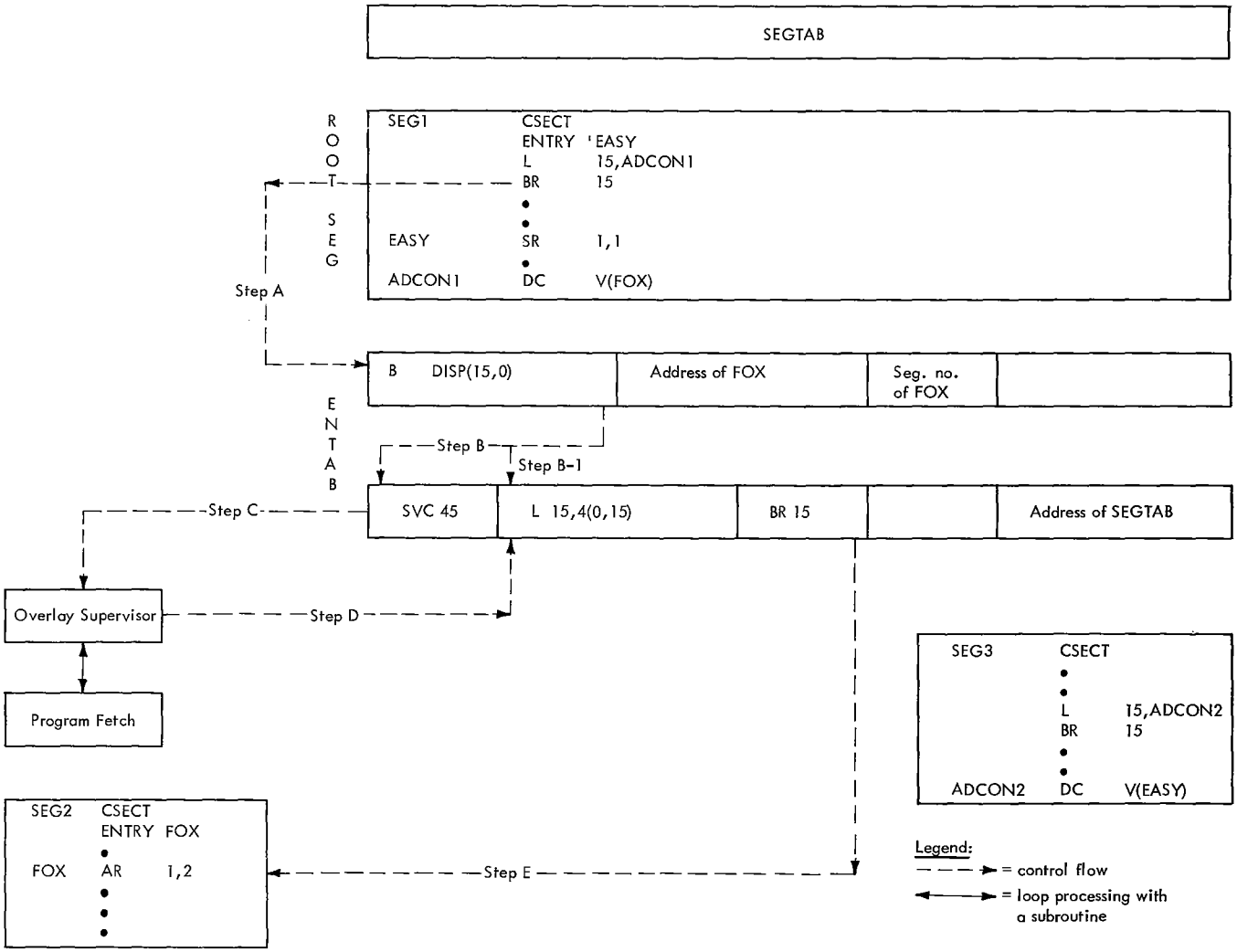


Figure 4-8. Use of the Caller's ENTAB to Branch to a Segment

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

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.

Figure 4-9. Types of Processing During Overlay Supervision

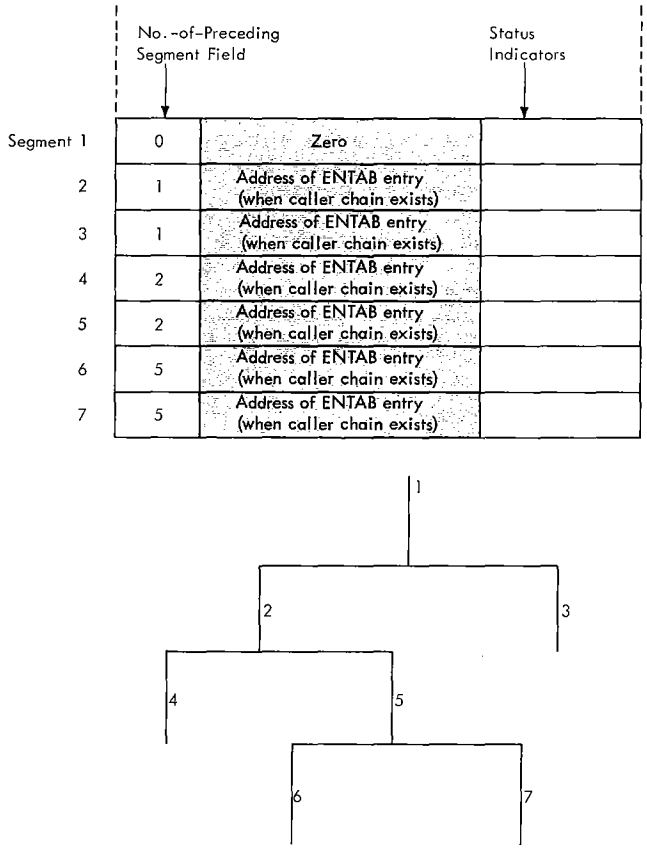


Figure 4-10. Organization of SEGTAB Entries for a Single-Region Overlay Structure

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 Figures 4-7 and 4-9).

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 Figure 4-11.

Passing of Control

The last function of the Overlay Supervisor is to pass control. Control is given to the requested segment or returned to the calling segment, depending on the type of invoking instruction (SEGLD, SEGWT, CALL, or branch). See Figures 4-7 and 4-9.

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

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 to 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 requeted segment are in main storage (except the requested segment itself).
6. Preceding entry is for a segment in the path, and indicates its segment is not in main storage.	Sets the status indicator of the entry whose segment is in the path to "loading scheduled" (01) and continues the scan.

Figure 4-11. Processing of Segment Table Entries

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 4-12).

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 4-13.)

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 Contents Supervision or by the Overlay Supervisor.

It is invoked by the common subroutines of Contents Supervision after a LINK,

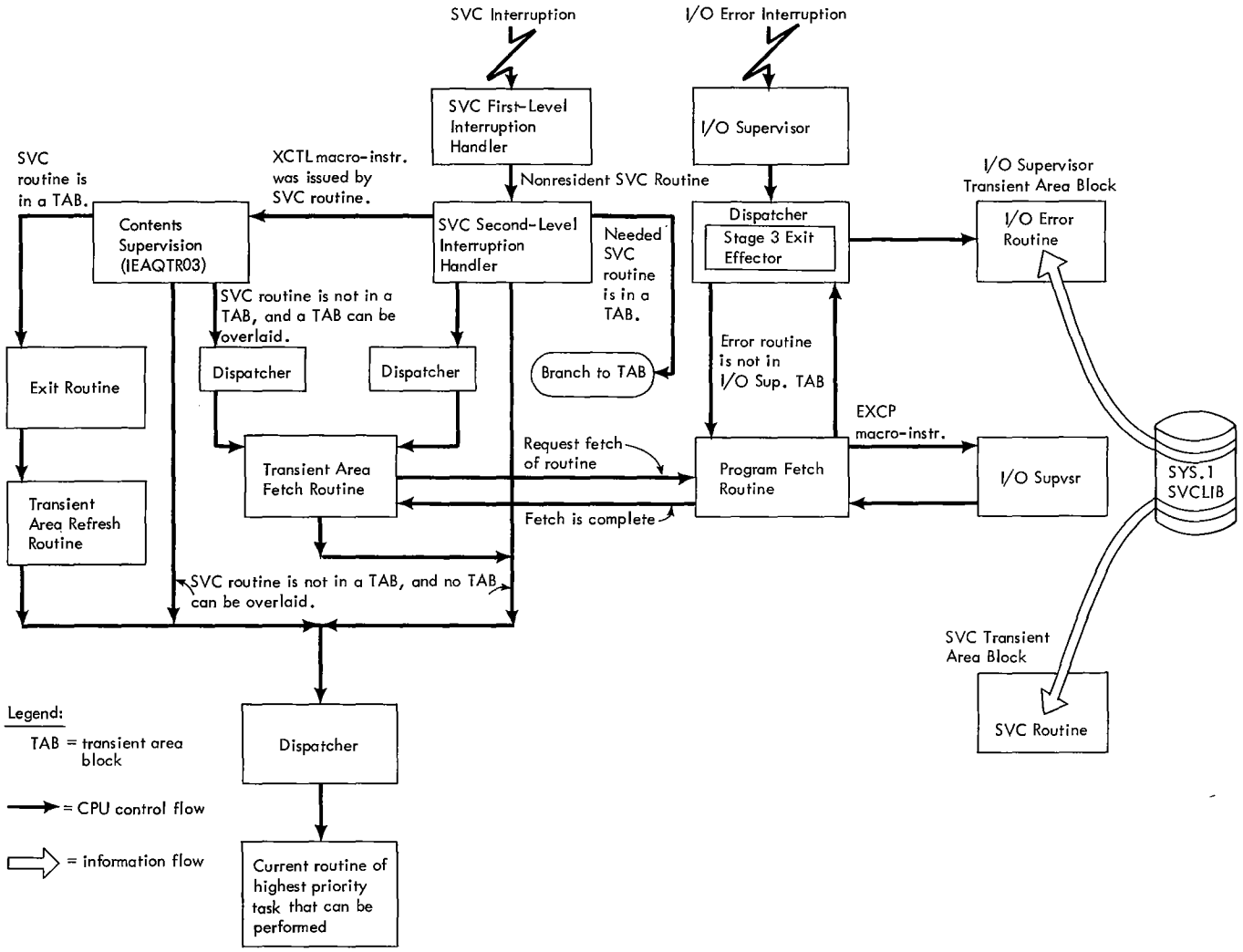


Figure 4-12. Relationships of Program Fetch Routine to Other Routines for the Fetch of an SVC Routine or an I/O Error Routine

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 4-14.

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.

Termination
: checks the completion of I/O operations, calculates the relocated module entry-point address, initializes the segments 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:

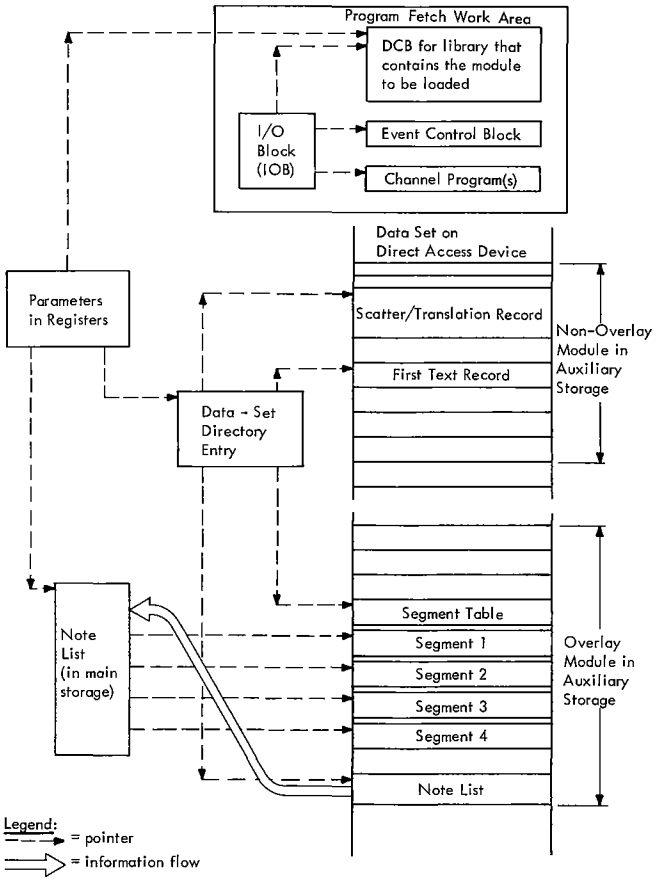


Figure 4-13. Control Blocks and Tables Used by the Program Fetch Routine

- 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.
- 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 4-15). 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 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

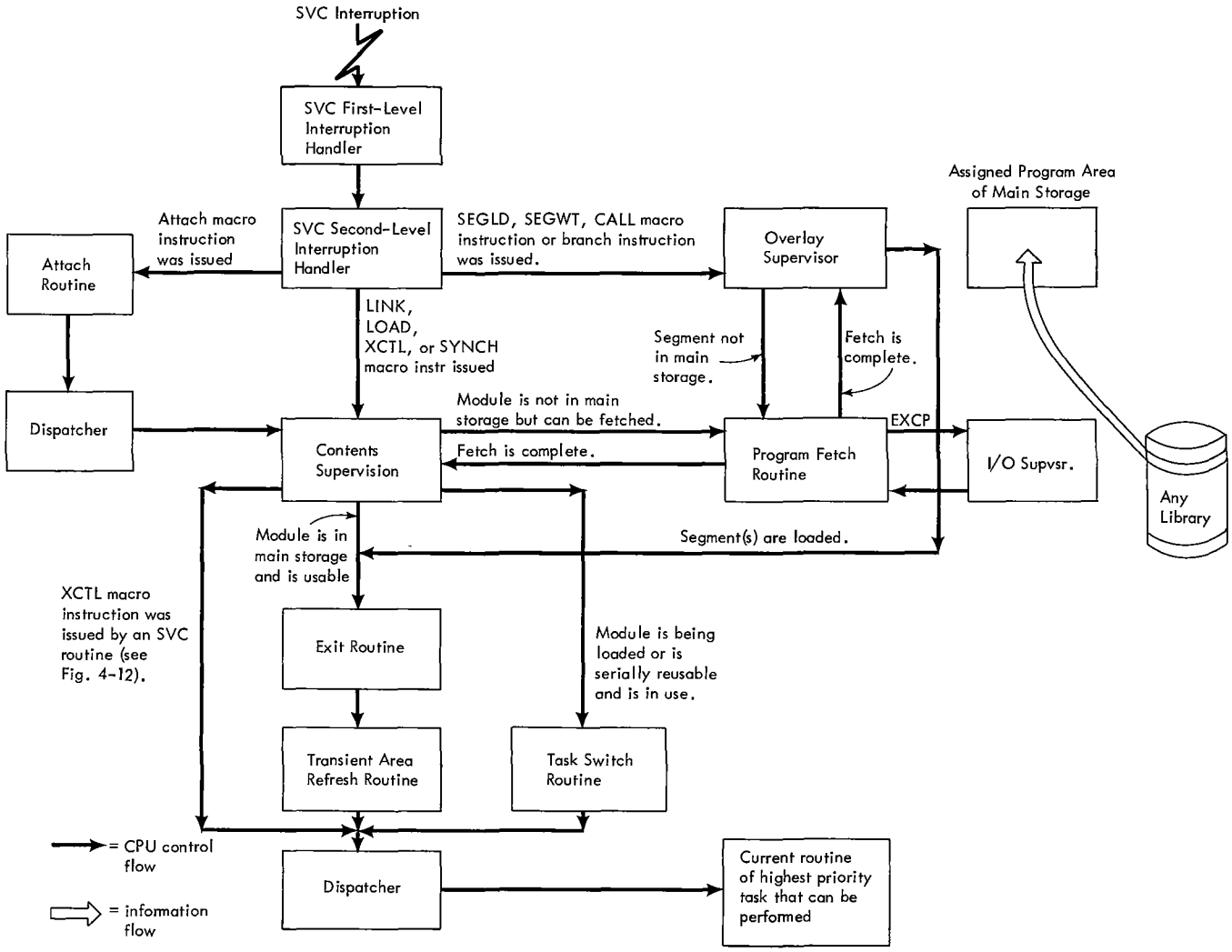


Figure 4-14. Relationship of Program Fetch Routine to Other Routines for the Fetch of a Module or Overlay Segment

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).

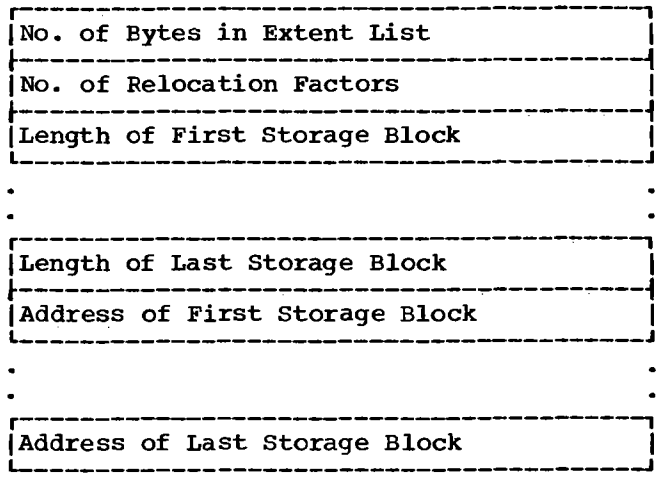


Figure 4-15. Extent List

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.

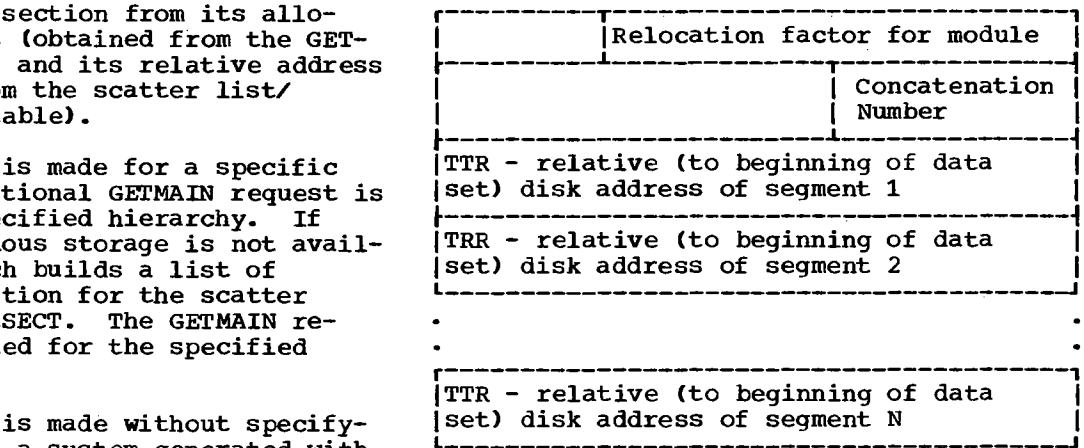
If the request is made without specifying a hierarchy in a system generated with storage hierarchies, initiation for hierarchy loading is performed. The size of the extent list for scatter and the size of the scatter list/translation table record are determined before the GETMAIN request is issued. The 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 stores in the note list the address of the segment table (SEGTAB), and the relocation factor for the module. The note list remains in main storage throughout the module's execution. (See Figure 4-16.)

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 sev-



Note: Concatenation Number - This is a value specifying this data set's sequential position within a group of concatenated data sets.

Figure 4-16. Note List as it Exists in Main Storage

eral types: control records, text records, RLD records, and composite control/RLD records. A typical logical sequence is shown in Figure 4-17. 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, 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 for 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 into 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 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 4-18.)

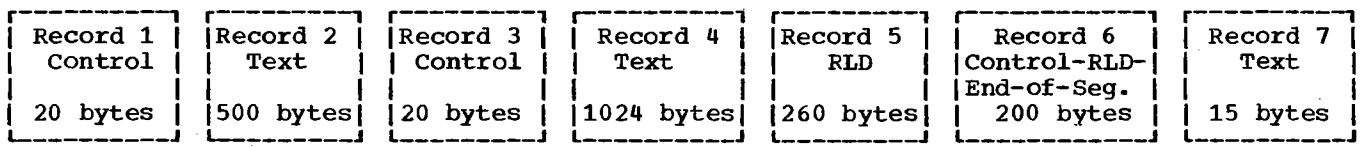


Figure 4-17. Typical Load-Module Logical Format on Direct Access Device

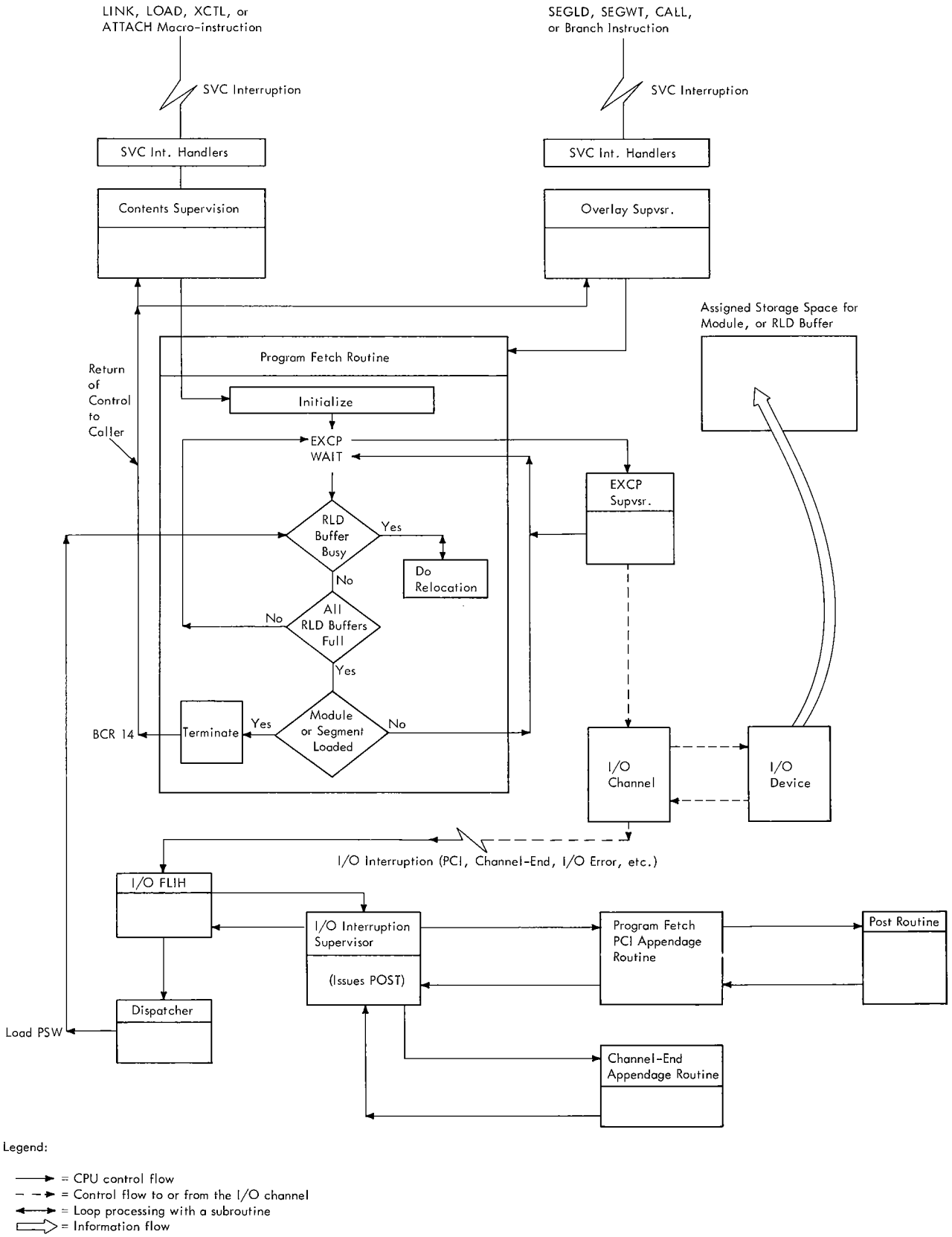


Figure 4-18. Overall Control Flow During the Loading of a Module or Segment

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.

Figure 4-19. Channel-Program Switching After a Program-Controlled Interruption

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 Figure 4-19.

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 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 4-13). 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 4-16).

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.

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 a completion code in the return register. The completion code informs the caller of the result of the attempted loading (see Figure 4-20).

The rest of the termination procedure depends on the type of module that has been loaded (see Figure 4-21). When termination is complete, the Program Fetch routine returns control to the caller.

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

Figure 4-20. Program Fetch Return Codes

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.

Figure 4-21. Termination Processing According to Module Type

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 exercised 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 requester belongs to a job step eligible to cause rollout. The rollout/rollin module is not scheduled if the requester 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 requester'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 requester'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 requester's job step. The rollout/rollin module returns control to the requester, 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 higher dispatching priority than the requester'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 requester'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 requester's job step has completed its use of the borrowed region (signaled 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 priority ready task, via the Exit routine and the Dispatcher.

INTERRUPTION HANDLING FOR MAIN STORAGE SUPERVISION

Both the GETMAIN and FREEMAIN macro instructions 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

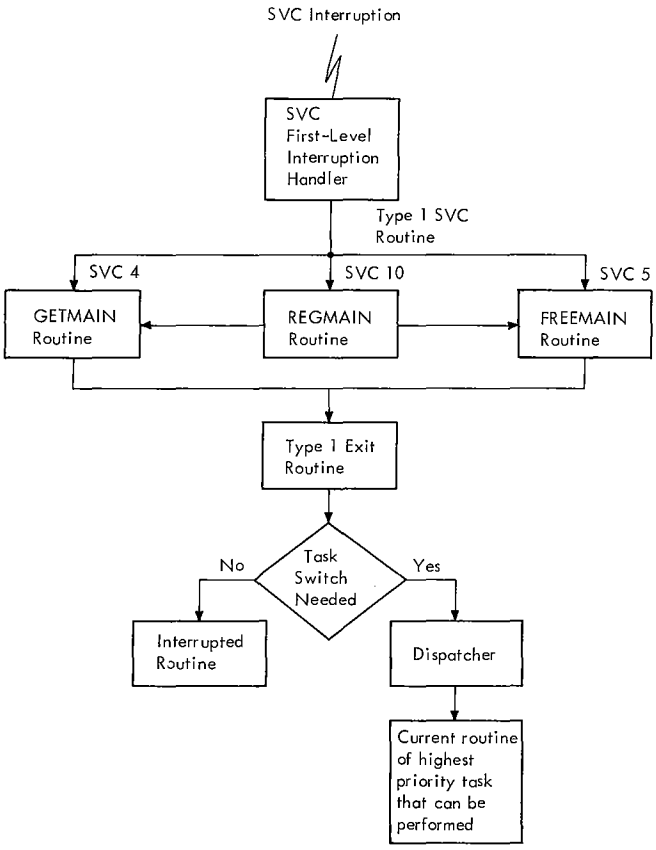


Figure 5-2. Main Storage Supervision Interruption Handling

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 requester. 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, space in the system queue area, or space in the local 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 requester, via the Type-1 Exit routine. If, however, the request is unconditional, the GETMAIN routine makes the requester'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, the GETMAIN routine tries to find space that may be freed and allocated to the requester's task. It first searches for unused modules in the requester's region that may be purged. If sufficient space cannot be made available by the module purge, and the request is conditional, the GETMAIN routine sets up a return code and returns control to the requester, via the Type-1 Exit routine. If, however, the request is unconditional, and if the rollout feature cannot be used, the GETMAIN routine causes the abnormal termination of the requester's task. If, however, the rollout feature is part of the system and the requester'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 requester'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 requester's job step or another job step, or makes the requester'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 the requested space in the system queue area is not available, the GETMAIN routine purges and frees CDEs within the system queue area. If this does not make sufficient system queue area available, the GETMAIN routine attempts to expand the system queue area by assigning to it 2K blocks of adjacent free dynamic area storage. If the system queue area cannot be expanded (because the adjacent dynamic area is assigned to a region), or if the system queue area is expanded but still cannot satisfy the request, the GETMAIN routine determines if 144 bytes of system queue area are available. If not, the GETMAIN routine places a completion code of E04 in the TCB of the requester, sets the TCB nondispatchable, and causes the CPU to be placed in an enabled wait state with a wait code of E04. If 144 bytes of system queue area are available, the GETMAIN routine uses ABTERM to schedule the requester for abnormal termination with a completion code of E04. The 144 bytes are necessary to build the SVRB for ABEND which terminate the requester. In a Model 65 Multiprocessing System, if the system queue area is expanded, the new size and origin of the dynamic area is placed in the PQE.

If requested space in LSQA is not available, the GETMAIN routine causes abnormal termination of the requester's task unless the task is already in abnormal termination processing. In this case, the request is changed to a request for space in SQA.

Following entry to the GETMAIN routine, the Subpool Check (CSPCHK) subroutine is entered to determine what type of space is requested. Figure 5-3 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 5-4). The PQEPTR field at offset 8 in location GOVRFLB contains the address of a two-word dummy partition queue element (DPQE) less 8 bytes. Word one of the DPQE contains the address of a partition queue element (PQE) that describes unassigned processor storage not assigned to any region. Word two of the DPQE contains the address of the last PQE constructed by NIP.

In a system with Main Storage Hierarchy Support, word three of the PQE for hierarchy 0 contains the address of the PQE that describes unassigned IBM 2361 Core Storage not belonging to any region. If IBM 2361 Core Storage was off-line at IPL, this word contains zeros and any requests for hierarchy 1 storage are logically satisfied from processor storage. If Main Storage Hierarchy Support is not included in the system, or if Main Storage Hierarchy Support is included but IBM 2361 Core Storage is not on-line at IPL, only one PQE, describing allocatable storage, is constructed.

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 subpools 0-127 are requested by programs executing in supervisor mode and key 0, 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 mode or key 0, subpool 0 is assigned.
251	Space within region	Job-step's storage protection key (reset to 0 when space is freed)	Nonreenterable modules in the Job Pack Area.
252	Space within region	0 storage protection key	Reenterable modules in the Job Pack Area.
243	Space within system queue area	0 storage protection key	Assigned space is freed when task terminates.
244	Space within system queue area	0 storage protection key	Assigned space is freed when job step terminates.
245	Space within system queue area	0 storage protection key	Assigned space must be explicitly freed.
253	Non-TSO task -- space within system queue area TSO task -- space within local system queue area	0 storage protection Key 0 storage protection Key	Assigned space is freed when task terminates. Assigned space is freed when task terminates.
254	Non-TSO task -- space within system queue area TSO task -- space within local system queue area	0 storage protection Key 0 storage protection Key	Assigned space is freed when job step terminates. Assigned space is freed when job step terminates.
255	Non-TSO task -- space within system queue area TSO task -- space within local system queue area	0 storage protection Key 0 storage protection Key	Assigned space must be explicitly freed. Assigned space must be explicitly freed.

Figure 5-3. Subpool Numbers Used for Requesting Space

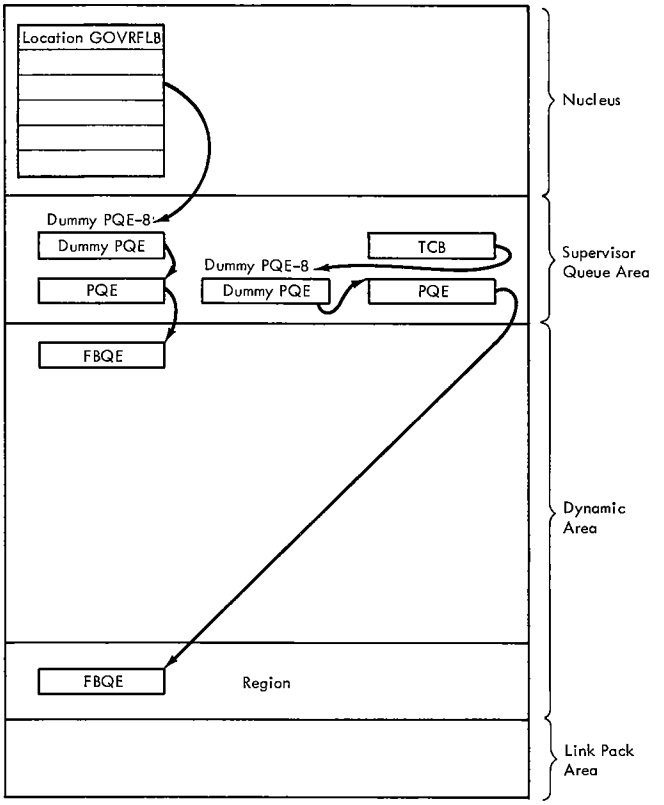


Figure 5-4. Element Relationships: Region Allocation

Words one and two of the PQE point to the first and last, respectively, free block queue elements (FBQEs) associated with the area of storage described by the PQE. FBQEs occupy the first three words of each free block of each type of storage and contain a count of the number of free bytes available in that block of storage. FBQEs are forward and backward chained in address order so that main storage supervision routines may scan them from either high to low address or low to high address.

To assign a region, the GETMAIN routine searches for the highest block of free storage in the dynamic area that is large enough to satisfy the request. It then determines the beginning address of the region:

Beginning Address = the size of Free Block of storage + the address of the FBQE (for that block of free storage) - the 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 block of free storage.

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 5-4). 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 less 8 bytes 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 may be requested with segments in both hierarchies. A PQE is constructed for each region segment and both PQEs are chained (by way of a dummy PQE) 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 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.

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. 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. Figure 5-5 shows the subpool use for list and register forms of GETMAIN requests for region allocation; Figure 5-6 shows the lists and pointers.

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 segment in same hierarchy (if H0 or H1 only) or in hierarchy 0 (if region has segments in both hierarchies)
247	Address = 0, get region anywhere Address ≠ 0, get region specified address	Register 1 negative, get region Register 1 zero or positive, free region
248	Request from Rollout/Rollin	Request from Rollout/Rollin

Figure 5-5. Subpool Use for List and Register Forms of GETMAIN (GETPART Module)

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 Model 65 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 Storage Reconfiguration or Multiprocessing NIP routines). 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 requester 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 FREEPART/GETPART (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 segment in hierarchy 0 is freed if

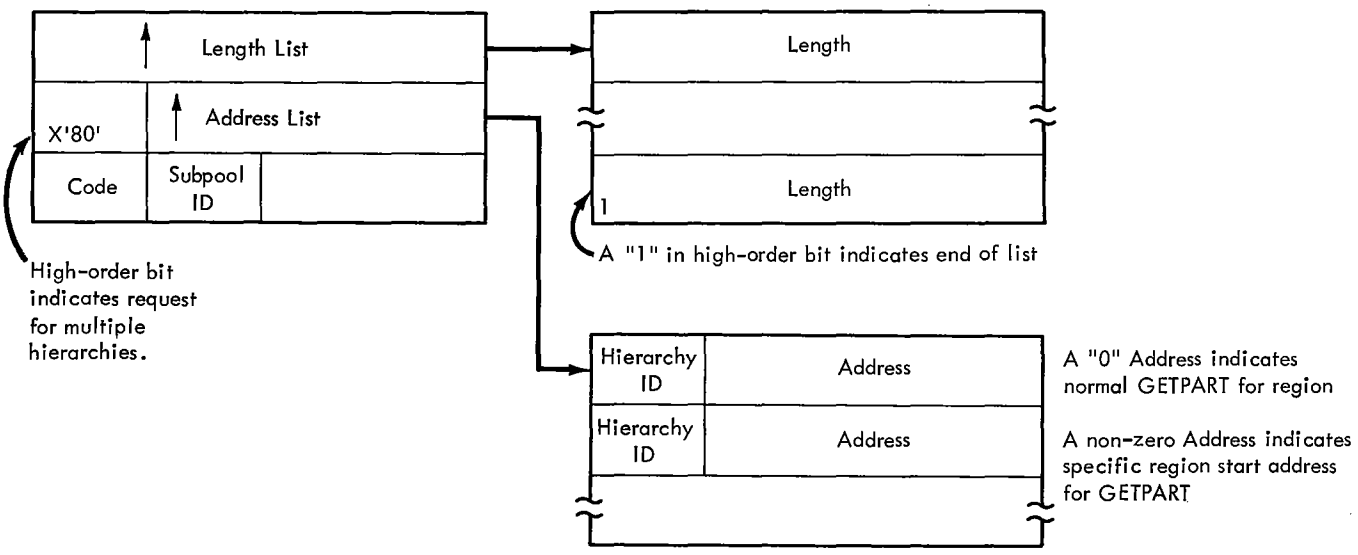


Figure 5-6. List Structure for List Form of GETMAIN (GETPART Module)

a region consists of both hierarchy 0 and hierarchy 1 segments (see Figure 5-5). The region segment described by the FREEPART/GETPART request must lie wholly within the dynamic area. All FREEPART/GETPART requests for specific addresses are assumed to be within the boundaries of the original region; no provision is made to handle an invalid request.

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 requester that space cannot be allocated. It then returns control to the requester, via the Type-1 Exit routine. If, however, the request is unconditional, the GETMAIN routine makes the requester'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 requester's TCB.
- Pointing the SVC old PSW to the invoking GETMAIN macro instruction, and storing this restart address in the requester'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. If Main Storage Hierarchy Support is included in the system, a region may consist of segments in both hierarchies, or may be contained entirely within either hierarchy 0 or hierarchy 1. If only a single-hierarchy region exists, all GETMAIN requests for tasks operating in that region will be directed to that hierarchy regardless of any hierarchy designation in the request. If a region consists of segments in both hierarchies, a GETMAIN request may specify the hierarchy from which storage is to be obtained. If hierarchy is not specified, allocation is made from hierarchy 0.

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 supervisor queue area (see Figure 5-7). 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 to subpools in multiples of 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. For each subsequent request for space in the same subpool that cannot be satisfied from space described by existing DQEs, additional space is allocated (in multiples of 2048-

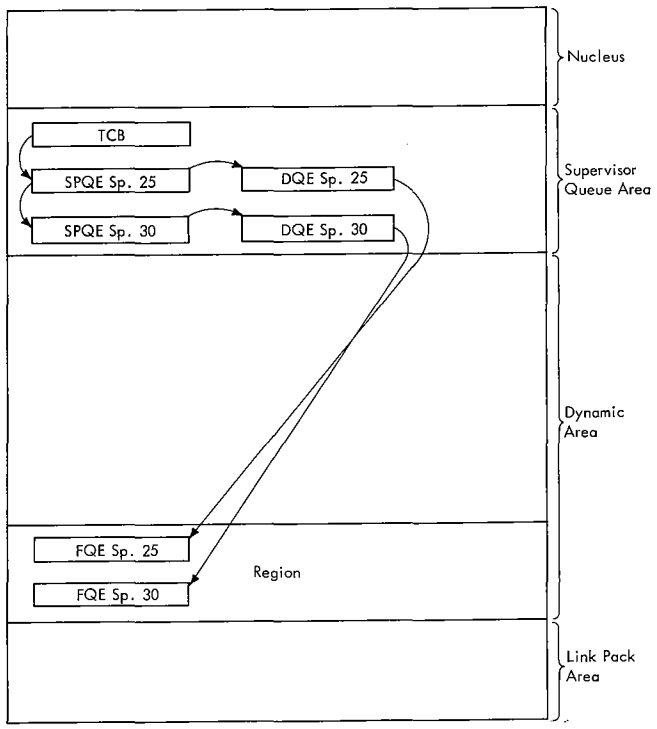


Figure 5-7. Element Relationships for Intra-Region Allocation

byte blocks) to the subpool. The space is obtained from the free area of storage described by FBQEs, and a DQE is constructed to describe the new block. The number of bytes allocated is subtracted from the FBQE for the area from which storage was obtained, and the FBQE is relocated if necessary. 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 Figure 5-3). 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 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.

If the System Management Facility (SMF) feature is present in the system, the GETMAIN routine passes control to its SMF Storage subroutine (GMSMFCRE). This routine maintains storage usage information in the timing control table (TCT). (If Main Storage Hierarchy Support is included in the system and IBM 2361 Core Storage is on-line at IPL, storage information is maintained for both processor storage and 2361 storage.)

The SMF Storage routine checks the TCB for the address of the TCT. If there is no TCT, SMF storage information is not being recorded for this user program. If there is a TCT, the SMF Storage routine performs the following functions for subpools 0-127 and 250-252:

- It determines whether the newly allocated storage exceeds either the "low water mark" (LWM) or the "high water mark" (HWM) for the region. The LWM is the address of the highest storage address allocated from the bottom of the region, and the HWM is the address of the lowest storage address allocated from the top of the region. If either is exceeded, the SMF Storage routine stores a new value in the TCT.

- It calculates the difference, in terms of 2048-byte blocks, between the LWM and HWM. A record of the minimum value for this difference is kept in the TCT. If the new allocation creates a new minimum, the SMF Storage routine records the new minimum difference in the TCT.

- If the rollout feature is included in the system and the current allocation is for borrowed storage, the SMF Storage routine records the current amount of borrowed storage. It also records the maximum amount of borrowed storage associated with this user at any one time.

The SMF Storage routine returns control to the GETMAIN routine.

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 requester'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 requester'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 requester's task. It then returns control to the requester, 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 possibility of satisfying the storage request from space outside the requester's region. The requester'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 requester, 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 requester's task. It sets up a condition code (hex. '804')[1] 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 problem program or by a system routine in behalf of a problem program.
- The requester'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')[1] to indicate that storage is unavailable, and branches to the ABTERM routine to schedule the abnormal termination of the requester'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 requester'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.)

---

[1]Condition Code 804 is set up for SVC 4. Condition Code 80A is set up for SVC 10.

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 5-8.

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. When linkage to the RO/RI module is needed, the scheduling process positions the IRB on the RO/RI task's RB queue. (See part 2 of Figure 5-9.) Since the RO/RI IRB is usually in a ready condition (its wait count equal to zero), it makes the RO/RI task dispatchable.

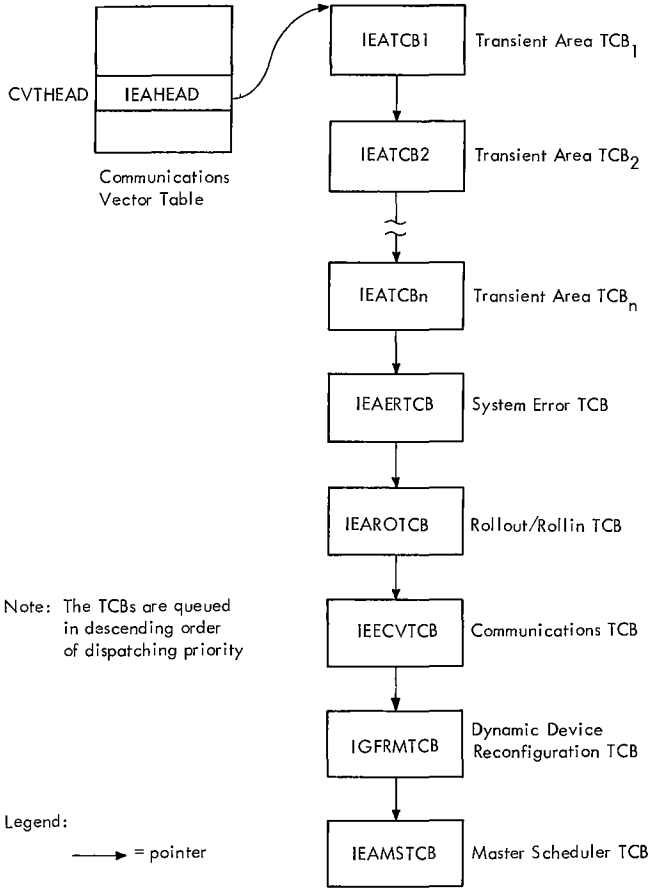


Figure 5-8. Position of Rollout/Rollin TCB on TCB Queue

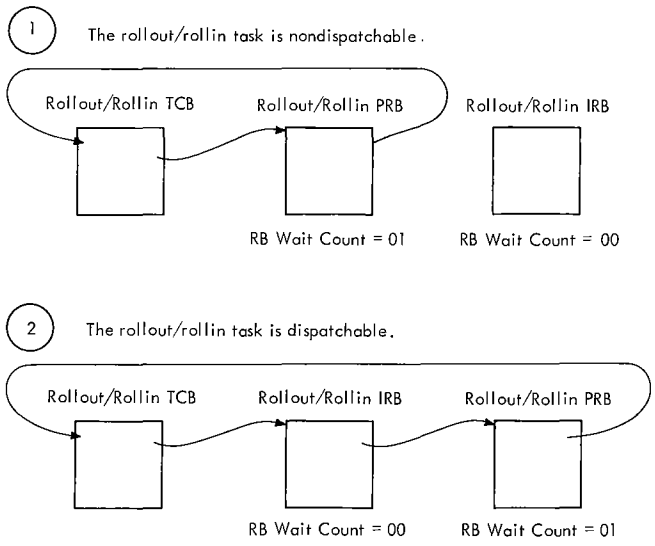


Figure 5-9. Relationship of the Rollout/Rollin TCB, PRB, and IRB During Scheduling of the Rollout/Rollin Task

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 5-10 for the overall flow and Figure 5-11 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 requester.

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

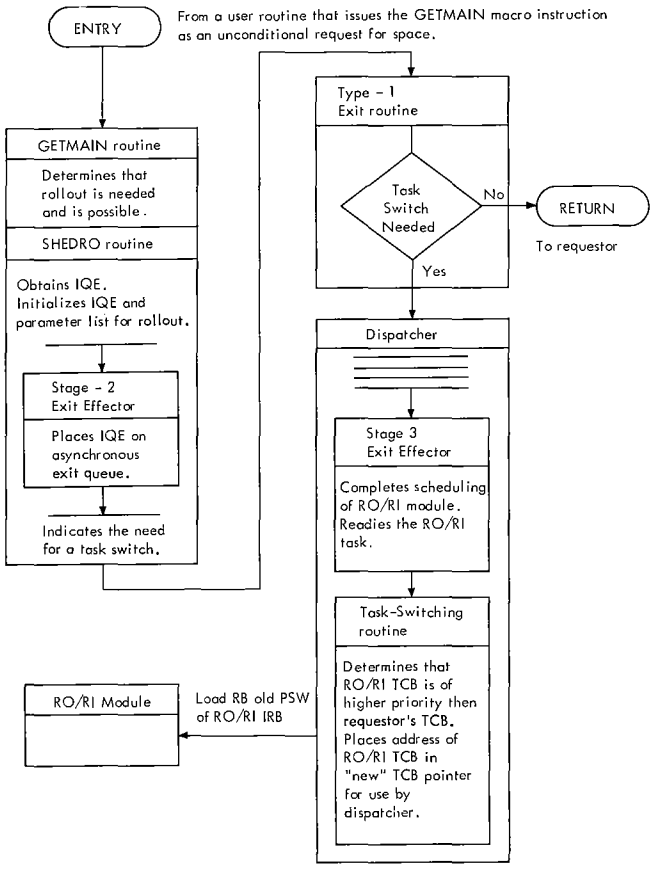


Figure 5-10. Scheduling of Rollout: Overall Flow

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 requester's TCB and the size of the requested space.

Placing the IQE on the Asynchronous Exit Queue: 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 5-11.) 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, when the Dispatcher is next entered, completes 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 are serviced by the Stage 3 Exit Effector on a task-priority basis.

Preparing for a Task Switch and for Eventual Return of Control to the Requester: The SHEDRO routine does three things to prepare for a task switch and to provide for eventual return of control to the requester:

- Indicates to the Type-1 Exit routine that a task switch is needed.
- Makes the requester's task nondispatchable (sets the TCBWFC flag).
- Points the SVC old PSW to a restart address in the requester'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 branches to the Dispatcher, which then determine the task to which it will give control.

The SHEDRO routine makes the requester's task nondispatchable to prevent accidental redispatching of the requester'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 requester. (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 requester can be redispatched to reissue its GETMAIN macro instruction. The GETMAIN routine is then 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

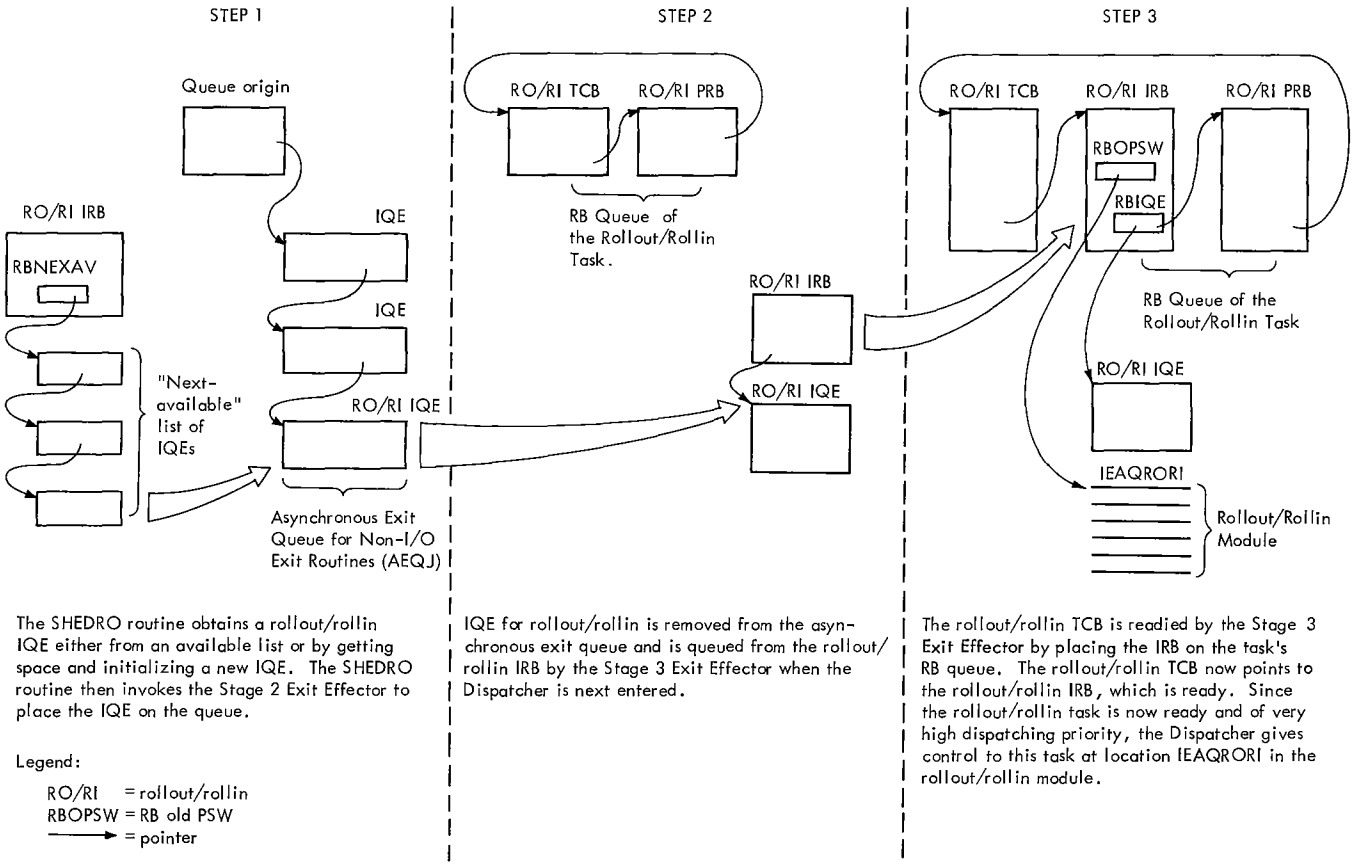


Figure 5-11. Steps in the Scheduling of the Rollout/Rollin Task

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 5-11.)
- 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 remains queued from the IRB throughout rollout processing. When the RO/RI module completes its processing of the rollout request, it dequeues the IQE from the IRB's active queue and returns it to the IRB's "next available" list (RBNEXAV).

The Stage 3 Exit Effector readies the RO/RI task by placing the IRB on the RO/RI task's RB queue, as illustrated in Figures 5-9 and 5-11. 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 changed.

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 requester'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 requester'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 whether 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 requester's job step.
- Processing if there was an unrecoverable I/O error during the rollout.
- Preparing for exit from the rollout/rollin module.

Determining whether Rollout Should Be Performed

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.

DETERMINING WHETHER THE CURRENT REQUEST IS FOR ROLLOUT: 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.)

DETERMINING WHETHER ANOTHER JOB STEP HAS CAUSED A ROLLOUT THAT IS STILL IN EFFECT: 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 requester'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.

DEFERRING THE CURRENT ROLLOUT REQUEST: 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 requester'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 available 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 248. 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 requester'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 is 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 requester'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 requester's task dispatchable by clearing the "core wait" nondispatchability flag (TCBWFC). This is done in preparation for the redispatching of the requester'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 requester'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 requester's job step has previously caused a rollout that is still in effect. (The routine tests the TCBFRI flag in the requester'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 requester'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 requester'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 requester'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 requester'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 requester'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 requester'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 requester'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 requester'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 requester'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 requester's job-step task is to be terminated, the RO/RI Criterion routine branches to the ABTERM routine, providing the address of the requester'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 requester'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 requester'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 requester'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 is allocated to the requester's job step. The region's contents can be in either of two conditions: already rolled out for a requester 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 requester'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 requester'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 requester'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 requester'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

---

[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.

the PQE queue of the requester's job step, set the "rollout invoked" flag (TCBFRI) in the requester's job step TCB, and clear the "core wait" nondispatchability flag (TCBWFC) in the requester's TCB. (See "Obtaining the Needed Space from Unassigned Storage" for a discussion of these actions.)

- Branches to the RETEXIT routine to prepare for exit from the RO/RI module and return control to the requester'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 are 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 5-12 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.

Setting Nondispatchable the Tasks of the Job Step: 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.

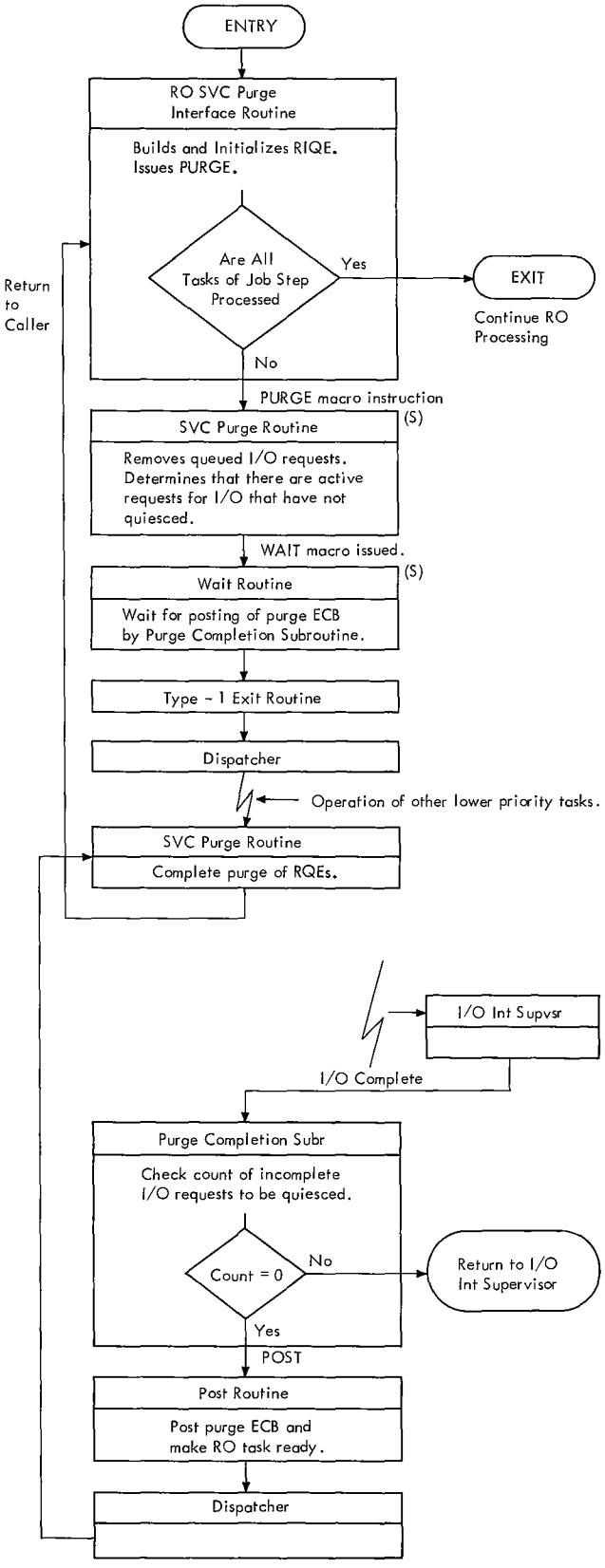


Figure 5-12. Interfaces Between Rollout Module and SVC Purge Routine

The operands of the STATUS macro instruction, as used above, have these meanings:

STATUS	SET ND,	(1)	(12)
	Causes setting of nondis-patchability 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.

Deferring the Job-Step's I/O Requests: 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 serves as a list origin for a queue of I/O blocks (IOBs) that represent the task's deferred channel programs. The IOBs are 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 are to 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 uses 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 (X'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 I/O Supervisor PLM for detailed information on the SVC Purge routine.)

  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 5-13.)

  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.

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) determines that the rollout flag is set in the reply queue element, and saves the reply in a temporary buffer until the job step is rolled in. (See "Reply Processing" in Section 7, "Console Communications and System Log.")

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

---

[1]See Section 12, "Control Blocks and Tables."

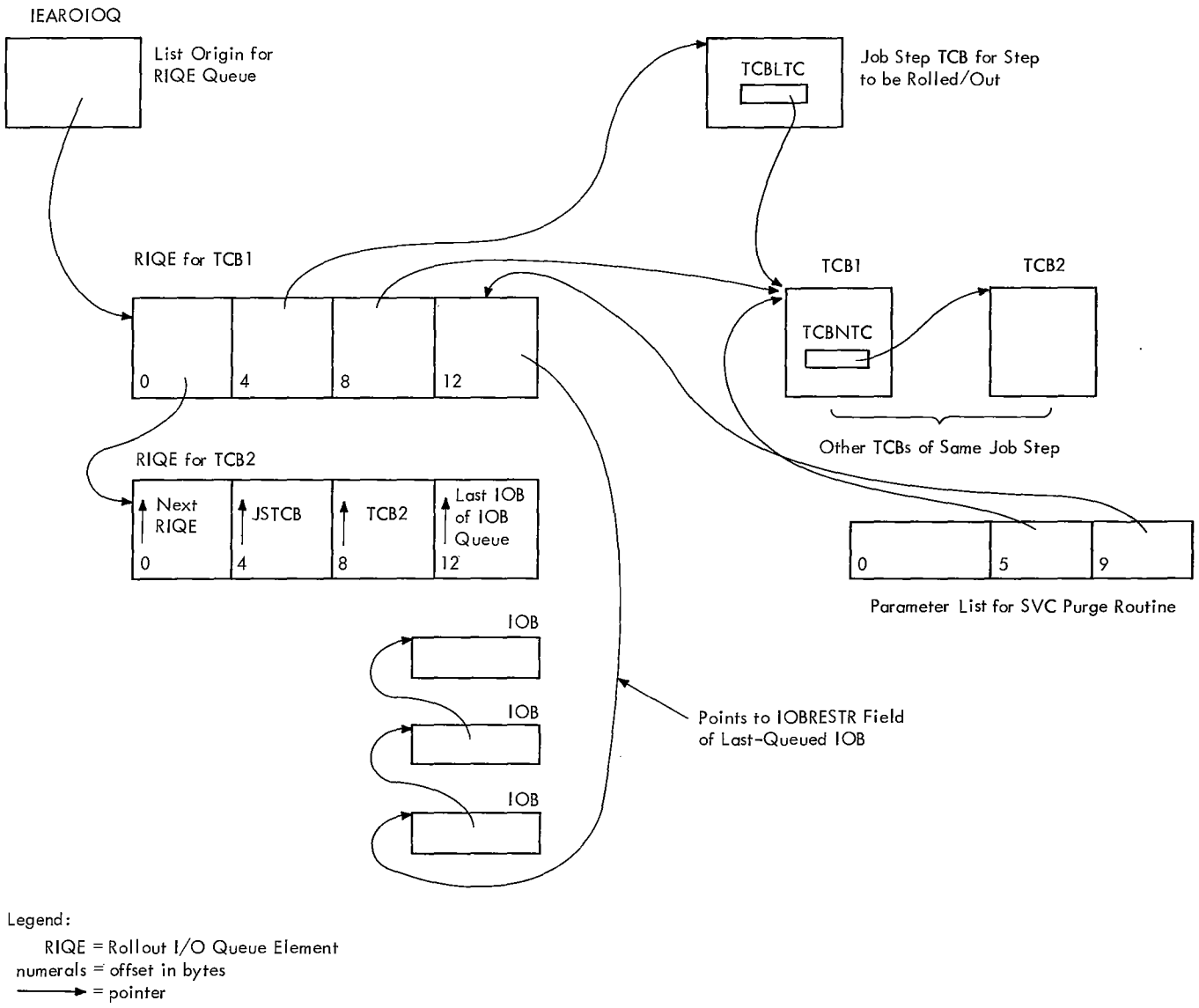


Figure 5-13. How IOBs for Deferred I/O Requests are Queued

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 are 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 uses 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:

  $TTR = ((R - K_1)/R_1)/N$

  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_1 = 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 super-

visor 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 identifies 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 requester'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.

---

[1]For this type of error the "IOB intercept" code appears in the completion code field of the ECB.

Allocating the Borrowed Region to the Requestor's Job Step

If the Start Transfer routine (STARTIO) determines that there was no permanent 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 "IEA405I 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 requester's task.
- Reallocates to the requester'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 AND LOCAL 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. SQA can be requested by any task using subpools 243, 244 and 245 and by non-time sharing tasks using subpools 253, 254 and 255.

The local system queue area is also restricted to control program routines and serves the same purpose as SQA except that it is swapped with a time sharing user job. Only time sharing tasks can obtain LSQA space. Requests by time sharing tasks for subpools 253, 254, and 255 are allocated from LSQA by the GETMAIN routine. Time sharing tasks must request subpools 243, 244 or 245 to obtain space in SQA.

In addition to determining which area (SQA or LSQA) the space is allocated from, the subpool numbers also indicate which of the following characteristics should apply to the space:

- Space within subpools 243 and 253, unless explicitly freed, is automatically released when the task for which it is being used is terminated.

- Space within subpools 244 and 254, unless explicitly freed, is released automatically when the job step for which it is being used is completed.

- Space within subpools 245 and 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 5-14). 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 (pointed to by the secondary CVT) contains a pointer to the descriptor queue element; the descriptor queue element contains a pointer to the free queue element.

The local system queue area is obtained and initialized when a time sharing region is started. The size of LSQA for each region is specified in the time sharing member of SYS1.PARMLIB. LSQA is contained within each time sharing region and is defined by the PQE for the region. An SPQE, DQE and FQE are built in the first 32 bytes of LSQA and are initialized to define the available space in LSQA.

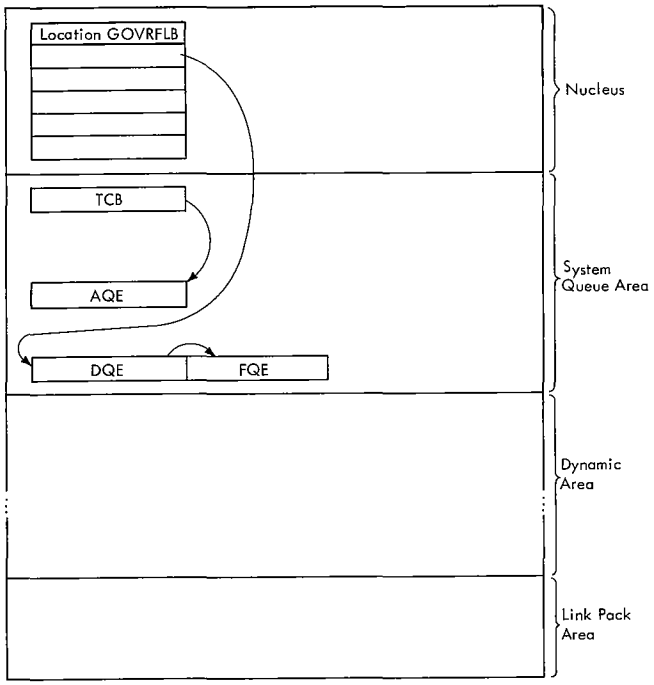


Figure 5-14. Element Relationships for System Queue Area Allocation

Subpool 243 or 253 Allocation

When subpool 243 or 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 scan the AQE queue and give a FREEMAIN macro instruction to free all space associated with subpool 243 or 253.

Subpool 244 or 254 Allocation

When subpool 244 or 254 is specified in a GETMAIN macro instruction, the GETMAIN routine builds an allocated queue element as it does for subpool 243 or 253, but chains the AQE to an AQE queue whose origin is in the job step TCB. When that job step is completed, supervisor termination routines scan the AQE queue and give a FREEMAIN macro instruction to free all space associated with subpool 244 or 254.

Subpool 245 or 255 Allocation

When subpool 245 or 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 requester 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, space in the system queue area, or space in the local 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 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 Model 65 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 a VARY STORAGE OFFLINE command has been issued. 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 region within the range of that VQE has been processed.

If the region being freed is not owned by the requester, 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. The CSPCHK subroutine then determines whether any task(s) has been set non-dispatchable because of the limit on queue space. If any task(s) has been, it branches to the GETPART routine which attempts to reset the task(s) dispatchable. If not, 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.

If the System Management Facility (SMF) feature is present in the system, the FREEMAIN routine passes control to its SMF Storage subroutine (FMSMFCRE). This routine maintains storage usage information in the timing control table (TCT). (If IBM 2361 Core Storage is included in the system, storage information is maintained both for processor storage and for 2361 Core Storage.)

The SMF Storage routine checks the TCB for the address of the TCT. If there is no TCT, SMF Storage information is not being recorded for this user program. If there is a TCT, the SMF Storage routine performs the following functions:

- It determines whether the newly released storage causes a change in the "low water mark" (LWM) or the "high water mark" (HWM) for the region. The LWM is the address of the highest storage address allocated from the bottom of the region, and the HWM is the address of the lowest storage address allocated from the top of the region. If either is changed, the SMF Storage routine stores the new value in the TCT.

- If the rollout feature is included in the system and the storage being released is borrowed storage, the SMF Storage routine subtracts the amount of released storage from the record in the TCT of the current amount of borrowed storage.

The SMF Storage routine returns control to the FREEMAIN routine.

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

---

[1]A rolled out step normally has only one PQE and region, since rollout of a step that has itself caused rollout is forbidden.

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 2K blocks from the borrower's key to that of the owner, except subpools which are given their protection key. This is done via a branch to location SETKEYS1.

- 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 "IEA406I 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 2K 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 (ABEND16) 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.")

- Calls the TCAM Post Pending routine (IGG019RQ) when there is an OS Post pending for a task that is currently being rolled in.

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 is 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 loads 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 indicates 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 AND LOCAL SYSTEM QUEUE AREA

To free space in the system queue area or in the local system queue area, the FREEMAIN routine determines whether space within subpools 243 or 253 and 244 or 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 243, 244, 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.

When address correction has been completed or if the FREEMAIN request is for subpools 245 or 255, a check is made to ensure that the specified area falls within the bounds of SQA or LSQA as applicable.

The DQE for the area is then obtained and an FQE is built for the area being freed and is placed on the DQE FQE chain. Any resulting contiguous free areas are combined by combining FQEs.

For non-time sharing tasks, subpools 243, 244, 245, 253, 254, and 255 all free SQA space. For time sharing tasks, subpools 243, 244 and 245 free SQA space and subpools 253, 254, and 255 free LSQA space (or SQA space if LSQA has overflowed into SQA).

SYSTEM/360 TIMER SUPERVISION

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 Model 65 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, or alternate, timer is always set to a value X'80000000' greater than the active timer and thus never expires. If the CPU is in partioned mode, or is the only online CPU in a Model 65 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 zeros 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.

SYSTEM/370 TIMER SUPERVISION

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

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

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

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

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

TIMER SVC INTERRUPTION HANDLING

The handling of interruptions resulting from issuance of timer-related macro instructions, is shown in Figure 6-1.

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.)

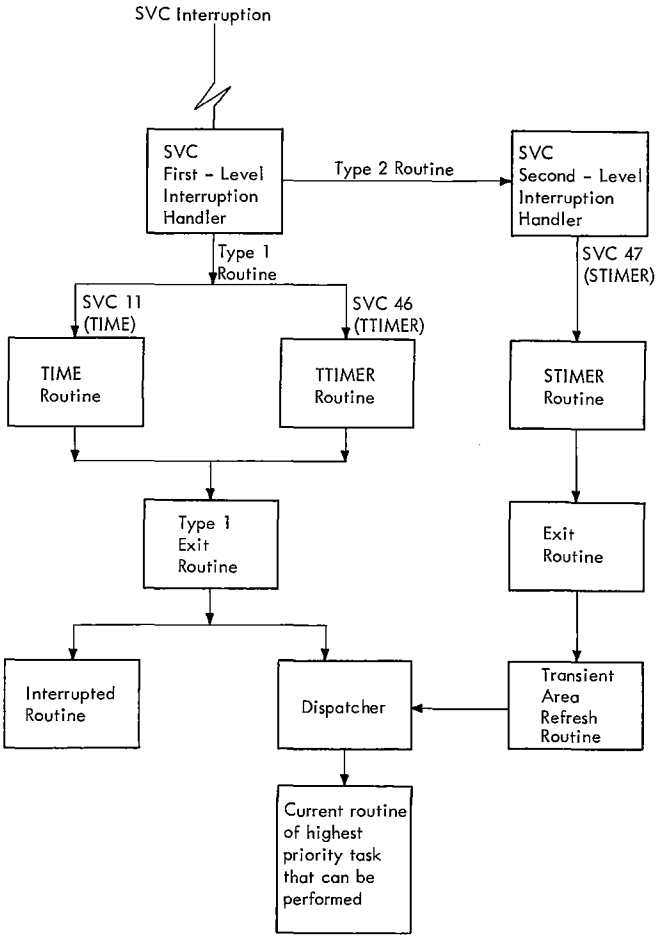


Figure 6-1. Timer SVC Interruption Handling

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.

THE TIME ROUTINE IN SYSTEM/360

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

THE TIME ROUTINE IN SYSTEM/370

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

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

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

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

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

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

CLOCK is the current value of the TOD Clock

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

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

If the TU operand was specified, the calculated time is converted to timer units by dividing the microsecond value by 26.04166. If the BIN or DEC operands were specified, the calculated time is divided by 10000 and the result is returned if BIN. If DEC was specified, the result is converted to HHMMSSth format.

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

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 in System/360

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

TOX = (SHPC - Timer) + interval requested

For example, assume that no requests are pending and that 3 hours have elapsed since the operator issued a SET command. Figure 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 occurs on

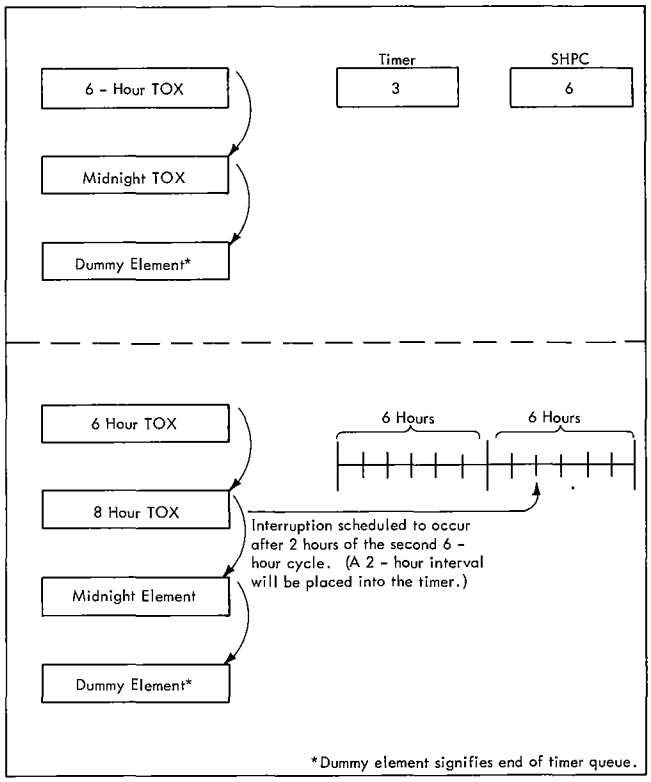


Figure 6-2. Positioning of Elements on the Timer Queue

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.

The Timer Queue in System/370

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

hour (in timer units) in the field TQEVAL in the element, sets on the synchronization indicator in the element, and uses the timer enqueue routine in the timer SLIH to convert the interval from timer units to a time of expiration and to place the element on the timer queue. Although the interval specified was greater than one hour, a timer interruption occurs in one hour and the element is examined by the timer SLIH for further processing (see "Timer Interruption Handling" below).

Determining Interval Values in System/360

The STIMER macro instruction allows the user to specify the time interval in one of four ways: decimal form (DINTVL), binary form (BINTVL), timer units (TUINTVL), and time of expiration (TOD). When decimal or binary form is given, the STIMER routine converts the value to timer units (one timer unit = 26.04166667 microseconds), before using the timer enqueue routine. When timer units are given, no conversion is necessary.

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

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

where:

LTPC = Starting time given by the operator in the SET command.

T4PC = Value in the 24-hour pseudo clock.

SHPC = Value in the 6-hour pseudo clock.

Timer = Value in the timer.

If the specified time of expiration is the same as the current time or has already passed, the calculation of the time of expiration results in either zero or a negative number. To this calculated expiration time (zero or a negative number), the STIMER routine adds 24 hours.

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

Determining Interval Values in System/370

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

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

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

CLOCK is the current value of the TOD Clock.

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

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 the format of the timer queue element (TQE) in Section 12, "Control Blocks and Tables".

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 6-3).

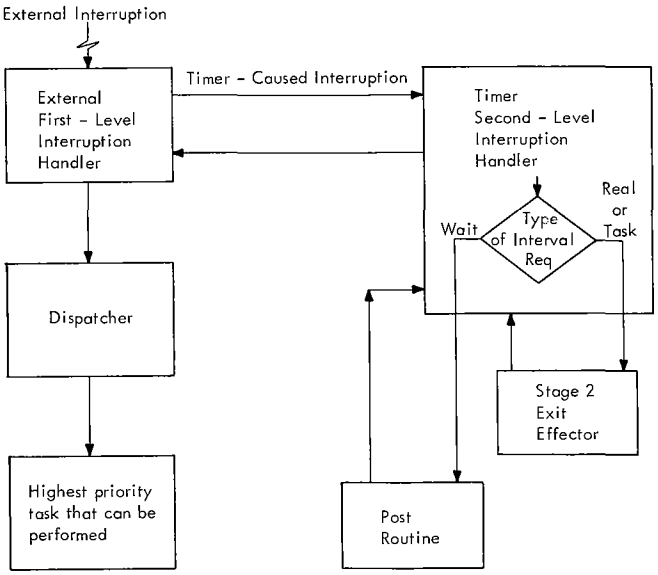


Figure 6-3. Timer Interruption Handling

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.

The module name of the Time Second-Level Interruption Handler depends on the options included at system generation. Figure 6-4 indicates these module names.

Determining What Actions Are To Be Performed in System/360

When a timer interruption occurs, the topmost element in the timer queue represents the expired interval. The Timer Second-Level Interruption Handler obtains the address of the topmost element from main storage location TQPTR, removes the element from the timer queue, and to determine what action to take, examines bits 6 and 7 of the first word in the element (see Figure 6-5).

Note: When the time sharing option is included in the system and the expired TQE is for the time sharing driver, the Timer SLIH issues a TSEVENT macro instruction with the TSLICE parameter. If the expired TQE is for a time sharing user not in main storage, a TSEVENT macro instruction is issued with the USERRDY parameter to indicate to the driver that the time sharing user is to be brought into main storage (swapped-in).

Module Name \ OPTION	MP65*	SMF	TSO	TOD
IEAQTI00		N	N	N
IEAQTI01		Y	N	N
IEAQTI02		N	N	Y
IEAQTI03		Y	N	Y
IEAQTIMP	*	N	N	N
IEAQTIM1	*	Y	N	N
IKJEATI0		N	Y	N
IKJEATI1		Y	Y	N
IKJEATI2		N	Y	Y
IKJEATI3		Y	Y	Y

Figure 6-4. Timer Second-Level Interruption Handler Module Name Determination

If the 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

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.			

Figure 6-5. Actions Taken After Timer Expiration

value from its save slot (TQESAV) to the time of expiration/time remaining slot (TQEVAL) within the TQE.

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

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

Determining What Actions Are To Be Performed in System/370

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

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

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

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

When intervals represented by either the 6-hour or midnight 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 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.

Note: When the time sharing option is included in the system and the midnight element expires, a TSEVENT macro instruction is issued with the CHGTOD operand so that the time sharing driver updates the time of day in its internal control blocks.

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

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

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

SMF Processing

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

If the timer interruption is recognized as following from the expiration of a supervisor 10-minute interval, the Timer SLIH obtains the accumulated system wait time for the 10 minutes from the second word of the save area SYSWSAVE. This value is added to the SMCAWAIT + 4 field in the system management control area. Each time that step termination is entered, this field is checked. If it is non-zero, a system 10-minute wait record is generated.

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

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

SMF TIME/OUTPUT LIMIT EXPIRATION ROUTINE (IEATLEXT)

This routine, which is resident in the nucleus, provides an interface with a user time limit expiration routine (IEFUTL).

The routine receives control after a job-step, or wait time limit has expired (see above). After setting the job or step nondispatchable, it passes control to the user time limit routine, indicating the type of expiration in Register 15. It also passes the address of a 72-byte register save area and the contents of the user data field (TCTUDATA) in the TCT.

The user routine determines whether or not a time extension will be granted. It returns control to the SMF Time/Output Limit Expiration routine with a return code of 0 for no time extension, 4 for time extension granted. If the return code is 4, Register 1 contains the number of timer units to be granted.

If an extension is granted, the SMF Time/Output Limit Expiration routine places the value of the extension into the expired TQE and places the TQE back on the timer queue.

If no extension is to be granted, the action taken depends on the type of time limit that has expired:

- If wait time expired, the problem program is abnormally terminated with an error code of 522.
- If job-step time expired, the step is terminated. The SMF Wait Time Expiration routine converts the TQE into an IRB/IQE for standard linkage to the Stage 2 Exit Effector.

Note: This routine (IEATLEXT) also handles output limit processing for SYSOUT data set. See Section 11, "Special Features".

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.

Determining Remaining Time in System/360

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

Determining Remaining Time in System/370

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

Canceling an Interval

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

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 task control block (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 without Multiple Console Support (MCS) is performed by three resident modules in the nucleus (Initialization module IEECVINT, Unit Control Module IEECVUCM, and the Wait module IEECVCTW) by the nonresident Graphic Console Initialization Module (IEECVGCI), and by the following nonresident SVC 72 modules: the Router (IEECVCTR), the External Interruption Handler (IEECVCTX), and the device processor modules with their associated open/close modules. The Initialization module is linked to by the Master Scheduler and 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 addresses (this list is called the event indication list). The address of this list is contained in the UCM. When one of the event control blocks (ECBs) is posted, the communications task becomes a ready task. When it becomes the active task, the Wait module issues an SVC 72 instruction. This SVC includes a common module, the Router module, and four service modules. The processing services performed, in order of priority, are: external interruption, attention, input/output completion, and WTO(R).

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. Due to possible ENQ interlocks, and ENQ macro instruction is not issued for the Communications Task, SYSLOG (in MCS), a task in DAR, and SIRB, or any TCB that is higher

than the Communications Task on the TCB ready queue. Instead, an additional buffer is obtained. The task receives 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 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.

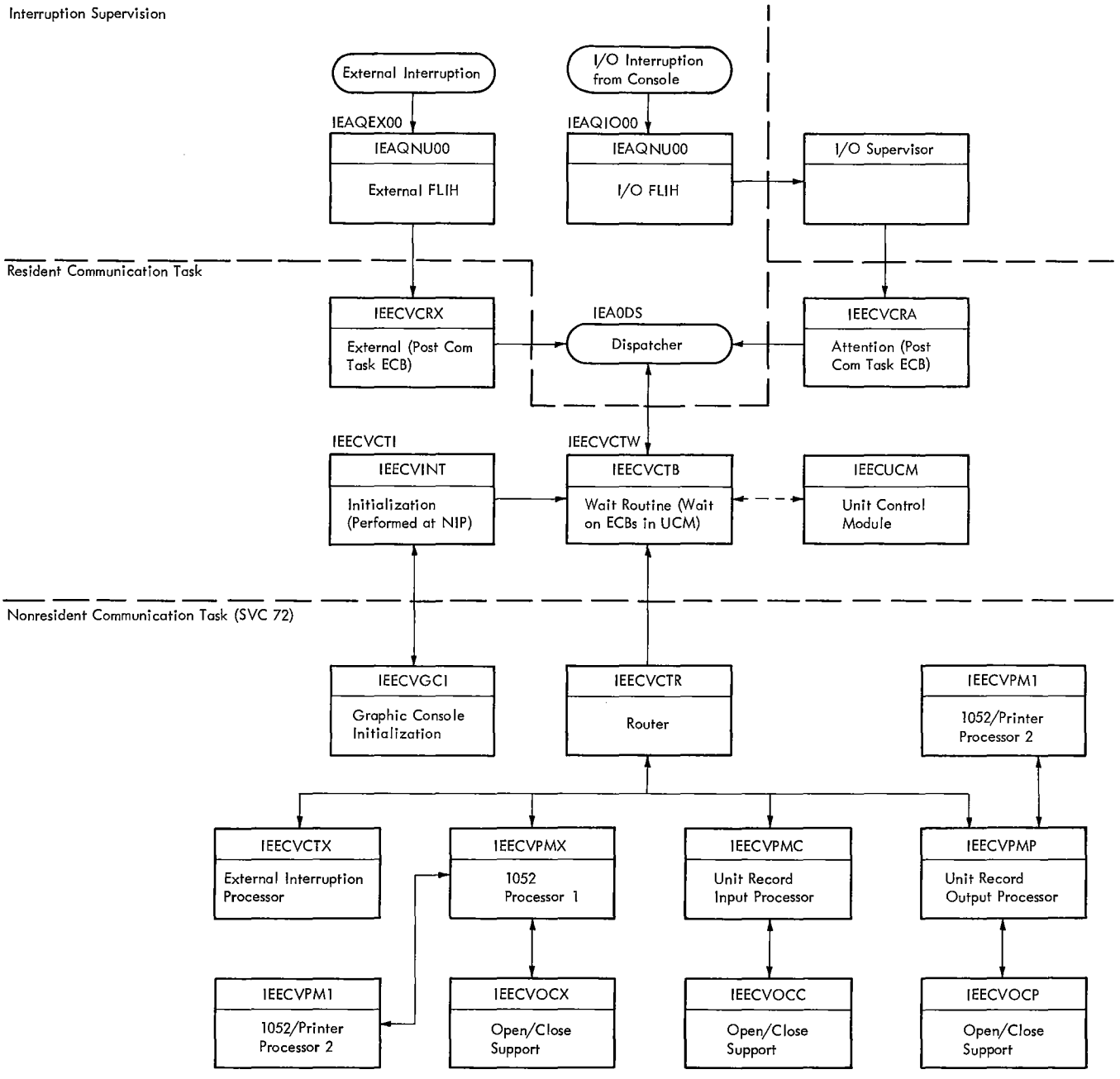


Figure 7-1. Console Support: Input

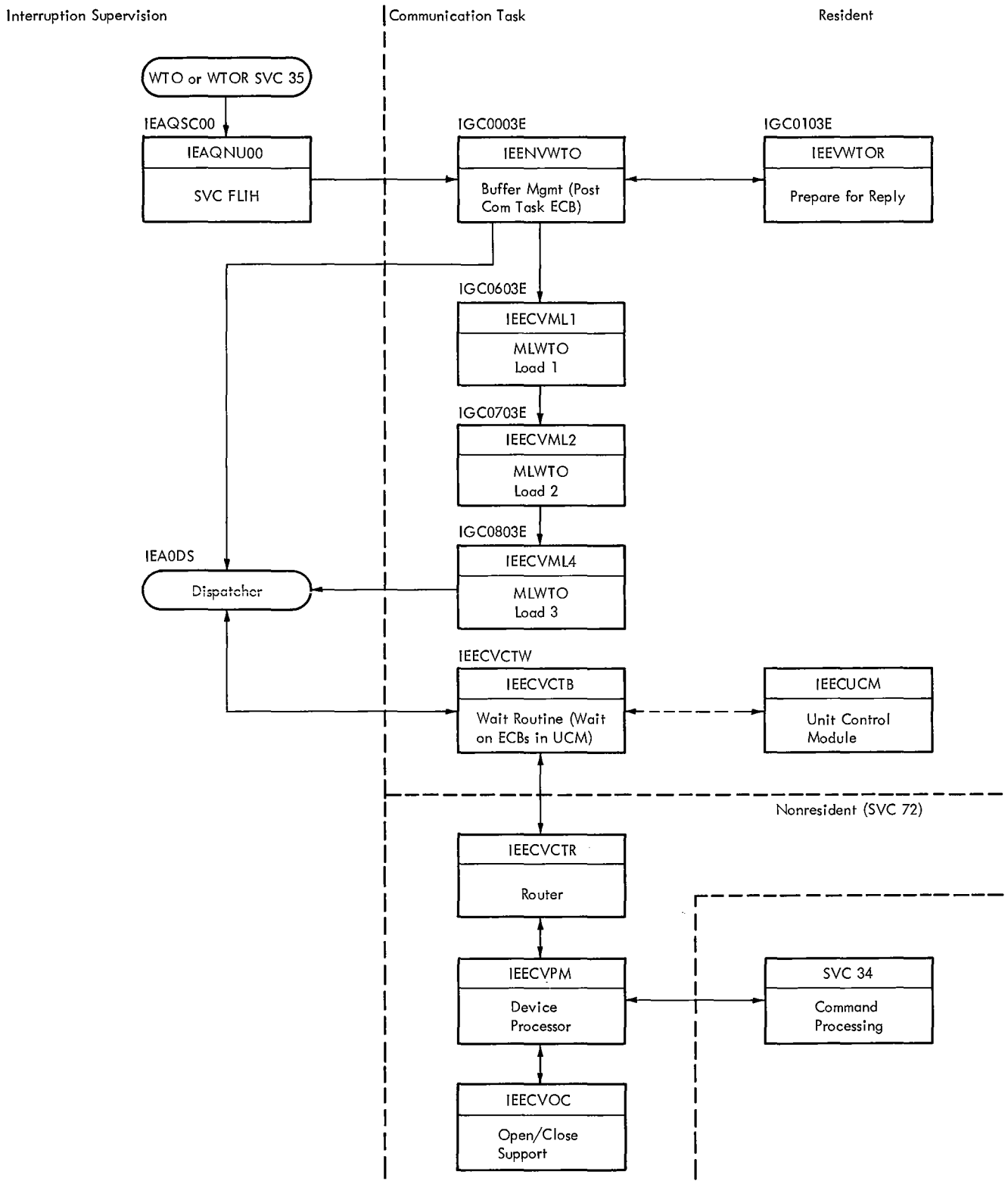


Figure 7-2. Console Support: Output

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 requester, 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 requester's region, and the specified ECB, also in the requester's region, is posted.

An operator reply is processed by the communications task Reply Processor routine (IEE1203D). 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 or swapped out.

If the job step is currently rolled or swapped 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 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 (queuing) 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 one the RPQE count in the unit control module. This count indicates the number of reply buffers that are in 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.

TSO Processing: When the time sharing option is included in the system, the Reply Processor routine determines from the RQETJID0 and RQETJID1 fields if the user who issued the WTOR macro instruction is in main storage. If not, a TSEVENT macro instruction with the USERRDY operand is issued to inform the time sharing driver that this user is to be brought into main storage (swapped-in).

MULTIPLE-LINE WRITE TO OPERATOR ROUTINES

Multiple-line Write to Operator (MLWTO) routines process multiple-line WTO requests by building write queue elements (WQEs). When all write queue elements have been built, the WTO ECB in the UCM is posted and control is returned to the calling routine.

Multiple-Line Write to Operator, Load 1

Multiple-line Write to Operator, Load 1 (IGC0603E) is entered from the Write-to-Operator routine (IEENVWTO) to process multiple-line WTO requests. IGC0603E builds the major WQE for the multiple-line

WTO. Upon initial entry, the routine determines if the entry has been made to add additional lines to an existing WTO. If so, the major WQE has already been built, and control passes to Load 2 (IGC0703E).

If IGC0603E is entered to build a major WQE, the first line passed is checked to ensure that it is within the user's storage. The number of lines passed is also resolved; this number determines the setting of the line number field in the WQE:

Number of lines passed	Line number field in WQE
Zero or less	1
Greater than 10 for a problem program	10
Greater than 255 for a supervisor mode or protect key 0 program	1

If no error conditions are found, IGC0603E attempts to obtain buffer space for the major WQE. If space is not available, and the routine issuing the MLWTO request is (1) the communications task, (2) the Damage Assessment routine (DAR), or (3) a system interruption request block (SIRB), buffer space innmain storage is immediately obtained by the GETMAIN macro instruction. Otherwise, IGC0603E issues an ENQ macro instruction and then a WAIT macro instruction, specifying the WQE message buffer request ECB in the UCM. When a buffer becomes available, it is obtained by means of a GETMAIN macro instruction.

When the buffer has been obtained, the text fields are filled in, and exit is made to Load 3 (IGC0803E).

Multiple-Line Write to Operator, Load 2

Multiple-line Write to Operator, Load 2 (IGC0703E) is entered to build minor WQEs. Upon entry, IGC0703E checks for available buffer space for the minor WQE. If none is available, and the routine issuing the MLWTO request is the communications task, DAR, or an SIRB, a GETMAIN macro instruction is issued immediately for the required main storage. Otherwise, IGC0703E issues an ENQ macro instruction and a WAIT macro instruction, specifying the WQE buffer request ECB in the UCM.

When a buffer is obtained, the text fields are filled in. If more lines remain to be written out, processing continues. When the end line is reached and all lines have been placed on the system output queue, IGC0703E posts the WTO ECB and exits to the calling routine.

Multiple-Line Write to Operator, Load 3

Multiple-line Write to Operator, Load 3 (IGC0803E) completes building the major WQE. Upon entry, IGC0803E stores the MLWTO identification number in the appropriate areas of the major WQE. If there are minor WQEs to be built, or if additional lines are to be added to an existing WTO requiring minor WQEs, control passes to Load 2 (IGC0703E). Otherwise, the WTO ECB in the UCM is posted and exit is made to the calling routine.

WRITE TO PROGRAMMER PROCESSING

An extension of the WTO/WTOR function allows the system to communicate with the programmer instead of, or in addition to, the operator. When a WTO or WTOR macro instruction is coded with the ROUTCDE=11 parameter, the message is written to the system message class output data set. The message may also be written to the console, depending upon other conditions. For a detailed explanation, refer to the MVT Job Management PLM.

SUPPORTING MULTIPLE CONSOLE COMMUNICATIONS

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

As in systems without MCS, control is routed by the External First-Level Interruption Handler for external interruptions, and by the I/O First-Level Interruption Handler for console device attentions and I/O complete interruptions (see Figure 7-3). WTO(R) and DOM macro instructions are handled by SVC 35 and SVC 87 respectively. The routing results in the posting of one of the 4 ECBs (external, attention, WTO(R), DOM) in the Unit Control Module (UCM), or the posting of one of the I/O ECBs pointed to by the event indication list (EIL) within the UCM. The communications task TCB, created within the nucleus at system generation, is then indicated as ready. The dispatcher passes control to the communications task when its TCB is the highest priority TCB on the queue.

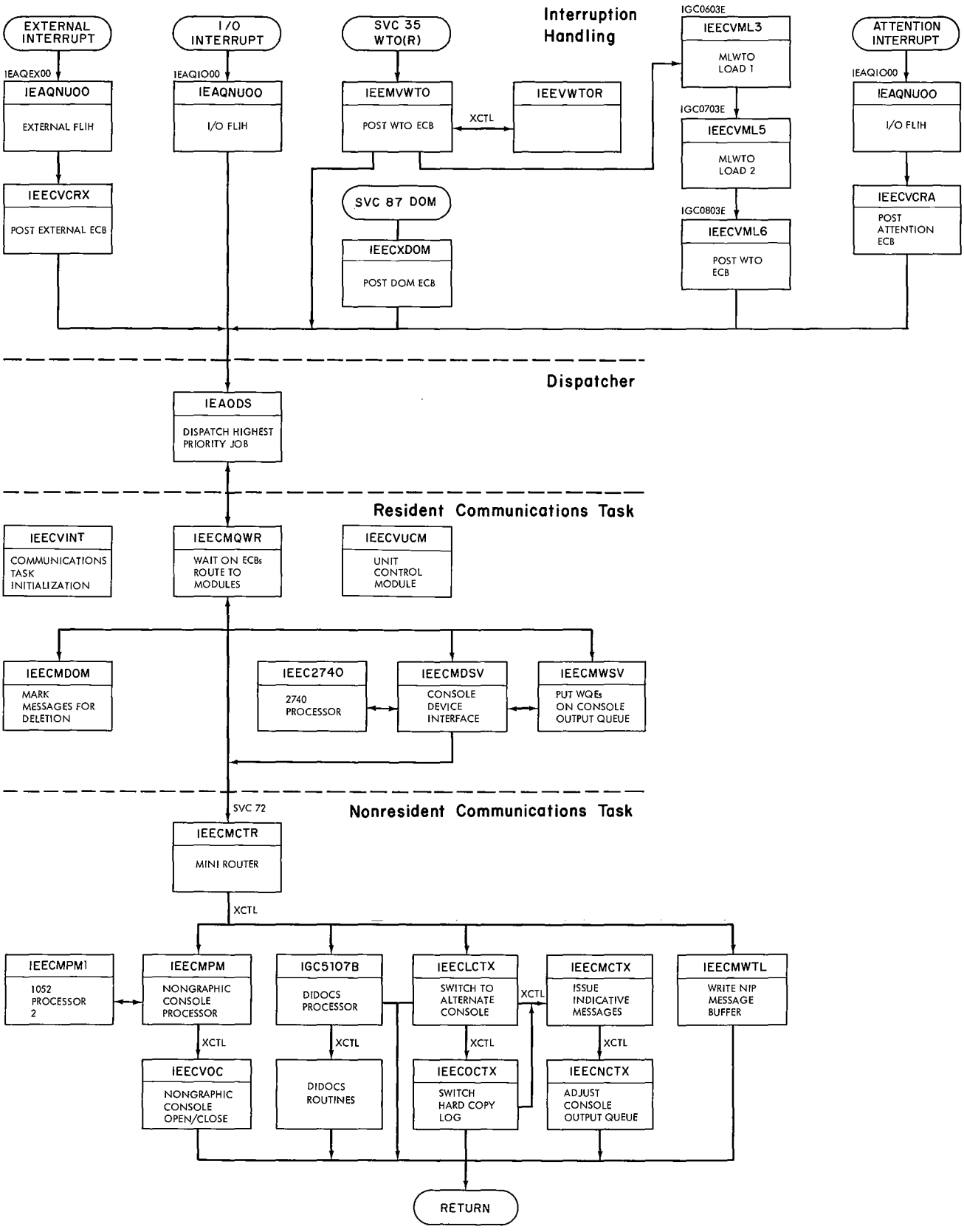


Figure 7-3. Communication Task with MCS

In addition to the ECBs and the pointer to the EIL, the UCM also contains pointers to the system output queue, the reply queue elements, and the UCM entries (see Figure 7-4). At system generation, a UCM entry is constructed for each console device specified by the SCHEDULR and SECONSLE macro instructions (a composite console has 2 UCM entries). Each UCM entry contains pointers to the processor module for that device, to the console output queue, and to the alternate console, and contains the routing codes and command authority codes assigned to the device. The console output queue is a series of pointers to selected WQEs on the system output queue. The first byte of each pointer contains indicators reflecting the status of the corresponding WQE in relation to the device.

The communications task for MCS consists of 8 modules in the nucleus, the Console Switch, Mini-Router, and NIP Message Buffer Writer modules in the SVCLIB, and the Graphic Console Initialization module in the LINKLIB. These modules are:

- Console Initialization Module (IEECVINT), which is loaded by the Master Scheduler, initializes the console configuration.

- Graphic Console Initialization Module (IEECVGCI), which initializes the display console configuration.

- Unit Control Module (IEECVUCM), which is a non-executable module containing pointers and indicators. This is the primary control table for console communications.

- Router Module (IEECMQWR), which receives control when the communications task is entered. It passes control to the service modules for ECB postings and for queue management.

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

- Device Interface Module (IEECMDSV), which passes control to appropriate device support routines when there is I/O to be performed, or consolidates system and console output queues.

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

- DOM Service Module (IEECMDOM), which marks specified operator messages as deletable from CRT (Cathode Ray Tube) consoles only.

- Mini-Router Module (IEECMCTR), which passes control from the resident communications task module to the appropriate nonresident communications task modules.

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

- Attention Handler Module (IEECVCRA), which receives control from the I/O Interrupt Handler to post the attention ECB.

- External Interruption Handler (IEECVCRX), which receives control from the External First-Level Interruption Handler to post the external ECB.

The following paragraphs describe the executable communications task modules. Unless otherwise stated, all returns are to the calling routine or module.

Console Initialization Module (IEECVINT)

The Console Initialization module is loaded by the Master Scheduler to initialize the operator consoles. If the hard copy log is a console, the NIP Message Buffer ECB in the UCM is posted. If the hard copy log is SYSLOG, the ECB posting is bypassed, permitting the System Log Initialization routine to print the buffer. The Console Initialization module then searches the UCM, changing the UCB name of each device to an address and setting the open pending flag. If an address cannot be determined for a UCB name, a message is issued to the master console and the search continues with the next UCB name. The console performing the IPL was assigned as the master console by NIP, and the Console Initialization routine only prepares secondary consoles. A message is written to each MCS console advising the operator of its unit address, its alternate's address, its console identification number, its display area configuration, its command code and routing code authorization, its status (active or inactive), and whether it is the master console, a secondary console, or the hard copy log. If the system includes display consoles, control passes to the Graphic Console Initialization Module (IEECVGCI). Upon return from IEECVGCI, the Console Initialization module returns control to the Master Scheduler IPL routine.

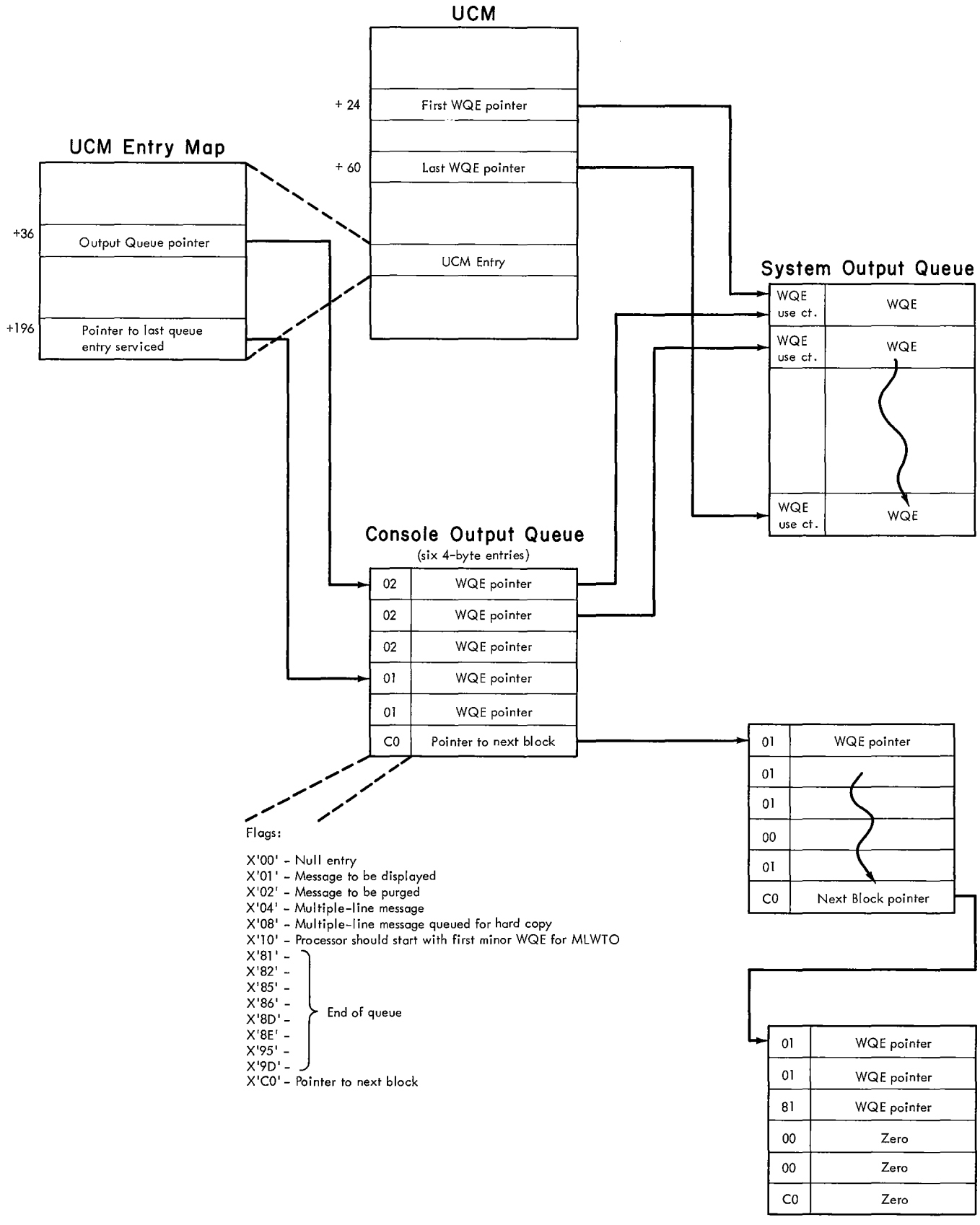


Figure 7-4. System and Console Output Queues

Graphic Console Initialization Module (IEECVGCI)

The Graphic Console Initialization module is entered from the Console Initialization Module when that routine determines that console initialization is required for one or more display consoles. IEECVGCI first searches for a UCM representing a display console. If none are found, control is returned to the Console Initialization routine. If one is found, the routine issues a LOCATE macro instruction to search for SYS1.DCMLIB. When SYS1.DCMLIB is found, the routine issues an OPEN macro to open SYS1.DCMLIB. If either the LOCATE or the OPEN fail, an error message is written to the operator's console, and one console in each transient group is made resident (the console whose TDCM was initially placed in the transient area). If SYS1.DCMLIB is successfully opened, IEECVGCI attempts to read a copy of the PFK definitions for each console into main storage. If the read is unsuccessful, a message is issued to the operator. When all display consoles in the system have been initialized, control returns to the Console Initialization Module.

Mini-Router Module (IEECMCTR)

This module is entered as a result of an SVC 72 instruction issued by a resident communications task routine. An SVRB is created as a result of the SVC instruction for the execution of this module and the other nonresident communications task and device processor modules. IEECMCTR issues an XCTL macro instruction to pass control to the appropriate device processor module.

Router Module (IEECMQWR)

The Router module contains a WAIT macro instruction and a series of ECB and status tests in the following order:

1. RMS Processing
2. External Interruption
3. Attention Interruption
4. I/O Completion Interruption
5. Output Processing
6. WTO(R) Processing
7. Queue Management
8. DOM Processing
9. NIP Message Buffer Writing

When one of the ECBs specified by the WAIT macro instruction is posted, the communications task becomes a ready task. On becoming an active task, the Router module determines the service needed and passes control to the appropriate module to process the interruption. When control returns, the Router module retests all ECBs and status indicators that may have changed while the first interruption was being processed. To allow for the servicing of higher-level interruptions, output processing returns to the Router after each console output queue is processed (see IEECMDSV below). When no further processing can be done, the WAIT macro instruction is re-issued and control returns to the dispatcher.

Console Switch Modules (IEECLCTX, IEECMCTX, IEECNCTX, IEECOCTX)

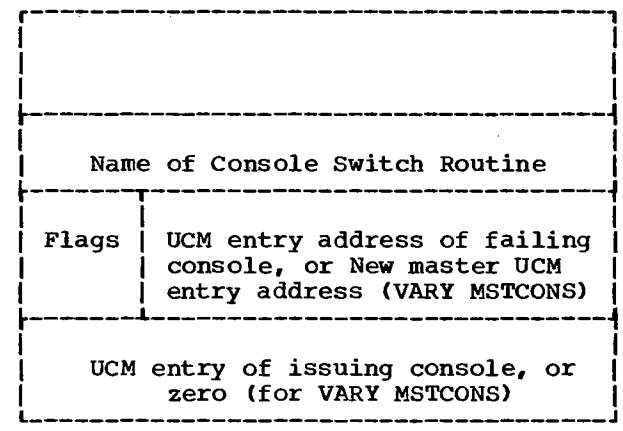
These modules receive control from:

- Router module to switch master consoles as a result of an external interruption.
- Device support routines to switch to the alternate console when there is an unrecoverable I/O error on a console.
- SVC 34 to switch to the alternate of the master console as a result of a VARY MSTCONS command.
- IEECMDSV to switch the hard copy function from the SYSLOG to the master console when SYSLOG is inoperative.

IEECLCTX uses the parameter list passed by the calling routine to determine the type of function required (see Figure 7-5). If entered because of a VARY MSTCONS command, IEECLCTX adds the routing and command codes of the master console to those of its alternate. IEECLCTX passes control to IEECOCTX if the master console was the hard copy log and its alternate is a graphic console. Otherwise, control is passed to IEECMCTX to issue indicative messages.

If IEECLCTX was entered because of hard copy failure on SYSLOG, control is passed to IEECOCTX.

If IEECLCTX was entered because of an external interruption or for a failing console, the routing and command codes of the master or failing console are added to those of its alternate. If the new console is a graphics console and the old console was also the hard copy log, control is passed to IEECOCTX. Otherwise, for a successful console switch, control is passed to IEECMCTX.



Flags: X'80' - Reserved
X'40' - Reserved
X'20' - Reserved
X'10' - Reserved
X'08' - VARY MSTCONS command
X'04' - External interruption
X'02' - Switch hard copy from SYSLOG to master console
X'01' - Reserved
X'00' - Console switch

Figure 7-5. Console Switch Parameter List

If the alternate of a failing console is a graphics console in output-only status, IEECLCTX may force a status switch so that the alternate can handle the functions of the failing console.

If a primary console does not have an active alternate, the routing and command codes are added to those of the master console. If the master console does not have an active alternate, a message is issued to all active consoles requesting a VARY ,STCONS command from any console. If there are no active consoles and a console device is included that has an audible alarm, an OPEN macro instruction is issued for the device and the alarm is sounded three times. For all cases where a console switch cannot be successfully made, IEECLCTX returns control to the calling routine.

IEECMCTX constructs two messages to indicate to the operator that a console switch has occurred and the attributes of the new console. IEECMWSV is used to queue the WQEs to the console output queue of the new console. IEECMCTX passes control to IEECNCTX to adjust the output queues of the new console.

IEECNCTX places the WQEs that were on the output queue of the old console on the console output queue of the new console, deleting duplicate messages. If the WQE represents a system status display, the WQE is deleted. Multiple-line WTO messages, other than status displays, are passed to the new console if the end line of the message is on the old console's message queue at the time of the console switch. The close pending indicator is set and the device busy flag is set off in the UCM entry of the old console, and IEECNCTX returns control to the calling routine.

IEECOCTX switches the hard copy log to the alternate of a new console when the new console is a graphic device and the old console was performing the hard copy function. If the alternate console is also a graphics device, IEECOCTX searches all active consoles starting with the master console for an active, nongraphic device. If none is found, the master console is specified as the hard copy device. When the hard copy log has been assigned to a console, a message indicating this is placed on the console output queue of that console, the WQEs for the hard copy log are placed on the console output queue. IEECOCTX passes control to IEECMCTX. If the WQE represents a system status display, the WQE is deleted. Multiple-line WTO messages, other than status displays, are passed to the new console if the end line of the message is on the old condole's message queue at the time of the console switch.

IEECOCTX also switches the hard copy log from SYSLOG to the master console when there is a SYSLOG failure. If the master console is a graphic console, IEECOCTX searches for an active, nongraphic device as previously described, but does not requeue WQEs, and returns control to the calling routine instead of IEECMCTX.

Device Interface Module (IEECMDSV)

This module consists of 4 subroutines that control the interface with the device support processor routines and manage the output queues for the devices. It receives control from the Router module when an attention or I/O ECB has been posted, when it has been processing output and there is more output to process (see DEVSERVA), or when an output queue needs consolidating. It also receives control from the WTO(R) processing routine IEECMWSV when there is output to be processed.

DEVSERVB receives control when an attention or I/O ECB has been posted. After satisfactorily testing console conditions, control is passed to DEVSERV to interface with the appropriate device support routine.

DEVSERVA receives control when there is more output to be processed. When it is first entered, it searches from the first

UCM entry for a UCM entry of an active console whose output queue can be processed. When it is subsequently entered from the Router, the search starts with the next entry after the last UCM entry processed. The search ends when an output queue is found that can be processed, or when the last UCM entry has been inspected. If an output queue is found, control is passed to DEVSERVA to interface with the appropriate device support routine.

Note: Because one WQE may be placed on several device output queues, and because DEVSERVA handles only one UCM entry each time it is entered, DEVSERVA may be re-entered from the Router module to finish processing output queues. DEVSERVA must return to the Router after processing each output queue to allow for the servicing of the higher priority external, attention, and I/O ECBs which may have been posted while DEVSERVA was processing.

DEVSERV is entered from DEVSERVA or DEVSERVB to branch to the appropriate device support routine. Upon return, if entries on the console output queue have been processed and marked as no longer needed, control is passed to the DEQ subroutine.

DEQ receives control when DEVSERV or CLEANUP need console output queues serviced. DEQ inspects the console output queue for WQE pointers marked as no longer needed. Each WQE pointer so flagged is marked as a null entry and the use count of the WQE is decremented by one. If this decrementing results in the use count reaching zero (all specified consoles have received the message), and DEQ determines that the message is to go to hard copy, control is passed to IEECMQCN (in IEECMWSV) to put the WQE pointer on the hard copy device's output queue, or a WTL is issued to SYSLOG. If the message is not to go to hard copy and there is no reply queue element associated with the WQE, the WQE is freed or marked as available. If a major WQE, representing a multiple-line WTO, is flagged as having one of its related minor WQEs with a use count of zero (which indicates that the minor WQE's message has been passed to all consoles), the remainder of the WQE chain is searched, and the storage of any available minor WQEs is returned to the system. If a major WQE is flagged as no longer needed, the entire major/minor chain is returned to the system. Return is made when the last WQE pointer on the console output queue has been examined.

CLEANUP receives control when system output queues need consolidation. It examines each WQE on the system output queue and control is passed to DEQ for each WQE that has been serviced and is to be sent to hard copy. If it isn't to be sent to hard copy, DEQ frees the WQE. Control returns to CLEANUP to process all WQEs on the system output queue.

WTO(R) Service Module (IEECMWSV)

This module gains control from the Router when the WTO ECB is posted. WQEs that have been queued to the system output queue and have not been serviced by this routine are processed at this time. If a user exit routine is provided, a parameter list containing the text, routing codes, and descriptor codes is passed to it. If the routing codes have not been modified or suppressed, if the WQE is for a WTOR, or if there is no user exit routine, control is passed to the subroutine IEECMENQ which compares routine codes, IDs, and hardcopy requirements, and places the WQE on the appropriate console output queues. If the WQE represents a multiple-line WTO, the routine sets an output-pending flag in the console to which the request has been queued. When all WQEs have been examined, control passes to IEECMDSV (DEVSERVA) to initiate output queue processing. Upon return, control is returned to the Router.

DOM Service Module (IEECMDOM)

This module gains control from the Router when the DOM ECB is posted (by the DOM macro instruction being issued in a problem program or system task). It may also be entered directly from the system purge routine (IEECMED2) at the end of a job step. When entered from the Router, message IDs in the DOM element list are compared with the WQEs on the system output queue. The matching WQEs are marked for deletion unless the WQE is to go only to hard copy. If a Model 85 Operator Console is active, control passes to the processor routine for that device for deleting messages from the console screen. Upon return, the DOM element list is freed by a FREEMAIN macro instruction and the DOM ECB is cleared.

When entered from the system purge routines, IEECMDOM compares WQEs with a protect key. A request for purge of protection key zero in ignored. Those that match are marked for deletion unless they are to go only to hard copy. If any graphic consoles exist, the messages in each device's Display Control Module are similarly compared and marked for deletion.

NIP Message Buffer Writer Module (IEECMWTL)

This module issues an SVC 36 if the SYSLOG has been specified as the hard copy log and has been initialized. Otherwise, an SVC 35 is issued to write the NIP messages to the operator.

REPLY PROCESSING

MCS reply processing, a function of SVC 34, differs from non-MCS reply processing in that when a WTOR is issued to more than one console, a reply is accepted from any console that received the WTOR. The first reply accepted is broadcast to all consoles that received the WTOR.

CONSOLE SUPPORT

Console support is part of the Communications Task. The following modifications are made to the Communications Task:

- Pointer to the console support processor is added to the UCM Entry for the 2740 only.
- Pointer to the master DCM is added to the UCM Entry for every CRT console.

UNIT CONTROL MODULE

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 Support routines.

CONSOLE SUPPORT ROUTINES

This section describes the logic flow of the support routines for the IBM 1052 Printer-Keyboard, the IBM 1403, 1443, and 3211 Printers, the IBM 2540 Card Read Punch, and the IBM 2740 Communications Terminal Model 1. The 1052, 1403/1443, and 2540 may operate with or without Multiple Console Support (MCS). Changes to module operation resulting from the inclusion of MCS are noted. The 2740 is available only in systems that include MCS (See Figure 7-6).

1052 Console Support Routines (MCS Optional)

The console support routines for the IBM 1052 Printer-Keyboard perform Read, Write, Open, and Close functions, as well as buffer management.

- 1052 Console Processor 1 (IEECVPMX): Provides I/O buffer management and constructs channel programs to perform read and write operations in systems without MCS.
- 1052 Console Processor 1 (IEECMPMX): Contains MCS modifications to IEECVPMX processing. IEECMPMX does not perform buffer management. This function is performed by IEECMDSV and IEECMWSV.
- 1052 Console Processor 2 (IEECMPM1): Provides the processing necessary to present multiple-line WTO messages, including system status displays, on the 1052 operator console and printer consoles in systems with MCS. It also rings the audible alarm on the 1052 when a permanent I/O error occurs.
- 1052 Open/Close routine (IEECVOCX): Provides open and close functions for the console device.

Printer Device Support Routines

The device support routines for printers are similar to those for the 1052 in that they provide Write, Open, and Close functions, and buffer management.

- Printer Processor (IEECVPMP): Provides I/O buffer management and issues a WRITE macro instruction to write to the printer. This routine operates in systems without MCS.
- Printer Processor (IEECMPMP): Contains MCS modifications to IEECVPMP processing. This routine does not perform buffer management.
- 1052/Printer Processor 2 (IEECVPM1): Processes multiple-line WTOs for 1052 Printer-Keyboards and printer consoles in systems without MCS.
- Printer Open/Close routine (IEECVOCP): Provides open and close functions for the console device.

Card Reader Device Support Routines

The device support routines for the card reader are similar to those for the 1052 in that they provide Read, Open, and Close functions and buffer management.

- Card Reader Processor (IEECVPMC): Provides I/O buffer management and issues a READ macro instruction to bring input from the card reader into a WQE in systems without MCS.
- Card Reader Processor (IEECMPMC): Contains MCS modifications to IEECVPMC processing. This module does not perform buffer management.
- Card Reader Open/Close Module (IEECVOCC): Provides open and close functions for the console device.

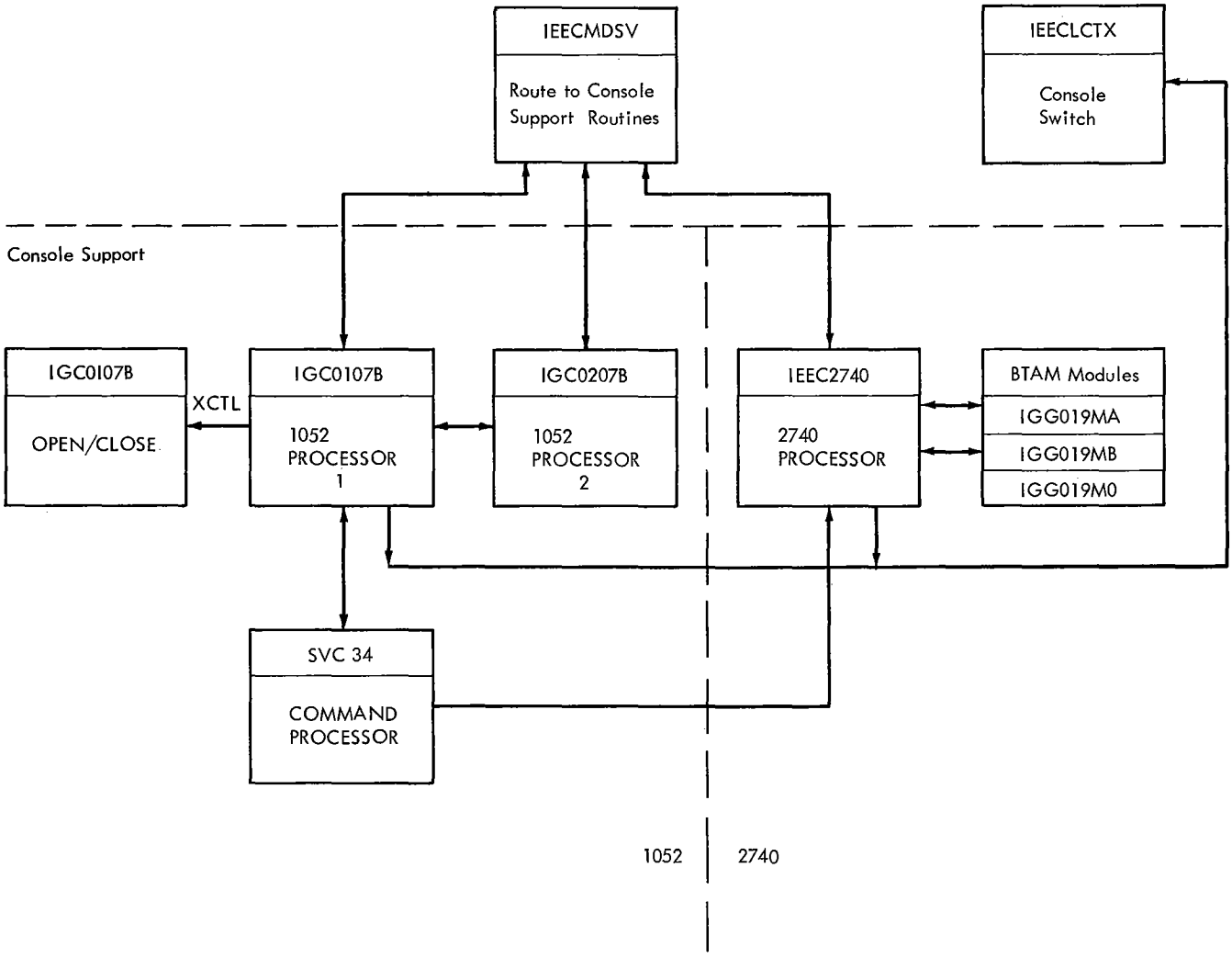


Figure 7-6. 1052 and 2740 Console Support Routines with MCS

2740 Console Support Routines (MCS Only)

The 2740 Processor routine (IEEC2740) is created at System Generation by the macro SGIHB000. It performs OPEN and CLOSE functions. The READ and WRITE functions are performed by the following unchanged BTAM modules:

- IGG019MA, BTAM Read/Write routine
- IGG019MB, BTAM Channel End and Abnormal End appendages
- IGG019M0, BTAM Device I/O module (table used by IGG019MA)

OPEN is performed when the open pending flag in the Unit Control Module (UCM) entry is on and the UCM entry is not already open. A LOAD is issued to obtain the addresses of the BTAM modules. The address of the 2740 ECB is placed in the UCM entry and Event Indication List. The DEB is initialized from the Communication Task TCB, and placed at the head of the TCB DEB queue. The Appendage Vector Table is initialized, and the line to the 2740 is initialized via the LOPEN macro instruction. The open flag of the DCB and UCM entry is turned on, and the open pending flag is turned off.

CLOSE is performed when the close pending flag in the UCM entry is on, the output queue is empty, and the console is not busy. The address of the ECB in the event indication list is replaced with the

address of the UCM entry. The address of the ECB in the UCM entry is zeroed. The DEB is removed from the TCB DEB queue. The UCB is set to indicate that it can be allocated. A DELETE macro instruction is issued on the BTAM modules. The active flags in the UCM entry and the control blocks are set to indicate that the device is closed.

READ is performed when the 2740 is not busy and a message has been sent to the terminal, or a READ I/O complete has been successfully processed and the output queue is empty. The BTAM module, IGG019MA is given control to perform this function, and, if the READ is successful, the busy flag is set in the UCM entry.

WRITE is performed when the output pending flag in the UCM entry is set, the 2740 is not busy, and the output queue is not empty. WQE pointers on the console output queue are examined. When a WQE is found that can be written, control is passed to the BTAM module, IGG019MA. If the WRITE is successful, the busy flag is set in the UCM entry.

I/O complete conditions cause the busy flag in the UCM entry to be turned off. If the complete is for a successful READ, the message text is translated from 2740 code to EBCDIC and searched for backspace and cancel codes. If the message is to be ignored, control is passed to the BTAM module, IGG019MA for another READ operation. If the message is to be accepted, as SVC 34 is issued so the Command Processor routines of the Master Scheduler can process the REPLY command. Upon return from the Master Scheduler, the processor attempts to perform output if the 2740 is not busy and there is output on the console output queue. If the 2740 is busy, control is returned to the Communications Task module (IEECMDSV). If the I/O complete condition is for a successful WRITE, the WQE pointer on the console output queue is marked as no longer needed, and control is returned to IEECMDSV.

If a condition code of 0 is returned by a BTAM module, the 2740 processor retries the I/O operation, or passes control to the Console Switch routine (IEECMCSW) in the Communications Task.

DEVICE INDEPENDENT DISPLAY OPERATOR CONSOLE SUPPORT (DIDOCS) ROUTINES (OPTIONAL)

Device Independent Display Operator Console Support (DIDOCS) support, also referred to in this publication as Display Console Support, provides uniform operator console support for the following display console devices:

- 2250 Display Unit, Models 1 or 3
- 2260 Display Station, Model 1 with 2848 Display Control, Model 3
- Model 85 CRT Display (Feature 5450)
- Model 165 Display Console
- Model 91 Display Console
- Model 195* Display Console

Note: The Model 91 and Model 195 Display Consoles are functionally equivalent to 2250 Display unit, Model 1.

The Display Console Support takes advantage of device-dependent features of each display device (such as the use of the light pen on the 2250 for message deletion). This section describes the logic flow of the Display Console Support routines (See Figure 7-7).

The Display Console Support routines receive control from MCS when one of the following events occurs for a display console:

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

The following routines comprise the Display Console Support:

- DIDOCS Processor Routines (IGC5107B, IGC5Z07B, IGC6107B, and IGC6Z07B) -- provide an interface between Display Console Support and MCS. The Processor routines receive control from MCS when an operator or system request is entered that requires processing by the DIDOCS routines. If the console involved in the request has a transient DCM, Processor 0, Load 1 (IGC6107B) receives control. Processor 0, Loads 1 and 2, (IGC6107B, IGC6207B) provide transient DCM swapping support and permanent program function keyboard (PFK) update support, and then pass control to Processor 1, Load 1 (IGC5107B) for

---

*The Model 195 applies to both System/360 and System/370 models.

continued processing. Processor 1, Load 1 receives control either from Processor 0, or directly from MCS (when the request involves a console that has a resident DCM). Processor 1, Loads 1 and 2 (IGC5107B, IGC5Z07B) route control to other display console routines as required to process the request.

- Open/Close routine (IGC5G07B) - opens and closes display console device.
- 2250 I/O 1 and 2 routines (IGC5P07B, IGC5Q07B) - handle input/output operations for the 2250 Display Unit.
- 2260 I/O 1 and 2 routines (IGC5R07B, IGC6R07B) - handle input/output operations for the 2260 Display Station.
- Model 85 I/O routine (IGC5H07B) - handles input/output operations for the Model 85 CRT Display used as an operator console.

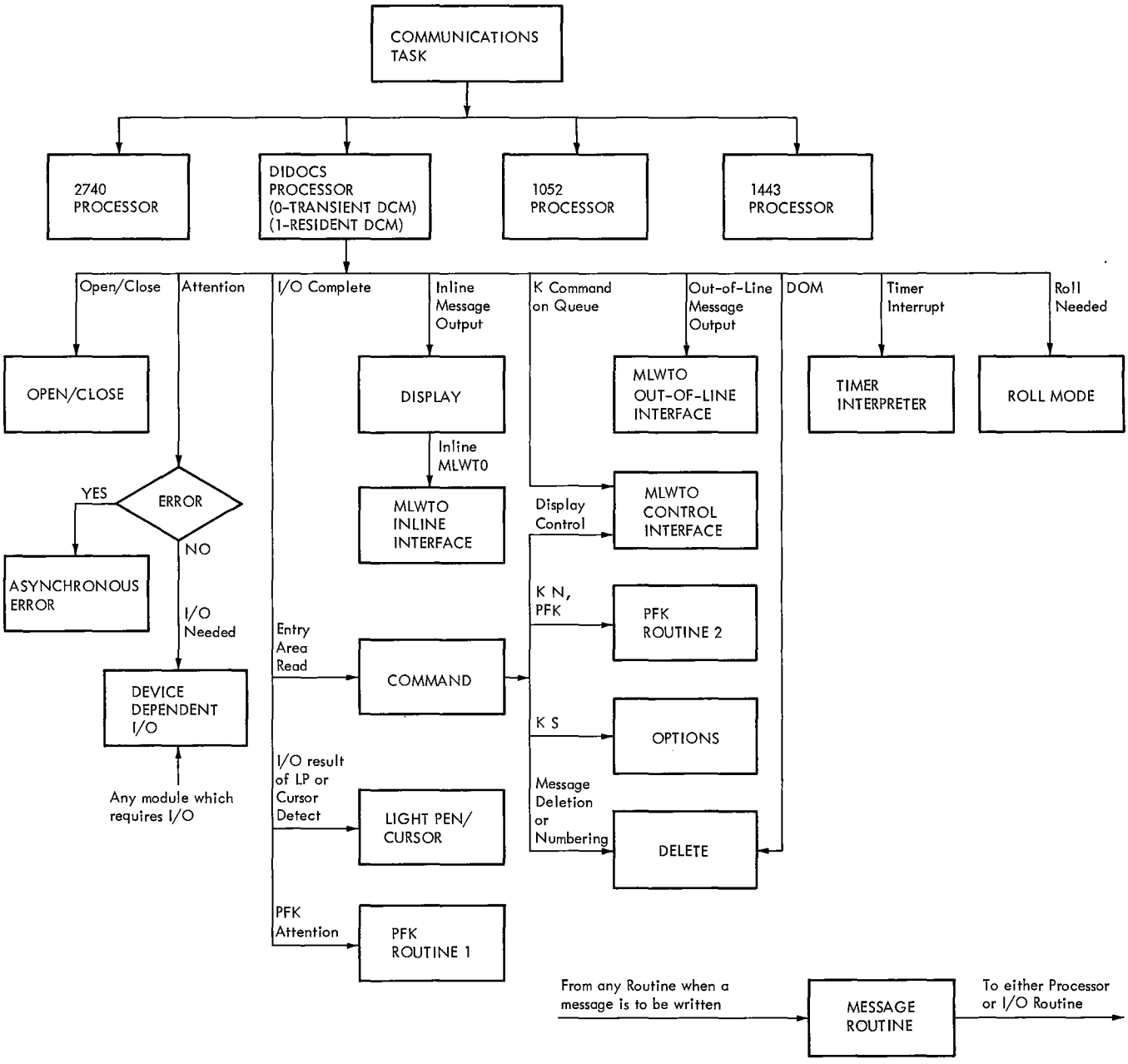


Figure 7-7. CRT Console Support (High Level)

- Asynchronous Error routine (IGC5C07B) - initializes DCM on an OPEN; blanks the screen and displays an appropriate error message for the operator. If a permanent error occurs, it passes control to MCS for console switching.

- Message 1, 2 and 3 routines (IGC5D07B, IGC5E07B, IGC6D07B) - contain all messages used for Display Console Support.

- Display 1, 2 and 3 routines (IGC5207B, IGC5307B, IGC6207B) - write all messages from the Operating System to the operator except those to be deleted and status display messages and mark messages according to their descriptor codes.

- Roll Mode routine (IGC5J07B) - removes messages from the screen at interval specified by the operator.

- Command routine (IGC5407B) - analyzes type of command in entry area and takes appropriate action, or passes control to another Display Console Support routine for action.

- Options routine (IGC5A07B) - analyzes CONTROL command entries specifying the S operand to determine their legitimacy. If legitimate, this routine changes the screen options as requested by the operator and routes control to another Display Console Support routine as required.

- Delete 1, 2, 3, and 4 routines (IGC5607B, IGC5707B, IGC5807B, IGC5907B) - erase answered WTOR messages from the screen as well as messages specified for deletion by the operator (CONTROL command, light pen, and cursor), or by the system (DOM macro instruction).

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

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

- Multiple-Line Write to Operator routines (IGC0603E, IGC0703E, IGC0803E, IGC0903E) - process multiple-line WTO requests by building the necessary write queue element (WQE), and chaining the MLWTO requests to the system output queue.

- Status Display Interface routines (IGC6L07B, IGC6M07B, IGC6N07B, IGC-6O07B, IGC6P07B, IGC6Q07B, IGC6T07B) - process inline and out-of-line MLWTOs, and handle control of out-of-line displays.

- Cleanup routine (IGC6G07B) - removes status displays from the message queues, and reinitializes the Screen Area Control Blocks (SACBs).

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

The following control blocks (including content description) are used by Display Console Support routines:

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

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

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

- Unit Control Block (UCB) - attention information.

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

- I/O Blocks - status of device.

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

Each unit DCM is divided into two sections: a resident section and a section that may or may not be resident, at the user's option. The resident portion of the DCM (the RDCM) contains screen control information and the address of the main

storage area assigned to the optionally transient portion of the DCM (the TDCM). (If the optionally transient portion of the DCM is actually transient, the main storage area addressed by the RDCM may be used by the transient DCMs of several consoles.)

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

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

- DOM addresses
- CCW area
- Input area
- Delete request buffer
- Processor name
- Option values
- Communication bits
- Field sizes for dependent section

The dependent section includes the fields shown in Figure 7-8:

Field	2260[1]	2260[2]	2250	M85
Buffer Address Table	0	0	10	8
CCW Area	80	80	112	72
Screen Image Buffer	960	960	3866	2800
COM ID Numbers	44	44	188	120
SCREEN Control Table	22	22	94	60
Secondary Screen Control Table	11	11	47	30

[1]input-output
[2]output-only

Figure 7-8. Variable Sized Fields of the TDCM

Display Console Support Routine Descriptions

This section contains a description of each of the Display Console Support routines. These descriptions provide a general explanation of each routine.

DIDOCS Processor Routines

The DIDOCS Processor routines (IGC5107B, IGC5Z07B, IGC6107B, and IGC6Z07B) receive control from MCS when any of the following operator or system requests is entered that requires processing by the DIDOCS routines:

- Open/Close request
- Attention
- Enter (command input)
- Cancel
- Light Pen (message deletion)
- PFK or Light Pen (command input)
- I/O Complete
- DOM request
- Hard copy went down
- Hard copy came back up
- Messages to be displayed
- Timer interruption
- Roll needed

DIDOCS Processor 0, Load 1

Processor 0, Load 1 (IGC6107B), receives control from Processor 1, Load 1 (IGC5107B) to begin processing of a request involving a display console with a transient DCM. The Processor 0 Load 1 routine queues the request and then determines if the transient portion of the DCM (the TDCM) is required to process the request. If the transient DCM is required and is not already in main storage, this routine brings it into main storage. Control then passes to the Processor 1, Load 1 routine to continue processing of the request.

Processor 0, Load 1, also checks for a request for a permanent PFK update. Permanent copies of PFK definitions are maintained on SYS1.DCMLIB; the operator may make permanent changes to the definitions. If a request for a permanent change to a PFK definition is encountered, Processor 0, Load 1, builds the appropriate channel program and issues an EXCP macro instruction to accomplish the change. When I/O is complete for a PFK update, Processor 0, Load 1, passes control to Processor 0, Load 2 (IGC6Z07B), to check for successful I/O.

When Processor 0, Load 1 is entered because an I/O operation is complete, control passes to Processor 0, Load 2, to

determine if an I/O error occurred or if additional I/O processing is required.

When Processor 0, Load 1, is entered from Processor 0, Load 2, (indicating that I/O is satisfactorily completed), control passes to Processor 1 Load 1.

DIDOCS Processor 0, Load 2

Processor 0, Load 2 (IGC6Z07B), receives control from Processor 0, Load 1, (IGC6107B) when I/O processing is complete. Load 2 determines if any additional I/O processing is required and if any I/O errors occurred. Action is taken as follows:

- I/O Error -- Writes an error message to the operator's console and passes control back to MCS with an indication that console switch is required.
- Additional I/O Necessary -- Updates and executes the channel program.
- I/O Complete -- passes control back to Processor 0, Load 1.

DIDOCS Processor 1, Load 1

Processor 1, Load 1 (IGC5107B), receives control to begin processing of an operator or system request (as listed above). Control is received either directly from MCS (when the console involved in the request does not have a transient DCM), or from the Processor 0, Load 1 routine (when the console has a transient DCM that has been brought into main storage).

The Processor 1 routine determines the reason for entry and passes control to the appropriate Display Console Support routine. Figure 7-9 summarizes the reasons for entry to the DIDOCS Processor routine, the resulting exit, and the reasons for exit.

DIDOCS Processor 1, Load 2

DIDOCS Processor 1, Load 2 (IGC5Z07B) receives control from Processor 1, Load 1 (IGC5107B) for processing of close requests, and for continued processing of request parameter lists.

If entry has been made for processing of a close request, Processor 1, Load 2, determines whether or not the screen has been erased. If it has not been erased, flags are set in the DCM and control is passed to the device dependent I/O routine to erase the screen. If the screen has been erased, control passes to the Open/Close routine (IGC5G07B) to complete processing of the close request.

If entry is made for processing of a request parameter list, the type of request is determined and control is passed to the appropriate Display Console Support routine. This process is a continuation of the request-handling process begun in Processor 1, Load 1. If the request in the parameter list has already been processed, control is returned to Processor 1, Load 1.

Open/Close Routine

The Open/Close routine (IGC5G07B) opens and closes the DCB for the display console devices. It decides whether to open or close a device by checking a parameter passed to it in the communication task extended save area (CXSA). It may be entered to open with a special exit to the MCS Console Switch routine (IEECLCTX) to sound the alarm three times indicating that no console is available.

After a successful open, control returns to either the Processor 1, Load 1 routine (IGC5107B), if a console exists, or the MCS External Interruption routine, if the no console condition is met. Following a successful close, control returns to the MCS Console Switch routine (IEECLCTX) via a branch on register 14.

2250 I/O 1 Routine

2250 I/O 1 routine (IGC5P07B) performs requested 2250 input/output (I/O) operations in proper screen format. It checks the communication bytes in the DCM (I/O communication bytes 1, 2, and 3 and message communication byte 1) against pre-established bit settings to determine the type and format of I/O requested.

Each I/O request is checked in this manner and, if applicable, the appropriate CCWs are built until all the I/O requests are set up in the channel program. This routine builds CCWs for the following I/O requests:

- Perform Read Manual Input (RMI)
- READ entry area
- WRITE message area
- WRITE instruction line
- WRITE entry area
- WRITE warning line
- Insert cursor

This routine also blanks the instruction line, entry area, and warning line as requested prior to performing I/O in these areas.

Reason for Entry	Exit	Reason for Exit
Open/Close		
Open	Open/Close routine	Open I/O blocks
Close	Open/Close routine	Close I/O blocks
Reopen	Asynchronous Error routine	Initialize DCM
I/O Error	Asynchronous Error routine	Handle error
Attention		
RMI Request ENTER, CANCEL, or Light Pen (LP)	I/O routine	Read Manual Input
I/O busy	MCS (BR 14)	Wait on I/O
Light Pen or PFK Command Entry	PFK routine 1	Process commands
I/O Complete		
RMI: CANCEL	Command routine	Perform CANCEL
Read	Command routine	Analyze command
In-line Status Display	Status Display Interface 1	Display status display
Out-of line Status Display	Status Display Interface 2	Display status display
Blanking to be Done	Status Display Interface 5	Blanks unused display area lines
PFK Attention	PFK routine 1	Handle PFK command entry
Transient DCM no Longer Required	Processor 0 routine	Transfer Transient DCM to secondary storage
LP, or cursor not in entry area	LP-Cursor Service routine	Interpret desired function
Hard Copy Down	Message 1 routine	Provide no hard copy message
Hard Copy Recovered	Message 1 routine	Remove no hard copy message
DOM Request	Delete 2 routine	Marks messages for deletion
Messages to be Displayed	Display 1 routine	Display message(s)
Timer Interruption	Timer Interpreter	Analyze timer interval
Roll needed	Roll Mode routine	Perform roll

Figure 7-9. DIDOCS Processor Routine Entries

If a light pen interruption occurs, control passes to the Light Pen/Cursor Service routine (IGC5F07B). If one of the following I/O operations is requested, control passes to the 2250 I/O 2 routine (IGC5Q07B):

- WRITE asynchronous error message
- WRITE message; message waiting
- WRITE status display
- Erase screen
- Sound alarm

For all other cases, control passes to the Processor 1, Load 1 routine (IGC5107B).

2250 I/O 2 Routine

2250 I/O 2 Routine (IGC5Q07B) performs its input/output operations in an identical manner to that of 2250 I/O 1 routine whenever it is called by the latter to process one or more of the following I/O requests via the building of CCWs:

- WRITE asynchronous error message
- WRITE message waiting message
- WRITE status display
- Erase screen
- Sound alarm

This routine also picks up the channel program address from DCMDSAV. The only exit from this routine, however, is to Processor 1, Load 1 (IGC5107B).

2260 I/O 1 Routine

The 2260 I/O 1 routine (IGC5R07B) performs 2260 input/output (I/O) operations in proper screen format. The module first tests for a status switch request. If this request is present, exit is made to the 2260 I/O 2 routine (IGC6R07B). It checks bits set in the DCM to determine which I/O operation is to be performed. One of three possible I/O operations may be requested: (1) perform Read Manual Input (RMI) in order to find cursor (2) WRITE screen, or (3) READ entry area. This routine normally exits to the Processor 1, Load 1 routine (IGC5107B) unless the cursor is determined to be located adjacent to the Start of Message (SOM) symbol and the screen is in "hold" mode. In the latter case, the CANCEL function is to be performed and the 2260 I/O 1 routine exits to the Command routine (IGC5407B). If the cursor is located other than in the entry area, the line and character posititioning is computed and exit is to the Light Pen/Cursor Service routine (IGC5F07B).

2260 I/O 2 Routine

The 2260 I/O 2 routine (IGC6R07B) is entered from the 2260 I/O 1 routine (IGC5R07B) when a change in a 2260 console's operating mode is requested by the operator or required by the system. IGC6R07B is called twice during each console switch. On the first pass, the routine determines the new console operating mode (full-capability or output-only) and passes control to the Cleanup routine (IGC6G07B) to continue processing of the console mode switch.

When IGC6R07B receives control for the second time during a console switch, the routine determines if "message stream" mode has been requested. If so, the routine sets a "roll delete" mode indicator in the DCM (this places the console in "roll-deletable" mode for message deletion) and exits to the Timer Interpreter routine (IGC5K07B) to continue processing of the console mode switch. If message stream mode was not requested, control is returned to the 2260 I/O 1 routine (IGC5R07B) to complete processing of the console mode switch request.

Model 85 I/O Routine

The Model 85 I/O routine (IGC5H07B) performs Model 85 input/output (I/O) operations in proper screen format. It checks the communication bytes in the DCM (I/O communication bytes 1, 2, and 3 and message communication byte 1) against pre-established bit settings to determine the type and format of I/O requested. The module first tests for a status switch request. If this request is present, exit is made immediately to the Cleanup routine (IGC6G07B).

Each I/O request is checked in this manner, and, if applicable, the appropriate CCWs are built until all the I/O requests are set up in the channel program. This routine builds CCWs for the following I/O requests:

- Read Manual Input (RMI)
- READ entry area
- WRITE message area
- WRITE asynchronous error message
- WRITE message waiting message
- WRITE status display
- WRITE instruction line
- WRITE entry area
- WRITE warning line
- Insert cursor
- Blank warning line
- Blank instruction line
- Blank entry area
- Erase screen
- Sound alarm

Control returns to the Processor 1, Load 1 routine (IGC5107B).

Asynchronous Error Routine

The Asynchronous Error routine (IGC5C07B) handles asynchronous errors and reopen conditions. In the case of permanent synchronous or asynchronous errors, this routine performs console switching by exiting to the MCS routine, Console Switch, Load 1 (IEECLCTX).

When a asynchronous error occurs, the Asynchronous Error routine sets indicators in the DCM to erase the screen and display an appropriate error message. These indicators are set for Message 2 (IGC5E07B) which moves the appropriate error message into the DCM, and the appropriate device I/O routine which erases the screen and writes the error message. Control passes to Message 2.

When a reopen condition is met, this routine initializes the DCM and passes control to the appropriate device I/O routine to write the screen.

Message 1 Routine

The Message 1 routine (IGC5D07B) contains messages used by Display Console Support. It tests bit settings in the DCM to determine which message to move into the screen image buffer while distinguishing between error and warning messages. This routine then sets indicators in the DCM to write the screen. Message 1 sets the sound alarm bit unless an "Unviewable Message" or "Deletion Requested" warning message is written. If the bit settings in the DCM indicate that a message is to be written, the Message routine moves the message into the DCM and exits to the appropriate device I/O routine. Otherwise, control passes to the Processor 1, Load 1 routine (IGC5107B).

Message 2 Routine

The Message 2 routine (IGC5E07B) contains the asynchronous error and deletion request error messages used by the Display Console Support. It tests bit settings in the DCM to determine which messages to move into the screen image buffer. This routine then sets indicators in the DCM to write the screen, sets the sound alarm bit, and exits to the appropriate device I/O routine.

Message 3 Routine

The Message 3 routine (IGC6D07B) contains the PFK support error messages used by Display Console Support. It tests bit settings in the DCM to determine which messages to move into the screen image buffer (in the DCM). The routine then sets indicators in the DCM to write the screen, sets the sound alarm bit, and exits to the appropriate device I/O routine.

Display 1 Routine

The Display 1 routine (IGC5207B) displays all Operating System messages except those to be deleted (unless DEL=N) and status display messages. It searches the output queue for write queue elements (WQEs) to be displayed. If this search is successful, control is passed to Display 3 (IGC6207B) which places the message(s) into the next available line in the screen image buffer of the DCM and marks the message(s) according to its descriptor code as provided by MCS.

If a status display is overlaying messages on the screen, Display 1 exits to either Delete 2 (IGC5707B), in case an intervention required action message(s) is on the screen, or Delete 4 (IGC5907B), if there are no action messages and automatic deletion is specified and not yet tried. If automatic deletion is not specified or has been tried, this routine exits to either Message 1 (IGC5D07B) to display an "Unviewable Message" message, or the appropriate device I/O routine to display a "Message Waiting" message. If roll mode is specified, Display 1 exits to Display 2 (IGC5307B) to set the timer as required. If a message is greater in length than the maximum text for a line in the screen image buffer, or an accepted reply is on the screen, Display 1 exits to Display 2. When no messages are added to the DCM, Display 1 returns control to MCS via a branch on register 14.

Display 2 Routine

The Display 2 routine (IGC5307B) checks bit settings in the DCM to determine whether the splitting of messages or the marking or replies is to be performed. It then either splits messages greater in length than 72 characters (70 for the 2250), or marks all accepted replies and associated WTORs as automatically deletable in the screen image buffer. In the latter case, Display 2 sets the timer as required. If splitting is performed, Display 2 exits to Display 1 (IGC5207B). It exits to whichever of the following routines is appropriate, if marking accepted replies is performed:

- Delete 2 routine (IGC5707B) - to delete intervention required action messages
- Delete 4 routine (IGC5907B) - to perform automatic deletion

- Device-dependent I/O routine - to perform input/output (I/O)
- Message 1 routine (IGC5D07B) - to write "Unviewable Message" warning
- Processor 1, Load 1 (IGC5107B) - to continue processing

Display 3 Routine

The Display 3 routine (IGC6207B) receives control initially from Display 1 (IGC5207B) to dequeue, mark, and move messages into the message area of the display control module (DCM) from the console output queue.

Display 3 searches the console output queue for messages to be displayed. When one is found, Display 3 takes the following action according to the message type:

- In-line multiple-line WTOs - control passes to Status Display Interface 1 (IGC6L07B) to process these messages. Upon return from Status Display Interface 1, processing continues if other messages are on the queue. Otherwise, control passes to Display 2 (IGC5307B) to continue processing.
- Out-of-line multiple-line WTOs - these messages are not processed by this routine; if encountered the routine ignores them. Out-of-line multiple-line WTOs will eventually be processed by the Status Display Interface routines which receive control from the Processor routines.
- Single-line WTOs and WTORs - Display 3 places the message in the first available line in the screen image buffer (in the DCM), and marks the message according to the descriptor code provided by MCS.

After Display 3 has processed all messages in the queue, control passes to Display 2 for processing of split messages and for marking of replies.

Roll Mode Routine

The Roll Mode routine (IGC5J07B) performs the roll function when messages are on the WQE and the time specified for roll is satisfied. When messages on a display console are ready to be rolled, the routine rolls the messages in the screen image buffer. If the CONTROL command operand DEL is set to roll deletable (RD) mode, the specified number of deletable messages are removed. If DEL is set to roll (R) mode, the specified number of messages, including action messages, are removed. It also stores the information for inserting the number of message lines remaining on the output queue into the first two character positions of the first new message to be displayed following a roll. Control passes to the Timer Interpreter (IGC5K07B) unless DEL=RD and no deletable messages are on the visible portion of the screen, in which case control passes to the appropriate device I/O routine to write the message waiting message.

The RNUM value specified in the CONTROL S command determines the number of lines the Roll Mode routine removes, unless one of the following exceptions exists:

- Fewer lines on the output queue than RNUM value (less lines rolled)
- Number of lines displaced by a status display is smaller than RNUM value (less lines rolled)
- Roll Deletable (RD) mode specified and number of deletable lines on screen is smaller than RNUM value (less lines rolled)
- Top line on screen following roll is continuation line (one more line rolled)

Command Routine

The Command routine (IGC5407B) analyzes the commands in the entry area, processes CANCEL attentions, and processes requests to remove message numbers. According to the reason for entry, the routine takes the following action:

- CANCEL attention: the Command routine sets flags, alters the DCM as appropriate, and then passes control to the appropriate I/O routine.
- CONTROL command (with no other operands): the Command routine passes control to the Delete 3 routine (IGC5807B) to erase a segment of messages from the screen.
- Delete verification: the Command routine passes control to the Delete 4 routine (IGC5907B) to process the deletion verification.
- CONTROL commands: the Command routine issues an SVC 34 to pass the command to the system's command processing routines. When control returns, the Command routine determines if a parameter list was provided by the SVC 34 routines. If not, the Command routine blanks the entry area and passes control to the appropriate I/O routine. If a parameter list was provided, and the command in the entry area was CON-

TROL E,N, the Command routine erases the message numbers from the screen and passes control to the appropriate I/O routine. For all other commands, the Command routine passes control to the routine indicated in the SVC 34 parameter list.

- All other commands: the Command routine issues an SVC 35 to write the command to the message area of the console screen; it then issues an SVC 34 to pass the command to the system's command processing routines. When control returns from the SVC 34 routines, the Command routine determines if a parameter list was provided. If not, the Command routine blanks the entry area and passes control to the appropriate I/O routine. If a parameter list was provided, the Command routine passes control to the routine indicated in the parameter list.

Options Routine

The Options routine (IGC5A07B) receives control from the Command routine to analyze the minor operand values (DEL, CON, SEG, RNUM, RTME) of the CONTROL command specification (S) operand and to determine their legitimacy. If these minor operands and their respective values are legitimate, the Options routine changes the current values accordingly and indicates the screen options currently in effect. If these minor operands and/or their values are not legitimate, this routine exits to the appropriate device I/O routine with instructions to write the appropriate message from the Message routine. The Options routine also displays all the current screen option values when requested by the CONTROL S[,REF] command.

- CON=N and no hard copy
- DEL=R or RD and no hard copy
- DEL=R or RD and no timer

If a warning message is required, this routine sets the appropriate indicators and continues processing. If, however, an error message is required, the Options routine exits to Message 1 (IGC5D07B).

Upon preparing to exit, the Options routine indicates the proper location of the cursor. If a warning message is to be displayed, this routine transfers control to Message 1. If the value of DEL is changed to or from R or RD, or if the RTME operand has been changed, the Options routine sets an indicator bit in the DCM and exits to the Timer Interpreter routine (IGC5K07B). If a warning or error message is required, and a return to the Timer Interpreter routine is also indicated, the Options routine exits to the Timer Interpreter with instructions to pass control to Message 1 when finished.

Delete 1 Routine

The Delete 1 routine (IGC5607B) handles the erase commands, CONTROL E,F and CONTROL E,nn[,nn]. If conversational mode is in effect, Delete 1 updates the screen image buffer by marking and numbering those messages selected for deletion, and passes control to Message 1 (IGC5D07B) to write the "Deletion Requested" message. If nonconversational mode is in effect, Delete 1 updates the screen image buffer by marking and numbering those messages to be deleted, and passes control to Delete 4 (IGC5907B) for immediate deletion. Delete 1 exits to Message 2 (IGC5E07B) if the erase command is inconsistent, or if it has an invalid operand.

Delete 2 Routine

The Delete 2 routine (IGC5707B) handles the deletion of intervention required action messages and messages indicated for deletion by the DOM macro instruction. Delete 2 marks as automatically deletable those messages successfully acted upon or indicated for deletion by the DOM macro instruction, provided the display device is not busy or a delete request is pending. If the device is busy or a delete request is pending, Delete 2 passes control to the MCS Router routine (IEECMQWR). Delete 2 marks all intervention required action messages as automatically deletable and passes control to the Processor 1, Load 1 routine (IGC5107B) if a delete request is pending. Otherwise, Delete 2 passes control to either Delete 4 (IGC5907B) if automatic deletion is desired, or to the appropriate device I/O routine to write the message area when automatic message deletion is not specified.

Delete 3 Routine

The Delete 3 routine (IGC5807B) handles the erase command, CONTROL [E,SEG], and deletion by cursor or light pen. It updates the screen image buffer by marking and numbering the message area segment for deletion. If conversational mode is in effect, Delete 3 passes control to Message 1 (IGC5D07B) to write the appropriate message on the instruction line requesting operator verification of the messages marked for deletion. If nonconversational mode is in effect, Delete 3 passes control to Delete 4 (IGC5907B) for immediate deletion. If the value of SEG is equal to zero, or if there are no deletable messages within the message area segment, Delete 3

exits to Message 2 (IGC5E07B) to display the appropriate error message.

Delete 4 Routine

The Delete 4 routine (IGC5907B) examines the message indicators in the screen control table in the DCM and blanks those lines in the screen image buffer marked for automatic or regular deletion while it moves non-deletable lines up to the first available message line. This routine also handles the command, CONTROL D,N[,HOLD] by numbering all visible message lines.

If the deletion of messages is successful when a status display is not on the screen and a full screen condition exists, Delete 4 returns control to Display 1 (IGC5207B). Otherwise, Delete 4 exits to either Message 1 (IGC5D07B) to display a warning message informing the operator that unviewable messages are on the output queue if the screen is not yet full, or to the appropriate device I/O routine to display the "Message Waiting" warning message and write the entire screen.

Light Pen/Cursor Service Routine

The Light Pen/Cursor Service routine (IGC5F07B) handles light pen and cursor interruptions. If a light pen or cursor delete request occurs on the message line of a non-action message or on the asterisk of an action message, the routine transfers control to Delete 4 (IGC5907B) or Delete 3 (IGC5807B) depending on whether delete verification is or is not indicated. If a light pen detect or cursor placement is on *ENTER* in the instruction line, this routine passes control to the appropriate device I/O routine to perform a READ operation. When the light pen or cursor is positioned on *CANCEL* in the instruction line (to cancel a request) or on *E* in the status display title line (to erase a status display), the Light Pen/Cursor Service routine passes control to the Command routine (IGC5407B). When the light pen or cursor is positioned on *D C,K* in the instruction line or *F* in the status display title line, control passes to the Command routine. If the light pen is positioned on a key number in the PFK display line, control is passed to the PFK 1 routine (IGC6A07B). If the light pen or cursor is positioned on any other location than mentioned above, it is considered an error. Control passes to Message 1 (IGC5D07B) or Message 2 (IGC5E07B) to display the appropriate error message.

PFK 1 Routine

The PFK 1 routine (IGC6A07B) receives control from Processor 1, Load 1 (IGC5107B) when a PFK is pressed, or when a PFK key is to be verified or redefined. It also receives control from the Light Pen/Cursor routine (IGC5F07B) when a light pen interruption occurs for a number displayed in the PFK display line.

Upon entry, this routine determines the reason for entry. If entry has been made to cancel a PFK request, the commands associated with the canceled PFK key are removed from the entry area of the DCM. Also, control information is removed from the PFK area of the DCM so that later use of the same PFK key will cause new copies of the definitions to be used. If entry has been made to enter a command, the number of the key is placed in the DCMPFKNM field of the DCM, and the routine checks to see that the key is valid. If it is not, exit is made to Message 1 (IGC5D07B) so that an error message can be issued. Otherwise, a check is made to see if the key is already in process. A key may have several commands associated with it, each of which must be processed separately. If the key is not in process, it is flagged as in process now, and a check is made to see if the key is defined as a list of key numbers. If it is, control returns to the entry point of the routine so that each key number in the list can be processed separately.

When the routine determines that a key number is associated with a command to be processed, the command is moved to the entry area buffer in the DCM. If the key is in conversational mode, the command is written to the screen where the operator may cancel or enter it. If the key is not in conversational mode, the command is executed by simulating an operator ENTER action.

After all commands have been processed, the PFK work area is cleaned up and control is returned to the Processor 1, Load 1 routine.

PFK 2 Routine

The PFK 2 routine (IGC6B07B) receives control from the Processor 1 Load 1 (IGC5107B) routine when entry is made to redefine a PFK or to write or erase the PFK display line (the line of the display console screen containing PFK key numbers which may be entered by light selection).

The routine first determines which function was requested. If the request is to erase or display the PFK display line, exit is to the I/O routine for the device for which the display is requested. If the request is to redefine a PFK, the routine scans the new command and, if no errors are encountered, replaces the old definition

with the new. Control is then returned to the Processor 1, Load 1 routine.

Multiple-Line Write to Operator Routines (MCS)

Multiple-line Write to Operator (MLWTO) routines process multiple-line WTO requests in systems with multiple console support (MCS).

Multiple-Line Write to Operator, Load 1

Multiple-line Write to Operator, Load 1 (IGC0603E) is entered when the Write-to-Operator routine (IEEMVWTO) determines that a multiple-line WTO request has been entered. IGC0603E builds a major WQE for a MLWTO request.

Upon entry, IGC0603E checks if a major WQE already exists for the request. If so, the request is to build a minor WQE, and control passes to Load 2 (IGC0703E) to connect minor WQEs to the existing major WQE.

If the request is to build a major WQE, IGC0603E attempts to obtain WQE buffer space in main storage for the major WQE. If space is not available, and the MLWTO request is from the communications task, DAR, or an SIRB, an ENQ macro instruction and a WAIT macro instruction are issued upon the WQE ECB. When buffer space has been obtained, the text fields are filled in. Control then passes to Load 2.

Multiple-Line Write to Operator, Load 2

Multiple-line Write to Operator, Load 2 (IGC0703E) completes processing of the major WQE and begins processing of the minor WQEs. If entry is to complete a major WQE, the MCS and hard copy fields of the major WQE are filled in. If no minor WQEs are to be chained to the major WQE, control is passed to Load 3 (IGC0803E) to chain the major WQE to the system output queue.

If entry is made to connect minor WQEs to a major WQE, or if, upon completion of a major WQE, minor WQEs are ready to be connected to it, buffer space for the minor WQEs is obtained in the same way as for a major WQE. On subsequent entries to build minor WQEs, a check is made to determine if any minor WQEs previously connected to the major WQE have become available for re-use by having been written to all the consoles required by their routing indicators. If so, the buffer space obtained earlier is re-used for another minor WQE.

As each minor WQE is completed, control passes to Load 3 to chain the MLWTO request to the system output queue.

Multiple-Line Write to Operator, Load 3

Multiple-line Write to Operator, Load 3 (IGC0803E) is entered to chain MLWTO requests to the system output queue. If a major WQE is to be chained, the MLWTO identification number is placed in the appropriate areas of the major WQE.

If entry is to chain minor WQEs to the major WQE, the minor WQEs are chained and appropriate use count information is moved into them from the major WQE. When there are additional minor WQEs to be processed, control is returned to Load 2 (IGC0703E). Otherwise, control passes to Load 4 (IGC0903E).

Multiple-Line Write to Operator, Load 4

Multiple-line Write to Operator, Load 4 (IGC0903E) receives control from Load 3 (IGC0803E). IGC0903E dequeues the WQE from the WTO resource if necessary. It then sets the appropriate return code in register 15 and returns control to the routine that issued the MLWTO request.

Status Display Interface 1 Routine

The Status Display Interface 1 routine (IGC6L07B) receives control from the Processor 1, Load 1 routine (IGC5107B) or Display 3 (IGC6207B) to process multiple-line WTO messages that do not have descriptor codes 8 and 9. Descriptor codes 8 and 9 indicate that a display is a response to an operator's request and is to be presented out-of-line, in a display area of a display console.

Upon initial entry, IGC6L07B sets a flag in the UCM indicating that a status display is in progress. This flag insures that all of the lines of the status display will be displayed contiguously to prevent interleaving of other messages with the lines of the status display. The routine then moves lines of the WTO into the DCM until a sufficient number of lines have been passed to fill the screen or until the last line is encountered. If the screen becomes full before the last line is encountered, exit is made to the device dependent I/O routine to issue a MESSAGE WAITING message. When the last line is passed to the DCM, the WQE turns off the multiple-line WTO flag in the UCM and checks the WQE buffer for additional messages to be processed. If any are found, control passes to the Display 1 routine (IGC5207B) for further processing. Otherwise, control is passed to the I/O routine to display the messages that were moved to the DCM.

Status Display Interface 2 Routine

The Status Display Interface 2 routine (IGC6M07B) receives control from the Processor 1, Load 1 routine (IGC5107B) to process requests for multiple-line WTO messages that have descriptor codes 8 and 9. These codes indicate that the message is a status display requested by the operator, and that the display is to be presented out-of-line in a display area on a display console screen.

Upon initial entry, IGC6M07B searches the console output queue for a WQE representing a multiple-line WTO. When one is found, the routine locates in the DCM the screen area control block (SACB) for the display area in which the display appears. If the WQE represents a new display, the SACB is initialized, and any status display in the area is dequeued. If the display area is now empty (that is, it contains no other operator messages), control passes to the Status Display Interface 6 routine (IGC6Q07B), which puts the message into the screen image buffer (in the DCM) three lines at a time. If the display will overlay other operator messages, control passes to Status Display Interface 4, (IGC6007B) which sets up a write-area. This separate write-area will be used by the I/O routines to write the display to the screen. (This avoids using the DCM screen image buffer as a write-area, which would destroy the overlaid operator messages). If message area lines below the display area in use require blanking, control passes to the Status Display Interface 7 routine (IGC6T07B). If the display area lines contain non-status display messages, a three line write-area is constructed. The status display is then written to the display area from the write-area by the device dependent I/O routine, leaving the messages in the DCM buffer intact.

Status Display Interface 3 Routine

The Status Display Interface 3 routine (IGC6N07B) receives control from the Processor 1, Load 1 routine (IGC5107B) to process CONTROL commands affecting out-of-line status displays (K D,H; K D,U; K D,F).

If entry has been made for processing CONTROL commands, IGC6N07B takes action according to the operands of the command:

- K D,H -- an internal PM A is issued to stop the time interval updating of the display. The display remains in the screen but the routine rewrites the control line to contain frame and update options.
- K D,U -- an internal MN A is issued to continue updating the display. The control line is rewritten to contain stop and hold options.
- K D,F -- a control line containing the new frame number is built in the write-area and control passes to the I/O routine to write the new frame.

Status Display Interface 4 Routine

The Status Display Interface 4 routine (IGC6007B) receives control from the Status Display Interface 2 routine (IGC6M07B) to move lines of a status display from the WQE into a temporary write-area in the DCM. This routine is entered only for status displays that overlay other operator messages on a display console screen.

IGC6007B uses the instruction line and the two lines of the entry area as the temporary write-area. It moves three lines of the message into this temporary write-area, and then exits to the I/O routine to write the lines to the screen. When the last line of the display is put into the temporary work area, the routine blanks any remaining display area lines and puts the FRAME LAST indicator in the control line.

Status Display Interface 5 Routine

The Status Display Interface 5 routine (IGC6P07B) is entered from the Processor 1, Load 1 routine (IGC5107B) to process CONTROL commands affecting out-of-line status displays (K E,D; PM A).

Upon entry, IGC6P07B determines which command has been entered, and action is taken as follows:

- K E,D -- the appropriate major and minor WQEs are removed from the queue, and control is passed by means of the XCTL macro instruction to the I/O routine, which erases the display from the screen.
- PM A -- the dynamic display indicator in the DCM is turned off, the appropriate WQEs are removed from the queue, and control passes to the I/O routine, which erases the display from the screen.

IGC6P07B also receives control from Status Display Interface 2 (IGC6M07B) when blanking of lines below a display area is required. Control passes to Status Display Interface 7 (IGC6T07B) to perform the required blanking.

Status Display Interface 6 Routine

The Status Display Interface 6 routine (IGC6Q07B) is entered from the Status Display Interface 2 routine (IGC6M07B) to move

lines of a status display from the WQE into the appropriate message area lines of the screen image buffer (in the DCM). This routine is entered only if the status display does not overlay any other operator messages.

Upon entry, IGC6Q07B moves lines from the WQE into the screen image buffer until the display area is filled. When all lines of the display have been moved, any lines remaining in the display area are blanked, and the control line is rewritten to indicate FRAME LAST.

Status Display Interface 7 Routine

The Status Display Interface 7 routine (IGC6T07B) receives control from Status Display Interface 5 (IGC6P07B) to blank message area lines below a status display that is being displayed in a display area.

Upon entry, IGC6T07B determines the number of lines that require blanking and locates the first line to be blanked. It then fills the instruction line and the entry area (in the DCM) with blanks. After these lines are blanked, control passes to the device dependent I/O routine to blank the first three lines that require blanking. The I/O routine uses the three lines as a write-area to write blanks to the screen. This procedure prevents the message area from being broken up into blocks of general message traffic and blocks of status displays. It also allows any messages that were overlayed with blanks to remain untouched in the DCM screen image buffer. When the status display is erased, the screen is again written from the screen image buffer, and the messages reappear.

Cleanup Routine

The Cleanup routine (IGC6G07B) receives control from the Open/Close routine when a request for closing a device has been entered; it also receives control from the Asynchronous Error routine when a request to vary console status has been entered or when a device is recovering from an asynchronous error. This routine removes status displays from the message queues, and reinitializes the Screen Area Control Blocks (SACBs).

Upon entry, IGC6G07B determines the reason for entry and then determines if the console involved in the entry has a MONITOR Active display in "update" mode. If so, the display is stopped (STOPMN). The routine then searches the consoles SACBs for partially output status displays. If any are found, they are freed. The Cleanup routine then takes the following action according to the reason for entry:

- CLOSE -- the SACB configuration is reinitialized to the value specified during system generation, all messages are removed from the queue, and control passes to Processor 1 Load 1 (IGC5107B).
- Change in status to SD -- inline messages are freed, the SACBs are reinitialized to the values specified during system generation, and control passes to the Asynchronous Error routine (IGC5C07B).
- Change in status to MS -- the SACB configuration is reinitialized to the values specified during system generation, all SACBs are marked as unused, and control passes to the Asynchronous Error routine.
- Change in status to FC -- the SACB configuration is reinitialized to the values specified during system generation, and control passes to the Asynchronous Error routine.
- Asynchronous Error recovery -- out-of-line displays in progress and on the output queue are freed, the SACB configuration is retained, and control passes to the Asynchronous Error routine.

Timer Interpreter Routine

The Timer Interpreter routine (IGC5K07B) analyzes the timer intervals at which each of the display console devices specifying roll mode is scheduled to be rolled, and sets indicators to notify the Roll Mode routine (IGC5J07B) whenever the timer interval has elapsed for one or more of the display console devices.

The routine initially stores the timer interval (RTME if roll mode is specified, zero if not) for each display console device in its respective DCM. The greatest common divisor (GCD) of all the non-zero timer intervals is constantly updated as new devices are added as display consoles. The Timer Interpreter routine sets the timer equal to the GCD.

Whenever the timer elapses, this routine adds the GCD to the time counter of each of the display console devices whose timer interval is not equal to zero. If the new total equals or exceeds the timer interval for one or more display console devices, the Timer Interpreter routine notifies the Processor 1, Load 1 routine (IGC5107B) that the messages on each of these devices are ready to be rolled.

The Timer Interpreter gets control for one of three reasons. First, if control is

passed from the Processor routine, the timer has elapsed, in which case it returns to the Processor 1, Load 1 routine. Secondly, if the routine gets control from the Options routine, the Open-Close routine, or the Asynchronous Error routine, the GCD needs to be updated for a new set of intervals. Control is passed from the Options routine when a display device is put into or taken out of roll mode, or the value of RTME is changed, in which case an exit is taken to the appropriate device I/O routine or one of the message routines. Entry is from the Open/Close routine when a display device in roll mode is removed as an operator's console and control returns to the Open/Close routine (IGC5G07B). Entry is from the Asynchronous Error routine to set a single timer interval of 30 seconds for the removal of the error message and an exit is taken to Message 2 (IGC5E07B). Thirdly, the routine gets control from the Roll Mode routine to insert in the first two character positions of the first new message the number of message lines remaining on the output queue. The Timer Interpreter then exits to the appropriate device I/O routine if the warning message bit is on. Otherwise, the Timer Interpreter passes control to the Display 1 routine (IGC5207B).

SUPPORTING THE SYSTEM LOG

The system log, an optional feature for systems with MVT or Model 65 multiprocessing, consists of two data sets cataloged at system generation on which critical system information is recorded. The existence of two data sets allows one (the primary data set) to receive data from the system, problem programs and/or operators, while the other (the alternate data set) is being written to an output device.

The log is initialized by IEEVLIN1 (see Figure 7-10), which locates the log data sets and establishes the Log Control Area and log buffers, and IEEVLIN01, which writes the log JFCBs to the job queue, creates log DCBs, attaches the Log Writer routine (IEELWAIT), and posts the log ECB. If the log data sets are not located by IEEVLIN1, the operator is notified that the log option is not supported. Consequently, WTL macro instructions are re-issued as WTO macro instructions to the primary/master console. LOG and WRITELOG operator commands are treated as NOPs by the control program and a message is sent to the issuing console informing the operator that the system log is not supported. If log initialization is unsuccessful, IEEVLIN1 returns control to IEEVWAIT (or to the System Management Facility initialization routine if that feature is included in the system).

Users communicate with the log through the WTL macro instruction and the operator commands LOG and WRITELOG. The WTL routines (SVC 36) schedule the entering of information into the system log. The LOG command is used to enter information into the system log from an operator's console. The WRITELOG command is used to request that the currently recording log data set be closed and queued to a SYSOUT writer of a particular class. If no class name is specified with the WRITELOG command, the data set will be scheduled for queueing to a SYSOUT writer of the class specified at system generation. When the system log is performing the functions of the hard copy log (a log function of the multiple console support option), WTO and WTOR messages, as well as operator and system commands and their responses, may be entered on the system log by converting them to a WTL macro instruction. WTO and WTOR conversion is done in the MCS communications task module IEECMDSV. Conversion of the LOG command is done in the Command Scheduler (SVC 34) routines.

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

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

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

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

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

When a WRITELOG CLOSE command is entered, the process of writing the remain-

ing text in the log buffer to the data set is the same as mentioned above for a full buffer condition. Then the currently recording data set is closed and the log function is deleted from the system.

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

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

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

Anytime the operating system enters step initiation and a log data set has been scheduled for queueing, the step initiation module IEFSD162 passes control to the Log Dispatcher IEEVLDSP. IEEVLDSP enqueues the log data set to a system output queue of a particular class and issues a corresponding message to the operator.

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

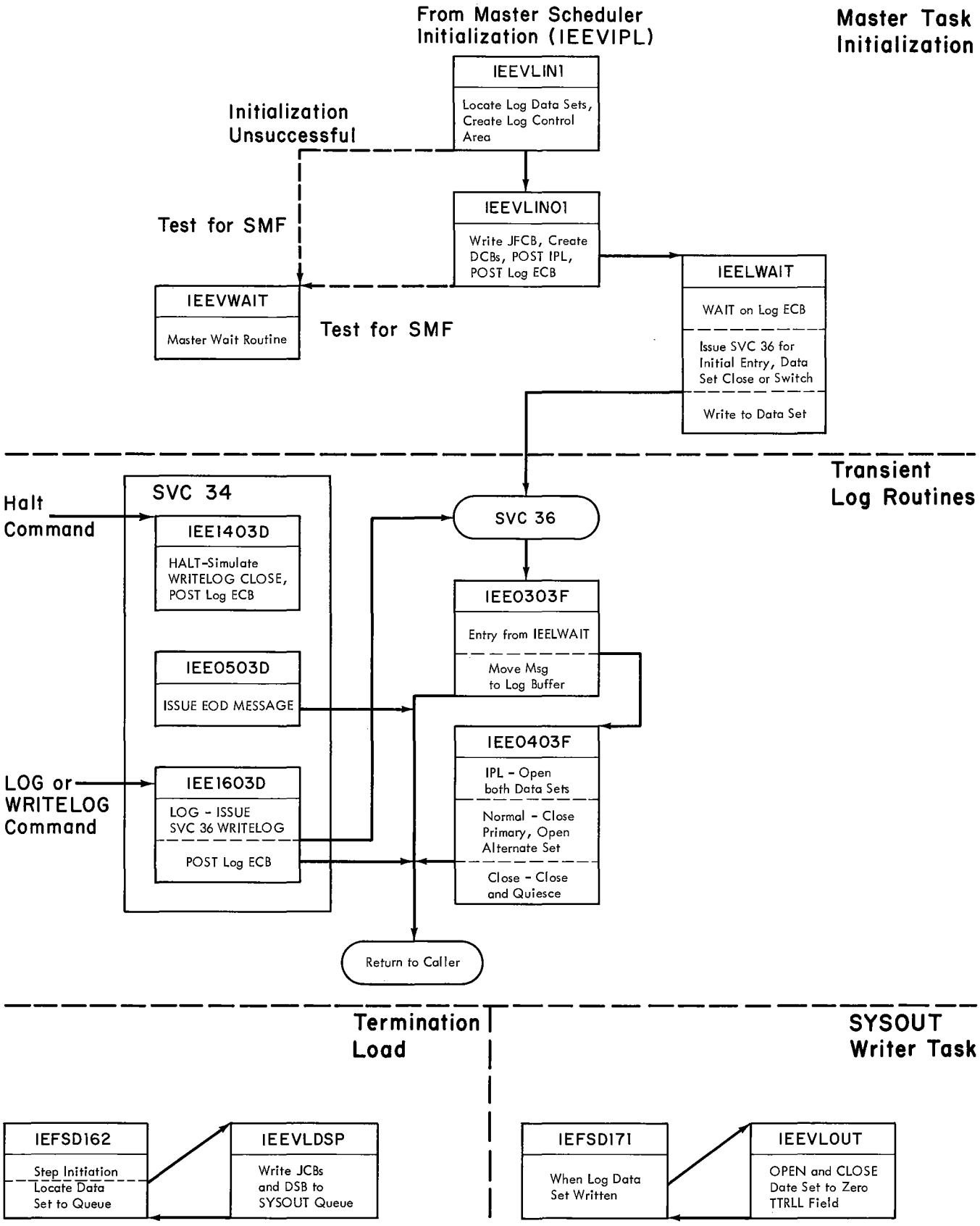


Figure 7-10. Log Function

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 MVT Job Management PLM.

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 flag 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 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.

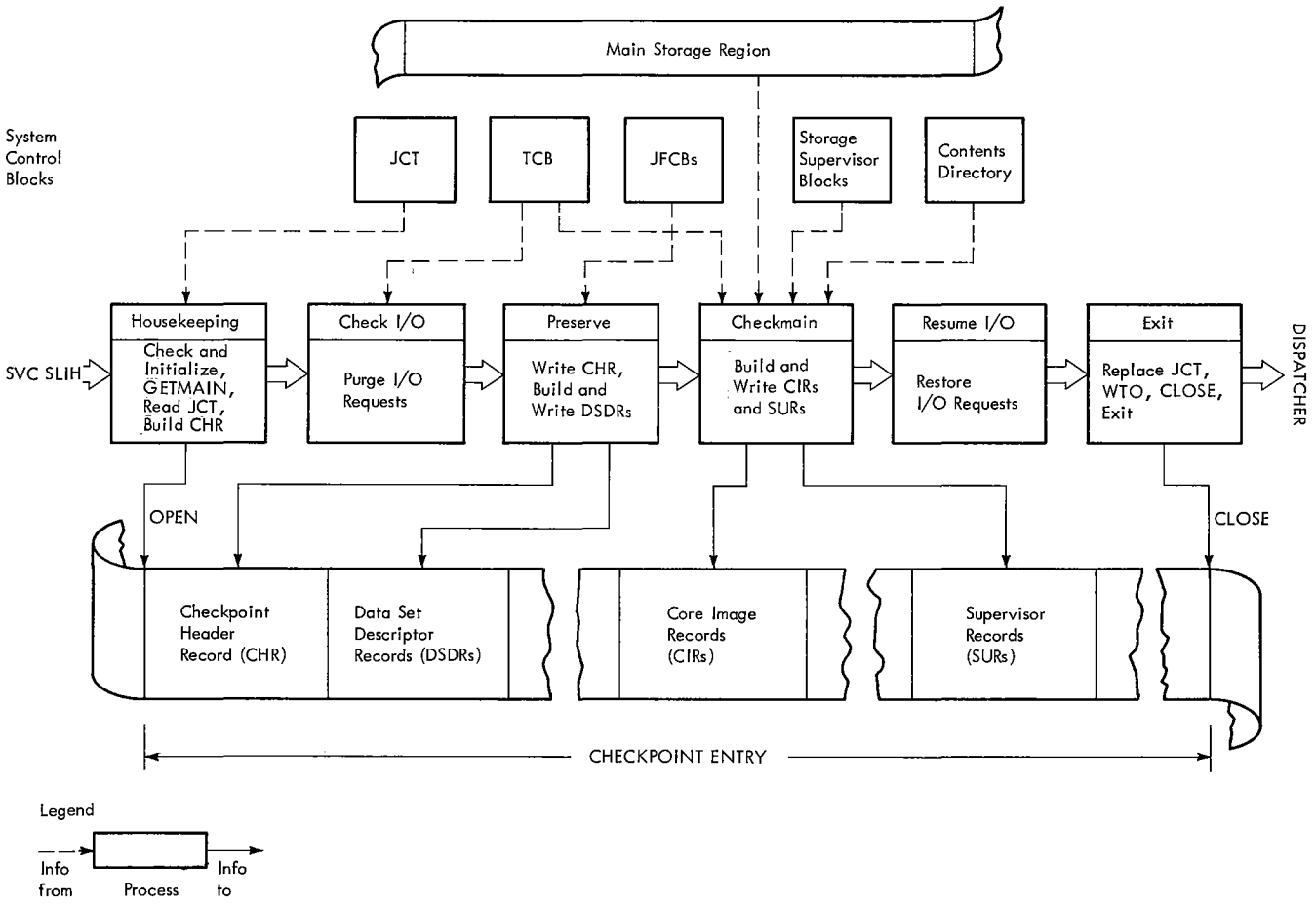


Figure 8-1. Checkpoint Processing Routines

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 is not 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 is issued, at completion of processing, informing the programmer it is 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 not want a checkpoint entry to be created,

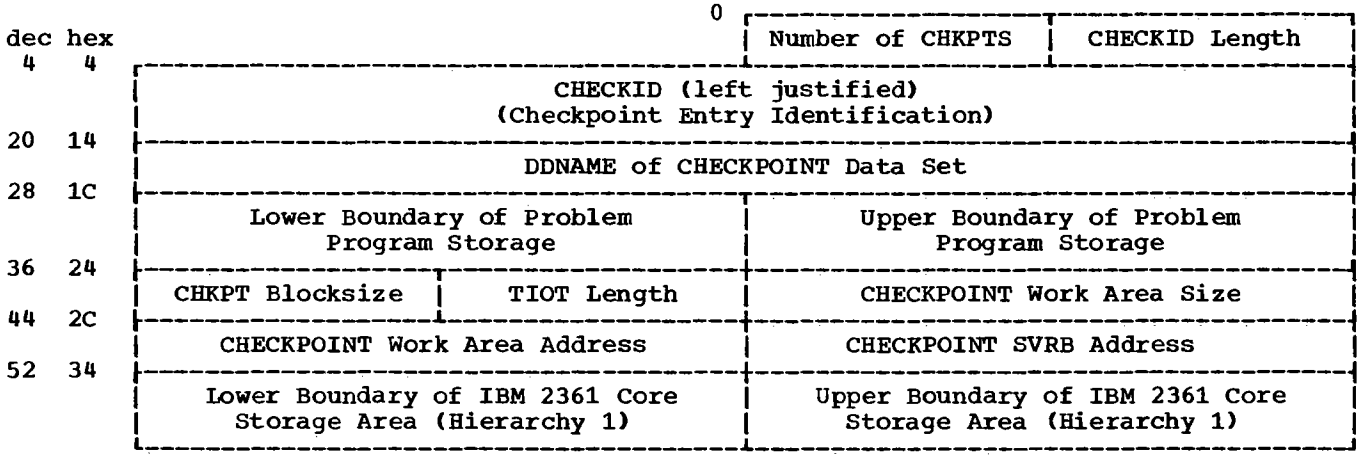


Figure 8-2. CHECKPOINT Header Record (CHR)

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 (IHJ000I) 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 8-3. 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 is not 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.

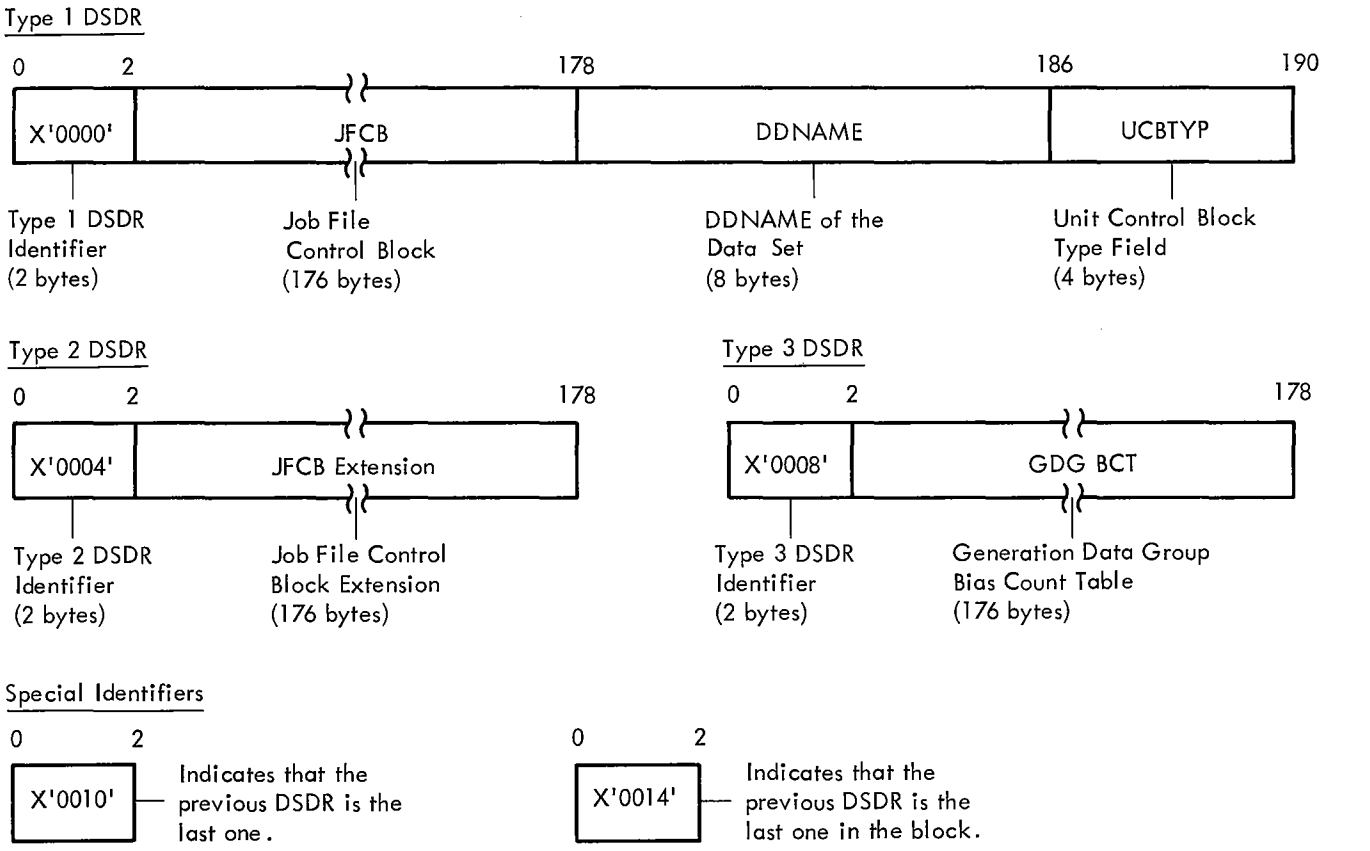


Figure 8-3. Data Set Descriptor Records (DSDRs)

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 is 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. The first load of the restart routine is used by job management routines. During the actual restart, it transfers control to the housekeeping routines. 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 8-4. The functions are listed below, with the names of the load modules implementing them:

- Obtaining and formatting storage (IGC0105B and IGC0205B). 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, IGC0G95B, IGC0I05B, and IGC0H05B). The JFCB processor interprets the JFCBs (already rebuilt by IEFDSDRP) and builds tables describing each open data set in the restart work area.
- TCAM Processing (IGC0J05B). The TCAM Data Set Processor initializes the TCAM region, sets up TCAM control blocks, and queues for each TCAM data set that is to be restarted.
- Mounting and verifying volumes (IGC0K05B and IGC0M05B). The Mount/Verify routine processes volume labels (calling a user label routine if necessary), and requests the operator to mount missing volumes.
- Processing SYSIN/SYSOUT non-DASD data sets (IGC0L05B). The SYSIN/SYSOUT Non-Direct Access Data Set Processor primes the buffers for a SYSIN data set from the card reader, or writes header labels on a SYSOUT tape data set with deferred restart.

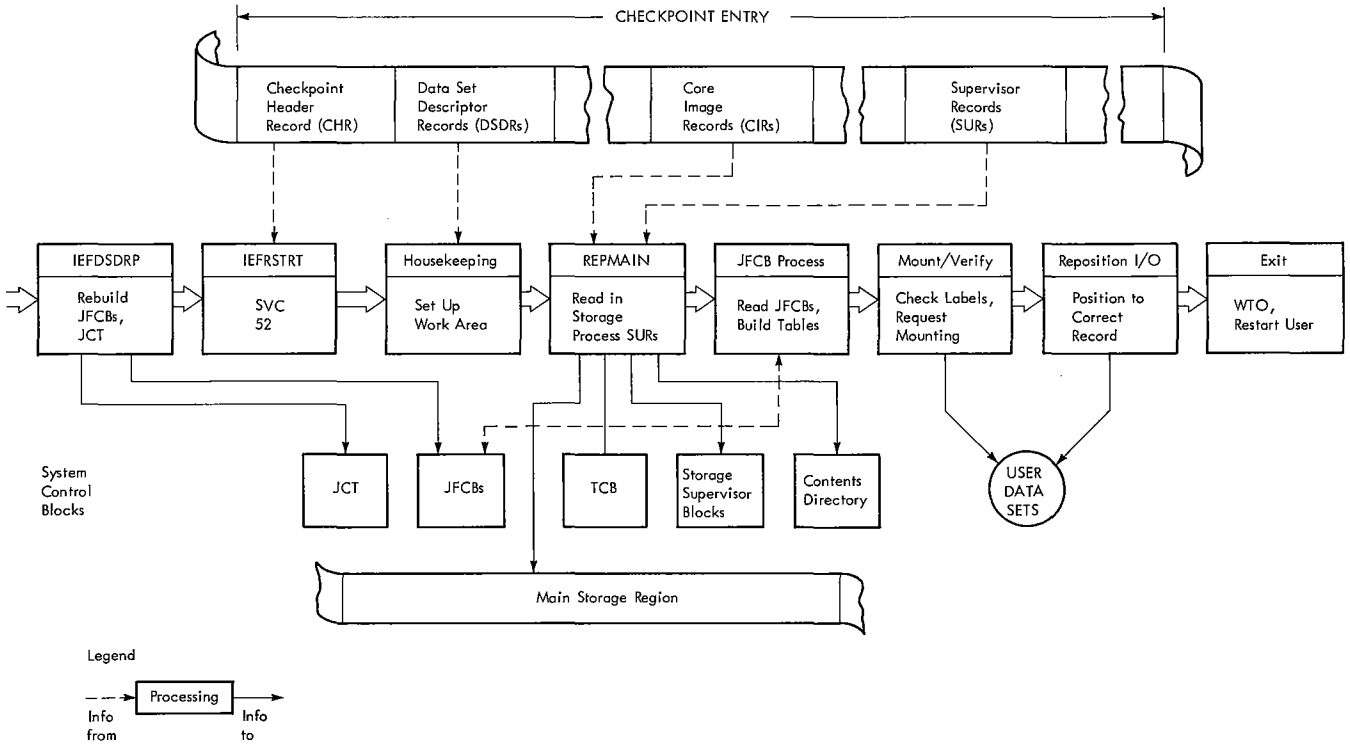


Figure 8-4. Restart Processing Routines

- Positioning open data sets (IGC0N05B, IGC0Q05B, IGC0P05B, IGC0R05B, IGC0S05B, and IGC0U05B). The Data Set Processor adjusts the problem program's data sets to the record being processed when CHKPT was issued.

- Processing ISAM or BDAM data sets (IGC0W05B). The ISAM and BDAM Data Set Processors prepare ISAM and BDAM data sets for restart processing.

- 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 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, IGC0105B and IGC0205B. 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 (IGC0105B)

The first load module of Restart Housekeeping 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 (IGC0205B)

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 passes 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 two modules of the JFCB Processing routine (IGC0G05B and IGC0G95B) build the tables and control blocks, the third (IGC-0I05B) makes adjustments for data sets residing on more than five volumes. If there are TCAM data sets, IGC0J05B is executed before IGC0I05B. IGC0H05B receives control if any data set description tables correspond to a dummy data set.

Table Build Module (IGC0G05B)

This JFCB Processing module assigns a 304-byte section within the restart work-area 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.

Table Build Module (IGC0G95B)

This JFCB processing module obtains the disk addresses 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. For non-TCAM data sets, 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.

For TCAM data sets, the queue name is saved in the associated data set description table and the TCAM flag is set to indicate that TCAM processing is required. If the TCAM flag is set, IGC0G95B passes control to the TCAM Data Set Processor (IGC0J05B). Otherwise, it passes control to IGC0I05B.

TCAM Data Set Processor Module (IGC0J05B)

If there are any TCAM data sets to be restarted, the TCAM data set processor receives control from IGC0G95B. It initializes the TCAM region and sets up TCAM control blocks and queues for each data set. Control passes to the Restart Error routine (IGC0V05B) if the TCAM message control program is not active in the system, the data set's QNAME parameter is not known to TCAM, the process entry defined by QNAME is already being used, or a GETMAIN by TCAM was unsuccessful. If IGC0J05B executes normally, control passes to the last JFCB Processor (IGC0I05B).

Table Complete Module (IGC0I05B)

The last 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 is 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.

Dummy Data Set Processor (IGC0H05B)

This module receives control from IGC-0I05B if any of the data set description tables correspond to a dummy data set. (For a discussion of dummy data sets, refer to Job Control Language Reference.)

For each dummy data set encountered, a check is made to determine if the data set was a dummy data set when the checkpoint was made. If it was, no further processing is done.

For each dummy data set that was not a dummy data set when the checkpoint was taken, IGC0H05B (1) deletes the subroutine loaded by the OPEN macro instruction, and (2) unchains and frees the DEB associated with the data set. If (1) the data set was being processed by the queued sequential or basic sequential access methods and (2) the checkpoint was not made during an end-of-volume exit for the data set, then IGC0H05B frees the IOBs created by the OPEN macro instruction, creates a dummy DEB and adds it to the DEB chain, loads the dummy access method (IGC019AV), and sets pointers to it in the DCB associated with the data set.

An error condition exists if (1) a data set that has been made a dummy is not being processed by either the basic sequential or the queued sequential access methods at the time the checkpoint was made or (2) the checkpoint was made during an end-of-volume exit. When an error occurs, an error code of 79 is placed in the restart work area.

If no errors are detected, control is passed to the mount verification portion of the Restart routine once all of the table entries have been processed.

If an error has been detected, control is passed to the Restart Error routine (IGC0V05B).

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 (IGC0K05B)

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 set's 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.

SYSIN/SYSOUT Non-Direct Access Data Set Processor (IGC0L05B)

This receives control from module IGC0K05B or IGC0M05B for certain SYSIN or SYSOUT data sets. It primes the buffers for a SYSIN data set from the card reader, or writes header labels on a SYSOUT tape data set with deferred restart. If an ASCII label is used for a SYSOUT tape or an error occurs in writing the SYSOUT header labels, control goes to the Restart Exit routine (IGC0V05B). Otherwise control goes to IGC0N05B (for SYSIN/SYSOUT data sets on direct access) or IGC0P05B.

POSITIONING OPEN DATA SETS

SYSIN/SYSOUT Data Set Processor 1 (IGC0N05B)

This module 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.

SYSIN/SYSOUT Data Set Processor 2 (Direct Access) (IGC0Q05B)

This module 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 Data Set Processor 1 (IGC0P05B) if there are any non-direct access data sets. Otherwise, control passes to Data Set Processor 2 (IGC0R05B).

Data Set Processor 1 (IGC0P05B)

IGC0P05B is the non-direct access table reformatting module. There is a table element entry in the Restart work area for each data set that is open when a CHKPT macro instruction was issued. There are at least two areas of 304 bytes each, located after the table elements, which contain a DEB, DCB, ECB, IOB, Channel Program, and JFCB. IGC0P05B counts the number of tape data sets, indicated by the table elements, and tries to reformat the 304-byte areas to contain 128 bytes for each tape data set (eliminating the JFCB). If more space is needed for 128-byte areas than is contained in the area occupied by the 304-byte areas, IGC0P05B issues a conditional GETMAIN macro instruction for additional storage. If this storage is not available, all tape data sets cannot be repositioned simultaneously. Control then passes to IGC0S05B.

Data Set Processor 1A (IGC0S05B)

This module works from the data set description tables, processing all magnetic tape data sets except those tapes created under DOS which contain either embedded checkpoint records or possibly a leading tapemark. On entry, all but two types of tape volumes are positioned at the load point. The exceptions are SYSIN data sets, which are positioned to read the first user input record, and non-standard labeled tapes, which are positioned at the first data record by the user label routine.

Data Set Processor 1A first advances the tape past the label, if necessary, to the correct data set, using the file sequence number. Then the DCB block count field (DCBBLKCT), saved at CHKPT, is used to advance the data set to the correct record. If the 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. IGC0S05B passes control to the DOS Tape Data Set Processor (IGC0U05B) if any data sets are open at CHKPT time for which either a leading tapemark or embedded DOS checkpoint record is indicated.

DOS Tape Data Set Processor (IGC0U05B)

This module is called to position tapes created under DOS which contain either a leading tapemark or embedded checkpoint record. Positioning of these tapes occurs similarly as in IGC0S05B with two exceptions:

1. For the indication of a leading tapemark, a read is issued and the resulting status is tested for indication of a unit exception. If a tapemark is not sensed, the tape is repositioned to the load point prior to performing record positioning.

2. To perform record positioning of data sets containing DOS embedded checkpoint records requires the reading of the first 20 bytes of every block. During record positioning, the embedded checkpoint records encountered are spaced over and are not reflected in the block count.

Data Set Processor 2 (IGC0R05B)

This module is called from IGC0S05B except when the above described DOS tape volumes require positioning. If either of these tape volumes requires positioning, Data Set Processor 2 is called from IGC0U05B. If there are no direct access data sets, the Access Method-Disposition module (IGC0T05B) gains control.

Data Set Processor 2 (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.

ISAM and BDAM Data Set Processor (IGC0W05B)

If there are any ISAM or BDAM data sets to be restarted, this module receives control from Data Set Processor 2 (IGC0R05B). For ISAM data sets, it reads the Format 2 DSCB and then transfers control to ISAM Open (IGG01920 or IGG01950) to validate certain pointers in the DSCB. If an I/O error occurs in the ISAM open module it sets a return code and IGC0W05B gives control to the Restart Exit routine (IGC0V05B). For BDAM data sets, IGC0W05B reinitializes addresses within reentrant BDAM access method modules. It passes control to the I/O Restart routine (IGC0T05B).

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 are started. Control then passes 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.

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 passes 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 MVT with Model 65 multiprocessing, the Type-1 Exit inspects the doubleword TCB pointers of both CPUs. If the TCB pointers for a CPU 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 passes control to External FLIH). In addition, the Dispatcher places zeros in the supervisor lock and CPU affinity bytes before returning control to the caller to show that neither CPU controls disabled Supervisor code 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.

Upon receiving control back from EOT, or if an End-of-task condition does not exist, 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 RB. 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 examines the STAE control block (SCB) queue pointed to by the TCBNSTAE field in the TCB. If there are no SCBs or if ABEND is in progress, the SCB processing is bypassed and Exit determines the RB type as described below. Exit removes from the queue and frees all SCBs for the exiting program except those SCBs for which the exiting program issued a STAE (Specify Task Abnormal Exit) macro instruction with the XCTL option. The search of the queue terminates when an SCB is found for an RB other than the RB for the exiting program. When the RB that is exiting is the last PRB (EOT condition), Exit frees all SCBs including STAI SCBs.

If there are no SVRBs (except possibly the exiting RB) queued to the TCB, the "prevent terminal attention exits" bit in the TCB is turned off.

The Exit routine then determines the type of program that is exiting by examining the RBSTAB field of its associated request block. This RB is always first on the RB queue when the Exit routine is entered. 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 contain the same values they had when the SVC was issued. Registers 15, 0, and 1 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.

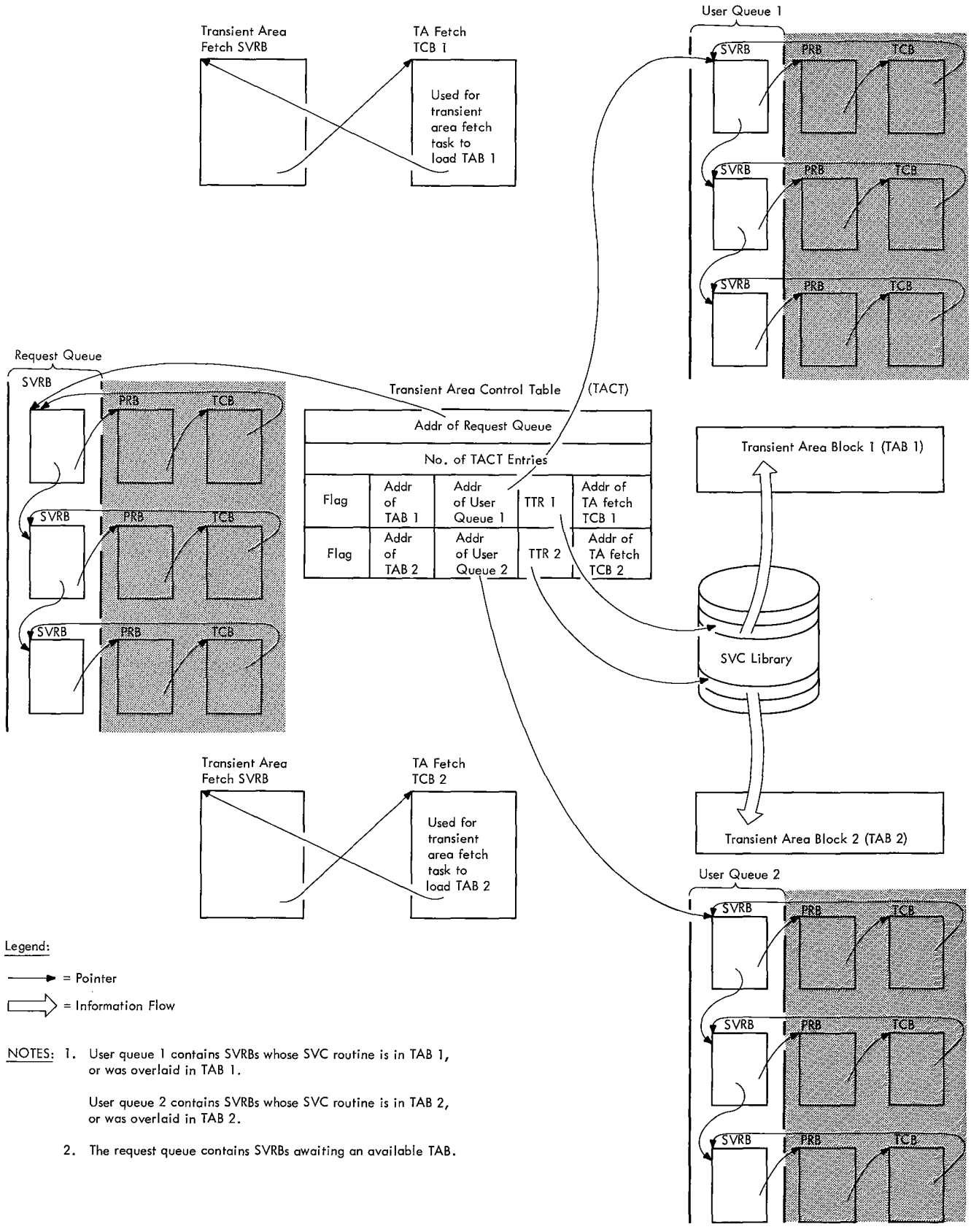
Before branching to the TAHEXIT subroutine, the RB queue is searched for SVRBs. If none are found, the TCBATT flag is reset to zero.

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 Dispatcher'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.



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

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 RBPGMQ field in the PRB), the CDEXIT routine updates the RB address in the CDE so it points to the next PRB that controls 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 tests 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 is 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 is later 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 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

---

[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.

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 program was controlled by an IRB, special processing is required.

When the time sharing option is included in the system, the Exit routine checks for an attention IRB. If one is found, the following special processing is performed by subroutine IKJATTNX:

- Start all subtasks via a branch entry to IEAQSETS.
- If this is not an exit from ABEND or a STAE exit routine, use the terminal attention interruption element (TAIE) to set up registers and resume PSW.
- Free the TAIE.

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 requester 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

Unless the Exit routine has branched to the dispatcher, control is passed from the RB-dependent processing to common Exit processing at label EDTNX. If the exiting program is represented by the last RB on the RB queue, the Exit routine removes the current TCB from the TCB queue because it is no longer needed. The Exit routine also sets the termination flag (TCBFE) which is later used by the Detach routine. This avoids an incorrect branch to ABTERM when the subtask is eventually detached. Exit then forces a task switch as described below.

If the TCBSTPPR flag is set, indicating that a task is to be set nondispatchable when no longer executing a privileged program, the TCBATT flag is not set, and the task is not abnormally terminating, Exit clears the TCBSTPPR flag and begins a search of the RB queue. This search is terminated when an SVRB or an RB with a protection key of zero is found. If an attention RB is found or if the entire queue is searched without finding an attention RB, the TCB is set nondispatchable and the stop flag in the TCB (set by the STATUS macro instruction) is set to zero.

If the next RB on the queue has the wait bit on and the "new" pointer contains the address of the current TCB, "new" is set to zero to force a task switch when Exit passes control to the dispatcher. The Exit routine then sets the RB for the exiting program inactive and removes it from the RB queue and frees, if possible, the RB area. If the exiting RB is an IRB with a problem program save area, the save area is also freed. The Exit routine then branches directly to the Transient Area Refresh routine.

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 asso-

ciated 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 for 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.
- ABTERM, when entered from a type-1 SVC which was entered by a branch entry.
- 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.
- ABEND11, when it has selected another terminating task whose resources are to be purged.
- DAR4, when the job-step region has been set nondispatchable.

In MVT with Model 65 multiprocessing, 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.

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. If the emulator option and the Model 85 were specified at system generation, the Dispatcher checks the TCB (bit 3 in the TCBTRN field) to see if the old task is the 7094 emulator program. If it is, the Dispatcher issues a Diagnose instruction to save emulator status and leave emulator mode. The Dispatcher then 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.
- ABTERM.
- SVC Second-Level Interruption Handler.
- External First-Level Interruption Handler.
- I/O First-Level Interruption Handler.
- ABEND11.
- DAR4.

The Dispatcher then determines the next task which receives 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 is 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. If the emulator option and the Model 85 were specified at system generation, the Dispatcher checks the TCB (bit 3 in the TCBTRN field) to see if the new task is the 7094 emulator program. If it is, the Dispatcher issues a Diagnose instruction to restore emulator status and enter emulator mode. 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 MVT with Model 65 multiprocessing, 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 (that is, the RBWCF field must contain zero).

2. The nondispatchability flags in the TCB must not be set (see Table 2 in "Termination Procedures", Section 10).

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 MVT with Model 65 multiprocessing, 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 both CPUs 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.

If the system includes the System Management Facility (SMF), the Dispatcher records the beginning of a system wait before loading the RB old PSW to dispatch the pseudo task. It reads the interval timer and stores its value in the first word of a special save area, SYSWSAVE. The value is later saved by the SMF Wait Time Collection routine and used to calculate elapsed wait time.

In MVT with Model 65 multiprocessing, 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 affinity 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 the

requested 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.

When time-slicing is included in the Model 65 Multiprocessing System, extensions to the logic of both MVT and MVT with Model 65 multiprocessing must be made. A basic change to MVT time-slicing logic is needed so that two time-sliced tasks of either the same or different dispatching priorities can run simultaneously. The logic for MVT with Model 65 multiprocessing is changed in only one way--time-slicing logic is added. A fundamental modification of MVT with Model 65 multiprocessing logic is the restriction on the definition of the "new" TCB pointers in terms of queue position in the following cases:

1. If the highest-priority ready task is the only ready task of its TCB dispatching priority (TCBDSP), and the next highest-priority task is in a lower-priority time-slice group (TSG), the next highest-priority task is not "new" on the second CPU if it is not the current, the next-to-run, or the only ready member of that TSG.

2. If the two highest-priority tasks are members of the same TSG, both tasks become the two "new" tasks if they are the two current, the next-to-run members, or the only ready members of the TSG.

These restrictions are important because, if either "new" task on both CPUs is time-sliced but is not running currently or is not entitled to resume (does not satisfy the conditions that "new = old" and the TSTQE is on the queue), that task is replaced by the next eligible task of the same TCBDSP. If both "new" TCB pointers are not current members of the same TSG, a search is made for the next two eligible tasks, the first of which becomes "new_1" (unless it happens to be "old_2", which indicates that the current task on the other CPU is entitled to resume).

The search loop for TCBDSPs considers the existence of "old_2" (the current task on the other CPU) and the possibility that either "new" may have been determined already.

A shoulder-tap is issued if "new_2 = old_2" but $TSTQE_2$ is off the queue (indicating that the task on the other CPU has lost its turn). No shoulder-tap is issued if "new_2" and "old_2" are members of the same TSG, and $TSTQE_2$ is on the queue (indicating that "old_2" is entitled to continue). In this case, "new_2" is set equal to "old_2".

TSO Processing: Prior to comparing the "new" and "old" pointers, the dispatcher passes control to the time sharing dispatcher to re-order the ready queue if required. Control returns to the test of the "new" and "old" pointers.

The time sharing dispatcher also receives control after a task switch has occurred so that the time sharing driver can provide its monitoring function with appropriate data.

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 Dis-

---

[1]The type of interval request is indicated in the TQEFLGS field of the task's timer queue element.

patcher 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: a task switch is needed, and the 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.

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.[1]

The Timer Second-Level Interruption Handler 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. 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.

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:

- 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. Ifthe TQE is REAL and on the timer queue, it must be removed from the

---

[1]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.

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.

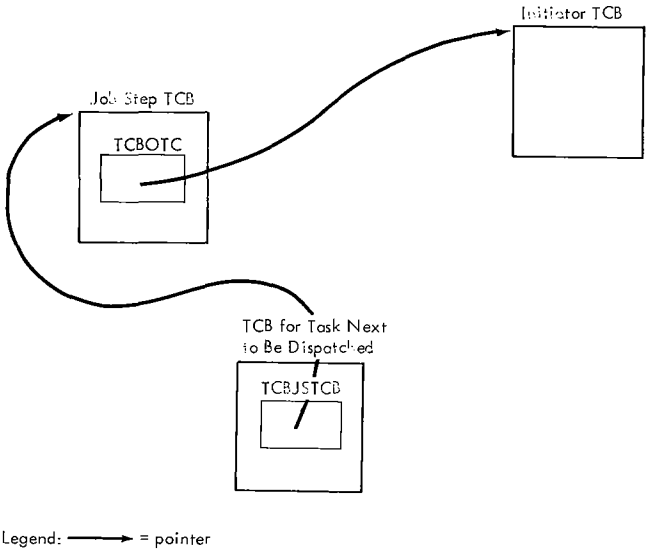


Figure 9-2. Locating the Initiator TCB Associated with the Task Next to be Dispatched

In MVT with Model 65 multiprocessing, when two tasks of the same job step are running simultaneously, the time-to-expiration value of the job is halved. Therefore, when job-step timing is suspended for a task, the Dispatcher must determine if the task on both CPUs belong 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 is 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.

In MVT with Model 65 multiprocessing, 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 other 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 simultaneously.

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 when it detects 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.

Note: If the time sharing option is included in the system, terminal attention exit elements (TAXES) are purged for the terminating TCB.

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 is provided to its parent task in the return code register. The parent task examines 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.

If the terminating task is a time sharing task, the Purge Timer subroutine uses the Purge TAXE routine (IEAKJXP) to remove terminal attention exit elements.

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.

If the EOT routine was not entered from ABEND, a branch is made to the "WTOR Purge" routine (IEECVED2). If the terminating task is not a time sharing task, IEECVED2 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 telling the operator to cancel outstanding replies.

If the terminating task is a time sharing task that is not in main storage, the "WTOR Purge" routine does not purge the reply queue elements and write queue elements.

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.

After closing data sets, a check is made to determine if the terminating task has a parent (originating) task which is abnormally terminating. If the parent task is abnormally terminating, the ABEND bit (TCBFA) is on. Because the parent task purges both itself and any subtasks, there is no need to go to the EOT routine for the terminating task. Instead, the task is set nondispatchable, and the parent task continues abnormal processing.

If there is no error detected during the closing of data sets, and there is no terminating parent task, 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 requester. (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. If a time sharing task is the requester, the program is freed unconditionally. (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. 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 completes the ETXR scheduling. It places 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.

If the time sharing option is included in the system, the time sharing EOT is entered. If the logon task is terminating and the user is to be assigned to a new region, the logon flags in the TJB and TSCVT are set, the TSC relogon flag is set, the TSEVENT macro instruction is issued for logoff, and the TSC ECB is posted. If the logon task is terminating and the user is to be disconnected, the disconnect flags in the TJB and TSCVT are set, the TSC disconnect flag is set, the TSEVENT macro instruction is issued for logoff, and the TSC ECB is posted. If the terminating task is not logon, the TCB is removed from the TCB queue and the terminal job block extension (TJBX) is updated.

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 zeros 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 does 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 is not 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, removes the TCB from its parent's subtask queue and frees 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 branches to the Dispatcher which gives control to the current routine of the highest priority ready task.

ABNORMAL TERMINATION

Abnormal termination is implemented primarily by four supervisor routines: the ABTERM routine, the ABEND routine, the Damage Assessment Routine (DAR), 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 <u>all</u> 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.
- All tasks in the job step are in a 30-minute wait.
- The job-step timer interval has expired.
- SYSOUT data exceeded the limit specified by the OUTLIM parameter of the associated DD statement.
- The Machine-Check Handler[1] is unable to recover from a machine check that occurs during the job step, but determines that the failure is not permanent.

---

[1]The Machine-Check Handler is optional system generation programming support for the System/360 Model 65 (MCH/65), and standard programming support for the Model 65 Multiprocessor, the Model 85 (MCH/85), and the System/370 Models 145 (MCH/145), 155 (MCH/155), and 165 (MCH/165). Refer to Section 2: "Interruption Handling."

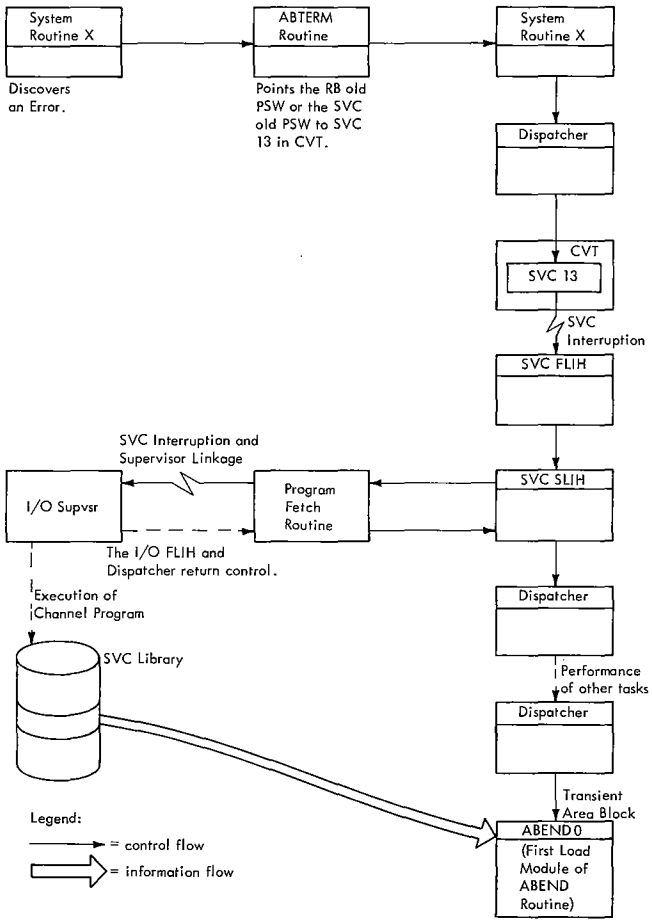


Figure 10-1. Scheduling of the ABEND Routine by the ABTERM Routine

The Damage Assessment Routine (DAR) receives control from various ABEND modules. DAR attempts to write a dump of main storage, and reinstates, or initiates reinstatement of a failing task. DAR informs the operator if reinstatement is impossible so that the operator may halt system processing.

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.

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:

- Refreshes the CVT address.
- 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.
- If the time sharing option is included in the system, a time sharing task in terminal I/O wait will be set dispatchable.
- 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 Model 65 Multiprocessing System, determines, through a branch to the Task Removal routine, whether the cur-

rent 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 the SVC FLIH and IBM type-1 SVC routines; the other (IEA0AB00) is for all other system routines that wish to schedule an abnormal termination.

For entry at IEA0AB01, the ABTERM routine assumes that the address of the TCB to be scheduled for termination is contained in register 4 and the system completion code is contained in the low order 12 bits of register 1. The dump option flag is added to the completion code passed. The dump option flag indicates that ABEND must invoke the ABDUMP routine during ABEND processing.

The ABTERM routine, when entered at IEA0AB00, first refreshes the CVT address in case it has been overlaid during prior processing. Then it saves the caller's register contents and obtains and validity checks the TCB address. If the TCB address is invalid (the TCB is not one of the TCBs on the priority queue), control is passed to entry point DISMISS in the I/O FLIH. At DISMISS, the I/O original entry switch (IORGSW) is set to zero and a branch is made to the Dispatcher.

If the TCB address is valid, the ABTERM routine interrogates flags to determine if the specified task should be scheduled for ABEND processing, and/or if its subtasks should be set nondispatchable.

If the time sharing option is included in the system, a task that is nondispatchable due to terminal I/O wait is set dispatchable. Various combinations of ABTERM processing are possible, depending on the condition of the task specified for termination. The following discussion describes each condition of the specified task and the resultant processing, as outlined in Figure 10-2.

Processing if Specified Task Has Already Been Terminated

(See Figure 10-2, 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 Figure 10-2, 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 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 10-3.) 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.")

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" and "open in process" 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.	

Figure 10-2. ABTERM Processing

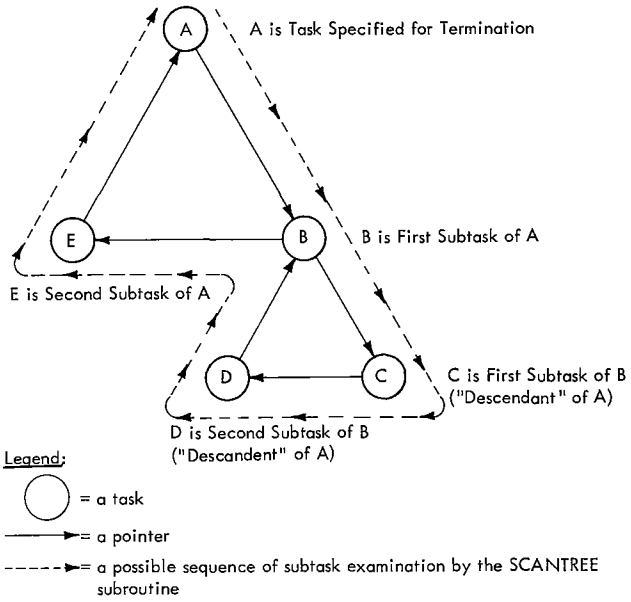


Figure 10-3. A Tree of Subtasks and a Possible Sequence of Examination

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 tests the nondispatchability flags and does 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 Figure 10-2, 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 a 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 is 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 Figure 10-2, 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" and "open in progress" nondispatchability flags (TCBSYS and TCBSTP). The Dispatcher later examines 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 Figure 10-4.)

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 to the parent task, via the ABEND routine. (The parent of the job-step task is the Initiator.)

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	32.6	This task is nondispatchable while the Open routine is being executed for another 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.
TCBNDISP	33.7	This task is nondispatchable. See bytes 173, 174 and 175 of the TCB for the cause.

Figure 10-4. The TCB Nondispatchability Flags

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 fetches the ABEND routine from auxiliary storage (if it is not already in a transient area of main storage) and passes 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 first considers 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 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 Figure 10-2, 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. If the recursion is valid, the ABEND routine continues the termination procedures. If, however, the recursion is invalid, the ABEND routine XCTLs to the Damage Assessment routine, which averts a CPU wait state by abnormally terminating only the failing task and its subtasks and by permitting the system to continue operating.

Because 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 Figure 10-2, condition 3c.) There are several possible causes of an ABEND request by the Initiator while the job-step task is already being abnormally terminated:

- The job-step timer expired for this task.
- The operator issued a CANCEL command for this task.
- All tasks in the job step are in a 30-minute wait (completion code 522).
- SYSOUT data exceeded the limit specified by the OUTLIM parameter of the associated DD statement (completion code 722).

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. 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 (indicated by TCB flags), entry to the ABEND routine is urgent. In order to stop the dump, the ABTERM routine gives 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. After clearing the flags, the ABTERM routine clears all nondispatchability flags, except the "must complete", "open in progress", and Recovery Management Support (RMS) nondispatchability flags, 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, 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 Figure 10-2, condition 4a.) In this case, entry to the 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 Figure 10-2, 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 Figure 10-2, 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 six main functions:

- Refreshes the CVT address.
- 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.
- Places the address of the Dispatcher in the return register.
- 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 refreshes the CVT address at absolute location 16 (decimal) in case it has been overlaid during prior processing. Then it 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), or if the program check occurred in SVC 0 (EXCP) or in SVC 15 (ERREXCP), the Prologue routine passes control to the IOS Program Interruption Handler routine (IECCPL00). If the program check did not occur in the I/O Supervisor or in SVC 0 or SVC 15, the Prologue routine obtains the TCB address from the "current" TCB pointer (IEATCBP+4).

After the determination of the TCB address, there are two streams of processing, depending on the source of the program check: a system or user program, or a type-1 SVC routine (or the SVC FLIH). The source of the program check is determined by a test of 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 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 are eventually 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, for example, 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 returns control when its processing is complete.

If the program check has occurred in a type-1 SVC routine other the SVC 0 or SVC 15 (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 stores 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 returns control.

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 causes 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 10-5).

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 belonging to the terminating task, its subtasks, and its direct ancestors, and dynamically acquired storage and programs belonging to the terminating task.

In systems with Main Storage Hierarchy Support, ABDUMP dumps main storage in each hierarchy associated with the terminating 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 first load of SVC DUMP. Only those modules of the dump routines whose functions are requested are then fetched and executed.

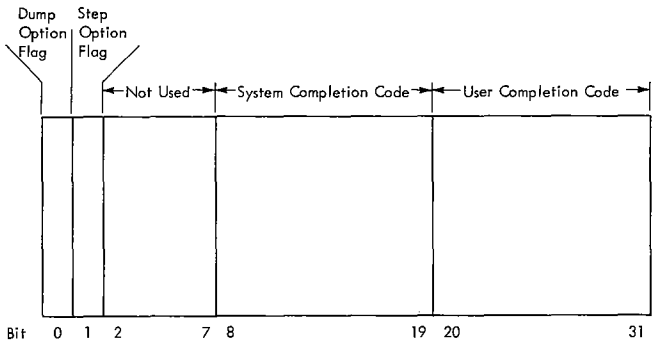


Figure 10-5. Format of the Completion Code and the Dump Option Flag in the Parameter Register

The ABDUMP routine consists of nonresident modules that are separately fetched and executed, and one "resident" module that remains in main storage for the entire dump procedure. The multiprocessing ABDUMP routine includes one additional nonresident 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, SVC DUMP 1, determines if an SDUMP macro instruction had been issued. If not, control is passed to ABDUMP 1. Otherwise, SVC DUMP 1 sets all non-system tasks nondispatchable, obtains the date and time of day and places these in the header record, and performs the dump to a direct access device. If the dump data set resides on a tape device, control is passed to SVC DUMP 2 to perform the dump. After the selected areas are dumped, SVC DUMP sets all tasks dispatchable and issues an SVC 3 instruction.

ABDUMP1 ensures that a dump data set has been opened for BSAM, provides a work area for use by the entire routine, loads the so-called "resident" module (format and output routines), and conditionally gets storage space for preserving the trace table and blocking the records.

ABDUMP 1.5 displays the identification code (if specified), job name, step name, time, and date. If the ABEND routine is the caller, ABDUMP 1.5 displays the completion code from the specified task's TCB. If the old PSW is requested as an operand of the SNAP macro instruction, it is also displayed.

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, and their extent lists (one for each major CDE).

ABDUMPQ formats and displays 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).

If the dump request is for TCAM or a time sharing task, TCAM ABDUMP 1-4 for TCAM or ABDUMP H-I for a time sharing task are processed next.

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.

ABDUMP11, executed in a multiprocessing system, displays the prefixed storage area in the nucleus. If the multi-system mode is operating, the prefixed storage area at upper main storage is also displayed.

ABDUMP12, executed only in a uniprocessing system, formats and displays the GTF trace data if GTF is active, and if the internal storage and formatting options were selected. If GTF tracing is displayed, the optional supervisor trace table display is bypassed.

ABDUMP13, executed only in a multiprocessing system, performs the same functions as ABDUMP12.

ABDUMP14 formats and displays GTF control and error records. ABDUMP14 receives control from, and returns control to ABDUMP12 (ABDUMP13 in a multiprocessing system).

ABDUMP15, executed only in a uniprocessing system, formats and displays the optional supervisor trace table if requested.

ABDUMP16, executed only in a multiprocessing system, formats and displays the optional supervisor trace table if requested.

ABDUMPH formats and displays the protected storage control block (PSCB), the user profile table (UPT), and the data set extension (DSE) for a dump request of a time sharing task.

ABDUMPI formats and displays the terminal job block (TJB) and the terminal job block extension (TJBX) for a dump request of a time sharing task.

TCAM ABDUMP1 formats and prints the header line, address vector table (AVT), and the basic section of the terminal name table (TNT) for Telecommunications Access Method (TCAM) Message Control Programs (MCPs).

TCAM ABDUMP2 dumps the entries in the TNT and their associated terminal table (TRM) entries.

TCAM ABDUMP3 dumps the TCAM destination queue control blocks associated with the TRM entries.

TCAM ABDUMP4 dumps open TCAM DCBs and the line control blocks associated with each line group DCB.

Processing during ABDUMP1
(Entry Point IGC0L05A)

After having been fetched by the first load of SVC DUMP, 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 been opened, 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 is later 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 frees 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 occupies 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 determines if GTF is active and if the GTF trace data is to be formatted. If these conditions are met, GTF tracing is suspended to ensure the validity of the GTF trace data. ABDUMP1 then issues an ENQ macro instruction for the dump data set. (The ABEND routine suspends GTF tracing and issues its own ENQ macro instruction for the dump data set.) It does this to prevent a program belonging to a 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.

If GTF is active, processing for the optional supervisor trace table is bypassed. However, if GTF is inactive, and 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 ABDUMP15. 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 ABDUMP15, the trace table will not be displayed.

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 invokes ABDUMP1.5 by issuing an XCTL macro instruction.

Processing during ABDUMP1.5 (Entry Point IGC0C05A)

ABDUMP1.5 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, ABDUMP1.5 displays the completion code from the TCB for the specified task. Then this module 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. 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 passes control to the selected module of the ABDUMP routine.

Processing during ABDUMP2 (Entry Point IGC0105A)

ABDUMP2 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 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.

Processing during ABDUMP3 (Entry Point IGC0205A)

ABDUMP3 displays the contents directory entries for the task, and their extent lists (one for each major CDE). 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, ABDUMP3 processing is complete. ABDUMP3 invokes ABDUMPQ via an XCTL macro instruction.

Processing during ABDUMPQ (Entry Point IGC0Q05A)

ABDUMPQ formats and displays the data extent blocks (DEBs) chained from the TCBDEB field, and the task I/O table (TIOT). DEBs chained from the OLTEP TCB are not formatted or displayed. When processing is complete, ABDUMPQ invokes ABDUMP4 via an XCTL macro instruction.

Processing during ABDUMP4 (Entry Point IGC0305A)

ABDUMP4 displays all 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:

For the Specified Task:
Subpool queue elements (SPQEs).
Descriptor queue elements (DQEs).
Free queue elements (FQEs).

For the Job Step's Region(s):
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 is active, control passes 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.

Processing during 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.

In addition to 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 message 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 is now described in greater detail. (See Figure 10-6.)

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 10-6, 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).

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 10-6, 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 10-6, part 2, block B).
- If the backward chain pointer is valid, it displays the save area and its address (Figure 10-6, 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 10-6, 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 10-6, part 6, block X). This save area address and the associated backward chain pointer (Figure 10-6, part 4, block B) are validity checked, and the two save areas and their addresses are displayed (Figure 10-6, 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).

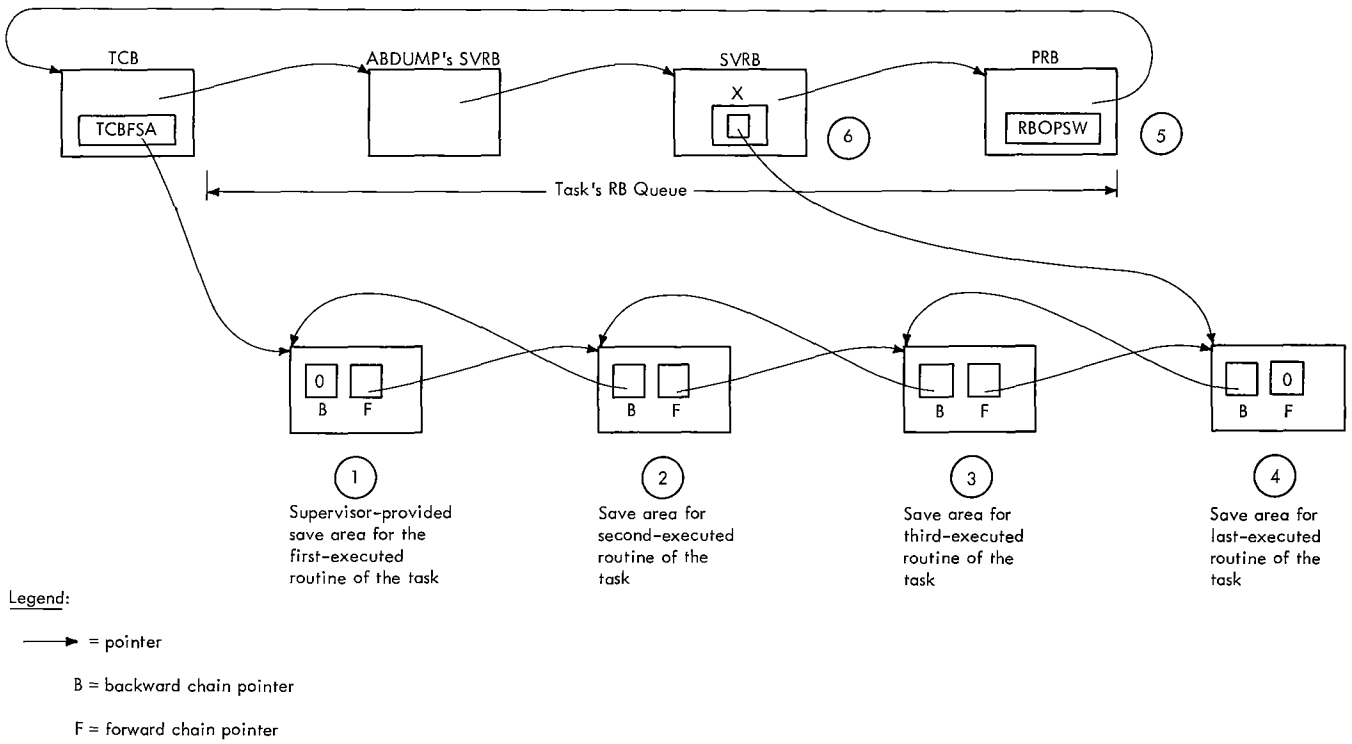


Figure 10-6. Pointers Used During the Save Area Trace

Processing during ABDUMP11 (Entry Point IGC0B05A)

ABDUMP11 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." ABDUMP11 invokes ABDUMP7 via an XCTL macro instruction.

In a multiprocessing system, ABDUMP7 omits the prefixed storage area from the display of the nucleus, and ABDUMP15 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 is displayed by ABDUMP15.

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 rou-

tine 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 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.

ABDUMP9 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 dispatchable. 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.

Processing during ABDUMP12 (Entry Point IGC0J05A)

ABDUMP12 displays the GTF trace data if it exists in main storage and is requested as part of the dump. A header record is printed first. Then, the oldest record in the GTF trace buffer is obtained and the data formatted and printed. Formatting proceeds from the oldest record to the most current. Control is transferred to ABDUMP14 (IGC0M05A) when a control record (time records and lost-event-count records) or error records are encountered.

When all records have been formatted and printed, GTF tracing is resumed and control is transferred to the ABDUMP resident module (IEAQAD0A).

Processing during ABDUMP13 (Entry Point IGC0K05A)

ABDUMP13 displays the GTF trace data in a multiprocessing system if it exists in main storage and is requested as part of the dump. The function of this module is very similar to ABDUMP12.

Processing during ABDUMP14 (Entry Point IGC0M05A)

ABDUMP14 formats and prints GTF control and error records. The control records consist of the time record (created if the keyword TIME=YES was used in the START command) and the lost-event-count record. When an error record is encountered, ABDUMP14 selects the appropriate message and dumps the error record in hexadecimal.

Control is returned to ABDUMP 12 or ABDUMP 14.

Processing during ABDUMP15 (Entry Point IGC0N05A)

ABDUMP15 displays the optional supervisor trace table if it exists in the system and was requested as part of the dump, and if the table was saved during ABDUMP1. In a multiprocessing system, the trace table is displayed by ABDUMP16 (IGC0P05A) and, therefore, is not displayed by ABDUMP15.

ABDUMP15 displays the trace table in two parts. (The program listing calls this procedure "unfolding" the trace table.) ABDUMP15 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, ABDUMP15 frees the space previously obtained for the table and then branches to the resident module (IEAQAD0A).

Processing during ABDUMP16 (Entry Point IGC0P05A)

ABDUMP16 is executed only in a multiprocessing system. It displays the trace table if it exists in the system, was requested as part of the dump, and was saved during ABDUMP1. The character A or B is printed on each line/entry to identify the CPU to which the line/entry applies. The trace table is displayed as in a uniprocessing system (see Processing During ABDUMP15).

Processing during ABDUMPH (Entry Point IGC0H05A)

If the dump request is for a time sharing task, ABDUMPH formats and displays the protected storage control block (PSCB), the user profile table (UPT), and the data set extension (DSE). ABDUMPH then passes control to ABDUMP9.

Processing during ABDUMPI (Entry Point IGC0I05A)

If the dump request is for a time sharing task, ABDUMPI formats and displays the terminal job block (TJB) and the terminal job block extension (TJBX). ABDUMPI then passes control to the Where-to-Go routine.

TCAM ABDUMP1 (Entry Point IGC0D05A)

TCAM ABDUMP1 receives control from the resident ABDUMP module (IEAQAD0A) when the program being dumped is the Telecommunications Access Method Message Control Program. TCAM ABDUMP1 formats and prints the header line, the address vector table (AVT), and the basic section of the terminal name table (TNT). The AVT is dumped in three parts -- the basic section, the storage queue section, and the disk section. Then the TNT section is dumped and control is passed to TCAM ABDUMP2.

TCAM ABDUMP2 (Entry Point IGC0E05A)

TCAM ABDUMP2 dumps the entries in the TNT and their associated terminal table (TRM) entries. First, the TNT entry is formatted and immediately after, its associated TRM entry is also formatted. This is repeated for every TNT entry. Control is then passed to TCAM ABDUMP3.

TCAM ABDUMP3 (Entry Point IGC0F05A)

TCAM ABDUMP3 dumps the TCAM destination queue control blocks associated with the TRM entries. They are located through the TNT entries, and are formatted in the order of ascending storage locations. After all the QCBs have been dumped, control is passed to TCAM ABDUMP4.

TCAM ABDUMP4 (Entry Point IGC0G05A)

TCAM ABDUMP4 dumps the three types of TCAM DCBs (line group, message queue, and checkpoint) and the line control blocks associated with each line group DCB. Only open TCAM DCBs are dumped. The order in which they are dumped depends on the order in which their corresponding DEBs are chained. The line control blocks for each line group DCB are formatted immediately after their associated DCBs. When all DCBs have been printed, control is transferred to the "Where-to-Go" routine in the resident ABDUMP routine.

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 (that is, 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."

SVC DUMP (Entry Point IGC0005A)

SVC DUMP provides a dump of selected areas of main storage to either tape or a direct access device. When invoked by DAR or SVC 34, it writes the dump of main storage to the SYS1.DUMP data set. When invoked by any other supervisor routine, the dump is written to a user defined data set.

SVC DUMP 1 (IEAQAD0Y) is entered at entry point IGC0005A when the caller issues an SVC 51. Initially, SVC DUMP 1 checks the ABDUMP parameter list to determine if the request is for a SNAP or an SVC DUMP. If the request is for a SNAP, control passes to ABDUMP1 (IEAQAD00). Otherwise, SVC DUMP1 checks the highest level RB of the invoking task to ensure it is operating with a protection key of zero, that is, to ensure that it is in supervisor state. If it is not, the dump exits with a return code.

The lock byte is then tested to determine whether a dump is already in progress. If it is, the dump exits with a return code. If it is not, the lock byte is set to prevent simultaneous access to dump data

sets. Next all tasks, except the Communications, Fetch, System Error, and current tasks, are set nondispatchable and the dump invoked bit in the TCB is turned on.

If the caller does not supply a DCB address, the SYS1.DUMP data set is used. If the data set, user-supplied or SYS1.DUMP, is not open, SVC Dump 1 exits with a return code.

SVC DUMP 1 then issues the TIME DEC macro instruction to obtain the date and time. These are stored in the header record.

The UCB representing the device upon which the dump data set resides is checked to ensure that it is in the "ready" state. If it is not, a return code is set and the routine exits after resetting dispatchability bits and the lock byte. The device type, either tape or direct access, is then determined and initialization of the control blocks is completed as required.

The optional trace table (if present), or GTF (if active), are made inoperative for the duration of the dump so that entries leading up to the failure are preserved. Prior to exiting, the supervisor trace table or GTF is reactivated.

If the device is direct access, a test is required to determine if the data set is available to receive a dump of main storage. A channel program is initialized to READ the first record on the data set. If EOF is not detected, the data set already contains a dump of main storage, in which case a return code is set and the routine exits. Otherwise, the data set is empty and available to receive a dump of main storage. The user supplied header record is the first data block written. After this the channel programs are reinitialized to perform the actual writing of the dump of main storage.

Abnormal end, channel end, and program controlled interrupt appendages are provided. The writing of the dump is performed via EXCP and WAIT. The PCI appendage performs the updating of the channel programs required to write the dump. The channel end and abnormal end appendages are provided to restart the channel program if end of cylinder or I/O errors are encountered. Standard ERPs are used while writing the dump except if SVC DUMP was invoked due to a failure of the System Error Task.

An unrecoverable error condition causes the writing of the dump to be terminated and a return code set prior to exiting. Upon completion of the dump, a normal completion return code is set. For SYS1.DUMP an EOF record is written after the dump is completed (normally or abnormally). For user DASD data sets, the DCBFDAD field in the user's DCB is filled in with the address of the last block written. The DCBTRBAL field is set to zero if the last record written filled a track. Otherwise, it is set to 1024. These steps are required to present a standard DCB interface to close. All tasks are set dispatchable prior to exiting, the lock byte is reset, and the "dump invoked" bit is turned off. Control then returns to the caller.

If the device is tape, control passes to SVC DUMP 2 (IEAQAD0Z) at entry point IGC0Z-05A. When writing to tape, processing is basically the same as direct access with the exception that the PCI and CE appendages are not used. A tapemark is written only for a NIP allocated tape unit. If the tape is user-supplied, no tapemark is written. The user must close the DCB. The "LEAVE" option may be specified thus allowing multiple dumps on tape.

SVC DUMP 2 checks for unit exception after data is written to tape. If a unit exception occurs, the dump is terminated for lack of space. In this case, a tapemark is written and the tape unloaded. A special return code is provided. If a channel data check occurs, the dump is continued at the next storage address to be displayed.

If dump storage boundaries are indicated in the parameter list, the dump routine dumps storage from the addresses specified. Beginning addresses are rounded down to a 2K boundary; ending addresses are rounded up to a 2K boundary. If the parameter lists indicate that nucleus and SQA are to be dumped, this is done first. All parameters are validity checked. If invalid, the dump is terminated with an error return code.

Upon completion of the last WRITE, SVC DUMP 2 returns to the caller after setting all tasks dispatchable, turning off the lock byte and dump invoked bit, and reactivating the optional trace table or GTF tracing.

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, SCBs, 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. 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. In task termination the resources of only the malfunctioning task and its previously unterminated descendants are released. 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 internal ABEND processing in the case of a "steal core" or a DAR (Damage Assessment Routine) conversion.

The termination procedure is performed by the ABEND routine. As stated before, the ABEND routine frees resources and control blocks belonging to the malfunctioning task and its previously unterminated descendants (subtasks, and subtasks of subtasks). For several unusual conditions (a task in "must complete" status, a terminating system task, or an invalid recursion), the ABEND routine exits to the Damage Assessment Routine (DAR), which alters the environment so ABEND or system processing can attempt to continue, or sets all tasks in the job step nondispatachable.

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.
- A 0F3 machine check for some task in the job step.

Recursions

An error condition may occur during ABEND processing that results in 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 ABEND can try, if possible, to complete termination procedures. Such a reentry to the ABEND routine for the same task is called a recursion. Certain recursions are valid (ABEND expects the possibility of an ABEND at that point), and the module continues normal processing with any special handling necessary. Invalid recursions, when detected, are handled by passing control to DAR which takes a dump and attempts to continue the termination if possible.

The task control block (TCB) contains a TCBRECDE field which specifies configurations for valid ABEND recursions. This field is checked for the particular type of recursion, and ABEND processing continues based on the configurations. ABEND3 checks this field and routes control to the proper modules to handle it. The valid recursion configuration flags are set prior to any processing during which a possible ABEND is

expected. They are then cleared immediately after completion of that particular processing. This is done so that ABEND does not misinterpret a future invalid recursion as valid. Following is a list of the TCBRECDE flags and the valid recursions they indicate. The TCBREC flag indicates that a valid recursion exists; this bit is checked in ABEND0 and ABEND1. For a first-time entry this bit is zero; for a valid recursion this bit is one. For a valid recursion, ABEND1 turns this bit off and processing continues based on the status of the right-most bits of the TCBRECDE field. These remaining bits, shown in Figure 10-7, indicate the specific recursion.

Communications

The TCBRECDE field is also used for parameter type information in communicating between various modules. When used for communications, the TCBREC bit is always zero. The following is a list of the communication configuration flags used between ABEND/ASIR/DAR modules:

TCBNOSTA -- Indicates that STAE/STAI (Specify Task Abnormal Exit/Subtask ABEND Intercept) not to be honored.
TCBSTRET -- Indicates a return from the "steal core" routine.
TCBCONVR -- Indicates a conversion to a job-step ABEND is necessary.
TCBDARET -- Indicates a return from DAR.
TCBNEWRB -- Indicates that ABEND issued an SVC 13 to pass control, via an XCTL, to a non-ABEND module.

Issuing A Conditional Freemain

Because certain ABEND modules must not be interrupted by other abnormal termination conditions, and to bypass excessive recursion processing, special linkage is provided within several ABEND modules and ABTERM to handle error conditions that may occur within a FREEMAIN. An ABEND condition may result during a FREEMAIN because of the following two cases: (1) the user may have freed up system control blocks in his storage area; the task abnormally terminates when it attempts to free these same control blocks; or (2) the user may have destroyed a free queue element (FQE) near the area being freed; FREEMAIN detects the invalid FQE when ABEND attempts to free the FQE.

The ABEND modules set the first byte in the SCVTFMSA field to X'80' before invoking a conditional FREEMAIN across a branch entry. The branch itself prohibits ABEND from losing control to the SVC FLIH and the Dispatcher. If an error condition occurs

Bit Field	Recursion Resulting from an:
TCBOPEN	Error in the Open routine for the dump data set.
TCBCLOSD	Error in the Close routine for Direct SYSOUT on tape.
TCBCLOSE	Error in the Close routine for open data sets.
TCBCLOSF	Error in the Force Close routine for graphics jobs.
TCBGREC	Error in the Graphics Debug routine.
TCBADUMP	Error in the ABDUMP routine.
TCBPTAXE	Error in the Purge TAXE routine.
TCBDYNAM	Error in the TIOT Check routine for dynamic DD entries.
TCBQTIP	Error during the purge of TSO Inter-Partition Posts.
TCBTCAMP	Error during the purge of TCAM Posts.
TCBTCAMR	Error in the TCAM Message Control Program (MCP) Reinitialization routine.
TCBSAVCD	Error in the ABEND/STAE Interface routines (ASIR).
TCBTYP1W	Error in the Write-to-Programmer routine for type-1 messages.

Figure 10-7. Valid ABEND Recursion Configurations

within FREEMAIN, ABTERM is entered to schedule abnormal termination of the task. However, when ABTERM encounters the X'80' in the first byte of SCVTFSMA, normal ABTERM processing is bypassed, and ABTERM loads registers 2 - 14 from the FREEMAIN save area, and returns control to the ABEND module using a BR 14 instruction. ABEND then resets the SCVTFSMA field, and processing continues even though the specified area was not freed.

Often the ABEND routine must branch to subroutines within the Exit/End-of-Task routine to fulfill certain necessary functions. If these subroutines may free areas in unprotected storage, ABEND sets the SCVTFMSA field to X'40'. This indicates to the subroutine that, because the entry is a branch entry from ABEND, a conditional FREEMAIN should be performed.

ABEND Processing

ABEND processing follows a logical, functional sequence. Certain actions must precede others. The following is a general list of the types of functions included in ABEND, arranged in the order in which they should occur. If a new function were to be added which did not fit into any of these ten classes, a new class of functions would be incorporated into its proper place.

1. Functions that must be performed prior to deciding whether the ABEND should continue.

2. Functions that must be performed for a real ABEND prior to losing control via an XCTL.

3. Functions that seriously degrade the performance of other tasks until they are performed.

4. Functions that must be performed prior to routing control to DAR.

5. Functions that must be performed (a) for all non-DAR entries, (b) prior to providing diagnostic aids to the problem programmer, and (c) while the originally terminating task is in control.

6. Functions that must be performed only once per ABEND, regardless of recursions, and must be performed prior to providing diagnostic aids.

7. Functions that provide diagnostic aids to the problem programmer (for example, a SYSUDUMP/SYSABEND dump).

8. Functions that must be performed under control of each terminating task.

9. Functions that may be performed under control of the top terminating task's TCB.

10. Final processing for the top task that must be performed at a time in which ABEND will not lose control.

ABEND Modules

The ABEND routine is composed of several transient modules. Some modules are entered only once per ABEND, and some on every recursion. All modules, except ABEND16 and ABEND20, must issue an XCTL macro instruction at the end of processing to cause supervisor linkage to the Transient Area Handler which fetches and passes control to the next module.

Following is a list of the ABEND modules and a brief description of their functions.

ABEND0 gains control after an SVC 13 instruction is issued either by the caller, or indirectly by the ABTERM routine. It handles initial interfaces for SVC DUMP, Generalized Trace Facility, STAE/STAI, TSO, and ABDUMP. ABEND0 also purges I/O for the current task, prevents asynchronous exits, performs ABEND/ABTERM housekeeping functions, and routes control to DAR for invalid recursions.

ABEND1 purges I/O and AEQs for each entry into the ABEND routine. Timer and WTOR requests are purged for first-time entries only. ABEND1 performs conversion to a job-step ABEND, sets subtasks nondispatchable, validity checks the FQE, releases the PIE, and routes serious errors to DAR.

ABEND3 releases partially loaded programs, purges type-1 message list elements (on recursion), and routes control for valid recursions.

ABEND 4 writes type-1 messages, purges message list elements, and frees type-1 message WTP buffers.

ABEND5 purges Rollout requests.

ABEND7 provides an interface with the Graphics Debug routine, and the Direct SYSOUT writer.

ABEND8 opens the dump data set, and saves and restores both the load list element (LLE) pointer, and the current TCBMSS pointer (pointer to SPQEs for the current task).

ABEND9 processing is concerned mainly with providing diagnostic aids. It loads ABDUMP's resident module, enqueues on the dump data set, and gives an ABDUMP via SNAP (SVC 51). After completing these functions, ABEND9 dequeues on the dump data set and deletes the ABDUMP module.

ABEND11 provides the following data management interfaces and related purges -- erases complete subtasks, moves the ABEND SVRB around the terminating tree and dispatches under each TCB, checks for storage for the Close routine and routes to the "steal core" routine, closes open data sets, and frees the PIE.

ABEND12 performs the "steal core" functions for the Close routine.

ABEND13 performs data management and subsystem interfaces and purges. It reinitializes some queues for the TCAM task, cancels pending Inter-Partition Posts for user and TSO tasks, and frees SCBs. ABEND13 also purges QELs, transient SVRBs, and routes for TIOT cleanup.

ABEND15 purges CDEs, associated programs, and extent lists.

ABEND16 purges IRBs, PRBs, resident SVRBs, the load list, and main storage. It also dequeues and erases subtask TCBs. ABEND16 then returns control to the current routine of the highest priority ready task by exiting to the Normal Termination rou-

tine and the Dispatcher (via an SVC 3 instruction).

ABEND20 performs Storage Reconfiguration in a Model 65 Multiprocessing System.

Processing during ABEND0 (Entry Point IGC0001C)

The SVC SLIH fetches the first module of the ABEND routine (ABEND0) from the SVC library. The SVC SLIH then gives control to ABEND0, via the Dispatcher.

ABEND0 handles the interfaces between ABEND and the following functions:

- SVC DUMP
- GTF (Generalized Trace Facility)
- STAE/STAI (Specify Task Abnormal Exit/ Subtask ABEND Intercept)
- TSO (Time Sharing Option)
- ABDUMP

ABEND0 first tests if this entry to ABEND is a return from a STAI exit routine through ASIR1 (ABEND/STAE Interface Routine 1). If so, ABEND0 sets the TCBFX flag to prevent asynchronous exits, and turns off the ABEND communication configuration indicator. Processing continues with the Purge I/O routine.

For a normal entry to ABEND, the ABEND SVRB extended save area is zeroed out and 'ABEND' is moved to its end. ABEND0 then sets the TCBFX flag, preventing asynchronous exits.

If the TCBNEWRB flag is on, a new SVRB has been created by an SVC 13 issued by ABEND. This new SVRB is used to give control to a non-ABEND module via XCTL. When the non-ABEND module is finished, it exits and the previous ABEND module gains control immediately following the SVC it issued. The name of the module to get control is in bytes 8-15 of the previous ABEND's SVRB extended save area; the recursion configuration to be used is in byte 16.

ABEND0 next determines if this is a nonrecursive entry from ABTERM. If so, the RB old PSW is restored from the field in which it was saved by ABTERM. If this entry is not through ABTERM and is nonrecursive, the completion code in register 1 is stored into the terminating task's TCB. At this time diagnostic information is also saved in the SVRB.

ABEND0 determines if SVC DUMP is in progress. If so, all tasks that were set nondispatchable by SVC DUMP are set dispatchable and the Task Switch routine (IEA0DS02) is entered to switch tasks if necessary. The CVT lock bit is set off and GTF (if active) or the trace table (if present) is reactivated. If GTF was started, the stop count is reduced.

For a first-time entry into ABEND, ABEND0 issues the HOOK macro instruction to allow the Generalized Trace Facility (GTF) to determine if it is being terminated. If it is, it performs cleanup to prevent any resulting system damage.

STAE/STAI INTERFACE: ABEND0 bypasses ASIR processing performed if any of the following conditions exist:

- An ABEND recursion condition exists.
- In a Model 65 Multiprocessing System, entry to ABEND was caused by the Machine-Check Handler of Recovery Management (indicated by TCBNSTAE field) to logically remove failing main storage (Storage Reconfiguration).
- No STAE environment exists.
- The job-step timer expired for this task (except for the OLTEP task[1]).
- The operator issued a CANCEL command for this task (except for the OLTEP task).
- A STAE recursion exists, and there is no STAE control block for a STAI request.
- A DETACH macro instruction was issued for a subtask that had not completed processing (completion code 13E). This is not applicable for DETACH with STAE= YES (completion code 33E).
- All tasks in the job step are in a 30-minute wait (completion code 522).
- SYSOUT data exceeded the limit specified by the OUTLIM parameter of the associated DD statement (completion code 722).
- Control was returned to ABEND0 from ASIR2, specifying that no further STAI exits should be honored.
- This task was previously terminating abnormally (close failure from a subtask).

If any of these conditons exist, processing continues with the Purge I/O routine.

---

[1]Information on OLTEP tasks may be found in the publication Online Test Executive Program.

If the STAE address is present, bit one of the address field is tested for a Purge I/O with the QUIESCE option. If bit one is on, all SVRBs existing between the current SVRB and the Purge SVRB are forced through the Exit routine to remove them from their RB queues. The exit is accomplished by storing the address of SVC 3 in the right-half of the RBOPSW location of the SVRB. The Purge SVRB, which is also forced through the Exit routine, points to the previous ABEND's SVRB (distinquished by the word 'ABEND0' in the extended save area). By removing all intervening SVRBs, control is returned to the RB level at which ASIR invoked Purge. ASIR then detects that the Purge routine has abnormally terminated, and takes appropriate action.

If a STAE or STAI environment is in effect and none of these conditions exists, ABEND0 passes control to ASIR0 (IGC0R01C) to process the STAE/STAI.

PURGING I/O: If I/O operations are active at this time, another ABEND condition might occur as soon as ABEND next loses control to the system such as across the XCTL to the next load of ABEND. Any outstanding I/O could terminate abnormally, or hinder normal processing. The Purge I/O routine is entered at this time to avoid ABEND processing for a recursion resulting from an I/O abnormal termination. After this initial ABEND, later ABEND processing of a valid recursion may also need to purge I/O. To accomplish this and avoid an ABEND/Purge loop, the Purge routine is skipped if both the ABEND bit (TCBFA) and the TCBREC flag indicate that an invalid recursion condition exists. The ABEND bit is on, and the TCBREC flag is off.

Before purging a first-time (nonrecursive) entry, an internal switch is set to indicate normal entry. The TCBFA bit is turned on, and the Purge routine is invoked. After the Purge routine has completed processing, the internal switch is tested. If a normal, first-time entry exists, the TCBFA bit is turned off and ABEND processing continues.

For a valid recursive entry, the internal switch is set to indicate this condition. The TCBREC flag is turned off, the TCBFA bit is turned on, and the Purge routine is invoked. If the internal switch (tested after Purge processing) indicates a valid recursion, the TCBREC flag is turned on prior to continuation of ABEND processing.

The Purge routine is not supposed to be able to abnormally terminate; if it does, due to a system error, DAR should be entered to set the job step nondispatchable. The TCBFA bit is turned on before calling the Purge routine to ensure that an abnormal termination condition will be treated as an invalid recursion. Control then passes to DAR without attempting another purge.

DETERMINING IF TERMINAL ATTENTION EXIT ELEMENTS (TAXE) ARE TO BE PURGED: ABEND0 determines from the TCBREC and TCBPTAXE flags in the TCBRECDE if this is a time sharing option (TSO) task and this entry to ABEND is not due to a TAXE purge recursion. If so, the TAXE recursion flag (TCBPTAXE) flag is set and the routine IEAKJXP is entered to purge the TAXEs for the terminating TSO task. Upon return, the recursion configuration flag is turned off.

CLEARING THE ABDUMP NONDISPATCHABILITY FLAG: ABEND0 clears the nondispatchability flag (TCBNDUMP) in the TCB and branches to the Task Switch routine for every task in the job step. ABDUMP may have previously set these flags to prevent alteration of dynamic queues during their display. ABEND0 clears these flags so that the Dispatcher may restart normal (nonterminating) tasks of the job step.

The following example illustrates this point. Assume the 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 progress and before the ABDUMP routine can reset nondispatchability, an error occurs that abnormally terminates task A. All tasks of the job step would remain nondispatchable if the ABEND0 routine did not clear ABDUMP nondispatchability soon after it gained control.

EXITING FROM ABEND0: If ABEND0 had not been previously entered for this task (no recursion) or if this entry is a valid recursion, control passes to ABEND1. ASIR0 (IGC0R01C) gains control if a valid STAE/STAI or I/O purge recursion is present.

Under the following conditions, exit from ABEND0 is to DAR1:

- Invalid ABEND recursion.
- Valid ABEND recursion in "must complete" status.
- Valid or invalid DAR recursion.

Processing during ABEND1 (Entry Point IGC0101C)

ABEND1 performs the following functions:

- Clears the "valid recursion" flags in the task's TCB.

- Recognizes whether a serious program error condition has occurred (such as a termination of a system task). If such a condition exists, ABEND1 passes control to the Damage Assessment routine (DAR) via an XCTL macro instruction, after purging asynchronous exit queues (AEQs).

- Defines the tree structure of the terminating task and performs housekeeping functions for the tasks in that tree.

- Releases the program interruption element (PIE) if it exists.

- Checks the validity of free queue elements (FQEs) in the main storage queues of the tree structure of the terminating task to avoid recursions during later execution of the GETMAIN and FREEMAIN functions.

- Purges those resources of the terminating task and its descendants that can operate asynchronously with those tasks 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, and unscheduled requests for user (asynchronous) exit routines. When the Storage Reconfiguration bit is on in a Model 65 Multiprocessing system, ABEND1 does not perform the I/O Purge and Timer Purge functions.

- Converts ABEND to job-step ABEND if necessary. The entry to ABEND1 is treated as a first-time entry after conversion.

- Passes control to ABEND3 which tests for specific valid recursions and routes control accordingly via an XCTL macro instruction.

ABEND1 first clears the "valid recursion" flag in the TCB for the current task. The "valid recursion" flag (TCBREC), if set during a previous partial termination of the task, must be cleared so that ABEND1 does not misinterpret as valid a future invalid recursion. The only valid recursions are those which ABEND expects to be possible, and which are handled via special ABEND processing.

PROCESSING FOR A VALID RECURSION: After determining that a valid recursion condition exists, the next step is to purge 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 earlier ABEND1 execution. New queue elements which were created for ABEND-initiated functions such as OPEN, DUMP, or CLOSE, must be eliminated. This is done to prevent waste of system resources.

ABEND1 then passes control to ABEND3 which eventually routes control for valid recursions.

RECOGNIZING A SEVERE ERROR CONDITION: After clearing the ABEND recursion flag (TCBREC), ABEND1 tests whether the current entry to the ABEND routine represents an error condition serious enough to warrant transfer of control to the Damage Assessment Routine (DAR). The two conditions which ABEND may not be able to handle are:

1. The attempted abnormal termination of any system task.

2. The attempted abnormal termination of a task in "must complete" status. A task in "must complete" status must be completed for the system or step to remain intact. It should not be abnormally terminated.

If such a condition exists, ABEND1 flags the current task as the top task in the tree structure of terminating tasks and removes requests for asynchronous exit routines before transferring control to the Damage Assessment Routine (DAR).

If neither of those conditions exists (as indicated by flags in the current TCB), ABEND1 continues processing.

PROCESSING FOR A FIRST-TIME ENTRY

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 or if a conversion was made to a step ABEND for a subtask of the job step, 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, ABEND1 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: ABEND1 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 ABEND0 as an invalid recursion, and would cause an entry to DAR. 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 these objectives, and to indicate that the tasks of the tree are in the process of abnormal termination, ABEND1 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 any routine of the task into execution.
- "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 ABEND0, on later reentry for the same task, that a recursion has occurred. TCBFA is also an indicator to other system functions that an ABEND condition exists.

To obtain the address of each TCB whose flags must be set, ABEND1 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 the current TCB is being processed, the "abnormal wait flag" (TCBABWF) in the current TCB remains cleared, so that the next module of the ABEND routine may be dispatched for the current task when ABEND1 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.

In a Model 65 Multiprocessing System, 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.

Checking the Validity of the Main Storage Queues: ABEND1 next checks the validity of the addresses of free queue elements (FQEs), and checks the correctness of their length fields. It does this for all unprotected subpools belonging to or shared by the job step. Then it checks unprotected subpools of the task currently selected by the "task select" subroutine. (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 abnormal terminations, with resulting recursions to the ABEND routine, during the later use of the GETMAIN and FREEMAIN functions.

Since FQEs are not in protected areas of main storage, they may be altered by a user program. When the GETMAIN or FREEMAIN routine tries to gain access to an altered FQE to satisfy a request, the result is an ABEND. The need for the validity check of FQEs is thus apparent.

ABEND1, via the MSSLOOP subroutine, scans the subpool queue for a selected task, searching for descriptor queue elements (DQEs). ABEND1 examines all FQEs for each DQE for an owned subpool. It makes three general checks for each FQE:

1. ABEND1 first verifies that the free-area length specified in the FQE length field does not exceed the length described by the associated DQE.
2. ABEND1 next examines the validity of the free-area address in the FQE. It determines if the address specifies a location that is on a doubleword boundary, is within the bounds of main storage, and specifies a location that is in the area described by the associated DQE.
3. As a last test, ABEND1 verifies that the next FQE (pointed to by the FQE being examined) is at a lower main storage location than the FQE under examination.

ABEND1 nullifies the effect of an invalid FQE by setting the FQE address to zero

in the DQE which describes the block. The entire space described by this DQE thus appears allocated. When all FQEs belonging to all subpool queue elements of the selected task have been examined (and altered if necessary), the validity check of the main storage queues is complete. After the check of main storage queues, ABEND1 can safely purge the resources for the specified task.

Purging Resources for the Specified Task and Its Descendants: ABEND1 (via separate routines) purges for the tree of tasks the program interruption element (PIE) if one has been specified, the timer queue, I/O requests and I/O operations in process, the WTOR queues, and the asynchronous exit queue for non-I/O requests. Using the "task select" subroutine, and starting with the newest descendant task of the specified or "top" task, ABEND1 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 is selected for FQE validity check and release of resources.

Releasing the Program Interruption Element (PIE): ABEND1 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, ABEND1 branches to the FREEMAIN SVC routine to free the storage space. Subpool 250 is specified in the FREEMAIN parameter list to free subpool 0.

Since an ABEND condition can occur while trying to free a PIE which was inadvertently freed by the user, ABEND1 sets the pointer to the PIE (TCBPIE) to zero and the PIE is conditionally freed. Any part of the PIE not freed now is released when the job step is terminated.

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.) Before ABEND1 branches to the Timer Purge routine, it sets the valid recursion flag TCBREC. This flag stays on when the Timer Purge routine enters the TAXE Purge routine via a branch and link. A TAXE purge of all tasks in the terminating tree (except the current task which was purged earlier) is performed at this time.

Purging I/O Requests and I/O Operations in Process: ABEND1 purges I/O requests and I/O operations 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. This could cause data or programs to be destroyed. Furthermore, an event control block may be posted in reallocated main storage, thus causing an additional error. ABEND1 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, ABEND1 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 ABEND16 frees dynamically acquired main storage.

Removing Requests for User (Asynchronous) Exit Routines: After purging the operator communication queues, ABEND1 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 are freed later during ABEND16 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 an invalid recursion if the Exit routine abnormally terminates. Such execution can also slow up the termination processing. After the TCBFX flag is cleared, the Stage 3 Exit Effector can schedule asynchronous exit routines for system functions that may be needed during I/O operations for Open, Close, and ABDUMP processing. (See "Scheduling of User Exit Routines" in Section 3, "Task Supervision.")

Processing during ABEND3 (Entry Point IGC0301C)

ABEND3 performs the following main functions:

- Releases partially loaded modules.
- For any recursion other than a type-1 message recursion, ABEND3 purges any entries in the information list for the TCB that just terminated.
- Allows asynchronous exits.
- Routes valid recursions.

RELEASING PARTIALLY LOADED MODULES: ABEND3 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 requesters waiting for the module cannot gain access to it. Of primary interest are the modules containing the ABDUMP and BSAM routines, needed by ABEND8 and ABEND9. These modules may have been requested by a subtask of a task that is now terminating. Since ABEND1 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 ABEND3, 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 program area and, if this is not a job-step task, the extent list.
- 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 requesters which are awaiting the loading of the module, prepares one or more RBs for reentry to Contents Supervision (IEAQLK00) 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.) Each CDE whose module is being loaded is purged if either of two requirements is met:

- The loading was initiated for the current task. In this case, the current task is being terminated and its I/O operations have already been purged by ABEND0.
- The loading was initiated for another task which is being abnormally terminated and is nondispatchable (TCBABWF flag set), and whose I/O operations, initiated for loading, have already been purged by ABEND1 (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 ABEND1. Otherwise, the Main Storage Supervision routines may reallocate the freed program area for another task. The requested module may later arrive in main storage, overlaying the reallocated program area, which now belongs to another task.

PURGING TYPE-1 MESSAGES ON RECURSION: On recursive entry to ABEND for any recursion other than the type-1 message recursion, a new entry might have been made in the list as part of the recursion. Therefore, ABEND3 purges any entries in the type-1 information list which are for the TCB that recursively terminated.

DECIDING THE NEXT PART OF ABEND TO BE INVOKED: After allowing the scheduling of I/O exit routines, ABEND3 determines which part of the ABEND routine it should invoke:

- ABEND4 - current entry is a first-time entry, or a type-1 message WTP recursion.
- ABEND7 - recursion due to an error in the Graphics Debug routine, or in closing the Direct SYSOUT data set.

- ABEND8 - recursion due to a failure in the Open routine.
- ABEND9 - recursion due to an error detected in the ABDUMP routine.
- ABEND11 - recursion due to an error in the Close routine.
- ABEND13 - given control if any other valid recursions are indicated in the TCBRECDE field.

Processing during ABEND4 (Entry Point IGC0401C)

ABEND 4 has three main purposes:

- Write type-1 SVC messages.
- Purge message list entries.
- Free type-1 message WTP buffers.

WRITE TYPE 1-SVC MESSAGES: Type-1 SVC routines run in a disabled state and thus are unable to issue WTP messages as a result of an abnormal termination. Instead, a message information list is maintained in the nucleus in which type-1 SVC routines can store data. The list is located by a pointer in the CVTQMSGA field of the CVT.

If the entry to ABEND4 is a message recursion (the TCBTYP1W flag is set), processing continues with the purging of message list entries. Otherwise, a check is made for an unprocessed entry in the message information list for the currently terminating task's TCB. If an entry is present, a conditional GETMAIN is issued to obtain storage from which to write messages. If the fifth word in the extended save area (ESA) is zero, there is no storage available to fulfill the GETMAIN request. No message list elements are formatted, and processing continues with the purging of the list elements.

If storage is available, the fifth word in the ESA contains the address of the storage area allocated for the WTP buffer. ABEND4 formats the list elements to be written. The following items are placed into the message:

- Reason code
- Job name or branch address if the routine causing the ABEND condition was entered via a branch entry
- Step name
- Flag characters
- Message text (maximum of 16 bytes)

After formatting has been completed, the descriptor, routing codes, and message recursion flags are set. A WTO macro instruction is then issued to write the message. After the message has been written, the valid recursion flags (TCBRECDE) are turned off, and any remaining list entries are processed.

PURGING OF MESSAGE LIST ELEMENTS: When all messages have been written, a check is made of the information list for entries corresponding to other tasks in the terminating tree. Such entries are purged so the elements can be used again.

FREEING OF TYPE-1 SVC MESSAGE WTP BUFFERS: A lower level task may have terminated earlier which was processing a type-1 SVC message for that task when the current ABEND condition occurred. If so, a previous entry into ABEND4 may have already obtained storage for a WTP buffer. ABEND4 checks the fifth word in the ESA for a storage address. If there is a buffer for any task in the terminating tree, it is freed.

EXITING FROM ABEND4: When all entries have been purged, ABEND4 determines where to route control, and issues an XCTL instruction to the proper module. If the rollout/rollin feature is in the system, ABEND5 gains control. Otherwise, ABEND7 receives control if a dump was requested, and ABEND11 if it was not.

Processing during ABEND5 (Entry Point IGC0501C)

The main purpose of ABEND5 processing is to purge queues related to the rollout/rollin feature. The Rollout Purge routine (ROLLPRG) is an internal routine used to remove the appropriate IQEs from the rollout queues.

When a request for rollout cannot be satisfied, the IQE representing that request may be added on a FIFO basis to a queue to defer the request until a later time when it can be satisfied. While a task has an IQE on the rollout queue (IEAROQUE), it is set nondispatchable. It may be abnormally terminated, however, and the IQE must be removed from the rollout queue because the task no longer needs the rollout request. This abnormal termination may be the result of issuing the CANCEL command, a time expiration, or a higher level task ABEND.

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 successively of the address of each TCB in the terminating task tree.

IQEs on the rollout queue are examined first. If the TCB address in the first 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 (AEQE) 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 AEQE 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.

EXITING FROM ABEND5: ABEND5 passes control to ABEND7 if a dump is requested and allowed. Otherwise exit is to ABEND11.

Processing during ABEND7 (Entry Point IGC0701C)

ABEND7 is entered only if an ABEND dump is requested and allowed. It does any feature-dependent processing necessary prior to the opening of the dump data set. The two main functions are:

- Processing for graphics jobs.
- Closing the Direct SYSOUT Writer if it has been started to tape.

PROCESSING FOR GRAPHICS JOBS: The Graphics Debug routine is entered for foreground jobs and the Graphics Job Processor. It is not entered for a Storage Reconfiguration ABEND condition. For a graphics recursion, the recursion flag is turned off, and control passes to the routine that checks for a Direct SYSOUT Writer.

Before calling the Graphics Debug routine, ABEND7 checks whether a dump is allowed for the terminating task. If a dump is allowed, a GETMAIN macro instruction is issued to test if there is sufficient storage available in subpool 252 for ABDUMP's resident module, IEAQAD0A. If there is either insufficient storage or a dump is not permitted, a parameter is passed to the Graphics routine indicating that the user may not request a dump.

In the event of a successful GETMAIN request, a FREEMAIN macro instruction is issued to free the storage. The parameter passed to the Graphics routine indicates that a dump is available to the user.

ABEND7 then turns on the recursion configuration flags and passes control to the Graphics Debug routine. If the dump parameter is on, the user has the option of accepting or bypassing the dump. The Graphics routine queries the user, sets the parameter to indicate the user response, and returns to ABEND7. If the dump is requested, ABEND7 processing continues. If the user indicated that he wished to bypass the dump, ABEND7 turns off the high-order bit of the completion code field in the TCB and issues an XCTL to ABEND11.

DETERMINING IF A DIRECT SYSOUT WRITER HAS BEEN STARTED: If a dump is requested, ABEND7 tests to determine if the SYSUDUMP/SYSABEND dump data set is allocated to a tape device to which a Direct SYSOUT Writer has been started.

If the dump is requested for a subtask ABEND, the dump is bypassed to prevent a later attempt to use the tape device. Such an attempt would cause another ABEND. The ABEND dump data set remains open for the entire job step. Thus if a subtask were allowed to take a dump, the direct SYSOUT device would remain open needlessly for the entire job step. Any other user attempting to use the direct SYSOUT device, even after the subtask had completed, would be unable to do so.

If the dump is requested for a job-step ABEND, and the SYSOUT data set is already open for the user requesting the dump, the SYSOUT data set is closed so that it can be reopened as an ABEND dump data set. After ABEND9 processing has completed, ABEND11 closes the data set.

If a recursion occurs when attempting to close the SYSOUT data set, the "prevent dump" indicator (TCBPDUMP) is set, and ABEND7 exits. ABEND7 passes control via an XCTL to (1) ABEND8 if a dump is requested and can be given at this point, or (2) ABEND11 if the dump is not requested or must be bypassed.

Processing during ABEND8 (Entry Point IGC0801C)

ABEND8 processing deals with the following dump-related functions:

- Processing for recursive entry due to an open failure.
- Determining if the "prevent dump" indicator is set.
- Determining if there is a dump data set to be opened.
- Determining if the dump data set is already open.
- Opening the data set.
- Ensuring that the dump data set remain open for the duration of the job step.
- Indicating if the dump data set has been opened.

DETERMINING IF THE ENTRY TO ABEND8 IS DUE TO RECURSION: If the entry to ABEND8 is due to recursion, the TCBREC and TCBOPEN configuration flags were set in the current TCB by previous ABEND8 processing. 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, ABEND8 does not try again. Instead it performs the following functions:

- Resets the Open recursion indicators in the current task.
- Locates the previous SVRB under which ABEND8 was operating, and extracts the saved load list elements and TCBMSS pointer.
- Indicates an internal "no dump now" condition.
- Updates the load list queue on the current TCB.
- Restores the current TCBMSS pointer and sets non-terminating tasks of the job step dispatchable.
- Passes control, via an XCTL, to ABEND11.

DETERMINING IF THE "PREVENT DUMP" INDICATOR IS SET: If the entry is not a recursion, ABEND8 next tests if the "prevent dump" indicator (TCBPDUMP) is set in the job-step TCB. The indicator may have been set at this time because of a Direct SYSOUT recursion. This indicator may also be set at a later point in ABEND processing for either of the following two conditions:

- "Steal core" situation exists.
- The "dump data set open" bit is on, but the data set cannot be found. This is the result of a dump data set which was opened (thus accounting for the bit setting), but later reclosed as part of Normal Termination for the job-step task, which then terminated abnormally. Because the close occurred before ABEND8 gained control, ABEND8 searches for the dump data set which it is unable to locate.

If the "prevent dump" indicator is set, ABEND8 cannot prepare for dumps. It therefore bypasses the rest of its processing and passes control to ABEND11 to close the data sets used by the terminating tasks.

DETERMINING IF THE DUMP DATA SET IS TO BE OPENED: ABEND8 performs tests to determine if it should open the dump data set (SYSABEND or SYSUDUMP) for use during ABEND9. These tests determine:

- Whether the caller of the ABEND routine has requested a dump.
- Whether the SYSUDUMP data set was allocated by means of a DD card.
- Whether the SYSABEND data set was allocated by means of a DD card.
- Whether the dump data set was previously opened for another task in the job step.

TESTING THE DUMP REQUEST AND THE STATUS OF THE DUMP DATA SET: ABEND8 makes three tests to determine whether it should open the dump data set, or whether it should invoke ABEND9 or ABEND11 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 if a dump was requested by the caller of the ABEND routine. ABEND8 tests the high-order bit of the completion code in the TCBCMP field of the current TCB. If the caller did not request a dump, or if neither data set was allocated, ABEND8 invokes ABEND11.

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.

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 if the Open function may be bypassed. (The test checks the TCBFDSOP flag in the job-step TCB.)

The DD card could be missing now, but defined later if dynamic device allocation is being used. In this case, ABEND8 skips the dump now, but allows an ABEND dump of a future terminating task.

DETERMINING IF THE DUMP DATA SET IS ALREADY OPEN: If the dump data set was previously opened, ABEND8 obtains the DCB address by searching the DEB queue of the job-step task for the DEB belonging to the data set. (When opening the dump data set originally, ABEND8 located the DEB belonging to the data set and set a special identifier.) It then extracts the DCB address from the DEB for use during I/O operations of the dump procedure of ABEND9. After obtaining the DCB address, ABEND8 passes control to ABEND9 to take the dump.

It is possible that the dump data set was previously opened but is no longer on the DEB chain. Consider the following case: The dump data set was previously opened for an abnormally terminating subtask of the job step. Then it was closed in normal termination for the job step. Subsequently, the job step was abnormally terminated. The dump data set cannot be reopened because that would overlay information in the dump for the subtask. In this case, ABEND8 sets on the prevent dump indicator and passes control to ABEND9.

OPENING THE DUMP DATA SET: After all tests have been made to ensure that the dump data set should be opened, ABEND8 performs the necessary Open functions. The field in the extended save area in which the load list pointer will be saved after the GETMAIN for the DCB must be zeroed out. This pointer is added to the present load list chain if the GETMAIN fails and there is a recursion. If this were not done, there would be an invalid recursion later when ABEND8 attempts to free the load list.

All tasks (except the current task) in the task tree of the job step are set non-dispatchable, and the TCBMSS pointer for the current task is saved in the extended save area. The TCBMSS pointer from the job step TCB is put into the TCBMSS field of the current task. This ensures that the IOBs for the dump data set will come from the job-step task's subpool 0.

ABEND8 issues a GETMAIN macro instruction to obtain storage for the DCB, which is then moved into the newly acquired area. The ABEND dump data set is then opened to perform one of two dumping functions. One of two DD cards may be specified for the ABEND dump data set. Although both may be specified in a JCL procedure, the first one found in the TIOT is used across the job step. For a SYSABEND DD card, a dump of the nucleus, control blocks, and job-step region is given. A SYSUDUMP DD card results in a dump of only the control blocks and job-step region.

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), ABEND8 tries to ensure that the dump data set remains "open" for the remainder of the job step. "Open" means the retention of 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 ABEND11 closes data sets belonging to the terminating tree of tasks, and ABEND16 frees 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 is removed and freed from its DEB list during ABEND11, and the BSAM routines possibly released during ABEND16. 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 Reference publication).

ABEND8 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.

ABEND8 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 ABEND8 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. New CDES are not created if the BSAM routines are already in main storage. After ABEND8 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. ABEND8 then issues the OPEN macro instruction. Regardless of whether the data set is actually "opened," ABEND8 restores the original load list pointer (TCBLLS) to the current TCB. If a recursive entry to the ABEND routine occurs because of an error during the execution of the Open routine, the load list pointer is also restored.

If the open attempt is successful (as indicated by a flag in the DCB of the dump data set), ABEND8 uniquely labels the data extent block (DEB) associated with the dump data set so that it can later find the DEB, and obtain from it the associated DCB address. ABEND8 passes the DCB address to ABEND9. The position of the DEB in the DEB queue can vary because of processing by the End-of-Volume (EOV) data management routine.

ABEND8 then places the DEB on the job-step task's DEB queue. It does this to prevent ABEND11 from closing the dump data set, when it closes data sets belonging to the terminating tree of tasks.

INDICATING IF 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, ABEND8 tests the appropriate flag of the DCB to learn if the data set has been actually opened. According to the results of the test, ABEND8 indicates, via a flag bit in the job-step TCB if the open request has been successful.

If the data set could not be opened, and therefore abnormal dumps are not possible, ABEND8 passes control to ABEND11.

If the dump data set was actually opened, ABEND8 sets an indicator in the job-step TCB. It sets the "data set open" indicator (TCBFDSOP) so that in a later ABEND within the job step, it may bypass much of ABEND8 processing and invoke ABEND9.

Regardless of whether the dump data set could be opened, ABEND8 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.

CONCLUSION OF ABEND8: ABEND8 processing is completed according to the results of tests already described in the topic "Determining if the Dump Data Set is to be Opened." Control passes to ABEND9 for dump processing or, if no dump is required, to ABEND11.

Processing during ABEND9 (Entry Point IGC0901C)

ABEND9 performs ABDUMP processing. This includes the dump of resources belonging to the terminating tasks of the tree (that is, the specified task and its descendants).

ABEND9 uses a number of SVC routines to perform its functions. Of primary importance are the ABDUMP routines. For each invocation, ABDUMP displays the resources of the selected task. ABEND9 invokes the ABDUMP routine separately for the "top" task, its descendants, and its direct ancestors.

There are two types of entries to ABEND9: first-time entries and recursive 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 ABEND8. A recursive entry represents a request for abnormal termination generated by ABDUMP because of an error detected during its processing. A recursive entry to ABEND9 is always made directly from ABEND3, via an XCTL macro instruction.

The scope of ABEND9 processing varies, depending on the particular type of entry. A first-time entry permits ABEND9 to perform the dump function. A recursion because of an ABDUMP error causes the bypassing of the dump function, but permits ABDUMP-related cleanup functions to be performed. A Storage Reconfiguration ABEND also causes the bypassing of the dump function.

DETERMINING THE SCOPE OF PROCESSING: Before ABEND9 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 (the ABDUMP routine can request a reentry to ABEND if it discovers an error).

If a recursion has occurred from the ABDUMP routine (as indicated by the "set" condition of the "ABDUMP recursion" configuration flag TCBADUMP in the current TCB), ABEND9 issues a DEQ macro instruction specifying the dump data set.

If no recursion from the ABDUMP routine has occurred, ABEND9 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 (TCBPDUMP) in the job-step TCB to learn if ABEND7 or ABEND8 had discovered an abnormal condition. ABEND7 sets the indicator if there was a Direct SYSOUT Close failure. ABEND8 sets the indicator if the dump data set was closed by normal termination of the job-step task.

If the "prevent dump" indicator is not set, ABEND9 tests the DCB address passed by ABEND8 in a register. If the address is zero, the caller of the ABEND routine has not requested a dump. Accordingly, ABEND9 bypasses the ABDUMP processing and passes control, via an XCTL, to ABEND11. But if the address is not zero, the DCB register contains the address of the DCB that ABEND8 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 are 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 or the Open or previous dump failed for the terminating task tree. The error condition might be corrected by successful termination of this tree (if there was not enough main storage available). Therefore, a later dump could be possible. But if the "prevent dump" indicator is set, no abnormal dump will be performed for the remainder of the current job step, since a permanent error exists.

For Storage Reconfiguration, the "prevent dump" has already been set as a result of Machine-Check Handler processing.

If the "prevent dump" indicator is not set, and the DCB address passed to ABEND9 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.

Preparation for the Dumps: Before ABEND9 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 (IEAQAD0A) 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. ABEND9 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. ABEND9 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, ABEND9 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, ABEND9 issues an ENQ macro instruction (with the "exclusive" option) for the dump data resource 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 ABEND9. 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 ABEND9, when the dump of the subtask's resources is complete. But since the subtask is now nondispatchable (its TCBABWF flag set during ABEND1), the DEQ macro instruction cannot be issued by ABEND9 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 for that resource belonging to the top terminating task and any of its subtasks. The current ENQ request issued by ABEND9 can then be serviced.

Performing the Dumps: Dumps of the terminating tasks of the tree are made if the following conditions have been met, tested by both ABEND8 and ABEND9:

1. A dump data set has been provided (as indicated by a search of the TIOT by ABEND8).

2. A preassembled DCB can be opened (also by ABEND8).

3. The caller of ABEND has requested dumps.

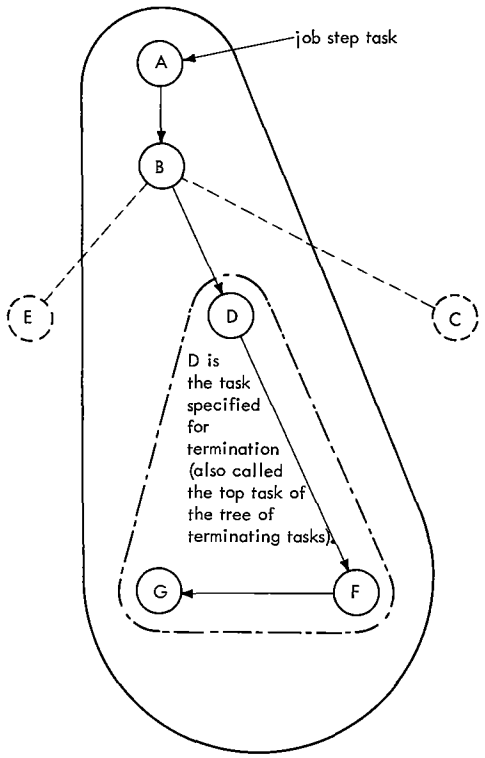
If the user has provided a SYSABEND DD statement, and if GTF is active to format (in internal mode), GTF is suspended for the dump by issuing a HOOK macro instruction. This stops GTF trace entries until the trace is dumped.

Via issuance of the SNAP macro instruction (SVC 51), ABEND9 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 10-8).

Each task of the tree of tasks (D, G, and F in Figure 10-8) is selected by means of the "task select" (TASKSEL) subroutine. As each task is selected by the subroutine, ABEND9 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, ABEND9 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 ABEND8, and the fact that the ABEND routine is the caller (via a bit that is set in the ABDUMP parameter list provided by ABEND in its SVRB ESA. On each return of control from the ABDUMP routine for a subtask, ABEND9 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 (skipped if the DD card was the SYSUDUMP), load modules, and the subpool blocks. If GTF is active and in internal mode, the GTF trace is also formatted by ABDUMP for a SYSABEND data set even though ABEND9 does not specify a trace in the SNAP request.

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.



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 10-8. Task Relationships During an Abnormal Termination

Cleanup after the Dumps: After the dumps of the ancestors, ABEND9 performs several cleanup steps. It first dequeues the dump data set[1] resource so that is is available for use by the next requester 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, ABEND9 deletes the resident module of ABDUMP (IEAQAD0A), 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

---

[1]The dump data set can be specified as SYSABEND or SYSUDUMP.

instruction for the same module, it is ineffective in releasing it, since the ABEND routine's request for the module is still outstanding.) When ABEND9 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 resource and issuing a DELETE macro instruction for ABDUMP's resident module, ABEND9 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, and ABEND9 passes control to ABEND11.

Processing during ABEND11 (Entry Point IGC0B01C)

ABEND11 performs the following functions:

- Turn on the Generalized Trace Facility (GTF) for each task in the tree which had previously turned it off.
- Purges ECBs awaited by programs associated with tasks in the terminating tree.
- Erases complete subtasks.
- Checks for storage to close and routes to steal.
- Closes open data sets.
- Frees the PIE.

ABEND11 calls the "task select" subroutine to select each task of the terminating tree, one task at a time. ABEND11 then examines each task to see if a GTF trace suspension is still outstanding against the TCB. This is determined by examining the TCBGTOFM bit in the selected TCB. If set, a HOOK macro instruction is issued to resume the GTF trace. The TCBGTOFM bit is then reset, and the next task is selected.

ABEND11 next purges any ECBs that are awaited by programs associated with tasks in the terminating tree. The task select subroutine is called for each task; the "wait pending" flag (RBWAITP) in each RB associated with the selected task is then examined. If the RBWAITP flag is set (indicating that the RB is waiting on one or more ECBs), ABEND11 locates the awaited ECBs by obtaining the contents of register 1 at the time the WAIT macro instruction was issued. ABEND11 marks the ECBs as completed, and turns off the RBWAITP flag in the RB. This processing has the effect of causing any subsequent POST macro instruction against the ECBs to be treated as a no-operation condition.

ABEND11 uses the Close routine of data management to close the data sets and purge the DEB queues. For a first-time entry ABEND11 closes all data sets, and purges all DEBs. A recursive entry resumes Close processing from the point of the error. If a recursion has occurred from the Close routine (as indicated by the "Close recursion" flag TCBCLOSE in the current TCB), ABEND11 performs special processing to continue closing data sets and purging DEBs. This processing is described under "Special Handling of a Recursion" below.

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 is performed separately for each task of the tree while the task is currently active. The "task select" subroutine 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 routine'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 ABEND11 is dispatched for the selected task it closes data sets, and purges DEBs. When all DEBs have been purged, the "task select" subroutine 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, the "task select" 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 10-8 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 "task select" subroutine tests the "completion" flag (TCBFC) to learn whether the task has

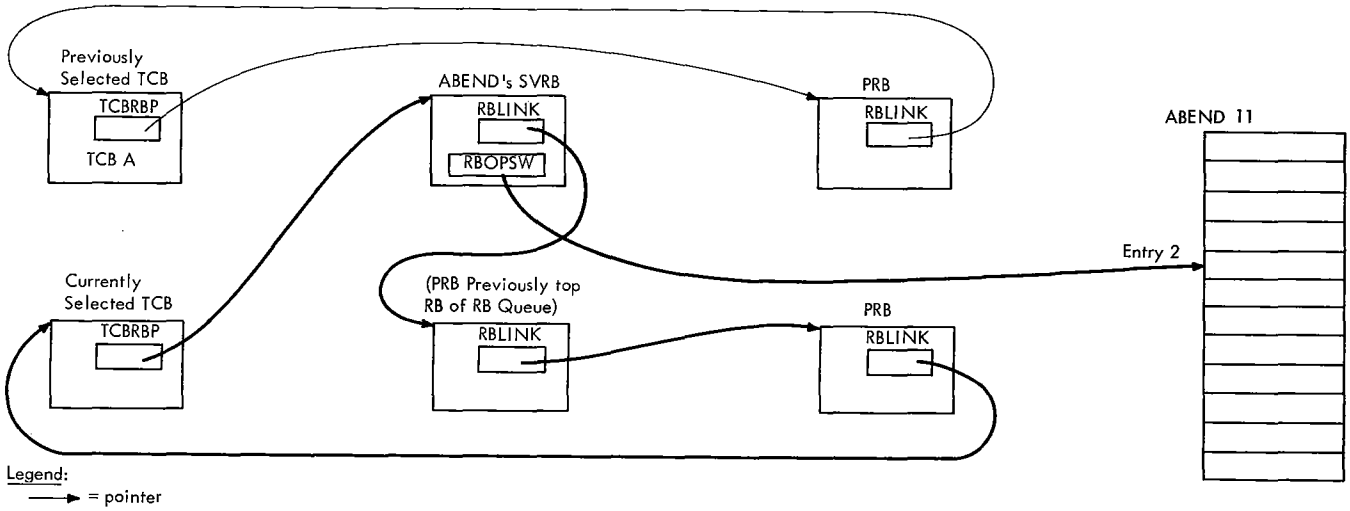
already been terminated (either normally or abnormally). If the selected task has already been terminated, and its TCB is thus no longer needed, ABEND11 branches to the resident "erase" routine (part of the EOT routine) with the address of the terminated TCB. The "erase" routine dequeues the selected TCB from the subtask queues (whose pointers are TCBLTC and TCBNTC in each TCB) and frees the space it occupies. When control returns from the "erase" routine, the "task select" 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 ABEND11 is prepared for dispatching under control of the selected TCB.

Preparation for the Dispatching of ABEND11 Under Control of the Selected Task: As stated before, closing data sets is 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, ABEND11 is redispatched (at location ENTRY2) to begin execution of its "close data sets" function.

Preparation for the redispatching of ABEND11 occurs as follows. First, the current task is set nondispatchable (TCBABWF flag is set) so that ABEND11 temporarily cannot be redispatched for this task. The task selected by the "task select" subroutine is then made dispatchable (two bytes of non-dispatchability flags are cleared in its TCB) in preparation for the branch to the Dispatcher. ABEND11 stores in the RB old 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 ABEND11 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 10-9.) The ABEND routine's SVRB is then removed from the RB queue of the previously current TCB, since ABEND11 can service only one task at a time.

To ensure that the registers contain the correct values when ABEND11 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, loads the registers from this TCB save area.

ABEND11 next branches to the Task Switch routine, making available the address of the selected TCB. The Task Switch 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 Switch 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



Note: ABEND's SVRB is shifted to the RB queue of the currently selected task.

Figure 10-9. Preparation for the Dispatching of ABEND11 for the Selected Task

another lower priority task by a search of the TCB queue in a downward-priority direction.

Finally, ABEND11 branches to the Dispatcher which passes 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), ABEND11 is redispatched (at location ENTRY2) to purge data sets for the selected task.

Closing Data Sets for a Selected Task: The closing of data sets for a selected task consists of issuing 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; the address of the DCB 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. After each DEB has been freed, ABEND11 frees the program interruption element (PIE), which may have been created by a SPIE macro instruction in a user tape label routine.

Prior to closing data sets, ABEND11 issues a GETMAIN instruction to test whether there is sufficient storage for the Close routine. If the GETMAIN should fail, an XCTL is issued to link to ABEND12, which attempts to free ("steal") the necessary storage. Once storage has been obtained, ABEND11 closes data sets and purges DEBs.

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. Before issuing CLOSE macro instructions, ABEND11 stores the flag byte, and the DCB address that was obtained from the first DEB, in the parameter list for the Close routine. 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 Data Management Macro Instructions.)

After setting up parameters for the Close routine, ABEND11 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: updating the DEB pointer (TCBDEB), and freeing the current DEB. After regaining control from the Close routine, ABEND11 compares 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, ABEND11 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 flags indicates a valid recursion to the ABEND routine, and causes reentry to ABEND11 from ABEND3 to continue the DEB processing.

ABEND11 invokes the Close routine 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 the SVRB 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 ABEND11 as the current routine for the still-active current task. ABEND11 resets the "close" and "recursion" flags in the selected TCB to indicate that a recursion (if it now occurs) is not valid. These flags are set again 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. ABEND11 then branches to free the PIE.

If the two DEB addresses are equal, the Close routine did not process the current DEB. Accordingly, ABEND11 provides a pseudo-link (via an SVC 13) to the GAM module IGGIFF01. This link results in a reentry into ABEND0, which issues an XCTL to link to IGGIFF01, thus giving the GAM module its own SVRB. ABEND uses a reentry configuration across the pseudo-link, and a valid recursion configuration (TCBCLOSE) across GAM processing. The valid recursion configuration across GAM processing is necessary should IGGIFF01 itself abnormally terminate such as for an invalid DEB or UCB data. (Note that ABEND does not ensure the validity of the DEB which GAM is process-

ing. The DEB is the one addressed by the current TCBDEB field.) If IGGIFF01 terminates abnormally, control passes through ABEND to the module that invoked the GAM routine; processing continues as if the interface had been normal.

After GAM has completed processing, it issues an SVC 3 (EXIT) which causes its SVRB to be dequeued, and control returned to ABEND11. ABEND11 first resets the recursion configuration flag. It then dequeues the DEB from the DEB queue, determines its size, and frees the space it occupies. If the TCB points to a PIE, the pointer is cleared and the PIE is freed. After the PIE has been freed, ABEND11 branches to repeat the processing for the next DEB. The GAM module IGGIFF01 is entered for each DEB that has not been dequeued by the Close routine.

Special Handling of a Recursion: An error can occur during the execution of the Close routine, causing a recursion to ABEND. Such an error can be caused by overlaying one or more DCBs because of the "steal core" function of ABEND12. Regardless of the cause, a recursion because of an error during the Close routine needs special handling to permit 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, ABEND3 invokes ABEND11 directly to continue closing data sets and purging DEBs. A test at the beginning of ABEND11 recognizes the set condition of the "close" indicater in the selected TCB, and branches to perform special handling.

ABEND11 first locates 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. This SVRB holds the last used DEB address in its extended save area. The DEB address was placed in the extended save area by ABEND11 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.

ABEND11 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 have been closed, ABEND11 determines where to transfer control. If the current task is the top task in a tree which is abnormally terminating, ABEND13 receives control. Otherwise, a subtask is selected for termination via redispatching to this ABEND module.

Processing during ABEND12 (Entry Point IGC0C01C)

ABEND12 has two main purposes:

- Determines whether the storage required by the Close routine is available.
- Attemptes to obtain storage if the ABEND is a job-step ABEND.

ABEND12 tests whether storage is available for use by the Close routine during ABEND11 processing. If sufficient storage is available (552 bytes), control is passed back to ABEND11 to continue the termination procedures.

If the required storage area cannot be obtained, ABEND12 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, ABEND12 "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.

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. ABEND12 prepares for a job-step termination. It does this by putting the completion code in the ESA, and setting the conversion configuration flag (TCBCONVR). It then passes control to ABEND1 which extends the ABEND to include the entire job step. When the "check for core" subroutine in ABEND11 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.

DETERMINING WHETHER STORAGE IS AVAILABLE: The "steal core" subroutine, used by ABEND12, tests for the availabliity of main storage for the Close routine. It issues a conditional GETMAIN macro instruction for 552 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, ABEND12 passes control back to ABEND11.

ATTEMPTING TO "STEAL CORE": If main storage is not available, the "steal core" subroutine tests if the current termination is for the entire job step. If it is not, conversion to step termination and purging 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 ABEND11 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. This flag is set only to indicate that storage has been stolen.

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 ordinarily does not hold data control blocks (DCBs). Thus, the "stealing" of allocated space from this subpool will probably not cause recursions to the ABEND routine when ABEND11 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 552 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.

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. If the SPQE is for subpool 0, the FREEMAIN parameters must specify subpool 250.

EXITING FROM ABEND12: ABEND12 passes control, via an XCTL, to ABEND1 if conversion to a job step ABEND is necessary. ABEND11 receives control if storage is available for the closing of data sets. In the situation in which the "steal core" subroutine fails to obtain storage, control passes to DAR1 because this is an invalid system condition.

Processing during ABEND13 (Entry Point IGC0D01C)

ABEND13 performs the following:

- Passes control to the TCAM close module.
- Frees SCBs.
- Cancels Inter-Partition Posts for TCAM and TSO tasks.
- Purges QELs.
- Purges transient SVRBs.
- Purges dynamic DD entries (in the TIOT) invalidly marked busy.

ABEND13 first checks whether entry to this module is a recursion or a first-time entry. If the recursion is due to an error in the purge routines of TSO or TCAM Inter-Partition Posts, the respective routines are not reentered. If the recursion is due to an error in the TIOT Check routine, processing continues with the removal of transient SVRBs. For a first-time time entry, the "task select" subroutine is called to select a task to be purged. If one is found, normal processing takes place. If a task is not selected to be purged, ABEND13 performs its exit processing.

TCAM CLOSE MODULE INTERFACE: ABEND13 gives control to the TCAM Close module which performs closing functions for TCAM if they have been bypassed in normal close processing or close recursion. The interface of ABEND13 with the TCAM Close module is via a pseudo SVC with a recursion configuration set across TCAM Close processing.

CANCELLATION OF PENDING INTER-PARTITION POSTS FOR TCAM: In a TCAM environment, there may be a pending request to post this task by another task. SVC 102 is an SVC instruction which effectively cancels such pending requests. If those requests ere not canceled, this task could terminate and another task, possibly having the same TJID, TCB address, and RB address, could be posted by mistake.

FREEING OF STAE CONTROL BLOCKS: If an SCB exists, as indicated by an address in the TCBNSTAP field, the SCB chain is updated and a FREEMAIN instruction is issued to free each SCB.

CANCELLATION OF PENDING INTER-PARTITION POSTS FOR TSO: In a TSO environment, there may be a pending request to post this task by another task, possibly swapped out at this time. QTIP is a macro which will effectively cancel all such pending requests. If those requests were not canceled, this task could terminate and another task, possibly having the same TJID, TCB address, and RB address as this task, could be posted by mistake.

If TSO is active, this macro must be called even if this is not a TSO task. This task could be a foreground initiated background task, and thus pending requests must be canceled.

This macro call is made for each task in the tree of terminating tasks. Because QTIP may be called more than once for a terminating task without complications, no special precautions are necessary to avoid a second call during recursive processing. However, if entry to ABEND13 is due to a QTIP recursion, QTIP processing is bypassed.

ENQ/DEQ PURGE: If the selected task is not the top task in the tree, ABEND13 enters the resident ENQ/DEQ Purge routine (at entry point IEA0EQ01) to remove resource requests generated by issuing the ENQ macro instruction for the selected task. This is the task selected by the "task select" 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 can 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 returns to ABEND13 which must reestablish addressability.

The ABEND13 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. When such a reservation is detected, the ENQ/DEQ Purge routine branches to the EXCP interface subroutine in the ENQ/DEQ module. This subroutine prepares control blocks for an EXCP command and issues the EXCP command that results in the release of the reserved device. For this reason it is essential that the ENQ/DEQ Purge be done in an ABEND module which runs partially enabled.

When all QELs and QCBs for the selected task have been purged, the "task select" (TASKSEL) routine is invoked to select the next higher level task of the tree.

REMOVAL OF REQUEST BLOCKS FOR TRANSIENT SVC ROUTINES: ABEND13 locates SVRBs for transient routines by searching 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 ABEND13 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 (deferred) 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 requester 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.

When control returns from the TAHABEND subroutine, ABEND13 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).

TIOT CLEANUP: After processing has been completed for each task in the terminating tree, a check is made to determine if the terminating task is a time-sharing subtask that did not reenter ABEND13 because of a TIOT recursion (TCBDYNAM). If this is the case, the TIOT Check module is entered to perform cleanup processing for dynamic DD entries in the TIOT.

EXIT PROCESSING: ABEND13 passes control to ABEND15.

Processing during ABEND15 (Entry Point IGC0F01C)

ABEND15 purges CDEs, associated programs, and extent lists.

ABEND15 selects each task by means of the previously described "task select" subroutine, starting with the newest descendant of the specified task and ending with the specified or "top" task itself. But unlike the processing of ABEND11, in which each task was redispatched under the control of ABEND's SVRB to purge its own resources, all of ABEND15's processing is done under the control of one task, the specified or top task of the terminating tree. This task remains dispatchable throughout the execution of ABEND for the current request.

Purging the Contents Directory for a PRB: A check is made of the RB chain of the selected TCB. If the RB examined 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, a branch is made to the subroutine CDTERM 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 CDTERM to update the contents directory. The search of the RB chain continues.

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) the FREEFWA subroutine 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, CDTERM branches to its DQRBS subroutine to remove the selected PRB from the queue of waiting PRBs. CDTERM 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 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 PRBs in the job step are waiting for the module to be loaded, the CDE and its related areas can be removed without adversely affecting other tasks. Accordingly, ABEND15 branches to

---

[1]Queuing field is RBPGMQ.

the FREEMAIN routine to free the module's storage area (conditionally) and its associated extent list (if it exists). 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: If PRBs for other tasks of the job step are queued, waiting for the module to be loaded, the loading process cannot be stopped without ensuring that the module is loaded under control of one of the waiting PRBs. Otherwise, the tasks whose PRBs are waiting would be permanently nondispatchable, waiting for a resource that is never available. Accordingly, ABEND15 prepares for the refetching of the module by the routines of Contents Supervision. 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. ABEND15 prepares the waiting PRBs for entry to location CDCONTRL by performing the following steps for each queued PRB, using the subroutine QDRBS:

1. Frees the fetch work area, if one has been allocated. (This action has already been done for the selected PRB.)

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 stores 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 creates 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 waiting 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 causes execution as soon as one of the tasks whose RBs have been readied is next dispatched.

After ABEND15 has reinitialized the module request, it frees the program area and extent lists, if they have been acquired. It dequeues and frees the major CDE and any minors from the Job Pack Area Queue (JPAQ). When Contents Supervision is eventually dispatched, it does 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, ABEND15 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, ABEND15 clears the "serially reusable" flag and sets the "nonreusable" flag in the CDE. If the module was not reusable, this flag was already set.

- Instead of freeing the program area, extent list, and the CDE unconditionally if the selected PRB is in control of the module, ABEND15 branches to entry point HKPPROC, 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, ABEND15 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 ABEND15 is that the selected PRB has been removed from the CDE's RB queue of waiting requesters, or the request for the module has been reinitialized and, if necessary, the program area, extent list, and CDE have been freed.

EXITING FROM ABEND15: After ABEND15 has completed processing all PRBs for all tasks of the terminating tree, it exits to ABEND16 to continue task purges.

Processing during ABEND16 (Entry Point IGC0G01C)

ABEND16 performs the following functions:

- Purges IRBs, PRBs, and resident SVRBs.
- Purges the load list.
- Purges dynamically acquired main storage.
- Releases the task control block.
- Provides final processing for the top task of the tree.

ABEND16 purges the remaining resources of the specified task and its descendants. These resources include: IRBs, PRBs, and resident SVRBs; the load lists; and dynamically acquired main storage if exclusively owned. ABEND16 selects the tasks whose resources are to be purged in a manner similar to that of ABEND15.

After ABEND16 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 main storage that the TCB occupies.

As a last function, ABEND16 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.

ABEND16 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 ABEND16, 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.

When control returns 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.

ABEND16 purges resources for and removes the TCB of each lower task in the tree structure beginning with the lowest task and moving upward. 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 ABEND16 exits.

PURGING REQUEST BLOCKS: The resources first purged by ABEND16 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

---

[1]Queuing field is RBPGMQ.

type of RB, see Section 12, "Control Blocks and Tables."

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, passes control to ABEND16's Load List 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 conditionally freed from subpool 250 by a branch to the FREEMAIN routine (address FMBRANCH). If the IRB is a TAXE, (in a system with TSO), the TAIE (terminal attention interruption element) is similarly freed. 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 passes control to ABEND16's Load List Purge routine.

Removing the PRB: If the RB is found to be a PRB, 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 passes control to the Load List Purge.

Purging an SVRB: Since SVRBs for transient routines have already been released by ABEND13 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 branch entry to FREEMAIN, 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.

If the SVRB is the last RB 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 ABEND16 is finished.

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, which purges all resources after the completion of ABEND16. 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 ABEND16 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 communications vector table (CVT) at location CVTEXIT. The SVC 3 instruction, when placed in execution by the Dispatcher, causes supervisor linkage to the Exit routine.

PURGING THE LOAD LIST: 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, is either reenterable or reusable, and rollout has not been invoked, 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, or if the current job step has invoked rollout, 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).

In a Model 65 Multiprocessing System, CDDESTRY tests the TCB, and, if it finds the Storage Reconfiguration completion code, bypasses the freeing of space occupied by the module.

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.

PURGING DYNAMICALLY ACQUIRED MAIN STORAGE: After control is returned from the Load List Purge routine, ABEND16 branches to the Subpool End-of-Task routine (SPEOT), whose entry point is IEAQSPET. In a Model 65 Multiprocessing System, the SPEOT routine is bypassed in the case of a Storage Reconfiguration entry. 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" SPQE 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 pools 251 and 252). Note that only reentrant modules (subpool 252) are left at this time if it is a job-step ABEND. 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.

RELEASING THE TASK CONTROL BLOCK (TCB): After freeing main storage and SPQEs for the selected task, ABEND16 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, ABEND16 sets the "completion" flag (TCBFC) in the selected TCB, to indic-

ate 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 ABEND16.

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 first save area for the TCB is freed conditionally. The "erase" routine then branches to the FREEMAIN routine to free the space occupied by the selected TCB. After the release of a selected TCB, ABEND16 returns control to the "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.

FINAL PROCESSING FOR THE TOP TASK: 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, ABEND16 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, ABEND16 causes linkage to the supervisor Exit routine by issuing an SVC 3 instruction or, if the Storage Reconfiguration bit is set in a Model 65 Multiprocessing System, invokes the Storage Reconfiguration ABEND module, ABEND20, via an XCTL.

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 during ABEND20 (Entry Point IGC0K01C)

In a Model 65 Multiprocessing System, ABEND20 performs Storage Reconfiguration when a solid storage failure has occurred. It is invoked by ABEND16 if the Storage Reconfiguration bit, set by a Recovery Management Support routine, is on in the TCB. ABEND20 has two functions:

- To execute a "dummy freepart" for the job step currently being processed by ABEND. This is done by inspecting the two bit indicator in the FSSEMAP for each 2K block within the boundaries of the current region. (See Section 12, "Control Blocks and Tables" for a description of FSSEMAP.) Only those blocks which are not marked solidly failing are added to the dynamic FBQE chain. Those blocks which are not on the chain become logically unavailable to the system.

- To dequeue and free the partition queue element (PQE) which described the region being processed, and to issue a conditional GETPART for 52K. If storage is available, the new region is chained to the PQE of the already existing job step TCB by GETPART. ABEND20 then issues an SVC 3 instruction which causes linkage to the supervisor exit routine. If storage is not available, this is indicated to the operator by a console message. ABEND20 then issues an unconditional GETPART which places the task in a wait state until the storage becomes available.

PERFORMING DAMAGE ASSESSMENT (DAR)

The Damage Assessment Routine (DAR) receives control from ABEND when one of the following conditions occurs:

- The termination of a task in "must complete" status.
- The termination of a system task.
- An invalid recursion.

DAR alters the environment so that the ABEND routine or system processing can attempt to continue. Under certain circumstances processing cannot continue; DAR then sets all tasks in the job step nondispatchable.

The first module, DAR1, gains control from ABEND1 to test for recursions. If there are no recursions, DAR1 attempts to write a dump of main storage; failing that, a message is issued to the operator, and control passes to the next module.

DAR2 processes tasks in "must complete" status.

DAR3 attempts to reinstate tasks that are not in "must complete" status.

DAR4 is entered for the sole purpose of setting tasks nondispatchable. After completing processing, control passes to the Dispatcher.

Processing during DAR1 (Entry Point IGC0L01C)

Damage Assessment Routine (DAR) Load 1 receives control from ABEND1 in the event of either an invalid ABEND recursion, a system task failure, or the failure of a task in "must complete" status.

DAR1 first tests the TCB for a DAR recursion. In the event of a secondary DAR recursion, indicating that a dump of main storage may have been written but the failing task has not been reinstated, control passes to DAR4.

In the event of a primary DAR recursion, indicating that the DAR function failed during the writing of a dump, a further attempt to write a dump is bypassed. The operator is notified that the dump has failed, and control is passed to the appropriate module, as described later.

If there is no evidence of recursion, DAR1 sets a bit in the TCB of the failing task as a notification in case of future primary recursion.

DAR1 then attempts to write a dump of main storage onto the SYS1.DUMP data set. If SYS1.DUMP is not available, the writing of the dump is bypassed and control passes to the next module. Otherwise, DAR1 issues an SVC 51 to pass control to the SVC DUMP routines.

Exiting From DAR1: After either writing a dump of storage or an operator message as needed, DAR1 determines if further DAR processing is necessary or if control should be returned to ABEND:

- DAR2 - The terminating task is in "must complete" status.
- DAR3 - A system task is abnormally terminating. DAR3 determines if the task is eligible for reinstatement.
- DAR4 - The failing task's TCB is the job-step TCB, or a subtask whose job step is terminating. DAR4 sets the task permanently nondispatchable.
- ABEND1 - In all other cases, ABEND1 gains control, via an XCTL, to convert a subtask ABEND to a job-step ABEND.

Processing during DAR2 (Entry Point IGC0M01C)

DAR2 is entered for tasks in "must complete" status. It searches the QELs to find the major and minor names of resources associated with the failing task. These names become input to the operator, who is requested to reply whether or not the resources are critical. When control is returned from the operator, the task is enabled. DAR2 must disable interruptions before continuing processing.

If the resources are specified as critical, control is passed to DAR4 which sets the failing task and subtasks nondispatchable. They remain in this state for the duration of the current IPL after the operator has been informed that the terminating task failed to be reinstated.

If the resources are not recorded as critical, DAR2 releases the task from "must complete" status. The secondary DAR recursion bit is turned off to allow a chance for true DAR reinstatement. The word "ABEND" is put in the extended save area, and control returns to ABEND1 to continue normal processing.

Processing during DAR3 (Entry Point IGC0N01C)

DAR3 receives control from DAR1 when the latter recognizes that the failing task is either the Master Scheduler or a subtask, and the task is not in "must complete" status. Task reinstatements are accomplished for system tasks as follows:

MASTER SCHEDULER TASK: The resume PSWs of all RBs queued off the failing Master Scheduler Task are pointed to the SVC 3 (EXIT) instruction in the CVT. The resume PSW of the highest level RB is redirected to an entry point within ABTERM (SCEDWAIT), where an SVC 6 (LINK) instruction is executed to provide linkage to the Scheduler Wait routine, IEEVWAIT. The ECBs on which the Scheduler Wait routine waits are cleared. The operator is informed by a WTO message that the task has been reinstated. All information list entries for the task to be reinstated are purged. DAR3 then exits via an SVC 3 to reinstate the Master Scheduler task.

TRANSIENT AREA FETCH TASK: The resume PSWs of all request blocks queued off the failing Transient Area Fetch TCB are pointed to the SVC 3 (EXIT) instruction in the CVT, while the highest level RB is modified to give control at entry point TBNOTFND in the Transient Area Fetch routine (IEAQTR33). A reinstatement message is written to the operator, and all information list entries are purged. DAR3 exits via an SVC 3 to reinstate the task.

SYSTEM ERROR TASK: The error flags in the SVCLIB DCB are cleared. If an RQE exists, the failing task's TCB is scheduled for a "B06" ABEND completion code. The RBOPSW of the SIRB is set to point to an SVC 3 instruction. If an RQE does not exist, only the latter action is taken. DAR3 then issues a WTO instruction to write the reinstatement message. All information list entries pertaining to this task are purged, and exit is via SVC 3.

COMMUNICATIONS TASK: The resume PSWs of all RBs queued off the failing Communications Task TCB are pointed to the SVC 3 (EXIT) instruction in the CVT, while the highest level RB is modified to give control to the Communications Task Wait routine, IEECVCTB. A message is written to the operator, all information list entries for this task are purged, and DAR3 exits via an SVC 3.

I/O RECOVERY MANAGEMENT SUPPORT TASK: The PSWs of all RBs queued off the IORMS TCB are set to point to an SVC 3 instruction in the CVT. The PSW of the top RB is modified to pass control to the entry point IORMSSVC in the RMS SVC. A reinstatement message is written to the operator, and all entries in the information list pertaining to the task to be reinstated are purged. An SVC 3 instruction is issued to begin the reinstatement process.

ROLLOUT/ROLLIN TASK: The rollout/rollin request is turned off and the task on whose behalf the rollout task is operating is scheduled for abnormal termination with a "D0A" completion code. DAR3 then passes control to DAR4 to set the rollout task nondispatchable.

If the failing task is a non-system task, the extended save area of the SVRB is cleared and the valid recursion flag is turned off. DAR3 then exits to ABEND1, via an XCTL, to continue termination processing.

Processing during DAR4 (Entry Point IGC0P01C)

DAR4 is entered solely for the purpose of setting tasks permanently nondispatchable. Entry to DAR4 is from any DAR module. If entry is from DAR1, the terminating task must be set nondispatchable because of the existence of one of the following conditions:

- Task recursed invalidly through DAR.
- Task failed reinstatement.
- Task is a subtask whose job step is abnormally terminated.
- Task is a job-step task which has recursed invalidly through the ABEND module, and is neither a special system task or a task in "must complete" status.

Entry from DAR2 is caused by a terminating task that is in "must complete" status, and the systems operator has indicated that the resources are critical. Entry from DAR3 is caused by a rollout task that must be set nondispatchable.

Upon entry, DAR4 sets a recursion indicater and issues a WTO message to inform the operator that the terminating task

failed to be reinstated. If the indicator was already set when DAR4 was entered, the WTO message is bypassed to avoid a WTO/ABEND recursion loop. The Subsystem Purge routine is entered to perform any necessary subsystem cleanup before the task is set permanently nondispatchable. If the GTF subsystem is the failing task, monitor call interruptions are disabled by the Subsystem Purge routine to prevent further system damage.

The job-step TCB address of each ready task on the TCB queue is then obtained, and the failing task, together with every task in the terminating tree, is set nondispatchable. However, if the Master Scheduler is failing, its subtasks are allowed to continue processing because they are tasks critical to the operation of the system.

If the GTF trace function has been suspended, it is resumed via a HOOK macro instruction. This is done to ensure that tracing is resumed when ABEND has bypassed the ABDUMP function.

The information list entries for each task in the terminating tree are purged. If the failing task is the Master Scheduler, however, only its information list entries are purged.

After setting all tasks in the terminating tree nondispatchable, a branch entry is made to the Dispatcher.

SPECIFYING A TASK ASYNCHRONOUS EXIT ROUTINE

The STAE (Specify Task Abnormal Exit) 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 and the five ABEND/STAE interface routine (ASIR) modules are:

The STAE Service Routine receives control via an SVC 60 when the STAE macro instruction is issued. It checks the validity of the STAE request, and creates, cancels, or modifies a STAE control block (SCB).

ASIR0 receives control from ABEND0 to halt I/O operations that are in progress for the terminating task.

ASIR1 receives control from ASIR0. ASIR1 establishes a work area and schedules the user-written STAE exit routine.

ASIR2 receives control from ASIR1. ASIR2 returns to ABEND processing if a STAE retry routine is not requested. If a STAE retry routine is requested, ASIR2 invokes ASIR3. If the program using STAE is in supervisor mode and requests a STAE retry routine without a purge of the RB chain, ASIR2 invokes ASIR5.

ASIR3 receives control from ASIR2. ASIR3 closes the data sets allocated to the RB of the RBs positioned between the STAE issuer up to and including the RB of the task scheduled for ABEND, and invokes the WTOR Purge routine. If any of the DCBs examined by ASIR3 is using BTAM, QTAM for a line group, BISAM, or QISAM, ASIR3 invokes ASIR4. If none of the above access methods are indicated, ASIR3 invokes ASIR5.

ASIR4 receives control from ASIR3. ASIR4 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 ASIR5.

ASIR5 receives control from ASIR2, ASIR3, or ASIR4. ASIR5 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.

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 parameter list. (See Section 12: "Control Blocks and Tables" for a description of this parameter list.) The last instruction of the macro expansion is an SVC 60, which invokes the STAE service routine.

Processing During the STAE Service Routine (Entry Point IGC00060+)

The STAE service routine 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 or in a STAE (subtask ABEND intercept) retry after a STAE or STAI exit ABEND (the high-order bit if the first byte of TCBNSTAE is on), an error code of eight is placed in register 15, and control is returned to the user. The STAE request is not serviced since STAE exit routine processing is already attempting to deal with an error situation.

The routine then determines whether the STAE macro instruction was issued with the TCB operand (only the ATTACH service routine may use the TCB option). If so, the STAE service routine recognizes a subtask ABEND intercept (STAI) condition and processing continues with the test of exit and parameter list addresses below.

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 twelve is placed in register 15, and control is returned to the user.

The STAE service routine issues a conditional GETMAIN macro instruction to obtain 16 bytes of storage for the SCB. The first word of the extended save area of the STAE 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 caller with a return code of four. If storage is available, the STAE or STAI control block is created and placed on the SCB queue. A STAE SCB is placed immediately before the last STAE SCB created or, if there is no previous STAE SCB, after the last STAI SCB on the queue. A STAI SCB is always placed at the top of the SCB queue and will be propagated to all lower-level subtask TCBs. The address of the previous SCB and its STAE flag byte (zero if this is the first SCB created for the task) are placed in the first word, the address of the STAE exit routine is placed in the second word, and the address of the STAE exit routine parameter list is placed in the third word. For STAE requests, the address of the user's RB is placed in the fourth word; for STAI requests, the address of the new subtask's TCB is placed in the fourth word and the STAI indicator is set. This allows an exit to be scheduled for a parent task when one of its subtasks is scheduled for abnormal termination.

If the XCTL option is requested in the STAE macro instruction (the high order bit of register 1 is on) and the TCB operand was not specified, the STAE service routine turns on the XCTL flag in the TCBNSTAE field. If this is a STAI request (the TCB operand is specified), the XCTL option is ignored.

If the contents of register 0 are 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 eight is placed in register 15, and control is returned to the user since an SCB that does not exist cannot be canceled 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 canceled or overlayed. If the RB addresses are not the same, a return code of sixteen 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 either contains a four or a zero with a STAE exit routine address of zero (the cancel option), the address of the previous SCB and the TCB STAE flag byte (which are contained in the first word of the current SCB) are moved into the TCBNSTAE field. A FREEMAIN macro instruction is then issued to free the storage occupied by the canceled SCB.

If register 0 contains an eight (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 twelve 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, canceled, or overlayed, the STAE service routine returns control to the user with a return code of zero in register 15.

Processing During ASIR0 (Entry Point IGC0R01C)

The ABEND/STAE Interface routine, load 0 receives control from ABEND0 when ABEND0 determines that an SCB exists, or from ASIR2 when no retry is specified. ASIR0 first branches to FREEMAIN to free the PIE if one is pointed to by the TCB. ASIR0 tests the recursion bit of the TCBNSTAE field to determine if a STAE or STAI recursion has occurred. If this is not a STAE or STAI recursion, the current STAE SCB is located, and the STAE recursion flag is set.

If the STAE user is in "must complete" status but is not a supervisor program, control is returned to ABEND0 and abnormal termination processing continues. If the RB address in the current SCB cannot be found on the RB chain, that SCB is canceled and the next SCB on the chain is tested. If none of the RB addresses in any of the SCBs are on the RB chains, ASIR0 returns control to ABEND0. If an SCB is found that

contains an RB address on the RB chain, that SCB is used for normal processing.

If the STAE user is a supervisor routine, ASIR0 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. Asynchronous exits are allowed if the STAE user requested them. Processing continues with the determination of I/O in progress below.

If ASIR0 determines that this is a STAE or STAI recursion or if no STAE SCB is found, ASIR0 searches the SCB queue for a STAI SCB that is indicated as not previously processed by ASIR0. (If no unused STAI SCB is found or if a STAE SCB is found, ASIR0 passes control to ABEND0 to continue processing.)

ASIR0 processing of the valid STAE SCB or unused STAI SCB continues by determining 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 is not zero, indicating that one or more data sets are open, and if a purge was requested by the user, ASIR0 determines what type of purge was requested. If a purge with quiesce was requested, ASIR0 sets the purge-quiesce flag 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, ABEND0 is reentered. ABEND0 tests the purge-quiesce bit and returns control to ASIR0. If the Purge I/O routine did not successfully quiesce I/O, or if a purge with halt was requested by the user, 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.

ASIR0 passes control to ASIR1 for normal exit processing.

Processing During ASIR1 (Entry Point IGC0S01C)

The ABEND/STAE Interface routine, load 1 establishes a work area and schedules the STAE exit routine. ASIR1 attempts to get 176 bytes of storage in subpool 250(0) 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 twelve 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 zero 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 terminating 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 terminating 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, an eight if no I/O operations were in progress at the time of the ABEND, or a sixteen if I/O intervention was bypassed.

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

When STAE exit routine processing is finished, control is returned to ASIR1 unless the STAE exit routine has requested, or encounters, an ABEND situation. ASIR1 prohibits asynchronous interruptions and passes control to ASIR2.

Processing During ASIR2 (Entry Point IGC0T01C)

The ABEND/STAE Interface routine, load 2 first frees the last 76 bytes of the work area obtained by ASIR1 (the user's register save area) via a FREEMAIN macro instruc-

tion. Register 3 is then examined to determine if the user indicated that all other STAI requests are to be ignored and ABEND processing is to be continued (completion code of sixteen). If so, the work area acquired by ASIR1 is freed and control is passed to ABEND0 via the XCTL macro instruction. If not, register 3 is examined to determine if the STAE user indicated that a STAE retry routine be scheduled. If the STAE user has not provided a STAE retry routine (return code=zero), ASIR2 frees the work area obtained by ASIR1. ASIR2 returns control to ASIR0.

If register 3 contains a four or a twelve, the STAE user has requested that a retry routine be scheduled. The address of the STAE retry routine, passed to ASIR2 is register 10, 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 ABEND0. If both addresses are valid, ASIR2 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. ASIR2 then passes control to ASIR3 if retry with RB purge (return code=four) is specified or if a STAI retry (return code=twelve) is specified.

If register 3 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 flag in the TCBNSTAE field, and the user has supplied a work area, ASIR2 frees the work area and returns control to ASIR0. If no work area is provided, ASIR2 sets the RB old PSW to the address of an SVC 13 (ABEND) instruction and issues an SVC 3 (EXIT) instruction. Further STAE processing is not performed since the option of not purging the RB chain is reserved only for a supervisor program.

If the STAE user is in supervisor mode, information is stored in the parameter registers, as described above. ASIR2 then sets the NORBPG flag in the TCBNSTAE field and invokes ASIR5 via the XCTL macro instruction.

Processing During ASIR3 (Entry Point IGC0U01C)

The ABEND/STAE Interface routine, load 3 receives control from ASIR2 when ASIR2 determines that the RB chain must be purged and the STAE retry routine scheduled. Upon entry, ASIR3 stores the contents of the parameter registers 7, 8, 10, 11, 12, and 13 in the ESA.

ASIR3 determines whether the exit routine is a STAI. If it is, the closing of data sets can be bypassed and the WTOR Purge routine can be called immediately because STAI SCBs are always associated with the first RB on the RB queue. If the exit routine is not a STAI, ASIR3 determines if the RB address of the terminating program is the same as the RB address of the program that issued the STAE macro instruction. If the RB addresses are equal, the terminating program is the program that issued STAE, and no intervening RBs exist. In this case also, the routine that closes open data sets can be bypassed. 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 TCBDEB field is not zero and the RB addresses of the STAE issuer and the terminating task are not equal, ASIR3 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, ASIR3 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. ASIR3 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, ASIR3 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 ASIR0, 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. Depend-

ing on the access method used, ASIR3 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, ASIR3 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. If the ISAM/TAM switch is on, indicating that ASIR3 found one or more DCBs using BTAM, QTAM for a line group, BISAM, or QISAM during the DCB search, ASIR3 invokes ASIR4 which repeats the DCB search. If the switch has not been set, ASIR3 invokes ASIR5 via the XCTL macro instruction.

Processing During ASIR4 (Entry Point IGC0V01C)

The ABEND/STAE Interface routine, load 4 receives control from ASIR3 when ASIR3 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 ASIR3 and closes the DCBs 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 ASIR3, 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 ASIR5 via the XCTL macro instruction.

Processing During ASIR5 (Entry Point IGC0W01C)

The ABEND/STAE Interface routine, load 5 receives control from one of the following ASIR modules:

- ASIR2 when STAE or STAI 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.)

- ASIR3 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 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.

If the parent (originating) task is the Master Scheduler, all subtasks associated with the task must be set dispatchable. The ABEND wait flag (TCBABWF) is turned off. This flag was set earlier by ABTERM for all tasks which were to be terminated. 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 Set Status routine is used to set all subtasks dispatchable. After all subtasks have been set dispatchable, the "task select" subroutine is called to select each subtask in the terminating job step in order. The ABDUMP nondispatchability flag (TCBNDUMP) is turned off, and the next subtask is selected.

If the NORBPG flag (set by ASIR2) in the TCBNSTAE field is not on, indicating that the user of STAE requested a purge of the RB chain, ASIR5 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 executes under the RB of the STAE user, a new RB need not be created. The RBOPSW of the STAE user is set to point to the address of the STAE/STAI retry routine.

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. ASIR5 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 ASIR5 exits. The user's register 14 is set to point to an SVC 3 instruction in the CVT.

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 ASIR5 must initialize parameter registers to be passed to the STAE/STAI retry routine. A twelve is placed in register 0, the ABEND completion code is register 1, and the address of the first IOB on the restore chain, or zero, 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.

Before scheduling a STAE retry routine, the current SCB is removed from the queue of SCBs originating in the TCBNSTAE field. The address of the next SCB on the queue, or zero if there is no other SCB, is placed in the TCBNSTAE field or in the STAI SCB that immediately precedes the current SCB on the queue. ASIR5 then issues a FREEMAIN macro instruction to free the removed SCB.

The ABTERM and prevent asynchronous exit flags in the TCB are cleared. The message Information List for Type-1 SVC routines is checked to see if it contains any entries for this TCB. If so, the entries are deleted. ASIR5 issues an SVC 3 instruction and gives control to the Dispatcher which schedules the STAE/STAI retry routine.

MODELS 91 AND 195 DECIMAL SIMULATOR (IEAXDS00) ROUTINE

The Decimal Simulator (IEAXDS00) routine is provided to perform decimal arithmetic instructions since the decimal feature on the Models 91 and 195[1] 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 11-1 indicates the relationship of the Decimal Simulator routines to the operating system. On the Models 91 and 195, 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.

If an error has occurred during the simulated execution of an instruction, the Decimal Simulator routine gives control back to the PFLIH routine indicated in Figure 11-2.

SIMULATOR ORGANIZATION

Figures 11-2 and 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 Model 195 applies to both System/360 and System/370 models.

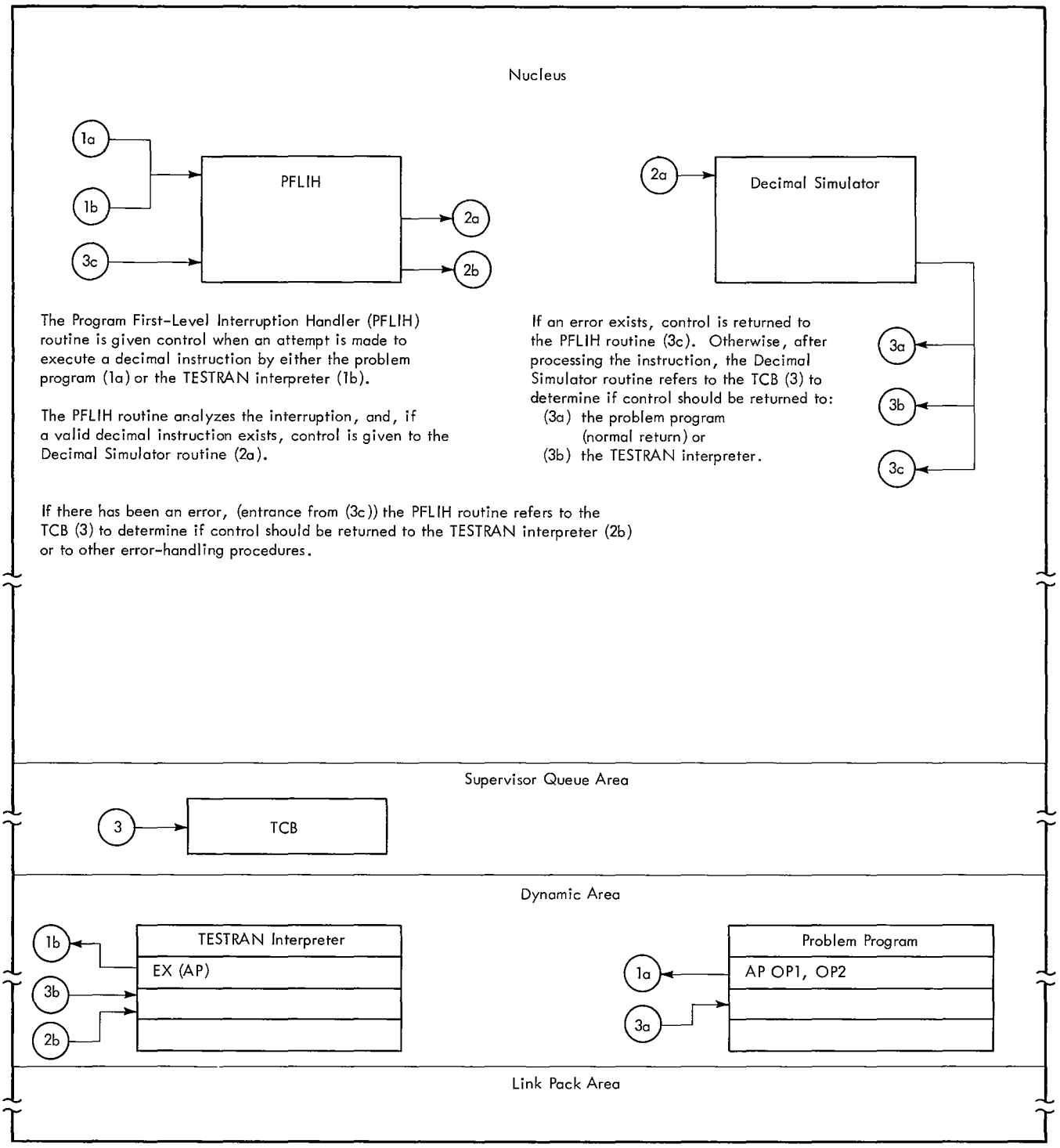


Figure 11-1. Relationship of the Decimal Simulator Routine (IEAXDS00) to the Operating System

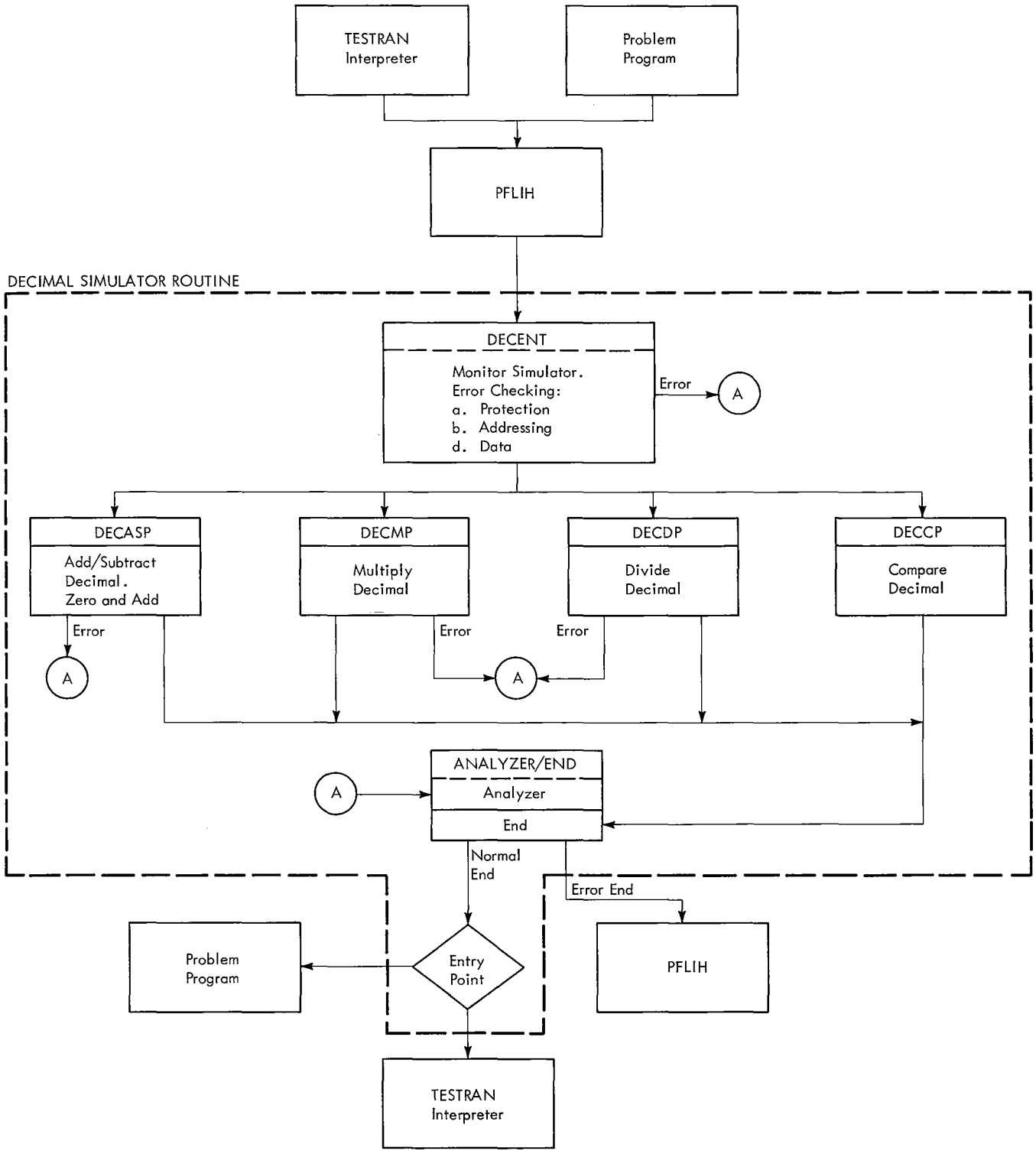


Figure 11-2. Decimal Simulator (IEAXDS00) Routine Organization and Flow of Control

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 • Subtract Decimal • Zero-and-Add Decimal
Multiply Decimal (DECMP)	Simulates execution of the Multiply Decimal instruction.
Divide Decimal (DECDP)	Simulates execution of the Divide Decimal 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.

Figure 11-3. Organization of the Decimal Simulator (IEAXDS00) Routine

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 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.)

   Note: 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 (that is, decimal values 10-15). An invalid sign code results in a data-check exception.

   Note: If the instruction is a ZAP instruction, the sign code check 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 (that is, a condition that would have caused an operation exception to occur) is detected, further checking by the control routine is ter-

minated 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.

PROTECTION CHECK FEATURE: 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.

Simulator Routine for Add, Subtract, Zero-and-Add Decimal Instruction (DECASP)

The DECASP routine is invoked by the Simulator Control (DECENT) 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 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 (that is, 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 occurred 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:

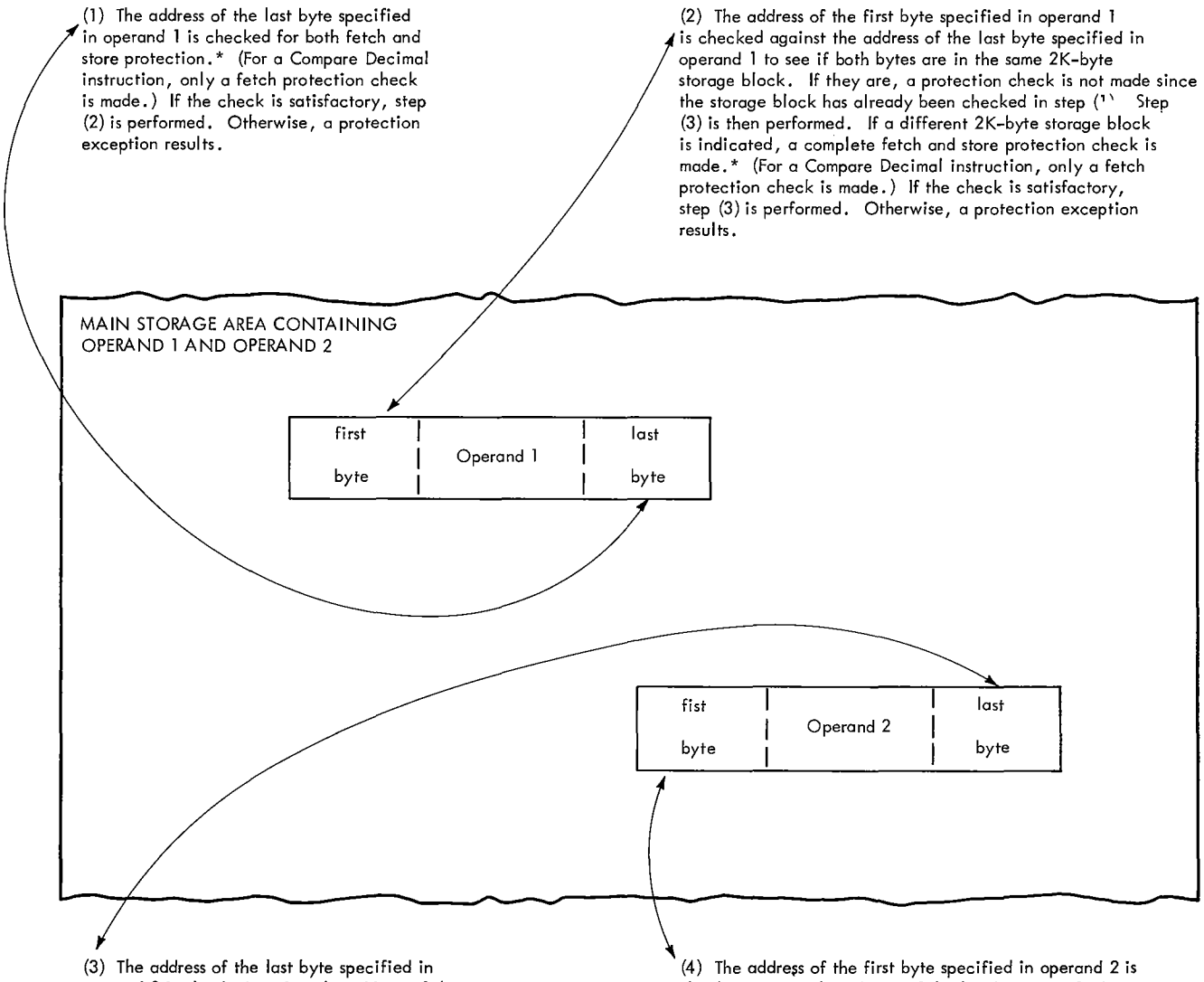
- The actual length of the second operand is greater than eight bytes (that is, instruction length field, denoted by L_2, is greater than seven).

- 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.



* 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

- 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 (that is, 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 (that is, 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.

EXAMPLE OF MULTIPLICATION BY DECIMAL SIMULATOR: 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_{16}). [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 format (0063_{16}). 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). See Figure 11-5.

The pertinent groups of digits from both operands can be arranged in the following manner for describing the next steps.

The formation of the final product proceeds 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').

	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
$000F1ACD_{16}$	$06034AAC_{16}$	$05F2D424_{16}$

Figure 11-5. Example of Multiplication by Decimal Simulation

- 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 Decimal 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 (that is, 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 11-6 illustrate the compare and shifting operations that are performed and the formation of the quotient and remainder digits.

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.

	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 11-6. Example of Division by Decimal Simulator

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 (for example, problem program or TESTRAN) to which control is to be returned.

EXTENDED PRECISION FLOATING POINT SIMULATOR

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

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

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

RELATIONSHIP TO THE OPERATING SYSTEM

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

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

IEAXPSIM Processing

The SPIE user routine passes in register 1 the address of a pointer to a doubleword area. IEAXPSIM determines from the field CVTOPTA (offset X'182.7') in the CVT whether the extended precision floating point feature is supported by CPU hardware. If so, IEAXPSIM moves the name of the module IEAXPDXR into the doubleword area; if not, the module name IEAXPALL is moved into the area. The user routine uses this name to bring the appropriate processing module into main storage.

IEAXPALL Processing

The user SPIE routine passes in register 1 the address of a parameter list containing the address of the PIE, the address of the register save area (containing the contents of the registers at the time of the interruption), a pointer to 400-byte work area, and a pointer to a byte of main storage. If the byte of main storage is not zero, the validity of the low order bit of the result of the DXR instruction has not been ensured. IEAXPALL examines the operation code and register specifications of the failing instruction pointed to by the program check Old PSW. If any of these are invalid, an appropriate code and error indicator are returned to the caller. For the MXD instruction, IEAXPALL also issues a SPIE macro instruction to intercept any interruptions caused by invalid addressing.

IEAXPALL then performs the necessary computation and indicates any exceptional conditions encountered during computation. For AXR and SXR instructions, the condition

code is set in the Old PSW in the PIE. A code in register 15 and an indicator in bits 28-31 of the PSW in the PIE are returned to the caller.

Code	Indicator	Meaning
00	unchanged	Operation successful; no exceptional conditions.
FF	1	Operation code did not specify an extended precision operation to be processed by this module; no simulation attempted.
	4	Protection exception encountered during simulation of an MXD instruction; operation suppressed.
	5	Addressing exception encountered during simulation of an MXD instruction; operation suppressed.
	6	Specification exception encountered during simulation of an MXD instruction; operation suppressed.
	C	Exponent overflow encountered; operation completed.
	D	Exponent underflow encountered; operation completed.
	E	Significance exception encountered; operation completed.
	F	Floating point divide exception encountered; operation is completed.

IEAXPDXR Processing

The user SPIE routine passes a parameter list identical to that passed to IEAXPALL except that the work area is 240 bytes. IEAXPDXR processing for the DXR instruction is identical to that in IEAXPALL except that only the divide instruction is handled. Any other operation code causes control to return to the caller with a code of X'FF' in register 15 and an interruption code of 1 in the PSW.

If IEAXPDXR is used on a CPU without the floating point feature, an 0C1 ABEND code will result when an attempt is made to simulate an extended precision divide instruction.

SYSTEM MANAGEMENT FACILITY

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

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

RECORDING SYSTEM WAIT TIME

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

When a supervisor 10-minute timer interval expires, the Timer SLIH routine reads out the value in the second word of SYSWSAVE, which represents the total system wait time for the 10-minute interval. This value is added to a field in the system management control area, SMCAWAIT + 4.

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

HANDLING TIME/OUTPUT LIMIT EXPIRATION

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

abnormal termination in case of wait time expiration. (The SMF Time/Output Limit Expiration routine is described in Section 6, "Timer Supervision.")

COUNTING REFERENCES TO USER DATA SETS

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

CONTROLLING OUTPUT LIMIT FOR SYSOUT

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

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

RECORDING STORAGE BLOCKS ASSIGNED TO USER PROGRAMS

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

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

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

The information is maintained by the SMF Storage routines, GMSMFCRE and FMSMFCRE. These routines are subroutines of the GETMAIN/FREEMAIN routine, and they are described in Section 5, "Main Storage Supervision."

SMF ROUTINES

A discussion of the two major SMF routines, which perform the functions described above, follows.

SMF Wait Time Collection Routine (IEAQWAIT)

This routine, which resides in the nucleus, records system wait time. It is entered from the first-level interruption handlers whenever an external or input/output interruption occurs.

If the current TCB represents the system wait pseudo task, the system has been in wait condition. (Otherwise, control is simply returned to the calling interruption handler.) The SMF Wait Time Collection routine reads the interval timer. The timer value is compared with the value in the first word of a special save area, SYSWSAVE. The value in SYSWSAVE was placed there by the Dispatcher when the system entered the wait condition. The comparison yields the elapsed system wait time. It is added to the value in the second word of SYSWSAVE, which gives the accumulated system wait time.

After recording the accumulated system wait time, the routine returns control to the interruption handler that called it.

SMF EXCP Counting Routine (IEASMFEX)

This routine, which resides in the nucleus, counts and records the number of EXCPs associated with user data sets. The count is maintained for all data sets recorded in the TIOT of a problem program except SYSABEND, SYSUDUMP, JOBLIB, STEPLIB, TASKLIB, and PGM=*.DD data sets. It includes both direct EXCPs (SVC 0) and indirect EXCPs (those resulting from channel end/abnormal end conditions or programmer-controlled interruptions). The routine counts references to the data sets by the Open, Close, and EOV routines.

Upon entry from IOS the EXCP Counting routine performs the following tests; if any test fails, control returns to the caller.

- The TCBTCT field of the TCB is checked for the existence of a timing control table (TCT).
- The TCTIOTBL field of the TCT is checked for the existence of an I/O extension of the TCT.
- The DCBOFLGS field of the DCB is checked to assure that the DCB is either open or in the process of being opened or closed.

If all tests are successful, the routine searches the TCT I/O table segment of the TCT to find the correct EXCP counter (TCTDCTR). There is a counter for each combination of DCB and UCB. Counts are accumulated on a data set/device basis. (See Figure 11-7.)

When the correct EXCP counter has been found, the routine adds 1 to the counter. Then, if the data set is not SYSOUT, the routine returns to the caller.

If the reference was made to a SYSOUT data set, the routine checks the TCTOUTLM field of the TCT I/O table to determine if an output limit is specified. If an output limit is found, it is compared to the total of the EXCP count fields for each device associated with the data set. If the output limit is not exceeded, or if none was specified, the routine returns to the caller.

Whenever an output limit is exceeded, one of the following actions is taken:

- If exits are not allowed, the program is abnormally terminated with an error code of 722.
- If exits are allowed, the routine creates an IRB/IQE representing the SMF Output Limit Expiration routine (IEATLEXT). The Stage 2 Exit Effector is then entered to schedule the execution of IEATLEXT.

SMF Output Limit Expiration Routine (IEATLEXT)

This routine, which is resident in the nucleus, provides an interface with a user output limit routine (IEFUSO).

The routine (IEATLEXT) receives control from the EXCP Counting routine when the output limit has been exceeded for a SYSOUT data set. The routine passes control to the user output limit routine. It also passes a two-word parameter list containing the address of the JMR (job management region) and the address of the DCB.

The user routine determines whether or not to grant an increase to the output limit. It returns control to the SMF Output Limit Expiration routine with a return code of 0 for no increase, or 4 for an increase. If the return code is 4, register 1 contains the amount of increase to be granted.

If an increase is granted, the SMF Output Limit Expiration routine adds the value of the increase to the output limit specified in the TCT I/O table. If an increase is not granted, the routine schedules an abnormal termination of the program with an error code of 722.

Note: This routine (IEATLEXT) also handles time limit expirations. See Section 6, "Timer Supervision."

TRACING FACILITIES

TRACE TABLE FACILITY

The Trace Table facility is an optional feature that may be specified during system generation. If the Generalized Trace Facility (GTF) is active, the Trace Table facility is inhibited. The facility provides a record of system conditions at the time of the following system events:

- SVC interruptions
- External interruptions
- Program interruptions
- I/O interruptions
- Start I/O operations
- Dispatcher task switches

Each of the above events is recorded in the supervisor trace table that is built in the nucleus by Trace routine IEAQTR. Most entries contain the old PSW, the contents of registers 0, 1, and 15, the current TCB address, and the time of the interruption. When the supervisor trace table is filled, the Trace routine overlays old entries with new entries beginning with the oldest entries. For the format of the supervisor trace table, refer to Section 12, "Control Blocks and Tables."

Trace Routine (IEAQTR)

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

GENERALIZED TRACE FACILITY

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

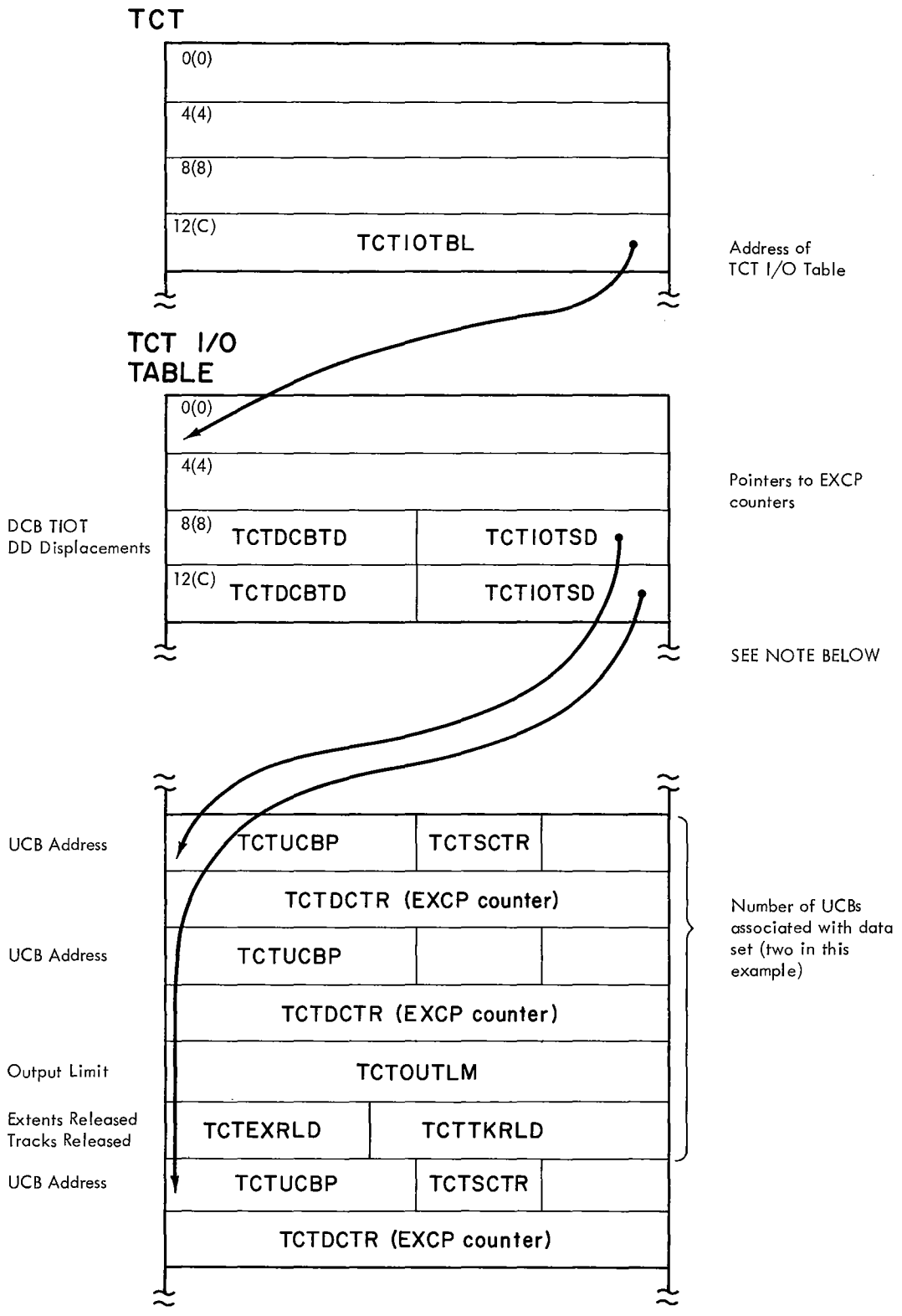
When the internal storage option has been selected, the data recorded for each event class is comparable to that recorded by the supervisor Trace Table facility. The event classes recorded are:

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

When the external storage option has been selected, the data recorded is more comprehensive and may include user supplied trace data. Also, the operator may specify events within an event class, such as: I/O interruptions from specific devices, only certain SVCs, etc. In a multiprocessing system, SSM (Set System Mask) interruptions (Event Class D) are also recorded.

The GTF routines are entered from the interruption handlers and the Dispatcher when a HOOK macro instruction is issued. This instruction specifies the event to be recorded. The termination routines use the HOOK macro instruction to temporarily suspend GTF tracing, or to terminate GTF processing.

For a more comprehensive discussion of the Generalized Trace Facility, refer to the publication Service Aids Logic PLM.

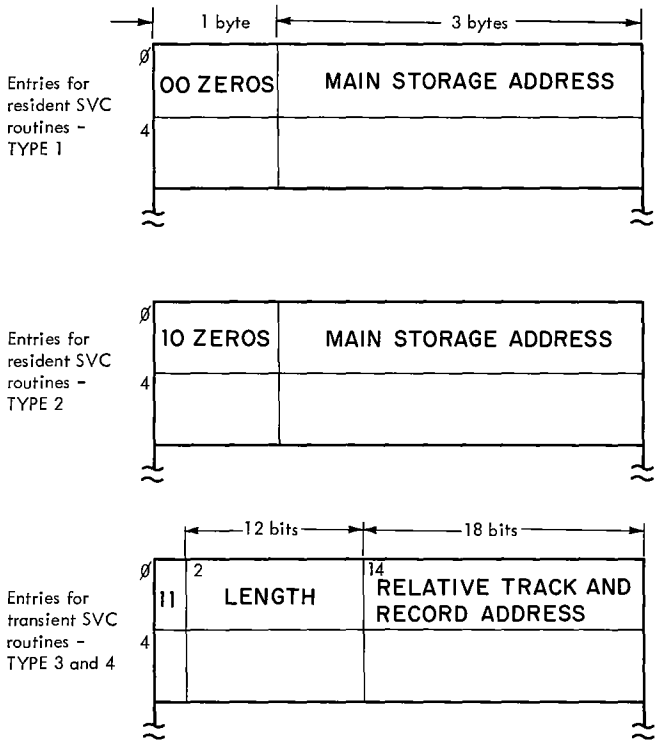


NOTE: The end of the first section of the TCT I/O Table segment is marked by an entry of all zeros. The second section follows immediately.

Figure 11-7. Example of TCT Pointers Used by EXCP Counting Routine

The following control blocks, tables, and related areas are included in this section.

SVC TABLE



Note: '00' flag in two high-order bits indicates resident type 1 routine

'10' flag in two high-order bits indicates resident type 2 routine

'11' flag in two high-order bits indicates transient (non-resident) type 3 and 4 routines

COMMUNICATIONS VECTOR TABLE (CVT)

The Communications 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 hexadecimal 10 during nucleus initialization.

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
-8	-8	2		Reserved.
-6	-6	2	CVTMDL	CPU model number in decimal.
-4	-4	4	CVTRELNO	Release number in EBCDIC.
0	0	4	CVTTCBP	Pointer to addresses for next and current TCB.
4	4	4	CVT0EF00	Address of Stage 2 Exit Effector.
8	8	4	CVTLINK	Address of DCB for SYS1.LINKLIB.
12	C	4	CVTJOB	Address of work queue control blocks.
16	10	4	CVTBUF	Address of buffer for Resident Console Interruption routine.
20	14	4	CVTXAPG	Address of IOS appendage table.
24	18	4	CVT0VL00	Entry-point address of Validity Check routine.
28	1C	4	CVTPCNVT	Entry-point address of routine for converting relative track address to absolute.
32	20	4	CVTPRLTV	Entry-point address of routine for converting absolute track address to relative.
36	24	4	CVTILK1	Address of channel and control unit section in UCB lookup table.
40	28	4	CVTILK2	Address of UCB address list section in UCB lookup table.
44	2C	4	CVTXTLER	Entry-point address to Stage 3 Exit Effector for system error routines.
48	30	4	CVTSYSAD	Address of system residence volume entry in UCB lookup table.
52	34	4	CVTBTERM	Entry-point address of ABTERM routine.
56	38	4	CVTDATE	Current date in packed decimal.
60	3C	4	CVTMSLT	Address of master common area.
64	40	4	CVTZDTAB	Address of I/O device characteristic table.
68	44	4	CVTXITP	Address of Error Interpreter routine.
72	48	4	CVTDAR	Address of I/O control blocks used by DAR; zero if function inoperative.
76	4C	4	CVT0FN00	Reserved.

80	50	2	CVTEXIT	An SVC 3 instruction.
82	52	2	CVTBRET	A BCR 14,15 instruction.
84	54	4	CVTSVDCB	Address of DCB for SYS1.SVCLIB data set.
88	58	4	CVTTPC	Address of pseudo clocks for Timer routine (SHPC first).
92	5C	4	CVTPBLDL	Branch and link entry-point address to BLDL routine.
96	60	4	CVTSJQ	Reserved.
100	64	4	CVTCUCB	Address of table with console UCB addresses.
104	68	4	CVTQTE00	Address of Timer Enqueue routine (IEAQTE00) in Timer SLIH.
108	6C	4	CVTQTD00	Address of Timer Dequeue routine (IEAQTD00) in Timer SLIH.
112	70	4	CVTSTB	Address of I/O device statistics table.
116	74	1	CVTDCB	Operating system configuration.
		...1	CVT4MS1	Uniprocessing.
		.... .1..	CVT4MPS	Multiprocessing.
		xxx. x.xx		Reserved.
117	75	3	CVTDCBA	Address of DCB for SYS1.LOGREC data set.
120	78	4	CVTIOQET	Address of I/O request element table.
124	7C	4	CVTIXAVL	Address of IOS Freelist pointer.
128	80	4	CVTNUCB	Lowest storage address not in nucleus.
132	84	4	CVTFBOSV	Address of Program Fetch routine.
136	88	4	CVT0DS	Entry-point address of Dispatcher.
140	8C	4	CVTILCH	Address of logical channel word table.
144	90	4	CVTIERLC	Address of logical channel error queue.
148	94	4	CVTMSER	Address of Master Scheduler resident data area.
152	98	4	CVT0PT01	Branch entry-point address of Post routine.
156	9C	4	CVTTRMTB	Address of QTAM terminal table.
160	A0	4	CVTHEAD	Address of highest priority TCB on TCB queue.
164	A4	4	CVTMZ00	Highest storage address in machine.
168	A8	4	CVTIEF00	Reserved.
172	AC	4	CVTQOCR	Address of a pointer to the Graphics Job Processor parameter list.
176	B0	4	CVTQMWR	Address of SYSOUT CDA area.
180	B4	2	CVTSNCTR	Data set sequence number.
182	B6	1	CVTOPTA	Flags.

		1...	CVTCCH	CCH option present.
		.1..	CVTAPR	APR present.
		..1.	CVTDDR	DDR present.
		...1	CVTNIP	NIP processing.
		.... .1..	CVTHIAR	Main storage hierarchy support operative.
		.... ..1.	CVTASCII	ASCII option present.
		.... x..x		Reserved.
183	B7	1	CVTOPTB	Flags.
		..1.	CVTTOD	CPU has Time-of-Day clock.
		xx.x xxxx		Reserved.
184	B8	4	CVTQCDSR	Address of CDE search routine.
188	BC	4	CVTQLPAQ	Address of first CDE in LPA queue.
192	C0	4	CVTMPCVT	Address of M65MP secondary CVT.
196	C4	4	CVTSMCA	Address of system management facilities control area.
200	C8	4	CVTABEND	Address of secondary CVT.
204	CC	4	CVTUSER	Field available to user.
208	D0	4	CVTMDLDS	Reserved.
212	D4	2	CVTQABST	An SVC 13 instruction.
214	D6	2	CVTLNKSC	Reserved.
216	D8	4	CVTTSCE	Address of first time-slice control element (TSCE).
220	DC	4	CVTPATCH	Address of 200-byte patch area in the nucleus.
224	E0	4	CVTRMS	Address of RMS work area.
228	E4	1	CVTTSFLG	Time sharing option flags.
		1...	CVTTSRDY	TSO ready and initialized.
		.xxx xxxx		Reserved.
229	E4	3	CVTTSCVB	Address of time sharing CVT.
232	E8	4	CVT0SCR1	Address of Sector Calculation routine for RPS.
236	EC	1	CVTGTFST	GTF status indicator flags.
		00..	CVTGTFIN	GTF inactive.
		01..	CVTGTFSR	GTF starting.
		10..	CVTGTFSP	GTF stopping.
		11..	CVTGTFAC	GTF active.
		..1.	CVTSTATE	GTF in control.
		...1	CVTMODE	GTF in external mode.
		.... 1...	CVTFORM	ABDUMP to format trace data.
		.... .1..	CVTUSR	USR trace.
		.... ..1.	CVTMCTYP	MC instruction valid.
		x		Reserved.
237	ED	3	CVTCMT	Address of class mask table.
240	F0	1	CVTTCMFG	TCAM flags.
		1...	CVTTCRDY	TCAM running.
		.xxx xxxx		Reserved.

241	F1	3	CVTAQAVT	Pointer to address of TCAM vector table.
244	F4	17		Reserved.
261	105	3	CVTPURGA	Address of Subsystem Purge routine.
264	108	5		Reserved.
269	10D	3	CVTQMSGA	Address of type-1 SVC communication area.
272	110	1		Reserved.
273	111	3	CVTDMSRA	Address of Open/Close/EOV supervisory routine.

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
-32	-20	8	TCBFRS0	Save area for floating point register 0.
-24	-18	8	TCBFRS2	Save area for floating point register 2.
-16	-10	8	TCBFRS4	Save area for floating point register 4.
-8	-8	8	TCBFRS6	Save area for floating point register 6.
0	0	1		Zero.
1	1	3	TCBRBP	Address of last RB on RB queue.
4	4	1		Zero.
5	5	3	TCBPIE	Address of PIE.
8	8	1		Zero.
9	9	3	TCBDEB	Address of last DEB on DEB queue.
12	C	1		Zero.
13	D	3	TCBTIO	Address of task I/O table.
16	10	4	TCBCMP	Task completion code.
20	14	1	TCBFLTRN	Flags.
		1...		Both TESTRAN and decimal simulator programs are being used on a Model 91.
		.1..		Suppress taking checkpoints for this step.
		..1.		Job-step TCB: this is a graphics foreground job or the Graphics Job Processor.
		...1		TCB for 7094 Emulator task on Model 85.
		.... 1...	TCBTCP	Time-shared task under control of TEST command processor.
		.... ..1.		OLTEP cleanup.
		.... .x.x		Reserved.
21	15	3	TCBTRN	Address of TESTRAN control area.
24	18	1	TCBNROC	Job-step TCB; rollout eligibility. Initialized by Attach routine from input parameters provided by job-step's initiator. Count increased by ENQ routine; decreased by DEQ routine; tested by TESTSTEP routine of rollout module.
		0000 0000		Job step may be rolled out.
		nonzero		Job step may not be rolled out.
25	19	3	TCBMSS	Address of last SPQE on SPQE queue.
28	1C	1	TCBPFK	Storage protection key.
		xxxx		Storage protection key.
		 0000		Must be zeros.
29	1D	5	TCBFLGS	Flags.
		Byte 0		
		1...	TCBFA	Indicates that abnormal termination, performed by the ABEND routine, is in progress for this task.

.1..	TCBFE	Indicates that normal termination, performed by the EOT routine, is in progress for this task.
..1.	TCBFERA	Indicates that the Erase Phase routine is to be entered when the ABEND routine is again executed for this task.
...1	TCBGTOFM	Indicates that GTF trace is suspended.
.... 1...	TCBPDUMP	Indicates that no abnormal termination dumps are to be taken for any task within the job step. Set in the job-step TCB.
.... .1..	TCBFT	Indicates that this task is currently the top task of a tree of tasks being abnormally terminated.
.... ..1.	TCBFS	Indicates that an abnormal termination dump has been performed for this task.
....1	TCBFX	Prohibits asynchronous exits from being scheduled for this task.
Byte 1		
1...	TCBFOINP	Indicates that the dump data set for the job step is being opened.
.1..	TCBFSTI	Indicates that a job step interval requested by an initiator has expired or the job step has been cancelled by the operator. (Set in the job-step TCB.)
..x.	TCBFRA	Meaningful only in a job-step TCB. Initialized by the Attach routine from input parameters provided by job step's initiator. 0 = job step cannot cause rollout. 1 = job step can cause rollout.
...1	TCBFSMC	Indicates that this task is in "system must complete" status.
.... 1...	TCBFJMC	Indicates that this task is in "job step must complete" status.
.... .1..	TCBFDSOP	Indicates that the ABEND routine has previously opened the dump data set for this job step. (Set in the job-step TCB.)
.... ..1.	TCBFETXR	Indicates that an end-of-task exit (ETXR) routine is to be scheduled for the task that attached this task.
....1	TCBFTS	Indicates member of time-slice group.
Byte 2		
1...	TCBFSM	Indicates that the RB old PSW for all programs executed as part of this task should be set to supervisor state.
.x..	TCBFRI	"Rollout Invoked" flag. Meaningful only in a job-step TCB. 0 = job step has not invoked rollout. 1 = 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.
..1.	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.
...1	TCBSXPK0	RB issued by STAE exit routine was in protection key 0.
.... xxx.		Reserved.
....1	TCBDWSTA	Indicates that this task was detached with the STAE=YES option.
Byte 3		
1...	TCBNDUMP	Indicates that the ABDUMP routine has made this task nondispatchable while it is displaying dynamic queues.
.1..	TCBSER	Indicates that this task is nondispatchable while the SER1 routine is being executed for this task.

		..1.	TCBRQENA	Indicates to the I/O Supervisor that there are no more request queue elements.
		.... 1...	TCBUXNDV	Indicates that this task is nondispatchable due to SMF time limit expiration. User exit routine is attempting to extend time limit.
		...x .x..		Reserved as status bits to indicate nondispatchability.
		.... ..1.		Indicates that this task is nondispatchable because VARY or QUIESCE processing is being done in a multiprocessing system.
		1	TCBONDSP	Indicates that the current task is nondispatchable while the dump data set is being opened for another task in the same job step.
		Byte 4		
		1...	TCBFC	Indicates that this task has terminated, normally or abnormally, and is nondispatchable.
		.1..	TCBABWF	Indicates that this task is nondispatchable because it is to be terminated by the ABEND routine.
		..1.	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.
		...1	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.
		.... 1...	TCBSYS	Indicates that this task is nondispatchable because another task is in "system must complete" status.
		.... .1..	TCBSTP	Indicates that this task is nondispatchable because another task in the same job step is in "step must complete" status.
		.... ..1.	TCBFCD1	Indicates that this task is nondispatchable because it is an initiator task that is waiting for a requested region of main storage.
		1	PNDSP	Indicates that this task has been set nondispatchable. See bytes 173, 174, and 175 for the specific reason.
34	22	1	TCBLMP	Limit priority.
35	23	1	TCBDSP	Dispatching priority.
36	24	1		Zero.
37	25	3	TCBLLS	Address of last load list element in the load list.
40	28	1		Zero.
41	29	3	TCBJLB	Address of job library DCB.
44	2C	1		JPQ purge flag.
		1...		Purge flag.
		.xxx xxxx		Zero.
45	2D	3	TCBJPQ	Address of last CDE for job pack area.
48	30	4	TCBGRS0	Save area for general register 0.
52	34	4	TCBGRS1	Save area for general register 1.

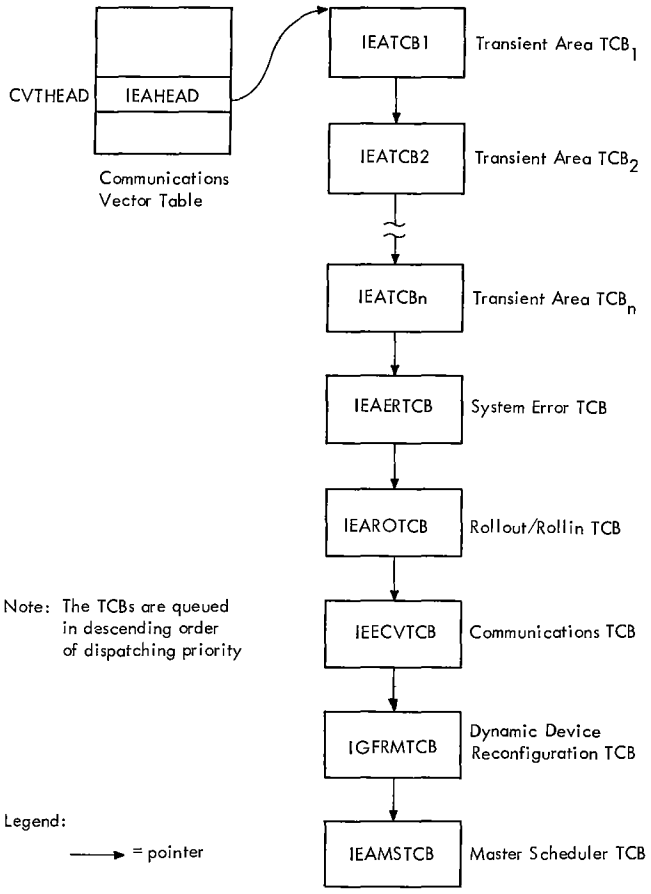
56	38	4	TCBGRS2	Save area for general register 2.
60	3C	4	TCBGRS3	Save area for general register 3.
64	40	4	TCBGRS4	Save area for general register 4.
68	44	4	TCBGRS5	Save area for general register 5.
72	48	4	TCBGRS6	Save area for general register 6.
76	4C	4	TCBGRS7	Save area for general register 7.
80	50	4	TCBGRS8	Save area for general register 8.
84	54	4	TCBGRS9	Save area for general register 9.
88	58	4	TCBGRS10	Save area for general register 10.
92	5C	4	TCBGRS11	Save area for general register 11.
96	60	4	TCBGRS12	Save area for general register 12.
100	64	4	TCBGRS13	Save area for general register 13.
104	68	4	TCBGRS14	Save area for general register 14.
108	6C	4	TCBGRS15	Save area for general register 15.
112	70	1	TCBQEL	Enqueue count.
113	71	3	TCBFSA	Address of first problem program save area for this task.
116	74	1		Zero.
117	75	3	TCBTCB	Address of next TCB on TCB queue. (In rollout TCB, contains address of first transient area TCB.)
120	78	1		Zero.
121	79	3	TCBTME	Address of timer queue element for this task.
124	7C	1		Zero.
125	7D	3	TCBJSTCB	Address of the job-step TCB.
128	80	1		Zero.
129	81	3	TCBNTC	Address of next TCB attached by originating task. (Always 0 in rollout TCB.)
132	84	1		Zero.
133	85	3	TCBOTC	Address of originating or parent TCB.
136	88	1		Zero.
137	89	3	TCBLTC	Address of last TCB on subtask queue. (Always 0 in rollout TCB.)
140	8C	1		Zero.
141	8D	3	TCBIQE	Address of the IQE for scheduling an end-of-task Exit routine.
144	90	1		Zero.

145	91	3	TCBECB	Address of ECB to be posted when this task is complete.
148	94	1	TCBTSFLG	Time sharing flags.
		1...	TCBTSTSK	Indicates a time sharing task.
		.1..	TCBSTPPR	Indicates that this task should be made nondispatchable when it is no longer executing a privileged program.
		..1.	TCBATT	Indicates that a system routine is executing and requires that this task not be interrupted by an attention exit or by the STATUS SVC routine.
		...1	TCBTIOTG	Indicates that TGET/TPUT should be purged due to an attention.
		 xxxx		Reserved.
149	95	1	TCBSTPCT	Number of STATUS starts which must be issued before the task becomes dispatchable.
150	96	1	TCBSLP	Limit priority of time sharing task.
151	97	1	TCBTSDP	Dispatching priority of time sharing task.
152	98	1		Zero.
153	99	3	TCBPQE	Address of dummy PQE-8 (first element on PQE list for job step).
156	9C	1		Zero.
157	9D	3	TCBAQE	List origin of allocated queue elements for this task.
160	A0	1		STAE flags.
		1...		ASIR routines were entered.
		.1..		STAE routine invoked the Purge I/O routine with the quiesce I/O option.
		..1.		Current SCB has the XCTL=YES option.
		...1		SCB was created by a program that is scatter loaded.
		.... 1...		Purge I/O routine did not successfully quiesce I/O, but I/O was halted.
		.... .1..		Program using STAE is in supervisor mode.
		.... ..1.		STAE user requested that a retry routine be scheduled but that the RB chain not be purged.
		1		Retry routine and parameter list addresses are both valid. This bit also set by RMS to indicate that Storage Reconfiguration is necessary because of a solid machine failure.
161	A1	3	TCBNSTAE	Address of STAE control block (SCB).
164	A4	1		Zero.
165	A5	3	TCBTCT	Address of timing control table (SMF only).
168	A8	4	TCBUSER	Field to be used by users.
172	AC	1	TCBDAR	Flags.
		1...	TCBDARP	Primary (valid) DAR recursion. (Always set for current task in a DAR dump.)
		.1..	TCBDARS	Secondary (invalid) DAR recursion. (Set prior to task reinstatement.)
		...1	TCBDARMC	DAR has been entered to handle a valid recursion in "must complete" status through ABEND.
		.... 1...	TCBDAROL	Reserved.

		.... .1..	TCBDARWT	WTO in process for 'Reinstatement Failure' message.
		1	TCBEXSVC	SVC dump is executing for this task.
		..x. ..x.		Reserved.
173	AD	3	TCBNDSP	Flags.
		Byte 0	TCBNDSP1	Flags.
		1...	TNONDISP	DAR has set the task temporarily nondispatchable.
		.1..	PNONDISP	DAR has set the task permanently nondispatchable.
		..1.		RMS or SER has set the task temporarily nondispatchable.
		...1		RMS or SER has set the task permanently nondispatchable.
		.... 1...	TCBDDRND	DDR has set the task (under which allocation is running) temporarily nondispatchable.
		xxx		Reserved.
		Byte 1	TCBNDSP2	Reserved.
		Byte 2	TCBNDSP3	Reserved.
176	B0	4		Reserved.
180	B4	1	TCBRECDE	Flags. Recursion Configuration Flags:
		1xxx xxxx	TCBREC	Valid recursion flag.
		x000 0001	TCBOPEN	OPEN macro instruction has been issued by ABEND for the dump data set for this task.
		x000 0010	TCBCLOSD	CLOSE macro instruction has been issued by ABEND for the direct SYSOUT data set on tape for the job step.
		x000 0011	TCBCLOSE	CLOSE macro instruction has been issued by ABEND for an open data set for this task.
		x000 0100	TCBCLOSF	Force Close routine has been given control by ABEND for graphics jobs.
		x000 0101	TCBGREC	Graphics Debug routine has been given control by ABEND.
		x000 0110		Reserved.
		x000 0111	TCBADUMP	ABDUMP is in progress for this task.
		x000 1000	TCBPTAXE	Purge TAXE routine in EOT given control for current task.
		x000 1001	TCBMESG	Reserved.
		x000 1010	TCBDYNAM	Data management module to check TIOT for dynamic DD entries invalidly marked busy has been given control.
		x000 1011		Reserved.
		x000 1100	TCBQTIP	ABEND is purging TSO Inter-Partition POST requests.
		x000 1101	TCBTCAMP	ABEND is purging TCAM POST requests.
		x000 1110	TCBTCAMR	TCAM Message Control Program (MCP) Reinitialization routine has been given control.
		x000 1111	TCBSAVCD	Save old TCB completion code. (ABEND during ASIR processing.)
		x001 0000	TCBTYP1W	Invalid ABEND recursion from type-1 SVC WTP message.
		x001 0001		Reserved for recursions.
		.		
		.		
		.		
		x010 1111		
				Communication Configuration Flags:
		0011 0000	TCBNOSTA	STAE/STAI not to be honored.
		0011 0001	TCBSTRET	Return from "steal core" routine.
		0011 0010	TCBCONVR	Convert to job step ABEND.
		0011 0011	TCBDARET	Return from DAR.
		0011 0100	TCBTYP1R	Reserved.

		0011 0101	TCBNEWRB	ABEND initiated SVC 13 to XCTL to non-ABEND routine.
		0011 0110 . . . 0011 1111		Reserved for communications.
		0100 0000 . . . 0111 1111		Reserved.
181	B5	3	TCBJSCB	Address of job-step control block.

Positions of Permanent System TCBs on TCB Queue



SUPERVISOR REQUEST BLOCK (SVRB) -- FOR RESIDENT ROUTINE

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	4		Reserved.
4	4	4	RBABOPSW	Bits 32-63 of user's PSW at time system detected abnormal termination condition.
8	8	1	RBWCSA	Wait count save area.
9	9	1	RBSIZE	Size, in doublewords, of RB.
10	A	2	RBSTAB	Status and attribute bits.
		Byte 0		
		xx..	RBFTP	RB type: 00 = PRB. 01 = IRB. 10 = SIRB. 11 = SVRB.
		...1	RBFNSVRB	SVRB for a transient SVC routine.
		.... 1...	RBWAITP	RB waiting on one or more ECBs.
		..x. .xxx		Reserved.
		Byte 1		
		1...	RBTCBNXT	RBLINK field points to a TCB.
		.1..	RBFACTV	IRB or SIRB queued to a TCB.
		..1.	RBATTN	Meaningful only for an IRB.
		...1	RBUSIQE	ATTACH created IRB for a user asynchronous exit routine. IQE must be freed by asynchronous exit routine.
		.... xx..	RBIQETP	Meaningful only for an IRB or SIRB.
		.... ..1.	RBFDYN	RB space can be freed at exit.
		x	RBECBWT	0 = wait for single event or n of n events. 1 = wait for m of n events (where m is less than n).
12	C	1	RBCDFLGS	Contents control flags.
		xxxx		Reserved.
		.... 1...	WAE	Work area exists.
		.... .1..	RBCDSYNC	SYNCH macro instruction issued.
		.... ..1.	RBCDXCTL	XCTL macro instruction issued.
		1	RBCDLD	LOAD macro instruction issued.
13	D	3	RBCDE	Address of CDE used by Link routine when forming a PRB.
16	10	8	RBOPSW	RB old PSW.
24	18	1		Zero.
25	19	3	RBPGMQ	Queue field for serially reusable programs.
28	1C	1	RBWCF	Wait count.
29	1D	3	RBLINK	Address of next RB on RB queue.
32	20	64	RBGRSAVE	General register save area, in the sequence 0 through 15.
96	60	48	RBEXSAVE	Extended save area for SVC routines.

SUPERVISOR REQUEST BLOCK (SVRB) -- FOR NONRESIDENT ROUTINE

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	2	RBTABNO	Displacement of TACT entry.
2	2	2	RBRTLNTH	Length, in bytes, of SVC routine.
4	4	4	RBABOPSW	Bits 32-63 of user's PSW at time system detected abnormal termination condition.
8	8	1	RBWCSA	Wait count save area.
9	9	1	RBSIZE	Size, in doublewords, of RB.
10	A	2	RBSTAB	Status and attribute bits.
		Byte 0		
		xx..	RBFTP	RB type: 00 = PRB. 01 = IRB. 10 = SIRB. 11 = SVRB.
		...1	RBFNSVRB	SVRB for a transient SVC routine.
		.... 1...	RBWAITP	RB waiting on one or more ECBs.
		..x. .xxx		Reserved.
		Byte 1		
		1...	RBTCBNXT	RBLINK field points to a TCB.
		.1..	RBFACTV	IRB or SIRB queued to a TCB.
		..1.	RBATTN	Meaningful only for an IRB.
		...1	RBUSIQE	ATTACH created the IRB for a user asynchronous exit routine. IQE must be freed by asynchronous exit routine.
		.... xx..	RBIQETP	Meaningful only for an IRB or SIRB.
		.... ..1.	RBFDYN	RB space can be freed at exit.
		x	RBECBWT	0 = wait for single event or for n or n events. 1 = wait for m of n events (where m is less than n).
12	C	4	RBSVTON	Address of next RB on transient area queue.
16	10	8	RBOPSW	RB old PSW.
24	18	1	RBTAWCSA	Wait count save area for transient area handling.
25	19	3	RBSVTTR	TTR for SVC routine.
28	1C	1	RBWCF	Wait count.
29	1D	3	RBLINK	Address of next RB on RB queue for task.
32	20	64	RBGRSAVE	General register save area, in the sequence 0 through 15.
96	60	48	RBEXSAVE	Extended save area for SVC routines.

INTERRUPTION REQUEST BLOCK (IRB)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1	RBTMFLD	Timer routine flags.
		1...	RBTMQUE	Timer element not on queue.
		.1..	RBTMTOD	Local TOD option used.
		..xx	RBTMIND1	00 = TUINTVL requested. 01 = BINTVL requested. 10 = reserved. 11 = DECINTVL requested.
		.... 1...	RBTMCMP	Interval is complete.
		.... .1..	RBTMIND2	Midnight supervisory timer element.
		xx	RBTMIND3	00 = task request. 01 = wait request. 10 = supervisory element. 11 = RBAL request.
1	1	3	RBPPSAV	Address of problem program register save area.
4	4	4	RBABOPSW	Bits 32-63 of user's PSW at the time system detected abnormal termination condition.
8	8	1	RBWCSA	Wait count save area.
9	9	1	RBSIZE	Size, in doublewords, of RB.
10	A	2	RBSTAB	Status and attribute bits.
		Byte 0		
		xx..	RBFTP	RB type: 00 = PRB. 01 = IRB. 10 = SIRB. 11 = SVRB.
		...1	RBFNSVRB	SVRB for a transient SVC routine.
		.... 1...	RBWAITP	RB waiting on one or more ECBs.
		..x. .xxx		Reserved.
		Byte 1		
		1...	RBTCBNXT	RBLINK field points to a TCB.
		.1..	RBFACTV	IRB or SIRB queued to a TCB.
		..1.	RBATTN	Exiting program is an attention exit.
		...1	RBUSIQE	ATTACH created the IRB for a user asynchronous exit routine. The IQE must be freed by the asynchronous exit routine.
		.... xx..	RBIQETP	Asynchronous exit queue element type. 00 = RQE 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 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.)

				Note: If rollout is included during system generation, NIP issues the CIRB macro instruction to create and initialize the rollout IRB.
		.... ..1.	RBFDYN	RB space can be freed at exit.
		x	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).
12	C	4	RBEP	Entry-point address.
16	10	8	RBOPSW	Old PSW.
				THREE-BYTE LINK-FIELD SEGMENT
24	18	1	RBUSE	Attach use count. Used only when the IRB schedules an end-of-task exit (ETXR) routine.
25	19	3	RBIQE	List origin for IQEs.
				TWO-BYTE LINK-FIELD SEGMENT
24	18	2		Reserved.
26	1A	2	RBIQE	List origin for RQEs.
28	1C	1	RBWCF	Wait count.
29	1D	3	RBLINK	Address of next RB or RB queue.
32	20	64	RBGRSAVE	General register save area, in the sequence 0-15.
96	60	4	RBNEXAV	Address of next available IQE. The RBNEXAV field and the IQE work space are available only in IRBs for which this work space was requested via a CIRB macro instruction.
100	64	Variable		IQE work space.

SYSTEM INTERRUPTION REQUEST BLOCK (SIRB)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	8	RBEXRTNM	1-8 character name of error exit routine. First 4 characters are IGE0. Last 4 are unpacked decimal characters, or RBABOPSW (bits 32-63 of user's PSW at time system detected abnormal termination condition).
8	8	1	RBWCSA	Wait count save area.
9	9	1	RBSIZE	Size, in doublewords, of RB.
10	A	2	RBSTAB	Status and attribute bits.
		Byte 0		
		xx..	RBFTP	RB type: 00 = PRB. 01 = IRB. 10 = SIRB. 11 = SVRB.
		...1	RBFNSVRB	SVRB for a transient SVC routine.
		.... 1...	RBWAITP	RB waiting on one or more ECBs.
		..x. .xxx		Reserved.
		Byte 1		
		1...	RBTCBNXT	RBLINK field points to a TCB.
		.1..	RBFACTV	IRB or SIRB queued to a TCB.
		..1.	RBATTN	Meaningful only for an IRB.
		...1	RBUSIQE	ATTACH created IRB for a user asynchronous exit routine. IQE must be freed by asynchronous exit routine.
		.... xx..	RBIQETP	Asynchronous exit queue element type. 00 = RQE 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 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 to be queued to "next available" list by 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.) Note: If rollout is included during system generation, NIP issues the CIRB macro instruction to create and initialize the rollout IRB.
		.... ..1.	RBFDYN	RB space can be freed at exit.
		x	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).
12	C	4	RBEP	Entry-point address.
16	10	8	RBOPSW	RB old PSW.
24	18	2		Reserved.

26	1A	2	RBIQE	List origin for RQEs.
28	1C	1	RBWCF	Wait count.
29	1D	3	RBLINK	Address of next RB on RB queue.
32	20	64	RBGRSAVE	General register save area, in the sequence 0 through 15.

PROGRAM REQUEST BLOCK (PRB)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	4		Reserved.
4	4	4	RBABOPSW	Bits 32-63 of user's PSW at time system detected abnormal termination condition.
8	8	1	RBWCSA	Wait count save area.
9	9	1	RBSIZE	Size, in doublewords, of RB.
10	A	2	RBSTAB	Status and attribute bits.
		Byte 0		
		xx..	RBFTP	RB type: 00 = PRB. 01 = IRB. 10 = SIRB. 11 = SVRB.
		...1	RBFNSVRB	SVRB for a transient SVC routine.
		.... 1...	RBWAITP	RB waiting on one or more ECBs.
		..x. .xxx		Reserved.
		Byte 1		
		1...	RBTCBNXT	RBLINK field points to a TCB.
		.1..	RBFACTV	IRB or SIRB is queued to a TCB.
		..1.	RBATTN	Meaningful only for IRB.
		...1	RBUSIQE	ATTACH created the IRB for a user asynchronous exit routine. IQE must be freed by the asynchronous exit routine.
		.... xx..	RBIQETP	Meaningful only with an IRB or SIRB.
		.... ..1.	RBFDYN	RB space can be freed at exit.
		x	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).
12	C	1	RBCDFLGS	Contents control flags.
		xxxx		Reserved.
		.... 1...	WAE	Work area exists.
		.... .1..	RBCDSYNC	SYNCH macro instruction issued.
		.... ..1.	RBCDXCTL	XCTL macro instruction issued.
		1	RBCDLD	LOAD macro instruction issued.
13	D	3	RBCDE	Address of contents directory entry.
16	10	8	RBOPSW	Old PSW.
24	18	1		Zero.
25	19	3	RBPGMQ	Queue field for serially reusable programs.
28	1C	1	RBWCF	Wait count.
29	1D	3	RBLINK	Address of next RB on RB queue.

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		Reserved	0	TCB found in RQE	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		Reserved	0	Reserved	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	New 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 15	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)

The TACT (entry point IEAQTAQ) consists of a four-word entry for each transient area block (TAB) in the system. The first entry is preceeded by a two-word prefix. Each entry has the format described below:

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
-8	-8	4		Request queue pointer.
-4	-4	4		Number of TACT entries.
Entry 1				
0	0	1		Flags.
		.1..		TAB is being loaded.
		..1.		TAB is free (unoccupied).
		0000 0000		TAB is being used.
1	1	3		Address of associated TCB.
4	4	4		SVRB chain of users overlayed by a higher priority task.
8	8	1		SVRB chain of requestors who could not find a usable TAB.
9	9	3		Track address in SVC library of the routine currently in the TAB. This address used to identify routine.
12	C	1		Count of recycles of BLDL and FETCH operations after report of permanent I/O error by BLDL and FETCH - area used by Transient Area Fetch.
13	D	3		Address of transient area fetch TCB under whose control routines are fetched to the TAB.

Entry 2 begins at location 16(10).

PROGRAM INTERRUPTION ELEMENT (PIE)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1		The PIE must begin on a doubleword boundary.
		x...		First-time logic switch. If set to one, indicates that the task cannot accept further program interruptions. (This bit is set whenever a user PI exit routine is entered. It is reset by the SVC Exit routine.)
		.xxx xxxx		Reserved.
1	1	3	PIEPICA	Address of current PICA.
4	4	8	PIEPSW	Program interruption old PSW stored at time of program interruption.
12	C	4	PIEGR14	Save area for register 14.
16	10	4	PIEGR15	Save area for register 15.
20	14	4	PIEGR0	Save area for register 0.
24	18	4	PIEGR1	Save area for register 1.
28	1C	4	PIEGR2	Save area for register 2.

PROGRAM INTERRUPTION CONTROL AREA (PICA)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1		The PICA must begin on a fullword boundary.
		xxxx		Zero.
		 xxxx	PICAPRMK	Program mask to be used in the PSW when the programs of the task are executing.
1	1	3	PICAEXIT	Address of the user's program interruption exit routine to be given control when a program interruption of specified type occurs.
4	4	2		Program interruption types.
		Byte 0, bit 0		Reserved.
		Remaining bits	PICAITMK	Mask that indicates on which program interruption types the exit routine is to be used. The bits are numbered 1 through 15, left to right. A bit set to one indicates user interest in that type.

STAE CONTROL BLOCK (SCB)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1		TCB STAE flag byte for the previous SCB for this task, or zero if this is the first SCB for this task.
1	1	3		Address of previous SCB for this task, or zero if this is the first SCB created for this task.
4	4	4		Address of user-written STAE exit routine as specified in STAE macro instruction.
8	8	1		Flags.
		xxxx x...		Reserved.
		.... .1..		Allow asynchronous interruptions.
		xx		00 = quiesce I/O. 01 = halt I/O. 10 = bypass I/O intervention.
9	9	3		Address of parameter list to be passed to STAE exit routine as specified in STAE macro instruction.
12	C	1		STAE flags.
		1...		SCB will not be canceled by Exit routine when XCTL is issued.
		.1..		ISAM/TAM switch.
		..1.		STAI SCB.
		...1		SCB previously used.
		.... xxx.		Reserved.
		1		STAI issuer is in supervisor mode.
13	D	3		Address of RB of task issuing STAE macro instruction.

STAE PARAMETER LIST

The STAE parameter list is pointed to by the STAE control block.

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1		Flags.
		1...		STAI parameter list.
		.xxx x...		Reserved.
		.... .1..		Allow asynchronous interruptions.
		xx		00 = quiesce I/O. 01 = halt I/O. 10 = bypass I/O processing.
1	1	3		Address of STAE or STAI exit routine.
4	4	4		Address of STAE exit routine parameter list, or zero.
8	8	4		Address of attached TCB.

EVENT CONTROL BLOCK (ECB)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	Bits 0-1 (W C)		W = bit 0; C = bit 1. W = wait flag; C = completion flag.
		0 0		Event has not been awaited and has not been posted complete. Bits 2-31 may contain meaningless information.
		0 1		Event has been posted complete, but it has not yet been awaited. Bits 2-31 contain the completion code in the high-order positions; zeros in the low-order positions.
		1 0		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		Task waiting for an event abnormally terminated before the event was posted. ABEND will purge the ECB. Bits 0-7 are zero, and bits 8-31 contain the address of the RB under which the WAIT macro instruction was issued.
		Bits 2-7		Completion code.
		111111		Normal completion (no errors).
		000001		I/O permanent error code.
		000010		Extent permanent error 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. Either the I/O request has not been started, or it has been purged.
		001111		Error could not be retried. Home address and/or R0 could not be read during error recovery procedures.
		Bits 8-31		Address of the RB under which the WAIT macro instruction was issued.

PARAMETER LIST ELEMENT (FOR THE ENQ/DEQ ROUTINES)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1	LISTEND	Last element in parameter list. Last element must have X'FF' in field; all other elements have any other value.
1	1	1	LMINOR	Length of minor name whose address is at offset 8, or zero. If 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, length byte does not include its own length.
2	2	1	PARMCDS	ENQ/DEQ parameters.
		x...		0 = exclusive request. 1 = shared request.
		.x..		0 = minor name known only to job step. 1 = scope of minor name SYSTEM.
		..1.		"Set Must Complete" = SYSTEM.
		...1		"Set Must Complete" = STEP.
		.... x...		Reserved.
		xxx		111 : RET = TEST. 011 : RET = USE. 001 : RET = HAVE. 000 : RET = NONE.
3	3	1		Return code field for codes returned to the issuer of the ENQ or DEQ macro instruction.
4	4	4		Address of major resource name (Qname).
8	8	4		Address of minor resource name (Rname).

MAJOR QUEUE CONTROL BLOCK (QCB)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	4		Address of next major QCB (if last, equals zero).
4	4	4		Address of previous major QCB (if first, equals IEAQQCB).
8	8	4		Address of first minor QCB on queue of minors.
12	C	4		Major QCB name (first four characters).
16	10	4		Major QCB name (last four characters).

MINOR QUEUE CONTROL BLOCK (QCB)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	4		Address of first QEL on QEL queue.
4	4	4		Address of previous minor QCB (if first, equals major QCB).
8	8	4		Address of next minor QCB (if last, equals zero).
12	C	1		Length of QCB name.
13	D	1	QCBPKF	'FF' = name known to entire system. '00', '10', '20', '30', or 'F0' = protection key of TCB under which request was enqueued. Name known only to job step.
14	E	Variable		Minor QCB name (variable in length from 1-255 characters).

QUEUE ELEMENT (QEL)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1	SMC	Request status represented by QEL.
		0010 0000		"System must complete" request.
		0001 0000		"Step must complete" request.
		0000 0000		"Must complete" status not requested.
1	1	3		Address of next QEL (zero if last QEL).
4	4	1	CODE	
		x...		0 = exclusive request. 1 = shared request.
		.1..		If shared DASD is included in system, indicates that a UCB address appears at byte 12 of QEL, and that QEL is associated with a RESERVE rather than an ENQ macro instruction.
5	5	3		Address of previous QEL on queue. In first QEL, address of minor QCB.
8	8	1	QELTJID1	First half of TSO user ID when user not in main storage (high order bit set to 1 indicates presence of TJID).
9	9	3	QELTCB	Address of TCB under which ENQ macro instruction issued.
12	C	1	QELTJID2	Second half of TSO user ID when user not in main storage.
13	D	3	QELSVRB	Address of 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 field contains the address of the UCB of the direct access device on which the requested resource resides.

INTERRUPTION QUEUE ELEMENT (IQE)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1		Reserved.
1	1	3	IQELNK	Address of the next IQE on the IQE queue.
4	4	4	IQEPARAM	Parameter list to be passed to the asynchronous exit routine.
8	8	1		Reserved.
9	9	3	IQEIRB	Address of the IRB that is to be scheduled because of this request.
12	C	1		Reserved.
13	D	3	IQETCB	Address of the TCB with which this request is associated.

The following constitutes the optional Rollout/Rollin Parameter List:

16	10	4	RPLTCB	Address of the TCB for the task requiring or releasing an extension to a region.
20	14	1		Reserved.
21	15	3	RPLSZPQE	Size of region requested (rollout request), or address of PQE describing area (rollin request).

MESSAGE INFORMATION LIST (FOR TYPE-1 SVC ROUTINES)

The Message Information List contains information stored by type-1 SVC routines. Each entry consists of seven words; the first entry is preceeded by a one-word prefix. The format of each entry is described below:

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
-4	-4	4		Address of end of list.
Entry 1				
0	0	4	INFTCB	Address of the TCB for which the information is pertinent.
4	4	4	INFBADDR	Return address (branch address) of the calling routine from register 14 if entry to the type-1 SVC routine was via a branch instruction.
8	8	1	INFRCL	
		xxx.		Reason code for messages having multiple causes (0 = no reason code).
		...x xxxx		Number of bytes of variable data contained in the INFVAR field.
9	9	1	INFFLG	
		1...		Signifies that the INFBADDR field contains a valid branch address.
		.xxx xxxx		Reserved.
10	A	2	INFCC	System completion code.
12	C	16	INFVAR	Information to be provided to the user. Up to four words of data may be included.

Entry 2 begins at location 28(1C).

REQUEST QUEUE ELEMENT (RQE)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	2	RQELNK	Pointer to next RQE on RQE queue.
2	2	2	RQEUCB	Pointer to UCB. If the low-order bit is 1, the RQE represents a request for a system error routine that operates under an SIRB.
4	4	1		Contains 'FF' if RQE is on free list; otherwise, contains zero.
5	5	3	RQEIOB	Address of associated IOB.
8	8	1	RQEPRI (RQETJID1)	First half of TSO user ID (when user not in main storage).
9	9	3	RQEDEB	Address of associated DEB.
12	C	1		Protection key of requestor's task or RQETJID2 (second half of TSO user ID).
13	D	3	RQETCB	Address of TCB with which I/O request is associated.

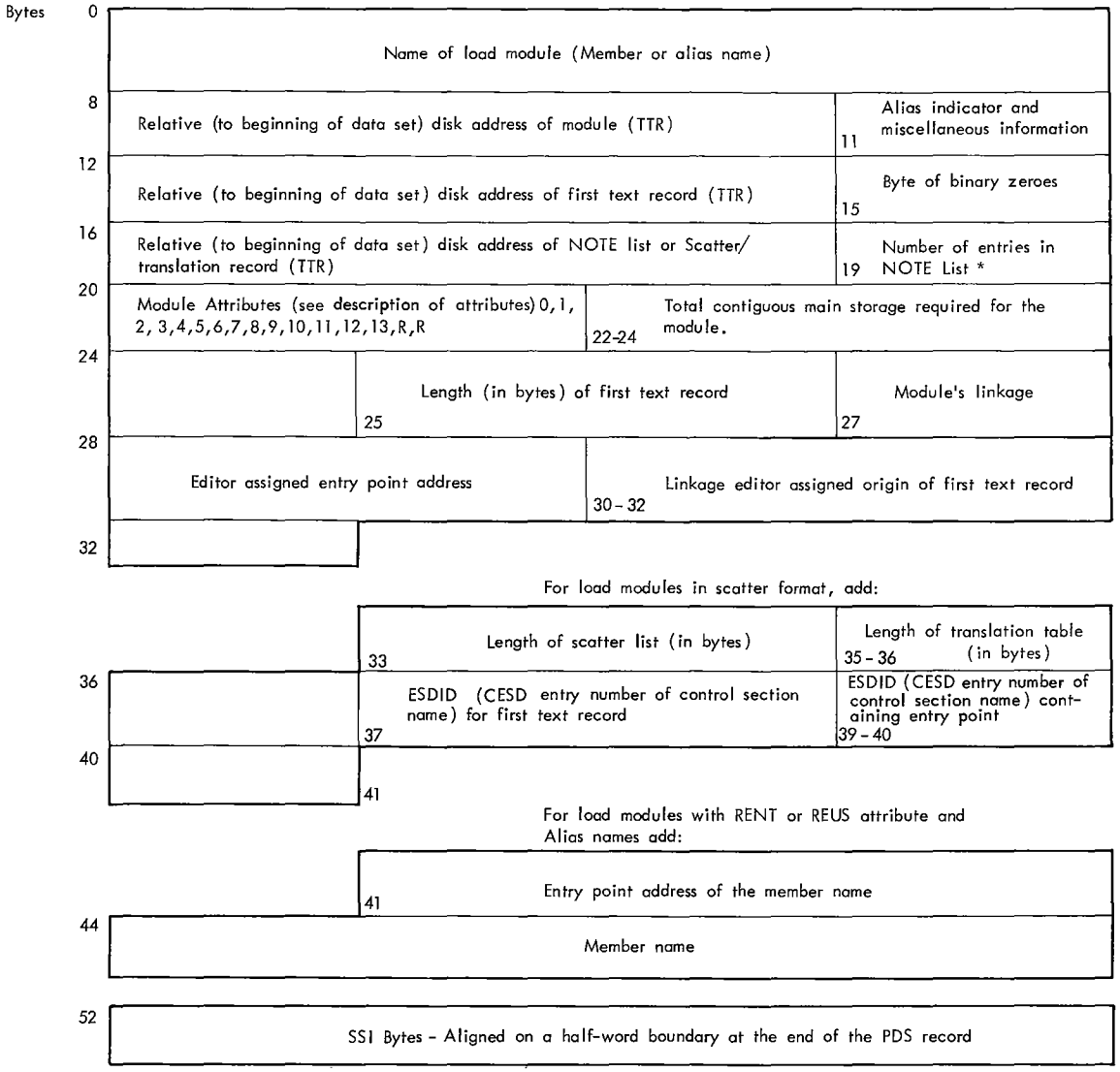
CONTENTS DIRECTORY ELEMENT (CDE)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1	CDATTR	Attribute field.
		1...	NIP	Module was loaded by NIP.
		.1..	NIC	Module is in process of being loaded.
		..1.	REN	Module is reenterable.
		...1	SER	Module is serially reusable.
		.... 1...	NFN	Module may not be reused. (Meaningless if REN or SER is set.)
		.... .1..	MIN	This is a minor CDE.
		.... ..x.	JPA	0 = module in subpool 252. 1 = module in subpool 251.
		1	NLR	Module is not loadable-only.
1	1	3	CDCHAIN	Address of next CDE in queue (either the JPACQ or the LPACQ).
4	4	1	CDROLL	Reserved.
5	5	3	CDRBP	RB address. If the module is reenterable, this field contains the address of the last RB that requested 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.
8	8	8	CDNAME	Module name, alias name, or a name that has been identified via an IDENTIFY macro instruction.
16	10	1	CDUSE	Use/responsibility count - the number of outstanding requests for the module's use.
17	11	3	CDENTPT	Entry-point address.
20	14	1	CDATTR2	Second attribute field.
		1...	SPZ	Module is loaded in subpool 0 by the loader, and there is no backup copy. The second entry in the extent list is the address of a condensed symbol table.
		.1..	REL	Module is inactive and may be released by the GETMAIN routine (CDPURGE) subroutine.
		..1.	XLE	Extent list has been built for the module.
		...1	RLC	CDE contains a minor entry-point address that has been relocated by the Program Fetch routine.
		.... 1...	REFR	Module is refreshable.
		.... .1..	CDOLY	Module is in overlay form.
		xx		Reserved.
21	15	3	CDXLMJP	Extent list address, or major CDE address if this CDE is a minor. (If this CDE is a minor, MIN field is also set.)

LOAD LIST ELEMENT (LLE)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1		Zero.
1	1	3	LLCHAIN	Address of first byte of the next element on the load list.
4	4	1	LLCOUNT	Responsibility count - number of requests for the module, via the LOAD macro instruction.
5	5	3	LLCDPTR	Address of the CDE for the module.

PARTITIONED DATA SET DIRECTORY ENTRY



Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	8		Load module name.
8	8	3		Relative disk address of module (TTR).
11	B	1		Alias indicator and miscellaneous information.
		x...		0 = no alias indicator. 1 = alias indicator.
		.xx.		Number of relative disk addresses in user data field.
		...x xxxx		Length of user data field in halfwords.
12	C	3		Relative disk address of first text record (TTR).
15	F	1		Byte of binary zeros.
16	10	3		Relative disk address of NOTE list or scatter/ translation record (TTR).
19	13	1		Number of entries in NOTE list.
20	14	2		Module attributes.
		Byte 0		
		x...		RENT: 0 = not reenterable. 1 = reenterable.
		.x...		REUS: 0 = not reusable. 1 = reusable.
		..x.		OVLY: 0 = not an overlay module. 1 = overlay module.
		...x		TEST: 0 = not under test. 1 = under test.
		.... x...		LOAD: 0 = not loadable-only. 1 = loadable-only. (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.)
		.... .x..		Format: 0 = block format. 1 = scatter format.
		.... ..x.		Executable: 0 = not executable. 1 = executable.
		x		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.
		Byte 1		
		x...		Compatability: 0 = module can be processed by all levels of linkage editor. 1 = module cannot be reprocessed by Linkage Editor-E.
		.x..		Format: 0 = linkage editor assigned origin of first text record is not zero. 1 = linkage editor assigned origin of first text record is zero.
		..x.		Format: 0 = linkage editor assigned entry point is not zero. 1 = linkage editor assigned entry point is zero.
		...x		Format: 0 = module contains RLD record(s). 1 = module does not contain an RLD

			record.
		.... x...	Editability: 0 = module can be reprocessed by linkage editor. 1 = module cannot be reprocessed by linkage editor.
		.... .x..	Format: 0 = module does not contain TESTRAN symbol records. 1 = module contains TESTRAN symbol records.
		.... ..x.	Reserved.
		x	Refreshability: 0 = module is not refreshable. 1 = module is refreshable.
22	16	3	Total contiguous main storage required for the module.
25	19	2	Length (in bytes) of first text record.
27	1B	3	Module's linkage editor assigned entry-point address.
30	1E	3	Linkage editor assigned origin of first text record.

For load modules in scatter format, add:

33	21	2	Length (in bytes) of scatter list.
35	23	2	Length (in bytes) of translation table.
37	25	2	ESDID for first text record.
39	27	2	ESDID containing entry point.

For load modules with RENT or REUS attribute and alias name, add:

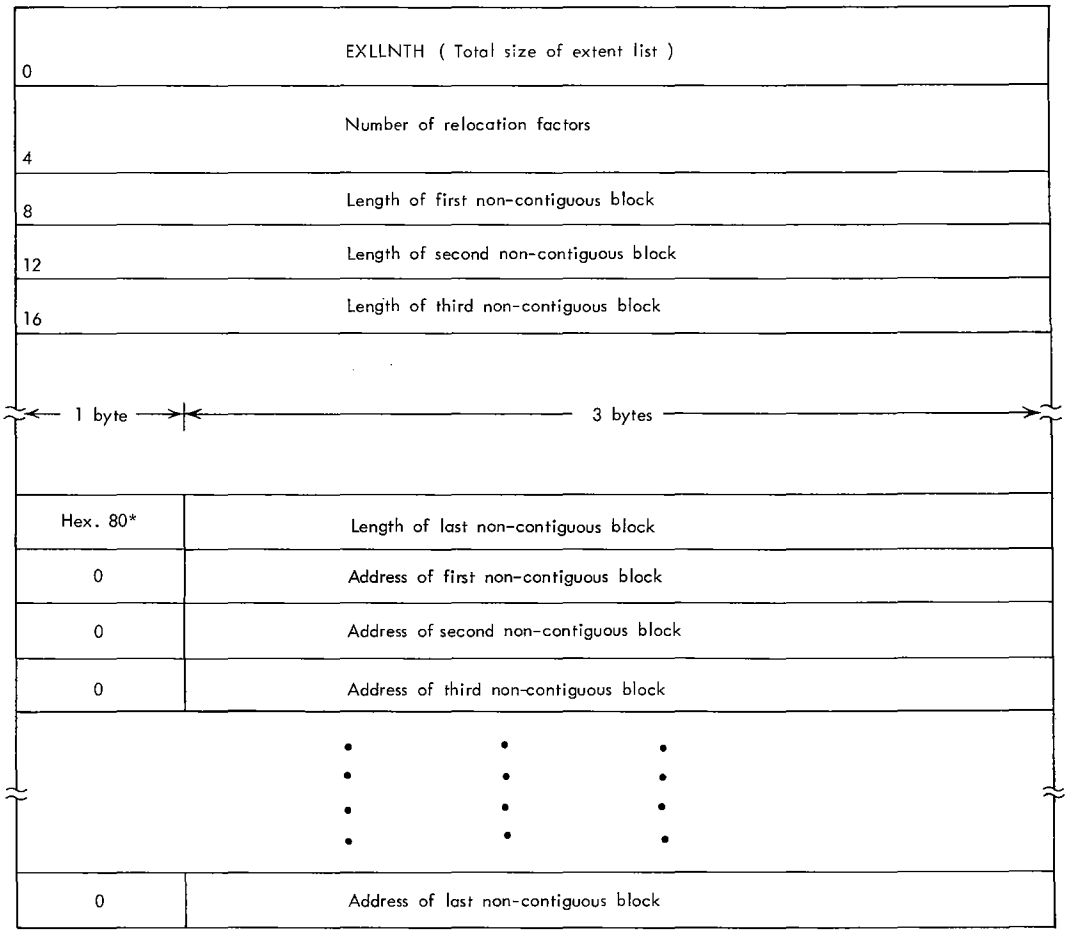
41	29	3	Entry-point address of member name.
44	2C	8	Member name.
52	34	4	SSI bytes - aligned on a halfword boundary at the end of the PDS record.

The following list contains PDS directory record sizes.

Block format	34 bytes (when rounded to a halfword boundary).
Block format with alias name	44 bytes.
Scatter format	42 bytes.
Scatter format with alias name	52 bytes.

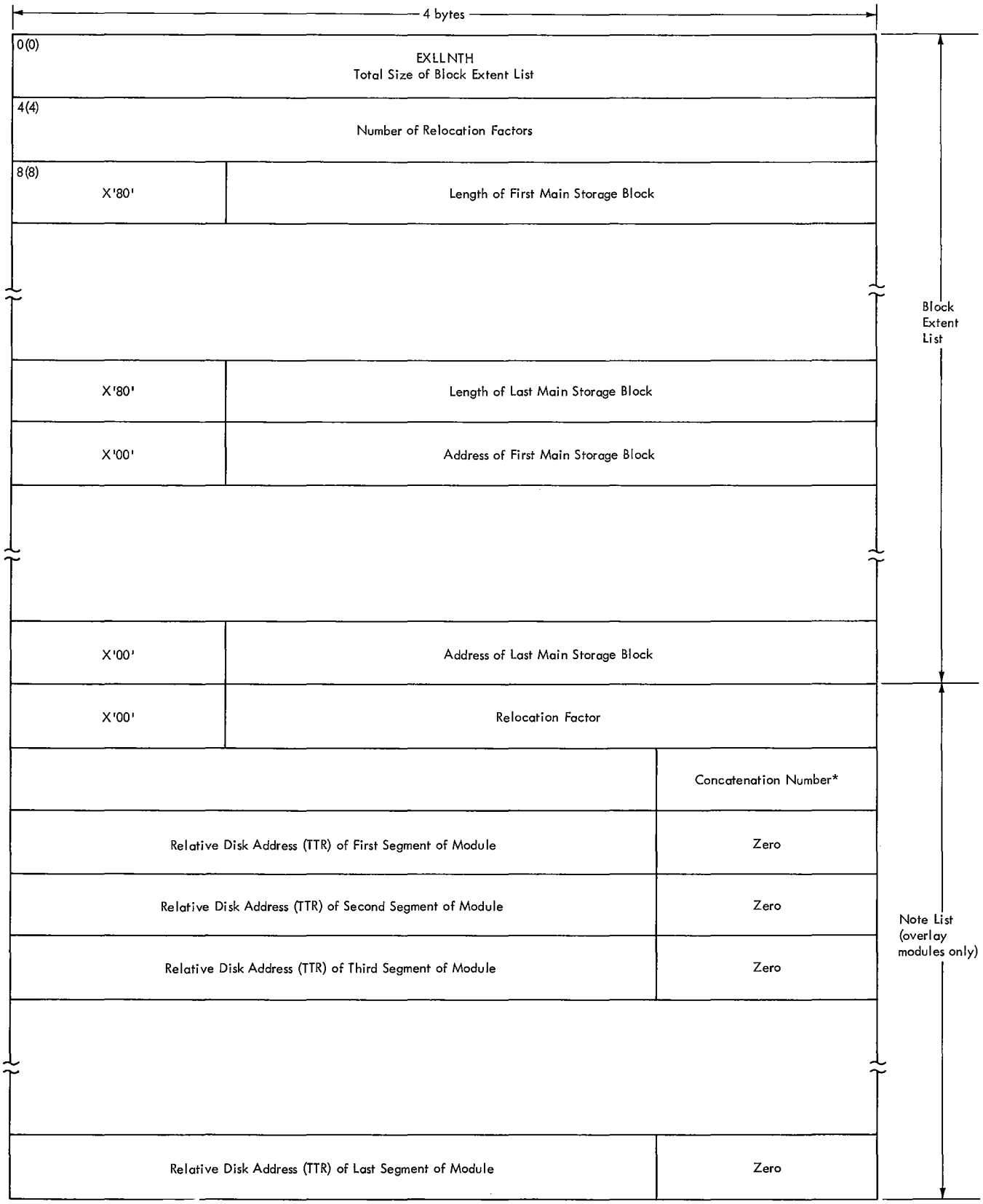
Note: For SSI, add 4 bytes to sizes given above.

SCATTER EXTENT LIST



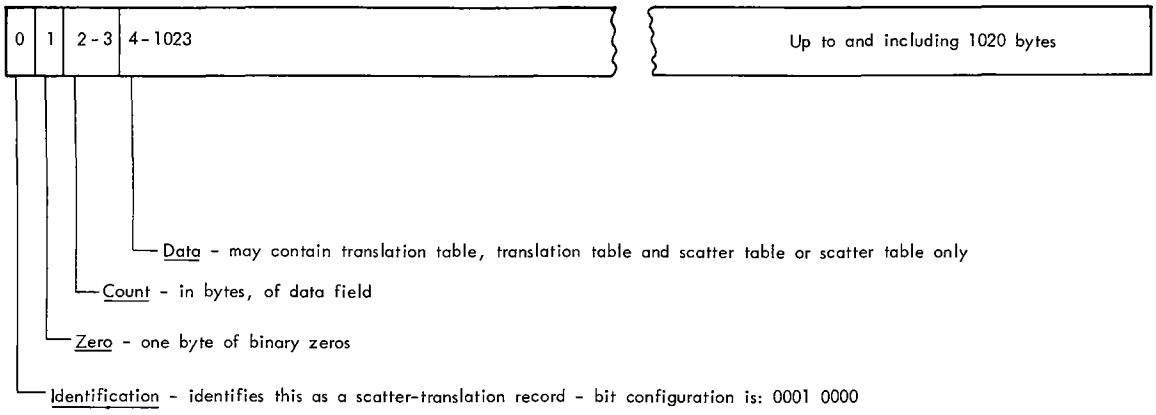
* Indicates the end of the immediately preceding length-of-block list. Used by the GETMAIN routine.

BLOCK EXTENT LIST AND NOTE LIST

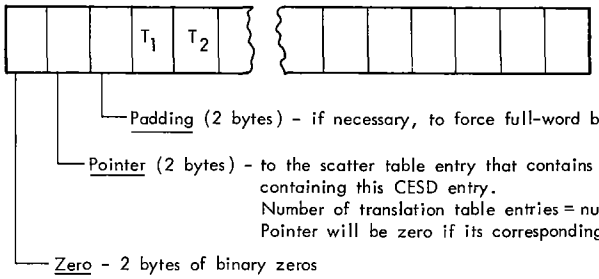


*Concatenation number is a value that specifies this data set's sequential position in a group of concatenated data sets.

SCATTER/TRANSLATION RECORD

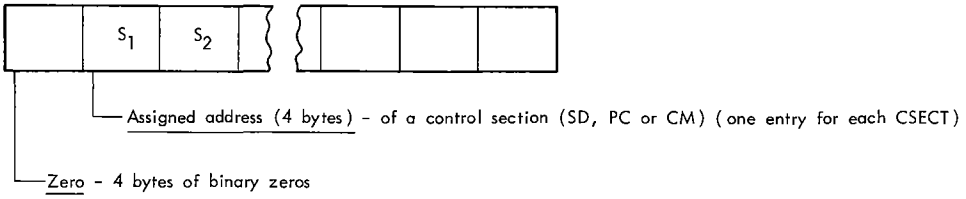


Translation Table

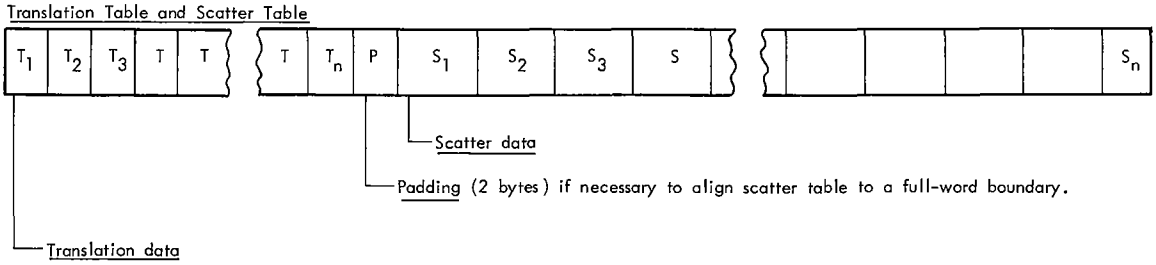


NOTE: (One 2-byte entry for each external symbol)

Scatter Table



Translation Table and Scatter Table



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

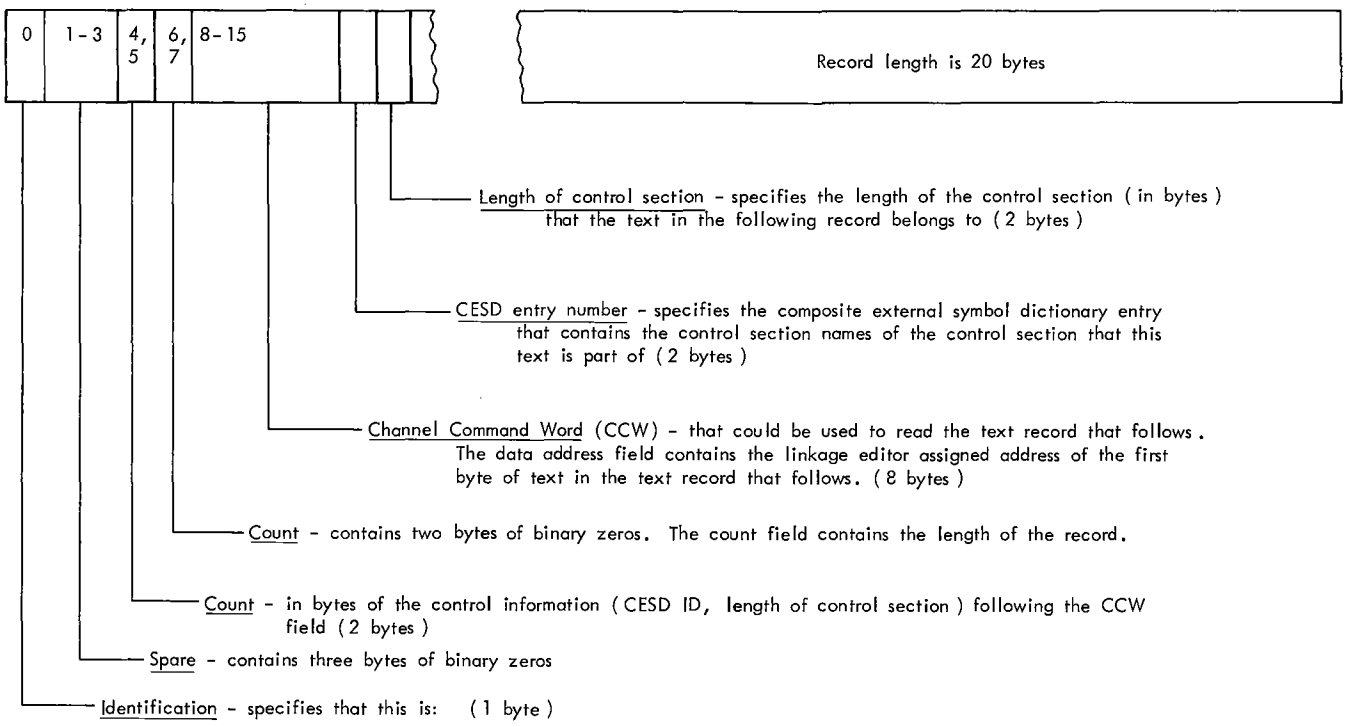
Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	32		IOB - 8 fullwords.
32	20	8		IOB seek address - 2 fullwords.
40	28	48		SEEK BUFFERS (4) - 12 FULLWORDS.
88	58	24		Search and TIC CCWs - 3 doublewords.
112	70	264		RLD buffer 1 - 33 doublewords.
376	178	40		Channel program 1 - 5 doublewords.
416	1A0	264		RLD buffer 2 - 33 doublewords.
680	2A8	40		Channel program 2 - 5 doublewords.
720	2D0	264		RLD buffer 3 - 33 doublewords.
984	3D8	40		Channel program 3 - 5 doublewords.
1024	400	4		I/O ECB - 1 fullword.
1028	404	4		ECB - 1 fullword.
1032	408	8		Buffer table pointer - 2 fullwords.
1040	410	36		Buffer table - 9 fullwords.
1076	434	64		Register save area - 16 fullwords.
1140	474	4		Address of translation table - 1 fullword.
1144	478	4		Address of scatter list - 1 fullword.
1148	47C	4		Address of R-pointer - 1 fullword.
1152	480	4		Address of P-pointer - 1 fullword.
1156	484	4		Boundary word for relocation - 1 fullword.
1160	488	8		Fetch flags - 2 fullwords.
		Byte 0		Reserved.
		Byte 1		FF = program is being scatter-loaded. 00 = program is being block-loaded.
		Byte 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.
		Byte 3		FF = end condition. Only termination processing by the Program Fetch routine is needed. 0F = end condition. Buffer processing is needed.
		Bytes 4-7		0 = a read operation was just completed. A text record, followed by an RLD or control record, was just 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 are saved to be used when channel-program restart is needed.
1168	490	8		ECB list - 2 fullwords.
1176	498	4		Last table entry - 1 fullword.

PROGRAM FETCH BUFFER TABLE

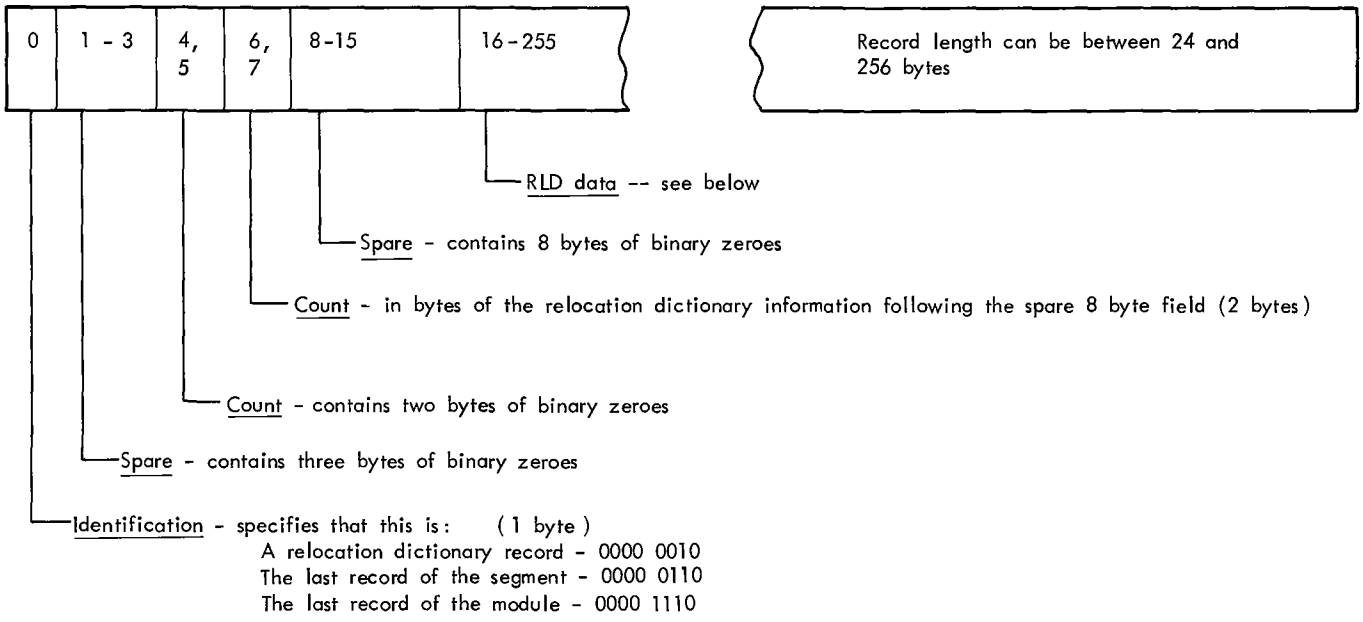
Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1		Buffer code.
		0000 0000		Buffer empty.
		1000 0000		Buffer full.
1	1	3		Pointer to next entry (byte 12).
4	4	1		TIC command.
5	5	3		Address of channel program 2.
8	8	1		Zero.
9	9	3		Address of buffer 1.
12	C	1		Buffer code (same as byte 0).
13	D	3		Pointer to next entry (byte 24).
16	10	1		TIC command.
17	11	3		Address of channel program 3.
20	14	1		Zero.
21	15	3		Address of buffer 2.
24	18	1		Buffer code (same as byte 0).
25	19	3		Pointer to first entry (byte 0).
28	1C	1		TIC command.
29	1D	3		Address of channel program 1.
32	20	1		Zero.
33	21	3		Address of buffer 3.

CONTROL RECORD

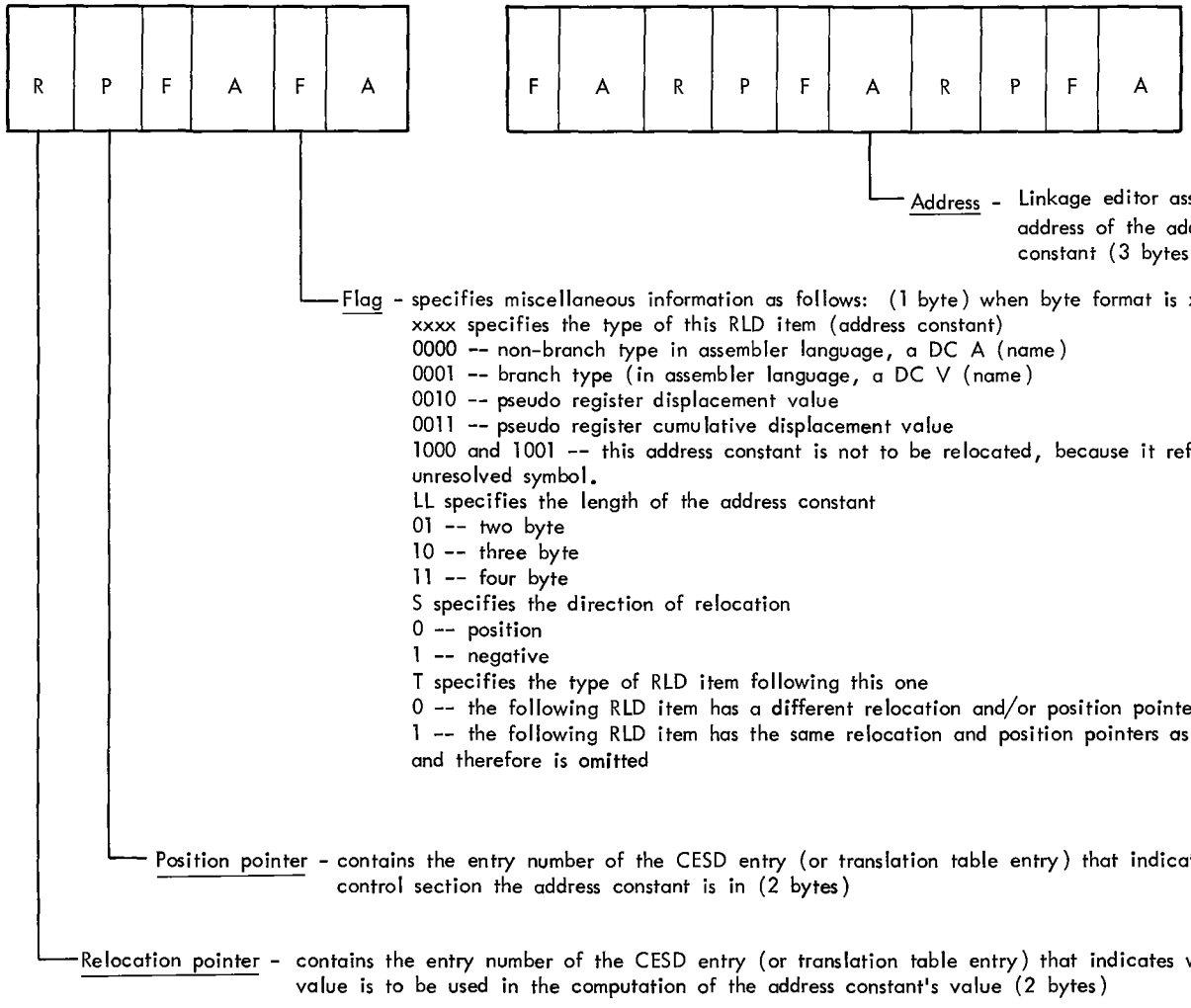


- A control record - 0000 0001
- The control record that precedes the last text record of this overlay segment - 0000 0101
- The control record that precedes the last text record of the module - 0000 1101

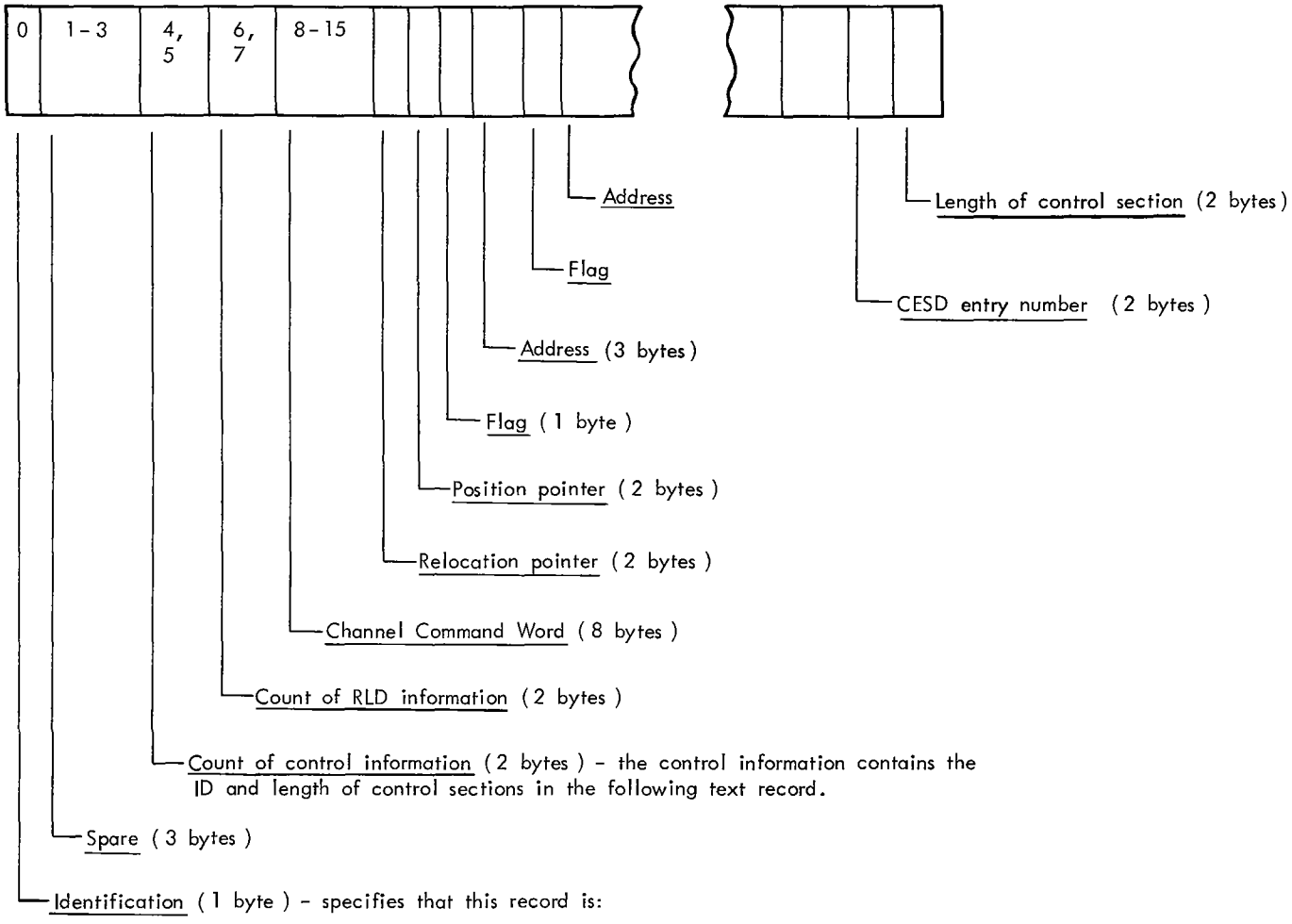
RELOCATION DICTIONARY (RLD) RECORD



RLD Data



CONTROL AND RELOCATION DICTIONARY RECORD



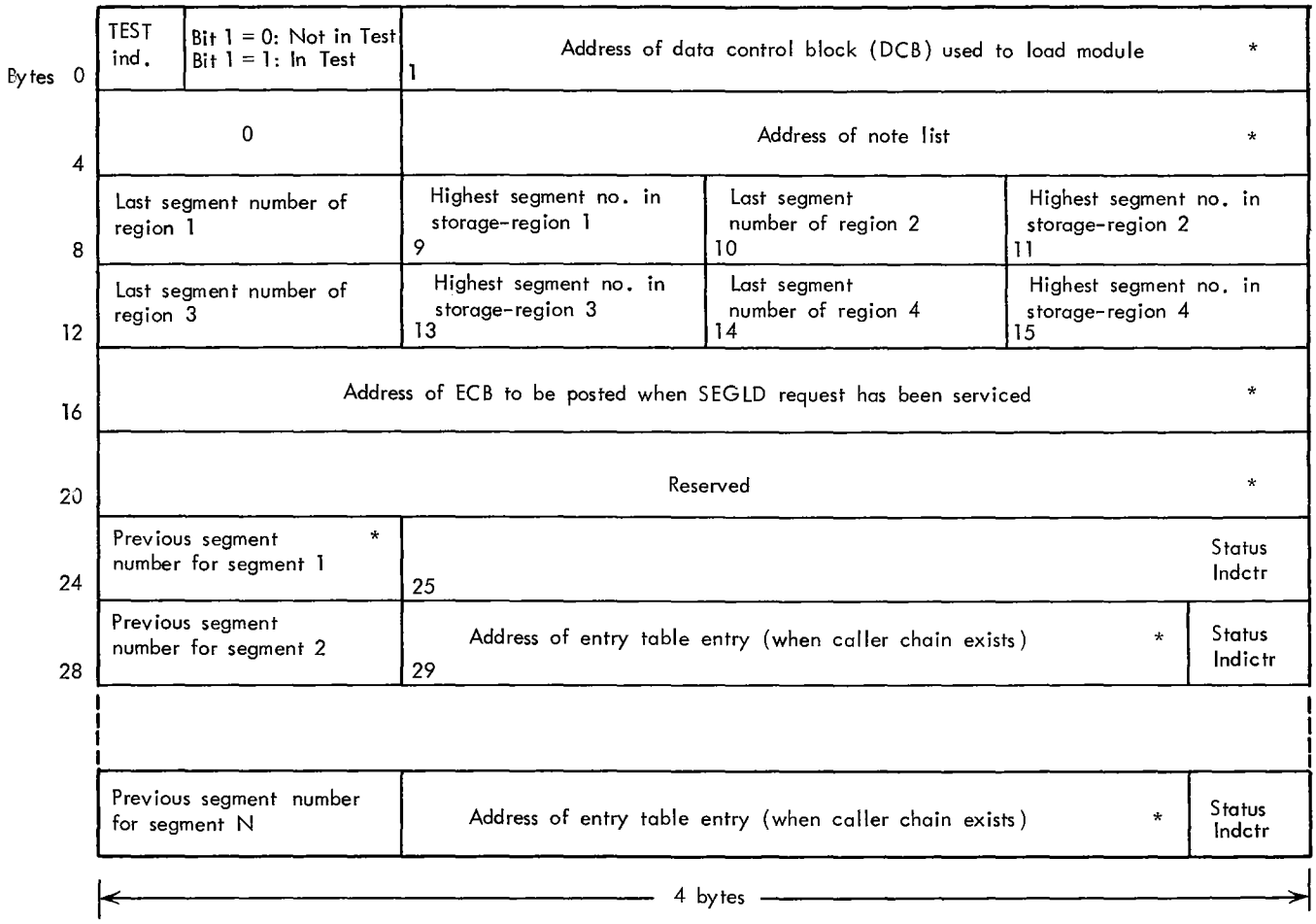
- A control and RLD record - 0000 0011
- A control and RLD record that is followed by the last text record of a segment - 0000 0111
- A control and RLD record that is followed by the last text record of a module - 0000 1111

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



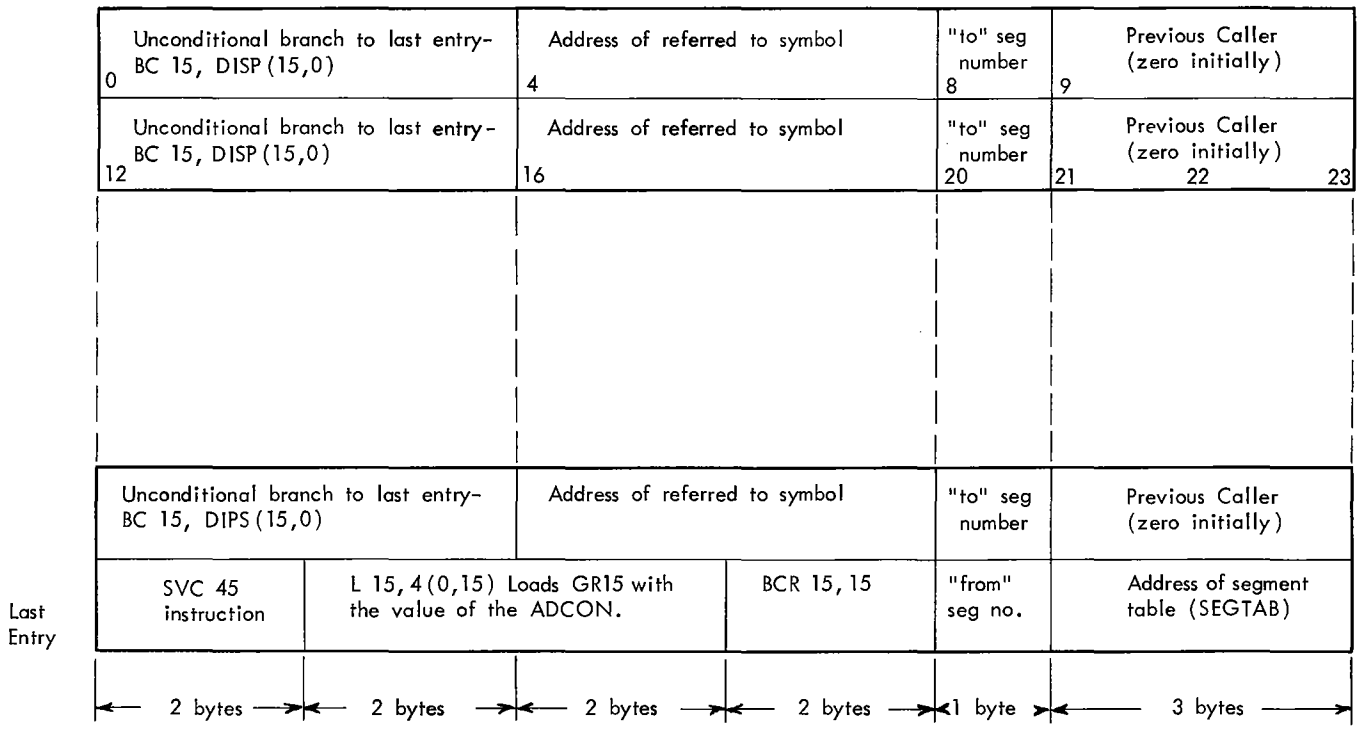
Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1		TEST indicators.
		x...		Module is "under test" using TESTRAN. Initialized by Program Fetch routine.
		.x..		0 = not in Test. 1 = in Test.
1	1	3		Address of DCB used to load module.
4	4	1		Zero.
5	5	3		Address of note list.
8	8	1		Last segment number of region 1.
9	9	1		Highest segment number in storage-region 1. (Initially set to 01 by linkage editor.)
10	A	1		Last segment number of region 2.
11	B	1		Highest segment number in storage-region 2. (Initially set to 00.)

12	C	1	Last segment number of region 3.
13	D	1	Highest setment number in storage-region 3. (Initially set to 00.)
14	E	1	Last segment number of region 4.
15	F	1	Highest segment number in storage-region 4. (Initially set to 00.)
16	10	4	Address of ECB to be posted when SEGLD request has been serviced.
20	14	4	Reserved.
24	18	1	Previous segment number for segment 1.
25	19	3	Address of entry table entry. Last two bits indicate status of segment: 00 = segment is in main storage as a result of a branch to the segment. 01 = segment is not in main storage, but is scheduled to be loaded. 10 = segment is in main storage, no caller chain exists. 11 = segment is not in main storage. (Initially set to 10.)
28	1C	1	Previous segment number for segment 2.
29	1D	3	Address of entry table entry. Status bits initialized to 11.
Variable		1	Previous segment number for segment n.
Variable		3	Address of entry table entry. Status bits initialized to 11.

*Set to zero by linkage editor.

Note: "Region" refers to the regions of a multiregion overlay structure, not to a job step's region of main storage (see Linkage Editor SRL).

ENTRY 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)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1		Flags.
		x...		0 = subpool belongs to associated task. 1 = subpool is shared.
		.1..		Element is last SPQE on queue. This bit is usually zero.
		..1.		Subpool shared with another task.
		...x xxxx		Reserved.
1	1	3	SPQEPTR	Pointer to next SPQE. When zero, this element is last SPQE on queue.
4	4	1	SPID	Identifying number of subpool.
5	5	3	DQEPTR	Pointer to first DQE for subpool. If subpool shared, field points to "owning" SPQE.

DESCRIPTOR QUEUE ELEMENT (DQE)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1		Reserved.
1	1	3	FQEPTR	Pointer to first free area (first FQE), or zero if entire block is allocated.
4	4	1		Reserved.
5	5	3	DQEPTR	Pointer to next DQE. Zero in last DQE.
8	8	1	DQEHRID	Flags.
		xxxx xxx.		Zero.
		x		0 = DQE describes storage obtained from hierarchy 0. 1 = DQE describes storage obtained from hierarchy 1.
9	9	3		Address of the first 2K block described by this DQE.
12	C	1		Reserved.
13	D	3		Length in bytes described by this DQE. (Always a multiple of 2K bytes.)

FREE QUEUE ELEMENT (FQE)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1		Reserved.
1	1	3	FQEPTR	Pointer to next lower free area.
4	4	1		Reserved.
5	5	3	LENGTH	Number of bytes in free area.

ALLOCATED QUEUE ELEMENT (AQE)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1		Reserved.
1	1	3	AQEPTR	Pointer to next allocated area.
4	4	1		Reserved.
5	5	3	LENGTH	Number of bytes in allocated areas.

GOVRFLB (ORIGIN LIST FOR MAIN STORAGE QUEUES)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1		X'80' = 2K of SQA not available to initiate a job in a region.
1	1	3	SQBOUND	Address of first byte beyond system queue area.
4	4	1		Reserved.
5	5	3	DQESQES	Address of the DQE describing the system queue area.
8	8	1		Reserved.
9	9	3	PQEPTR	Address of a dummy PQE minus 8 bytes. The dummy PQE points to the PQE describing unassigned main storage (storage not assigned to any region).
12	C	1		Reserved.
13	D	3	SZDPRS	Amount of storage available in hierarchy 0 after NIP.
16	10	1		Reserved.
17	11	3	SZDLCS	Smount of storage available in hierarchy 1 after NIP.
20	14	1	COUNT	Number of "VARY STORAGE, OFFLINE" commands in master scheduler region.
21	15	3	VQEPTR	(M65MP only) address of the first VQE describing storage areas scheduled for removal in a multi-processing system. Zero if no VQEs exist.
24	18	4	NIPSQSBD	Address of first byte beyond system queue area (used only by Rollout/Rollin).

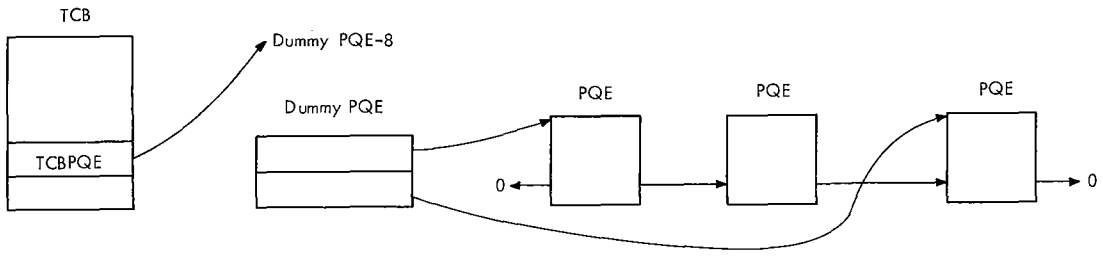
PARTITION QUEUE ELEMENT (PQE)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	4	PQEFFBQE	Address of first FBQE in region described by this PQE. If there are no FBQEs, contains the PQE address.
4	4	4	PQEBFBQE	Address of last FBQE in region described by this PQE. If there are no FBQEs, contains the PQE address.
8	8	4	PQEFPQE	Address of next PQE on queue. Contains zeros in last PQE.
12	C	4	PQEBPQE	Address of preceding PQE on queue. Contains zeros in first PQE.
16	10	4	PQETCB	Address of TCB for the job step to which the space belongs. Contains zeros if the space was obtained from unassigned free space.
20	14	4	PQESIZE	Size of region described by this PQE. (Always a multiple of 2048.)
24	18	4	PQEREGN	Address of first byte of region described by this PQE.
28	1C	1	PQERFLGS	Rollout flags.
		x...		0 = space described by this PQE is owned. 1 = space is borrowed.
		.1..		Region has been rolled out. Meaningful only if bit 0 = 0.
		..1.		Region is in use by borrower. Meaningful only if bit 0 = 0.
		...1		Region cannot be rolled in due to a machine check.
		 xxxx		Reserved.
29	1D	1	PQEHRID	Hierarchy identifier.
		xxxx xxx.		Zero.
		x		0 = PQE describes region in hierarchy 0. 1 = PQE describes region in hierarchy 1.
30	1E	2		Reserved.

DUMMY PARTITION QUEUE ELEMENT (DPQE)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	4		Address of first PQE on chain.
4	4	4		Address of last PQE on chain.

Relationship of Dummy PQE to TCB and PQE Chain



FREE BLOCK QUEUE ELEMENT (FBQE)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1		Reserved.
1	1	3	FWDPTR	Pointer to the next higher address FBQE in the region. In the highest address FBQE, contains the address of the PQE.
4	4	1		Reserved.
5	5	3	BCKPTR	Pointer to the next lower FBQE in the region. In the lowest FBQE, contains the address of the PQE.
8	8	1		Reserved.
9	9	3	SIZE	Number of bytes in the set of 2K blocks.

ROLLOUT I/O QUEUE ELEMENT (RIQE)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	4		Address of next RIQE.
4	4	4		Address of rolled-out job step's TCB.
8	8	4		Address of I/O-purged TCB.
12	C	4		Beginning of address of IOB chain.

REPLY QUEUE ELEMENT (NON-MCS)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	4	RQERQE	Address of next reply queue element.
4	4	2	RQELD	Reply identification number.
6	6	2	RQEXA	Flags.
		Byte 0		
		1...		Associated reply will be purged.
		..1.		Associated task has been rolled out.
		.x.x xxxx		Reserved.
		Byte 1		Reserved.
8	8	1	RETJID1	First half of TSO user ID for swapped-out task.
9	9	3	RQETCB	Address of TCB for task that issued message for which this RPQE represents a reply.
12	C	4	RQEXB	Address of purging message buffer, or temporary buffer if reply was deferred by rollout.
16	10	1	RQELNTH	Maximum length of reply.
17	11	3	RQERPTR	Address of user's buffer.
20	14	1	RQETJID2	Second half of TSO user ID for swapped-out task.
21	15	3	RQEECB	Address of user's ECB.

REPLY QUEUE ELEMENT (MCS)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	4	RQELKP	Address of next reply queue element.
4	4	2	RQEID	Reply identification number.
6	6	1	RQEXA	Purge flags.
		1...		Associated reply will be purged.
		..1.		Associated task has been rolled out.
		.x.x xxxx		Reserved.
7	7	1	RQEAVAIL	Buffer flags.
		1...	RQEBUFA	Buffer is free.
		.1..	RQEBUFB	Buffer is in use.
		...1	RQEBUFD	Buffer has been obtained dynamically.
		.... 1...	RQEBUFE	Buffer has been serviced.
		..x. .xxx		Reserved.
8	8	1	RETJID1	First half of TSO user ID for swapped-out task.
9	9	3	RQETCB	Address of TCB for task that issued message for which this RPQE represents a reply.
12	C	4	RQEXB	Address of communications task emergency message to cancel replies.

16	10	1	RQELNGTH	Maximum length of reply.
17	11	3	RQEPTR	Address of user's buffer.
20	14	1	RETJID2	Second half of TSO user ID for swapped-out task.
21	15	3	RQEECB	Address of user's ECB.

SVC PURGE PARAMETER LIST

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1	PURGOPT	Purge options (always X'02' for rollout, requesting 'TCB' and 'quiesce' options).
1	1	3	PURGDEB	Address of DEB (not used for rollout).
4	4	1		Completion code to be placed in ECB.
5	5	3	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	8	1		Count field for quiesce option. Number of request elements whose I/O operations have not yet completed.
9	9	3	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.
12	C	1	PRGFGL	Purge flags.
		x...		0 = purge entry. 1 = wait entry.
		.x..		0 = return before wait. 1 = purge and wait.
		..x.		0 = DEQ before wait. 1 = no DEQ before wait (set before system DEB purged).
		...x xxxx		Reserved.
13	D	3	PRGQPL	Address of quiesce I/O parameter list.

TIMER QUEUE ELEMENT (TQE)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1	TQEFLGS	Flags.
		1...		Timer element is not on timer queue.
		.1..		Local TOD option used.
		..xx		00 = TUINTVL requested. 01 = BINTVL requested. 10 = reserved. 11 = DECINTVL requested.
		.... 1...		Interval is complete.
		.... .1..		Exit specified.*
		xx		00 = task request. 01 = wait request. 10 = supervisory request.* 11 = real request.
			*5-7	110 = midnight supervisory timer element.
1	1	3	TQETCB	Address of the TCB for the task for which this timer element is being used.
4	4	1		Zero.
5	5	3	TQEFLNK	Forward link field. Contains the address of the first byte of the next TQE on the timer queue.
8	8	1		Zero.
9	9	3	TQEBLNK	Backward link field. Contains address of first byte of previous TQE on timer queue.
12	C	4	TQEVAL	Interval value. If the element is on the timer queue, this field represents the time of expiration (TOX) of the interval relative to the 6-hour interval. If the element is off the timer queue, this field represents 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.
16	10	4	TQEPSW	First word of current PSW - used when TQE serves as IRB.
20	14	4	TQESAV	Used to save contents of TQEVAL when TQE is converted from TASK to REAL.
24	18	1	TQETJID1	First half of TSO user ID (when user not in main storage).
25	19	3	TQESAADR	Address of the problem program register save area.
28	1C	1	TQETJID2	Second half of TSO user ID (when user not in main storage).
29	1D	3	TQEEXIT	Exit routine address. 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, the field is zero.
32	20	64	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.

96	60	16	TQEECB	Event control block to be posted for a "wait" type interval. Used for ECB when WAIT parameter given in STIMER macro instruction.
			TQEIQE	Interruption queue element passed to Stage 2 Exit Effector to schedule a timer exit routine. Used for interruption queue element when TQE serves as IRB.

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 (SCVT)

This table appears in module IEAQET00, beginning at symbolic location IEABEND. It consists of a list of address constants that point to routine entry points or system control blocks.

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	4	SCVTPGTM	Address of EOT Purge Timer routine (IEAQPGTM).
4	4	4	SCVTPGWR	Address of WTOR Purge routine (IEECVPRG).
8	8	4	SCVTSPET	Address of Release Main Storage routine (IEAQSPET).
12	C	4	SCVTACT	Address of TACT (IEAQTAQ).
16	10	4	SCVTERAS	Address of EOT Erase Phase routine (IEAQERA).
20	14	4	SCVTQCBO	Address of QCB origin (IEAQQCBO).
24	18	4	SCVTPGEQ	Address of ENQ/DEQ Purge routine (IEA0EQ01).
28	1C	4	SCVTRMBR	Address of REGMAIN branch entry (RMBRANCH).
32	20	4	SCVTPGIO	Address of SVC Purge routine (IGC016).
36	24	4	SCVTRACE	Address of Trace routine switch (IECXTRA).
40	28	4	SCVTTASW	Address of Task Switching routine (IEA0DS02).
44	2C	4	SCVTCDCL	Address of CDCONTRL in common subroutines of Contents Supervision (IEAQCS02).
48	30	4	SCVTLFRM	Branch entry-point to FREEMAIN routine (FMBRANCH).
52	34	4	SCVTPABL	Address of Release Loaded Programs routine (IEAQABL) in EOT.
56	38	4	SCVTDQTC	Address of Dequeue TCB routine (IEADQTCB) in EOT.
60	3C	4	SCVTHSKP	Address of CDHKEEP (CDHKEEP) in CDEXIT routine.
64	40	4	SCVTRPTR	Address of trace table pointers (TRPTR).
68	44	4	SCVTGMBR	List format GETMAIN branch entry-point (GMBRANCH).
72	48	4	SCVTAUCT	Transient area user count (TAUSERCT).
76	4C	4	SCVTROCT	Address of rollout counters (IEARCTRS).
80	50	4	SCVTROQ	Address of rollout queue (IEAROQUE).
84	54	4	SCVTRIRB	Address of rollout IRB (IEAROIRB).
88	58	4	SCVTRTCB	Address of rollout TCB (IEAROTCB).
92	5C	4	SCVTCOMM	Address of Communications Task routine (IEECVCTW) for DAR.
96	60	4	SCVTABLK	Entry to ABTERM routine (SCEDWAIT) for DAR.
100	64	4	SCVTNFND	Entry to Transient Area Handler routine (TBNOTFND) for DAR.

104	68	4	SCVTRMTC	Address of RMS TCB (IGFRMTCB).
108	6C	4	SCVTMSSQ	Address of GOVRFLB.
112	70	4	SCVTCTCB	Address of Communications Task TCB (IEECVTCB).
116	74	4	SCVTETCB	Address of System Error TCB (IEARTCB).
120	78	4	SCVTRXLQ	Address of recovery extent list.
124	7C	4	SCVTRQND	Address of end of I/O RQE table (IECITSAR).
128	80	4	SCVTTAR	Address of Transient Area Refresh routine.
132	84	4	SCVTSVCT	Address of SVC table (IBMORG).
136	88	4	SCVTSTXP	Address of STAX Purge routine (IEAKJXP).
140	8C	4	SCVTTQE	Address of TSO subsystem TQE (IEATSELM).
144	90	4	SCVTRMSV	Address of SVC 85 instruction (IORMSSVC).
148	94	4		Reserved.
152	98	4	SCVTFMSA	
		Byte 0		
		1...	SCVTSW1	Conditional branch entry to FREEMAIN.
		.1..	SCVTSW2	Branch entry to IGC003 from ABEND requires use of conditional FREEMAIN for unprotected storage.
		..1.	SCVTSW3	Branch entry to SVC 5 type FREEMAIN (internal to IGC003).
		...x xxxx		Reserved.
		Bytes 1-3		Address of FREEMAIN register (2-14) save area (IEA10FS).

ABDUMP PARAMETER LIST

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1		ID.
1	1	1		Zero.
2	2	2		Option flags.
		Byte 0		
		x...	PFABEND	0 = ABEND request 1 = SNAP request.
		.1..	PFTCB	TCB address is given.
		..1.	PFSUPDAT	Display all supervisor data.
		...1	PFTRACE	Display trace table (if possible).
		.... 1...	PFNUC	Display the nucleus.
		.... .1..	PFSNAP	Snapshot list is given.
		.... ..1.	PFID	ID given.
		1	PFQCB	Display the QCBs.
		Byte 1		
		1...	PFSAVE	Save area (see next flag).
		.x..	PFSAVE2	0 = display entire save area. 1 = display headings only.
		..1.	PFREGS	Display registers on entry to ABEND or SNAP.
		...1	PFLPA	Display link pack area.
		.... 1...	PFJPA	Display job pack area.
		.... .1..	PFPSW	Display PSW on entry to ABEND or SNAP.
		.... ..1.	PFSPALL	Display all subpools less than subpool 128.
		x		Reserved.
4	4	1		Zero.
5	5	3		Pointer to DCB.
8	8	1		
		xxxx xxx.		Zero.
		1		SDUMP request.
9	9	3		Pointer to TCB.
12	C	1		Zero.
13	D	3		Pointer to storage list.

TIME-SLICE CONTROL ELEMENT (TSCE)

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.

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1		Dispatching priority of time-slice group.
1	1	3		Address of first TCB on TCB queue that is a member of the time-slice group.
4	4	1		Zero.
5	5	3		Address of last TCB on TCB queue that is a member of the time-slice group.
8	8	1		Zero.
9	9	3		Address of next TCB to be dispatched when the priority obtains control.
12	C	1		TSCE flags.
		1...		Last TSCE.
		.xxx xxxx		Reserved.
13	D	3		Length of time-slice. In milliseconds before NIP; in timer units after (26.04166 micro-seconds per timer unit).

DISPLAY CONTROL MODULE (DCM)

The DCM is composed of a resident portion (RDCM) and a transient portion (DDCM) which may be resident or transient at the user's option.

RESIDENT DISPLAY CONTROL MODULE (RDCM)

The RDCM contains screen control information and the address of the main storage area assigned to the TDCM. If the TDCM is transient, this main storage area may be used for the TDCMs of several consoles.

The RDCM may also contain one or more screen area control blocks. These control blocks contain information about each status display area defined for the screen.

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	4	DCMADTRN	Address of the main storage area assigned to the TDCM.
4	4	1	DCMINDEX	Display console identification number assigned by the Graphic Console Initialization routine.
5	5	1	DCMRFLGS	Display console status flags.
		1...	DCMREAD	DCM read in process.
		.1..	DCMTYPE	Console uses transient DCMs.
		..1.	DCMWRITE	DCM write in process.
		...1	DCMDOM	DOM must be tried.
		.... 1...	DCMNIPP	DCM was used by NIP.
		.... ..1.	DCMFRSRD	First read of DCM.
		1	DCMSRCH	Processor 0 routing flag.
6	6	2	DCMLEN	Length of the transient portion of the DCM.
8	8	4	DCMADKP	Address of routed K command parameter list.
12	C	1	DCMTOPAR	Address of the top-most display area defined for the screen.
13	D	1	DCMTOPDS	Address of the top-most display area that is in use (contains a display).
14	E	1	DCMRESTA	Flags for attention and open requests.
		1...	DCMATTN	Attention status has been saved.
		.1..	DCMOPNSV	Open status has been saved.
		..xx xxxx		Reserved.
15	F	1	DCMDEVTY	Device type flags.
		1...	DCMTY60	Device is a 2260.
		.1..	DCMTY50	Device is a 2250.
		..xx xxxx		Reserved.
16	10	4	DCMADSDS	Pointer to the first screen area control block. Zero if no display areas are defined for the screen.
20	14	1	DCMRMS	Recovery management support (RMS) information.

21	15	3	DCMADRMS	Address of RMS channel programs.
24	18	4	DCMWLAST	Address of last console queue entry to be displayed.
28	1C	2	DCMRMSAL	Number of lines in message area.
30	1E	2		Reserved.
32	20	4	DCMMSGSV	Address of a list of saved NIP messages.
36	24	4	DCMADPFK	Address of the resident PFK area.
40	28	2	DCMINTVL	Time interval for the DCM.
42	2A	2	DCMTMCTR	Time counter for the DCM.
44	2C	1	DCMR2FLG	Timer flags.
		1...	DCMRXSFL	Full screen.
		.1..	DCMRXUNV	Unviewable message displayed.
		..1.	DCMRXTMR	Timer flag.
		...1	DCMRXRLL	Ready to roll.
		.... 1...	DCMRXDEL	Pending delete request.
		.... .1..		Entry was from options.
		.... ..1.	DCMRXTIM	Timer elapsed for this display.
		1	DCMRXDCM	TDCM is in storage.
45	2D	1	DCMR3FLG	Flags.
		1...	DCMSTSWT	Changing status of output-only console.
		.1..	DCMKVIP	Entry for K VARY command.
		..1.	DCMCLPR	Close in process.
		...1		Asynchronous error message on screen.
		 xxxx		Reserved.
46	2E	2		Reserved.

Screen Area Control Block (SACB) - one complete SACB is created for each display area defined for the screen; if no display areas are defined, no SACBs are created. SACBs may be contiguous with the RDCM, or may be chained to the RDCM by pointers.

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
48	30	2	DCMPLN	Length of the display area in bytes.
50	32	2	DCMPLNPR	Length of the SACB in bytes.
52	34	4	DCMACBNX	Address of the next SACB.
56	38	1	DCMAID	Alphabetic identifier assigned to the display area.
57	39	1	DCMASACB	SACB flags.
		1...	DCMAUSE	SACB in use.
		.1..	DCMAGM	SACB space obtained by GETMAIN.
		..xx xxxx		Reserved.
58	3A	2	DCMALN	Length of the display area in bytes.
60	3C	1	DCMATOP	Line number of the top line of the display area.
61	3D	1	DCMAROW	Line number of the line to be written next.
62	3E	2	DCMAFR	Frame number of the frame being displayed.

64	40	4	DCMAMJWQ	Address of the console queue entry for the major WQE.
68	44	4	DCMAMIN	Address of the console queue entry for the minor WQE being written.
72	48	4	DCMATIME	Time indicator written in the control line of the display being written.
74	4C	2	DCMAMT	Message type flag.
		..1.	DCMAMTC	Monitor active.
		xx.x xxxx		Reserved.
76	4E	1	DCMAFLG1	Display area flags.
		1...		Display coming to area.
		.1..	DCMADISP	Display in area.
		..1.	DCMADEND	End of display on screen.
		...1	DCMAFRPR	Framing in process.
		.... 1...	DCMAFULL	Frame full.
		.... .1..	DCMABL	Blanking to be done.
		xx		Reserved.
77	4F	1	DCMAFLG2	Display area status flags.
		1...	DCMALMIN	Saved pointer to last minor output.
		.1..	DCMAWCON	Write control line.
		..1.	DCMARCON	Rewrite control line.
		...1	DCMAMJFR	Major WQE has been found.
		 xxxx		Reserved.
78	50	1	DCMADFLG	Display flags.
		1...	DCMADD	Dynamic display.
		.1..	DCMAHOLD	Dynamic display in hold mode.
		..1.	DCMACSIB	Dynamic display with control line in SIB.
		...x xxxx		Reserved.
79	51	3		Reserved.

TRANSIENT DISPLAY CONTROL MODULE (TDCM)

The TDCM contains two sections: the "device independent" section and the "device dependent" section. The device independent section is the same size and form for all display devices; the device dependent section varies according to the device it is supporting.

Device Independent Section: These areas comprise save areas, work areas and communications task information.

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	2		Length of TDCM.
2	2	2		Reserved.
4	4	1	DCMFLG1	TDCM flags.
		...1	DCMQUE	Queue this DCM for WRITE.
		.... ..1.	DCMOUTPT	DCM updated for output only.
		xxx. xx.x		Reserved.
5	5	3		Reserved.
8	8	4	DCMWTINT	Initial value for DCMWTBUF.
12	C	4		Reserved.
16	10	4	DCMPACK	Area used for packing decimal digits.
20	14	4	DCMCVBIN	Area used for conversion to binary.
24	18	1	DCMTIMES	Timer routine flags.
		1...	DCMTIMER	Time elapsed for this display.
		.1..	DCMOPTTI	Options to Timer routine.
		..1.	DCMGROLL	Display ready to roll.
		...1	DCMOTTMM	Option to Timer routine to message module.
		.... 1...	DCMSTTMR	STIMER should be issued.
		.... .1..	DCMASYN	Timer set for asynchronous error message.
		.... ..1.	DCMOCTTI	OPEN-CLOSE to Timer routine.
		1	DCMRMTTI	Roll mode to Timer routine.
25	19	1		Reserved.
26	1A	2	DCMELGN	Pointer to the last character in the entry area.
28	1C	4	DCMBUFAD	Address of buffer address table.
32	20	4	DCMDOMPK	Address of first DOM number.
36	24	4	DCMAMTAB	Address of first screen control table entry.
40	28	4	DCMADSEC	Address of first secondary screen control table entry.
44	2C	4	DCMADDRL	Address of last screen control table entry.
48	30	4	DCMASCRN	Address of screen image buffer.
52	34	4	DCMLSCRN	Address of last screen image buffer line.
56	38	4	DCMWTBUF	Address of the PFK number line (or the blank line between the message area and the instruction line on display consoles that do not support light pen command entry) in the screen image buffer.

60	3C	4	DCMAINS	Address of the instruction line in the screen image buffer.
64	40	4	DCMAENTR	Address of the entry area in the screen image buffer.
68	44	4	DCMAWARN	Address of the warning line in the screen image buffer.
72	48	4	DCMADCHP	Address of the channel program area.
76	4C	4	DCMPFKLN	Reserved.
80	50	4	DCMCXSVE	CXSA save area.
84	54	4	DCMADOPN	Address of the command operand.
88	58	20	DCMDSAV	Command save area and work area.
108	6C	2	DCMINLGN	Input length.
110	6E	2	DCMMCSFL	Space for MCS flags.
112	70	128	DCMINPUT	Space for input message text.
240	F0	8	DCMRQDEL	Buffer for pending delete requests.
248	F8	2	DCMLGNTH	Length of a line.
250	FA	2	DCMBAINC	Position of cursor in screen image buffer.
252	FC	2	DCMIRCTR	Intervention requested message counter.
254	FE	2	DCMBADLN	Location in buffer to begin writing message.
256	100	2	DCMBYTCT	Number of bytes to write.
258	102	2	DCMADNUM	The line number to be assigned to the next line.
260	104	2	DCMAXLGN	Maximum line length.
262	106	2	DCMMSGAL	Number of lines in the message area.
264	108	2	DCMRMINC	RMI information.
266	10A	2	DCMSCTCN	Length of one SCT entry.
268	10C	2	DCMCORLN	Length of DCM line in main storage.
270	10E	2		Reserved.
272	110	1	DCMPFKNM	Number of the PFK key being processed.
273	111	1	DCMPFKKN	Number of the PFK key, in a list of PFK keys, that is being processed.
274	112	2	DCMDEL	Contains the current value of the DEL parameter of the CONTROL S command (Y or N).
276	114	1	DCMCON	Contains the current value of the CON parameter of the CONTROL S command (Y or N).
277	115	1	DCMSEG	Contains the current value of the SEG parameter of the CONTROL S command (numerical value).
278	116	1	DCMDL	Contains the current value of the DL parameter of the CONTROL S command (numerical value).

279	117	1	DCMRNUM	Contains the current value of the RNUM parameter of the CONTROL S command (numerical value).
280	118	2	DCMRTME	Contains the current value of the RTME parameter of the CONTROL S command (numerical value).
282	11A	1	DCMSEGDF	The default value for SEG.
283	11B	1	DCMRNUMD	The default value for RNUM.
284	11C	2	DCMRTMED	The default value for RTME.
286	11E	1	DCMASKEN	The ENTER mask.
287	11F	1	DCMASKCN	The CANCEL mask.
288	120	1	DCMASKCR	The cursor mask.
289	121	1	DCMASKLP	The light pen mask.
290	122	1	DCMASKPF	The PFK mask.
291	123	1	DCMOPTST	Screen control option flags.
		1...	DCMOPTVR	Delete verification (Y=1; N=0).
		.1..	DCMOPTAD	Automatic deletion (Y=1; N=0).
		..1.	DCMOPTSG	Default SEG specified.
		...1	DCMOPRLL	Roll mode specified.
		 xxxx		Reserved.
292	124	1	DCMCS	Open/Close request.
		1...	DCMCSC	Close requested.
		.1..	DCMCSO	Open requested.
		..xx xxxx		Reserved.
293	125	1	DCMUTILT	Reserved.
294	126	1	DCMDSTAT	Current display status flags.
		1...	DCMDSTFL	Full screen.
		.1..	DCMUNVW	UNVIEWABLE MESSAGE written to screen.
		..1.	DCMDSTNM	Messages are numbered.
		...1	DCMDSTNH	Messages numbered -- HOLD option.
		.... 1...	DCMDSINR	Intervention requested deletion tried.
		.... .1..	DCMDSAUT	Automatic deletion tried.
		xx		Reserved.
295	127	1	DCMMCSST	MCS Interface flags.
		1...	DCMDUSE	Status Display Support.
		.1..	DCMNOHRD	No hardcopy message written.
		.... ..1.	DCMOOMSS	Message stream entry.
		1	DCMOOSDS	Status display entry.
		..xx xx..		Reserved.
296	128	1	DCMIOUNQ	Device I/O information flags.
		1...	DCMIO226	(2260) RMI performed.
		.1..	DCMRPCUR	(2260) Advance cursor to blanks.
		..1.	DCMFRSCN	(2260) Put output in hold mode.
		...1	DCMRDARM	(2250) Perform read after RMI.
		.... 1...	DCMW2250	(2250) Device is a 2250.
		.... .1..	DCMINNOR	(2250) Normal instruction line.
		.... ..1.	DCMINERR	(2250) Error instruction line.
		x		Reserved.

297	129	1	DCMIOCM1	I/O Communications byte 1.
		1...	DCMDORMI	Issue RMI.
		.1..	DCMSOUND	Sound alarm.
		..1.	DCMWRWRN	Write warning line.
		...1	DCMWRMSG	Write full message area.
		.... 1...	DCMWRPAR	Write partial message area.
		.... .1..	DCMWRINS	Write instruction line.
		.... ..1.	DCMWRENT	Write entry area.
		1	DCMINSC	Insert cursor.
298	12A	1	DCMIOCM2	I/O Communications byte 2.
		1...	DCMBLENT	Blank entry area.
		.1..	DCMBLWRL	Blank left half of warning line.
		..1.	DCMBLWRR	Blank right half of warning line.
		...1	DCMINSSH	Initialize and shift instruction line.
		.... 1...	DCMWINFD	Write informational display.
		.... .1..	DCMERASE	Perform erase.
		.... ..1.	DCMIOCRD	Perform read.
		1	DCMWRASY	Write asynchronous error message to mid-screen.
299	12B	1	DCMIOCM3	I/O Communications byte 3.
		1...	DCMOPRMI	RMI after OPEN to unlock keyboard.
		.1..	DCMSSRG	Suppress start regeneration.
		..x.		Reserved.
		...1	DCMWRPFK	Write PFK area.
		.... 1...	DCMPFKAT	PFK attention.
		.... .1..	DCMRDPFK	PFK area read.
		.... ..1.	DCMACPFK	Turn active PFK lights on.
		1	DCMLTPFK	Turn all PFK lights on.
300	12C	1	DCMLINEN	Line numbers to begin write.
301	12D	1	DCMCULNO	Line in entry area to insert cursor.
302	12E	1	DCMPOSCU	Position to insert cursor.
303	12F	1	DCMASYNC	Asynchronous error retry flags.
		1...	DCMASCAN	Asynchronous error message on screen.
		.1..	DCMASDA	Retry bit.
		..1.	DCMASIN	Retry bit.
		...1	DCMASBA	Retry bit.
		.... 1...	DCMASLOG	Log asynchronous error.
		xxx		Reserved.
304	130	1	DCMCOM1	Communications byte 1.
		1...	DCMLPENT	Enter by light pen or cursor.
		.1..	DCMIOPRD	Read performed.
		..1.	DCMCOMRM	RMI performed.
		...1	DCMCONAU	Perform automatic deletion.
		.... 1...	DCMCOMRD	Perform regular deleteion.
		.... .1..	DCMCOMNM	Number messages.
		.... ..x.		Reserved.
		1	DCMCANCL	Indicate CANCEL to Command routine.
305	131	1	DCMCOM2	Communications byte 2.
		1...	DCMCM2I	Input to be processed.
		.1..	DCMSPLIT	Message to be split.
		..1.	DCMCOMAR	Accepted reply.
		...1	DCMCLOSE	Cleanup for close.
		.... 1...	DCMERPF	Erase performed; close device.
		.... .1..	DCMCMIN5	Return to Interface 5 for blanking.

		.... ..1.	DCMBLNK	Blanking required.
		1	DCMAE	Cleanup for asynchronous error.
306	132	1	DCMCOM3	Communications byte 3.
		1...	DCMCDSP3	Display 3 work complete.
		.1..	DCMRTPFK	Return to PFK routine.
		..1.	DCMVLPFK	Verifying last command.
		...1	DCMXINT1	Entry for Interface 1 routine.
		.... 1...	DCMOLUNV	Out-of-line message caused UNVIEWABLE MESSAGE.
		.... .1..	DCMPFKWR	Write PFK updates to library.
		.... ..x.		Reserved.
		1	DCMCMIN7	Return to Interface 7 for blanking.
307	133	1	DCMCMSG1	Message Module Communications byte 1.
		1...	DCMMSGWT	Move in MESSAGE WAITING.
		.1..	DCMUNMSG	Move in UNVIEWABLE MESSAGE.
		...1	DCMCHOPT	Move in CHANGE OPTIONS.
		.... 1...	DCMELONG	Move in ENTRY TOO LONG.
		.... .1..	DCMWRCDL	Move in CON=N,DEL=Y.
		.... ..1.	DCMDELNT	Move in DEL UNCHANGED,NO TIMER.
		1	DCMMNHRD	Move in NO HARDCOPY.
308	134	1	DCMCMSG2	Message Module Communications byte 2.
		1...	DCMDLREQ	Move in DELETION REQUESTED.
		.1...	DCMRQINC	Move in REQUEST INCONSISTENT.
		..1.	DCMMSGCR	Move in INVALID CURSOR OPERATION.
		...1	DCMINVOP	Move in INVALID OPERAND.
		.... 1...	DCMCILLP	Move in ILLEGAL LIGHT PEN OPERATION.
		.... .1..	DCMDELRI	Move in DELETE REQUEST INCONSISTENT.
		.... ..1.	DCMASYRT	Move in ASYNCHRONOUS ERROR RETRYABLE.
		1	DCMASYCD	Move in ASYNCHRONOUS ERROR MAY BE RETRYABLE.
309	135	1	DCMCMSG3	Message Module Communications byte 3.
		1...	DCMCNRLL	Move in ROLL MODE.
		.1..	DCMCDLR1	Move in NO DELETABLE MESSAGES.
		..1.	DCMCDLR2	Move in INVALID RANGE.
		...1	DCMCDLR3	SEG equal to zero.
		.... 1...	DCMCDLR4	Display not on screen.
		.... .1..	DCMCDLR5	Invalid operand.
		.... ..1.	DCMNHCIN	Move NO HARDCOPY message to instruction line.
		1	DCMDTBSY	COMMAND REJECTED -- TASK BUSY.
310	136	1	DCMCMSG4	Message Module Communications byte 4.
		1...	DCMPFKNA	Move in PFK NOT ALLOCATED.
		.1..	DCMPFKND	Move in PFK NOT DEFINED.
		..1.	DCMPFKNO	Move in NO PFK ALLOCATION.
		...1	DCMPFKIP	Move in PFK IN PROCESS.
		 xxxx		Reserved.
311	137	1	DCMSVC34	SVC 34 Communications byte.
		1...	DCMMYCMD	Command to be handled by this console.
		.1..	DCMINVLD	Invalid K command.
		..1.	DCMTYPE1	K command is not routable.
		...x xxxx		Reserved.
312	138	1		Reserved.
313	139	1	DCMIORTN	Indicates which I/O routine handles I/O for the console.
314	13A	1	DCMTEST	Reserved for testing.

316	13C	2	DCMBASRT	Location in buffer for start of the channel program regeneration orders.
318	13E	2	DCMBAMI	Location of first message line for use by the channel program.
320	140	2	DCMBAPFK	Location of the PFK display line in the buffer for use by the channel program.
322	142	2	DCMBAINS	Location of the instruction line in the buffer for use by the channel program.
324	144	2	DCMBAENT	Location of the entry area in the buffer for use by the channel program.
326	146	2	DCMBAWRN	Location of the warning line in the buffer for use by the channel program.

Device Dependent Section: The following areas vary in size according to the type of display device that the DCM is supporting.

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
328	148	Variable		Buffer Address Table: contains addresses of the buffers for the various display screen fields.
Variable		Variable		Channel Program Area - work area for channel programs.
Variable		Variable		Screen Immage Buffer - contains a copy of the display screen. It is from this area that the screen is written and information of the screen is manipulated.
Variable		Variable		DOM Identification Numbers Area - contains information pertaining to delete-operator-message messages.
Variable		Variable		Screen Control Table - flags for each line in the screen image buffer.
		Byte 0		
		1...	DCMMSGWR	WTOR message display in-line.
		.1..	DCMMSGIN	Message Display in-line.
		..1.	DCMMSGCN	Message continued on next line.
		...1	DCMMSGJK	To write out-of-line display.
		.... 1...	DCMMSGAD	Message can be deleted automatically.
		.... .1..	DCMMSGRD	Request has specified this message be removed.
		.... ..1.	DCMMSGIF	Informational message displayed in-line.
		1	DCMMSGST	End of table indicator.
		Byte 1		
		1...	DCMMSGAC	Action message.
		.1..	DCMMSGC7	Descriptor code 7 message.
		..1.	DCMMSGDM	DOM issued for this message.
		...1	DCMMSGAR	Message is an accepted reply.
		.... 1...	DCMMSGIR	Intervention required message.
		.... .1..	DCMMSGCT	Continuation line.
		.... ..1.	DCMMSGPP	Message issued by problem program.
		1	DCMMSGCL	Control line of MLWTO.
Variable		Variable		Secondary Screen Control Table - Flags for out-of-line display area messages.
		1...	DCMSECCL	Control line of out-of-line display.
		.1..	DCMSECLL	Label line of out-of-line display.
		..1.	DCMSECDL	Data line of out-of-line display.

...1	DCMSECBL	This line is blanked.
.... xx..		Reserved.
.... ..1.	DCMSECDD	Line reserved for dynamic display.
....1	DCMSECST	End of table indicator.

MULTIPROCESSING COMMUNICATIONS VECTOR TABLE (MPCVT)

The MPCVT 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.

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1	CVTAFFLK	CPU identity.
		1100 0001		CPU A.
		1100 0010		CPU B.
		0000 0000		Neither.
1	1	1		Supervisor lock.
		1111 1111		Set.
		0000 0000		Not set.
2	2	2		Reserved.
4	4	4	CVTSTPTR	Address of SHOLDTAP routine.
8	8	4	CVTWTTCB	Address of Dispatcher Wait Task.
12	C	4	CVTTKRM	Address of Task Removal (TESTDSP) routine.
16	10	4	CVTGOV	Address of GOVRFLB table.
20	14	4	CVTIOTIO	Address of Multiprocessing Unit TIO routine in IOS.
24	18	4	CVTIOTCH	Address of Multiprocessing Channel TCH routine in IOS.
28	1C	4	CVTSTOR	Address of Notify Storage Inline routine (IEAMPSTR).
32	20	4	CVTVRYOF	Address of Deferred Vary Storage routine (IFSVRYOF).

VARY QUEUE ELEMENT (VQE)

The VQE describes the main storage area to be logically removed from a Model 65 Multiprocessing system due to a VARY STORAGE offline command. The address of the Vary Queue Element is located in the GOVRFLB table.

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1		Zero.
1	1	3		Address of next VQE on vary queue.
4	4	1		Zero.
5	5	3		Lower address of area specified in VARY command.
8	8	1		Zero.
9	9	3		Length of area specified in VARY command.
12	C	1		Zero.
13	D	3		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
01	Reserved
10	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:

$$b = \frac{X}{2048}$$ (disregard remainder).

The number (n) of the first of the two bits which describes the 2K block is: $n = 2*b$.

UNIT CONTROL MODULE (UCM) BASE

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
-8	-8	4		Address of UCM extension prefix.
-4	-4	4		Address of MCS prefix.
0	0	4	UCMXECB	External interruption ECB.
4	4	4	UCMAECB	Attention interruption ECB.
8	8	4	UCMOECB	WTO/R request ECB.
12	C	4	UCMDECB	DOM request ECB.
16	10	4	UCMRECB	RMS request ECB.
20	14	4	UCMLSTP	Address of event indication list (UCMEIL).
24	18	4	UCMWTOQ	Address of first WQE (system output queue).
28	1C	4	UCMRPYQ	Address of first RQE (reply queue element).
32	20	13	UCMRPYI	Reply ID assignment pattern (100 bit positions used).
45	2D	1	UCMRQLM	ID assignment limit.
46	2E	2	UCMWQLM	WQE buffer limit.
48	30	4	UCMRQECB	Reply request waiting ECB.
52	34	4	UCMWQECB	Buffer request waiting ECB.
56	38	2	UCMRQNR	Current RQE count.
58	3A	2	UCMWQNR	Current WQE count.
60	3C	4	UCMWQEND	Address of last WQE, or zero.
64	40	4	UCMPXA	Address of communications task ECB (IEECVTCB).
68	44	1	UCMMODE	Mode flags.
		.... 1...	UCMAMFA	Accept VARY command with MSTCONS operand from any MCS secondary console.
		.... .1..	UCMOGCE	Only graphics consoles exist.
		.... ..1.	UCMMCS	MCS generated with system.
		0000 0000	UCMFIX	MVT mode.
69	45	1	UCMCORE	WTO Purge routine switches.
70	46	1	UCMMODEL	System model number.
71	47	1	UCMINCR	Used by Console Initialization routine for error handling.
72	48	4	UCMVEA	Address of first UCM entry.
76	4C	4	UCMVEZ	Size of UCM entry.
80	50	4	UCMVEL	Address of last UCM entry.
84	54	56	UCMSAVE3	Save area for ED2 (refreshability).

140	8C	64	UCMSAVE4	Save area for CRA (refreshability).
204	CC	4	UCMR9SV	Save area for ED2 (refreshability).

UCM EXTENSION PREFIX TO UNIT CONTROL MODULE (UCM) BASE

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
-12	-12	2	UCM2WID	Terminal job ID of task for UCMWQECB.
-10	-10	2	UCM2RID	Terminal job ID of task for UCMRQECB.
-8	-8	4	UCM2PST	Address of POST validity checking.

MULTIPLE CONSOLE SUPPORT PREFIX TO UNIT CONTROL MODULE (UCM) BASE

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	4	UCMMCENT	Address of Master Console UCM entry.
4	4	72	UCMSAVE0	Resident and communications save area.
76	4C	4	UCMDOME	Address of first DOM element.
80	50	4	UCMWTOX	Address of WTO/R Exit routine (IEECVXIT).
84	54	2	UCMFLGS	System control flags.
		Byte 0		
		x...	UCMSYSA	Reserved.
		.1..	UCMSYSB	Hard copy support required.
		..1.	UCMSYSC	Commands to hard copy.
		...1	UCMSYSD	Console switch for master.
		.... 1...	UCMSYSE	No consoles exist.
		.... .1..	UCMSYSF	Graphic consoles exist.
		.... ..1.	UCMSYSG	Hard copy device SYSLOG.
		1	UCMSYSH	Timer present and operative.
		Byte 1		
		1...	UCMSYSI	WQE housekeeping needed.
		.1..	UCMSYSJ	Hard copy to be written.
		..1.	UCMSYSK	New console composite.
		...1	UCMSYSL	OPEN being issued to ring console bell.
		.... 1...	UCMSYSM	Failing console composite.
		.... .1..	UCMSYSN	Model 85 Operator Console with CRT Display.
		.... ..1.	UCMSYSO	Dummy attention by WTL.
		x		Reserved.
86	56	2	UCMOWTOR	Default values for old WTOR macros.
88	58	4	UCMCMID	Current message identification number.
92	5C	4	UCMHCUCM	Address of hard copy UCM entry, or zero.

96	60	1	UCMXCT	External request count.
97	61	3	UCMUEXIT	Address of user exit data, or zero.
100	64	2	UCMHRDRT	Hard copy routing code assignments.
102	66	2		Reserved.
104	68	24	UCMXSA	Parameter list area for SVC 72.
128	80	4	UCMQRTN	Address of ENQ entry point (IEECMENQ).
132	84	4	UCMRUTCK	Route checking field.
136	88	4	UCMDOMRT	Address of DOM routine entry point.
140	8C	4	UCMTPPTR	Address of save area for 2740 device support processor, or zero.
144	90	4	UCMNPECB	NIP ECB (posted when NIP routine's hard copy can be written).
148	94	4	UCMLOGAD	Address of MCS Log.
152	98	4	UCMDTINT	Dynamic display time interval.
156	9C	1	UCMSDS1	Status display flags.
		1...	UCMSDS1A	STCMDS to hard copy.
		.1..	UCMSDS1B	INCMDS to hard copy.
		..xx xxxx		Reserved.
157	9D	3		Reserved.
160	A0	4	UCM2EXT	Address of UCM extension.
164	A4	4	UCMMCENT	Address of MCS prefix.

UCM ENTRY INDIVIDUAL DEVICE MAP

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	4	UCMECB	I/O completion ECB, or address of I/O completion ECB for 2740.
4	4	4	UCMSRB	Address of resident processor module.
8	8	4	UCMDCB	Address of DCB.
12	C	4	UCMUCB	Address of UCB.
16	10	8	UCMNAME	Processing module name.
24	18	1	UCMSTS	Status flags.
		1...	UCMAF	Attention pending.
		.1..	UCMPF	Output pending.
		..1.	UCMBF	Device busy.
		...1	UCMCF	Close pending.
		.... 1...	UCMTA	Open pending.
		.... .1..	UCMTB	Dequeue appropriate output queue entries.
		.... ..x.		Reserved.
		1	UCMTC	Working on in-line status display.
25	19	1	UCMATR	Attribute flags.
		1...	UCMOF	WTO support.
		.1..	UCMIF	Attention support.
		..1.	UCMXF	External interruption support.
		...1	UCMUF	Device active.
		.... 1...	UCMLF	Load flag.
		.... .1..	UCMAT04	Device status to change.
		xx		Reserved.
26	1A	1	UCMID	Unique entry ID.
27	1B	1		Reserved.
28	1C	4	UCMXB	Address of DCM (Graphics), or zero.
32	20	2	UCMRTCD	Routing codes assigned to this console.
34	22	2		Reserved.
36	24	2	UCMOUTQ	Address of output queue.
40	28	2	UCMAUTH	Command code authorization.
		Byte 0		
		1...	UCMAUTH1	Command group 1 (System).
		.1..	UCMAUTH2	Command group 2 (I/O).
		..1.	UCMAUTH3	Command group 3 (CONS).
		...x xxxx		Reserved.
		Byte 1		Reserved.
42	2A	2	UCMDISP	Disposition flags.
		Byte 0		
		1...	UCMDISPA	Master console.
		.1..	UCMDISPB	Hard copy device/console.
		..1.	UCMDISPC	Graphics.
		...1	UCMDISPD	Output only.
		.... 1...	UCMDISPE	Console has full I/O capability.
		.... .1..	UCMDISPF	Console is in message stream only.

		.... ..1.	UCMDISPG	Console is in status display only.
		1	UCMDISPH	MCS.
		Byte 1		Reserved.
44	2C	4	UCMALTEN	Address of alternate input UCM entry.
48	30	4	UCMOAOEN	Address of alternate output.
52	34	4	UCMWLAST	Address of last WQE entry serviced in output queue.
56	38	4	UCMCOMPC	Address of other device if console is composite.
60	3C	2	UCMMSG	Message flags.
		Byte 0		
		1...	UCMMSGA	Monitor JOBNAMES requested.
		.1..	UCMMSGB	Monitor STATUS requested.
		..1.	UCMMSGC	Monitor ACTIVE.
		...x	UCMMSGD	Reserved.
		.... 1...	UCMMSGE	SHOW requested.
		.... .1..	UCMMSGF	Monitor SESS requested.
		.... ..x.	UCMMSGG	Reserved.
		x	UCMMSGH	Reserved.
		Byte 1		Reserved.
62	3D	1	UCMXOR	Zero.
63	3E	1	UCMDEVC	Device control flags.
		1...	UCMDEVA	Full screen on graphics console.
		.1..	UCMDEVB	Prepare command issued.
		..1.	UCMDEVCC	Tested for console switch.
		...1	UCMDEVD	DOM issued.
		.... 1...	DCMDEVE	I/O complete.
		.... .1..	DCMDEVF	Modified DCM for DOM.
		.... ..1.	DCMDEVG	Halt I/O issued on 2740.
		x	DCMDEVH	Reserved.
64	40	4	UCMMLAST	Address of last minor WQE serviced.
68	44	1	UCMSDS5	Status display flags.
		1...	UCMSDS5A	MLWTO line: need to keep writing.
		.1..	UCMSDS5B	In-line output pending.
		..1.	UCMSDS5C	Out-of-line output pending.
		...1	UCMSDS5D	Transient DCM blocked.
		.... 1...	UCMSDS5E	Transient DCM locked.
		.... .1..	UCMSDS5F	For CRT, UCMMLAST valid.
		.... ..1.	UCMSDS5G	I/O hardware in output only.
		x	UCMSDS5H	Reserved.
69	45	3	UCMRCT	Address of message routing control table.

UCM MESSAGE TEXT AND EVENT INDICATION LIST (EIL) AREAS

The following fields are used only if a user exit was specified.

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	128	UCMMSTXT	Message text (128 characters).
128	80	4	UCMROUTC	Routing codes.
132	84	4	UCMDESCD	Message descriptor codes.
136	88	64	UCMXTSAV	Save area for IEECMWSV interface.

The following fields constitute the event indication list.

0	0	1	UCMEIL	Length in doublewords.
1	1	1	UCMRPYL	Last assigned reply ID.
2	2	1	UCMRTCT	Route count.
3	3	1		Reserved.
4	4	4		Address of 2K NIP message buffer.
8	8	4		Address of external ECB.
12	C	4		Address of attention ECB.
16	10	4		Address of WTO/R ECB.
20	14	4		Address of DOM ECB.
24	18	4		Address of RMS ECB.
28	1C	Variable		List of all I/O ECB addresses. One word for each console device specified at system generation, with a minimum of 2 entries. The list is variable at SYSGEN only. Last entry has a high-order byte = X'80'. (Maximum of 128 bytes.)

The following field is only present when a 2740 is used as the system console.

Variable	72	UCMTPSAV	Save area for 2740 Device Support Processor.

WRITE QUEUE ELEMENT (WQE) FORMAT FOR MULTIPLE CONSOLE SUPPORT (SINGLE-LINE WTO)

The Write Queue Element (WQE) is a control block representing a message to be written to an operator's console. There are three forms of the WQE: the WQE (representing a single-line write-to-operator (WTO) message), the Major WQE (representing the first line of a multiple-line WTO message), and the Minor WQE (representing one or more lines following the first line of a multiple-line WTO message).

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1	WQEUSE	WQE use count.
1	1	3	WQELKPTR	Address of next WQE.
4	4	4	WQENBR	Message length.
8	8	Variable	WQETXT	Message text (maximum of 128 bytes).
136	88	1	WQEXA	Disposition flags.
		1...	WQEPURGE	Purge.
		.1..	WQEQFHC	Queue for hard copy.
		..1.	WQERQE	RQE exists for this WQE.
		...1	WQEQDFHC	Queued for hard copy.
		.... 1...	WQEXWTOR	WQE created for WTOR.
		xxx		Reserved.
137	89	2	WQETJID1	Time sharing user ID for swapped-out task.
139	8B	1	WQEAVAIL	Buffer flags.
		1...	RQEBUFA	Buffer is free.
		.1..	RQEBUFB	Buffer is in use.
		...1	RQEBUFD	Buffer obtained dynamically.
		.... 1...	RQEBUFE	Buffer has been serviced.
		..x. .xxx		Reserved.
140	8C	4	WQEXB	Reserved for rollout/rollin.
144	90	1	WQERTCT	Routed WQE count.
145	91	3	WQESEQN	24-bit ID sequence number.
148	94	2	WQEMCSF	MCS flags.
		Byte 0		
		1...	WQEMCSA	Routing or descriptor codes exist.
		.1..	WQEMCSB	UCM entry identifier passed in register 0.
		..1.	WQEMCSC	Command response (includes hard copy).
		...1	WQEMCSD	WQE WQEMSGTP field to be used for message identification.
		.... 1...	WQEMCSE	Accepted reply to a WTOR.
		.... .1..	WQEMCSF	Broadcast to all active consoles.
		.... ..1.	WQEMCSG	Queue for hard copy only.
		1	WQEMCSH	Queue to UCM entry passed in register 0.
		Byte 1		
		1...	WQEMCSI	Time stamp exists in message text.
		.... .1..	WQEMCSN	Bypass hard copy queuing.
		.... ..1.	WQEMCSO	Reserved for DOM function.
		1	WQEMCSP	Reserved for Graphics.
		.xxx x...		Reserved.
150	96	2	WQEMSGTP	Message flags.
		Byte 0		
		1...	WQEMSGTA	Monitor JOBNAMES.

		.1..	WQEMSGTB	Monitor STATUS.
		.... 1...	WQEMSGTE	Display SHOW.
		.... .1..	WQEMSGTF	Monitor SESS.
		..xx ..xx		Reserved.
		Byte 1		Reserved.
152	98	2	WQEROUT	Routing codes.
154	9A	2		Reserved.
156	9C	1	WQECMID	Unique UCM entry ID.
157	9D	1	WQEPKE	TCB key of WTO issuer.
158	9E	2		Reserved.
160	A0	2	WQEDESCD	Descriptor codes.
162	A2	2		Reserved.
164	A4	4	WQETIME	Timer element.

MAJOR WRITE QUEUE ELEMENT (NON-MCS)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1	WMJUC	Use count for WQE.
1	1	3	WMJNXT	Address of next WQE.
4	4	1	WMJMLW	MLWTO flags.
		1...	WMJMLWA	Message displayed.
		.1..	WMJMLWB	Major.
		..1.	WMJMLWC	Minor.
		...1	WMJMLWD	Chain altered.
		.... 1...	WMJMLWE	WTL issued.
		.... ..1.	WMJMLWG	Chain to be serviced.
		1	WMJMLWH	Minor queued has no text.
		.... .x..		Reserved.
5	5	1	WMJAREA	Identifier of the display area to which the message is to be routed.
6	6	2	WMJTXTL	Text length.
8	8	72	WMJTXT	Text.
80	50	40	WMJRES	Reserved.
120	78	2	WMJSER	Previous line types for error checking.
122	7A	2	WMJLTYP	Line type of message in text field.
		Byte 0		
		1...	WMJLTYPA	Control line.
		.1..	WMJLTYPB	Label line.
		..1.	WMJLTYPC	Data line.
		...1	WMJLTYPD	End indicator.
		 xxxx		Reserved.
		Byte 1		Reserved.
124	7C	4	WMJNXTM	Address of first minor WQE associated with this major WQE.
128	80	4	WMJTCB	Address of the TCB that issued the MLWTO.
132	84	4	WMJMSGN	Message identification number assigned to the MLWTO.
136	88	1	WMJDISP	Disposition flags.
		1...	WMJDISPA	Purge this WQE.
		.1..	WMJDISPB	Queue for hard copy.
		..1.	WMJDISPC	This WQE has an RQE.
		...1	WMJDISPD	Queued for hard copy.
		.... 1...	WMJDISPE	WQE is for a WTOR.
		xxx		Reserved.
137	89	2	WMJTJID	TSO user identifier.
139	8B	1	WMJBUF	Buffer status flags.
		1...	WMJBUFA	Buffer space available.
		.1..	WMJBUFB	Buffer space in use.
		..1.	WMJBUFC	Reserved.
		...1	WMJBUFD	Buffer space acquired by GETMAIN.
		.... 1...	WMJBUFE	WQE serviced.
		xxx		Reserved.
140	8C	4	WMJRORI	Information pertaining to rollout/rollin.

MAJOR WRITE QUEUE ELEMENT (MCS)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1	WMJMUC	WQE use count.
1	1	3	WMJMNXT	Address of next WQE.
4	4	1	WMJMMLW	MLWTO flags.
		1...	WMJMMLWA	Entire first minor available.
		.1..	WMJMMLWB	Major.
		..1.	WMJMMLWC	Minor.
		...1	WMJMMLWD	Chain altered.
		.... 1...	WMJMMLWE	WTL issued.
		.... .1..	WMJMMLWF	(For IEECMWSV) start at top of chain.
		.... ..1.	WMJMMLWG	Chain to be serviced.
		1	WMJMMLWH	Minor queued has no text.
5	5	1	WMJMAREA	Identifier of the display area to which the display is to be routed.
6	6	2	WMJMTXTL	Text length.
8	8	6	WMJMTS	Time stamp for hard copy messages.
14	E	1	WMJMPAD	Reserved.
15	F	4	WMJMRR	Routing codes for hard copy messages.
19	13	1	WMJMPAD1	Break for hard copy text.
20	14	72	WMJMTXT	Text of the message to be passed to the operator.
92	5C	18	WMJMRESA	Reserved.
110	6E	2	WMJMSER	Previous line types for error checking.
112	70	8	WMJMCONS	Framing indicators (for display consoles).
120	78	2	WMJMRESB	Reserved.
122	7A	2	WMJMLTYP	Line type of line in text field.
		Byte 0		
		1...	WMJMLTYPA	Control line.
		.1..	WMJMLTYPB	Label line.
		..1.	WMJMLTYPC	Data line.
		...1	WMJMLTYPD	End indicator.
		 xxxx		Reserved.
		Byte 1		Reserved.
124	7C	4	WMJMMIN	Address of first minor WQE chained to the major WQE.
128	80	4	WMJMTCB	Address of the TCB that issued the MLWTO.
132	84	4	WMJMMSGN	Message number assigned to the MLWTO.
136	88	1	WMJMDSP	Disposition flags.
		1...	WMJMDSPA	Purge the WQE.
		.1..	WMJMDSPB	Queue for hard copy.
		..1.	WMJMDSPC	This WQE has an RQE.
		...1	WMJMDSPD	Queued for hard copy.

		.... 1...	WMJMDSPE	WQE is for WTOR.
		xxx		Reserved.
137	89	2	WMJMTJID	TSO user identifier.
139	8B	1	WMJMBUF	Status flags of the buffer used by the WQE.
		1...	WMJMBUFA	Buffer space available.
		.1..	WMJMBUFB	Buffer in use.
		..1.	WMJMBUFC	Reserved.
		...1	WMJMBUFD	Buffer acquired by GETMAIN.
		.... 1...	WMJMBUFE	WQE serviced.
		xxx		Reserved.
140	8C	4	WMJMRORI	Information pertaining to rollout/rollin.
144	90	1	WMJMRTCT	Routed WQE count.
145	91	3	WMJMSEQ	Sequence number assigned to message.
148	94	1	WMJMCS1	MCS flags.
		1...	WMJMCS1A	Routing or descriptor codes exist.
		.1..	WMJMCS1B	UCM entry ID passed in register 0.
		..1.	WMJMCS1C	Command response (hard copy).
		...1	WMJMCS1D	WMJMMT1 field indicates message type.
		.... 1...	WMJMCS1E	Accepted reply to WTOR.
		.... .1..	WMJMCS1F	Queue to all active consoles.
		.... ..1.	WMJMCS1G	Queue to hard copy only.
		1	WMJMCS1H	Queue to UCM entry passed in register 0.
149	95	1	WMJMCS2	MCS flags.
		1...	WMJMCS2A	Time stamp in message text.
		.1..	WMJMCS2B	WQE represents a multiple-line message.
		.... .1..	WMJMCS2F	Bypass hard copy queuing.
		.... ..1.	WMJMCS2G	Reserved for DOM.
		1	WMJMCS2H	Reserved for graphics.
		..xx x...		Reserved.
150	96	1	WMJMMT1	Types of messages the WQE represents.
		1...	WMJMMT1A	Display JOBNAMES.
		.1..	WMJMMT1B	Display STATUS.
		..1.	WMJMMT1C	Monitor ACTIVE.
		...1	WMJMMT1D	Reserved.
		.... 1...	WMJMTT1E	SHOW requested.
		.... .1..	WMJMTT1F	Monitor SESS.
		xx		Reserved.
151	97	1	WMJMMT2	Reserved.
152	98	4	WMJMRTC	Routing codes assigned to message.
156	9C	1	WMJMUID	UCM entry ID.
157	9D	1	WMJMTID	TCB key of the task that issued the WTO/R.
158	9E	2	WMJMRESC	Reserved.
160	A0	4	WMJMDEC	Descriptor codes assigned to message.
164	A4	4	WMJMTIM	Timer element.

MINOR WRITE QUEUE ELEMENT (NON-MCS)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1	WMNUC	Use count for the minor WQE. This use count is set equal to the use count of the major WQE when the MLWTO is queued. As each console finishes with the line represented by the minor WQE, the use count is decremented by 1. When the use count equals zero, the WQE is ready to be removed from the system.
1	1	3	WMNEXT	Address of the next minor WQE.
4	4	1	WMNMLW	Flags pertaining to the message represented by the minor WQE.
		1...	WMNMLWA	Message displayed.
		.1..	WMNMLWB	Major.
		..1.	WMNMLWC	Minor.
		...1	WMNMLWD	Chain altered.
		.... 1...	WMNMLWE	WTL issued.
		.... ..1.	WMNMLWG	Chain to be serviced.
		1	WMNMLWH	GETMAIN issued for minor.
		.... .x..		Reserved.
5	5	2	WMNLT	Flags indicating the line type of the text field.
		Byte 0		
		1...	WMNLTA	Control line.
		.1..	WMNLTB	Label line.
		..1.	WMNLTC	Data line.
		...1	WMNLTD	End indicator.
		 xxxx		Reserved.
		Byte 1		Reserved.
7	7	1	WMNTXL	Length of message in text field.
8	8	4	WMNHCT	Hard copy identifier.
12	C	72	WMNTXT	Message to be passed to operator.
84	54	52	WMNRESA	Reserved.
136	88	1	WMNDISP	Disposition flags.
		1...	WMNDISPA	Purge this queue.
		.1..	WMNDISPB	Queue for hard copy.
		..1.	WMNDISPC	This WQE has an RQE.
		...1	WMNDISPD	Queued for hard copy.
		.... 1...	WMNDISPE	This WQE is for a WTOR.
		xxx		Reserved.
137	89	2	WMNTJID	TSO user identifier.
139	8B	1	WMNBUF	Flags indicating the status of the buffer used by the WQE.
		1...	WMNBUFA	Buffer available.
		.1..	WMNBUFB	Buffer in use.
		..1.	WMNBUFC	Reserved.
		...1	WMNBUFD	Buffer acquired by GETMAIN.
		.... 1...	WMNBUFE	WQE serviced.
		xxx		Reserved.
140	86	4	WMNRORI	Information pertaining to rollout/rollin.

MINOR WRITE QUEUE ELEMENT (MCS)

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1	WMNMUC1	Use count for the message in the Text 1 field.
1	1	3	WMNMNX1	Address of the second half of the WQE (address of WMNMUC2).
4	4	1	WMNMML1	MLWTO flags for the message in the Text 1 field.
		.1..	WMNMML1B	Major.
		..1.	WMNMML1C	Minor.
		...1	WMNMML1D	Chain altered.
		.... 1...	WMNMML1E	WTL issued.
		.... ..1.	WMNMML1G	Chain to be serviced.
		1	WMNMML1H	GETMAIN issued for minor.
		x... .x..		Reserved.
5	5	2	WMNMLT1	Line type of message in the Text 1 field.
		Byte 0		
		1...	WMNMLT1A	Control line.
		.1..	WMNMLT1B	Label line.
		..1.	WMNMLT1C	Data line.
		...1	WMNMLT1D	End indicator.
		 xxxx		Reserved.
		Byte 1		Reserved.
7	7	1	WMNMTL1	Length of message in the Text 1 field.
8	8	4	WMNMHCT1	Hard copy ID for the message in the Text 1 field.
12	C	72	WMNMTXT1	Text of the first message passed to the operator by this WQE.
84	54	1	WMNMUC2	Use count for the message in the Text 2 field.
85	55	3	WMNMNX2	Address of the next minor WQE.
88	58	1	WMNMML2	MLWTO flags for the message in the Text 2 field.
		.1..	WMNMML2B	Major.
		..1.	WMNMML2C	Minor.
		...1	WMNMML2D	Chain altered.
		.... 1...	WMNMML2E	WTL issued.
		.... ..1.	WMNMML2G	Chain to be serviced.
		1	WMNMML2H	GETMAIN issued for minor.
		x... .x..		Reserved.
89	59	2	WMNMLT2	Line type of message in the Text 2 field.
		Byte 0		
		1...	WMNMLT2A	Control line.
		.1..	WMNMLT2B	Label line.
		..1.	WMNMLT2C	Data line.
		...1	WMNMLT2D	End indicator.
		 xxxx		Reserved.
		Byte 1		Reserved.
91	5B	1	WMNMTL2	Length of message in the Text 2 field.
92	5C	4	WMNHCT2	Hard copy ID for the message in the Text 2 field.
96	60	72	WMNMTXT2	Text of the second message passed to the operator by this WQE.

WTO/R MACRO EXPANSION

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
WTOR Prefix				
-8	-8	1		Length of reply.
-7	-7	3		Address of requestor's reply buffer.
-4	-4	4		Address of requestor's reply ECB.
0	0	1		Zero.
1	1	1		Output message length + 4.
2	2	2		MCS flags.
		Byte 0		
		1...		Routing and descriptor fields exist. (New WTOR/R Expansion.)
		.1..		UCM entry identification passed in register 0.
		..1.		Command response.
		...1		Message Type Flags field exists.
		.... 1...		This WTO is a reply to a WTOR.
		.... .1..		Broadcast to all active consoles.
		.... ..1.		Queue for hard copy only.
		1		Queue to UCM entry passed in register 0.
		0000 0000		No routing or descriptor fields exist.
		Byte 1		
		1...		Time stamp exists in message text.
		.... .1..		Bypass hard copy queuing (protect key zero users only).
		.... ..1.		Reserved for DOM function.
		1		Reserved for graphics.
		.xxx x...		Reserved.
4	4	Variable		Message ID and message text area (maximum of 128 bytes).
n	n	2		16-bit descriptor code.
n	n	2		16-bit routing code.
n	n	1		Message Type Flags.
		1...		Monitor JOBNAMES.
		.1..		Monitor STATUS.
		.... 1...		Display SHOW.
		.... .1..		Monitor SESS.
		..xx ..xx		Reserved.
n	n	1		Reserved.
n	n	2		SVC 35.

MULTIPLE-LINE WTO MACRO EXPANSION

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1	WTOMK1	One-byte flag set to zeros. It signifies the start of a new text field.
1	1	2	WTOLNT1	Length of Text 1 + 4.
2	2	2	WTOMCSF	MCS flags.
		Byte 0		
		1...		Route and descriptor fields exist.
		.1..		UCM entry identification passed in register 0.
		..1.		Command response.
		...1		Message type flag field exists.
		.... 1...		This WTO represents a reply to a WTOR.
		.... .1..		Broadcast to all active consoles.
		.... ..1.		Queue for hard copy only.
		1		No route or descriptor code field exists.
		Byte 1		
		1...		Time stamp exists in message text.
		.1..		Multiple-line WTO.
		.... .1..		Bypass hard copy queuing (protect key zero users only).
		.... ..1.		Reserved for DOM function.
		1		Reserved for graphics.
		..xx x...		Reserved.
4	4	Variable	WTOTXT1	First line of WTO text.
Variable		2	WTODESC	Descriptor codes.
		Byte 0		
		1...		System failure - another IPL is required.
		.1..		Immediate operator action is required.
		..1.		Eventual action required.
		...1		System status is indicated by the message.
		.... 1...		Immediate command response - for error and non-error messages that are written as a result of an operator or system command.
		.... .1..		Job status indicated by the message.
		.... ..1.		Application program/processor - for messages issued by a problem program and processors executed as problem programs.
		1		Operator request for status information.
		Byte 1		
		1...		Out-of-line message.
		.xxx xxxx		Reserved.
Variable		2	WTOROUT	Routing codes.
		Byte 0		
		1...		Master console.
		.1..		Master console informational.
		..1.		Tape pool console.
		...1		Direct access pool console.
		.... 1...		Tape library pool console.
		.... .1..		Disk library pool console.
		.... ..1.		Unit record pool console.
		1		Teleprocessing control.
		Byte 1		
		1...		System security.
		.1..		System error/maintenance.

	..1.		Programmer information console.
	...x ...x		Reserved.
	.... xxx.		User routing codes.
Variable	2	WTOMSGT	Message type flags.
	Byte 0		
	1...		Display JOBNAMES.
	.1..		Display STATUS.
	..1.		Monitor ACTIVE.
	.... .1..		Display SHOW.
	.... ..1.		Monitor SESS.
	...x x..x		Reserved.
	Byte 1		Reserved.
Variable	2	WTOLT1	Line type of Text 1.
Variable	1	WTOAID	Area identifier for routing of the out-of-line status display represented by the multiple-line WTO.
Variable	1	WTOLCT	Number of lines in list.
Variable	1	WTOMK2	Next line marker.
Variable	1	WTOLNT2	Length of Text 2 + 4.
Variable	2	WTOLT2	Line type of Text 2.
Variable	Variable	WTOTXT2	Second line of WTO text.
Variable	1	WTOMK3	Next line marker.
Variable	1	WTOLNT3	Length of Text 3 + 4.
Variable	2	WTOLT3	Line type of Text 3.
Variable	Variable	WTOTXT3	Third line of WTO text.

Each subsequent text field is preceeded by four bytes of information.

MACHINE CHECK RECORD FOR SER0 AND SER1

SER0 and SER1 produce two types of record entries corresponding to the two types of errors processed: machine-check and channel errors. Record size varies with the type of record and the machine model.

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1	CLASRC	
		1...		Machine check record.
		xxxx		MCH.
		xxxx ..1.		Converted MCH.
		xxxx 1...		SER1.
		xxxx 1..1		SER0.
		xxxx 1.1.		Converted SER1.
		xxxx 1.11		Converted SER0.
1	1	1	SYSREL	
		...x xxxx		OS.
		..1x xxxx		DOS.
		Bits 3-7		
		0-1F		Release level 0-31.
2	2	4	SWITCHES	
		Byte 0		
		1...		More records follow.
		0...		Last record.
		.1..		Time-of-day clock.
		..1.		Extended control mode.
		.... 1...		Time macro used (HHMMSS).
		...x .xxx		Reserved.
		Byte 1		
		1...		Short form of record.
		.1..		Record incomplete.
		..1.		System terminated.
		...1		First record of two record recording.
		.... 1...		Channel record included.
		.... .1..		Portion of data overlayed.
		.... ..1.		External machine check.
		x		Reserved.
		Bytes 2-3		Reserved.
6	6	1	RCDCNT	
		xxxx		Sequence number of a physical record.
		 xxxx		Total number of physical records in this logical record.
7	7	1		Reserved.
8	8	4	DATE	The system date the record was made.
12	C	4	TIME	The system time the record was made.
16	10	1		Reserved.
17	11	3	CPUSER	CPU serial number (System/370 only).
20	14	2	CPUID	CPU identifier.

22	16	2	MCELLNG	Maximum machine check extended log length (System/370 only).
24	18	8	PROGID	The module name of the program being processed and/or requesting service at the time of the error.
32	20	8	JOBID	The name assigned to the job being executed at the time of the failure.
40	28	8	PSW	The machine check old PSW.
48	30	Variable	LOGOUT	Register contents and hardware logout information. Logout size and format is model dependent.
Variable		Variable	DAMAGE	Recovery Management Support's assessment of damage to the system.

STORAGE UTILIZATION BLOCK (SUB)

The storage utilization block (SUB) contains information pertaining to graphics consoles that have been designated system operator's consoles. The SUB also contains RMS and SER channel programs used in system recovery procedures.

Offset Dec	Offset Hex	Bits and Byte Pattern	Field Name	Field Description, Contents, Meaning
0	0	1	SUBDAIO	Status flags of the I/O routine.
		1...		I/O routine in use.
		.1..		Read in process.
		..1.		Write in process.
		...1		Expecting control record.
		.... 1...		PFK write in process.
1	1	1	SUBFLGS	Routing flags.
		1...		Exit to Processor 1 routine.
		.1..		Initialize SUB on first entry.
		.... 1...		Write PFK updates.
		.... .1..		PFK keys are supported.
2	2	2	SUBNUM	Number of transient consoles.

The following address constants are always generated:

4	4	4	SUBDOM	Address of the SUBKEY field of the SUB.
8	8	4	SUBPBASE	Address of the base DCM.
12	C	4	SUBTIOT	Address of the task I/O table (in the transient DCM control block of the SUB).
16	10	4	SUBSPACE	Address of the SUB work area (SUBAREA field of the SUB).

The following address constants are only generated if transient DCMs have been defined for the system:

20	14	4	SUBQUE	Address of the DCM request queue (SUBREQ field of the SUB).
24	18	4	SUBBLDL	Address of the DCM BLDL list (SUBTTR field of the SUB).
28	1C	4	SUBBLK	Address of the DCB (in the transient DCM control block of the SUB).
32	20	4	SUBPFKAD	Address of the PFK area.
36	24	Variable	SUBAREA	Work areas required by the SUB including a CXSA save area, a work area, and two register save areas.
n	n	Variable	SUBKEY	Delete-operator-message (DOM) elements. Each DOM element is two bytes in length; there is one element for each display console in the system.
n	n	Variable	SUBREQ	DCM request elements. Each element is two fullwords in length; there is one element for each console in the system that has a transient DCM. The first word contains the address of the console's UCMENTRY; the second word contains the address of the DCM in auxiliary storage.
n	n	Variable	SUBTTR	Marks the end of the request queue.

The following areas contain RMS/SER channel programs used in system recovery procedures. The only areas expanded are those areas that support the type of display consoles in the systems.

n	n	Variable	SUB2260	RMS/SER channel programs for 2260 display consoles. This area is expanded only if the system includes one or more 2260 display consoles.
n	n	Variable	SUB5450	RMS/SER channel programs for the Model 85 and Model 165 operator consoles. This area is expanded only if the system includes a Model 85 or Model 165 display operator console.
n	n	Variable	SUB2250	RMS/SER channel programs for the 2250, Models 91 and 195 display operator consoles. This area is expanded only if the system includes one or more of these consoles.
n	n	Variable		Transient DCM Control Blocks - contain information pertaining to reading and writing transient DCMs. This area is expanded only if transient DCMs are included in the system.
n	n	Variable		BASE DCM - executable code which gains control whenever an STIMER interval elapses. This routine sets flags in the DCM and posts the DCM for the Timer Interpreter routine.

```
JOB SAMPLDMP          STEP GO              TIME 001030   DATE 99366                                    PAGE 0001

COMPLETION CODE       USER = 0100

PSW AT ENTRY TO ABEND  FFF5000D 4005AFD6

TCB  03D340  RBP   0003BC68   PIE    00000000   DEB  0003B5D4   TIO 0003D850   CMP   80000064   TRN  00000000
             MSS   01040268   PK-FLG F0850400   FLG  00001B1B   LLS 0003E2D0   JLB   00000000   JPQ  0003D948
             FSA   01068F68   TCB    0003B740   TME  00000000   JST 0003D340   NTC   00000000   OTC  0003D770
             LTC   0003B660   IQE    00000000   ECB  0003DCF4   STA 20000000   D-PQE 00040258   SQS  0003B460
             NSTAE 00000000   TCT    0003BD00   USER 00000000   DAR 00000000   RESV  00000000   JSCB 8703D538


ACTIVE RBS

PRB  03E7B0  RESV   00000000   APSW   00000000   WC-SZ-STAB 00040082   FL-CDE 0003FEB0   PSW FFF5000D 4005AFD6
             Q/TTR  00000000   WT-LNK 0003D340

SVRB 03D478  TAB-LN 00180220   APSW   F9F0F1C3   WC-SZ-STAB 0012D002   TQN    00000000   PSW 00040033 5001902A
             Q/TTR  00011113   WT-LNK 0003E7B0
             RG 0-7   00000030   80000064   0003E368   00000001   0003D968   0003D770   0003D8BC   0003E1D8
             RG 8-15  0003DCD0   0003BD00   0003DCF8   00000000   4007D482   0105AD50   0001D4CA   0000000C
             EXTSA    000029BE   8F068E70   00000000   00000000   FF030000   0003D4F4   0003D4FC   E2E8E2C9
                      C5C1F0F1   C9C5C140   C1C2C5D5   C4F90000

SVRB 03BC68  TAB-LN 005803C8   APSW   F1F0F5C1   WC-SZ-STAB 0012D002   TQN    00000000   PSW FF04000C 5005BFA6
             Q/TTR  00010B0D   WT-LNK 0003D478
             RG 0-7   001058B0   0003D4D8   80018F42   00001A880   0003D340   0003D478   0403D340   0003D478
             RG 8-15  0003D340   40018E8A   0003D340   8F068E70   0003D894   0003D4FC   4001A1A8   00000000
             EXTSA    E2E8E2C9   C5C1F0F1   00000000   00000000   00000000   00000000   00000000   00000000
                      00000000   00000000   000C1901   80000C18


LOAD LIST

        NE 0003E488   RSP-CDE 0103D948        NE 0003E750   RSP-CDE 0103E650        NE 0003E898   RSP-CDE 0103FB68
        NE 0003E8A0   RSP-CDE 0103FA98        NE 0003EBE0   RSP-CDE 0103FA68        NE 0003EBE8   RSP-CDE 0103FA38
        NE 00000000   RSP-CDE 0103F908


CDE

     03FEB0       ATR1 0B   NCDE 03ECA8   ROC-RB 0003E668   NM GO         USE 03   EPA 05AD50   ATR2 20   XL/MJ 03FEA0
     03D948       ATR1 03   NCDE 03E650   ROC-RB 00000000   NM IGC0A05A   USE 01   EPA 05B858   ATR2 20   XL/MJ 03D838
     03E650       ATR1 03   NCDE 03FEB0   ROC-RB 00000000   NM IGC0A05A   USE 01   EPA 05B058   ATR2 20   XL/MJ 03E490
     03FB68       ATR1 B0   NCDE 03FBA8   ROC-RB 00000000   NM IGG019CD   USE 02   EPA 07CC90   ATR2 20   XL/MJ 03FB58
     03FA98       ATR1 B0   NCDE 03FAD8   ROC-RB 00000000   NM IGG019CJ   USE 02   EPA 07C3B0   ATR2 20   XL/MJ 03FA88
     03FA68       ATR1 B0   NCDE 03FA98   ROC-RB 00000000   NM IGG019BA   USE 02   EPA 07C208   ATR2 20   XL/MJ 03FA58
     03FA38       ATR1 B0   NCDE 03FA68   ROC-RB 00000000   NM IGG019BB   USE 02   EPA 07C828   ATR2 20   XL/MJ 03FA28
     03F908       ATR1 B0   NCDE 03F938   ROC-RB 00000000   NM IGG019AV   USE 01   EPA 07BBB0   ATR2 20   XL/MJ 03F8F8


XL                                                LN          ADR          LN          ADR          LN          ADR

     03FEA0   SZ 00000010   NO 00000001        800002B0    0005AD50
```

```
     03D838   SZ 00000010   NO 00000001       800007A8    0005B858
     03E490   SZ 00000010   NO 00000001       800007A8    0005B058
     03FB58   SZ 00000010   NO 00000001       80000270    0007CC90
     03FA88   SZ 00000010   NO 00000001       80000220    0007C3B0
     03FA58   SZ 00000010   NO 00000001       800001A8    0007C208
     03FA28   SZ 00000010   NO 00000001       80000128    0007C828
     03F8F8   SZ 00000010   NO 00000001       80000080    0007BBB0


DEB

03B5A0                                        000065A8 000065A8 000065A8 0007C3B0   *.........o.....................c.*
03B5C0   000065A8 00000000 00000106 00002BE0  11000000 0403D340 1003D678 88000000   *.....................L ..o.....*
03B5E0   8F000000 01000000 1B000000 FF068E70  0403B5B0 1800206C 00000003 0005000D   *.....................................*
03B600   00040064 00010001 00000000 00000000  0000007D C2C2C2C1 C3D1C3C4 00000000   *.....................BBBACJCD.....*
03B620   00000000 00000000 00000000 00000000  00000000 002307D0                     *.....................#.............*


DEB

03D660                     00000000 00000000  00003BE0 07000000 0003D340 00000000   *.......;.....................L ....*
03D680   08000000 00000000 00000000 00000000  0F05AE94 00000000 00000300 00000000   *.....................................*


TIOT   JOB  SAMPLDMP   STEP GO                PROC  TEST
     DD             14040140    PGM=*.DD     000A1400    800028A8
     DD             14000040    SNAPPER      000A1800    00000000
     DD             14040140    SYSABEND     000B0F00    8000206C
     DD             14040100    DELETE       000B1A00    800028A8


MSS            ************ SPQE ************      *************** DQE ***************      ******* FQE ********
               FLGS  NSPQE     SPID     DQE          BLK       FQE       LN       NDQE          NFQE         LN

     040268    00    040398    251      03ED08     0005A800 0005A800 00000800 0003DCA0      00000000     00000550
                                                   0005B000 0005B000 00000800 0003D828      00000000     00000058
                                                   0005B800 0005B800 00000800 00000000      00000000     00000058
     040398    00    0401B8    252      0401C0     00069000 00069000 00000800 00000000      00000000     000007F8
     0401B8    C0    000000    000      040210
     040210    60    000000    000      03FF78     00063800 00068800 00000800 0003D650      00000000     00000538
                                                   00068000 00068000 00000800 00000000      00000000     000001A0

D-PQE  00040258   FIRST 00040218   LAST 00040218
PQE  040218   FFB 0005C000   LFB 0005C000   NPQ 00000000   PPQ 00000000
              TCB 0003D770   RSI 0000F000   RAD 0005A800   FLG 0000

FBQE 05C000   NFB 00040218   PFB 00040218   SZ 0000C000


QCB TRACE

MAJ 03DC88   NMAJ 00000000   PMAJ 0001ECB8   FMIN 0003DC28   NM  SYSIEA01

MIN 03DC28   FQEL 0003DC18   PMIN 0003DC88   NMIN 00000000   NM F0  IEA

             NQEL 00000000   PQEL 0003DC28   TCB  0003D340   SVRB 0003BC68
```

SAVE AREA TRACE

INTERRUPT AT 05AFD6

PROCEEDING BACK VIA REG 13

```
SA   05AD50  WD1 90ECD00C   HSA 18DF4500   LSA D012E2E4   RET C2404040   EPA 40404110   R0  D2860A29
             R1  12FF4780   R2  D02E0700   R3  47F0D028   R4  80000065   R5  5810D024   R6  0A0D41F0
             R7  D03847F0   R8  F0243088   R9  0005ADA4   R10 00000000   R11 00000000   R12 00000000
```

NUCLEUS

```
000000   00000000 00000000 00000000 00000000   0001A880 00000000 01040080 800755CE   *.....................................*
000020   FF040001 4002645C 00000000 00000000   0000FF00 00000000 FF060233 80000000   *.....d*.....................3.....*
000040   00018A38 0C000000 00006E90 0001A880   083C2CD5 0000F198 00040000 00016EE0   *.....8.....n.....1.....n.....*
000060   00040000 00017690 00040000 00016FDE   00000000 0002229E8 00040000 00016F6A   *.....v.....v.....¥.....¥.....*
000080   000229E0 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....¥.....*
0000A0   00000000 00000000 10000060 00034148   FF000000 00000000 00000000 00000000   *.....-.....A.....*
0000C0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....*
      LINES 0000E0-000140 SAME AS ABOVE
000160   00000000 00000000 00000000 82000170   00040000 00074DC0 00000000 00000000   *.....b.....*
000180   0004000C 5005BFA6 0000018A 018A018A   FF000190 FF000190 00068800 00068D38   *.....*
0001A0   000000C0 0005BB1C 0003D340 9005B9A8   0005B858 80038468 00038501 00DF4500   *.....L .....H.....*
0001C0   0000078A A005B99C 40018A42 00068D4C   4005BB38 00078828 00000000 00000000   *.....H.....*
0001E0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....*
      LINE 000200 SAME AS ABOVE
000220   000068F0 00000000 07000700 07000700   07000700 07000700 07000700 07000700   *...0.....*
000240   07000700 07000700 0000FF00 00000008   07010000 00F0F0F0 10000808 00000000   *.....000.....*
000260   0000FF00 00010010 2B020000 00F0F0F1   56904092 00000000 0008FF00 00020018   *.....001..-.....*
000280   06030000 00F0F0F2 10800809 00000000   06007CF8 00007D00 0008FF00 00030018   *.....002.....8.....*
0002A0   08040000 00F0F0F3 1000080B 00000000   06007D10 00007D18 0000FF00 00050010   *.....003.....*
0002C0   2B050000 00F0F0F5 56904092 00000000   0000FF00 00060010 2B060000 00F0F0F6   *.....005.....006*
0002E0   56904092 00000000 0000FF00 00070010   2B070000 00F0F0F7 56904092 00000000   *..-.....007..-.....*
000300   0000FF00 00080010 2B080000 00F0F0F8   56904092 00000000 0000FF82 00090008   *.....008..-.....*
000320   04090004 00F0F0F9 10000820 52DC0000   0000FF30 00DC0008 0D0A0000 00F0F0C3   *.....009.....00C*
000340   10000801 52CC0000 0000FF80 000D0008   0D0B0000 00F0F0C4 10000802 00000000   *.....00D.....*
000360   0000FF88 000E0008 070C0000 00F0F0C5   10000808 52DC0000 0008FF00 000F0020   *.....00E.....-.....*
000380   070D0000 00F0F0C6 10800808 00000000   0200038E 00007D20 0000FF00 00100008   *.....00F.....*
0003A0   070E0000 00F0F1F0 10000808 00000000   0008FF00 00120018 0B0F0000 00F0F1F2   *.....010.....012*
0003C0   10010806 00000000 06007D30 00007D38   0008FF00 00130018 0B100000 00F0F1F3   *.....013*
0003E0   1001080C 00000000 06007D48 00007D50   0000FF00 00180008 07110000 00F0F1F8   *.....018*
000400   10000808 00000000 0000FF00 00190008   0D120000 00F0F1F9 10000801 00000000   *.....019.....*
000420   0000FF00 001A0008 0D130000 00F0F1C1   10000802 00000000 0000FF00 001F0008   *.....01A.....*
000440   04140000 00F0F1C6 10000820 00000000   0000FF00 00210010 29150000 00F0F2F1   *.....01F.....021*
000460   55114011 00000000 0000FF00 00220010   29160000 00F0F2F2 51114051 00000000   *.....022.....*
000480   0000FF00 00230010 29170000 00F0F2F3   55114011 00000000 0000FF00 00240010   *.....023.. .....*
0004A0   29180000 00F0F2F4 55114011 00000000   0000FF00 00250010 29190000 00F0F2F5   *.....024.....025*
0004C0   55114011 00000000 0000FF00 00260010   291A0000 00F0F2F6 51114011 00000000   *.. .....026.....*
0004E0   0000FF00 00270010 291B0000 00F0F2F7   55114011 00000000 0000FF00 00280008   *.....027.. .....*
000500   071C0000 00F0F2F8 10000808 00000000   0000FF00 00290008 0D1D0000 00F0F2F9   *.....028.....029*
```

```
002780  00000000 00000000 00000000 00009798  00C0FF00 02154058 01E40200 00F2F1F5  *.....................U...215*
0027A0  30C02008 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
0027C0  00000000 00000000 00000000 000097E8  0000FF00 02164058 01E40200 00F2F1F6  *.............Y.....U...216*
0027E0  30C02008 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
002800  00000000 00000000 00000000 00009838  0000FF00 02174058 01E40200 00F2F1F7  *.....................8U...217*
002820  30C02008 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
002840  00000000 00000000 00000000 00009888  0000FF00 021F0008 04ED0200 00F2F1C6  *.....................h....21F*
002860  10000820 00000000 0001FF88 02300048  01EE0300 04F2F3F0 30E02008 52EC0000  *.....................0.....*
002880  00000000 F7F7F7F7 F7F70800 00040100  00000000 00000000 00000000 00000000  *....777777.....................*
0028A0  01000000 000098D8 F001FF88 02310048  01EE0300 04F2F3F1 30E02008 52EC0000  *.....Q0.....h.....231.....*
0028C0  00000000 F9F9F9F9 F9F90801 00010100  00000000 00000000 00000000 15000007  *....999999.....................*
0028E0  02000000 00009928 0001FF00 02324048  01EE0300 00F2F3F2 30E02008 00000000  *.....................232.....*
002900  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
002920  00000000 00009978 F001FF86 02330048  01EE0300 00F2F3F3 30E02008 52EC0020  *.....Q.....h.....233.....*
002940  00400000 E2E8E2D9 C5E20800 00040100  00000000 00033F74 00000000 0F000805  *. ..SYSRES.....h...............*
002960  00000000 000099C8 F001FF84 02340048  01EE0300 04F2F3F4 30E02008 52EC0004  *.....H0.....h.....234.....*
002980  00400100 E2E8E2D3 C9C20801 00040100  00000000 0005A564 00000000 24000B0F  *. ..SYSLIB.....h...............*
0029A0  00000000 00009A18 0001FF00 02354048  01EE0300 00F2F3F5 30E02008 00000000  *.....................235.....*
0029C0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
0029E0  00000000 00009A68 0001FF00 02364048  01EE0300 00F2F3F6 30E02008 00000000  *.....................236.....*
002A00  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
002A20  00000000 00009AB8 0001FF00 02374048  01EE0300 00F2F3F7 30E02008 00000000  *.....................237.....*
002A40  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
002A60  00000000 00009B08 0000FF00 02404060  01F70200 00F2F4F0 30E02008 00000000  *.....................7...240.....*
002A80  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
002AA0  00000000 00009B58 0000FF00 02414060  01F70200 00F2F4F1 30E02008 00000000  *.....................7...241.....*
002AC0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
002AE0  00000000 00009BA8 0000FF00 02424060  01F70200 00F2F4F2 30E02008 00000000  *.....................7...242.....*
002B00  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
002B20  00000000 00009BF8 0000FF00 02434060  01F70200 00F2F4F3 30E02008 00000000  *.....8.....h.....7...243.....*
002B40  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
002B60  00000000 00009C48 0000FF00 02444060  01F70200 00F2F4F4 30E02008 00000000  *.....................7...244.....*
002B80  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
002BA0  00000000 00009C98 0000FF00 02454060  01F70200 00F2F4F5 30E02008 00000000  *.....................7...245.....*
002BC0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
002BE0  00000000 00009CE8 0000FF00 02464060  01F70200 00F2F4F6 30E02008 00000000  *.....Y.....h.....7...246.....*
002C00  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
002C20  00000000 00009D38 0000FF00 02474060  01F70200 00F2F4F7 30E02008 00000000  *.....................7...247.....*
002C40  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
002C60  00000000 00009D88 0008FF00 02504068  01000200 00F2F5F0 30502009 00000000  *.....................&...250.....*
002C80  18009E40 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *...h............................*
002CA0  00000000 00009E08 0008FF00 02514068  01000200 00F2F5F1 30502009 00000000  *.....................&...251.....*
002CC0  18009EE8 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *...Y............................*
002CE0  00000000 00009EB0 0008FF00 02524068  01000200 00F2F5F2 30502009 00000000  *.....................&...252.....*
002D00  18009F90 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
002D20  00000000 00009F58 0008FF00 02534068  01000200 00F2F5F3 30502009 00000000  *.....................&...253.....*
002D40  1800A038 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
002D60  00000000 0000A000 0008FF00 02544068  01000200 00F2F5F4 30502009 00000000  *.....................&...254.....*
002D80  1800A0E0 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
002DA0  00000000 0000A0A8 0008FF00 02554068  01000200 00F2F5F5 30502009 00000000  *.....................&...255.....*
002DC0  1800A188 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
002DE0  00000000 0000A150 0008FF00 02564068  01000200 00F2F5F6 30502009 00000000  *.....................&...256.....*
002E00  1800A230 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
002E20  00000000 0000A1F8 0008FF00 02574068  01000200 00F2F5F7 30502009 00000000  *.....8.....h.....&...257.....*
002E40  1800A2D8 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *...Q............................*
```

```
0050C0  00000000 00000000 0000CE98 0000FF00  05504070 01700600 00F5F5F0 30C02008  *.....................550....*
0050E0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
005100  00000000 00000000 0000CEE8 0000FF00  05514070 01700600 00F5F5F1 30C02008  *.........Y.....551....*
005120  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
005140  00000000 00000000 0000CF38 0000FF00  05524070 01700600 00F5F5F2 30C02008  *.....................552....*
005160  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
005180  00000000 00000000 0000CF88 0000FF00  05534070 01700600 00F5F5F3 30C02008  *.....................553....*
0051A0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
0051C0  00000000 00000000 0000CFD8 0000FF00  05544070 01700600 00F5F5F4 30C02008  *..........Q.....554....*
0051E0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
005200  00000000 00000000 0000D028 0000FF00  05554070 01700600 00F5F5F5 30C02008  *.....................555....*
005220  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
005240  00000000 00000000 0000D078 0000FF00  05564070 01700600 00F5F5F6 30C02008  *.....................556....*
005260  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
005280  00000000 00000000 0000D0C8 0000FF00  05574070 01700600 00F5F5F7 30C02008  *..........H.....557....*
0052A0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
0052C0  00000000 00000000 0000D118 531C2928  FF01A3A8 00033F74 00018380 52FC2928  *..........J.....t.....)(*
0052E0  FF0196D0 00033F74 00017FC0 52DC206C  FF068E20 1B03B5D4 F003D340 530C2868  *.....t.....M0.L ....(h*
005300  FF059F00 0003D334 F003D3C0 52CC2498  FF03E318 FF03DBC4 000186C8 532C0000  *.....L.0.L.....T...D...H.....*
005320  FF000000 00000000 00000000 533C0000  FF000000 00000000 00000000 534C0000  *................................*
005340  FF000000 00000000 00000000 535C0000  FF000000 00000000 00000000 536C0000  *................................*
005360  FF000000 00000000 00000000 537C0000  FF000000 00000000 00000000 538C0000  *................................*
005380  FF000000 00000000 00000000 539C0000  FF000000 00000000 00000000 53AC0000  *................................*
0053A0  FF000000 00000000 00000000 53BC0000  FF000000 00000000 00000000 53CC0000  *................................*
0053C0  FF000000 00000000 00000000 53DC0000  FF000000 00000000 00000000 53EC0000  *................................*
0053E0  FF000000 00000000 00000000 53FC0000  FF000000 00000000 00000000 540C0000  *................................*
005400  FF000000 00000000 00000000 541C0000  FF000000 00000000 00000000 542C0000  *................................*
005420  FF000000 00000000 00000000 543C0000  FF000000 00000000 00000000 544C0000  *................................*
005440  FF000000 00000000 00000000 545C0000  FF000000 00000000 00000000 546C0000  *................................*
005460  FF000000 00000000 00000000 547C0000  FF000000 00000000 00000000 548C0000  *................................*
005480  FF000000 00000000 00000000 549C0000  FF000000 00000000 00000000 54AC0000  *................................*
0054A0  FF000000 00000000 00000000 54BC0000  FF000000 00000000 00000000 54CC0000  *................................*
0054C0  FF000000 00000000 00000000 54DC0000  FF000000 00000000 00000000 54EC0000  *................................*
0054E0  FF000000 00000000 00000000 54FC0000  FF000000 00000000 00000000 550C0000  *................................*
005500  FF000000 00000000 00000000 551C0000  FF000000 00000000 00000000 552C0000  *................................*
005520  FF000000 00000000 00000000 553C0000  FF000000 00000000 00000000 554C0000  *................................*
005540  FF000000 00000000 00000000 555C0000  FF000000 00000000 00000000 556C0000  *................................*
005560  FF000000 00000000 00000000 557C0000  FF000000 00000000 00000000 558C0000  *................................*
005580  FF000000 00000000 00000000 559C0000  FF000000 00000000 00000000 55AC0000  *................................*
0055A0  FF000000 00000000 00000000 55BC0000  FF000000 00000000 00000000 55CC0000  *................................*
0055C0  FF000000 00000000 00000000 55DC0000  FF000000 00000000 00000000 55EC0000  *................................*
0055E0  FF000000 00000000 00000000 55FC0000  FF000000 00000000 00000000 560C0000  *................................*
005600  FF000000 00000000 00000000 561C0000  FF000000 00000000 00000000 562C0000  *................................*
005620  FF000000 00000000 00000000 563C0000  FF000000 00000000 00000000 564C0000  *................................*
005640  FF000000 00000000 00000000 565C0000  FF000000 00000000 00000000 566C0000  *................................*
005660  FF000000 00000000 00000000 567C0000  FF000000 00000000 00000000 568C0000  *................................*
005680  FF000000 00000000 00000000 569C0000  FF000000 00000000 00000000 56AC0000  *................................*
0056A0  FF000000 00000000 00000000 56BC0000  FF000000 00000000 00000000 56CC0000  *................................*
0056C0  FF000000 00000000 00000000 56DC0000  FF000000 00000000 00000000 56EC0000  *................................*
0056E0  FF000000 00000000 00000000 56FC0000  FF000000 00000000 00000000 570C0000  *................................*
005700  FF000000 00000000 00000000 571C0000  FF000000 00000000 00000000 572C0000  *................................*
005720  FF000000 00000000 00000000 573C0000  FF000000 00000000 00000000 574C0000  *................................*
005740  FF000000 00000000 00000000 575C0000  FF000000 00000000 00000000 576C0000  *................................*
005760  FF000000 00000000 00000000 577C0000  FF000000 00000000 00000000 578C0000  *................................*
005780  FF000000 00000000 00000000 579C0000  FF000000 00000000 00000000 57AC0000  *................................*
```

```
00E320   D203D04C 701047F0 F2BE9180 F6994780   F2B858B0 F6581BDB 06B044B0 F2B247F0   *K.....02...6...2...6.....2..0*
00E340   F1A4D700 D000D000 9640F698 1BDD47F8   F2C247F0 F35E47F0 F36647F0 F2DE47F0   *1.P.....6....,82B.03.,03.,02.,0*
00E360   F2E647F0 F35E47F0 F37647F0 F3C69680   F69A47F0 F2EA9620 F69A9610 F69845A0   *2W.03..03..03F..6..02...6...6...*
00E380   F4C89106 00454780 F3029180 F6974710   F31A48A0 F6EA12AA 4780F31A 06A040A0   *4H.....3...6...3...6.....3... .*
00E3A0   F6EA41C0 000847F0 F3DA9104 F6984710   F3529106 00454780 F34A9604 200DD707   *6.....03...6...3.....3.....P.*
00E3C0   50005000 50750000 96015004 9140F698   4710F34A D203D048 500441C0 000C47F0   *.............. 6...3.K.........0*
00E3E0   F3DA94FB 50049601 500447F0 F34A9680   F69A47F0 F36A9620 F69A45A0 F4C841C0   *3.........03...6..03...6...4H..*
00E400   000047F0 F3DA9640 F69A45A0 F4C89102   F6984710 F39A9120 70124780 F39A9106   *...03...6...4H..6...3.....3...*
00E420   70064790 F39A9602 00449108 F6984780   F3AE9180 F6994710 F3AE94F9 00459140   *....3.....6...3...6...3..9... *
00E440   F6994710 F3BE41C0 000447F0 F3DA41C0   001047F0 F3DA9610 F69A9610 F69845A0   *6...3.....03.....03...6...6...*
00E460   F4C841C0 000047F0 F3DA1BAA 9140F698   4710F3FE 41A00008 41E0D028 58B09028   *4H.....03.... 6...3.............*
00E480   489B0000 12994780 F4704720 F4689180   F69947E0 F4389140 F6984710 F4389566   *.........4...4...6...4.. 6...4...*
00E4A0   D0074740 F4384780 F4325840 F66C58A0   401012AA 4780F438 D26FA000 D05847F0   *.... 4...4...6.........4.K.....0*
00E4C0   F438D217 0130D058 18EC980C F6E85AE0   F7209180 F69947E0 F45E9140 F6984710   *4.K.........6Y...7...6...4.. 6...*
00E4E0   F00C9620 D00B920F 00738200 007058D0   F71C9200 007307FE 91809006 4710F478   *0.............7............4.*
00E500   41BB0002 47F0F3F0 18468840 00084839   00048930 00108830 00181734 4770F470   *.....030.....................4.*
00E520   12AA4780 F3FE4130 D028D501 30009004   4780F470 193E47B0 F4B24133 000247F0   *....3.....N.......4.....4.....0*
00E540   F49AD201 E0009004 41EE0002 06A012AA   4780F3FE 47F0F470 90ACF6D0 95083000   *4.K.............3..04...6.....*
00E560   4780F4E4 95073000 4780F4E4 91060045   4780F5B0 D200F694 F69AD200 F694F69A   *..4U......4U.....5.K.6.6.K.6.6.*
00E580   5070F690 5880F728 54830000 4780F580   9604F694 9612F696 58330000 4180F690   *..6...7.......5...6...6.......6.*
00E5A0   05A39110 F6994780 F5524130 F6705880   F72891FF 30004710 F5529508 30004780   *....6...5...6...7.......5.......*
00E5C0   F53E9507 30004780 F53E47F0 F54A5483   00004780 F54A47F0 F5084133 000447F0   *5........5..05.......5..05......0*
00E5E0   F5229106 00454780 F5B091C0 F6974710   F5789101 F69647E0 F5701277 4770F5A0   *5.....5...6...5...6...5.....5.*
00E600   9140F69A 4710F58E 9680F699 47F0F59C   12DD4780 F5789610 D00B47F0 F578D703   *. 6...5...6..05.....5.....05.P.*
00E620   F704F704 9640F699 9602F698 9610F698   9180F69A 4780F5B0 94FBF696 94FDF694   *7.7.. 6...6...6...6...5...6...6.*
00E640   9140F698 4710F61A D203D048 F69458A0   004841AA 00009140 F69A47E0 F5F29108   *. 6...6.K...6..........  6...52..*
00E660   00454710 F5DE9104 F69647E0 F61A58A0   004041AA 000012AA 4780F61A 41B00008   *....5...6...6...  ........6.....*
00E680   1BAB59A0 90A44720 F61A42A0 F6CC9107   F6CC4750 F61AD207 D038A000 9180F69A   *.........6...6...6...6.K.......6.*
00E6A0   4780F61A D702D041 D0419110 F6984710   F6529120 70064780 F63E48A7 000454A0   *..6.P.........6...6.......6.....*
00E6C0   F73819A6 4770F63E D2075000 F6909108   F6964710 F6529180 F6994710 F6529200   *7.....6.K...6...6...6...6...6...*
00E6E0   004498AC F6D007FA 07000700 07004000   00012345 014500C0 000000BC 00025B48   *....6.........  .....................*
00E700   010343E8 030343E8 030343E8 00000000   00000000 00000000 00000000 FF000000   *...Y...Y...Y.....................*
00E720   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
00E740   00000000 00000000 00000000 00000000   00000000 00000000 00000000 FF000000   *................................*
00E760   00000000 00000000 00000000 00000000   3A3F0000 C5C10000 00000000 00000000   *.....................EA.........*
00E780   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
00E7A0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
00E7C0   00C90000 00017AE0 00003FFF 00000000   05F091F0 F3F34700 F370910F F3F34700   *.I.............0.033..3...33..*
00E7E0   F37096F0 F3F3D203 0010F3E2 58300010   58403000 58440004 5880F3C2 91F08001   *3..033K...3S.....  .....3B.0...*
00E800   4770F0B4 18045880 F3CA91F0 80004770   F0985850 40001945 4700F370 4770F058   *..0.....3...0....0.. .....3...0.*
00E820   D2074028 002847F0 F05ED207 50100028   5810F3D6 D23F4030 10004110 000F5410   *K.....00.K.....30K. .........*
00E840   00284111 00C058E0 30885610 F3DE8910   000C4120 0FFF8920 00045420 00288820   *.............3.................*
00E860   00041612 41F0F0D6 07FF9500 00234780   F0B4950F 00234780 F0B44110 00F258E0   *.....000.....0.....0.....2...*
00E880   F3CE47F0 F078940F F3F358F0 F3E607FF   05F04140 40001804 5610F31E 8910000C   *3..00...33.03W..0.  .....3.....*
00E8A0   47F0F016 0000EB54 05F047F0 F0160000   0000F12C 00006EB0 0000EBBC 0000EBC2   *.00.....0.00.....1.........B*
00E8C0   940FF31B 4700F298 960FF31B 900FF2AA   D2030010 F30A5830 00105820 30C89180   *..3...2...3...2.K...3........H..*
00E8E0   20984780 F04A5840 2098982E 400094F0   F31B07FE 94BF2098 58403000 58440004   *....0.........03.....  .....*
00E900   58A030A0 18204122 000019A2 4780F078   58AA0074 41AA0000 12AA4770 F06047F0   *.............0.............0..0*
00E920   F2209180 A0214710 F2209180 A0A04710   F0909120 A01F4710 F2205890 A07C199A   *2.....2.....0.....2.....*
00E940   4780F0CC 9140A021 4710F220 47F0F110   9180A01D 4710F0DA 9401A020 940CA021   *...0.....2..01.....0.....*
00E960   9200A095 D40200AD F2A64780 F1109601   A02147F0 F1105880 A0841984 4770F09A   *....M...2...1.....01.........0.*
00E980   47F0F0A6 41200001 8920001F 14214770   F220D701 A020A020 D40200AD F2A64780   *.00.........2.P.....M...2...*
00E9A0   F0FC9601 A0219200 A0B4947F A010947F   A01D5850 A00047F0 F1849180 A01D4710   *0.....01.....*
00E9C0   F124950F A0B44780 F1245010 A0105850   A0005880 F2F291F0 80004770 F15C9180   *1.....1.....22.0....1...*
00E9E0   A01D4710 F1849120 A01F4710 F184D203   50045014 91C0500A 4740F158 D2005008   *....1.....1.K.........  1.K...*
```

```
                                                                                                                                PAGE 0027
00EA00   501C47F0 F1849180 A01D4710 F16AD203   50040024 19A44770 F1944120 30D45020   *...01.......1.K.........1.....M..*
00EA20   00245810 F3029801 100047F0 F1BED503   F2E2F2F6 4770F194 D203F2E2 30884120   *....3......01.N.2S26..1.K.2S....*
00EA40   30D45020 501491D0 500A47E0 F1B495FF   501C4770 F1B49200 501847F0 F1B89400   *.M..........1.......1......01....*
00EA60   501C58B0 F2FA05EB 9620A01F 9601A01D   58B0A084 12BB4780 F22058B0 A08812BB   *....2..........................2.......*
00EA80   4780F220 18BA4570 F27847F0 F1E847F0   F2209130 B0214710 F1DC9180 B01D4710   *..2....2..01Y.02.......1.......*
00EAA0   F1DC91FF A0204770 F20891BF A0214780   F21891FF B0204770 F21891FF B0214780   *1.......2.......2.......2.......*
00EAC0   F1DC9640 B02147F0 F1DC5880 F2F291F0   80004770 F244D502 F2E3F2F7 4770F256   *1.. ...01...22.0....2.N.2T27..2.*
00EAE0   58E03088 982DF2B2 94F0F31B 07FE982E   F2B294F0 F31B58F0 F30258FF 003C07FE   *......2..03.....2..03..03........*
00EB00   982EF2B2 94F0F31B 07FE0700 45F0F276   0000EB18 00000000 C9C5C5E5 E6C4C1D9   *..2..03......02.........IEEVWDAR*
00EB20   0A0618CB 58B0C088 12BB0777 58B0C080   12BB0777 58C0C084 19CA4787 000447F0   *.....................................0*
00EB40   F28205F0 8200F004 00060000 00000D01   10000000 00000000 00000000 00000000   *2..0..0.........................*
00EB60   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
00EB80   00000000 00000000 00000000 00000000   00000000 00016F70 00016F48 00017B90   *.....................................*
00EBA0   0001714C 000170E4 000001D8 00000198   00080000 00014880 000069FE 0A0F0A03   *.......0...Q.....................*
00EBC0   0A010A0D 000091F3 200C475E 000891FF   200D475E 00089180 21144710 F1549108   *.......3.....................1...*
00EBE0   20D4471E 00089108 20EC471E 000858A0   20C041AA 000041B0 20C819AB 4770F03E   *.M.....................H....0.*
00EC00   41B020E0 50B020C0 920820C0 9648B00C   91802115 47E0F06C 947F2115 58C0B000   *.....................0........*
00EC20   45D0F462 D202B009 20499203 B01047F0   F09C45D0 F2A21B99 91902114 4710F154   *..4.K.........00...2.........1.*
00EC40   58B020C0 589B0000 D202B009 204945D0   F43A45D0 F39C47F0 F1E847F0 F09C47F0   *........K.......4...3..01Y.00..0*
00EC60   F09C45D0 F44C47FE 0008917F 200447E0   F09A41A0 20C858B0 200841BB 000015BA   *0...4..........0....H...........*
00EC80   4720F026 58A020C0 94F7A00C 47FE0008   41A020E0 15BA4720 F03C41A0 20C841B0   *..0.......7.................0....H..*
00ECA0   20E047F0 F04041B0 20C89180 21154710   F08A9108 A00C47E0 F07A94F7 A00C9108   *....00 ...H......0.......0..7....*
00ECC0   200D47EE 00089120 2114471E 000858CA   00009620 C0081BCC 50C0A008 921DA008   *.....................................*
00ECE0   47FE0008 96802115 50A020C0 94F7B00C   45D0F3A8 58C0A000 45D0F3BE 9203A010   *.............7....3.......3......*
00ED00   47F0F054 58A020C0 58BA0000 96882114   9120B008 071E9608 211405A0 41B0205C   *.00.....................................*
00ED20   41440000 19B4077E 58B020C0 91082114   4710A072 419B0000 41B020C8 19B94770   *.............................H.....*
00ED40   A02A41B0 20E050B0 20C0589B 00009208   20C09508 70134770 A05A45D0 A24647F0   *.....................................0*
00ED60   A09247F0 A04E47F0 A04E58B0 20C0589B   000047F0 A07643B0 202741BB 000142B0   *...0...0.........0..............*
00ED80   202741BB 000142B0 900447F0 A076589B   000041B0 202850B0 301CD702 90059005   *..........0..............P......*
00EDA0   922020CC 96802114 1B9947FE 00089648   211407FE 58C00040 41CC0000 58D02130   *.....................................*
00EDC0   19CD4780 F01E41DD 000819CD 4780F01E   07FE45D0 F0B41B99 91902114 47E0F036   *....0.........0.....0.........0..*
00EDE0   94F720EC 94F720D4 07FE9108 20D44190   21044710 F0464190 20F845D0 F24C41A0   *.7...7.M.....M.....0....8..2...*
00EE00   20E041C0 20D89108 20D44710 F06A94DF   211441A0 20C841C0 20F094F7 20EC94F7   *....Q...M..0.......H...0.7...7*
00EE20   20D445D0 F1AE47FE 000047F0 F08847F0   F08247F0 F0949680 211507FE 41B020B0   *.M..1.....00..00..00............*
00EE40   50BC0000 47F0F098 50AC0000 50A020C0   9648A00C D202A009 20499203 A0109208   *....00.............K...........*
00EE60   C00045D0 F25E07FE 059098AB 204841AA   040019AB 50A02048 074D9101 21144710   *....2.....................H.....*
00EE80   901E9690 211407FD 58C02118 9180C004   47109018 41CC0008 9140C004 47109022   *.....................................*
00EEA0   50C02118 98ABC000 90AB2048 1B9907FD   058058A0 30A058A0 A08447F0 80389180   *.....................0.....*
00EEC0   30484710 804894DF A0AED402 A0ADA0AD   47708034 94FEA021 58B030C8 58BB0028   *.....M.....................H....*
00EEE0   186E05EB 18E658A0 A07419A4 47808034   12AA4780 805447F0 800C9620 A0AE9601   *.....W.....................0.....*
00EF00   A02147F0 80349180 30484710 80809110   401D47E0 807294EF 401DAF01 0A034701   *...0.....................0.....*
00EF20   0A0307FD 58103034 58110008 92001001   07FD91C0 30EC47E0 80A29108 30EC07ED   *.....................................*
00EF40   9110401D 071DAF01 0A024701 0A029610   401D07FD 58F03034 58FF0008 92F0F001   *.....................0.....00..*
00EF60   07FD5080 20500580 D5002027 211C47B0   8030D203 90002023 1BBB43B0 202741BB   *.......N.........K.............*
00EF80   000142B0 202741BB 000142B0 90045880   20501B99 47FD000C D5002026 211D47B0   *.....................N.....N.....*
00EFA0   806643B0 202641BB 000142B0 900342B0   2026D503 2023302A 47B08090 92002027   *.................N.....N.....*
00EFC0   92019004 58802050 1B9947FD 000443B0   202441BB 000142B0 900142B0 20249400   *.....................................*
00EFE0   90039400 2026D503 2023302A 47B08090   41DD0004 47F08054 58802050 1B9907FD   *......N.............0.....*
00F000   1BCC09CA 88C00003 50A09008 42C09008   07FD58B0 213058C0 213450B0 213450C0   *.....................................*
00F020   21309208 20C007FD D2032023 C0004390   C0040690 42902027 1B9907FD 00000000   *........K.....................*
00F040   50B0A13C 45A0A0EA D207C000 00209620   C00241A0 A0C247F0 A00A50B0 A12245A0   *........K.............B.0.....*
00F060   A0D0D207 C0000038 9650C002 D207C008   004047F0 A10690AB 910245A0 90B458A0   *..K.........K.....0.............*
00F080   9102506C 000042AC 00009430 C000D203   C0040048 D207C008 0040D203 C018100C   *..............K....K.... K.....*
00F0A0   47F090F2 50B0A0D8 45A0A086 58F0A0E0   90F1C008 D207C000 00189610 C0029180   *.0.2...Q.....0...1..K...........*
00F0C0   001B4780 A0BC58E0 A0E858EE 000CD203   C018E000 47F0A0BC 50B0A0A4 45A0A052   *.........Y....K......0.........*
```

```
00F0E0   58D0A0A0 D207C000 D01096D0 C00250EC   001858DE 006C98EF E03090DF C00847F0   *....K.....................0*
00F100   A08850B0 A07A45A0 A028D207 C0000028   9630C002 58F0A082 90F1C008 58E0A086   *..........K.........0...1.....*
00F120   D203C018 E00447F0 A05E05B0 4700B044   90CFB054 58F00010 91C0F0EC 4710B03C   *K.....0.............0....0.....*
00F140   58C0B06C 41CC0020 55C0B074 47A0B02C   50C0B06C 47F0B034 58C0B070 47F0B024   *.....................0......0..*
00F160   D203C01C 005007FA 05A098BF A01207FB   98ABB04C 07FB0000 00000040 400171CA   *K.........................  ...*
00F180   0003BC68 00068D4C 0003D340 00000000   00017BD4 000203E8 00015D00 0000F1E0   *..........L .....M...Y......1.*
00F1A0   00016EE0 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
00F1C0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*

NUCLEUS CONT.

016EE0   90EF0218 05E058F0 ECF2900D F03045B0   E0C6D207 F0680218 58F0F000 D207F010   *.......0.2..0....FK.0....00.K.0.*
016F00   0018AF0D 0FFE470D 0FFE58A0 E05E05BA   9140001B 4780E038 58A0E056 052A05E0   *..............  ..............*
016F20   9180001B 4780E00E 58A0E020 052A05E0   58600050 5050E4E8 47F0E25C 00028BB0   *....................UY.0S.....*
016F40   0001E198 0000F0A4 940FA029 4700A00C   47F0A244 58E0AC90 58D0E000 D2070038   *.....0.........0.........K....*
016F60   D010980F E0308200 003890EF 021805F0   4700F024 96F0F001 58E0FC68 900DE030   *..............0..0..00........*
016F80   45B0F03C D207E068 021858D0 E000D207   D0100038 58A0F038 05BA9856 F03007F6   *..0.K.........K.......0.....0..6*
016FA0   00001000 000060E0 0000F05A 05A05840   AC2A5850 A02A1945 077B5860 00509878   *..........0.... ..............*
016FC0   A46A1277 4780A026 1B761A87 5080A46E   5060A45A 07FB0000 00017A60 0A03900F   *.....................Q.0.......*
016FE0   01D805B0 58F0B0B8 05EF58F0 B0BC051F   470D0FFD 9801101D8 58F00214 58A0B0B4   *.Q...0.....0.........Q.0.......*
017000   05BA98CD 00289101 00294780 B05858A0   BBD658A0 A00412AA 47C0B052 181A58A0   *..............0...............*
017020   A00012AA 47C0B052 92801000 90C21004   D2031018 01DC98F0 A0008900 C0001200   *............B..K......0........*
017040   47A0B052 50F0002C 58E0B092 98AD0200   82000028 58E0B08E 07FE9500 002B4780   *.....0.........................*
017060   B05241DD 000055D0 B08A4780 B07450D0   B08A47F0 B0525890 B08696F0 90011B99   *..................0.......0....*
017080   5090B08A 47F0B052 0000F12C 00000000   0000E7D0 00016FDC 0000F102 00028BF8   *.....0....1.......X.......1....8*
0170A0   00028D1C 50B00220 05B01211 4740B022   48A0018C D2011000 A0004010 A0004010   *.............  .....K..........*
0170C0   018C92F0 B0E558B0 022007FE 131158A0   0194D203 1000A000 5010A000 50100194   *...0.V..........K.............*
0170E0   47F0B018 50C00220 05C058B0 CAEA12BB   4770C00E 58B0CAEE D500B023 A0234720   *.0.....................N.......*
017100   C05C4740 C0429101 A01E4770 C05C58B0   B07412BB 4780C042 D500B023 A0234770   *...  .....................N.......*
017120   C04219BA 4770C024 47F0C05C 48B0A020   12BB4770 C05C58B0 A0009500 B01C4770   *.........0.....................*
017140   C05C50A0 CAEA58C0 022007FE 05E09200   EA425820 EA8A5920 EA864770 E018982E   *.....................  ........*
017160   01A08200 00205850 20009834 00209034   5010983E 01A4900F 2030D203 203801A0   *.....................K.....*
017180   05E047F0 E00A0000 00017418 05E04700   E2FA98AB EA4615AB 4770E154 9101B01E   *...0.............S.............*
0171A0   4780E022 5860E9CE 91806000 4710E076   58C0A000 D2070180 C010AF04 0FE74704   *.....Z.............K........X..*
0171C0   0FE758A0 EA0618EB 05BA980F E0308200   01805860 E9CE9180 60004710 E07658C0   *.X.................Z...........*
0171E0   B00091FF C01C4770 E06648C0 B02012CC   4780E06C 1BAA47F0 E16E18AB 90ABEA46   *.....................0.........*
017200   47F0E022 D703EA46 EA4694F7 60001BAA   47F0E188 5810E9CE 50A0EA46 58F0E9BE   *.0..P......7....0....Z.....0Z.*
017220   052F58A0 29B258E0 293247F0 E1F018DA   58700010 587070D8 D5007000 D0234780   *............0.0.........QN.....*
017240   E0BC4177 001047F0 E0AA58A0 700812AA   4780E262 58C0A000 91FFC01C 4770E122   *.....0.........S.............*
017260   48C0A020 12CC4770 E1225860 E9CE9180   60004770 E10650A0 EA465810 E9CE58F0   *.............Z..............Z..0*
017280   E9BA1896 1807052F 58E028CC 18691870   58A0EA46 9200600C D202600D 700D55A0   *Z.....................K.......*
0172A0   70044770 E13ED202 70097001 47F0E1E2   55A07004 4730E148 58AA0074 55A07008   *.....K.....0.S................*
0172C0   4770E0C6 58D07004 47F0E262 D2027009   A07547F0 E1E258A0 700041AA 000047F0   *...F.....0S.K.....0.S.........0*
0172E0   E12E9101 B01E4780 E18812AA 4780E16E   D500B023 A0234780 E04418DA 5810E9CE   *...............N.............Z.*
017300   50A0EA46 58F0E9BA 1807052F 58E02848   187058A0 EA4658D0 E9CA1ADB 6000D000   *.....0Z.....................Z.....*
017320   6020D008 6040D010 6060D018 5810B078   18071211 4780E1C4 918B1000 4750E1C4   *.....  .....................D.....D*
017340   50A0EA46 58F0E9BA 052F58E0 280A58A0   EA4658F0 E9BA188B 45C0E292 58A0EA46   *.....0Z.........0Z.....S......*
017360   187012AA 4780E23C 9101A01E 4710E0A0   18BA90AB EA469101 A01E4710 E0865810   *......S.......................*
017380   A0781211 4780E20C 50A0EA46 58F0E9BE   052F58E0 27C258A0 EA461807 50A0EA46   *.....S.....0Z.....B...........*
0173A0   188A58F0 E9BE45C0 E29258A0 EA461870   58D0E9CA 1ADA6800 D0006820 D0086840   *...0Z...S.........Z.............*
0173C0   D0106860 D01847F0 E02218DB 58C0D000   9101D01E 4710E0A2 91FFC01C 4770E262   *.......0.....................S.*
0173E0   48C0D020 12CC4770 E26218AD 47F0E1E2   58D0D074 12DD4770 E23E41C0 E8D218BC   *.........S....0.S.........S...YK..*
017400   90BCEA46 58A00050 50A0E28A 58C0E286   07FC0000 0001B666 10817959 023421A8   *...........S...S..............*
017420   59808000 078C58D0 808812DD 4770E2A4   18D858D0 D07C58D0 D08412DD 078C91F0   *..............S..Q.............0*
```

```
017B40  00016801 68000000 0001E130 0001E4B8  000188B8 0001718E FFFFFFE0 0001A858  *.............U.................*
017B60  00017C00 00018380 000170E4 000000FF  00020148 000001B8 00000324 00020304  *...........U...................*
017B80  0001714C 0001B4C0 0000E8A8 0000E890  00000000 0000F0D8 0000F040 0001A3F0  *.........Y...Y.....0Q..0 ..,0*
017BA0  0001793E 00021410 00017B24 0001A7C8  00026428 0001A3A8 0001718C 0001791C  *.............,H...............*
017BC0  00025C40 80806000 80B06000 FF000000  0A030000 0003D340 0003D340 00000000  *... ..................L ..L ....*
017BE0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
017C00  00017CB8 00000000 00000000 00000000  00000000 00000000 01000000 00000080  *................................*
017C20  0000FFFF 00000000 00000000 00000000  00000000 00000000 00018998 0001A880  *................................*
017C40  00017C00 00017CB8 00000000 000189F8  00000000 00026428 00018990 0003D340  *..............8.............L *
017C60  0001B4C0 0003BC68 00000000 00000000  00000000 00017D40 00000000 00017C00  *.............................. ..*
017C80  00000000 000188B8 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
017CA0  00000000 00000000 00000000 00000000  00000000 00000000 00000308 F6F0F5C1  *.............................605A*
017CC0  000DC080 00000000 00040000 0001BB7C  00000000 01017C00 00000000 00000000  *................................*
017CE0  00018998 0001A880 00017C00 00017CB8  00000000 000189F8 00000000 00026428  *.....................8.........*
017D00  00018990 0003D340 0001B4C0 0003BC68  00000000 00000000 C9C7C3F0 F6F0F5C1  *......L ..................IGC0605A*
017D20  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
017D40  00017DF8 00000000 00000000 00000000  00000000 00000000 01000000 00000080  *...8............................*
017D60  0000FFFF 00000000 00000000 00000000  00000000 00000000 000189A8 0001A880  *................................*
017D80  00017D40 00017DF8 00000000 00018E40  00000000 00026428 00018990 0003D340  *.......8.......  ..........L *
017DA0  0001B4C0 0003D478 00000000 00000000  00000000 00017E80 00000000 00017D40  *......M.......................  *
017DC0  00000000 000188B8 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
017DE0  00000000 00000000 00000000 00000000  00000000 00000000 00000220 F9F0F1C3  *.............................901C*
017E00  000DC080 00000000 00040000 0001BB7C  00000000 01017D40 00000000 00000000  *.............................. ..*
017E20  000189A8 0001A880 00017D40 00017DF8  00000000 00018E40 00000000 00026428  *.............8.............8.....*
017E40  00018990 0003D340 0001B4C0 0003D478  00000000 00000000 C9C7C3F0 F9F0F1C3  *......L .......M.........IGC0901C*
017E60  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
017E80  00017F38 00000000 00000000 00000000  00000000 00000000 01000000 00000080  *................................*
017EA0  0000FFFF 00000000 00000000 00000000  00000000 00000000 000189B8 0001A880  *................................*
017EC0  00017E80 00017F38 00000000 00019288  00000000 00026428 00018990 0003D340  *.............................. L *
017EE0  0001B4C0 0003BC68 00000000 00000000  00000000 00017FC0 00000000 00017E80  *................................*
017F00  00000000 000188B8 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
017F20  00000000 00000000 00000000 00000000  00000000 00000000 00000220 D8F0F5C1  *.............................Q05A*
017F40  000DC080 00000000 00040000 0001BB7C  00000000 01017E80 00000000 00000000  *................................*
017F60  000189B8 0001A880 00017E80 00017F38  00000000 00019288 00000000 00026428  *.............................. *
017F80  00018990 0003D340 0001B4C0 0003BC68  00000000 00000000 C9C7C3F0 D8F0F5C1  *......L ..................IGC0Q05A*
017FA0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
017FC0  00018078 00000000 00000000 00000000  00000000 00000000 01000000 00000080  *................................*
017FE0  0000FFFF 00000000 00000000 00000000  00000000 00000000 000189C8 0001A880  *.............................H....*
018000  00017FC0 00018078 00000000 000196D0  00000000 00026428 00018990 0003D340  *.............................. L *
018020  0001B4C0 0003BC68 00000000 00000000  00000000 00018100 00000000 00017FC0  *................................*
018040  00000000 000188B8 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
018060  00000000 00000000 00000000 00000000  00000000 00000000 000003C8 F3F0F5C1  *.............................H305A*
018080  000DC080 00000000 00040000 0001BB7C  00000000 01017FC0 00000000 00000000  *................................*
0180A0  000189C8 0001A880 00017FC0 00018078  00000000 000196D0 00000000 00026428  *...H.............................*
0180C0  00018990 0003D340 0001B4C0 0003BC68  00000000 00000000 C9C7C3F0 F3F0F5C1  *......L ..................IGC0305A*
0180E0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
018100  000181B8 00000000 00000000 00000000  00000000 00000000 01000000 00000080  *................................*
018120  0000FFFF 00000000 00000000 00000000  00000000 00000000 000189D8 0001A880  *.............................Q....*
018140  00018100 000181B8 00000000 00019B18  00000000 00026428 00018990 0003D340  *.............................. L *
018160  0001B4C0 0003BC68 00000000 00000000  00000000 00018240 00000000 00018100  *.............................. ..*
018180  00000000 000188B8 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
0181A0  00000000 00000000 00000000 00000000  00000000 00000000 00000358 F4F0F5C1  *.............................405A*
0181C0  000DC080 00000000 00040000 0001BB7C  00000000 01018100 00000000 00000000  *................................*
0181E0  000189D8 0001A880 00018100 000181B8  00000000 00019B18 00000000 00026428  *...Q.............................*
018200  00018990 0003D340 0001B4C0 0003BC68  00000000 00000000 C9C7C3F0 F4F0F5C1  *......L ..................IGC0405A*
```

```
018220   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
018240   000182F8 00000000 00000000 00000000   00000000 00000000 01000000 00000080   *...8............................*
018260   0000FFFF 00000000 00000000 00000000   00000000 00000000 000189E8 0001A880   *...........................Y....*
018280   00018240 000182F8 00000000 00019F60   00000000 00026428 00018990 0003D340   *... ...8......................L *
0182A0   0001B4C0 0003BC68 00000000 00000000   00000000 00018380 00000000 00018240   *............................... *
0182C0   00000000 000188B8 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
0182E0   00000000 00000000 00000000 00000000   00000000 00000000 00000378 F5F0F5C1   *............................505A*
018300   000DC080 00000000 00040000 0001BB7C   00000000 01018240 00000000 00000000   *....................... ........*
018320   000189E8 0001A880 00018240 000182F8   00000000 00019F60 00000000 00026428   *...Y....... ...8................*
018340   00018990 0003D340 0001B4C0 0003BC68   00000000 00000000 C9C7C3F0 F5F0F5C1   *......L ................IGC0505A*
018360   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
018380   00018438 00000000 00000000 00000000   00000000 00000000 01000000 00000080   *................................*
0183A0   0000FFFF 00000000 00000000 00000000   00000001 0001A3E8 0001A3C8 0001A4D5   *.......................Y...H...N*
0183C0   00000000 00000012 0001791C 0001A3A8   4001799E 4002644E 00000000 00000000   *................ ... ...........*
0183E0   00021A12 00033F74 40026448 6001DAEA   00000000 00018478 00000000 00018380   *........ .......................*
018400   00000000 000188B8 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
018420   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
018440   00040080 00000000 00040000 00000000   00000000 01018380 00000000 00000000   *................................*
018460   00000000 00000000 00000000 00000000   00000000 00000000 00018530 00000000   *................................*
018480   00000000 00000000 00000000 00000000   01000000 00000080 0000FFFF 00000000   *................................*
0184A0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
0184C0   00000000 00000000 00000000 00000000   00018990 00000000 00000000 00000000   *................................*
0184E0   00000000 00000000 00000000 000186C8   00000000 00018478 00000000 000188B8   *...............H................*
018500   00000000 00000000 00000000 00000000   00034490 00000000 00000000 00000000   *................................*
018520   00000000 00000000 00000000 00000000   00000000 00000000 00040080 00000000   *................................*
018540   00040000 00000000 00000000 01018478   00000000 00000000 002B4008 00021A38   *.......................... .....*
018560   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
     LINE 018580 SAME AS ABOVE
0185A0   00000000 00000000 00000000 00000000   000185B8 00000000 000185D0 00000000   *................................*
0185C0   00018550 00018478 00000000 00000000   000185E8 00000000 00018550 00018478   *...................Y............*
0185E0   00000000 00000000 00018600 00000000   00018550 00018478 00000000 00000000   *................................*
018600   00018618 00000000 00018550 00018478   00000000 00000000 00018630 00000000   *................................*
018620   00018550 00018478 00000000 00000000   00018648 00000000 00018550 00018478   *................................*
018640   00000000 00000000 00018660 00000000   00018550 00018478 00000000 00000000   *................................*
018660   00018678 00000000 00018550 00018478   00000000 00000000 00018690 00000000   *................................*
018680   00018550 00018478 00000000 00000000   00000000 00000000 00018550 00018478   *................................*
0186A0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
0186C0   00000000 00000000 00018780 00000000   0003DBC4 00000000 00000000 00000000   *...................D............*
0186E0   0103FE48 00000080 0000FFFF 00000000   00000000 00000000 00000001 FFFDBB00   *................................*
018700   00024428 00024628 00024380 000243CC   00024554 80024AA8 000244F8 4001FF92   *...........................8 ...*
018720   0002AE28 00024AF0 00000048 00024AA8   400200BC 6001DAEA 00000000 000187C0   *.......0........ ...............*
018740   00000000 000186C8 00000000 000188B8   00000000 00000000 00000000 00000000   *.......H........................*
018760   00034490 0003DB98 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
018780   00000000 00000000 00040081 00000000   FF040001 5001FFE2 00000000 010186C8   *.......................S.......H*
0187A0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
0187C0   00018878 00000000 00025C0C 00000000   00000000 00000000 01000000 00000080   *................................*
0187E0   0000FFFF 00000000 00000000 00000000   00000000 00000000 00000000 0001A880   *................................*
018800   000187C0 00018878 00000000 00000000   00000000 00000000 00018990 00025B48   *................................*
018820   000204F0 00000000 000204F0 00025C40   00000000 000188B8 00000000 000187C0   *...0.......0... ................*
018840   00000000 000188B8 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
018860   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
018880   00040080 00000000 00040000 000204FA   00000000 010187C0 00000000 00000000   *................................*
0188A0   00000000 00000000 00000000 00000000   00000000 00000000 00040650 00000000   *................................*
0188C0   0003E434 0003E200 00000000 00000000   0103FE98 00000080 0000FFFF 0003EC78   *..U...S.........................*
0188E0   00000000 80040690 00000001 00027160   00000004 0006F3B0 00000000 0003D4D0   *......................3.......M.*
```

```
018FE0   5810A004 50A0F000 0A0718A7 5870A088   12774770 917E5870 A08019A8 47802004   *.....0.....*
019000   58A0A084 47F09166 18A707F2 92621002   92801003 50A01008 50B01004 95D7C007   *.....0.....2.....P..*
019020   4770919E 94E71002 0A3307F2 00200000   00000004 C9C7C3F0 C1F0F5C1 C9C7C3F0   *.....X.....2.....IGC0A05AIGC0*
019040   C2F0F1C3 F9F0F1C3 FD000000 FE000000   FF000000 FF030000 E2E8E2C9 C5C1F0F1   *B01C901C.............SYSIEA01*
019060   C9C5C1C1 C2C5D5C4 F9000000 00000000   00000000 00000000 00000000 00000000   *IEAABEND9.....................*
019080   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....*
0190A0   00000000 0000B0A8 000495F2 202C4770   322C5822 008C18B2 41202004 41A00004   *.........2.....2.....*
0190C0   41000001 D205F010 F00041F0 F01047F0   30A841A0 00091B00 47F0323A 5810B01C   *....K.0.0..00..0.....0.....*
0190E0   D2011015 B0245827 00009180 202A4780   32709200 800047F0 3298910E 70004780   *K.....0.....*
019100   32929102 201A4710 32929102 20344780   3292D204 800033E6 47F03298 D2048000   *.............K...W.0..K...*
019120   33EC4180 80084170 7004D501 800032E8   0783D501 800032E6 47703298 18754180   *.........N....Y..N....W.....*
019140   60209500 80004770 32CC4180 80084170   700447F0 32B8D201 60068000 D202600E   *.............0..K.....K...*
019160   800241F0 41C85060 F0009280 F0000A07   F0E2F1D5 00000000 00000000 00000000   *...0.H..0...0..0S1N.........*
019180   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....*
         LINES 0191A0-019240 SAME AS ABOVE
019260   00000000 00000000 00000000 00000000   F1E8009E 0830F1C2 008E1580 00000000   *.............1Y....1B.....*
019280   00000000 F0F1F97D 02000000 7F0192C8   000192C8 0C000000 400192B0 0001A7C8   *....019.....H...H.... .....H*
0192A0   00000000 00000000 00000000 0F000F0C   310192AB 60000005 080192B0 40000001   *.....*
0192C0   060192D0 00000220 7F000000 00000000   05C01B56 41560001 18768000 C1AD9102   *.....A...*
0192E0   40144770 C08458A0 400847F0 C02058A0   A0044670 C01C41AA 000012AA 4780C084   * .....0.....*
019300   180A4B00 C1F01820 41220014 1B114312   000C8910 000358E0 A01C41EE 00008000   *....A0.....*
019320   C1AC1890 188E1871 D203D098 C2014550   C19C92F0 D0981809 18E81817 12114780   *A.....K...B...A..0.....Y.....*
019340   C002190E 4780C07C 41090014 18174550   C19047F0 C0028000 C1AC5860 400CD20A   *.............A..0...A....K.*
019360   D098C205 D207D0A5 6000D203 D0B0C210   D207D0B5 60089540 60104780 C0B8D203   *..B.K.....K...B.K..... .....K.*
019380   D0C7C214 D207D0CD 60104550 C19C41B0   D1039500 60184780 C110D201 D09EC218   *.GB.K.....A...J.....A.K...B.*
0193A0   1BAA43A6 001842A0 D06888A0 00024126   00184130 C1A84550 C178D207 D0B82004   *.....A...A.K.....*
0193C0   4122000C 4180D0C4 41000003 1BA047C0   C1104570 C1401B99 4390D068 1A6947F0   *.....D.....A...A .....0*
0193E0   C0C041A0 C11E41F0 D06850A0 F0000A07   C9C7C3F0 F3F0F5C1 D203D069 2000F384   *....A..0....0..IGC0305AK.....3.*
019400   D069D069 DC07D069 C0BED207 8000D069   07F54550 C12646A0 C14E4550 C19C07F7   *.........K.....5..A...A...A..7*
019420   41220004 4188000C 198B4740 C1404550   C19C9110 D1124710 C1704180 D0C747F0   *.......... A ..A...J...A....G.0*
019440   C1404180 D0AC47F0 C1405060 D1245860   D12847F0 60005060 D1245860 D12847F0   *A ....0A ..J...J..0....J...J..0*
019460   60105060 D1245860 D12847F0 60825060   D1245860 D12847F0 62DA1203 0040FF00   *....J...J..0....J...J..0..... ..*
019480   F0F1F2F3 F4F5F6F7 F8F9C1C2 C3C4C5C6   00000000 00000000 00000000 00000000   *0123456789ABCDEF............*
0194A0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....*
0194C0   00000024 D3D54040 40404040 40404040   C1C4D960 C4C5C260 E3C9D6E3 404040D1   *....LN          ADR.DEB.TIOT   J*
0194E0   D6C2E2E3 C5D7D7D9 D6C3C4C4 C030B0A8   C23258A0 705450A0 704850A0 7050D202   *OBSTEPPROCDD....B.............K.*
019500   704D30A5 91043074 47E0C258 58B03034   58BB0018 45DB0004 12FF4770 C25858B0   *.........B.............B...*
019520   70485AB0 705450B0 70489102 711441A0   C39E4710 C3869200 704C4110 70F841A0   *.............C...C.....8...*
019540   71045010 70C850A0 70E058A0 20081BBB   416A0400 596030A4 4740C296 8CA0000A   *.....H.....B.....*
019560   89A0000A 88B00016 50A070D0 40B07102   D7017100 71009680 7100921D 70C89280   *.....P.....H...*
019580   70CC920C 70CF921D 70D09248 70D44110   04004010 70FE9204 70FDD202 710970FD   *.....M.....K.....*
0195A0   401070D6 920370D8 922070DC 921D70E0   41B070D8 41A070F0 50B07130 50A07134   * ..0...0.....0...0.....*
0195C0   41100002 0A0B5860 20089001 60089620   711445D0 C35A41A0 705C19A9 4780C330   *.....C.....C.*
0195E0   58A070C0 58AA0000 D2049008 A000D500   70277110 41A00400 4780C32C 1BAA40A0   *.....K.....N.....C.....*
019600   901241F0 001C9120 71144710 C09C41F0   00189104 71144710 C09C1BFF 91107114   *...0.....0.....*
019620   4710C09C 41F00020 47F0C09C 92427000   D7067009 70099200 70901817 0A004110   *.....0...0.....P.....*
019640   70904110 10004100 00010A01 8000C37B   07FDC084 41A0C396 50A070C8 41F070C8   *.............C.....C...H.0.H*
019660   0A07F008 406040E2 C9C7C3F0 D3F0F5C1   C9C7C3F0 E9F0F5C1 C9C5C5F5 F1F0F3C4   *..0.. SIGC0L05AIGC0Z05AIEE5103D*
019680   58B03034 58BB000C 07FB0000 00FFF800   1613C2C2 00A20125 F7C70000 0000F7C8   *.............8...BB....7G....7H*
0196A0   00000000 F6D4008A 01800000 00000000   00000000 00000000 00000000 00000000   *....6M.....*
0196C0   00000000 00000000 00000000 F0F1F96A   02000000 7F019710 00019710 0C000000   *.............019.....*
0196E0   400196F8 0001A7C8 00000000 00000000   00000000 0F000801 310196F3 60000005   * ..8...H.....3....*
019700   080196F8 40000001 06019718 000003C8   7F000000 00000000 05C09180 40214710   *...8 .....H.....  ...*
019720   C1C4D203 D098C362 925CD0A8 D25DD0A9   D0A8D205 D0B4C366 D204D0C6 C36CD204   *ADK...C.....K.....K...C.K..FC.K.*
019740   D0DAC371 D204D0EE C36CD204 D0FAC376   4550C2A8 D21BD0A8 C37BD21E D0CDC397   *..C.K...C.K...C...B.K...C.K...C.*
```

```
01A520  0A034397 000A8990 000358B4 008C4159   B0001B99 47F08076 41D004E7 58E4002C   *.....0.....X.U..*
01A540  07FE41D0 0CB558E0 402C07FE 48000000   00FFFFFF FF201D4E C18892C1 C3929233   *.....  .....A..AC...*
01A560  C361D203 C389C3A8 9103C3A7 4780C188   92C2C392 89800002 91013001 48703004   *C.K.C.C...C...A..BC.............*
01A580  4780C1B2 43705001 5470C30C 19874770   C1E44870 500091F0 50014750 C1B24370   *..A.....C.....AU.....0....A...*
01A5A0  30058870 00044270 C37BF321 C37BC37B   DC01C37B C2A79240 C37D1881 4110C360   *.....C.3.C.C...C.B.. C.....C.*
01A5C0  96202000 911040B6 4710C1DE 0A231818   0A0F0A03 41505002 47F0C194 5880407C   *.....  ...A.............0A....*
01A5E0  1B664360 80185870 80189101 70004780   C05691FE 70004710 C23AD501 30147002   *.....................B.N.....*
01A600  4770C23A D400300C 700196FE 70004360   30045460 C30C5870 C3108870 60005670   *..B.M.............C...C.......*
01A620  80145070 801447F0 C13E4170 70044660   C20247F0 C0569101 300141B0 30044150   *.......0A.......B..0...........*
01A640  30064170 000A4780 C29E5850 408C1B77   4370300A 89700003 48575006 5450C314   *.........B... ...............C.*
01A660  06500650 43705001 5470C30C 91144074   4710C292 4380300C 5480C30C 197847F0   *.........C.....B.......C....0*
01A680  C04A4170 000F4180 000418B5 1BB84190   02A058A0 02B01AA9 1B004380 B0005480   *.....................C.........*
01A6A0  C30C1A88 41D89000 18F945E0 C2E041D8   A00018FA 45E0C2E0 41B0B002 19B54740   *C....Q...9..B..Q......B........ *
01A6C0  C2AA4380 300C5480 C30C1680 47F0C28C   9170D000 078E9101 30014160 00084780   *B........C....0B.............*
01A6E0  C2FA4360 B0015460 C30C95C1 F0184780   C3068850 00021606 07FE4770 0000000F   *B........C..A0...C.............*
01A700  80000000 0000FFFF 00080000 8000C9C5   C1F0F0F0 C140C3E4 C16BC9D5 E340D9C5   *.............IEA000A CUA.INT RE*
01A720  D86B6BC3 C37EF361 D5D640D7 C1E3C8E2   40C1E5C1 C9D3C1C2 D3C56B6B 6BE5D6D3   *Q..CC.3.NO PATHS AVAILABLE..,VOL*
01A740  E2C5D96B D1D6C2D5 40404040 40001000   00298000 C9C5C1F0 F0F1C940 E4D5C9E3   *SER.JOBN     .....IEA001I UNIT*
01A760  40F2F3F0 6B40D7C1 E3C840F5 F340C9D5   D6D7C5D9 C1E3C9E5 C5100010 4040C3D7   * 230. PATH 53 INOPERATIVE...  CP*
01A780  E440E710 001040F0 F1F2F3F4 F5F6F7F8   F9C1C2C3 C4C5C604 40C6D6D9 08022010   *U X... 0123456789ABCDEF. FOR....*
01A7A0  40018020 200095D2 41E0960C 58B0800C   47F096DC 41E095DA 47F095CA 58D08010   * .....K.....0.....0.....*
01A7C0  4AD0B00E 06D04870 D0045A70 80140670   9120B016 478095FE C9C7C5F0 F0F0F0C1   *.....................IGE0000A*
01A7E0  60F2F006 4710961A 40404040 00000000   00033FC8 00033F74 0003F5B8 00000000   *.20.....     .....H.....5.....*
01A800  3B000000 00000000 00000000 00001E0C   5B000000 00000000 00000000 0005DE40   *.............................. *
01A820  6B000000 00000000 00000000 00020DD7   7B000000 00000000 00000000 00012C7B   *.............P..............*
01A840  AB000000 00000000 00000000 80001E0C   00000000 00000000 80000000 00000000   *................................*
01A860  00000000 00000000 00000000 00000000   00000000 00000000 00000065 F2F1F0F0   *.........................2100*
01A880  00017BD4 000170A4 0001A7C4 000270C0   00000000 00006BB4 000179E6 000218A0   *...M.......D.............W....*
01A8A0  000217D4 000070DC 00007170 000175F8   00002928 0000E8A8 0099366F 00027158   *...M.........8.....Y.........*
01A8C0  000219BE 00027220 00034C10 00000000   0A0307FE 0001A7C8 000203E8 00021410   *.....  .....H...Y.....*
01A8E0  00000000 00024428 0001E4B8 0001E164   0000D1D0 10027280 000052CC 00006B3C   *..........U.....J.............*
01A900  00040800 0002646E 0001718C 000070A4   0000018A 00027158 0001DD0E 00000000   *................................*
01A920  00017C00 0007E7FF 00000000 00000000   000273D8 0000E700 0001D03E 0001A7F8   *.....X.............Q..X.......8*
01A940  00000000 0003C0C8 0001B424 00000000   00000000 0A0D0A06 0001A800 000273E8   *........H.....................Y*
01A960  00000000 00000000 000274B0 00000224   00000000 00000000 00000000 00000000   *................................*
01A980  00000000 00027550 00000000 00020510   00027558 00017A14 00000000 00000000   *................................*
01A9A0  00000000 00000000 05205880 29CA9200   8000D502 0025006D 478021E2 9180401D   *.............. N........S....*
01A9C0  47102076 186F4190 40A05879 00001277   47802074 9120700C 47802044 9180500B   *.....  .....  .....  .....*
01A9E0  47102056 18975877 000047F0 2024D502   4001700D 47702074 9180700C 4710206E   *.....  .....0..N. .....  .....*
01AA00  41170000 D2039000 70005800 29A65880   29C605E8 47F02020 189747F0 202018F6   *....K.....  .F.Y.0....0...6*
01AA20  91C0500A 47402210 58C02996 471020A6   58802992 D23F4030 80009180 500B4710   *.....  .....K.  .....  .....*
01AA40  25285860 501CD203 40686014 41C022F2   95125009 477020BC D5045088 29AE4770   *.....K.  .....2.....N.....*
01AA60  20BC925C 508C1865 9180600B 471020D6   5860601C 91C0600A 471020DA 47F020BE   *.............0.....0..*
01AA80  94DF4094 05EC05E0 5820E8B0 9180500B   471021D6 91404094 4780214E 9180401D   *..  .....Y.....0.  .....  .*
01AAA0  4710210A 9120401F 4710210A 918040A0   47802112 94BF4094 47F0214E 5860501C   *.....  .....  .....0.....*
01AAC0  91C0600A 4710214E 9120600B 4710213E   91F06011 4730214E 9180600B 4770213E   *.............0.....  .....*
01AAE0  5860601C 47F02116 96014021 964040AE   94BF4094 47F0215A 5860501C 91FF601C   *.....0.....  .....  ..0.....*
01AB00  4780216C 58703000 59407000 4770216C   1B885080 700094BF 500BD202 4001501D   *.....  .....  .....K.  .....*
01AB20  58C029CE 9102500B 078C91C0 500A4790   21AE9500 5018077C 58105000 41101000   *.....  .....  .....  .....*
01AB40  12114780 21A65800 29A25880 29C605E8   48005008 47F021B4 1B004300 50091815   *.....F.Y.....0.....  .....*
01AB60  89000003 5600299E 91C0500A 479021CA   5400299A 588029C6 05E858C0 29CE07FC   *.....F.Y.....  .....  .....*
01AB80  45E027EE 96804021 47F0215A 58604004   947F6000 987C6008 50705014 90894068   *.....  ..0.....  .....  .....*
01ABA0  90AC4030 58E02992 D22B403C E00CD201   50100020 58E03088 07FE910C 500B4780   *.....K.....K.....  .....  .....*
01ABC0  20E25860 50184166 00004710 223C9108   500B4780 229A5806 00009110 500B4710   *.S.....  .....  .....  .....*
01ABE0  223C47F0 22C41B11 43105018 12114780   224A0610 58006000 18705000 50184210   *...0.D.....  .....  .....  .....*
```

```
01B2E0   E00E58F0 29F218C4 47F0F004 58004084   58102A6E 58D02A6A 05ED47F0 25D60580   *...0.2.D.00... ............0.O..*
01B300   50E08034 914081BC 47808010 968081BC   91208 1BC 47808024 94DF81BC 58F0806C   *.....  ..................Q..*
01B320   47F08028 58F08070 05EF947F E19258E0   E00A07FE 8001AE58 0001A9AA 00000198   *.0...0.....................*
01B340   0001B604 FDFFFFFF FF000000 FA000048   FF000010 FD000010 C1C2C5D5 C4000000   *.....................ABEND...*
01B360   00001000 0000000D 00006286 0001C4A8   0001BEEA 00017B90 0001B666 000170E4   *..............D.............U*
01B380   0001CF64 0001D42C FF000018 0003FF80   0003FF84 80FB0000 FA000020 0001DCE2   *......M.....................S*
01B3A0   80D03000 0001E018 80C03000 0001AD72   FF000008 80A03000 FD0000D8 00000020   *.....................Q.....*
01B3C0   0001E132 F3000070 0001B004 00000000   00000000 00000000 00000000 00000000   *....3.....................*
01B3E0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....................*
01B400   00000000 00000000 00000000 FD000000   0001AE22 0000E8A8 80103000 00000002   *.....................Y.......*
01B420   00000A01 0001B002 0001E7B8 0001B0B2   00018990 0001B134 0001ECB8 0001EF94   *...........X.....................*
01B440   0001BEEA 0001F9B0 0000F12C 000170E4   0001CF64 0001C4A8 0001B05A 0001B198   *......9...1...U.....D.........*
01B460   0001AD72 0000F198 0001BE40 0001BCE0   0001A998 0001A99C 00018550 00018478   *......1.....................*
01B480   0001FF90 0000EB0A 0001BB74 000187C0   0001CE80 000186C8 00018380 00000000   *.....................H.........*
01B4A0   000058BC 0001B666 00020148 0001B0B2   00020468 000204FA 00020510 0001C514   *.....................E.*
01B4C0   4520C404 96204094 5E60C7D0 4710C022   92C0D00A 9801D020 185D98DF D05405E6   *..D.. ....G.....................W*
01B4E0   0A034815 00124E10 C7B8F332 C7B0C7BD   9801C7AC 45E0C4A4 47F0C048 50FD000C   *.........G.3.G.G..G...D..0.....*
01B500   586F0010 47F0C010 18768870 0012407D   00025460 C7CC506D 0018501D 00045890   *.....0.........G.........*
01B520   C7845889 00044129 00089140 20004710   C07A5962 00084780 C0C04122 00104680   *G.....................*
01B540   C06A5810 C82845B0 C4C85830 00105890   C7841222 4770C0D6 D202D00D 900150D9   *....H...DH.....G.....OK......R*
01B560   000092FF D01CD207 D010C7F0 900F4030   58A30000 1B11501A 000058A0 C79007FA   *......K...G0.. ..............G...*
01B580   92002000 45E0C476 58620000 1B29402D   000047F0 C01445B0 C45245B0 C48E92FF   *......D.........0....D...D...*
01B5A0   D01C45E0 C4769801 D020185D 98DFD054   41E0C020 9005 4030 90DF4064 D2045010   *....D.....................K...*
01B5C0   C7F8D202 50152001 18621B69 40650000   5872000C 58570000 D2076060 C7AC5872   *G8K.....................K...G...*
01B5E0   000C5863 00005076 00005867 00009200   601C9240 20005047 005C5057 006458A0   *.....................  .....*
01B600   C79007FA 58C0C6A8 D2334038 50289001   403050F4 006C91D0 500A07EE 5890C784   *G.....F.K.......4..........G.*
01B620   48650000 41269000 1B004162 000458D6   000012DD 4730C19C 19D54780 C188180D   *..............O......A..N..A...*
01B640   416D000C 47F0C16E D2026001 D00D5860   C8200660 5060C820 47F0C180 1200077E   *.....0A.K.......H.....H..0A.....*
01B660   92202000 07FE58C0 C6461B11 5910C820   4780C25A 5890C784 58890004 41290008   *.........F.....H...B...G.........*
01B680   91402000 4710C212 4150C578 4590C538   12FF4780 C256585F 0000D502 20095019   *.  ....B...E...E.....B.....N.....*
01B6A0   4780C212 45B0C452 58300010 5872000C   58670000 D2076060 C83050F7 005C5057   *..B...D.............K...H..7....*
01B6C0   00649200 601C58B0 C7A018A7 05EB9240   20004122 00104680 C1C01211 4780C25A   *.........G.....  ........A.....B.*
01B6E0   1B005890 C78441E0 C2325879 00005009   000012A7 4780C25A 9200701C 587A000C   *....G...B.............B.........*
01B700   500A000C 9180A00B 58AA001C 4780C244   58B0C7A0 07FB4510 C21258E0 C79007FE   *................B...G.....B...G...*
01B720   58C0C58C 9110800A 4780C276 18751858   45E0C15C 185758DF 0000D207 C7B8D000   *..E.......B.....A.......K.G...*
01B740   9801C7B8 45E0C4A4 47F0C2AE 92C0800A   D7078000 8000D702 80198019 58FF0010   *..G...D..0B.....P.....P.........*
01B760   41FF0000 50F80014 98015020 0A03900F   40301B77 50780004 92D0800A 50180004   *......8.....................*
01B780   5890C784 58790004 41990008 91409000   4710C2F0 5829000C 58220000 D5032004   *..G.......... ....B0........N...*
01B7A0   20644770 C2F0D507 20604030 4780C336   41990010 4670C2CC 5810C828 45B0C4C8   *....B0N.......C.......B...H...DH*
01B7C0   12224720 C36C5854 00005890 C784D202   500D9001 50590000 9201501C D2075010   *....C........G.K.............K...*
01B7E0   C81858E0 001058E0 E0001B00 5000E000   58E0C790 07FED202 80199009 D2018002   *H.................G...K.....K...*
01B800   200258F0 C82041FF 000150F0 C820D202   800D9005 50890004 92009000 58F90000   *...0H.....0H.K.............9...*
01B820   5B90C784 40980000 47F0C2A0 45B0C452   45B0C48E 18ED58D4 0000585D 001C9201   *..G. ....0B...D...D....M.........*
01B840   501C58F2 000041FF 000050F5 00145862   000C5866 0000D207 60604030 D207D010   *...2.......5.............K...K...*
01B860   C800D202 500D2005 50520004 5860C820   41660001 5060C820 12EE4720 C3E6D202   *H.K.........H.....H.....CWK...*
01B880   5019E00E D2015002 E01B1892 5B90C784   40950000 D23B4030 D02050F4 006C5830   *....K.........G.....K.....4....*
01B8A0   001047F0 C11E5872 000C5867 0000D207   6010C810 D2076060 E000D702 20092009   *...0A.........K...H.K.....P.....*
01B8C0   47F0C3CA D2075010 00209801 C7A458F0   C7E405EF 18D1D23F D0207000 D71FD000   *.0C.K.......G..0GU...JK.....P...*
01B8E0   D0009212 D00992D0 D00A9202 D00B505D   001C50D4 000007F2 1B77507A 00009200   *.................M...2.........*
01B900   601C5816 000C5076 000C1261 4770C43E   07FB5872 00041277 078BD502 20097019   *.............D.............N.....*
01B920   4770C46E D2007018 701C92FF 701C5877   000C47F0 C456D202 D00D2005 50D20004   *..D.K.............0D.K......K...*
01B940   5860C820 41660001 5060C820 07FE4162   00105960 C82C4770 C49E5860 C8245060   *..H.......H.........H...D...H...*
01B960   C82807FB 58F0C79C 41FF0000 12FF078E   590F0008 4770C4C0 591F000C 478E0004   *H.....0G.............D.........*
01B980   58FF0000 47F0C4A8 1B001821 1B331B88   91402000 4710C500 91202000 077B4150   *......0D.......... ....E.........*
01B9A0   C57E4590 C53812EF 4780C52C 12F34780   C4FC4190 C50045A0 C59207F9 183E1882   *E...E.....E..3..D...E...E..9....*
```

```
01E9E0   12114780 42CA9680 108894DF 108818D3   4BD044C4 58DD0000 9680D054 D5001000   *.............L...D.....N...*
01EA00   10904770 425294BF 10881277 47D04266   91801094 47804266 D6017098 109818D1   *.....................O.....J*
01EA20   18134B10 44C45811 00009104 10554780   42CA4110 00085012 00001812 0A0412FF   *.....D.....................*
01EA40   477042CA 5812000C D3001000 D09DD202   1005D091 92801004 18D34BD0 44C458DD   *.........L.....K.........L...D..*
01EA60   000058F0 D04C50F1 00005010 D04C12CC   478042C2 D2001000 C01C4110 00A85012   *...0...1.............BK.....*
01EA80   00001816 58601000 18D00700 47F042DC   F5000018 580042D8 41110000 0A0A180D   *.............0..5.....Q.....*
01EAA0   48103038 06104010 303847F0 40EC1277   47D04324 18A941F0 70081BAF 50A07004   *.....  ....0 .........0.....*
01EAC0   0690924B 90005890 30181299 4770431C   5070303C 50907000 50703018 12664780   *.....................*
01EAE0   43301B77 47F041E6 58703018 12774780   44029500 70044780 43D812CC 47804350   *.....0.W.............Q.....*
01EB00   9180C094 47E043D0 D5017089 30924770   43D85890 707C1B88 12994770 437A9110   *........N.....Q.............*
01EB20   707A4710 43D89610 707A9610 700447F0   43D89101 70044780 439A94FE 7004D200   *.....Q.........0.Q.........K.*
01EB40   90007000 D7029001 90019210 90059200   9007D402 90019001 477043C4 91109005   *....P.............M.....D....*
01EB60   471043D8 19894770 43BCD500 90007000   477043E4 96109005 47F04372 41809054   *...Q......N.......U.....0.....*
01EB80   58909000 47F0439A 59C07080 4780435A   58707000 41770000 47F04334 D20B705C   *.....0.................0..K...*
01EBA0   70009224 70609210 70614180 705C5680   90005080 900047F0 437212CC 47804444   *.....................0.....*
01EBC0   9180C094 47104444 59CC007C 47704444   900D3054 18D34BD0 44C458DD 000058F0   *.................L..D....0*
01EBE0   D0881B00 430C001C 12004780 444005EF   58300010 58303064 980D3054 1864184C   *.....  ..... ..... ..... ....*
01EC00   12004780 64621BAA 41B0300C 58C03040   58F00010 58F0F098 05EF1BAA 41B03008   *.....  .0...00.....*
01EC20   58C03040 58F00010 58F0F098 05EFD500   3039302D 47B064A8 1BAA41B0 303058D0   *....0...00...N.....*
01EC40   64F816BD 18D34BD0 64C658DD 000048D0   D00258C0 304058F0 001058F0 F09805EF   *.8...L...F.....  .0...00...*
01EC60   981E308C 58300010 18F007FE 97001000   F5000090 0001BEEA 0001BE40 00040008   *.........0......5.....  ....*
01EC80   0F0F20F5 00F09180 40C9C5C5 F4F0F0C9   40E3C8C5 E2C540D4 C5E2E2C1 C7C5E240   *....5.0.. IEE400I THESE MESSAGES *
01ECA0   C3C1D5C3 C5D3D3C5 C4406040 00007FFF   80000000 919C59A0 0003DC88 5001B4E0   *CANCELLED ....................*
01ECC0   0001ECB8 00000000 5001ED0A 00000000   0003D340 0003BC68 0003D4E8 0003DC28   *................L  .....MY....*
01ECE0   0003DC88 0003DC28 0003D4E8 E2E8E2C9   00000002 A001F2F8 5001B4E0 0003D4FC   *..........MYSYSI......28.....M.*
01ED00   C8C5C1C4 D8C3C201 052018A1 186145D0   2A4847F0 2C4247F0 2C3A47F0 2C3245D0   *HEADQCB.........0..0..0....*
01ED20   290E1288 474024E2 45D02948 12774740   24E25897 00005549 00084780 205A9180   *.....S..... .S.............*
01ED40   90044780 24F25899 00004199 00001299   478024E2 55490008 47702034 91809004   *.....2.........S...........*
01ED60   47802508 91409004 4780206A 5839000C   45D02168 91016002 47802076 9200A003   *.....  ....................*
01ED80   45D02498 18B918C9 91809004 47802088   13C945D0 245C4100 00101819 45D024BC   *.......I.........I.........*
01EDA0   58D70000 41D0D000 190D4780 20D85897   00009180 90044780 20D012CC 47402100   *.P..........Q..............  ..*
01EDC0   45D02116 58990000 12994780 21009180   90044710 20B647F0 210045D0 211647F0   *.....................0.....0*
01EDE0   210045D0 28EA45D0 247C1801 181745D0   24BC5508 00084770 210018B8 45D0245C   *.....................  .....*
01EE00   41000018 181845D0 24BC91FF A00041AA   000C47E0 201445D0 224218A6 47F028B0   *.....................  .....0..*
01EE20   1BCC58F9 000C9500 F01C078D 43CF001C   06C042CF 001C9140 90044780 214E58C9   *...9....0.............  .....I*
01EE40   000858CC 005050C9 000C1BBB 43BC0028   41BB0001 42BC0028 9500F01C 077D18CA   *.......I.....................0.....*
01EE60   58A90008 58B02C8E 18FE05EB 18AC18EF   07FD1BBB 43B30028 95013028 4780217E   *.....................  .....*
01EE80   06B042B3 002807FD 95003029 47802176   18CF0700 45102192 00000064 58010000   *.....................  .....*
01EEA0   0A0A18FC 06B042B3 002818B1 D763B000   B0009202 B04850B0 B050920F B050D300   *.............P..............L..*
01EEC0   B050B01C 58C00010 58CC0014 50C0B054   9204B054 5030B058 9218B058 D201B05C   *.....................  .....K..*
01EEE0   303192FF B062D202 B063B062 41C00001   40C0B066 9202B004 9200B005 50B0B008   *.....K.....  ..... .....*
01EF00   41C0B030 50C0B014 50B0B018 D203B025   303141C0 B03850C0 B02C9203 B0309220   *.............K..................*
01EF20   B03492EE B03718CF 4110B004 0A004110   B0004100 00010A01 8000222F 181B4100   *.....  ..... ..... .....*
01EF40   00644111 00000A0A 18FC07FD 91306002   078D183E 91206002 4780226E 47F0225E   *.....  ..... ..... .....0..*
01EF60   00000002 80000000 58002256 1B115610   225A0A4F 47F02286 47F0227A 00000001   *.....................0...0.....*
01EF80   80000000 58002272 1B115610 22760A4F   47F02110 052041F0 028C1B2F 1B005830   *.................0.....0.....*
01EFA0   2CA24180 30005508 0000078E 58880000   55080008 47802444 4130240E 58780008   *.....  ..... ..... .....*
01EFC0   1BBB5897 00004190 90005549 00081869   4780238C 91309004 477022D4 05B045D0   *.....  ..... ..... .....M....*
01EFE0   23DA07F3 0700189C 55490008 477022CA   186912BB 477023F6 91409004 47802316   *...3.....  ..... .....6. .....*
01F000   18F35839 000C91FF 300247E0 231095C0   300D47D0 231045D0 21681896 183F1BBB   *.3.....  ..... ..... .....*
01F020   45D023DA 47F023F6 12FF4770 23F612BB   47702332 91309004 471023F6 45D023DA   *.....0.6.....6.....  .....6.....*
01F040   47F023F6 12FF4770 23F658FC 000C41D0   2332189C 9500F01C 078D43AF 001C06A0   *.0.6.....6.....  .....0.....*
01F060   42AF001C 91409004 4780237E 58A90008   58CA0050 50C9000C 1BBB43BC 002841BB   *.....  ..... ..... .....I.....*
01F080   000142BC 002818C9 9500F01C 077D58AC   000847F0 215A9140 90044780 23A694BF   *.....I..0.....0...  .....*
01F0A0   900418F3 5839000C 45D02168 1896183F   91809004 471023C8 45D023DA 47F023F6   *...3.....  ..... .....H.....0.6*
```

```
0236C0   F3E04160 F3E05060 00489C00 10004710   F3BC4780 F36E9104 00444780 F3BC9604   *3..3.....3..3.....3...*
0236E0   200647F0 F34A9D00 10004710 F3BC4720   F36E4780 F3869106 00444780 F36E4122   *...03.....3..3..3.....3...*
023700   000A4910 F4244780 F3B29611 F45F47F0   F2CC4540 F3A64122 000A47F0 F3B2D207   *....4...3...4..02.. 3.....03.K.*
023720   2002F3D0 40120000 07F492FF 20005860   FC8807F6 41000A10 D20FFD29 F4485830   *..3. ....4.......6...K..4...*
023740   FC8C07F3 00000000 00000000 00000000   00000000 00000000 04000000 20000006   *...3.....................................*
023760   00000000 60000006 00000000 20000006   00000000 20000006 00000000 00000000   *.....-.....................-.....*
023780   F0F1F2F3 F4F5F6F7 F8F9C1C2 C3C4C5C6   00000000 00000000 00005000 00000000   *0123456789ABCDEF..,.............*
0237A0   00000000 000000F8 000007FF 00000000   00000000 00000000 00000000 00000000   *.......8.....-.....-.....-.....*
0237C0   C9C7C6F0 F0F6E240 C961D640 C5D9D940   00000000 00000000 00000000 00000000   *IGF006S I.O ERR ,...............*
0237E0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....-.....-.....-.....-.....*
      LINES 023800-023F40 SAME AS ABOVE
023F60   00000000 00000000 00000000 00000000   00000000 00000000 00000000 0E000A07   *.....-.....-.....-.....-.....*
023F80   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....-.....-.....-.....-.....*
023FA0   00000000 00016F70 00000000 00000000   00000000 00000000 00000000 00000000   *.....-.....-.....-.....-.....*
023FC0   00000000 00000000 00000000 00000000   0001A880 000229E0 00000000 FF07E604   *.....-.....-.....-.....W.*
023FE0   00000000 0003EEB8 0001A3A8 00000000   00017AE0 00000000 005E000B 0900006C   *.....-.....-.....-.....-.....*
024000   00A00000 005E000B 02028170 02900000   005E000B 03028038 01380000 005E000B   *.....-.....-.....-.....-.....*
024020   0401E130 06880000 005E000B 05025A90   00600000 005E000B 0601FF90 01B80000   *.....-.....-.....-.....-.....*
024040   005E000B 07028BB0 00480000 005E000B   0801E7B8 05000000 00000000 FFFFFFFF   *.....-.....-.....X.....-.....*
024060   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....-.....-.....-.....-.....*
      LINES 024080-024340 SAME AS ABOVE
024360   00000000 00000000 00000000 00000000   00000000 00000000 00000000 0001DD26   *.....-.....-.....-.....-.....*
024380   00024628 400200BC 00024380 000243CC   00000000 80024AA8 000244F8 4001FF92   *.....-.....-.....-.....8 ...*
0243A0   0002AE28 00024AF0 00000048 00024AA8   4002B048 00024628 00024380 000243CC   *.....0.....-.....-.....-.....*
0243C0   00000318 0003E0D8 00000000 00000000   0002B590 67040000 0000005F 00000000   *.....Q.....-.....-.....-.....*
0243E0   00024558 FFFF0000 C9C7C3F5 F1F0F7C2   C9C7C3F0 F1F0F7C2 00024628 00024428   *.....IGC5107BIGC0107B.....*
024400   0002B04E 00000000 0002B368 00000000   00000000 0003C170 00000BB8 00000000   *.....-.....A.....-.....-.....*
024420   00024378 00024380 80018780 80018780   80018780 80018780 80018780 000244F8   *.....-.....-.....-.....-.....8*
024440   0003CDB8 00000000 00000000 00000000   00000000 00530064 40000000 40000000   *.....-.....-.....-.....-.....*
024460   00000019 0003BAC0 000186C8 02000005   00024628 00000048 00024AA8 000000A8   *.....-.....H.....-.....-.....*
024480   00024488 20F5007C 00024428 0001E7B8   00024444 00000000 00000000 0003D340   *.....5.....X.....-.....L *
0244A0   00024378 0001F9B0 000401B8 0003D340   00017BD4 00017BD4 80018C58 00024428   *.....9.....L ...M...M.....*
0244C0   0003B740 0003D478 0001E7B8 00000000   0003B740 40018A42 0001F9B0 0003ED00   *.....-...M...X.....-.....9.....*
0244E0   0003D4E0 0001FAD6 70018C64 70006590   00024428 0003DCB0 18020000 00000000   *..M....O.....-.....-.....-.....*
024500   00024428 0002442C 00024430 00024434   00024438 00024628 00024670 000246B8   *.....-.....-.....-.....-.....*
024520   00024700 00024748 00024790 000247D8   00024820 00024868 000248B0 000248F8   *.....-.....-.....Q.....-.....8*
024540   00024940 00024988 000249D0 00024A18   00024A60 80024AA8 00000000 00000000   *.....-.....-.....-.....-.....*
024560   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....-.....-.....-.....-.....*
      LINES 024580-024600 SAME AS ABOVE
024620   00000000 00000000 80018780 00000000   0003ED88 00000318 C9C7C3F0 F1F0F7C2   *.....-.....-.....IGC0107B*
024640   00F00100 00000000 FFFF0000 0003E0D8   E0008800 000247D8 00000000 00000000   *.0.....-.....Q.....Q.....*
024660   00000000 80000000 0003E488 40000000   80018780 00000000 00000000 00000330   *.....-.....U.....-.....-.....*
024680   C9C7C3F1 F1F0F7C2 00600200 00000000   00000000 00000000 E0000800 00024628   *IGC1107B.....-.....-.....*
0246A0   00000000 00000000 000246B8 00000000   00000000 00000000 80018780 00000000   *.....-.....-.....-.....-.....*
0246C0   00000000 00000360 C9C7C3F2 F1F0F7C2   00A00300 00000000 00000000 0003ED38   *.....IGC2107B.....-.....*
0246E0   00000800 00024628 00000000 00000000   00024670 00000000 00000000 40000000   *.....-.....-.....-.....-.....*
024700   80018780 00000000 00000000 00000378   C9C7C3F2 F1F0F7C2 00A00400 00000000   *.....IGC2107B.....-.....*
024720   08000000 00000000 00001800 00024628   00000000 00000000 00000000 00000000   *.....-.....-.....-.....-.....*
024740   00000000 00000000 80018780 00000000   00000000 000003F0 C9C7C3F2 F1F0F7C2   *.....0IGC2107B*
024760   00A00500 00000000 08000000 00000000   00001800 00024628 00000000 00000000   *.....-.....-.....-.....-.....*
024780   00000000 00000000 00000000 00000000   80018780 00000000 00000000 000004F8   *.....-.....-.....-.....8*
0247A0   C9C7C3F2 F1F0F7C2 00A00600 00000000   08000000 00000000 00001800 00024628   *IGC2107B.....-.....-.....*
0247C0   00000000 00000000 00000000 00000000   00000000 00000000 80018780 00000000   *.....-.....-.....-.....-.....*
0247E0   00000000 00000438 C9C7C3F0 F1F0F7C2   00E00700 00000000 00000000 00000000   *.....IGC0107B.....-.....-.....*
024800   E0000800 00024628 00000000 00000000   00000000 00000000 00000000 00000000   *.....-.....-.....-.....-.....*
```

```
02B720   00000000 00000000 000E0004 00000000   C180000E 00000000 00000000 00000000   *.............A.............*
02B740   00000000 00000000 00000000 00000000   0002ED88 02401228 00000000 C1000040   *....................A..  *
02B760   0002B784 0302CC08 00000000 002F0000   00000000 0002E312 00000000 00000000   *.....................T.......*
02B780   000E0004 00000000 C180000E 22000000   00000000 00000000 00000000 00000000   *........A.....................*
02B7A0   00000000 00000000 0002B928 09101228   00000000 C1000040 0002B7DC 0302CC08   *..................A..........*
02B7C0   00000000 002F0000 00000000 0002E83B   00000000 80000000 000E0004 00000000   *.............Y...............*
02B7E0   C180000E 22000000 00000000 00000000   00000000 00000000 00000000 00000000   *A.....................................*
02B800   000336A8 034005A8 00000000 C1000080   0002B834 0302CBE0 00000000 00080000   *.....  .....A.....................*
02B820   00000000 00000000 00000000 00000000   00080004 00000000 C1800008 00000000   *.....................A.........*
02B840   00000000 00000000 00000000 00000000   00000000 00000000 000336A8 044005A8   *.................................*
02B860   00000000 C1000080 0002B88C 0302CBE0   00000000 00080000 00000000 00000000   *....A...........................*
02B880   00000000 00000000 00080004 00000000   C1800008 00000000 00000000 00000000   *.............A...............*
02B8A0   00000000 00000000 00000000 00000000   000336A8 054005A8 00000000 C1000080   *.....................  .....A...*
02B8C0   0002B8E4 0302CBE0 00000000 00080000   00000000 00000000 00000000 00000000   *...U............................*
02B8E0   00080004 00000000 C1800008 00000000   00000000 00000000 00000000 00000000   *........A.....................*
02B900   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
02B920   0002CB50 00000000 12280000 00000000   0002C916 00000000 00000000 00000000   *................I...............*
02B940   0000007D 0002BA64 0002CA04 0002CAC0   0002CB1F 0002CB1C 0002BAEA 0002C836   *.............................H.*
02B960   0002C916 0002C8CA 0002C916 0002C9AC   0002BA70 0002C880 0003D358 00000000   *..I...H...I...I.....H...L.....*
02B980   0002BAD0 00000000 000249D0 00000000   00000000 00000000 40404040 40404040   *.....................           *
02B9A0   40404040 40404040 40404040 40404040   40404040 40404040 40404040 40404040   *                               *
      LINES 02B9C0-02B9E0 SAME AS ABOVE
02BA00   40404040 40404040 40404040 40404040   40404040 40400000 40404040 40404040   *                    ..         *
02BA20   004A4E36 00000000 00940000 0047002F   00000002 004A0000 FFFFE840 E823082F   *.........................Y Y...*
02BA40   005E232F 005E2010 000040E0 00008000   0C000000 2E010100 00000000 00000000   *................................*
02BA60   00D70000 4000400A 4DA04DEA 4E364ECC   0702BA64 60000002 0102BAE0 60000E36   *.P.. . ........................*
02BA80   2702BA64 20000002 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
02BAA0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
      LINE 02BAC0 SAME AS ABOVE
02BAE0   2A822A02 40000FFF 2A444040 4040C9C5   C5F1F4F0 C940E2E8 E2E3C5D4 40C3D6D5   *.... ..... .    IEE140I SYSTEM CON*
02BB00   E2D6D3C5 E2000000 00000000 00000000   00000000 00000000 00000000 00000000   *SOLES..........................*
02BB20   00000000 00000000 00000000 00000000   00000015 40404040 4040C3D6 D5E2D6D3   *.....................       CONSOL*
02BB40   C561C1D3 E34040C3 D6D5C440 C1E4E3C8   40404040 40C9C440 C1D9C5C1 4040D9D6   *E.ALT  COND AUTH     ID AREA  RO*
02BB60   E4E3C3C4 00000000 00000000 00000000   00000000 00000000 00000000 00154040   *UTCD.........................  *
02BB80   40404040 4040E2E8 E2D3D6C7 40404040   4040C840 4040C3D4 C4E24040 40404040   *       SYSLOG      H   CMDS      *
02BBA0   40404040 40404040 C1D3D300 00000000   00000000 00000000 00000000 00000000   *         ALL.....................*
02BBC0   00000000 00000015 40404040 40404040   F0F0F961 F0F1C640 40404040 D4404040   *.....          009.01F      M   *
02BBE0   C1D3D340 40404040 40F0F140 40404040   4040C1D3 D3000000 00000000 00000000   *ALL       01        ALL.........*
02BC00   00000000 00000000 00000000 00000000   00154040 4040F0F0 C36BF0F0 C561F0F0   *.,.............      00C.00E.00*
02BC20   F9404040 4040C140 4040C1D3 D3404040   404040F0 F3404040 40404040 D5D6D5C5   *9     A   ALL      03        NONE*
02BC40   00000000 00000000 00000000 00000000   00000000 00000000 00000015 40404040   *.............................  *
02BC60   40404040 F0F0C661 F0F0F940 40404040   D5404040 D5D6D5C5 40404040 40F0F440   *    00F.009     N   NONE     04 *
02BC80   40404040 4040F500 00000000 00000000   00000000 00000000 00000000 00000000   *      5.........................*
02BCA0   00000000 00154040 40404040 4040F0F1   F861F0F0 F9404040 4040D540 4040D5D6   *.....          018.009     N   NO*
02BCC0   D5C54040 404040F0 F5404040 40404040   F5000000 00000000 00000000 00000000   *NE    05        5..............*
02BCE0   00000000 00000000 00000000 00000015   40404040 40404040 F0F2F861 F0F0F940   *.............          028.009 *
02BD00   40404040 D5404040 D5D6D5C5 40404040   40F0F640 40404040 4040F500 00000000   *    N   NONE     06        5.....*
02BD20   00000000 00000000 00000000 00000000   00000000 00000000 00154040 40404040   *.............................  *
02BD40   4040F0F1 C661F0F0 F9404040 4040D540   4040C1D3 D3404040 404040F0 F7404040   *  01F.009     N   ALL      07   *
02BD60   40404040 D5D6D5C5 00000000 00000000   00000000 00000000 00000000 00000000   *    NONE........................*
02BD80   00000015 40404040 40404040 F3C5F061   F0F0F940 40404040 D5404040 C1D3D340   *....          3E0.009     N   ALL *
02BDA0   40404040 40F0F840 E96BC140 4040D5D6   D5C50000 00000000 00000000 00000000   *     08 Z.A    NONE..............*
02BDC0   00000000 00000000 00000000 00154040   40404040 4040F2C4 F261F2C4 F3404040   *.............          2D2.2D3   *
02BDE0   4040D540 4040C1D3 D3404040 404040F0   F940E96B C1404040 C1D3D300 00000000   *  N   ALL      09 Z.A    ALL.....*
02BE00   00000000 00000000 00000000 00000000   00000000 00000015 40404040 40404040   *.....................          *
```

```
03B7A0   4007D482 00068F20 0001D4CA 6001DAEA   00068F20 0003B660 00000000 0003D340   * .M.......M.....................L *
03B7C0   00000000 0003D340 00000000 00000000   00000000 00000000 00040258 00000000   *.....,L .....................*
03B7E0   00000000 0003BD00 00000000 00000000   00000000 0003D538 0103BAC0 00000026   *.....................N.........*
03B800   40C9C5C1 F0F0F1C9 40E4D5C9 E340F2F3   F16B40D7 C1E3C840 F5F340C9 D5D6D7C5   * IEA001I UNIT 231. PATH 53 INOPE*
03B820   D9C1E3C9 E5C50000 00000000 00000000   00000000 00000000 00000000 00000000   *RATIVE.......................*
03B840   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....,.....,.....,.....,.....*
      LINE 03B860 SAME AS ABOVE
03B880   40000058 00000000 0200005D 80000000   10400000 01000000 10000000 00100078   * ...............,.....,.....*
03B8A0   0103D6A0 0000001A 40C9C5C6 F1F6F1C9   40F0F0C3 60D9C5C1 C4C5D940 C3D3D6E2   *..O.....  IEF161I 00C.READER CLOS*
03B8C0   C5C40000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *ED..,.....,.....,.....,.....*
03B8E0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....,.....,.....,.....,.....*
      LINE 03B900 SAME AS ABOVE
03B920   00000000 00000000 40000058 00000000   02000052 E0000000 40000000 00000000   *.....,.....,.....,.....,.....*
03B940   18000000 00091343 0003B5A8 00000078   0012D002 00000000 00040007 00018A40   *.....,.....,.....,.....,.....*
03B960   00001216 0003B9D8 0003BA50 0003BA38   0003BA38 0001A880 0003D770 0003B9D8   *.....Q.....,.....P....,Q*
03B980   0005B7C8 00000000 40055786 0005B7C8   00000024 400192D2 00000000 FFFFFFFE   *...H.... .....H.... ..K.....*
03B9A0   0001B4E0 00000000 0003B9B0 00000000   C9C7C3F5 F4F0F3C4 0003B9D8 6001DAEA   *.....,.....IGC5403D...Q....*
03B9C0   0103B7F8 00000025 5CC9C5C6 F2F3F3C1   40D440F2 F3F16BE2 C3D9E3C3 C86B6BE2   *...8.....IEF233A M 231.SCRTCH.,S*
03B9E0   C1D4D7D3 C4D4D76B D3D2C5C4 40000000   00000000 00000000 00000000 00000000   *AMPLDMP.LKED ................*
03BA00   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....,.....,.....,.....,.....*
      LINE 03BA20 SAME AS ABOVE
03BA40   00000000 00000000 80000058 00000000   02000058 80000000 14000000 1C000000   *.....,.....,.....,.....,.....*
03BA60   40000000 00092859 00000000 00080408   E2C1D4D7 D3C4D4D7 E2C1D4D7 D3C4D4D7   * ..............,.SAMPLDMPSAMPLDMP*
03BA80   40404000 F0000000 0003BA98 00000000   00000000 00000000 00000000 80040238   *    .0.....,.....,.....,.....*
03BAA0   00000000 00000000 0003B948 00000018   00000000 00000000 00000000 00000000   *.....,.....,.....,.....,.....*
03BAC0   01000000 00000026 40C9C5C1 F0F0F1C9   40E4D5C9 E340F2F3 F06B40D7 C1E3C840   *........ IEA001I UNIT 230. PATH *
03BAE0   F5F340C9 D5D6D7C5 D9C1E3C9 E5C50000   00000000 00000000 00000000 00000000   *53 INOPERATIVE...............*
03BB00   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....,.....,.....,.....,.....*
      LINE 03BB20 SAME AS ABOVE
03BB40   00000000 00000000 40000058 00000000   0200005E 80000000 10400000 01000000   *.....,.....,.....,.....,.....*
03BB60   10000000 00100999 0003BAA8 00000088   000065A8 000065A8 000065A8 0007C3B0   *.....,.....,.....,.....C.....*
03BB80   000065A8 00000000 00000106 00002BE0   10000000 0403D340 1003D578 88000000   *.....,.....,.....L ..O.....*
03BBA0   8F000000 02000000 1B000000 FF068E70   0403BB70 1800206C 00000003 0005000D   *.....,.....,.....,.....,.....*
03BBC0   00040064 1800206C 00000001 00050002   0004000A 00010001 00000000 00000000   *.....,.....,.....,.....,.....*
03BBE0   0000007D C2C2C2C1 C3D1C3C4 00000000   0003D318 00000078 1403D770 00020400   *....'BBBACJCD.....L.....P.....*
03BC00   000203F0 05AD3DD6 FF05002F 040B1A7A   00068FB0 0007D7C2 40404040 4040000A   *...0...O.....PB       ..*
03BC20   0F66C000 00000000 F11000D8 4C99366F   00000000 20000000 00000000 00000000   *.........1..Q.....,.....,.....*
03BC40   00000000 00000000 0003BB68 000000B0   00000004 000A1500 00000000 00000000   *.....,.....,.....,.....,.....*
03BC60   000B1100 000C1900 00080308 F6F0F5C1   0012D002 00000000 0004000C 5005BFA6   *.....,.....605A.....,.....,.....*
03BC80   00010C05 0003D478 001058B0 0003D4D8   80018F42 0001A880 0003D340 0003D478   *.....M.....MQ.....L ..M.*
03BCA0   0403D340 0003D478 0003D340 40018E8A   0003D340 8F068E70 0003D894 0003D4FC   *..L ..M...L ....L .....Q...M.*
03BCC0   4001A1A8 00000000 E2E8E2C9 C5C1F0F1   00000000 00000000 00000000 00000000   * .....SYSIEA01.....,.....,.....*
03BCE0   00000000 00000000 00000000 00000000   000C1901 80000C18 0003D5C0 00000088   *.....,.....,.....,.....N.....*
03BD00   00000000 0003D770 0003BD38 0003DB38   FD000080 00000000 0003BD7C 0003BD58   *.....P.....,.....,.....,.....*
03BD20   00000000 00000000 00000000 0840C1F2   00000000 00000000 0000F158 0000F52A   *.....,.....  A2.....1...5.*
03BD40   0005A800 00069000 001D001E 00000000   00000000 00000000 E2C1D4D7 D3C4D4D7   *.....,.....,.....,.....SAMPLDMP*
03BD60   0000D43C 0099366F D6E2F6F5 40404040   40404040 01C00000 00000000 0003BD58   *..M.....OS65         .....,.....*
03BD80   0103B8A0 00000024 40C9C5C6 F2F3F7C9   40F0F0C3 404040C1 D3D3D6C3 C1E3C5C4   *.........  IEF237I 00C   ALLOCATED*
03BDA0   40E3D640 C9C5C6D9 C4C5D940 00000000   00000000 00000000 00000000 00000000   * TO IEFRDER .....,.....,.....*
03BDC0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....,.....,.....,.....,.....*
      LINE 03BDE0 SAME AS ABOVE
03BE00   00000000 00000000 40000058 00000000   02000051 80000000 42000000 00000000   *.....,.....,.....,.....,.....*
03BE20   04000000 00085988 0103BD80 0000002E   40C9C5C5 F2F3F6C9 40C1D3D3 D6C34B40   *.....  IEF236I ALLOC. *
03BE40   C6D6D940 D9C4D940 40404040 40F0F0C3   40404040 40404040 40404040 40400000   *FOR RDR      00C             ..*
03BE60   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....,.....,.....,.....,.....*
```

```
03CC60   40404040 40404040 40404040 40404040   40404040 40404040 40404040 40404040   *                                       *
03CC80   40404040 40404040 40404040 40404040   40404040 40004502 8040F0F0 F0F1F0F0   *                          .,,, 000100*
03CCA0   40F0F0F0 F040C9C5 C1F2F0F8 C94040E2   E8E2F14B C4E4D4D7 40404040 40404040   * 0000 IEA208I  SYS1.DUMP         *
03CCC0   40404040 C6E4D5C3 E3C9D6D5 40C9D5D6   D7C5D9C1 E3C9E5C5 40400019 028040F0   *    FUNCTION INOPERATIVE  ...  0*
03CCE0   F0F0F5F4 F840F0F0 F0F040D9 40F0F16B   7DE47D00 1E028040 F0F0F0F5 F5F340F0   *00548 0000 R 01.,U....,  000553 0*
03CD00   F0F0F040 E540F1F5 F06BD6C6 C6D3C9D5   C50033E0 8040F0F0 F0F5F4F9 40F0F0F0   *000 V 150.OFFLINE.... 000549 000*
03CD20   F040C9C5 C5F3F0F8 C940C4C9 E2D7D3C1   E84040E3 C5D9D440 D3C5D5C7 E3C840C5   *0 IEE308I DISPLAY  TERM LENGTH E*
03CD40   D9D9D6D9 F416B004 4FC0F410 42C0A001   D200A000 F4454160 00034480 F4504158   *RROR4.....4.....K..4.......4...*
03CD60   50004660 F1BA92FF 50004150 50015050   100041A0 001018B4 1BBA48A0 B0049518   *....1.....................*
03CD80   F4474770 F1F641A0 010050A0 00AC41A0   001647F0 F2909509 F4474780 F28C49A0   *4...16.....,..02...4...2...*
03CDA0   F4084780 F29C49A0 F40A4770 F2165850   00AC47F0 F23E49A0 010402F0 0000001A   *4...2...4...2.....02.....0....*
03CDC0   40E2E8E2 D3C9C240 D6D540F2 F3F40000   404DD761 D960D7E4 C25D0000 00000000   * SYSLIB ON 234...P.R.PUB.......*
03CDE0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
      LINES 03CE00-03CE20 SAME AS ABOVE
03CE40   40000058 00000000 02000016 80000000   50000000 00000000 10000000 00050998   * ...........,..................*
03CE60   0003BB68 00000060 002803C8 F0F0F2C8   0012D002 00000000 FF040001 600195E8   *...........H002H..............Y*
03CE80   0000B916 0003D248 00000000 00002928   0003D508 0001A880 40018E8C 0003D2A8   *......K.........N.....,.....K.*
03CEA0   00067C08 00067AA8 0003D374 00000003   00067C2A 00002928 00067AAD 00069668   *..........L.....,.....,.....*
03CEC0   0103C020 00000029 40C9C5C6 F5F0F3C9   40E6D9D6 D5C740C4 C5D5E2C9 E3E840D6   *........ IEF503I WRONG DENSITY O*
03CEE0   D940C9D5 C3D6D9D9 C5C3E340 D3C1C2C5   D3000000 00000000 00000000 00000000   *R INCORRECT LABEL.............*
03CF00   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
      LINE 03CF20 SAME AS ABOVE
03CF40   00000000 00000000 40000058 00000000   0200004C 80000000 20000000 50000000   *........ ....,.....,.....,.....*
03CF60   10000000 00085506 0103CEC0 0000000F   40C9C5C5 F2F3F4C5 40C440F2 F8F04000   *.............. IEF234E D 280 .*
03CF80   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
      LINES 03CFA0-03CFC0 SAME AS ABOVE
03CFE0   00000000 00000000 00000000 00000000   40000058 00000000 0200004B 80000000   *....,.....,.....,.....,.....*
03D000   20000000 70000000 20000000 00085451   0103CF68 00000029 40C9C5C6 F5F0F3C9   *....,.....,.....,.....IEF503I*
03D020   40E6D9D6 D5C740C4 C5D5E2C9 E3E840D6   D940C9D5 C3D6D9D9 C5C3E340 D3C1C2C5   * WRONG DENSITY OR INCORRECT LABE*
03D040   D3000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *L...............,.....,.....,.....*
03D060   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
03D080   00000000 00000000 00000000 00000000   00000000 00000000 40000058 00000000   *....,.....,.....,.....,.....*
03D0A0   0200004A 80000000 20000000 50000000   10000000 00085429 0003CE60 00000028   *....,.....,.....,.....,.....*
03D0C0   0003D690 0003D248 00000000 0003E360   00000000 02FFD8F8 0003D0B8 00000008   *..O...K.......T.......Q8.....*
03D0E0   0103D010 00000037 40C9C5C1 F0F0F0C9   40F2F8F0 6BC4C3D2 6BF0F26B F0C5F0F0   *........ IEA000I 280.DCK.02.0E00*
03D100   6BF0F8F5 F2F0F3F8 F0F0F06B 6B6BD9C4   D9404040 40406BF0 F04BF0F8 4BF5F200   *.0852038000...RDR     .00.08.52.*
03D120   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
      LINE 03D140 SAME AS ABOVE
03D160   00000000 00000000 40000058 00000000   02000049 80000000 20600000 98000000   *........ ....,.....,.....,.....*
03D180   10000000 00085322 0003D0B8 00000060   0003D010 0003CF68 0003CEC0 C003D200   *.....,.....,.....,.....,.....K.*
03D1A0   0003D0B8 00000048 40F0F0F0 F7F5F440   F0F0F0F0 40C9C5C6 00000000 0003D1D0   *.....,.....  000754 0000 IEF.....J.*
03D1C0   00000000 06FEF3F3 00000000 00000000   00040002 04000000 FF050000 0002B590   *.....33.....,.....,.....*
03D1E0   00000000 00018780 0103BF78 0103BED0   0103BE28 0103BD80 0103B8A0 C003E798   *.....,.....,.....,.....X.*
03D200   0003D188 00000038 0003BED0 0003BE28   0003BD80 C003D650 0003D188 000000C8   *..J.....,.....O...J....H*
03D220   00000000 0003D2E0 00000000 06FEF3F3   F3F3F3F3 60042C88 0103B9C0 00000025   *.....K.....,333333.....,.....*
03D240   5CC9C5C6 F2F3F3C1 40D440F2 F3F06BE2   C3D9E3C3 C86B6BE2 C1D4D7D3 C4D4D76B   *.IEF233A M 230.SCRTCH.,SAMPLDMP.*
03D260   D3D2C5C4 40000000 00000000 00000000   00000000 00000000 00000000 00000000   *LKED ..........,.....,.....,.....*
03D280   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *................................*
      LINE 03D2A0 SAME AS ABOVE
03D2C0   80000058 00000000 02000057 80000000   14000000 1A000000 40000000 00092855   *....,.....,.....,..... .....*
03D2E0   0003D200 00000038 00000000 E2E8E2E5   E3D6C340 0005C248 00000000 00000000   *..K.....,.....SYSVTOC ..B.....,*
03D300   00040002 04000000 FFF50001 4007C850   00000000 0003BC68 0003DB30 000000E0   *.....5.. .H.....,.....,.....*
03D320   00000000 0003D478 FE000088 0003BBF8   00043004 40073402 0405A5D0 07027568   *.....M.....,8.....,.....,.....*
03D340   0003BC68 00000000 0003B354 0003D850   80000064 00000000 01040268 F0840400   *.....,.....Q.....,.....0.....*
03D360   00001B1B 0003E2D0 00000000 0003D948   00068800 00068D38 0003D2E0 0005BB1C   *.....,.....S.....R.....,.....K.....*
```

```
03D380   0003D340 9005B9A8 0005B858 8003B468   0003B501 00DF4500 000001AA A005B99C *..L ..............................*
03D3A0   40018A42 00068D4C 5005BFA6 0007C828   01068F68 0003B740 00000000 0003D340 * .............H.......  .....L *
03D3C0   00000000 0003D770 0003B660 00000000   0003DCF4 20000000 00040258 0003B328 *......P.............4.............*
03D3E0   00000000 0003BD00 00000000 00000000   00000000 8703D538 0003E360 00000080 *.....................N...T.....*
03D400   000065A8 000065A8 000065A8 000065A8   000065A8 00000000 00000105 000007E0 *................................*
03D420   0F000000 0003D770 00000000 A8000000   00000000 01000000 FB000000 FF03E368 *......P.....................T.*
03D440   04026D94 580028A8 00000015 00000015   00080009 00000000 00000000 00000000 *................................*
03D460   00000000 00000000 00000000 00000000   00000000 0003D538 00180220 F9F0F1C3 *....................N.....901C*
03D480   0012D002 00000000 00040033 5001902A   00011113 0003E7B0 00000030 80000064 *.....................X..........*
03D4A0   0003E368 00000001 0003D968 0003D770   0003D8BC 0003E1D8 0003DCD0 0003BD00 *..T.......R...P...Q....Q.........*
03D4C0   0003DCF8 00000000 4007D482 0105AD50   0001D4CA 0000000C 000029BE 8F068E70 *...8.....M.......M.............*
03D4E0   00000000 00000000 FF030000 0003D4F4   0003D4FC E2E8E2C9 C5C1F0F1 C9C5C140 *.............M4..M.SYSIEA01IEA *
03D500   C1C2C5D5 C4F90000 0103D238 0000001F   40C9C5C6 F2F3F4C5 40D940F2 F3F16BF9 *ABEND9....K.....IEF234E R 231.9*
03D520   F9F9F9F9 F96BE2C1 D4D7D3C4 D4D74000   00000000 00000000 00000000 00000000 *99999.SAMPLDMP ................*
03D540   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
      LINE 03D560 SAME AS ABOVE
03D580   00000000 00000000 00000000 00000000   40000058 00000000 02000056 80000000 *.....................  ..............*
03D5A0   10000000 58000000 20000000 00092850   0003D2E0 00000010 0003D340 0000206C *.....................K......L .....*
03D5C0   0003D620 00000060 FD000058 7F000000   42000000 7F03D5CC 000599C8 0C000000 *..O.....................N....H.....*
03D5E0   4003D600 000270C0 00009A80 00000000   00000000 24000C15 2303D618 40000001 * .O.....................O.  ....*
03D600   3103D5F3 40000005 0803D600 00000000   0D03DA84 00000014 00000000 68000000 *..N3 .....O.....................*
03D620   0003D8C8 00000030 000C0E00 0003DA60   0003DA84 0003D8D0 0003BA68 00000A0A *..QH.....................Q..........*
03D640   00000000 00000000 80040238 C3C30000   00068000 00000000 00068000 00000800 *.............CC..............  .....*
03D660   0003B638 00000040 00000000 00000000   00003BE0 07000000 0003D340 00000000 *.......  .....................L ....*
03D680   08000000 00000000 00000000 00000000   0F05AE94 00000000 00000000 00000000 *................................*
03D6A0   0103D508 0000001F 40C9C5C6 F2F3F4C5   40D940F2 F3F06BF7 F7F7F7F7 F76BE2C1 *..N.....IEF234E R 230.777777.SA*
03D6C0   D4D7D3C4 D4D74000 00000000 00000000   00000000 00000000 00000000 00000000 *MPLDMP ..........................*
03D6E0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *................................*
      LINE 03D700 SAME AS ABOVE
03D720   00000000 00000000 40000058 00000000   02000055 80000000 10000000 98000000 *.....  ..... ........................*
03D740   20000000 00092842 0003E3A0 000000E0   F06B7DE4 7D000000 00000000 00000000 *.........T.....Q..U............*
03D760   00000000 00000000 00000000 00000000   00040238 00000000 0003D424 0003E1D8 *.....................M....Q*
03D780   00000000 00000000 01040258 F0000000   0000FFFB 00000000 00000000 00000000 *.............Q..............*
03D7A0   00000001 FFFC2318 0003BA9C 0103BA68   0003D968 0003D770 00000000 00000000 *.....................R...P.....*
03D7C0   0003DCD0 0003BD00 0003D9BC 0003D340   4007D482 0003D98C 10000000 6001DAEA *.........R...L  .M...R.....*
03D7E0   00000000 0003D340 0003BBF8 0003D770   00000000 0003DD58 0003D340 00000000 *.....L ...8..P.....  .....L ....*
03D800   0003E11C 00000000 00040258 0003BBF0   00000000 0003BD00 00000000 00000000 *.................0..............*
03D820   00000000 0003E720 0005B800 00000000   0005B800 00000800 00000010 00000001 *.....X..........................*
03D840   800007A8 0005B858 0003BCF8 00000080   E2C1D4D7 D3C4D4D7 C7D64040 40404040 *.............8....SAMPLDMPGO    *
03D860   E3C5E2E3 40404040 14040140 D7C7D47E   5C4BC4C4 000A1400 800028A8 14000040 *TEST     ... PGM..DD.............  *
03D880   E2D5C1D7 D7C5D940 000A1800 00000000   14040140 E2E8E2C1 C2C5D5C4 000B0F00 *SNAPPER .........  SYSABEND.....*
03D8A0   8000206C 14040100 C4C5D3C5 E3C54040   000B1A00 800028A8 00000000 0003D850 *.........DELETE  ..............Q.*
03D8C0   0000006C 404007FE 0003D8E0 00000018   000C1920 0003DA84 00000000 00000C1A *.....  ....Q..........................*
03D8E0   0003E758 00000060 FD000058 7F000000   42000000 7F03D8EC 000681C8 0C000000 *..X.....................Q....H.....*
03D900   4003D920 000270C0 00009A80 00000000   00000000 24000A0A 2303D938 40000001 * .R.....................R.  ....*
03D920   3103D913 40000005 0803D920 00000000   0D03BB98 000000B0 0803D930 00000000 *..R.  .....R.............R.....*
03D940   0003D5B0 00000008 0303E650 00000000   C9C7C3F0 C1F0F5C1 0105B858 2003D838 *..N.......W.....IGC0A05A.....Q.*
03D960   0003E7D0 00000188 0103BA68 00007170   0003D770 0003DB48 000A0A00 000A1900 *..X.............P..............*
03D980   000A1900 000273D8 00000000 00000000   00000000 00000000 00000000 0003D9D0 *.......Q.....................R.*
03D9A0   00A00200 00000000 00000000 0003DA84   0001D4CA 0005BB80 00000094 0002BF20 *.....................M..........*
03D9C0   00000000 00000094 00000000 0003DB30   00000004 0003DA60 00000020 4007D520 *.....................  ....N.*
03D9E0   0003BD00 0840C1F2 0003D8BC 0103BA68   5C03DCF8 0003D968 0003D770 0003BB98 *.....  A2..Q.....8...R...P.....*
03DA00   0003D538 0003DCD0 00FFFFF8 0003DCF8   00000000 4007D482 0003BB98 000A0A00 *..N.....8...8....  .M.........*
03DA20   0003BC48 000A1900 000000AD 00000023   00000089 00067C58 00067E48 000B0E00 *................................*
03DA40   70061226 00067D80 0003D968 0003DA10   400612BC 00000000 00000000 00000000 *.....R.....  ..................*
```

```
03DA60   00000000 0003D994 0000E811 00000228   04210000 01120000 0003DAA8 0003D8EC   *.....R...Y.....................Q.*
03DA80   0003DA18 00000000 00000000 0100FD07   00000203 00200000 01270000 0003DAA8   *.....................................*
03DAA0   0003D5CC 20067E40 01000000 0103BA70   0003E764 C9D5C9E3 40404040 00000000   *..N.....,.....X.INIT    ....*
03DAC0   00000000 DB00FB00 0003E7D8 0003DB08   0003D538 00000000 00000000 00000000   *.........XQ.....N..............*
03DAE0   00000000 0003D4B0 0003BA68 04084000   C9D5C9E3 40404040 C9D5C9E3 40404040   *.....M........INIT    INIT    *
03DB00   40404001 00010000 0003DB18 00000000   40404040 40404040 0003E7B0 800401D0   *   .............          ..X.....*
03DB20   40404040 40404040 0003D940 00000008   0003D3F8 00000068 FD00005C 00200000   *        ..R .....L8............*
03DB40   0018001C 002C002C 0040003C 0054004C   00000000 28A80100 00000000 00000000   *.....,.....,.....,.....,.....*
03DB60   00000000 00000100 00000000 00000000   00000000 206C0100 000001B4 00000000   *.....,.....,.....,.....,.....*
03DB80   00000000 28A80100 00000000 00000000   00000000 00000000 0003DE10 00000080   *.....,.....,.....,.....,.....*
03DBA0   000065A8 000065A8 000065A8 000065A8   000065A8 00000000 00000000 000007E0   *.....,.....,.....,.....,.....*
03DBC0   0F000000 000186C8 1003DE3C 08000000   03000000 01000000 FF000000 0F03E2D8   *.......H.....,.....,.....SQ*
03DBE0   0203DBA0 33002498 00000000 00000000   00000000 00000000 00000000 00000000   *.....,.....,.....,.....,.....*
03DC00   00000000 00000000 00000000 00000000   00000000 00000000 00000000 0003DC28   *.....,.....,.....,.....,.....*
03DC20   0003D340 0003BC68 0003DC18 0003DC88   00000000 03F0C9C5 C1042800 00000000   *..L .....,.....,..0IEA.....*
03DC40   0003EA70 00000040 00068FF8 0003DAE8   00068FF0 0003DB38 0003DC80 8003DC60   *.......,...8...Y..,0.....,.....*
03DC60   00200000 0003EB18 C080C9C5 C5E5C9C3   00000000 00000000 00000000 00000000   *.....,.....IEEVIC.....,.....,.....*
03DC80   0003DB28 00000008 00000000 0001ECB8   0003DC28 E2E8E2C9 C5C1F0F1 0003ECA8   *.....,.....,.....SYSIEA01.....*
03DCA0   0005B000 0003D828 0005B000 00000800   8103BAC0 00000000 00000000 00000000   *.....Q.....,.....,.....,.....*
03DCC0   00000000 C0000000 0003D848 00000068   0003D340 0003E368 5C03DCF8 0003D968   *.....Q.....,..L ..T...8..R.*
03DCE0   0003D770 0003D8BC 0003DCF4 0003BA9C   8003D648 80040238 0003DD14 0103E368   *..P...Q.....4.....O.....T.*
03DD00   0003DCF4 00000000 00000000 02000000   FF2EE462 C7D64040 40404040 0003D538   *...4.....,...U.GO       ..N.*
03DD20   00000000 00000000 00000000 4005AC3A   00040030 0003000E0 00027158 000198EF   *.....,..... ...,.....,.....*
03DD40   0003DDE4 00024428 6001B4E0 0003DDE0   00040260 00000000 000401D0 00000000   *...U.....,.....,.....,.....*
03DD60   00000000 0003EA7C 00000000 00000000   01040210 00000000 0000FFFF 00000000   *.....,.....,.....,.....,.....*
03DD80   00000000 00000000 00000001 FFFC1EF0   0003DB1C 0B03DAE8 0003D968 0003DD58   *.....,.....O.....Y..R.....*
03DDA0   0003E3A8 00000001 0003E0F8 00000000   0003D9B0 0003D770 4007D482 0003E3A8   *..T.....,.....8.....R...P...M...T.*
03DDC0   4007D520 6001DAEA 00000000 0003DEB8   00000000 00003DD58 00040058 000188B8   * .N.....,.....,.....,.....*
03DDE0   0003D770 00000000 00000000 00000000   00040390 00003D748 00000000 00000000   *..P.....,.....,.....P.....*
03DE00   00000000 00000000 00000000 0003E830   00000000 00000080 000065A8 000065A8   *.....,.....Y.....,.....,.....*
03DE20   000065A8 000065A8 000065A8 00000000   00000000 000007E0 0F000000 000186C8   *.....,.....,.....,.....H*
03DE40   10000000 08000000 03000000 01000000   FF000000 0F03ED88 0203DE18 33000318   *.....,.....,.....,.....,.....*
03DE60   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....,.....,.....,.....,.....*
03DE80   00000000 00000000 00000000 00000000   0003EC08 000000E0 0003DE10 00000080   *.....,.....,.....,.....,.....*
03DEA0   000065A8 0007D808 000065A8 000065A8   000065A8 00000000 000401F0 00000000   *.....Q.....,.....,.....0.....*
03DEC0   0003E974 0003E718 00000000 00000000   010406B8 00000000 0000FFFB 00040580   *..Z...X.....,.....,.....,.....*
03DEE0   00000000 800404F8 00000001 FFF9230C   0006DEFC 0003FF08 8006DCFC 0006DCF8   *.......8.....9.....,.....,.....8*
03DF00   0006DEFC 0006DCF4 E2C4F7F9 5006AF8A   0006AF88 00000003 0006DCFC 0006DFA8   *.......4SD79.....,.....,.....*
03DF20   5006ADC0 6001DAEA 0006DFA8 0003D770   00000000 0003DEB8 00000000 00040058   *.....,.....P.....,.....,.....*
03DF40   00000000 00000000 0003E6D4 00000000   000406C8 0003E948 00000000 00000000   *.....,.....WM.....H..Z.....,.....*
03DF60   00000000 00000000 00000000 00040188   0103D0E0 00000024 40C9C5C6 F2F3F7C9   *.....,..... IEF237I*
03DF80   40F0F0C5 404040C1 D3D3D6C3 C1E3C5C4   40E3D640 C9C5C6D9 C4C5D940 00000000   * 00E   ALLOCATED TO IEFRDER ....*
03DFA0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....,.....,.....,.....,.....*
      LINE 03DFC0 SAME AS ABOVE
03DFE0   00000000 00000000 00000000 00000000   00000000 00000000 40000058 00000000   *.....,.....,.....,.....,.....*
03E000   02000043 80000000 42000000 58000000   04000000 00071687 0103DF70 0000002E   *.....,.....,.....,.....,.....*
03E020   40C9C5C6 F2F3F6C9 40C1D3D3 D6C34B40   C6D6D940 E6E3D940 40404040 40F0F0C5   * IEF236I ALLOC. FOR WTR      00E*
03E040   40404040 40404040 40404040 40400000   00000000 00000000 00000000 00000000   *                ...............*
03E060   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....,.....,.....,.....,.....*
      LINE 03E080 SAME AS ABOVE
03E0A0   40000058 00000000 02000042 80000000   42000000 01000000 04000000 00071684   * .....,.....,.....,.....,.....*
03E0C0   0003CE60 0103CDB8 010402F0 0103EB38   01040110 C003EAF8 8003BAC0 00000000   *.....,.....0.....,.....8.....,.....*
03E0E0   00000000 00000000 00000000 C0000000   0003E1D0 00000068 0003D770 00000000   *.....,.....,.....,.....P.....,.....*
03E100   5C03E120 0003D968 0003DD58 0003E1F4   0003E11C 0003DB1C 8003E830 800401D0   *.....R.....,.....4.....,.....Y.....*
03E120   0003E13C 01000000 0003E11C 00000000   00000000 02000000 FFFC0062 C9C5C6C9   *.....,.....,.....,.....IEFI*
```

```
03E140   C9C34040 0003E720 00000000 00000000   00000000 4005AC3A 00000000 0006F6A2  *IC  .,X.............................6.*
03E160   00000000 0003FE78 00000068 0003E200   000005C8 0003C228 E2E8E2D9 C5E221F2  *.............S.,..H..B.SYSRES.2*
03E180   F3F3E800 000716B8 E2E8E2D9 C5E221F2   F3F3E700 00071728 00000000 40000000  *33Y.,,,,SYSRES.233X.........   ...*
03E1A0   00000000 29282928 00000901 00000000   0003E1B8 00000000 C9C7C3F0 F4F0F3C6  *.....................,,,,IGC0403F*
03E1C0   10000000 00D30000 00000000 00000000   0003E4A0 00000030 C9D5C9E3 40404040  *.....L.............U.....INIT    *
03E1E0   E2C1D4D7 D3C4D4D7 40404040 40404040   00000000 0003E1D8 0000001C 02FFD8F4  *SAMPLDMP       ..,,,,,Q......Q4*
03E200   D4C1E2E3 C5D94040 E2C3C8C5 C4E4D3D9   40404040 40404040 14000000 C9C5C6D9  *MASTER  SCHEDULR          ....IEFR*
03E220   C4C5D940 00000000 80000000 14000000   C9C5C6C4 C1E3C140 00000000 80000000  *DER .,,,,,,,,,,.IEFDATA ........*
03E240   14000000 C9C5C6D7 C4E2C940 00000000   80002928 14000080 E2E8E2D1 D6C2D8C5  *....,IEFPDSI .,,,,,,,,,,,SYSJOBQE*
03E260   00000000 80002968 14000040 C9C5C5D3   D6C7E740 00011900 80002928 140000C0  *.........  IEELOGX .............*
03E280   C9C5C5D3 D6C7E840 00011A00 80002928   14000000 E2D4C6D4 C1D5E740 00000000  *IEELOGY .............SMFMANX ....*
03E2A0   80000000 14000000 E2D4C6D4 C1D5E840   00000000 80000000 00000000 40000060  *........SMFMANY ..............,...*
03E2C0   1203E390 00000008 7F000000 42010000   0003E488 0103D948 00000000 00000000  *..T.............,.U...R.........*
03E2E0   00000000 00000000 00000000 00000001   00004000 0003E318 04000001 00000000  *.,,,,,,,,,,,,,,,,,,,,...T.,,,,,,,,*
03E300   0018D008 0003DBC4 1200D008 00000000   00000000 00000000 42008000 7F0249D0  *.......D.....................*
03E320   0002BA78 0C000000 4002BA70 0003E2D8   00000000 00000100 00000000 60000001  *.,,,,,,,,.....SQ..,,,,,,,,,,,,,,*
03E340   0803E350 00000000 09000000 60000001   03000000 20000001 0B000000 20000001  *..T.,,,,,,,,,,,,,,,,,,,,,,,,,,,,*
03E360   0003DCC8 00000040 00000000 00000000   00000000 00021A12 00280000 00000001  *...H.. .....,,,,,,,,,,,,,,,,,,,,*
03E380   00000200 00000001 04000001 C0000000   0018D008 0003D424 1200D008 0003D838  *.,,,,,,,,,,,,,,,,,,,,,M.....Q.*
03E3A0   0003E0F0 00000068 0003E3A8 00061408   00000000 00000000 01000000 0003DAF0  *...0.....T.....................0*
03E3C0   00000000 40404040 03007F0F 00000128   046B0000 007F0000 0003E3B8 0003E844  *....    .....................T...Y.*
03E3E0   0103DA18 00000000 00000000 01009406   00000100 00200000 00940000 0003E3B8  *.,,,,,,,,,,,,,,,,,,,,,,,,,,,,T.*
03E400   0003E4AC 20067E40 0003EE10 00000080   000065A8 000065A8 000065A8 000065A8  *..U.... ......................*
03E420   000065A8 00000000 0000040B 000003E0   0F000000 000188B8 0803EE3C 68000000  *.,,,,,,,,,,,,,,,,,,,,,,,,,,,,,*
03E440   03000000 01000000 FF000000 0F02CC48   0403E410 10002968 00000054 00000054  *.....................U.............*
03E460   00130014 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *.....................................*
03E480   00000000 00000000 0003E750 0103E650   00000010 00000001 800007A8 0005B058  *.,,,,,,,,,X...W.,,,,,,,,,,,,,,,,*
03E4A0   0003E808 00000060 FD000058 00000000   42000000 0003E4AC 00000000 00000000  *..Y.....................U.........*
03E4C0   0003E4E0 000270C0 00000000 00000000   00000000 00000000 2305B53C 40000001  *..U.,,,,,,,,,,,,,,,,,,,,,,,,,...*
03E4E0   3103E4D3 40000005 0803E4E0 00000000   0E000000 00000000 00000000 00000000  *.,UL .....U.,,,,,,,,,,,,,,,,,,,,*
03E500   0103FF88 00000036 40C9C5C6 F2F4F4C9   40E6E3D9 40404040 404BF0F0 C5404040  *........ IEF244I WTR     .00E   *
03E520   40404BC9 C5C6D9C4 C5D94040 E4D5C1C2   D3C540E3 D640C1D3 D3D6C3C1 E3C50000  *  .IEFRDER  UNABLE TO ALLOCATE..*
03E540   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *.....................................*
      LINE 03E560  SAME AS ABOVE
03E580   00000000 00000000 40000058 00000000   0200003B 80000000 42000000 40000000  *.........,,,,,,,,,,,,,,,,,,,,,,,,,,,*
03E5A0   04000000 00063787 0103E500 00000025   40C9C5C5 F2F8F1C9 40F1F5F0 404040D5  *.,,,,,,,,,V.....IEF281I 150   N*
03E5C0   D6E640D6 C6C660D3 C9D5C540 40404040   40404040 40000000 00000000 00000000  *OW OFF.LINE          .,,,,,,,,,,*
03E5E0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *.....................................*
      LINE 03E600  SAME AS ABOVE
03E620   00000000 00000000 00000000 00000000   40000058 00000000 0200003A 80000000  *.....................,,,,,,,,,,,,,,,,*
03E640   50000000 80000000 10000000 00063619   0303FEB0 00000000 C9C7C3F0 C1F0F5C1  *.....................IGC0A05A*
03E660   0105B058 2003E490 00000000 00000000   00040082 0003ECA8 FFF50001 4005AFF0  *.,,,,,U.,,,,,,,,,,,,,,,,,5.. ..,0*
03E680   00000000 0103B660 00000000 00000000   00040082 0003ECA8 FFF50001 4005AFF0  *.....................5.. ..,0*
03E6A0   00000000 0103B740 0003E710 00000068   0003DEB8 00000000 5C03E6D8 0003E758  *.,,,,,,,...X.............WQ..X.*
03E6C0   00040058 0003E748 0003E6D4 0003FF0C   80040298 80040610 0003E6F4 01000000  *.,,,,X..,WM.,,,,,,,,,,,W4....*
03E6E0   0003E6D4 00000000 00000000 00000000   FFFC0062 C9C5C6E2 C4F0F8F0 00040188  *..WM.............IEFSD080.....*
03E700   00000000 00000000 00000000 4005FC3A   0003ED50 00000040 E6E3D940 40404040  *.,,,,,,,,,,,,,,,,,,,,,WTR     *
03E720   F0F0C540 40404040 40404040 40404040   14040140 C9C5C6D9 C4C5D940 00040100  *00E             ... IEFRDER ....*
03E740   80000360 00000000 0003E718 00000030   0003E898 0103FB68 0003D960 00000020  *.,,,,,,,,,X.......Y.......R.....*
03E760   FD000014 0003DB18 8003E76C 40000000   28000000 00070500 00200000 00000000  *.,,,,,,,,,X.,,,,,,,,,,,,,,,,,,,,*
03E780   C9D5C9E3 40404040 C9D5C9E3 40404040   01000000 00000000 0103D6A0 0103D508  *INIT    INIT    ..........O...N.*
03E7A0   0103D238 0103B9C0 0103B7F8 C003DCB0   00000000 00000000 00040082 0003FEB0  *..K.,,,,,,8.,,,,,,,,,,,,,,,,,,*
03E7C0   FFF5000D 4005AFD6 00000000 0003D340   00000000 00000038 80068FEE 00000000  *.5.. ..O.....L ..................*
03E7E0   0003E7F0 00000000 00000000 8003E7F8   00080000 00000000 00100000 0003DB08  *..X0.........X8..................*
03E800   00000000 00000000 0003FE50 00000030   00000000 0003E3C0 0003E3E4 0003FE58  *.,,,,,,,,,,,,,,,,,,,,T...TU....*
```

```
03EF00  C9C6C7F0 F2F0F2D3 01072C00 2803EEE8  0003EF40 0103EF28 00000010 00000001  *IFG0202L.......Y...............*
03EF20  80000400 00073000 B103EF68 00000000  C9C6C7F0 F2F0F2D1 01073000 2803EF18  *................IFG0202J........*
03EF40  0003EF80 0103EF68 00000000 0003EED8  00073000 00000800 00000010 00000001  *...............Q...............*
03EF60  80000400 00073400 B103EF98 00000000  C9C6C7F0 F2F0F2D2 01073400 2803EF58  *................IFG0202K........*
03EF80  0003EFB0 0103EF98 00000010 00000001  80000400 00073800 B103EFD8 00000000  *.............................Q....*
03EFA0  C9C6C7F0 F2F0F2C5 01073800 2803EF88  0003EFF0 0103EFD8 00000000 0003EF48  *IFG0202E..........0...Q........*
03EFC0  00073800 00000800 00000010 00000001  80000400 00073C00 B103F008 00000000  *.............................0....*
03EFE0  C9C7C7F0 F2F0F1E9 01073C00 2803EFC8  0003F020 0103F008 00000010 00000001  *IGG0201Z.......H...0...0.......*
03F000  80000400 00074000 B103F048 00000000  C9C7C7F0 F2F0F1E8 01074000 2003EFF8  *.......  ...0.....IGG02019Y.....8*
03F020  0003F060 0103F048 00000000 0003EFB8  00074000 00000800 00000010 00000001  *...0...0.....  ................*
03F040  80000400 00074400 B103F078 00000000  C9C6C7F0 F2F0F0E8 01074400 2803F038  *..........0.....IFG0200Y.......0.*
03F060  0003F090 0103F078 00000010 00000001  80000400 00074800 B103F0B8 00000000  *...0...0.....................0....*
03F080  C9C6C7F0 F2F0F0E6 01074800 2803F068  0003F0D0 0103F0B8 00000000 0003F028  *IFG0200W.......0...0...0.......0.*
03F0A0  00074800 00000800 00000010 00000001  80000400 00074C00 B103F0E8 00000000  *.............................0Y....*
03F0C0  C9C6C7F0 F2F0F0E5 01074C00 2803F0A8  0003F100 0103F0E8 00000010 00000001  *IFG0200V.......0...1...0Y.....*
03F0E0  80000400 00075000 B103F128 00000000  C9C6C7F0 F1F9F8D5 01075000 2803F0D8  *..........1.....IFG0198N.......0Q*
03F100  0003F140 0103F128 00000000 0003F098  00075000 00000800 00000010 00000001  *...1  ...1.......0.............*
03F120  80000400 00075400 B103F158 00000000  C9C6C7F0 F1F9F6E6 01075400 2803F118  *...........1.....IFG0196W.......1.*
03F140  0003F170 0103F158 00000010 00000001  80000400 00075800 B103F198 00000000  *...1...1.....................1....*
03F160  C9C6C7F0 F1F9F6E5 01075800 2803F148  0003F1B0 0103F198 00000000 0003F108  *IFG0196V.......1...1...1.......1.*
03F180  00075800 00000800 00000010 00000001  80000400 00075C00 B103F1C8 00000000  *.............................1H....*
03F1A0  C9C6C7F0 F1F9F6D4 01075C00 2803F188  0003F1E0 0103F1C8 00000010 00000001  *IFG0196M.......1...1...1H.....*
03F1C0  80000400 00076000 B103F208 00000000  C9C6C7F0 F1F9F6D3 01076000 2803F1B8  *...........2.....IFG0196L.......1.*
03F1E0  0003F220 0103F208 00000000 0003F178  00076000 00000800 00000010 00000001  *...2...2.......1..............*
03F200  80000400 00076400 B103F238 00000000  C9C6C7F0 F1F9F6D1 01076400 2803F1F8  *...........2.....IFG0196J.......18*
03F220  0003F250 0103F238 00000010 00000001  80000400 00076800 B103F278 00000000  *...2...2.....................2....*
03F240  C9C7C7F0 F1F9F6C2 01076800 2803F228  0003F290 0103F278 00000000 0003F1E8  *IGG0196B.......2...2...2.......1Y*
03F260  00076800 00000800 00000010 00000001  80000400 00076C00 B103F2A8 00000000  *.............................2....*
03F280  C9C7C7F0 F1F9F6C1 01076C00 2803F268  0003F2C0 0103F2A8 00000010 00000001  *IGG0196A.......2...2...2.......*
03F2A0  80000400 00077000 B103F2E8 00000000  C9C6C7F0 F1F9F5D1 01077000 2803F298  *..........2Y.....IFG0195J.......2.*
03F2C0  0003F300 0103F2E8 00000000 0003F258  00077000 00000800 00000010 00000001  *...3...2Y.....2...............*
03F2E0  80000400 00077400 B103F318 00000000  C9C6C7F0 F1F9F5C1 01077400 2803F2D8  *..........3.....IFG0195A.......2Q*
03F300  0003F330 0103F318 00000010 00000001  80000400 00077800 B103F358 00000000  *...3...3.....................3....*
03F320  C9C6C7F0 F1F9F4C5 01077800 2803F308  0003F370 0103F358 00000000 0003F2C8  *IFG0194E.......3...3...3.......2H*
03F340  00077800 00000800 00000010 00000001  80000400 00077C00 B103F388 00000000  *.............................3....*
03F360  C9C6C7F0 F1F9F3C1 01077C00 2803F348  0003F3A0 0103F388 00000010 00000001  *IFG0193A.......3...3...3.......*
03F380  80000400 00078000 B103F3C8 00000000  C9C7C7F0 F1F9F1F7 01078000 2803F378  *..........3H.....IGG01917.....3.*
03F3A0  0003F3E0 0103F3C8 00000000 0003F338  00078000 00000800 00000010 00000001  *...3...3H.....3...............*
03F3C0  80000400 00078400 B003F3F8 00000000  C9C7C7F0 F1F9F1F1 01078400 2003F3B8  *..........38....IGG01911.......3.*
03F3E0  0003F410 0103F3F8 00000010 00000001  80000400 00078800 B003F438 00000000  *...4...38.....................4....*
03F400  C9C7C7F0 F1F9F1F0 01078800 2003F3E8  0003F450 0103F438 00000000 0003F3A8  *IGG01910.......3Y...4...4.......3.*
03F420  00078800 00000800 00000010 00000001  80000400 00078C00 B003F468 00000000  *.............................4....*
03F440  C9C7C7F0 F1F9F1C4 01078C00 2003F428  0003F480 0103F468 00000010 00000001  *IGG0191D.......4...4...4.......*
03F460  80000400 00079000 B003F4A8 00000000  C9C7C7F0 F1F9F1C2 01079000 2003F458  *..........4.....IGG0191B.......4.*
03F480  0003F4C0 0103F4A8 00000000 0003F418  00079000 00000800 00000010 00000001  *...4...4.....4................*
03F4A0  80000400 00079400 B003F4D8 00000000  C9C7C7F0 F1F9F1C1 01079400 2003F498  *..........4Q.....IGG0191A.......4.*
03F4C0  0003F4F0 0103F4D8 00000010 00000001  80000400 00079800 B103F518 00000000  *...40...4Q....................5....*
03F4E0  C9C7C3F0 F0F0F5C5 01079800 2003F4C8  0003F530 0103F518 00000000 0003F488  *IGC0005E.......4H...5...5.......4.*
03F500  00079800 00000800 00000010 00000001  80000400 00079C00 B103F548 00000000  *.............................5....*
03F520  C9C7C3F0 F0F0F2C0 01079C00 2803F508  0003F560 0103F548 00000010 00000001  *IGC0002.......5...5...5.......*
03F540  800002A8 0007A278 B103F588 00000000  C9C7C3F0 F0F0F2C2 0107A278 2803F538  *..........5.....IGC0002B.......5.*
03F560  0003F5A0 0103F588 00C7A000 0003F4F8  0007A000 00000800 00000010 00000001  *...5...5.......48.............*
03F580  800002E0 0007A520 B1000000 00000000  C9C7C3F0 F0F0F1C9 0107A520 2803F578  *..............IGC0001I.......5.*
03F5A0  0003F5D0 0103F5B8 00000010 00000001  800001D8 0007A918 B103F5E8 00000000  *...5...5.....................5Y....*
03F5C0  C9C7C7F0 F1F9C5D2 0107A918 2003F5A8  0003F600 0103F5E8 00000010 00000001  *IGG019EK.......5...6...5Y.....*
```

```
                                                                                                                                PAGE 0109
0403A0    0003E8D8 00000018 00041600 0003EC4C    00000000 00000419 00069800 00000000    *..YQ.....................................*
0403C0    00069800 00002000 00000010 00000001    80001C80 00069B80 00040468 00000090    *.....................C.....S*
0403E0    000065A8 000065A8 000065A8 0007C3B0    000065A8 00000000 00000409 000023E2    *.....A.....................C.....S*
040400    11000000 0403C170 10040494 88000000    4F000000 01FFFFFF FF000000 0F0716B8    *.....A.....................BBBACJCD.....*
040420    040403E0 18002928 00000C44 00010045    00000014 00010001 00000000 00000000    *....BBBACJCD.....A.....................C.*
040440    00000000 C2C2C2C1 C3D1C3C4 00000000    00000000 00000000 00000000 00000000    *.....A.....S.....A.....................*
040460    00000000 0001B4E0 0003C148 00000090    000065A8 000065A8 000065A8 0007C3B0    *.....BBBACJCD.....BBBACJCD.....*
040480    000065A8 00000000 00000408 000023E2    11000000 0403C170 10000000 A8000000    *.....A.....S.....A.....................*
0404A0    4F000000 01000000 FF000000 0F071728    04040470 18002928 00000043 00010044    *.....BBBACJCD.....BBBACJCD.....*
0404C0    00000014 00010001 00000000 00000000    00000000 C2C2C2C1 C3D1C3C4 00000000    *.....BBBACJCD.....Y...0.....*
0404E0    00000000 00000000 00000000 00000000    00000000 40021614 0B0405E8 000401F0    *IEFSD079.....H.....*
040500    C9C5C6E2 C4F0F7F9 0106AE60 200403C8    00040540 00000030 000403B0 000403B8    *...Q.....IEERGN.....*
040520    0003FED8 0006DD90 0004052C 00100000    00000000 C030C9C5 C5D9C7D5 00000000    *.....*
040540    00000000 00000030 00000000 00000000    00000000 00040588 00000000 00000000    *.....*
040560    00000000 00000000 00000000 0003EBE0    00040578 0203FB68 000405A8 0203FA98    *.....*
040580    000405B8 0103F9A8 00000000 00000000    00000000 00000000 00000000 0006EF28    *.....9.....*
0405A0    0006F388 00040650 000405B0 0203FA68    00000000 0203FA38 000405C0 0103F938    *..3.....9.*
0405C0    000405E0 0103FA08 00000010 00000001    80000278 0006E080 60000000 00040600    *.....*
0405E0    00000000 0103FBA8 39000000 00000000    C9C5C6E2 C4F0F7F8 0006E080 680405C8    *.....IEFSD078.....H*
040600    0006D800 00000000 0006D800 00000800    00000000 00000000 00040083 0003FD98    *..Q.....Q.....*
040620    FF050001 5007D632 00000000 01040058    0006B800 0006B800 00000000 00000000    *.....0.....*
040640    00040058 00005000 00069800 00000000    00000000 00000000 00040002 00040690    *.....*
040660    FF000001 5006EABC 00000000 01018970    0006E800 00000000 0006E800 00000800    *.....Y.....Y.....*
040680    00000010 00000001 800005F8 0006EA08    0B03EC90 00040650 C9C5C5E5 E6C1C9E3    *.....8.....IEEVWAIT*
0406A0    0106EA08 28040680 000406B0 FC0406C0    C0000000 000405D8 000406A8 FB0403B8    *.....Q.....*
0406C0    0006E2F8 00000000 0006E000 00000800    00040630 00040630 40000000 FC040760    *..S8.....*
0406E0    0007D800 0003FD78 0007D800 00000800    00000010 00000001 800000A0 0007DF60    *..Q.....Q.....*
040700    BF040748 0003FE48 C9C5C5D9 C7D54040    0007DF60 18040718 B9040700 00000000    *.....IEERGN .....*
040720    C9C5C5D7 D9E3D540 01C7DF60 200406F0    00000000 01040748 00000010 00000001    *IEEPRTN .....0.....*
040740    80000078 0007E088 B9000000 00000000    C9C5C5D7 D9E6C9F2 0107E088 28040738    *.....IEEPRWI2.....*
040760    0007E100 000406E0 0007E000 00000800    00000000 00000000 00000000 00000000    *.....*
040780    00000000 00000000 00000000 00000000    00000000 00000000 00000000 00000000    *.....*
      LINES 0407A0-0407E0 SAME AS ABOVE


REGS AT ENTRY TO ABEND

     FLTR 0-6      000000000003D478      FE0000880003BBF8              0004300440073402      0405A5D007027568

     REGS 0-7      00000030   80000064   0003E368   00000001           0003D968   0003D770   0003D8BC   0003E1D8
     REGS 8-15     0003DCD0   0003BD00   0003DCF8   00000000           4007D482   0105AD50   0001D4CA   0000000C

LOAD MODULE    GO

05AD40                                               90ECD00C 18DF4500 D012E2E4 C2404040    *.....K.....SUB   *
05AD60    40404110 D2860A29 12FF4780 D02E0700    47F0D028 80000065 5810D024 0A0D41F0    *  ..K.....0.....0*
05AD80    D03847F0 F0243088 0005ADA4 00000000    00000000 00000000 00000000 02000000    *...00.....*
05ADA0    00000000 E2E4C240 40404040 0A2A41F0    D06847F0 F024E7E7 0005ADD4 00000000    *....SUB      ...0...00.XX...M....*
05ADC0    00000000 00000000 00000000 02000000    00000000 E2E4C240 40404040 0A2A0700    *.....SUB     ....*
05ADE0    4510D098 8F05AE94 0A134130 00010700    4510D134 0100B7BE 0005AE94 00000000    *.....J.....*
05AE00    0005AE04 0005AF30 0005AF50 0005AF38    0005AF58 0005AF40 0005AF60 0005AF48    *.....*
05AE20    0005AF68 0005AF30 0005AFA0 0005AF30    0005AFA8 0005AF38 0005AFA0 0005AF38    *.....*
05AE40    0005AFA8 0005AF30 0005AF48 0005AF38    0005AF50 0005AF40 0005AF58 0005AF48    *.....*
05AE60    0005AF60 0005AF30 0005AF40 0005AF38    0005AF48 0005AF40 0005AF50 0005AF48    *.....*
05AE80    8005AF58 42310000 9280108C 0A3347F0    D2784780 00000000 00000000 00000000    *.....0K.....*
```

```
                                                                                                                                PAGE 0110
05AEA0   00000000 00000000 00000001 00004000   00000001 04000001 54000000 002C0020   *.............. ................*
05AEC0   0003D678 9207BBF6 0007BBDE 00000001   00000000 00000000 0007BBE0 00000001   *..O....6.....................*
05AEE0   00000001 0000007D 0007BBB0 C010D503   40404040 40404040 40404040 40404040   *.............N.               *
05AF00   40404040 40404040 40404040 404040E2   E3D6D9C1 C7C540E3 C5E2E340 C1D9C5C1   *                STORAGE TEST AREA*
05AF20   40404040 40404040 40404040 40404040   11111111 11111111 22222222 22222222   *                ................*
05AF40   33333333 33333333 44444444 44444444   55555555 55555555 66666666 66666666   *................................*
05AF60   77777777 77777777 88888888 88888888   99999999 99999999 AAAAAAAA AAAAAAAA   *................................*
05AF80   BBBBBBBB BBBBBBBB CCCCCCCC CCCCCCCC   DDDDDDDD DDDDDDDD EEEEEEEE EEEEEEEE   *................................*
05AFA0   FFFFFFFF FFFFFFFF 40404040 40404040   40404040 40404040 40404040 40404040   *........                       *
05AFC0   40404040 40404040 47F0D280 80000064   5810D27C 0A0D4100 00084510 F0080A0A   *        .0K.....K........0....*
05AFE0   D7071000 10004110 10004100 00010A01   47F0F022 80000066 5810F01E 0A0D1020   *P..............00.......0.....*

LOAD MODULE    IGC0A05A

05B840                                                             4180D099 1B114313   *87I DIRECT SYSOUT.NY............*
05B860   00001A81 41330001 95FF3000 47806068   1BEE1BFF 1B001B11 43E30000 43030001   *....................T.....*
05B880   8CE00004 88F0001C 8C000004 8810001C   1A2044E0 60701A1E 41818001 44F06076   *.....0.....................0..*
05B8A0   F384D069 D069DC07 D06962C6 41FFF001   44F0607C 47F066F8 413E3003 47F06010   *3...........F..0..0...0.8.....0..*
05B8C0   41330001 47F060E2 D2008000 3002D200   D0692000 D2008000 D0695050 D08C5000   *.....0.SK.....K.....K..........*
05B8E0   D12094FC D1235B00 D1201A10 5800D120   41110003 5010D064 94FCD067 D703D06C   *J...J...J.....J..........P....*
05B900   D06C1810 54006290 19014780 60BC1B10   4010D06C 5810D064 4A10D06C 1B005D00   *.....................Y.....J..5*
05B920   66804000 D06A1211 4770611C 4810D06A   12114770 60E85850 D08C5860 D12407F5   *..J.......J....D................*
05B940   4120D121 45B06236 5820D120 413062C4   48A0D05A 4BA0D06C 88A00002 45B0623E   *..J.....J....D.................*
05B960   46A06104 9640D112 455062DA 94BFD112   47F060DE 18A15810 D1204B10 D06C5010   *..... J.....J..0.....J.....*
05B980   D0704120 D07145B0 62364810 D06C8810   00011A31 5820D120 45B0623E 5020D120   *.....................J.....J.*
05B9A0   9640D112 455062DA 94BFD112 4810D06C   12114770 61984700 615E4110 0001191A   *. J.....J.....................*
05B9C0   47B061A6 18125B10 6680D51F 10002000   477061A6 4810D06E 41110001 4010D06E   *.........N....................*
05B9E0   41220020 5020D120 46A0615E 47F061B0   1B114010 D06C46A0 611E47F0 60D44810   *.....J.....0.....0.M...*
05BA00   D06E1211 4780619E 5810D120 4800D06E   89000005 1B105010 D0704120 D0714810   *..........J....................*
05BA20   D06E0610 12114770 6202D203 D09F629D   41306294 45B0623E D20DD0AA 62A29640   *.........K.............K.....*
05BA40   D1124550 62DA94BF D112D701 D06ED06E   12AA4780 60D447F0 619ED204 D09F629D   *J.....J.P.........M.0..K.....*
05BA60   41306297 45B0623E 9260D0AB 5810D120   5B106680 5010D070 4120D071 4130629A   *.............J................*
05BA80   45B0623E D20DD0B2 62A247F0 61E64130   62B047F0 65744180 D09995FF 30004780   *....K.....0.W....0............*
05BAA0   627C1B00 43030000 1B114313 00011A80   44106284 F384D070 D070DC07 D07062C6   *.....................3.........F*
05BAC0   41111001 4410628A 41330002 41220004   47F0623E 41330001 07FBE2E2 D200D070   *..............0........SSK...*
05BAE0   2000D200 8000D070 FFFFFFE0 0B02FF0C   02FF1302 FFD3C9D5 C5E240E2 C1D4C540   *..K.............LINES SAME *
05BB00   C1E240C1 C2D6E5C5 0002FF09 0312031B   03240330 03390342 034B03FF 0903FF12   *AS ABOVE.....................*
05BB20   03FF1B03 FF2403FF 3003FF39 03FF4203   FF0098E0 D08012EE 47806310 D27CF000   *.....................K.0.*
05BB40   D09441FF 007D50F0 D084411F 007D1910   47D06332 1BFE40F0 D090D203 E000D090   *.......0.............. 0..K.....*
05BB60   41FE0004 50F0D084 47F06722 92201005   58F10008 58F0F030 05EF5817 000058E0   *....0...0.......1...00........*
05BB80   100858F0 E03405EF 0A030700 47F06770   9560D098 47406348 47206346 1A011A01   *...0.........0.....  .........*
05BBA0   1A019240 D098D277 D099D098 41100038   190147B0 636C4000 D05C9140 D1120715   *.....K.....................J...*
05BBC0   47F060E2 4810D05E 41110001 4010D05E   92F1D098 D203D105 63AE4E10 D078F333   *.0.S.........1..K.J.......3.*
05BBE0   D10AD07C 96F0D10D D201D05C 63B247F0   62DA98E0 D03012EE 0785411E 0004191F   *J....0J.K.....0...............*
05BC00   078547F0 62FCD7C1 C7C5FFFF 07FEF0F1   F2F3F4F5 F6F7F8F9 C1C2C3C4 C5C69120   *...0..PAGE....0123456789ABCDEF..*
05BC20   D1384710 64C69101 D1384710 668C9180   D13947F0 6616910C D1384750 64D29120   *J....F..J.....J..0....J....K..*
05BC40   D1394710 64D29118 D1394750 64E29110   D1384710 66C69102 D1394710 64EE58E4   *J....K..J....S..J....F..J.....U*
05BC60   007C13EE 45506504 47F06418 47F06434   947FE020 58100010 58B010C8 58BB0028   *.........0...0.............H....*
05BC80   188E18AE 47F06694 4700640C 94FE401D   D20BD098 652CD203 D124D128 455062DA   *.....0.........K....K.J.J.....*
05BCA0   4550639A 9879D080 12774780 646A1B97   410000FA 89000018 16091817 41110000   *.....................P.....0..*
05BCC0   0A0A47F0 67589180 E0024780 64A24110   D070D70B 10001000 41E0656C 58F0D060   *..0.............0.....J.....*
05BCE0   41FF0004 41300004 92FF1000 42301001   50E01004 50F01008 0A3058A0 D1305800   *.....................J.....*
05BD00   6568181D 41110000 0A0A4100 6560D205   A06064C0 47FA0060 0A091BFF 0A0394DF   *.............K................*
05BD20   D13841A0 653847F0 64FA94F3 D13894DF   D13941A0 654847F0 64FA94E7 D13941A0   *J.....0...3J...J.....0...XJ...*
05BD40   655047F0 64FA94FD D13994EF D13841A0   655841F0 D07050A0 F0000A07 12EE47A0   *...0....J...J.....0....0......*
```

```
05B420   D1384710 64C69101 D1384710 668C9180   D13947F0 6616910C D1384750 64D29120   *J....F..J.......J..0....J....K..*
05B440   D1394710 64D29118 D1394750 64E29110   D1384710 66C69102 D1394710 64EE58E4   *J....K..J....S..J....F..J.....U*
05B460   007C13EE 45506504 47F06418 47F06434   947FE020 58100010 58B010C8 58BB0028   *.........0...0............H....*
05B480   188E18AE 47F06694 4700640C 94FE401D   D20BD098 652CD203 D124D128 455062DA   *.....0........ .K.....K.J.J.....*
05B4A0   4550639A 9879D080 12774780 646A1B97   410000FA 89000018 16091817 41110000   *.............................*
05B4C0   0A0A47F0 67589180 E0024780 64A24110   D070D70B 10001000 41E0656C 58F0D060   *...0.............P..........0..*
05B4E0   41FF0004 41300004 92FF1000 42301001   50E01004 50F01008 0A3058A0 D1305800   *.............................*
05B500   6568181D 41110000 0A0A4100 6560D205   A06064C0 47FA0060 0A091BFF 0A0394DF   *.............K..............*
05B520   D13841A0 653847F0 64FA94F3 D13894DF   D13941A0 654847F0 64FA94E7 D13941A0   *J.......0...3J...J.....0...XJ...*
05B540   655047F0 64FA94FD D13994EF D13841A0   655841F0 D07050A0 F0000A07 12EE47A0   *.....0.....J...J.....0.....0.....*
05B560   650E13EE 07F518FE 58EF0088 12EE0775   58EF0080 59FF007C 47850004 58FF0084   *.....5.....................*
05B580   47F06514 F0C5D5C4 40D6C640 C4E4D4D7   C9C7C3F0 F1F0F5C1 C9C7C3F0 F4F0F5C1   *.0..0END OF DUMPIGC0105AIGC0405A*
05B5A0   C9C7C3F0 F6F0F5C1 C9C7C3F0 F7F0F5C1   C9C7C3F0 F8F0F5C1 C9C7C3F0 C1F0F5C1   *IGC0605AIGC0705AIGC0805AIGC0A05A*
05B5C0   FD00013D E2E8E2C9 C5C1F0F1 4180D099   907AD09C 187858A0 D12088A0 000589A0   *....SYSIEA01.....J.........*
05B5E0   0005D21F 7057A000 4187001F 4197001F   41800001 925C7056 925C7077 91C07057   *..K.....................M...*
05B600   471065B8 95407057 478065BC 920F7057   943F7057 877865A4 DC1F7037 65D4987A   *.....0...ABCDEFGHI.....JKL*
05B620   D09CD213 D09CD09B 47F0623E 4BC1C2C3   C4C5C6C7 C8C94B4B 4B4B4B4B 4BD1D2D3   *MNOPQR.....STUVWXYZ.....0123*
05B640   D4D5D6D7 D8D94B4B 4B4B4B4B 4B4BE2E3   E4E5E6E7 E8E94B4B 4B4B4B4B F0F1F2F3   *456789.....J.....J...*
05B660   F4F5F6F7 F8F94B4B 4B4B4B4B 40004710   668C9180 D13A4780 6636947F D13A41A0   *...0..IGC0D05A. J.....J.....0*
05B680   662E47F0 64FAC9C7 C3F0C4F0 F5C19140   D13A4780 665294BF D13A41A0 664A47F0   *..IGC0H05A.....J.....*
05B6A0   64FAC9C7 C3F0C8F0 F5C158A0 00109104   A07447E0 63DE9110 D1384710 66A49108   *J.....7J...J.....0.....IGC0*
05B6C0   D13847E0 63DE94F7 D13894DF D13941A0   668447F0 64FA9060 00000020 C9C7C3F0   *B05A.....0.....Y.0....J.*
05B6E0   C2F0F5C1 41A06540 47F064FA 800067A0   05EB8000 67A118E8 47F0640C 94EFD138   *.....0.....0.....*
05B700   58A00010 91C0A0EC 47E066F0 9108A0EC   47E06666 41A06788 1B8847F0 64FA58A0   *....J.....Y.............0..*
05B720   001094EF D13891C0 A0EC47E0 66E89108   A0EC47E0 63FE1B88 41A05780 47F064FA   *.....0.....0...J.....0..*
05B740   41A06790 47F064FA 41A06798 47F064FA   9180D13B 47106708 418F8004 47F06060   *..J.K.J.J...J..J.....0.....*
05B760   9200D13C D200D13D D13B9407 D13D4A80   D13C418F 8001 47F0 6060507D 0008581D   *..M.....K.....*
05B780   0000D403 D080D080 47706742 4171005C   5071000C D2817000 D090187D 58DD0004   *.....P.....0.....*
05B7A0   41F06314 0A0C18D7 587D0008 47F06332   581D0000 5800677C 18114111 00000A0A   *.0.....,0.....IGC0J05A*
05B7C0   58E0D060 47F0646E 41100001 4800D05C   47F06338 FA0000E0 C9C7C3F0 D1F0F5C1   *IGC0K05AIGC0N05AIGC0P05A.....*
05B7E0   C9C7C3F0 D2F0F5C1 C9C7C3F0 D5F0F5C1   C9C7C3F0 D7F0F5C1 00FFB00C 12AA4780   *
```

LOAD MODULE IGG019CD

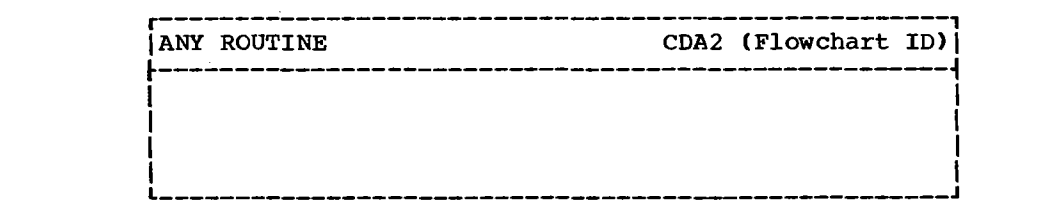
```
07CC80                                         909AD040 9120203C 4710F194 58A0200C   *.....1.....*
07CCA0   5810202C 48673006 9101203C 4780F024   4860205A 12664740 F09C1B44 4340A007   *.....0.....0.....*
07CCC0   18049108 A00947E0 F03E4800 A0064340   20041B04 91382024 47E0F054 91403000   *.....0..... .....0.. ..*
07CCE0   4710F05A 1E0647F0 F08C1B44 43401010   89400004 1A414890 402C1299 4720F08A   *..0....00.....  ..... .....0.*
07CD00   4740F09C 41120000 11110A19 4890402C   129947D0 F09C5810 202C1E09 49002012   *. 0.....  .....0.....*
07CD20   47C0F1B6 4900A004 47C0F0AE 4810F0AC   41000080 89000018 16100A0D 20001882   *..1.....0...0.............*
07CD40   41208005 42730004 183F184E 185B186C   187D58F0 001058F0 F02005EF 18905E00   *.....0..00.....*
07CD60   32685E90 32645590 80004770 30E65000   80009201 80021880 58F00010 58F0F01C   *.....W.....0...00.*
07CD80   05EF12FF 47803100 96202037 47F0315A   95002000 4770315A 91041008 47E0315A   *.....0.....*
07CDA0   1B991BAA 1BBB1BCC 43920000 89900004   43A91029 43B9102D 43C20006 19CA4740   *.....B.....*
07CDC0   313819BC 47B0315A 18081BBA 58902007   43A09003 1BAB46A0 314A89A0 00101E0A   *.....*
07CDE0   58F00010 58F0F01C 05EF18F3 41900005   1B295880 20305830 204458A0 200C18D7   *.0...00....3.....P*
07CE00   1B774373 000418B5 18C618E4 48673006   9101203C 4780F18C 4860205A 9120203C   *.....F.U.....1.....*
07CE20   4780F1A8 9620203C 96013014 5843000C   92404000 47F0F25E 58A0200C D2012012   *..1.....  ..02.....K...*
07CE40   A0049200 200C1846 4850A00A 1C448E40   00094340 A0069108 A00947E0 F1D24840   *.....1K. *
07CE60   A0061E54 1B444340 20041F54 48402012   1B454040 2012D207 30282005 1B444340   *.....  ..K..... *
07CE80   200C4144 00014240 200C9140 30004710   F20A4243 002F47F0 F2564340 20401853   *.....2.....02.. ...*
07CEA0   1A544340 20418940 00031A54 D2045000   20084B50 F25C9180 20344780 F2324060   *.....K.....2.....2. .*
07CEC0   50001B44 43402010 1B644065 00081266   4770F256 48402012 06401244 4720F252   *.....  .....2.. ... ....2.*
07CEE0   1B444040 20124143 00081814 0A00989A   D04007FE 00000100 00010000 00020700   *.. ..... ..... .....*
```

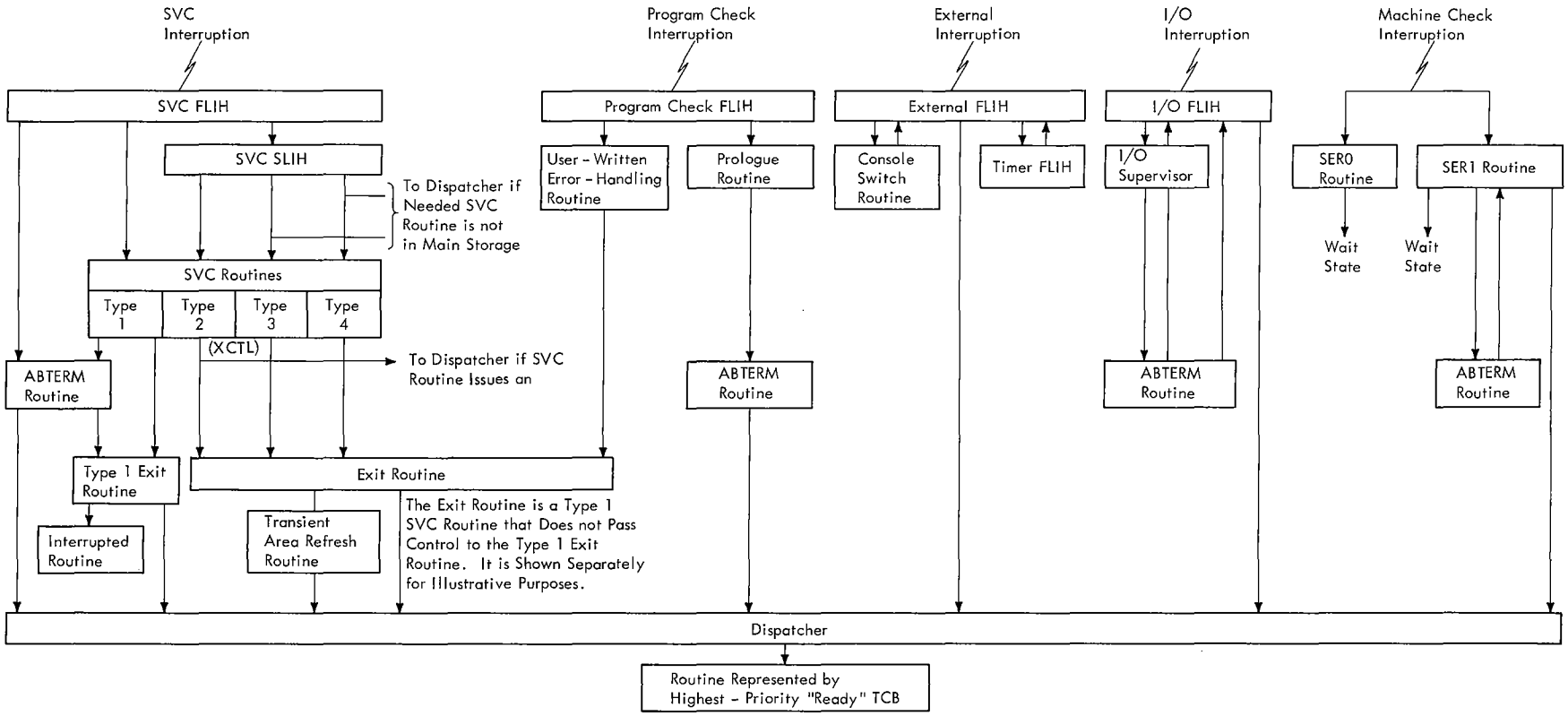
LOAD MODULE IGG019CJ

```
07BC00   1BFF07FE 91011023 4710F03E 58F01020   41FF0000 12FF4780 F03E07FF 41100337   *.........0..0.........0.......*
07BC20   8910000C 41000080 89000018 16100A0D                                       *....................04....*
SP 252

0697E0                                                          00000000 00000000   *333333                 ..........*
SP 000

068D20                                                          7F000000 01200000   *.....................................*
068D40   8F068E70 000681A0 00068E20 00000000   00000000 00000000 00009400 00000000   *.....................................*
068D60   4005BB88 0007C828 00068800 00068D38   00069800 0005BB1C 0003D340 9005B9A8   * .....H.....................L .....*
068D80   0005B858 0003B468 0003B501 00000000   00000001 00000000 00000000 00000000   *.....................................*
068DA0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....................................*
      LINE 068DC0 SAME AS ABOVE
068DE0   00000000 00000000 00000000 000B0103   E2D5C1D7 D7C5D940 00000000 00000000   *..............SNAPPER ........*
068E00   00000000 00000000 000B0200 000A1800   00000000 00000000 41068E18 7F06065D   *.....................................*
068E20   00000000 7F068D38 00068E68 0C000000   40068E48 00068E70 00000000 00000000   *.....................................*
068E40   0F000000 1A000501 31068E43 40000005   08068E48 00000660 1D068E68 A0000008   *.....................................*
068E60   000681A0 0000065D 001A0005 0200065D   00000100 140F0000 001A0005 020219C8   *.....................................H*
068E80   00210057 00000001 00004000 00000001   04000001 54000000 00400020 0003B324   *.....................................*
068EA0   9207C208 0007C828 0B000001 00000660   30040048 41068E18 0107CC90 0007CC90   *..B...H.....................*
068EC0   0000007D 00000001 8003E668 00000000   00000000 00000000 00000000 00000000   *.........W...........................*
068EE0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....................................*
068F00   00000000 00000000 00000000 00000000   00000000 00000000 8003E688 00000000   *.....................W.....*
068F20   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000   *.....................................*
      LINE 068F40 SAME AS ABOVE
068F60   00000000 00000000 00000000 00000000   00000000 0001D4CA 0105AD50 FD000008   *.....................M.........*
068F80   00068FF8 0003E368 5C03DCF8 0003D968   0003D770 0003D8BC 0003E1D8 0003DCD0   *...8..T....8..R...P...Q....Q....*
068FA0   0003BD00 0003DCF8 00000000 4007D482   00000000 00000000 00000000 00000000   *.......8.... .M.............*
068FC0   00000000 0000C7D6 E2C5E340 40404040   40404040 40404040 40404040 40404040   *......GOSET                          *
068FE0   40404040 40404040 40404040 40404040   40404040 40404040 80068FFE 00000000   *                                .......*

0681A0   065D0000 007D0000 40F0F6F8 C5C1F040   4040F9F2 F0F7C3F2 F0F840F0 F0F0F7C3   *........ 068EA0   9207C208 0007C*
0681C0   F8F2F840 F0C2F0F0 F0F0F0F1 40F0F0F0   F0F0F6F6 F0404040 40F3F0F0 F4F0F0F4   *828 0B000001 00000660    3004004*
0681E0   F840F4F1 F0F6F8C5 F1F840F0 F1F0F7C3   C3F9F040 F0F0F0F7 C3C3F9F0 4040405C   *8 41068E18 0107CC90 0007CC90   .*
068200   F840F4F1 F0F6F8C5 F1F840F0 F1F0F7C3   C3F9F040 F0F0F0F7 C3C3F9F0 4040404B   *8 41068E18 0107CC90 0007CC90   .*
068220   5C007D00 0040F0F6 F8F2F0F0 404040C6   F8F4F0C6 F4C6F140 C6F0C6F6 C6F8C3F5   *..... 068200   F840F4F1 F0F6F8C5*
068240   40C6F1C6 F8F4F0C6 F040C6F1 C6F0C6F7   C3F34040 4040C3F3 C6F9C6F0 F4F040C6   * F1F840F0 F1F0F7C3    C3F9F040 F*
068260   F0C6F0C6 F0C6F740 C3F3C3F3 C6F9C6F0   40F4F0F4 F0F4F0F4 C2404040 5CF840F4   *0F0F0F7 C3C3F9F0 4040404B   .8 4*
068280   F1F0F6F8 C5F1F840 F0F1F0F7 C3C3F9F0   40F0F0F0 F7C3C3F9 F0404040 4B5C007D   *1068E18 0107CC90 0007CC90   ....*
0682A0   000040F0 F6F8F2F2 F0404040 F5C3F0F0   F7C4F0F0 40F0F0F4 F0C6F0C6 F640C6F8   *.. 068220   5C007D00 0040F0F6 F8*
0682C0   C6F2C6F0 C6F040F4 F0F4F0F4 F0C3F640   404040C6 F8C6F4C6 F0C3F640 C6F4C3F6   *F2F0F0 404040C6    F8F4F0C6 F4C6*
0682E0   C6F1F4F0 40C3F6C6 F0C3F6C6 F640C3F6   C6F8C3F3 C6F54040 405C4B4B 4B4B4B40   *F140 C6F0C6F6 C6F8C3F5   ..... *
068300   F0F6F8F2 F0F04040 40C6F8F4 F0C6F4C6   F140C6F0 C6F6C6F8 C3F55C00 7D000040   *068200   F840F4F1 F0F6F8C5.... *
068320   F0F6F8F2 F4F04040 40F4F0C3 F6C6F1C3   F640C6F8 C6F4C6F0 C3F640C6 F0F4F0C3   *068240   40C6F1C6 F8F4F0C6 F040C*
068340   F6C6F140 C3F6C6F0 C3F6C6F7 40404040   C3F3C6F3 F4F0F4F0 40F4F0F4 F0C3F3C6   *6F1 C6F0C6F7    C3F34040 4040C3F*
068360   F340C3F6 C6F9C3F6 C6F040C6 F4C6F0F4   F0C3F640 40405C40 C6F1C6F8 F4F0C6F0   *3 C6F9C6F0 F4F040C6   . F1F840F0*
068380   40C6F1C6 F0C6F7C3 F3404040 40C3F3C6   F9C6F0F4 F040C65C 007D0000 40F0F6F8   * F1F0F7C3    C3F9F040 F.....068*
0683A0   F2F6F040 4040C6F0 C3F6C6F0 C3F640C6   F0C3F6C6 F7F4F040 C3F3C6F3 C3F3C6F3   *260   F0C6F0C6 F0C6F740 C3F3C3F3*
0683C0   40C3F6C6 F9C3F6C6 F0404040 40F4F0C6   F4C6F0C6 F440C6F0 C6F4C6F0 C6F440C3   * C6F9C6F0    40F4F0F4 F0F4F0F4 C*
0683E0   F2F4F0F4 F0F4F040 F5C3C6F8 F4F0C6F4   4040405C F0C6F0C6 F0C6F740 C3F3C3F3   *2404040 5CF840F4   .0F0F0F7 C3C3*
068400   C6F9C6F0 40F4F0F4 F0F4F0F4 C2404040   4BF840F4 5C007D00 0040F0F6 F8F2F8F0   *F9F0 4040404B   .8 4..... 068280*
068420   404040C6 F1C6F0C6 F6C6F840 C3F5C6F1   C6F8F4F0 40C6F0C6 F1C6F0C6 F740C3F3   *   F1F0F6F8 C5F1F840 F0F1F0F7 C3*
068440   C6F5C3F6 C6F14040 4040C3F6 C6F8C6F4   C6F040F4 F0C3F6C6 F0C3F640 C6F1C3F6   *F5C6F1    C6F8F4F0 40C6F0C6 F1C6*
```

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.





Type 1 SVC Routines	Type 2 SVC Routines	Type 3 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

Type 4 SVC Routines
ABDUMP ABEND Checkpoint Comm. Task Router Log and Writelog Post Restart

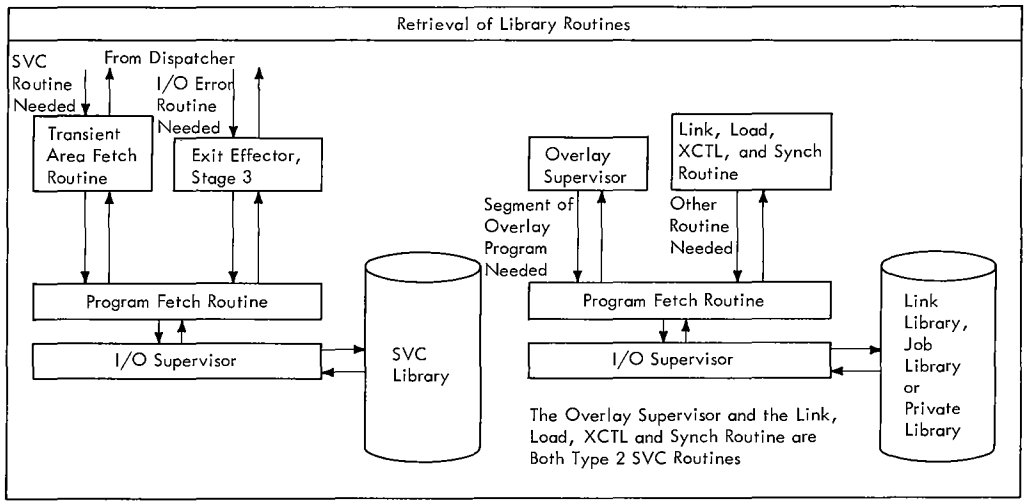


Chart AA. SVC First-Level Interruption Handler

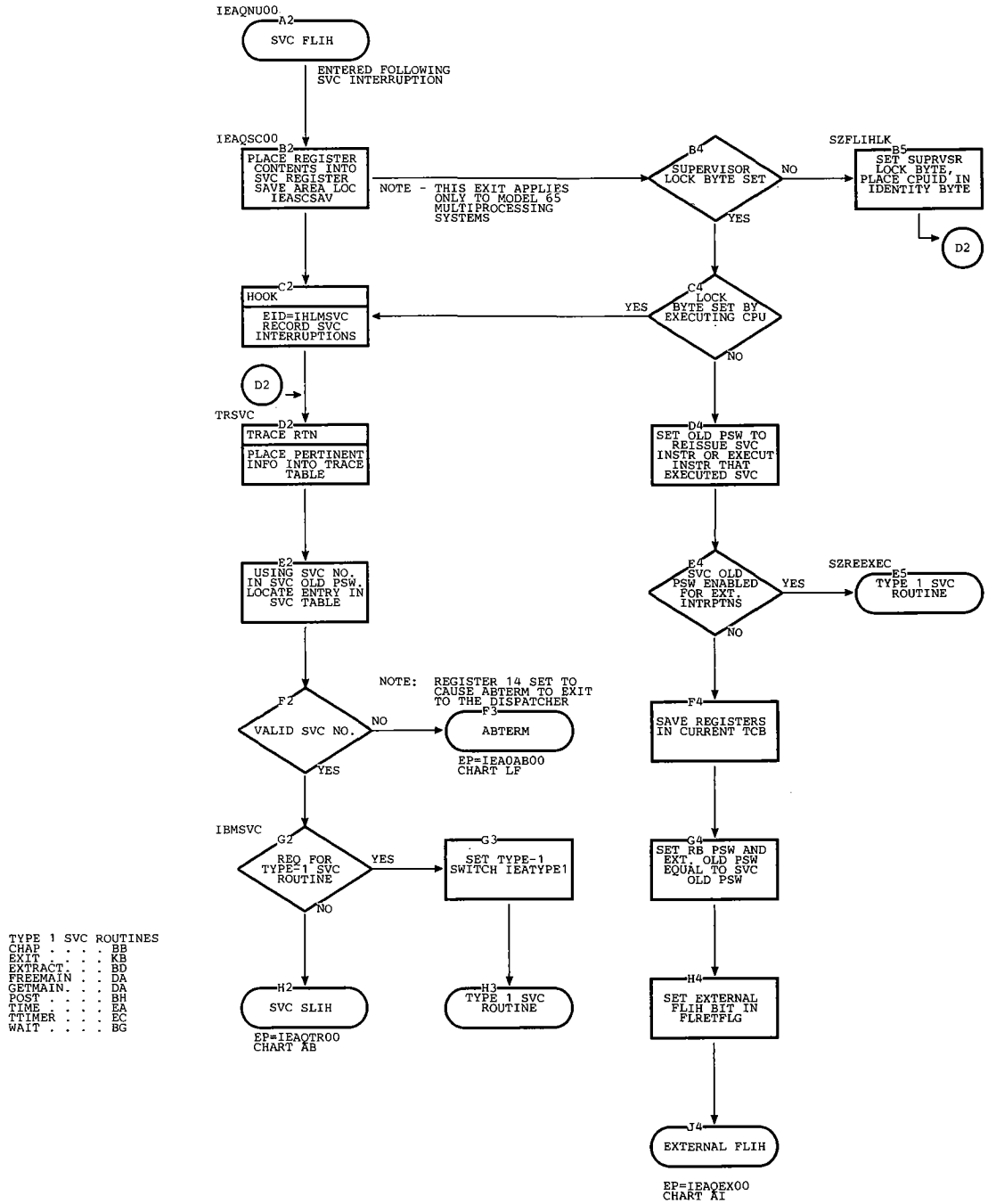


Chart AB. SVC Second-Level Interruption Handler

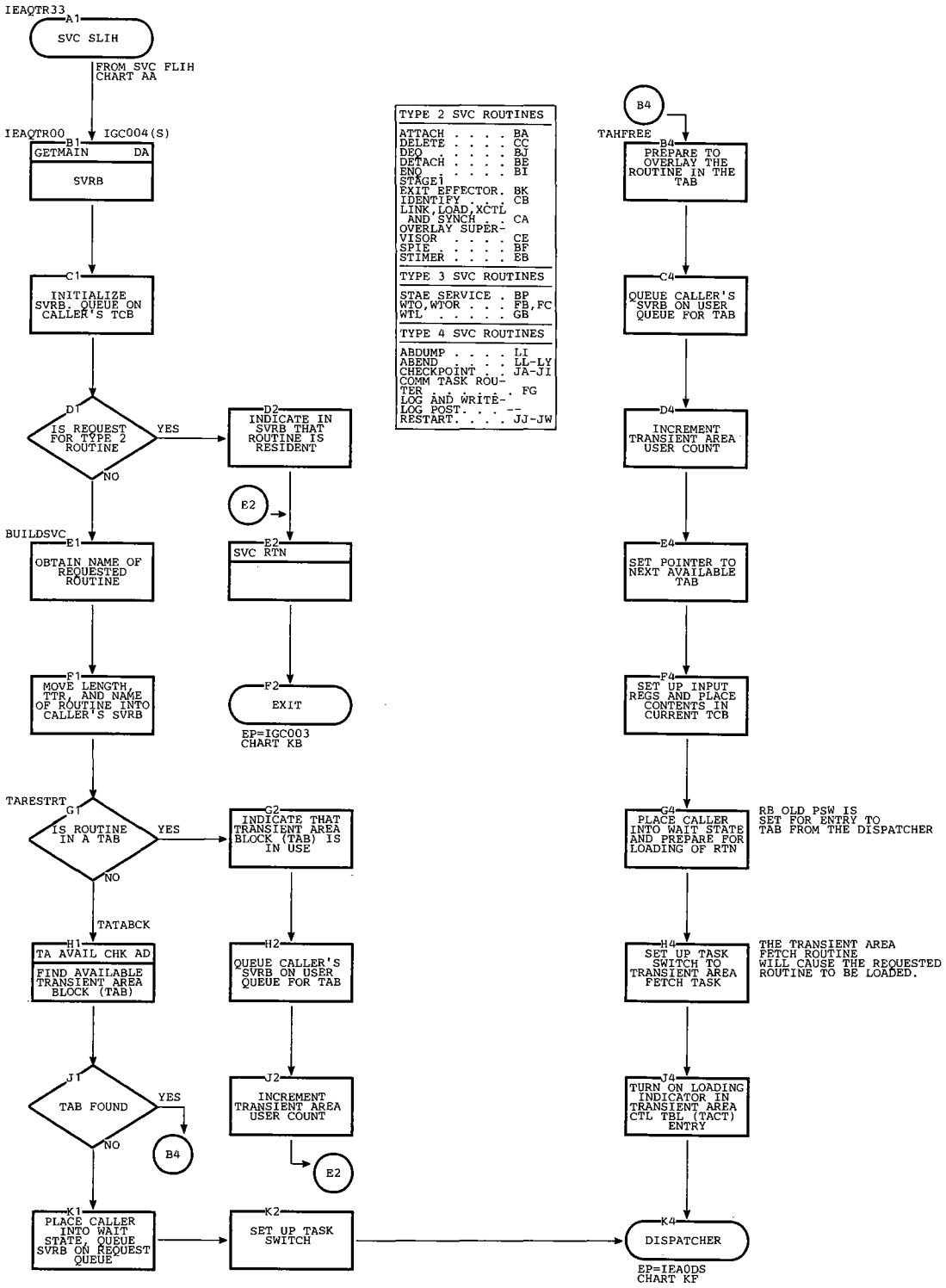


Chart AC. Extended SVC Router

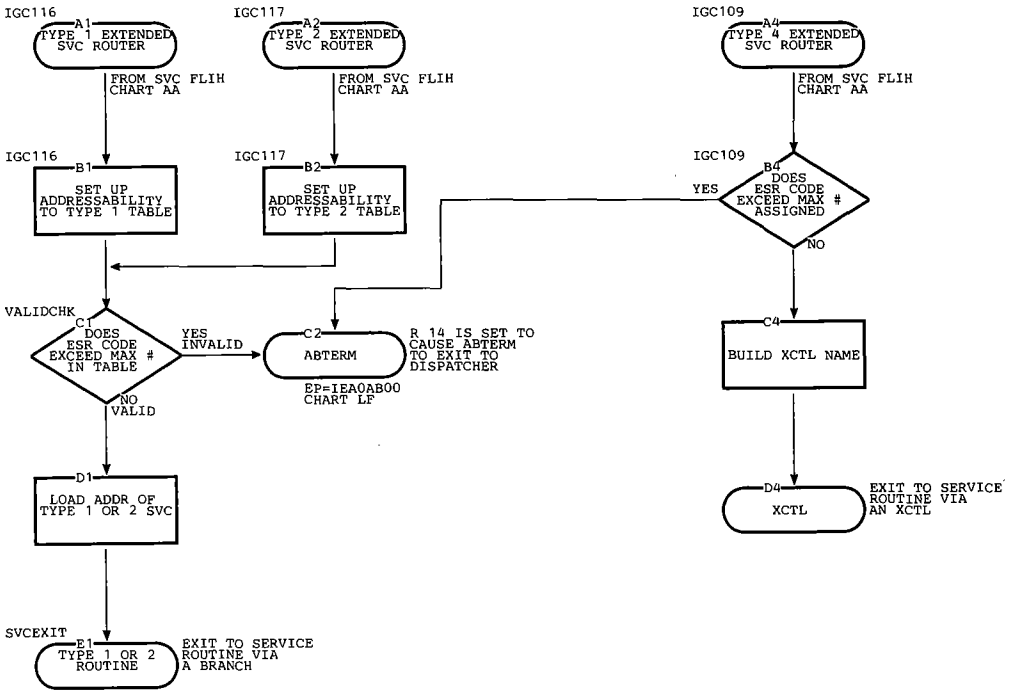


Chart AD. Transient Area Availability Check Routine

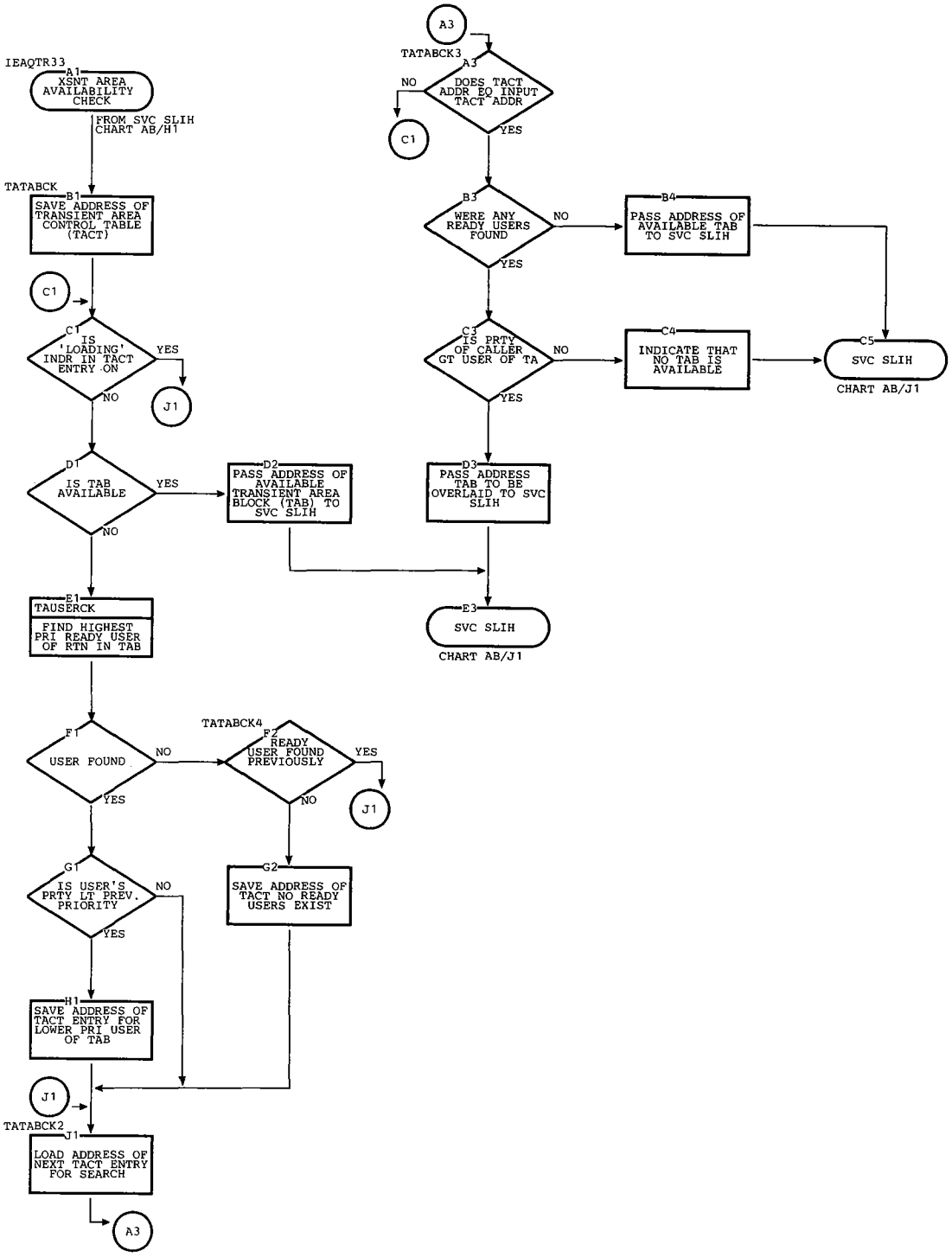


Chart AE. Transient Area Fetch Routine

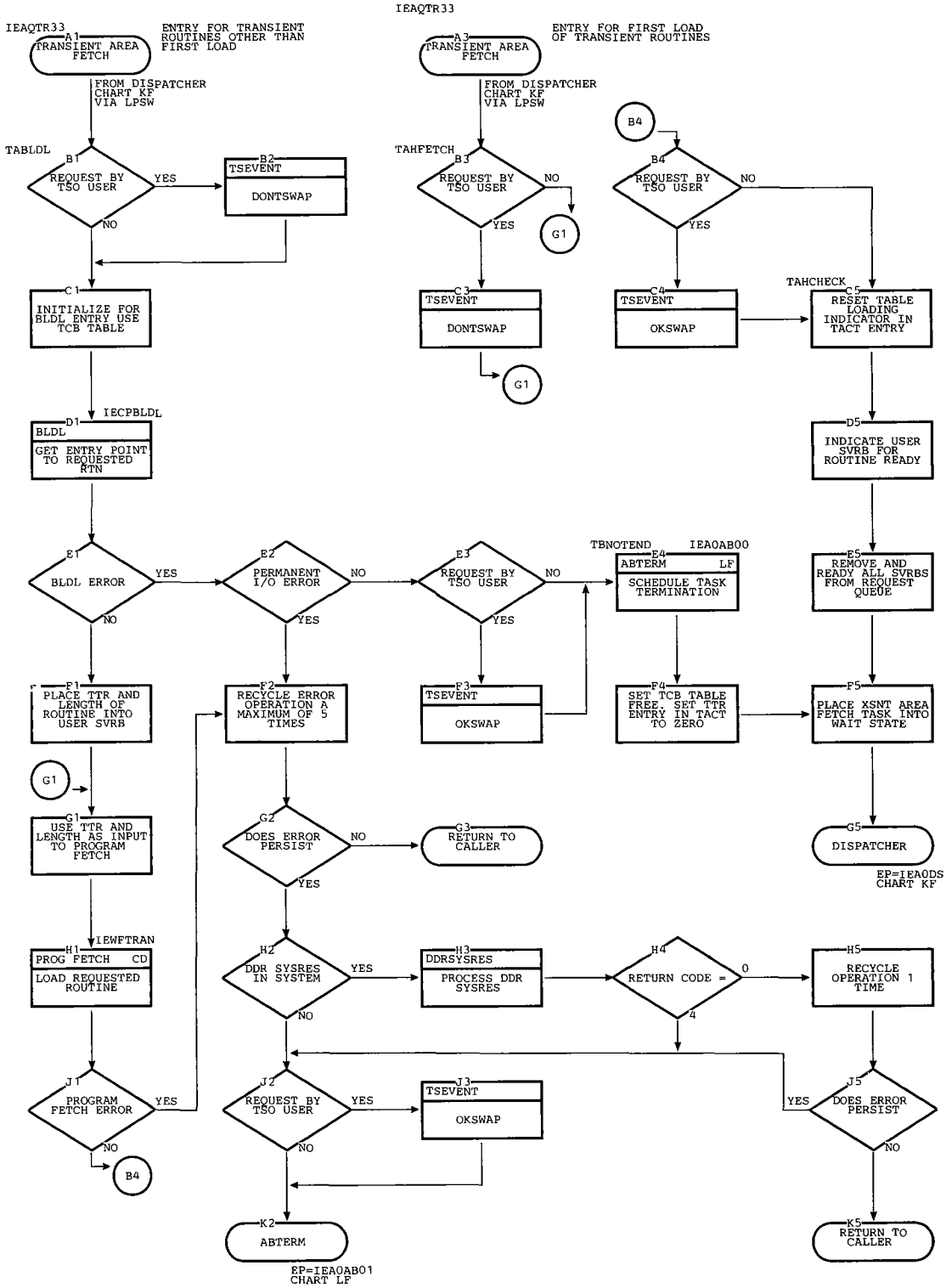


Chart AF. Program Check First-Level Interruption Handler (Page 1 of 2)

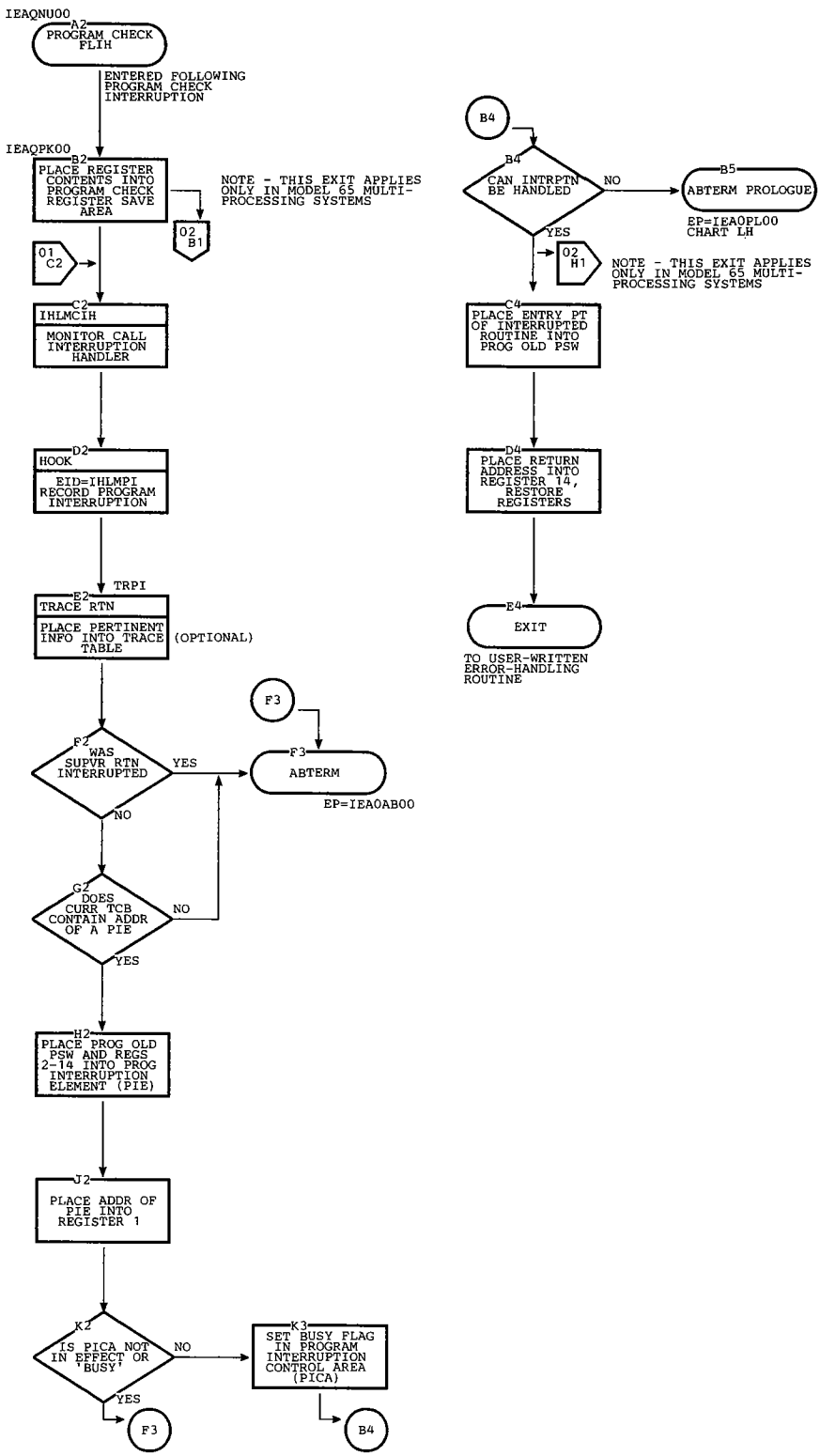


Chart AF. Program Check First-Level Interruption Handler -- Multiprocessing (Page 2 of 2)

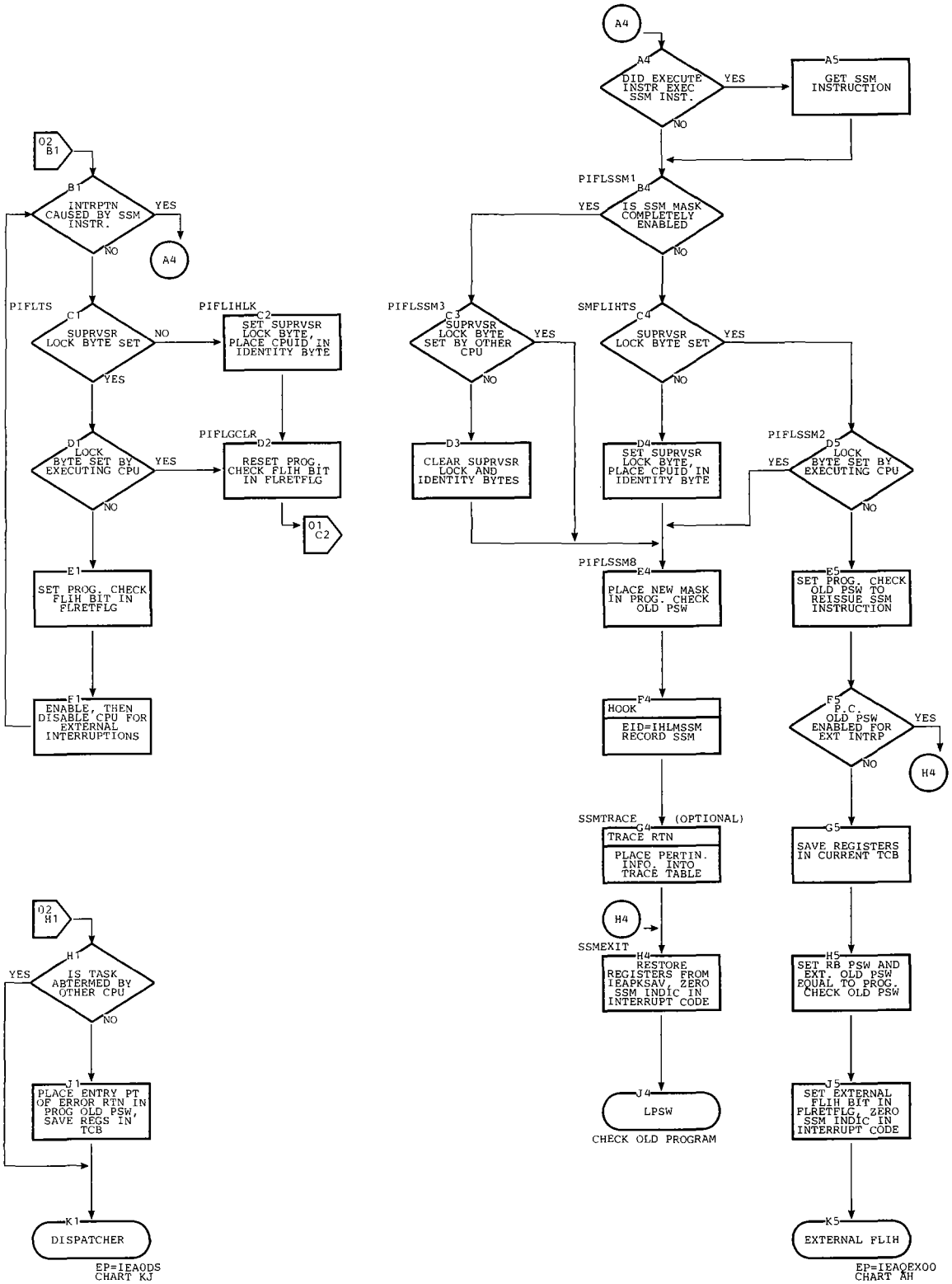
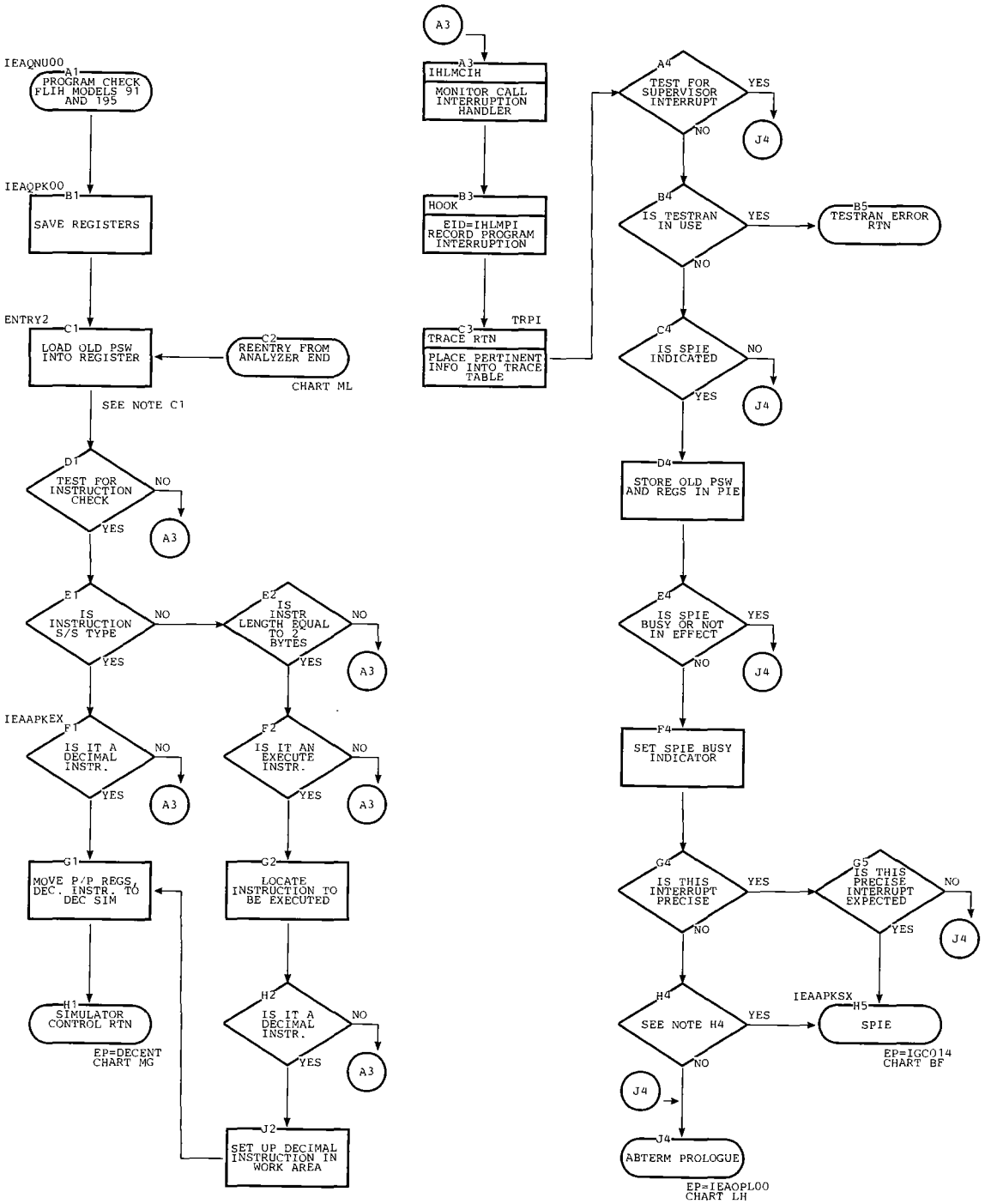


Chart AG. Program Check First-Level Interruption Handler -- Models 91 and 195



NOTE C1: BLOCKS D1-H1 AND E2-J2 ARE USED ONLY WHEN THE DECIMAL SIMULATOR IS INCLUDED IN THE SYSTEM. OTHERWISE, BLOCK C1 CONNECTS TO BLOCK A3.

NOTE H4: ARE ALL IMPRECISE INTERRUPTS, THAT HAVE BEEN RECEIVED, EXPECTED?

Chart AH. External First-Level Interruption Handler -- Uniprocessing

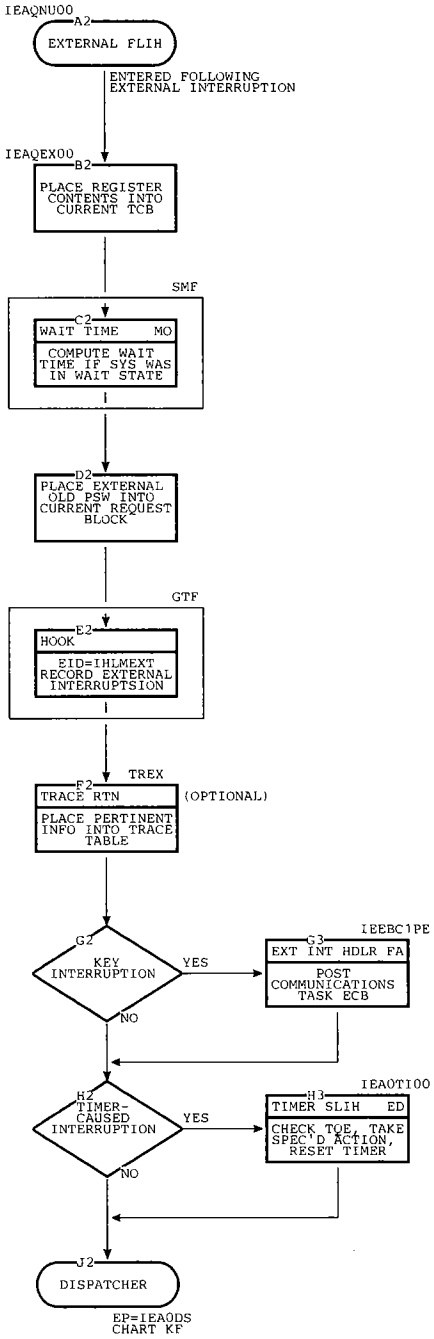


Chart AI. External First-Level Interruption Handler -- Multiprocessing

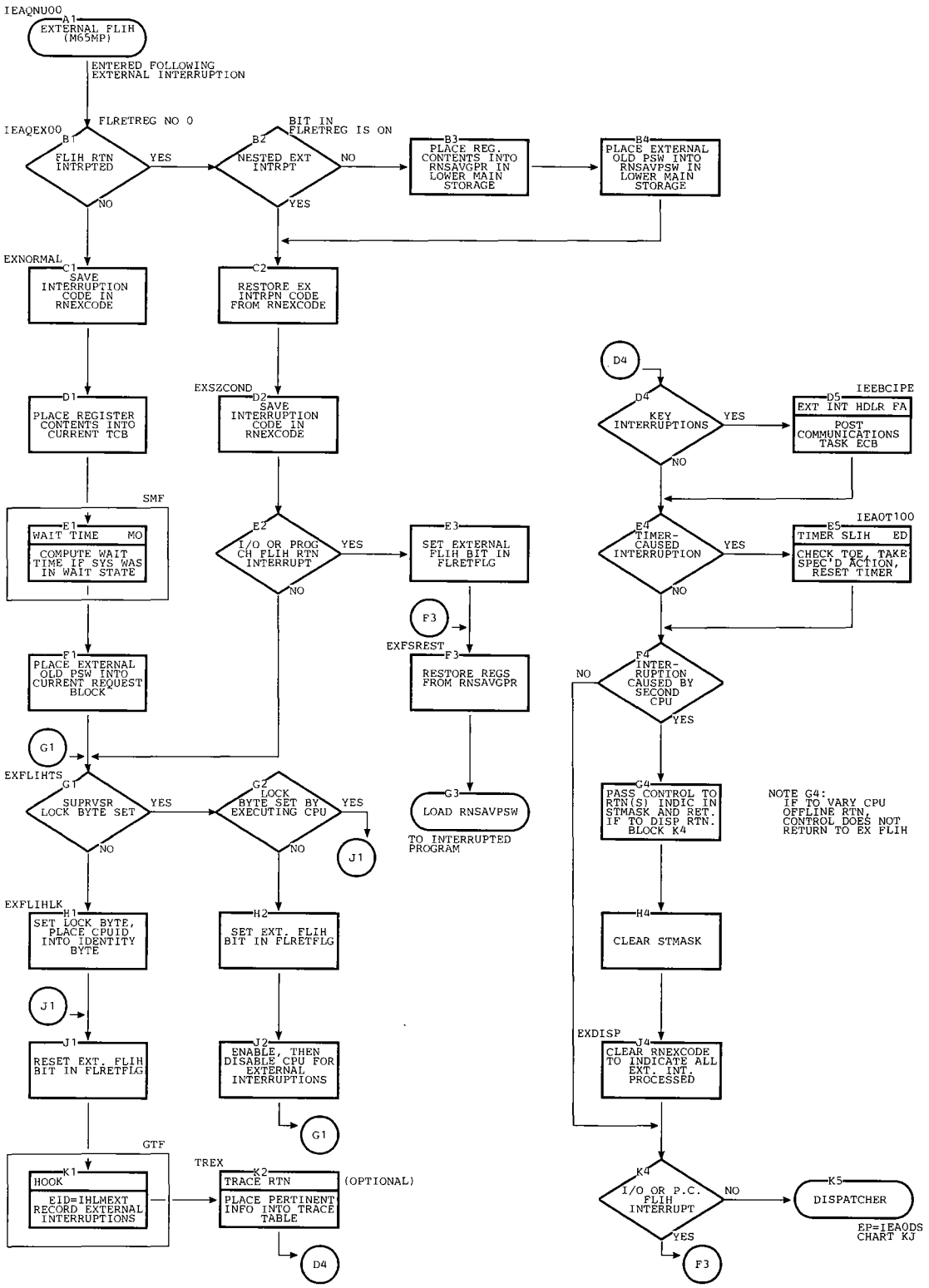


Chart AJ. Input/Output First-Level Interruption Handler

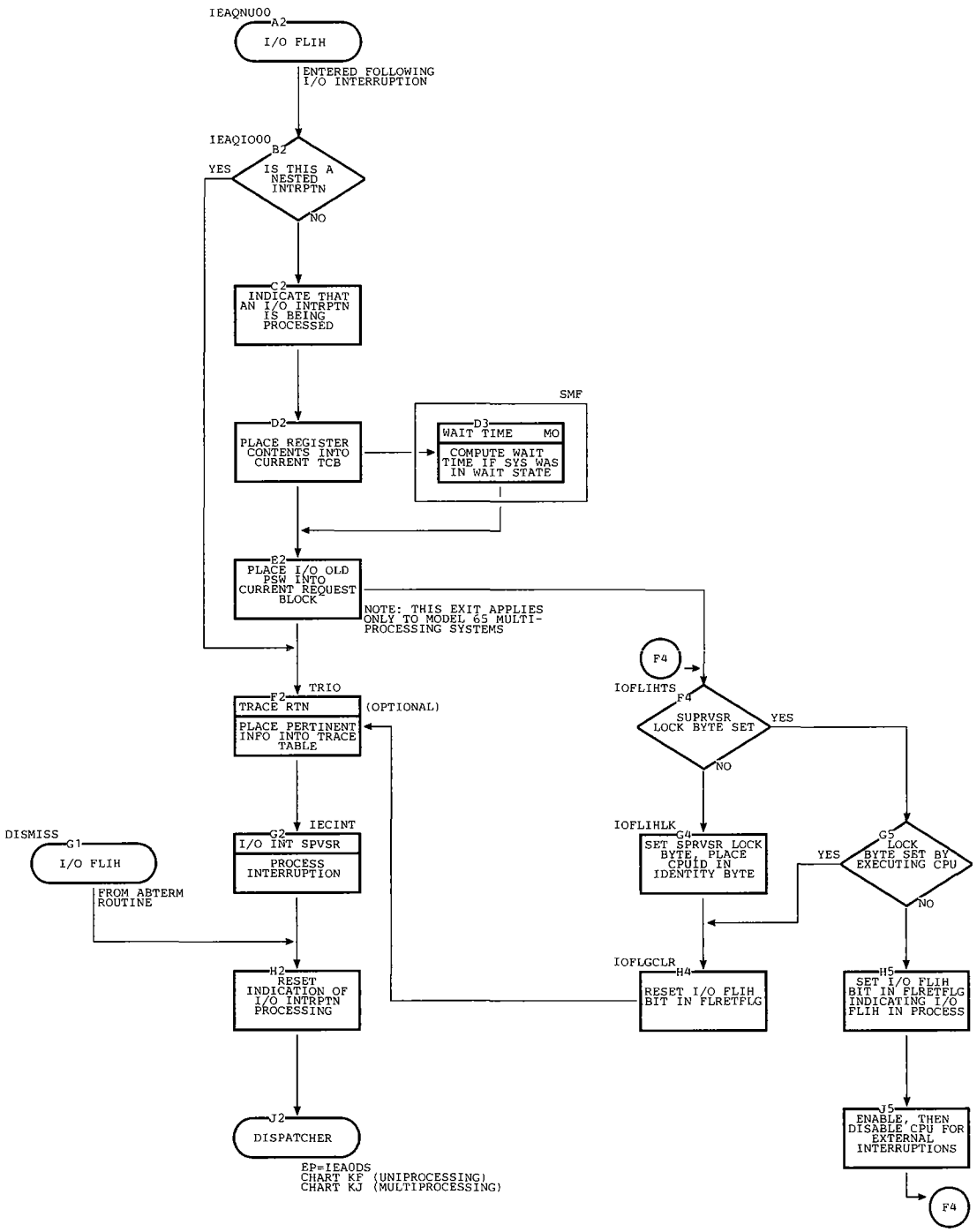


Chart AK. SER0 Routine

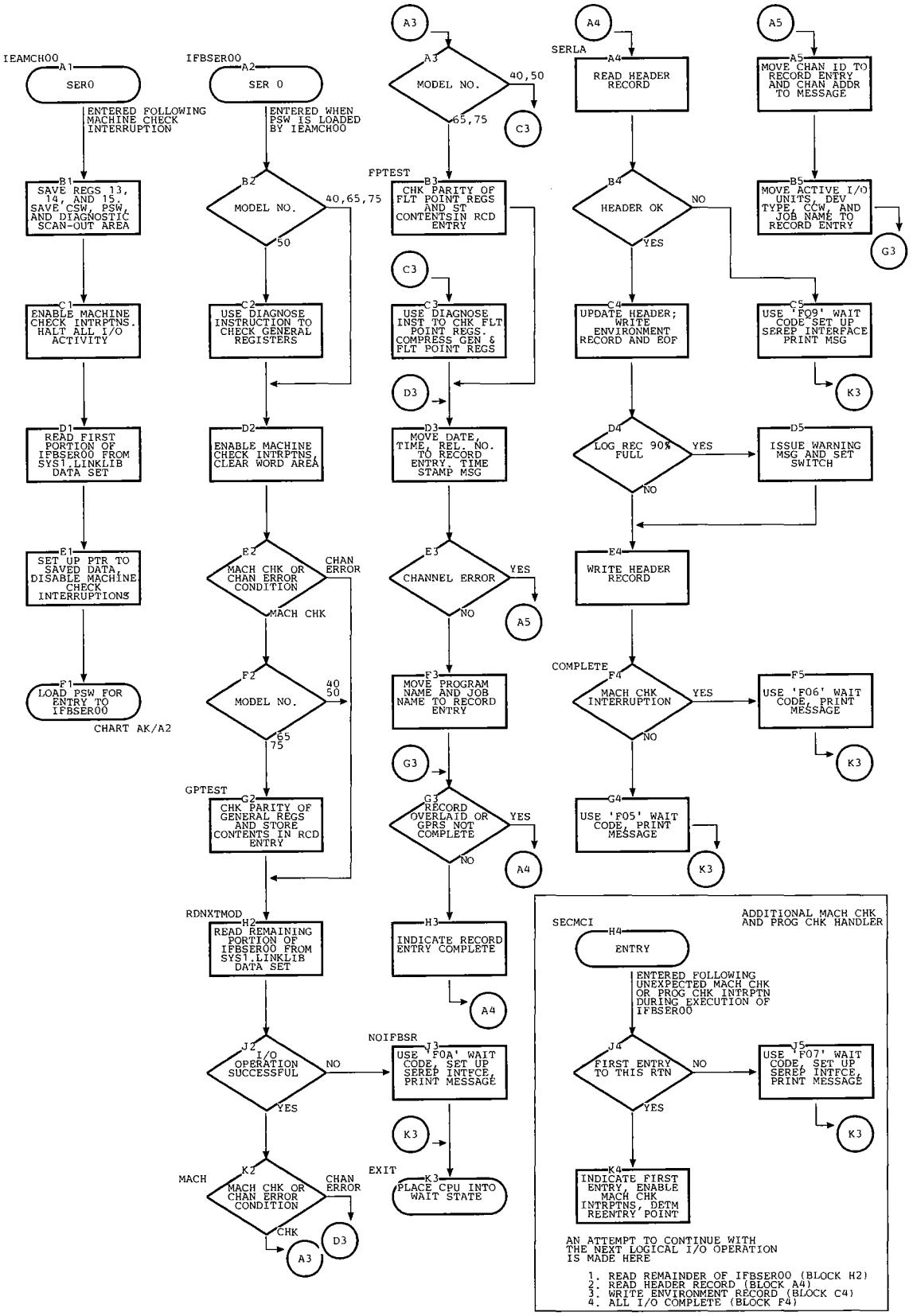


Chart AL. SER1 Routine -- Models 40, 50, 65, and 75

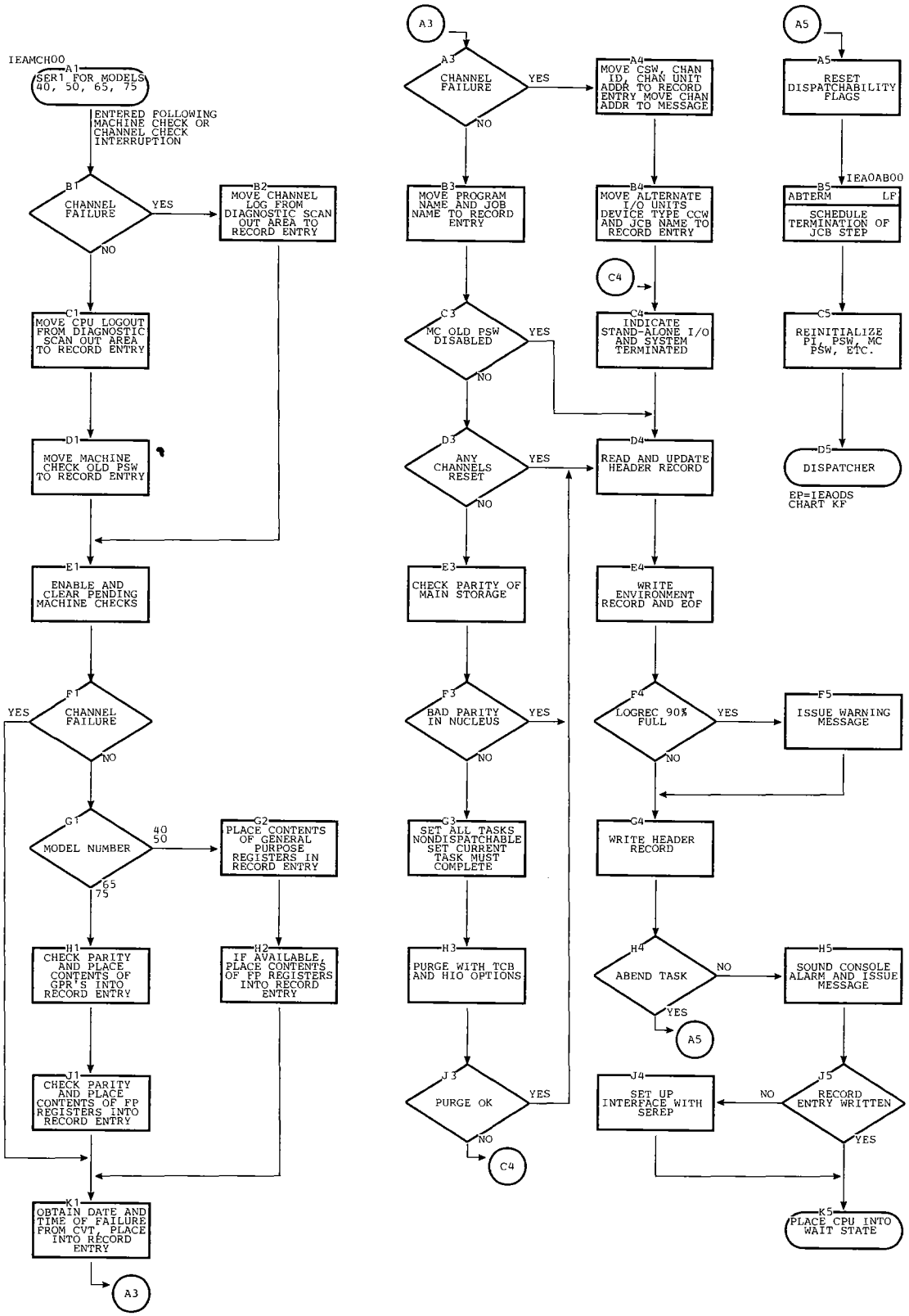


Chart AM. SER1 Routine -- Models 91, 95, and 195 (Page 1 of 2)

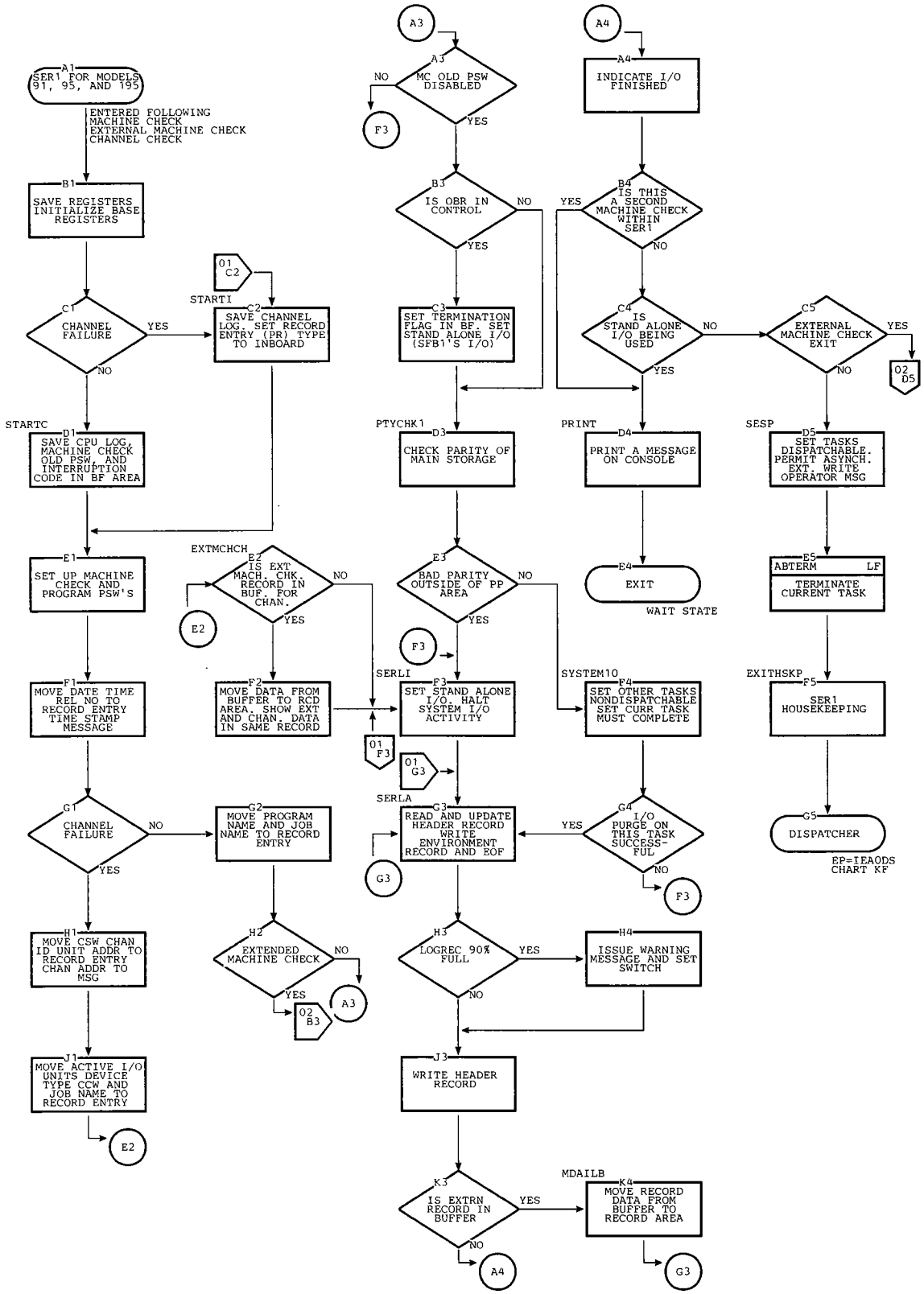


Chart AM. SER1 Routine -- Models 91, 95, and 195 (Page 2 of 2)

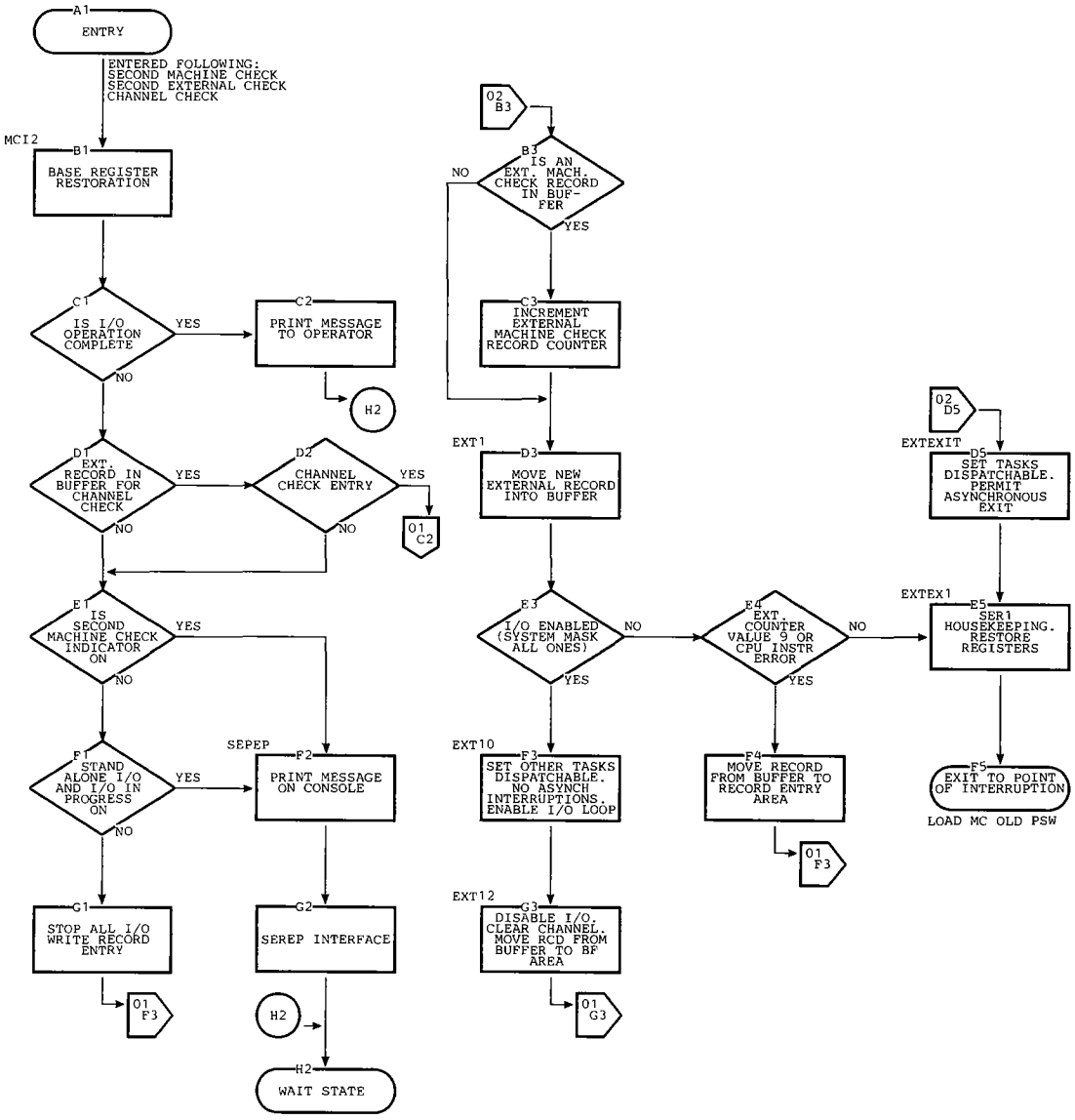


Chart BA. Attach Routine (Page 1 of 5)

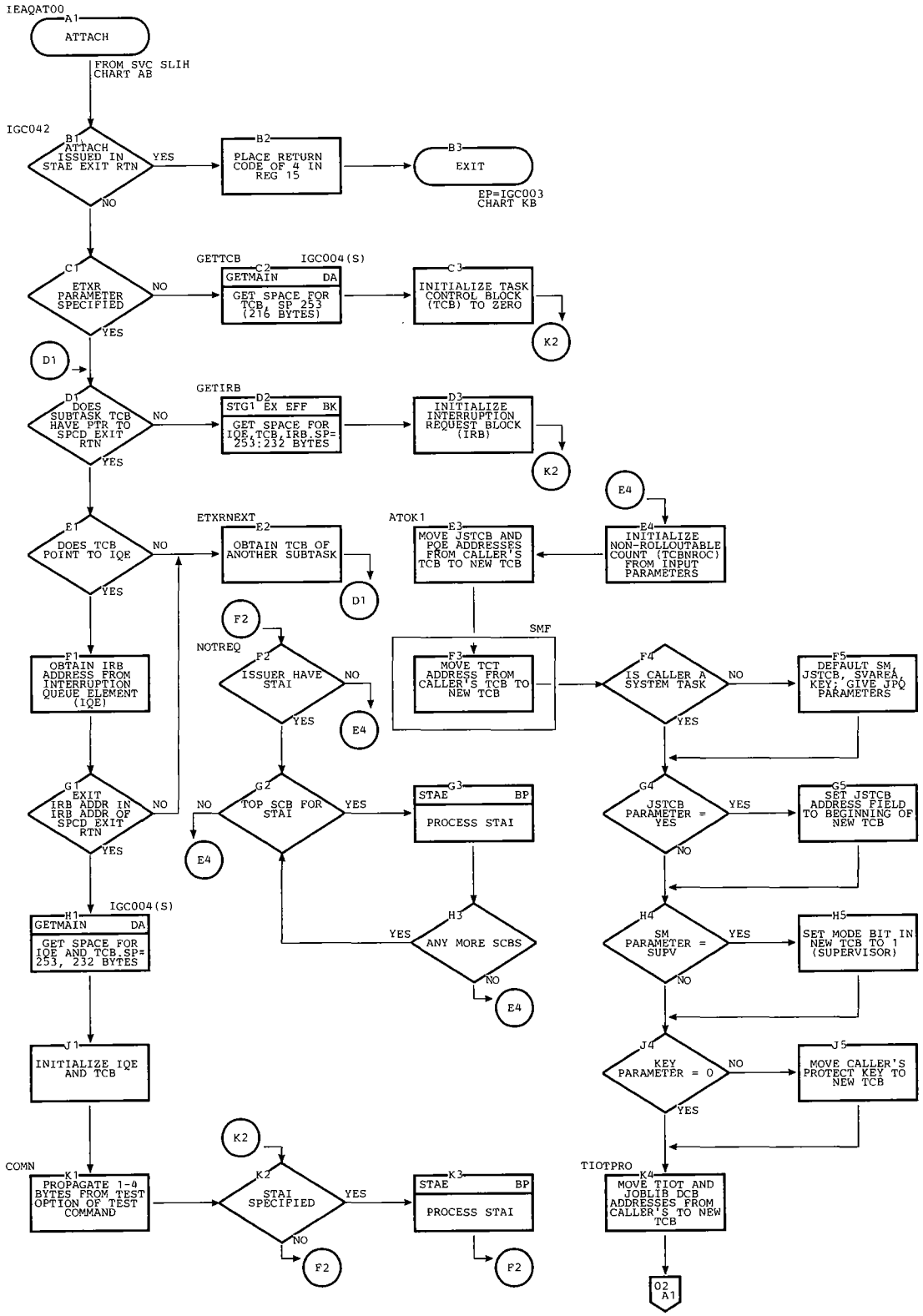
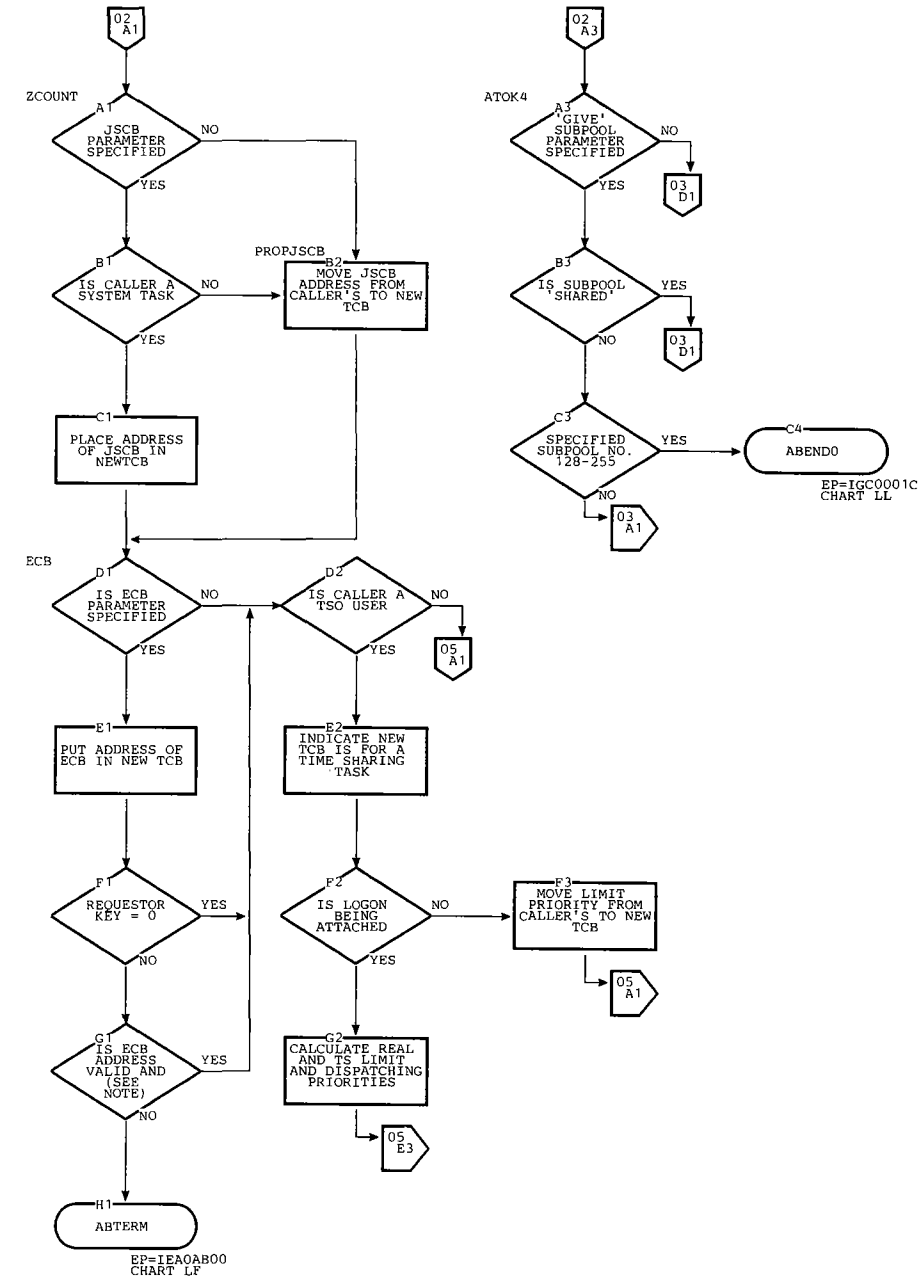


Chart BA. Attach Routine (Page 2 of 5)



NOTE: IS ECB KEY = ATTACHER'S KEY OR IS ATTACHER'S KEY = 0.

Chart BA. Attach Routine (Page 3 of 5)

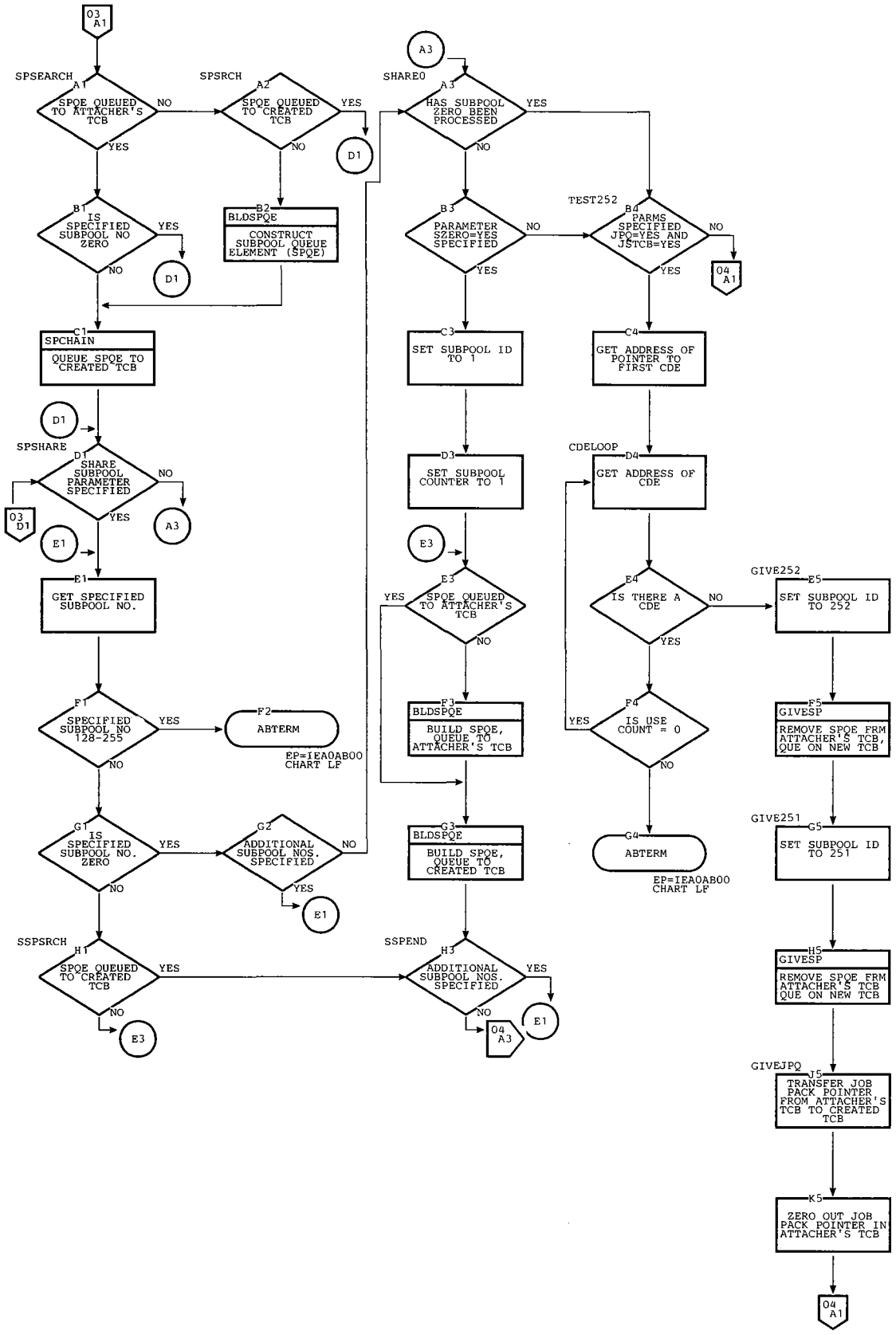


Chart BA. Attach Routine (Page 4 of 5)

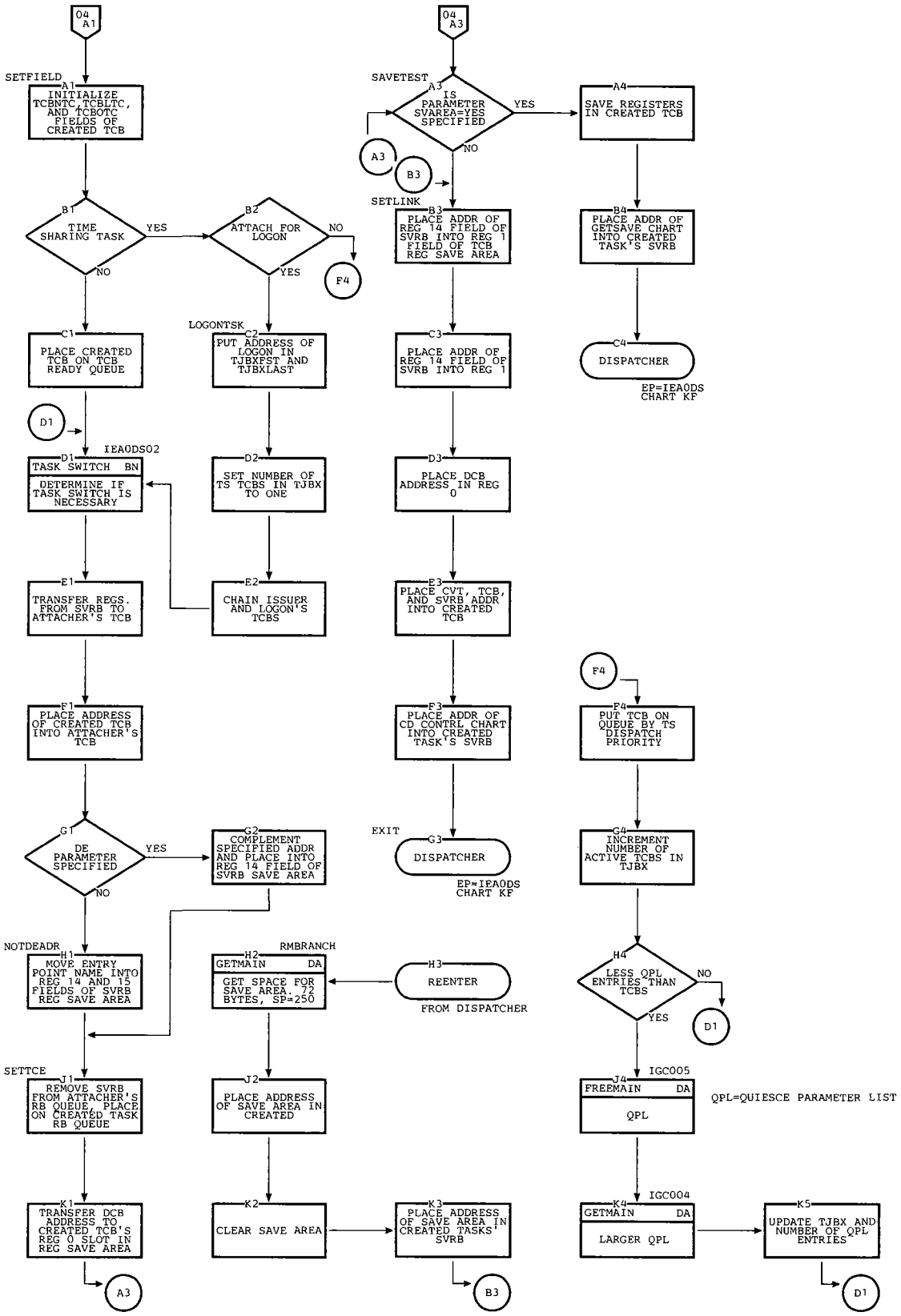
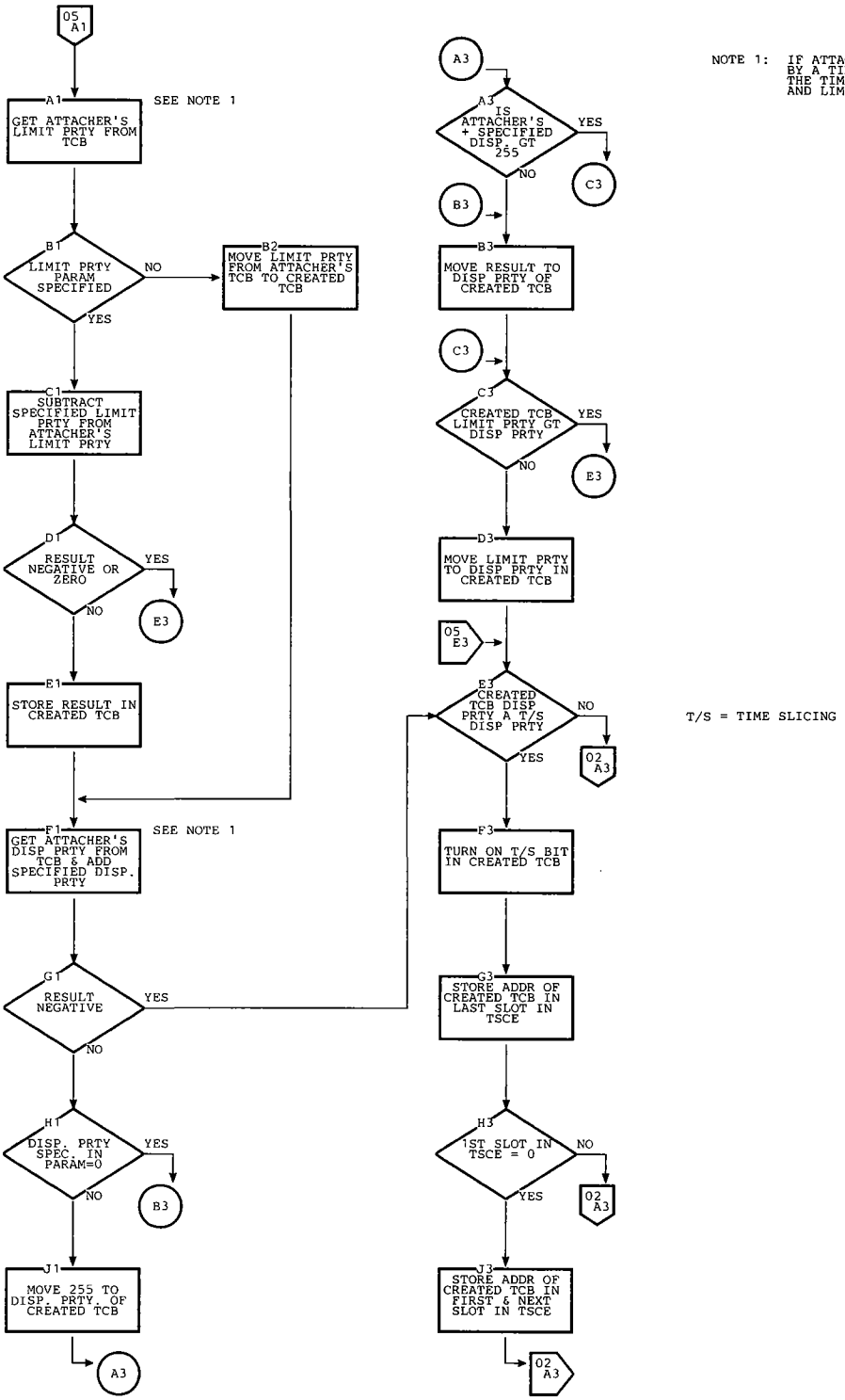


Chart BA. Attach Routine (Page 5 of 5)



NOTE 1: IF ATTACH IS ISSUED BY A TIME-SHARING TASK, THE TIME-SHARING DISPATCHING AND LIMIT PRIORITIES ARE USED.

T/S = TIME SLICING

Chart BB. Chap Routine

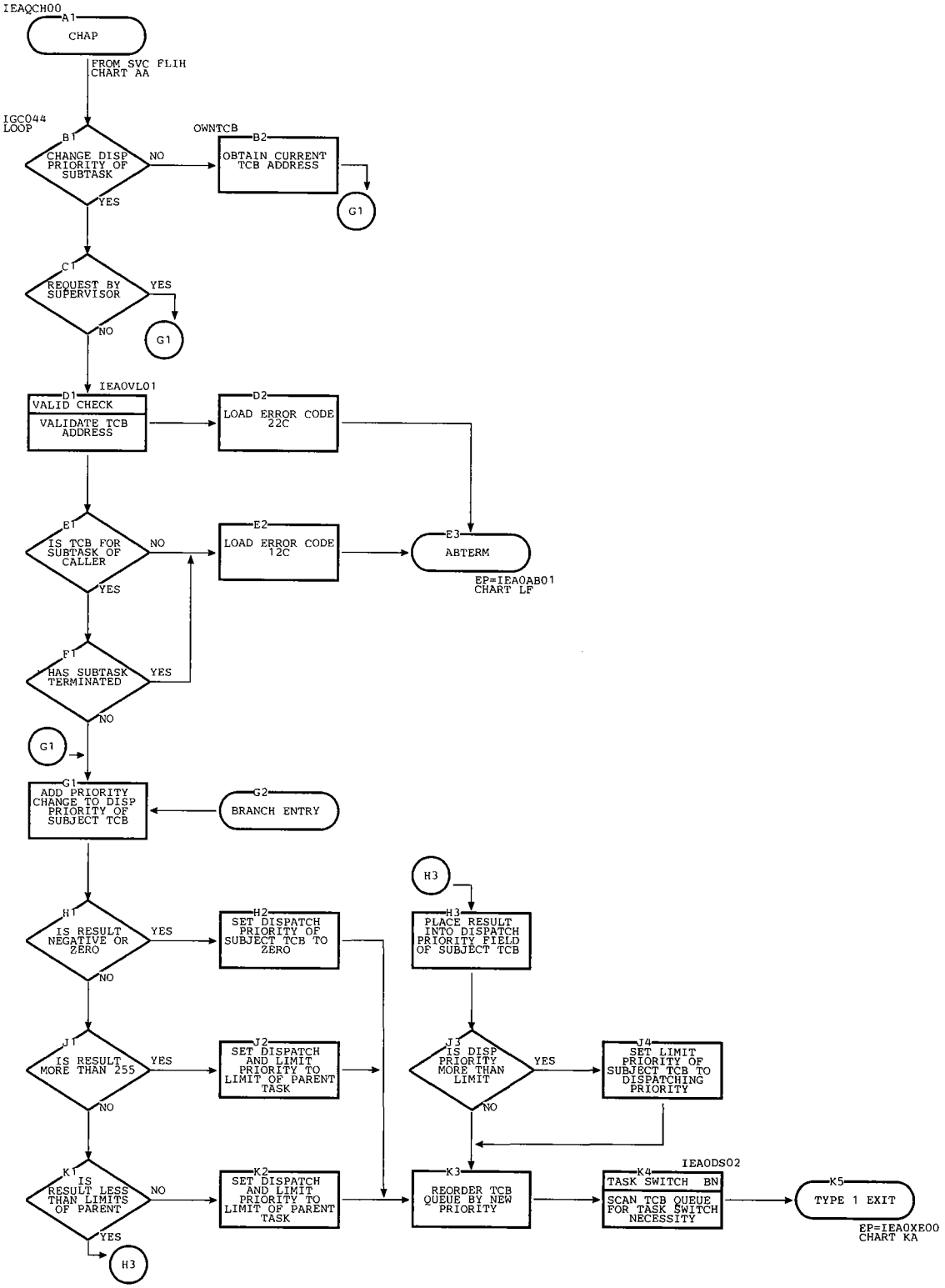


Chart BC. Chap Routine with Time-Slicing (Page 1 of 3)

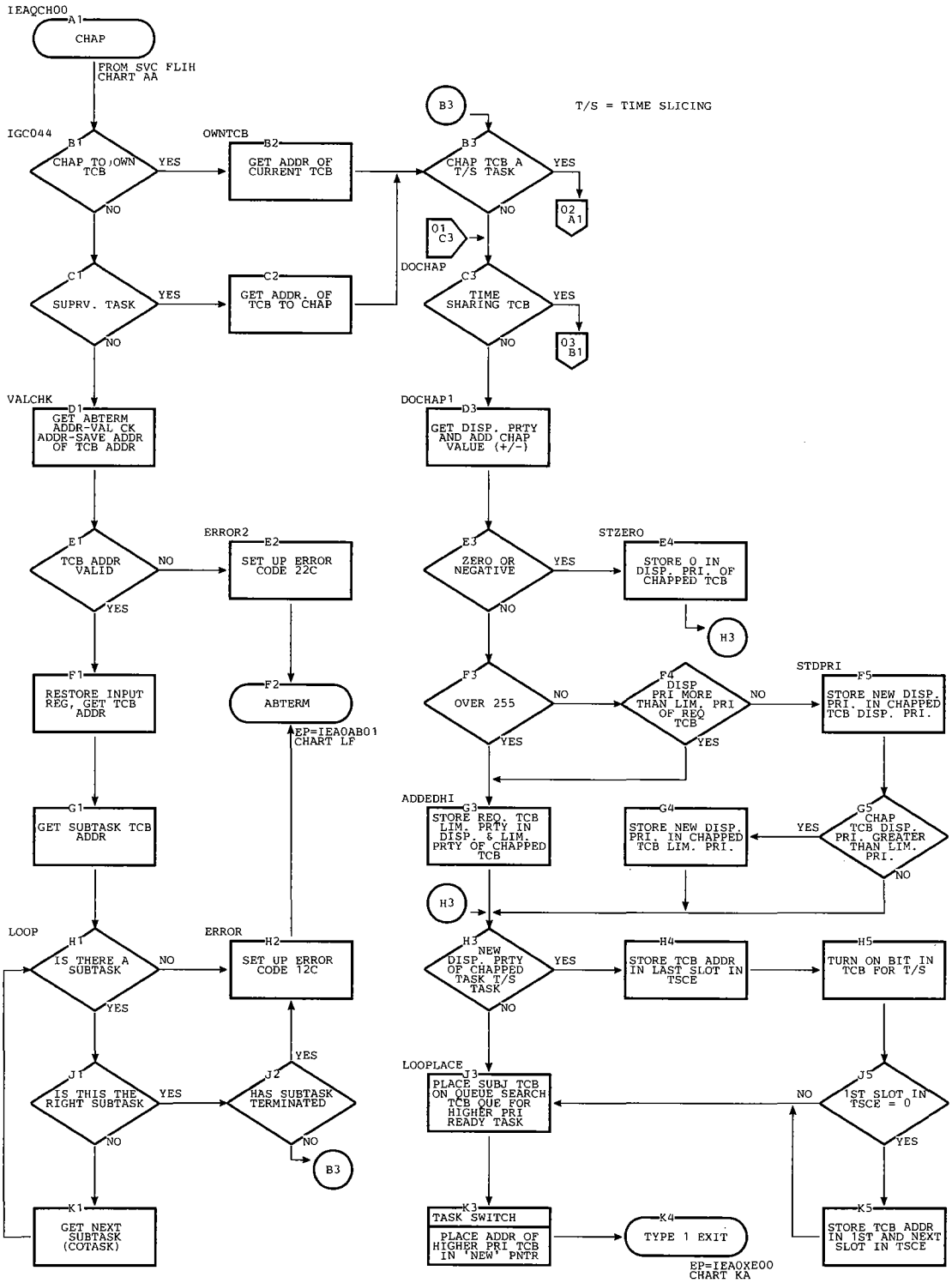


Chart BC. Chap Routine -- with Time-Slicing (Page 2 of 3)

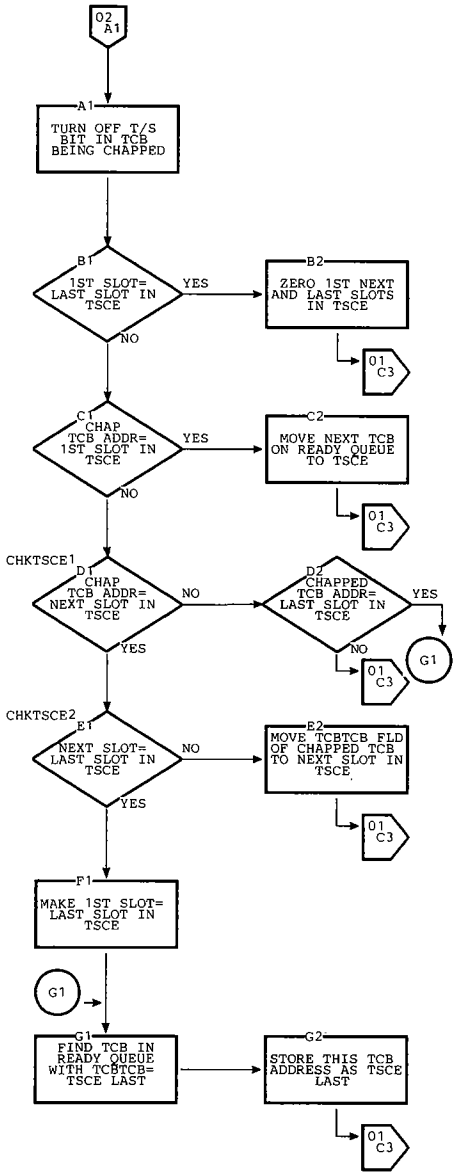


Chart BC. Chap Routine -- with Time Sharing Option (Page 3 of 3)

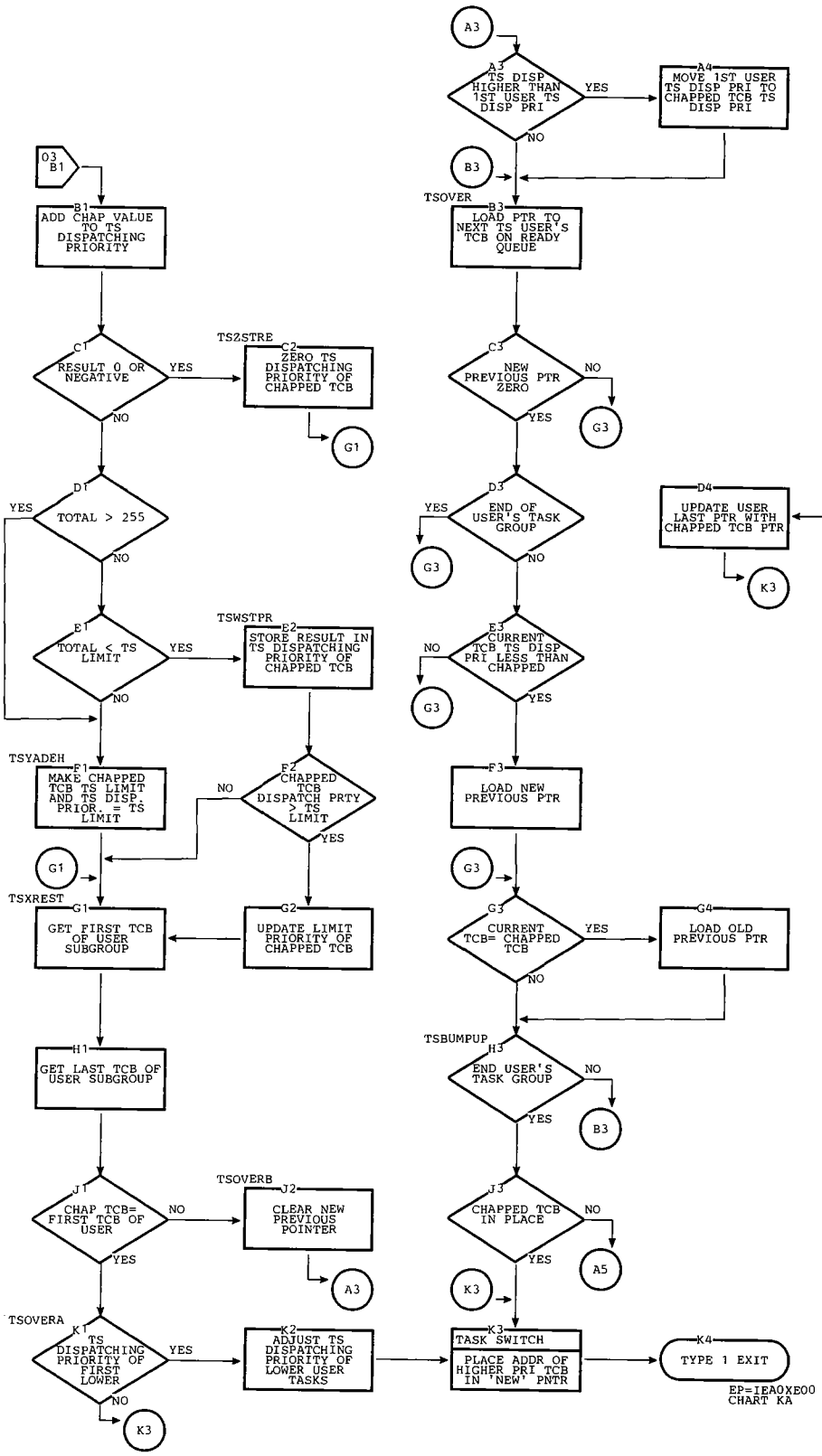


Chart BD. Extract Routine

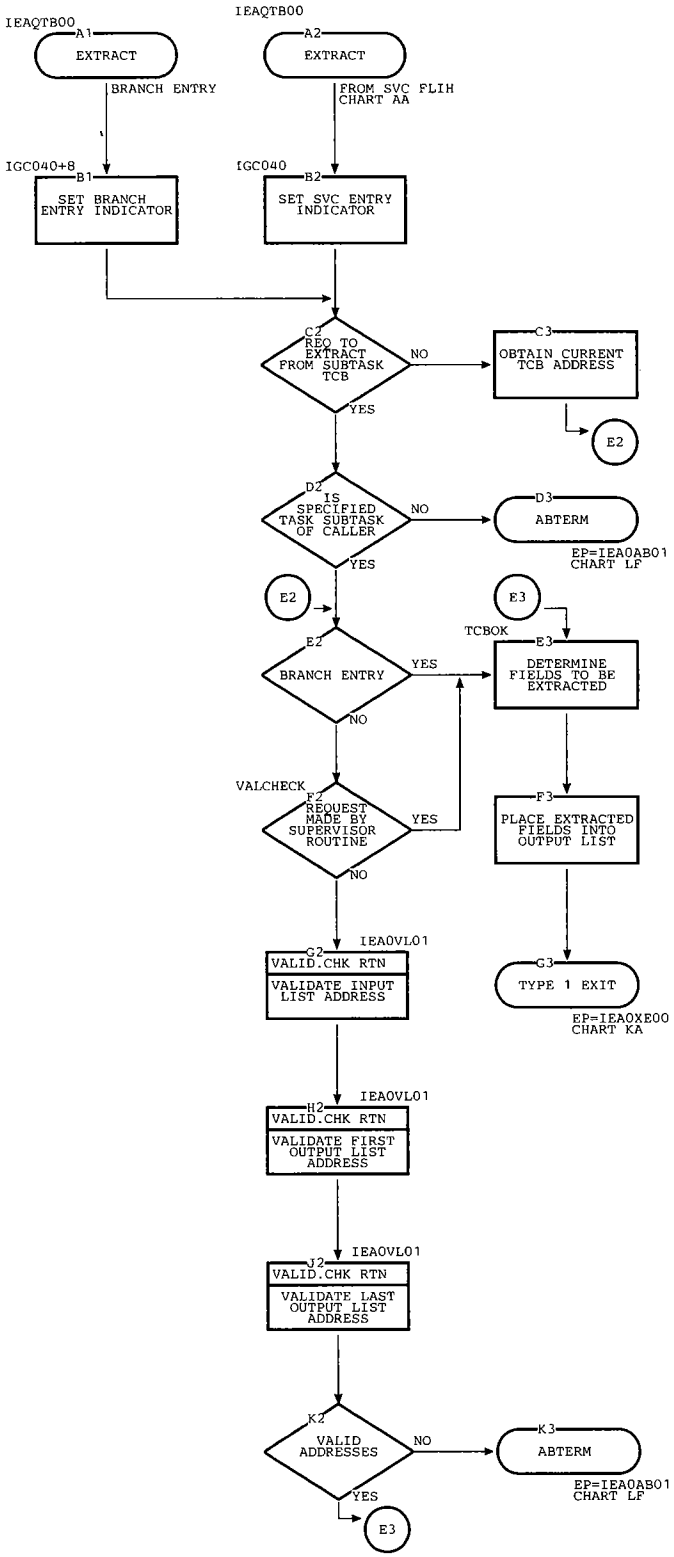


Chart BE. Detach Routine

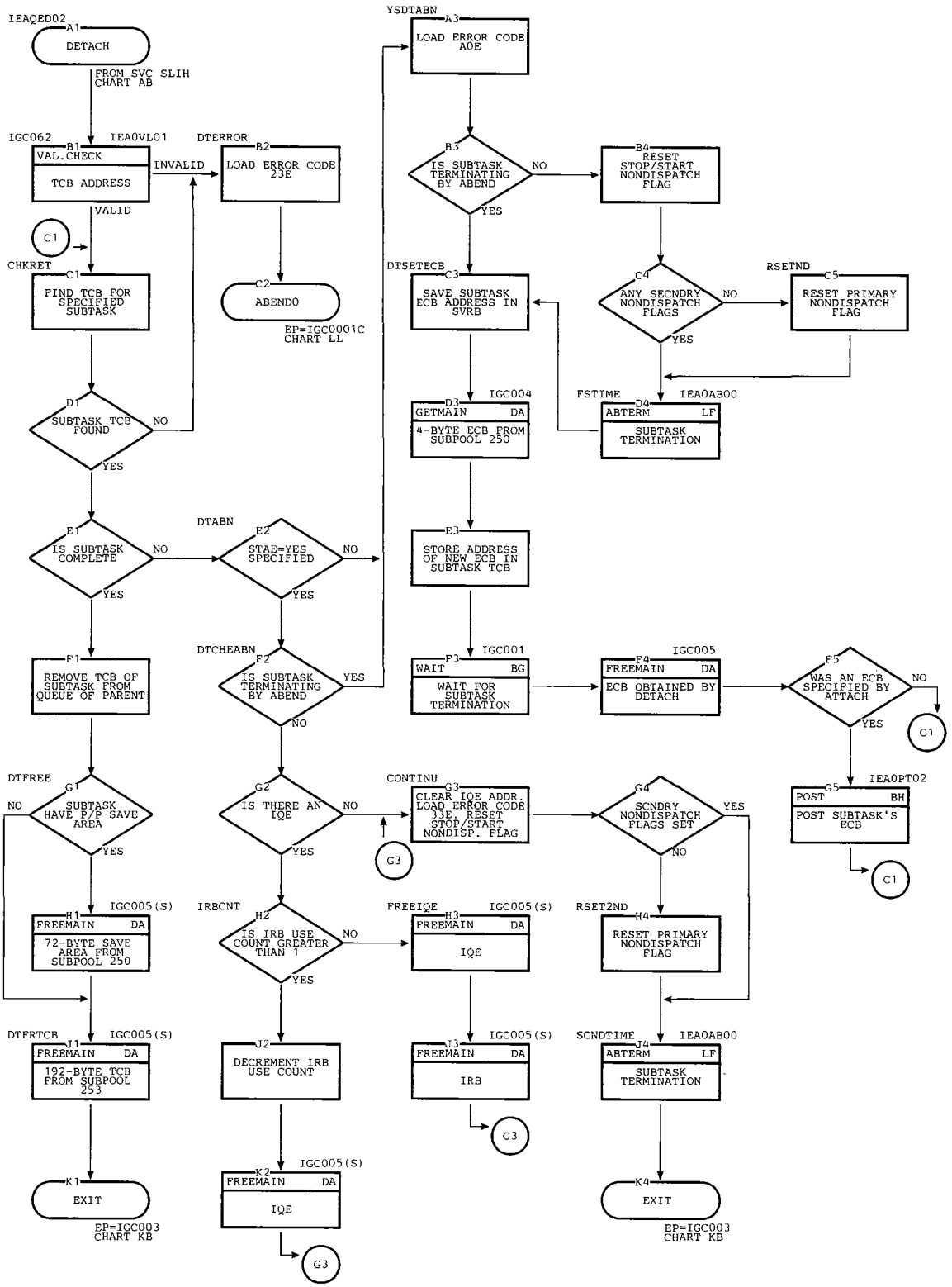


Chart BF. SPIE Routine

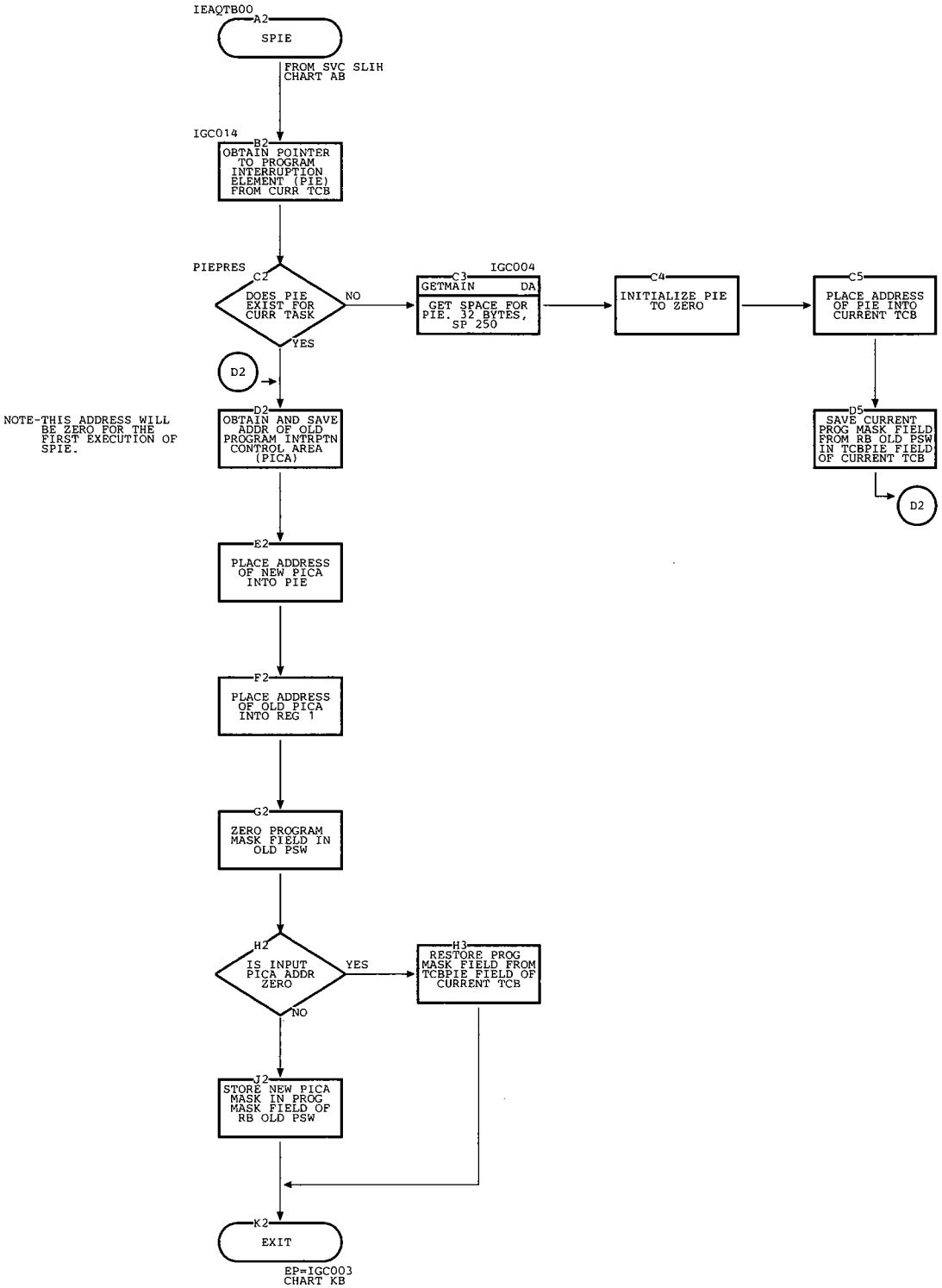


Chart BG. Wait Routine (Page 1 of 2)

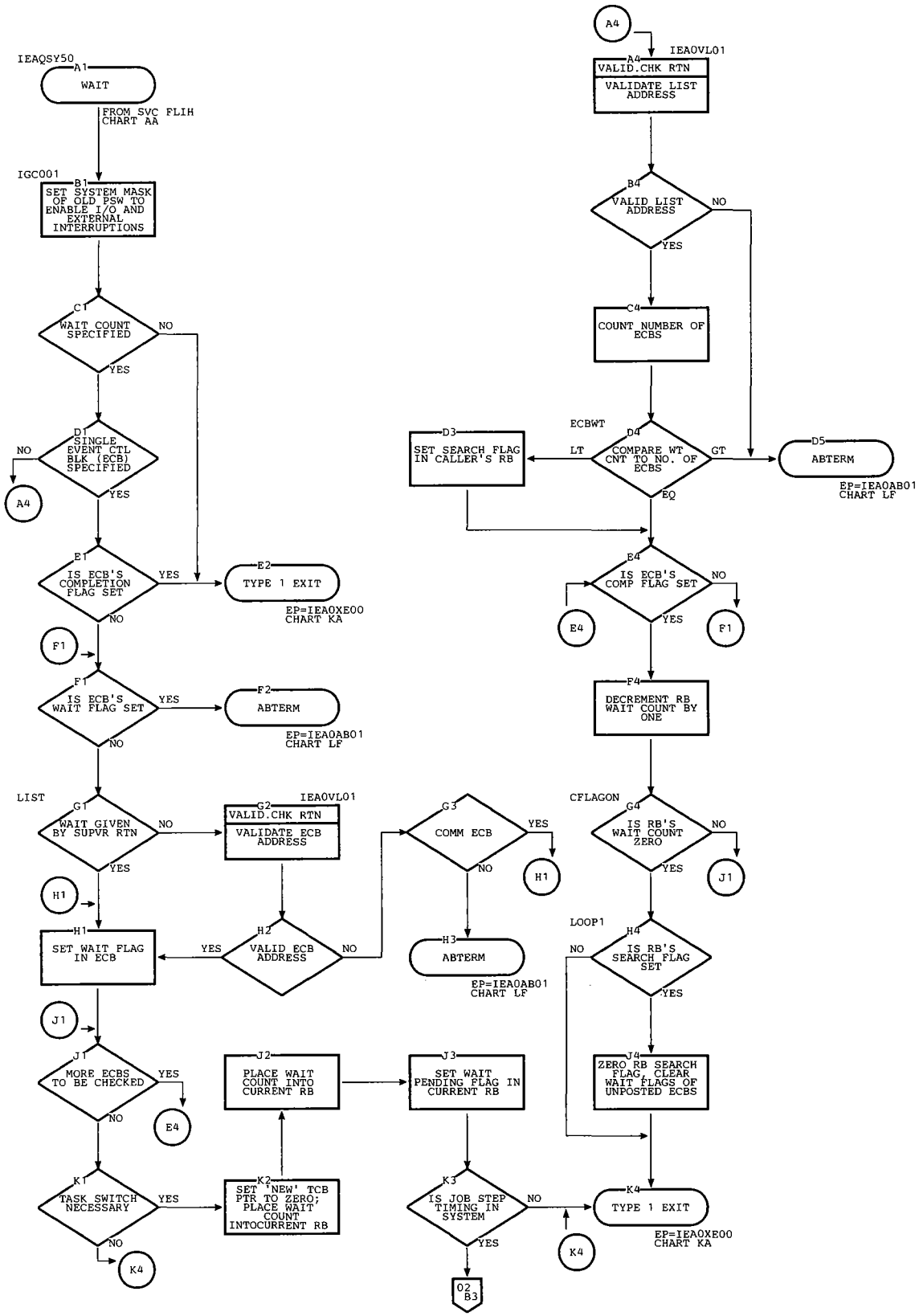


Chart BG. Wait Routine -- with Job Step Timing (Page 2 of 2)

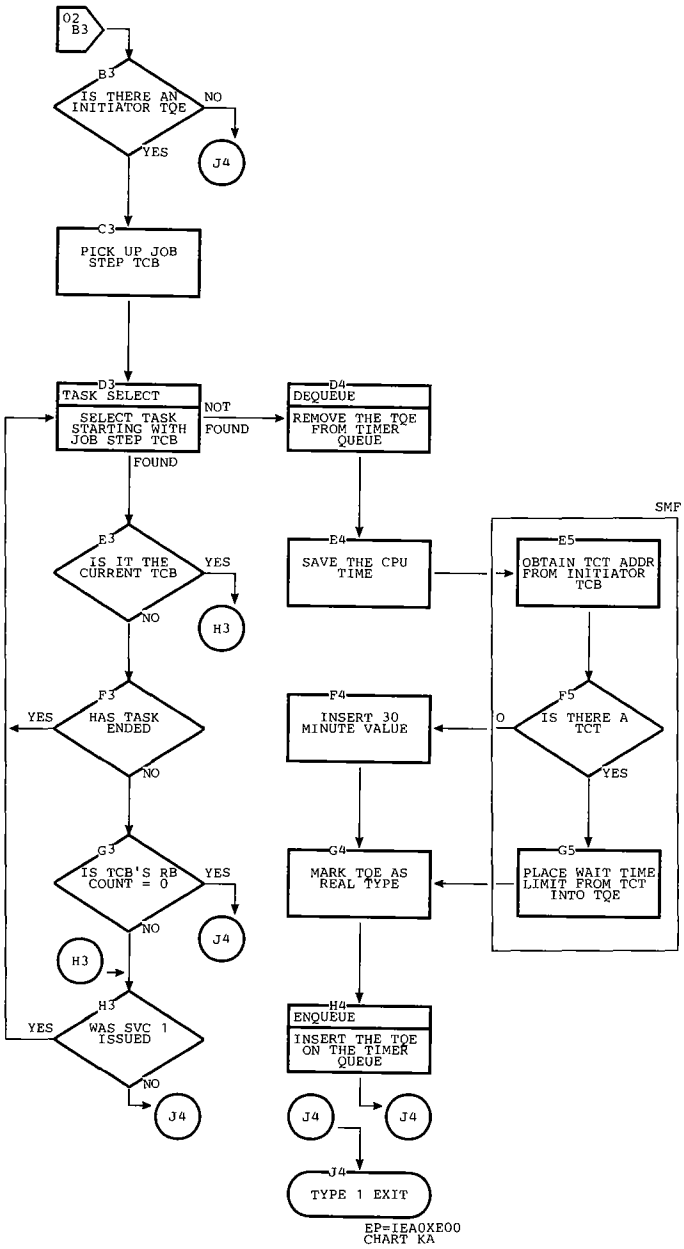


Chart BH. Post Routine (Page 1 of 2)

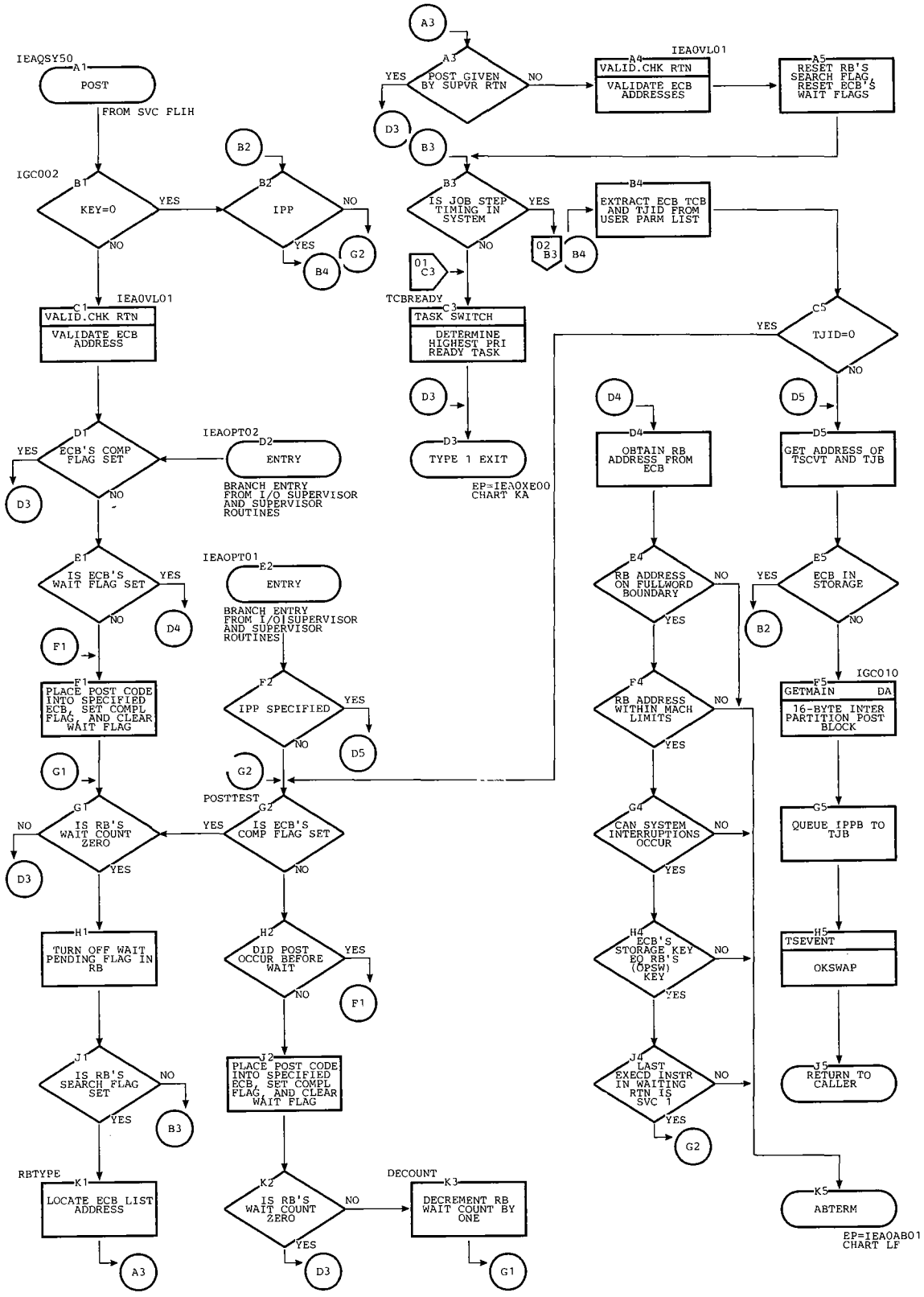


Chart BH. Post Routine -- with Job Step Timing (Page 2 of 2)

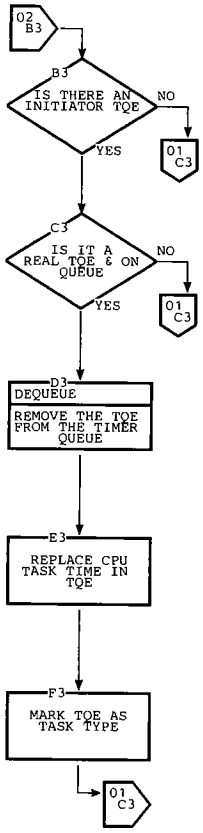


Chart BI. ENQ Routine (Page 1 of 2)

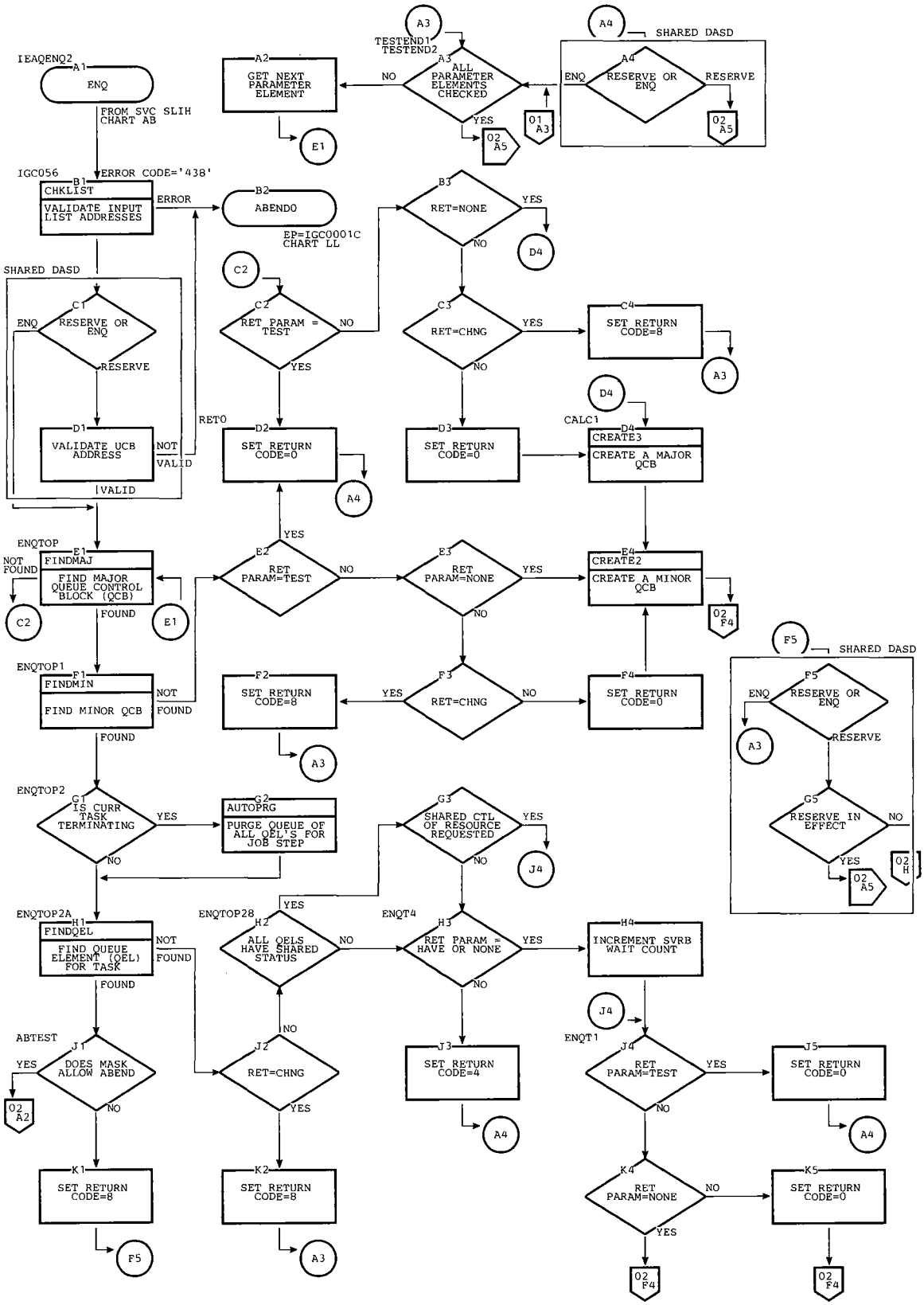


Chart BI. ENQ Routine (Page 2 of 2)

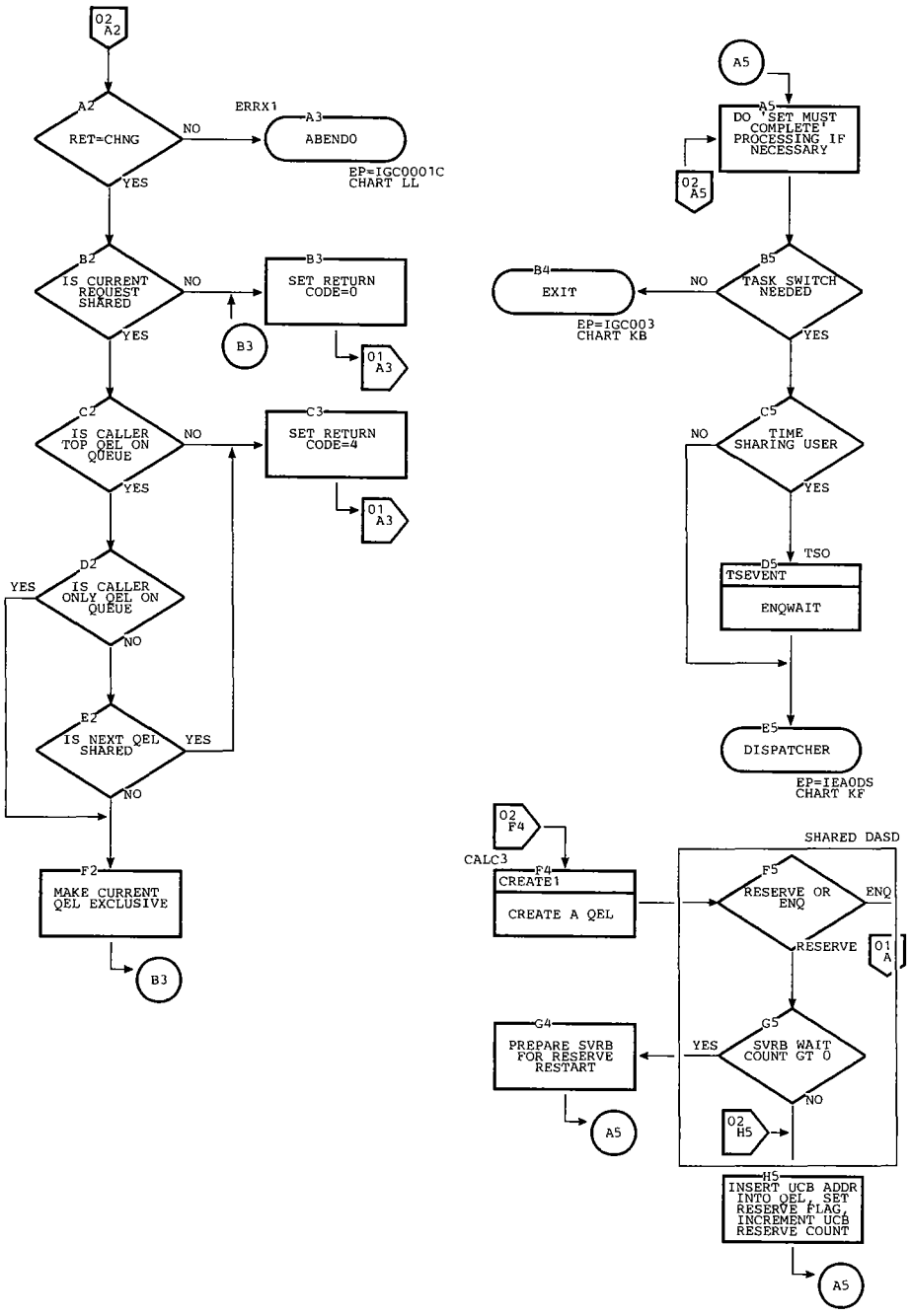


Chart BJ. DEQ Routine -- with Time Sharing Option (Page 1 of 2)

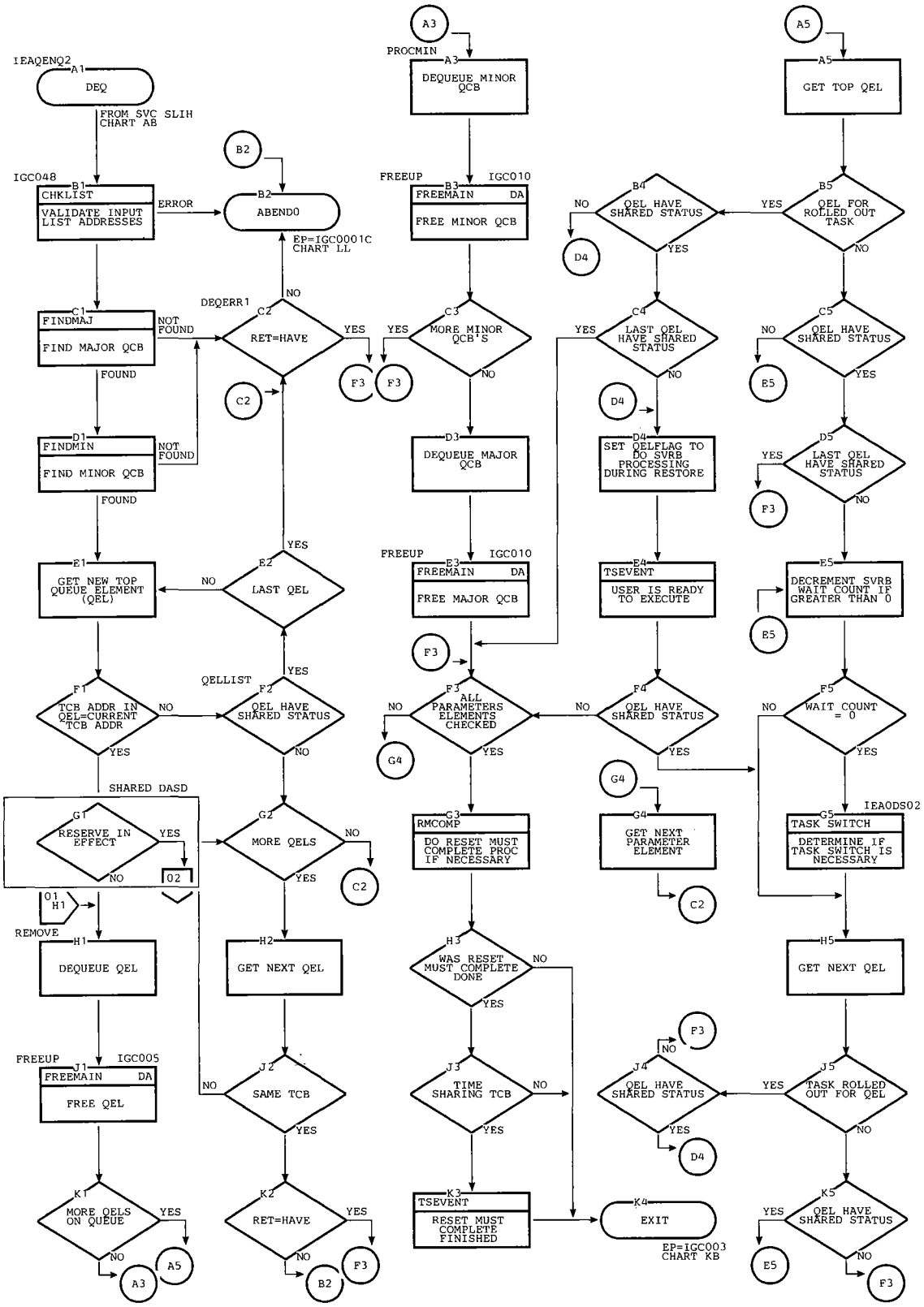


Chart BJ. DEQ Routine -- with Shared DASD (Page 2 of 2)

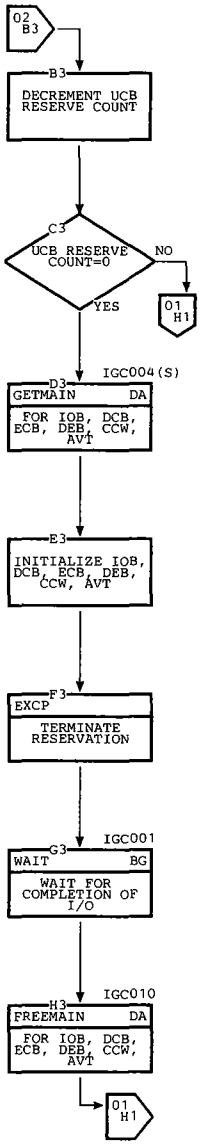


Chart BK. Stage 1 Exit Effector

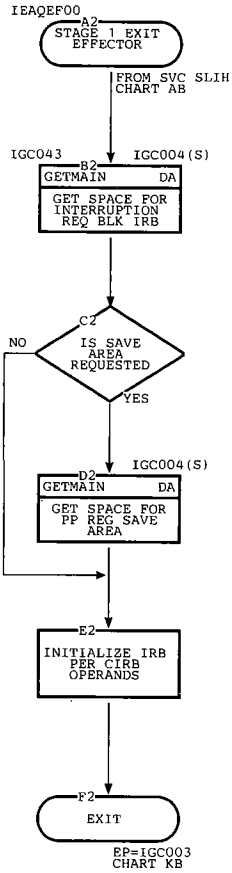


Chart BL. Stage 2 Exit Effector

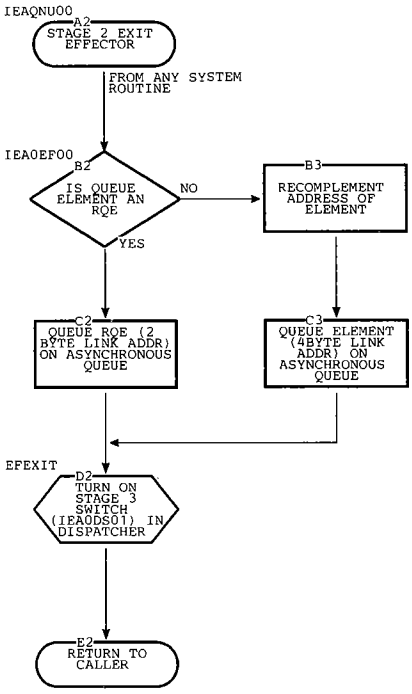


Chart BM. Stage 3 Exit Effector (Page 1 of 2)

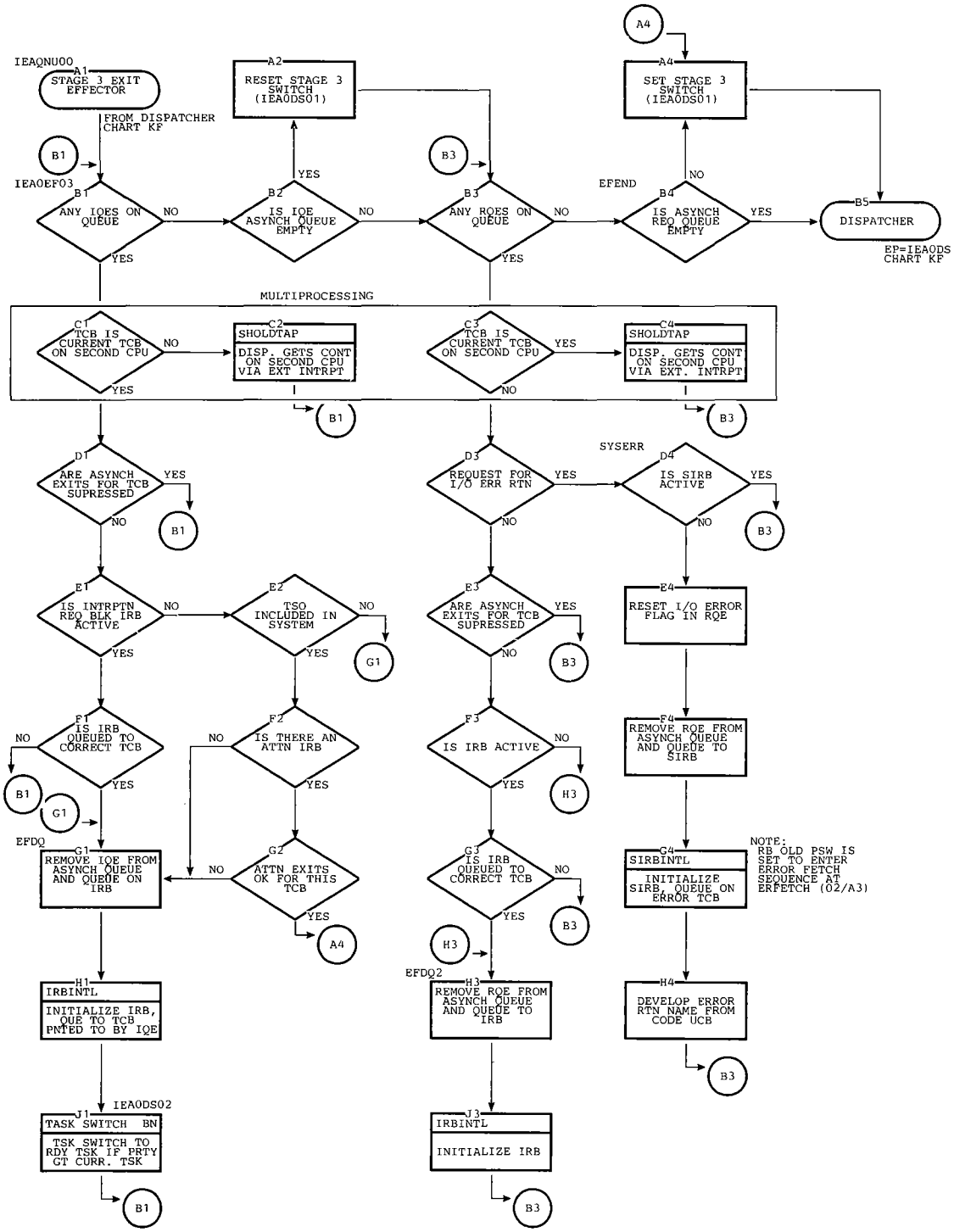


Chart BM. Stage 3 Exit Effector (Page 2 of 2)

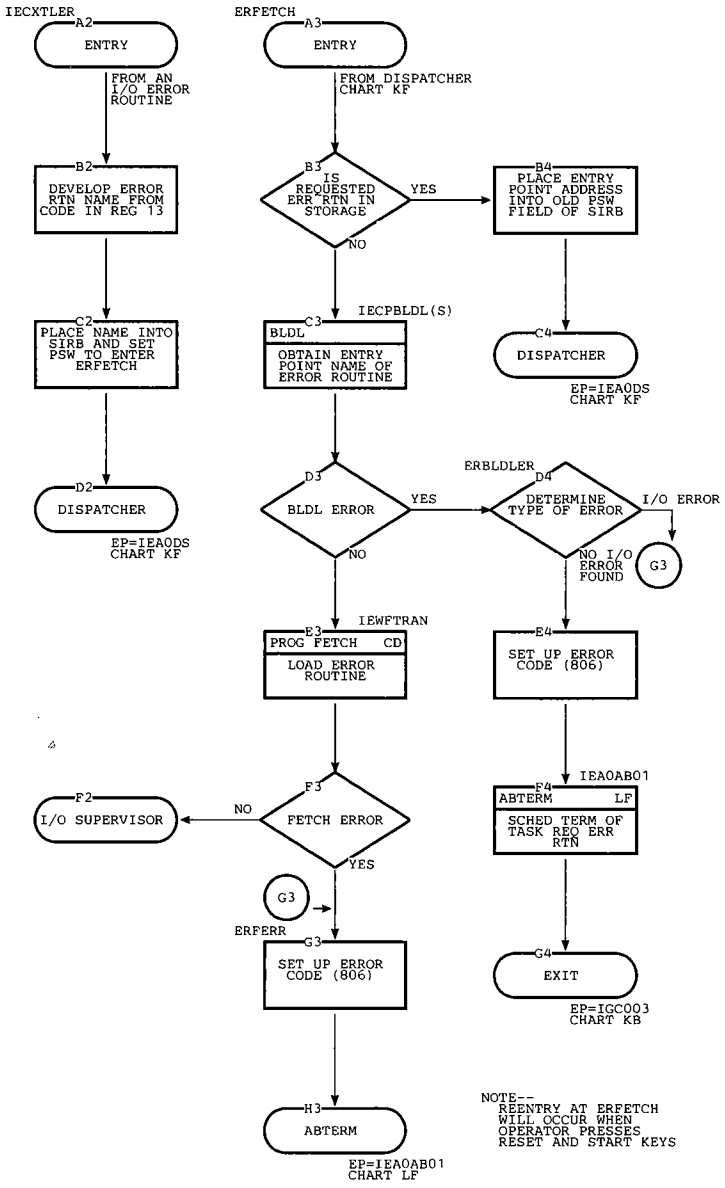


Chart BN. Task Switching Routine -- Uniprocessing

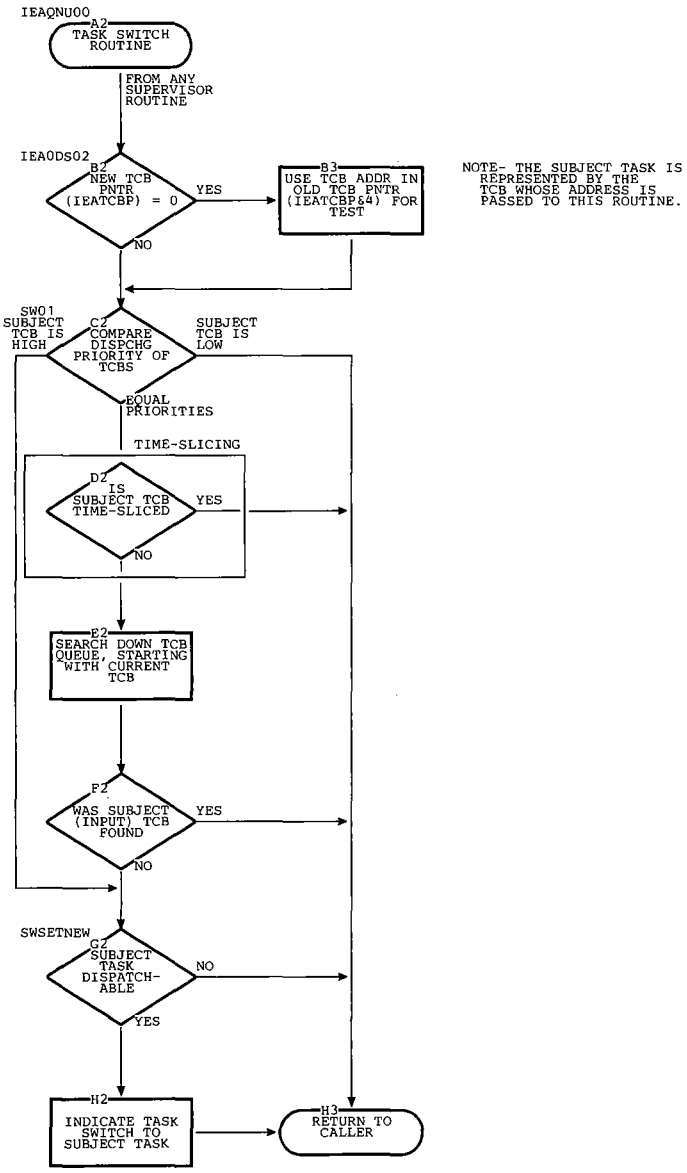


Chart BO. Task Switching Routine -- Multiprocessing

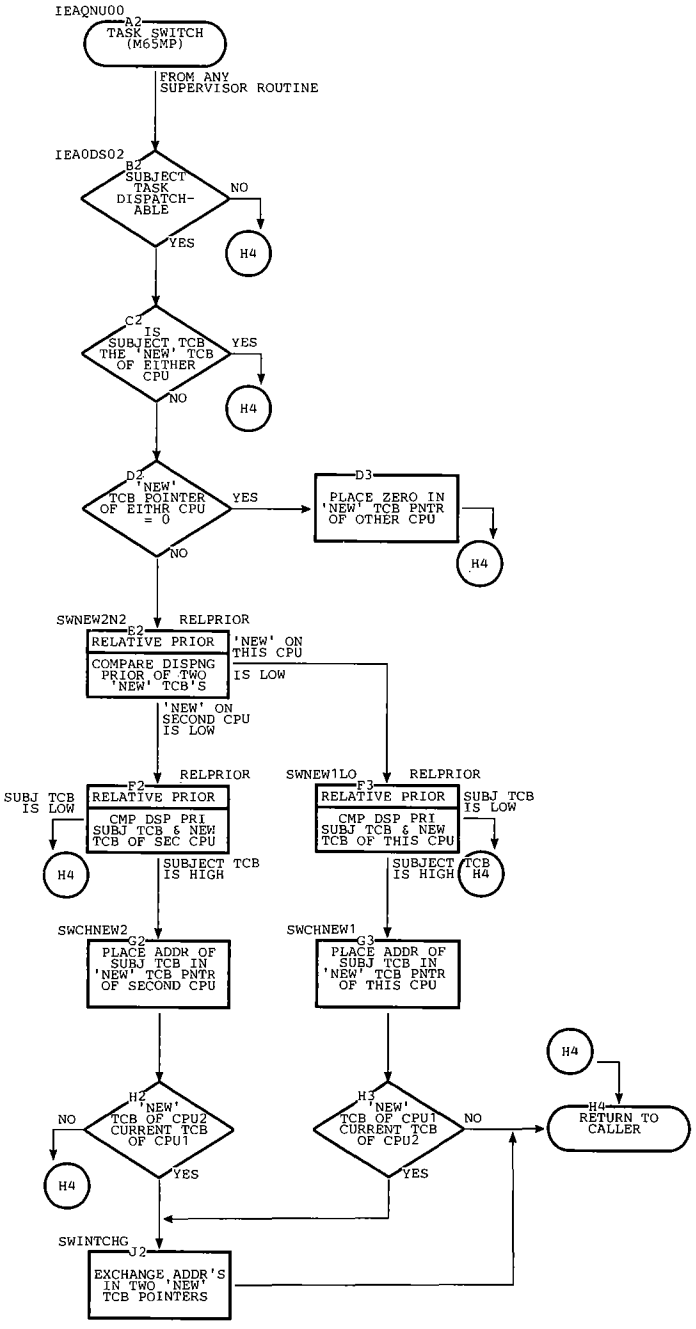


Chart BP. STAE Service Routine

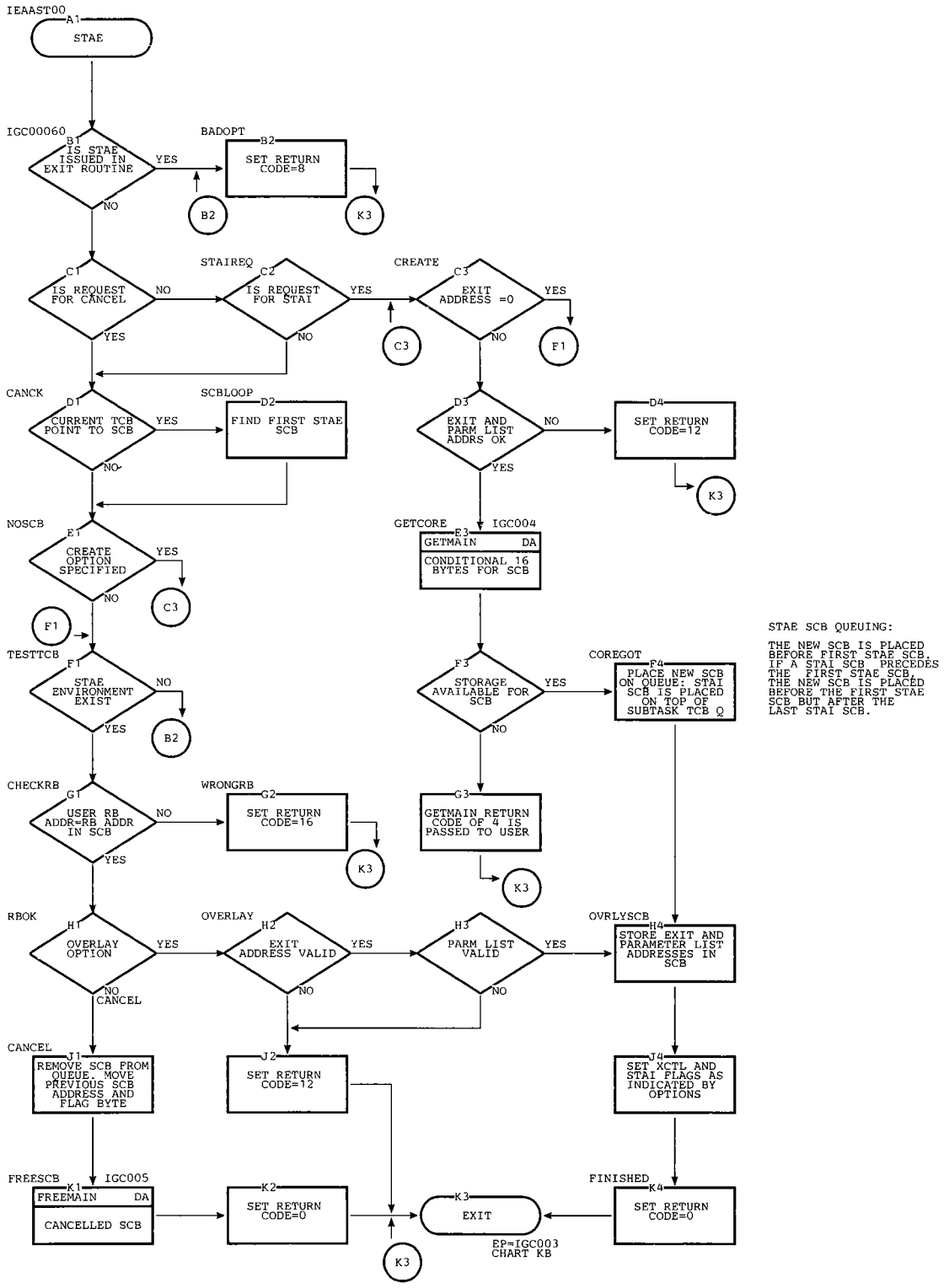


Chart BQ. ABEND/STAE Interface 0 Routine (ASIR0) (Page 1 of 2)

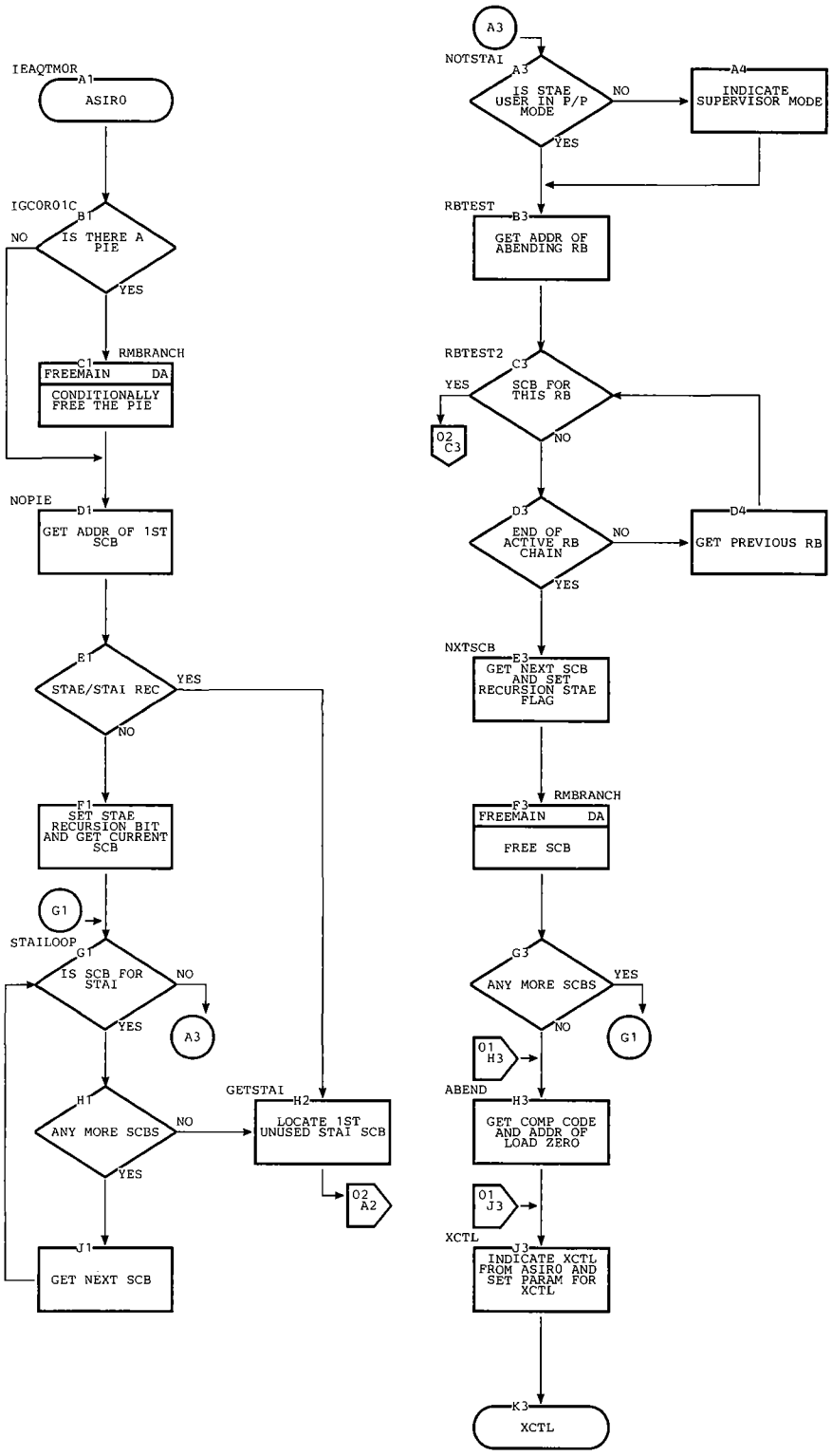


Chart BQ. ABEND/STAE Interface 0 Routine (ASIR0) (Page 2 of 2)

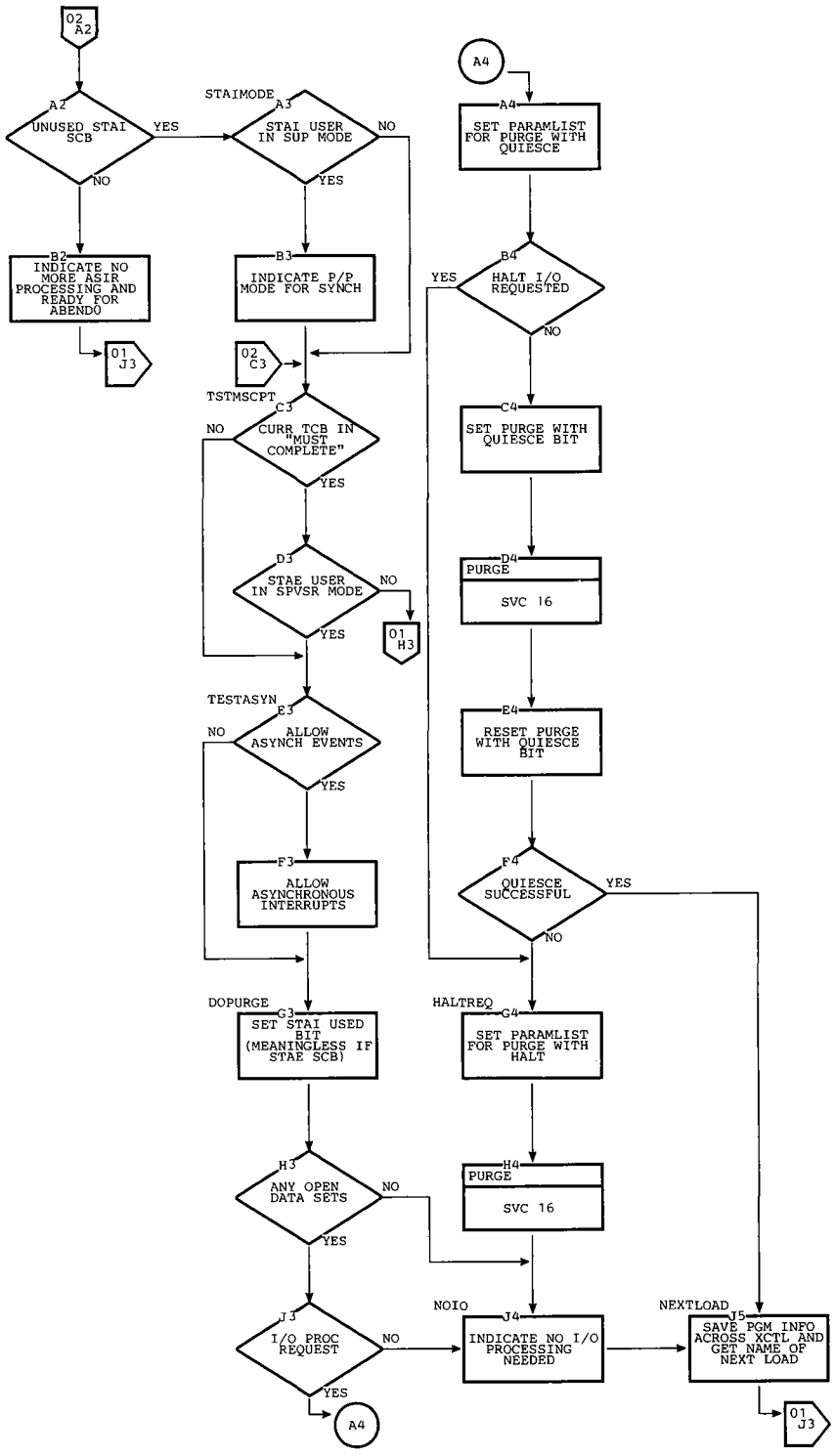


Chart BR. ABEND/STAE Interface 1 Routine (ASIR1) (Page 1 of 2)

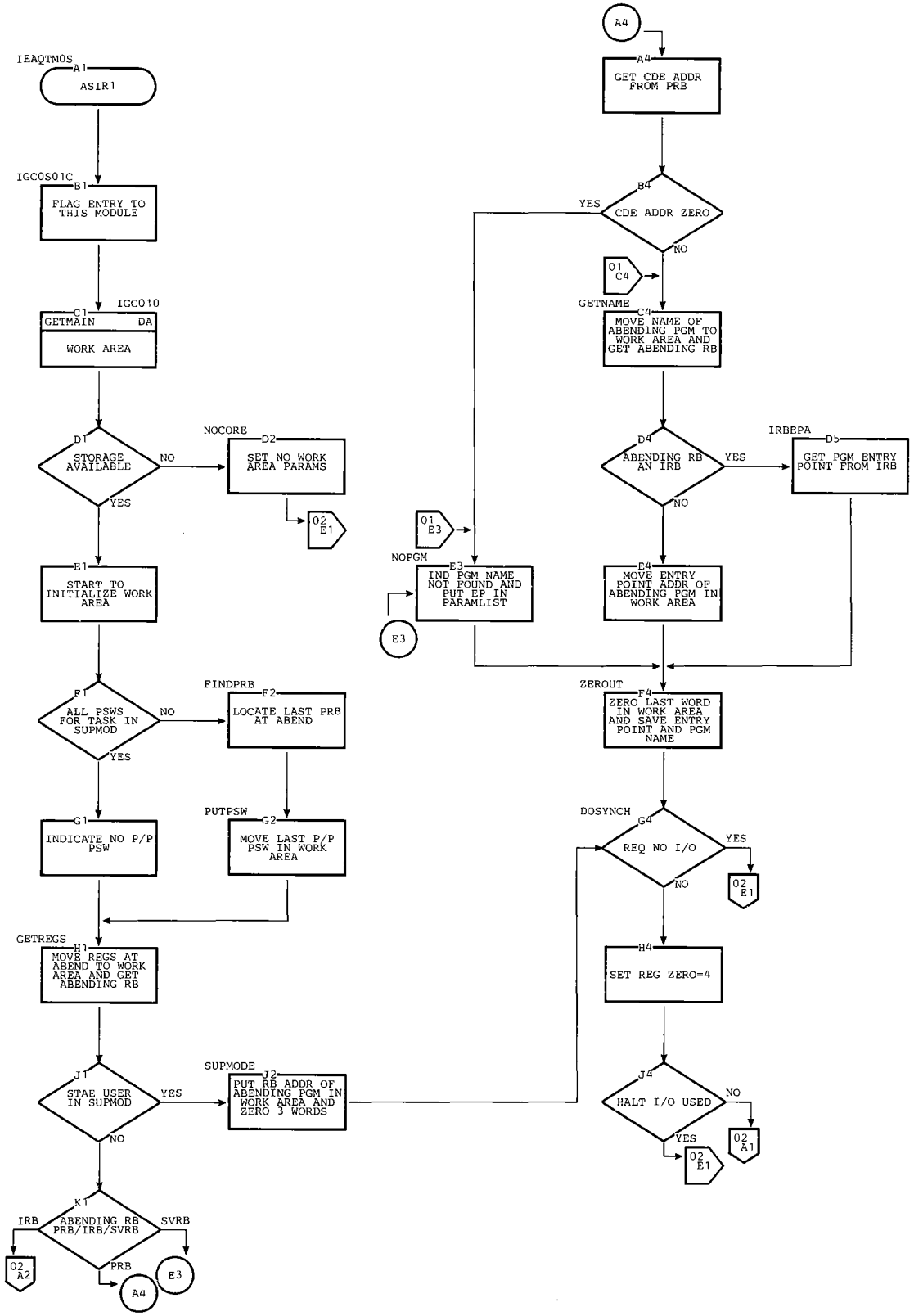


Chart BR. ABEND/STAE Interface 1 Routine (ASIR1) (Page 2 of 2)

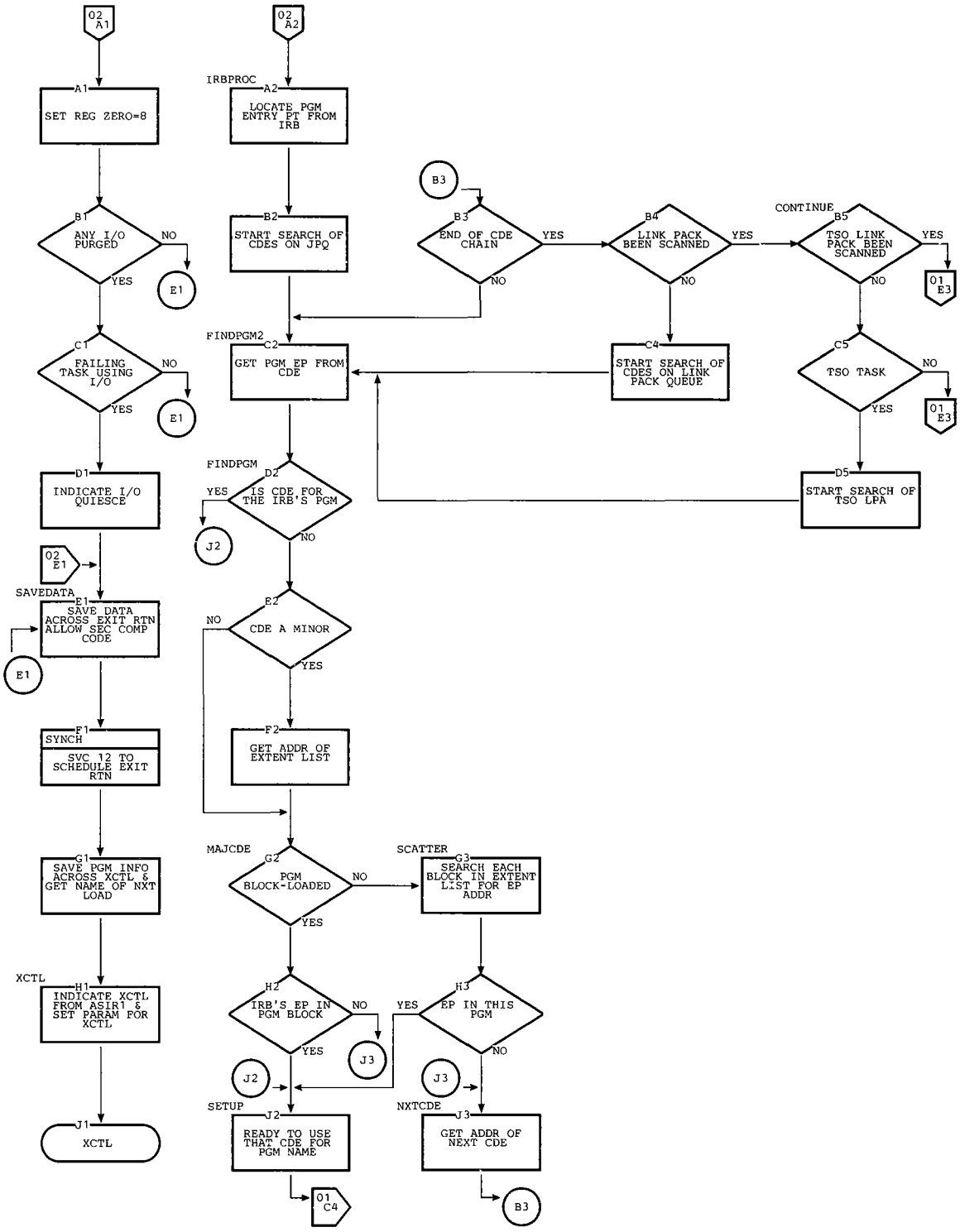


Chart BS. ABEND/STAE Interface 2 Routine (ASIR2)

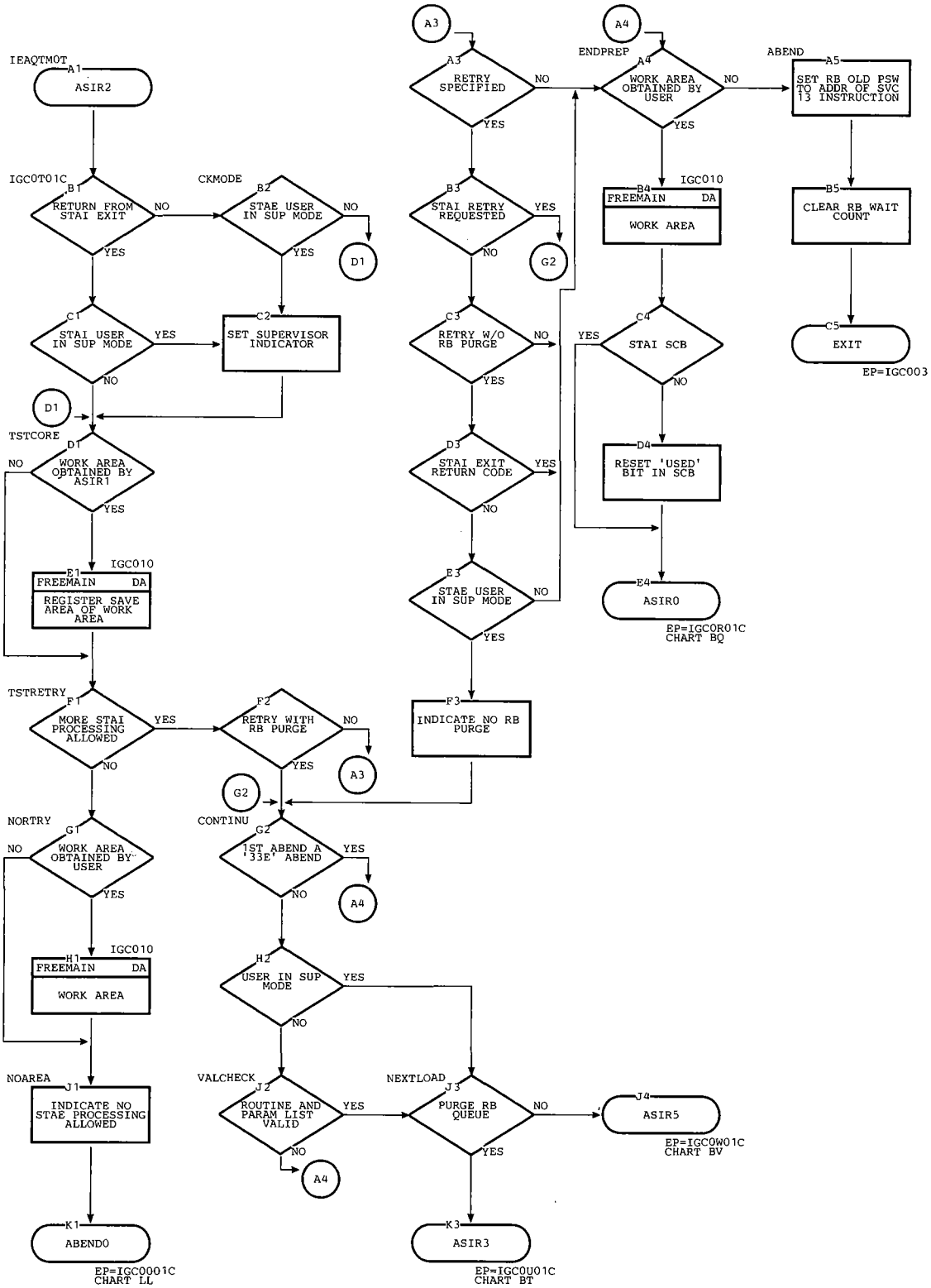


Chart BT. ABEND/STAE Interface 3 Routine (ASIR3) (Page 1 of 3)

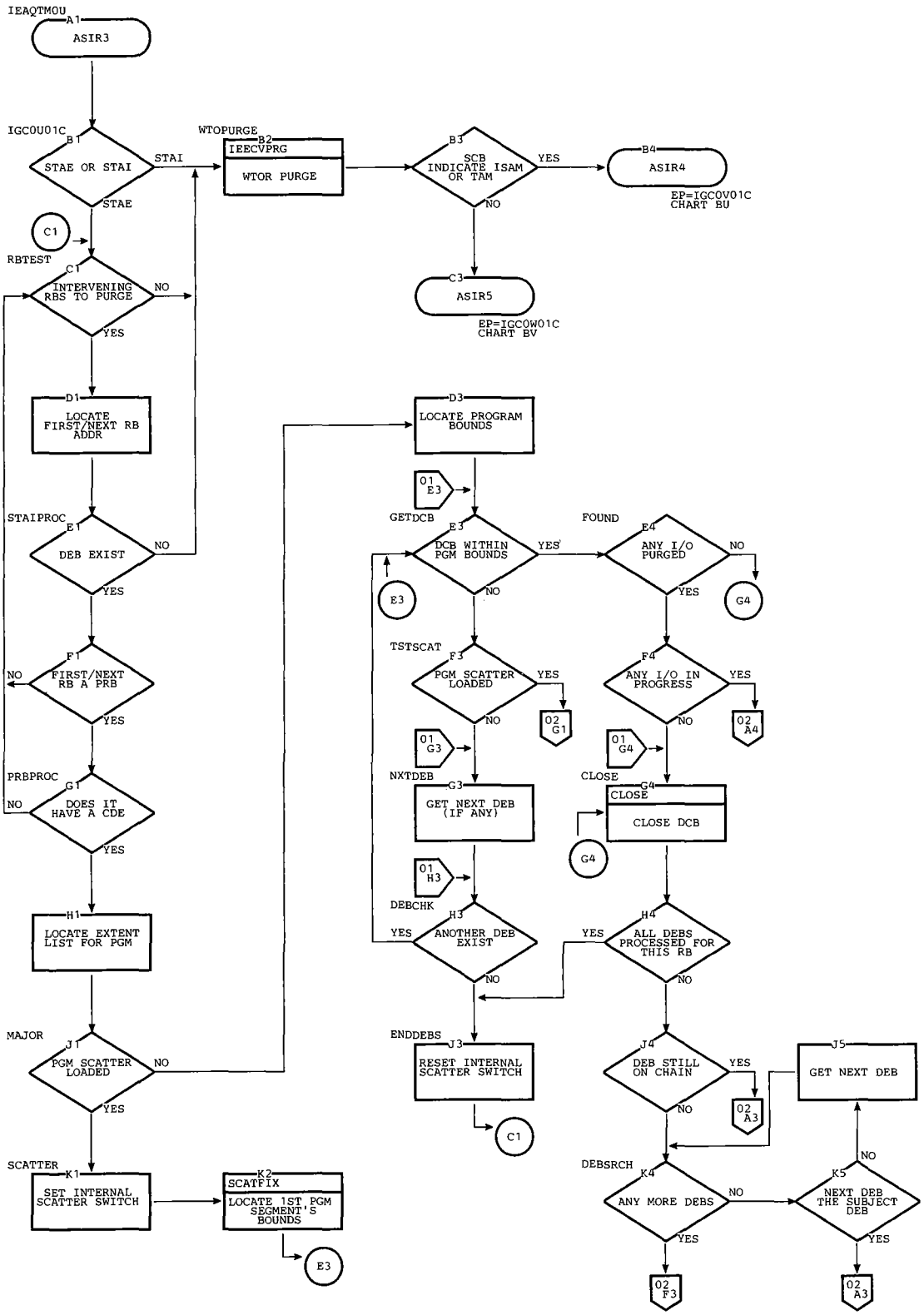


Chart BT. ABEND/STAE Interface 3 Routine (ASIR3) (Page 2 of 3)

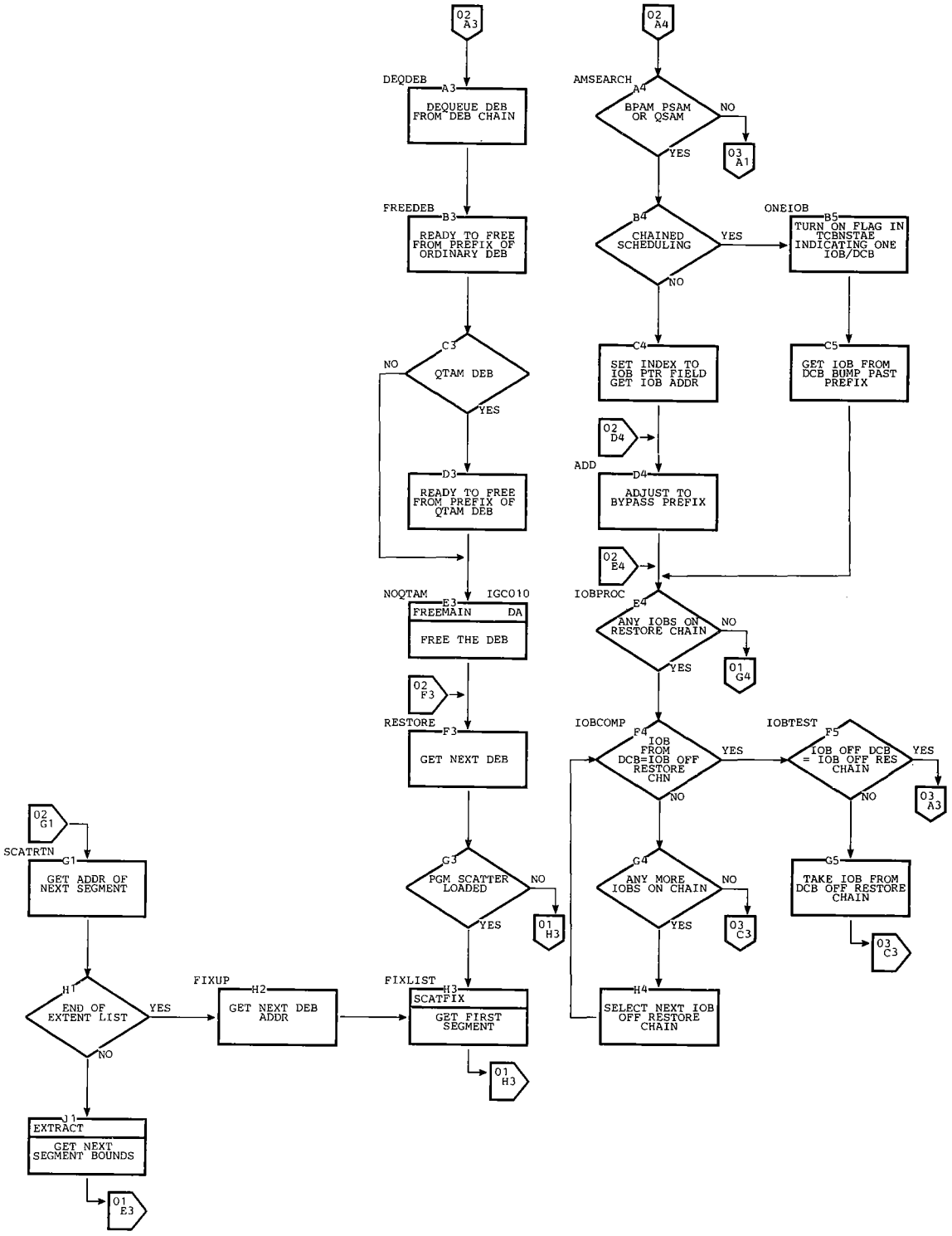


Chart BT. ABEND/STAE Interface 3 Routine (ASIR3) (Page 3 of 3)

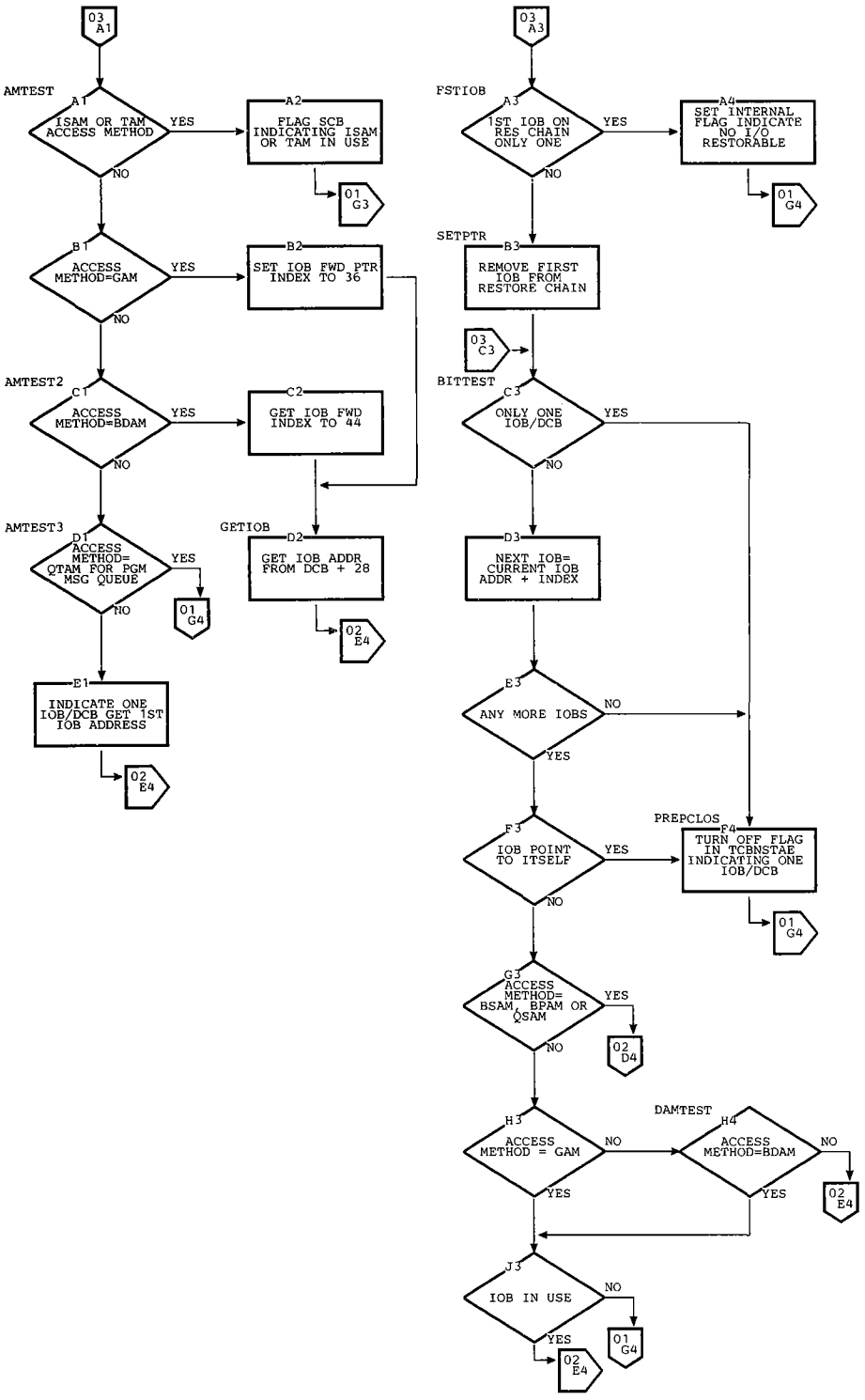


Chart BU. ABEND/STAE Interface 4 Routine (ASIR4) (Page 1 of 4)

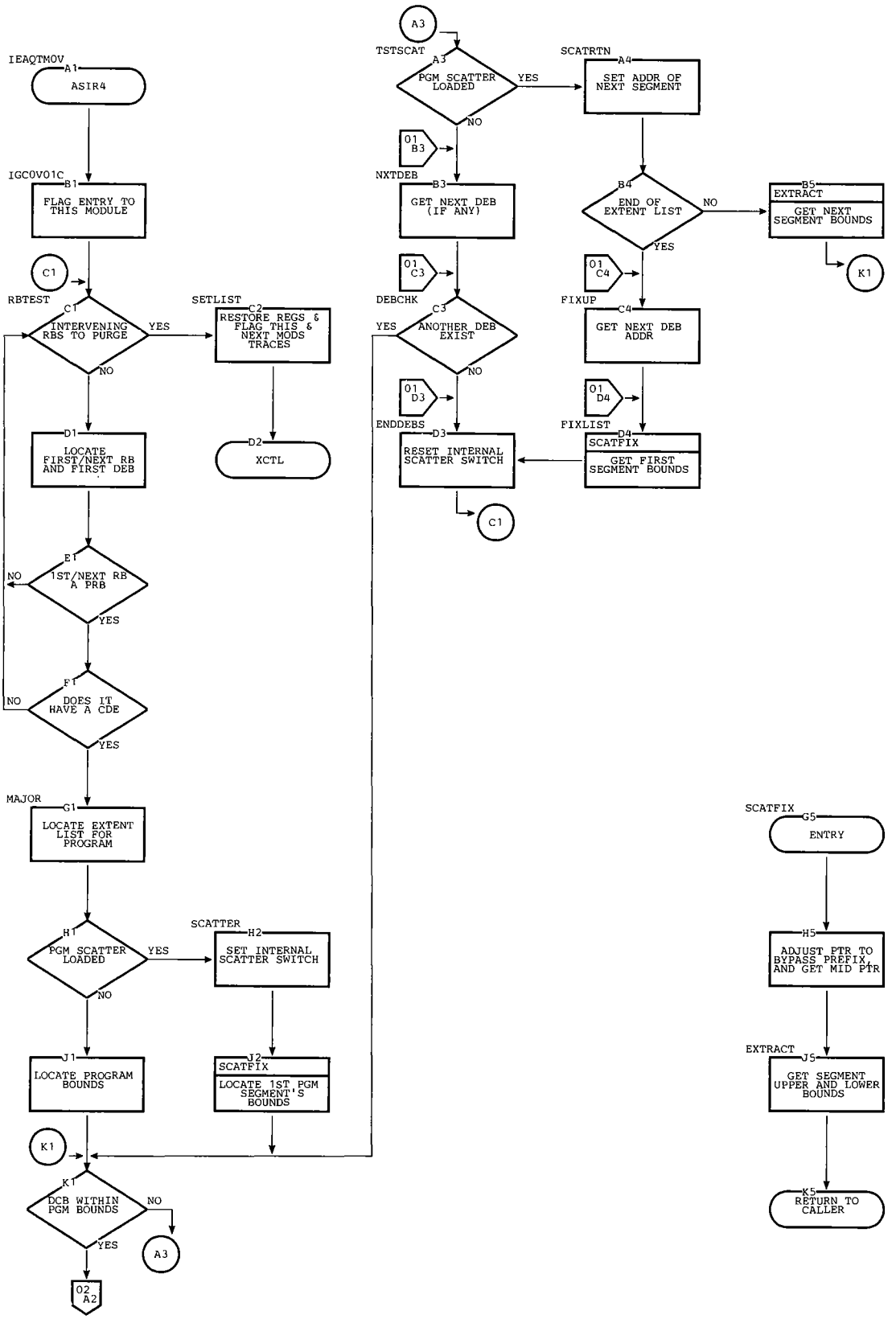


Chart BU. ABEND/STAE Interface 4 Routine (ASIR4) (Page 2 of 4)

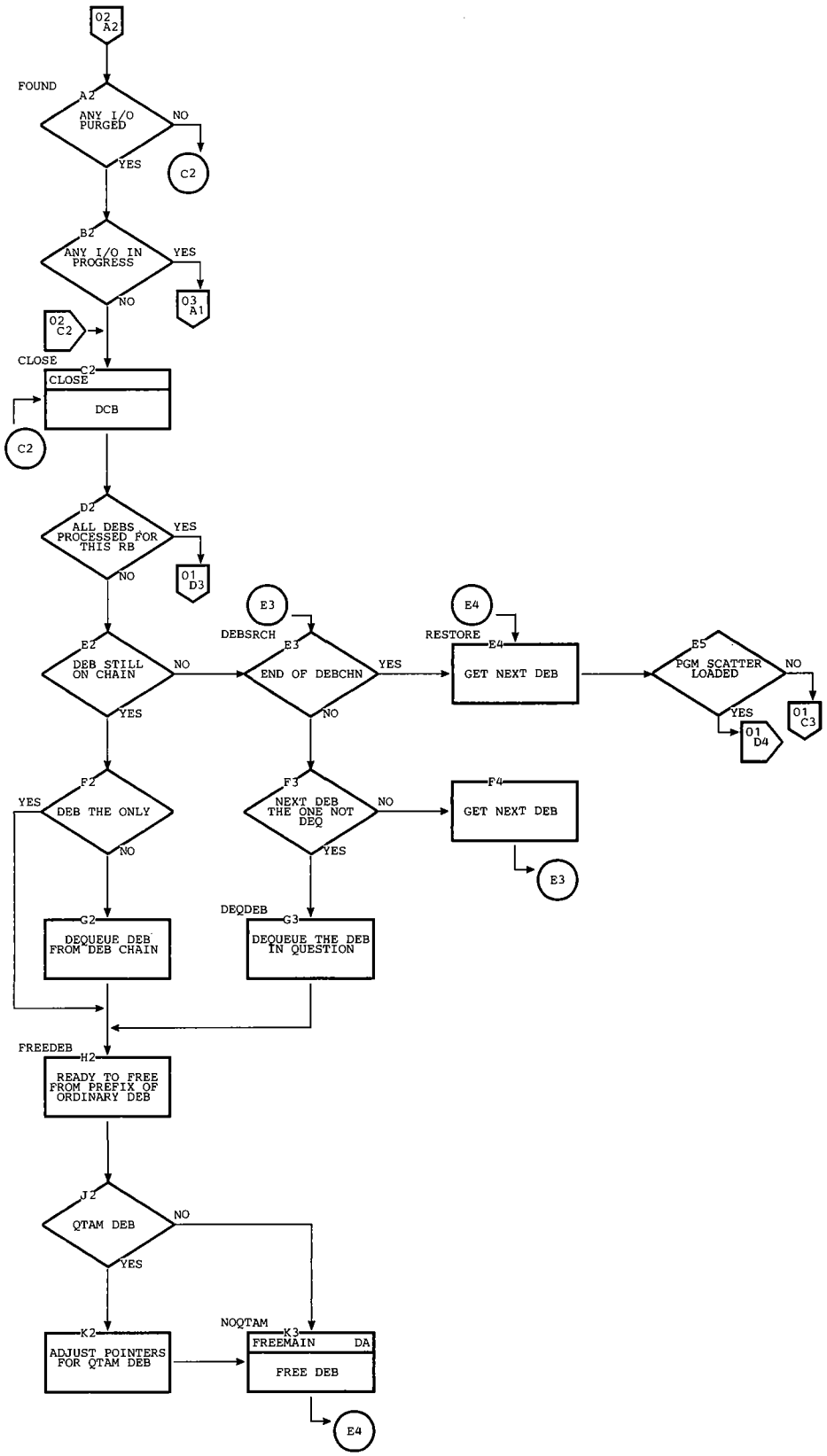


Chart BU. ABEND/STAE Interface 4 Routine (ASIR4) (Page 3 of 4)

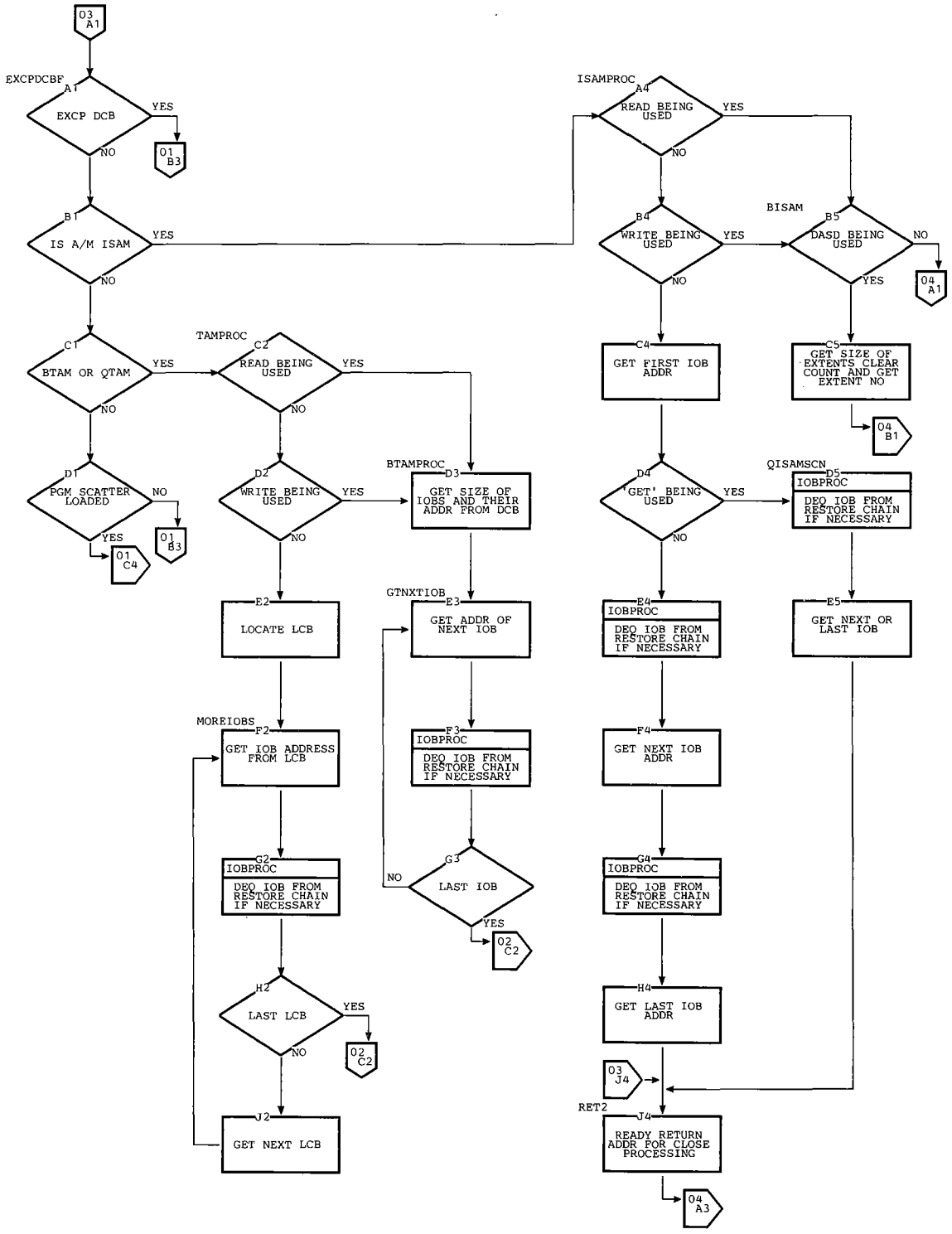


Chart BU. ABEND/STAE Interface 4 Routine (ASIR4) (Page 4 of 4)

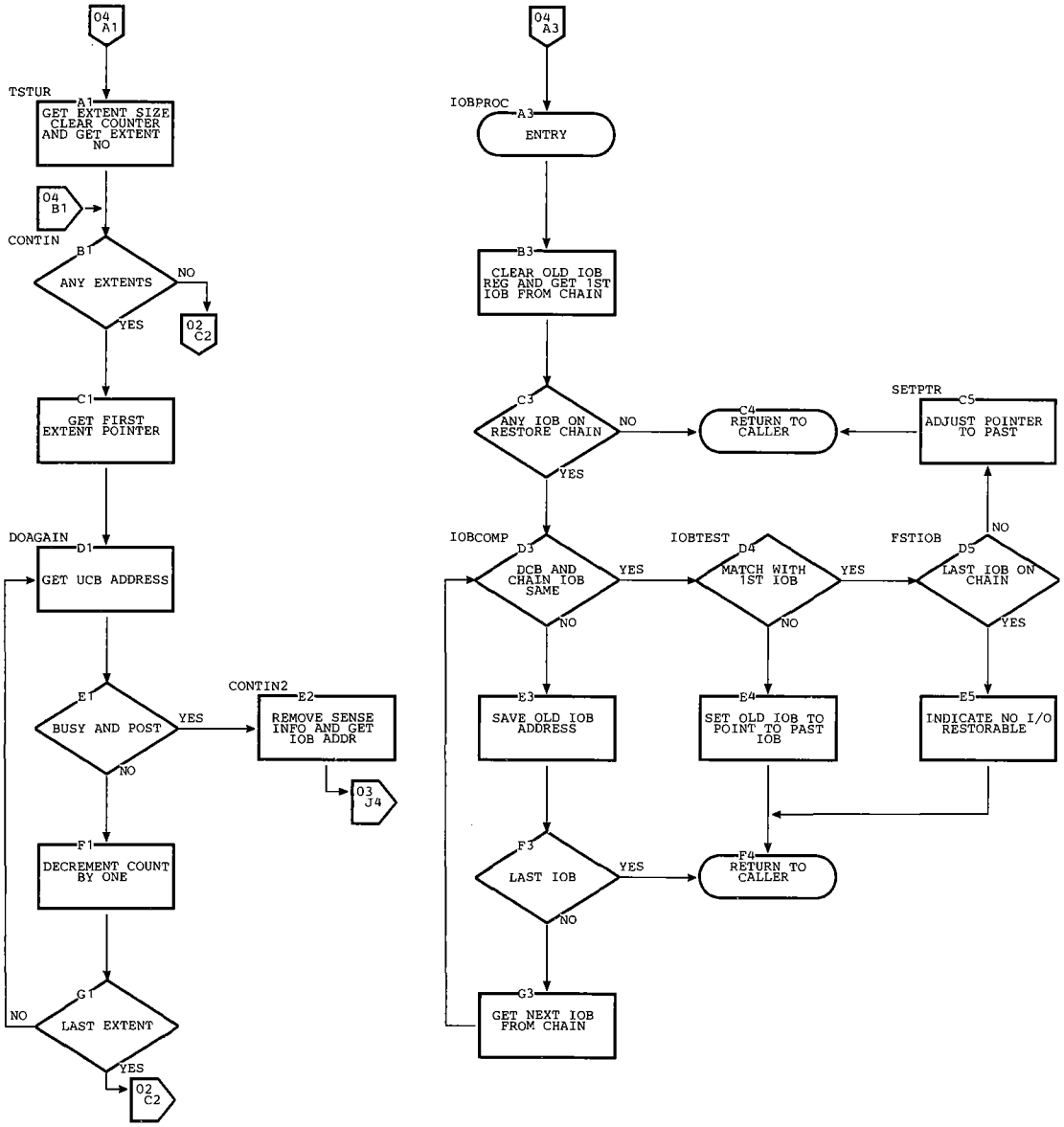


Chart BV. ABEND/STAE Interface 5 Routine (ASIR5)

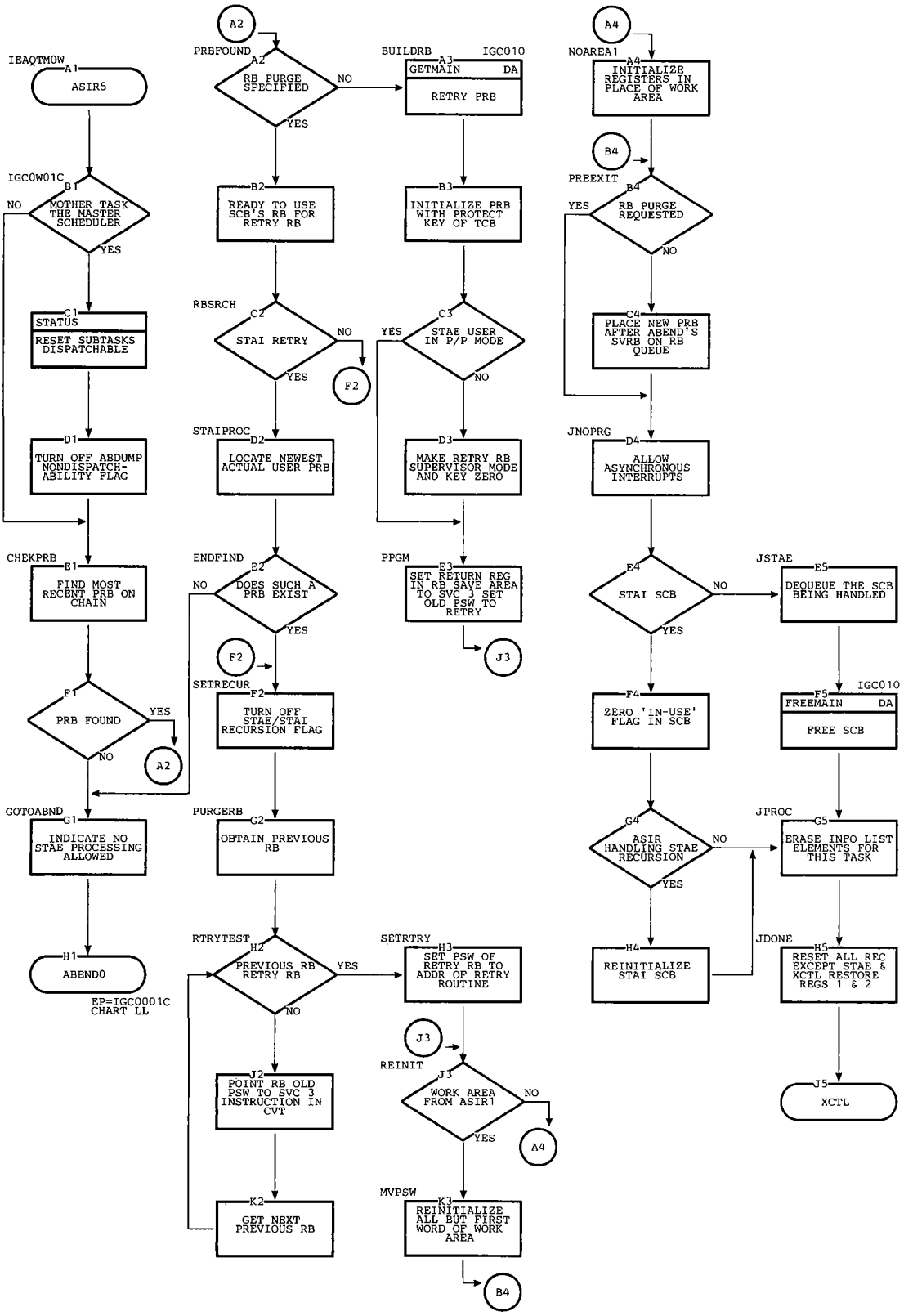


Chart BW. Set Status Service Routine (Page 1 of 2)

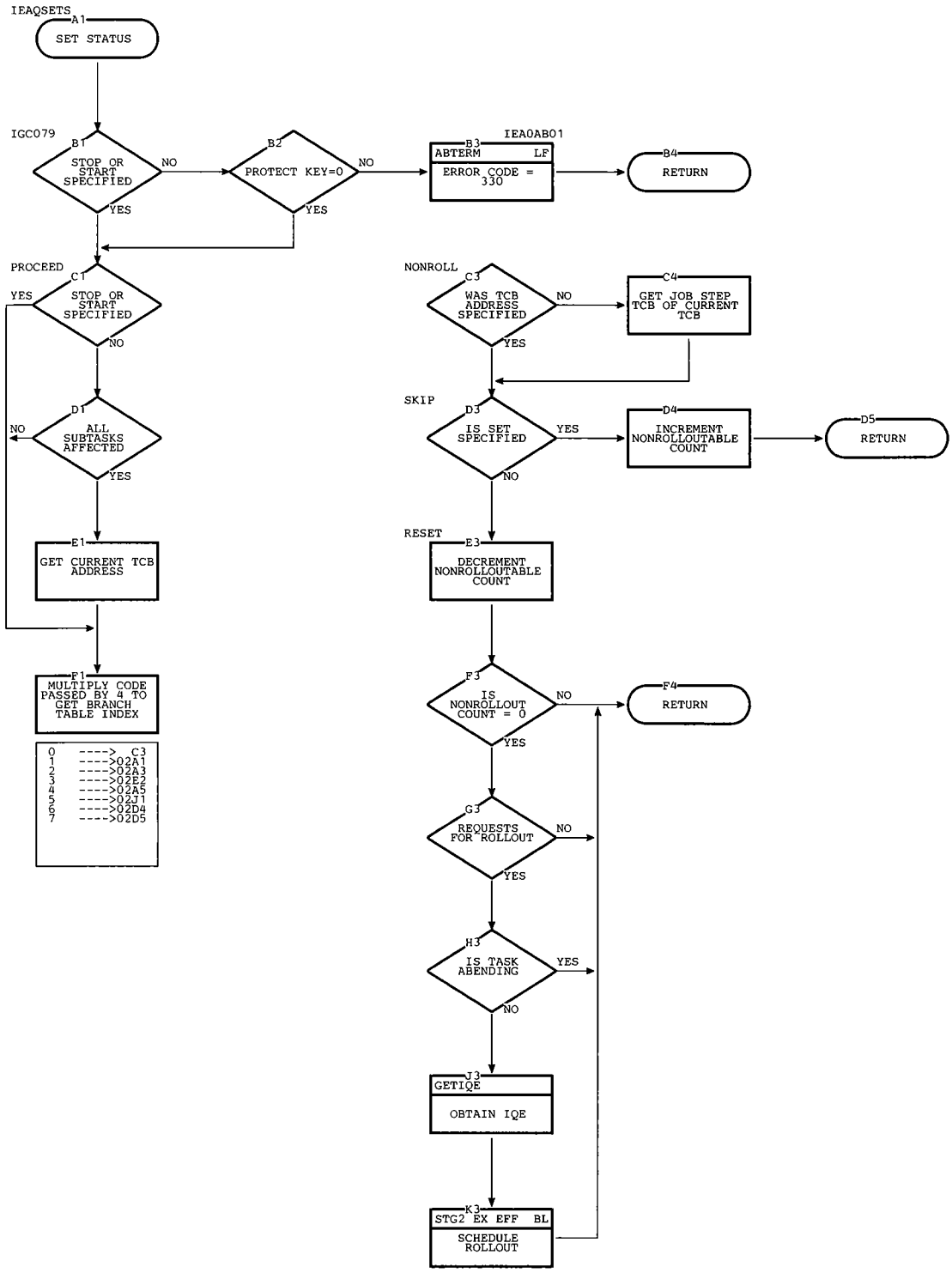


Chart BW. Set Status Service Routine (Page 2 of 2)

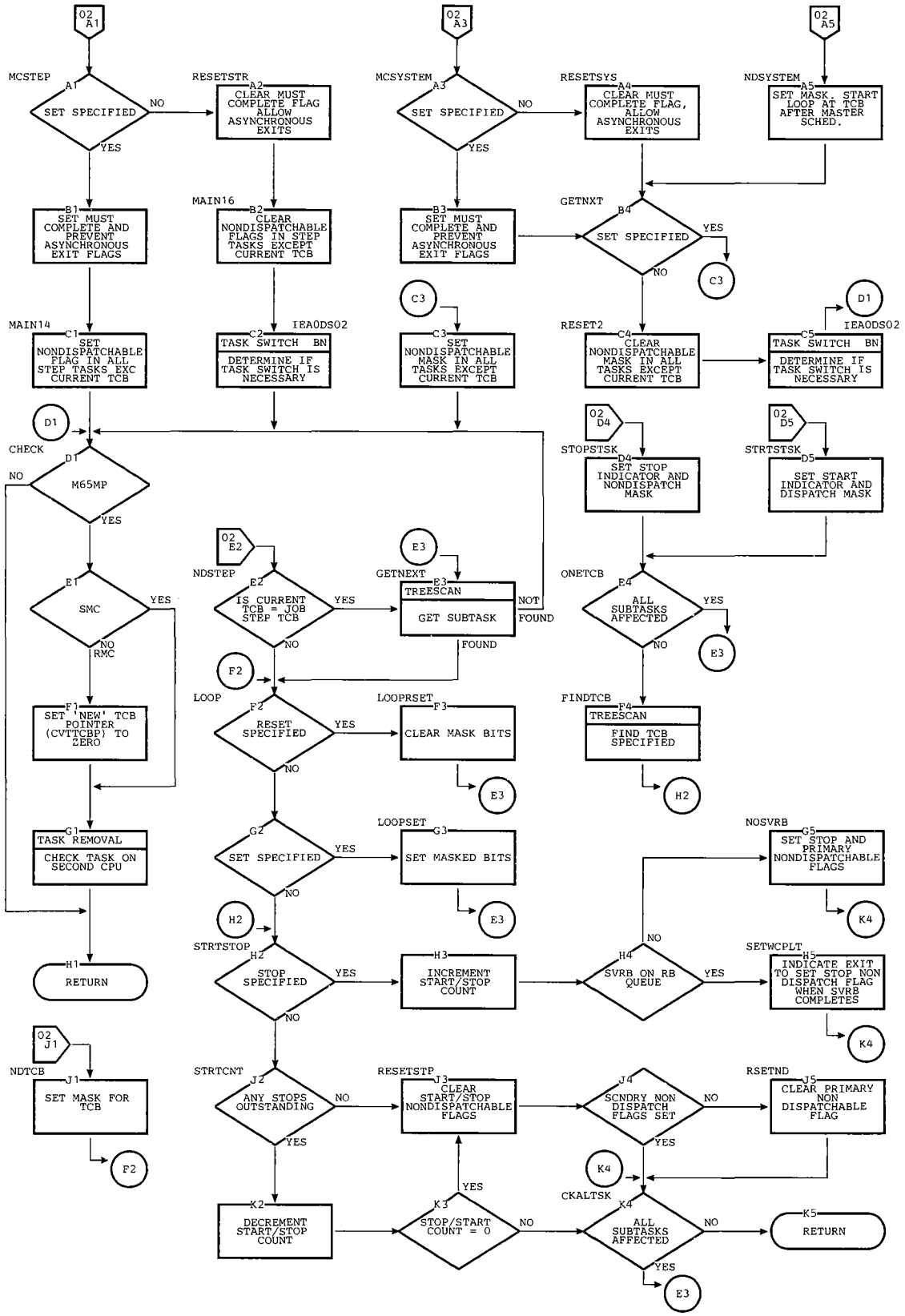


Chart CA. Link, Load, XCTL, and SYNCH Processing (Page 1 of 3)

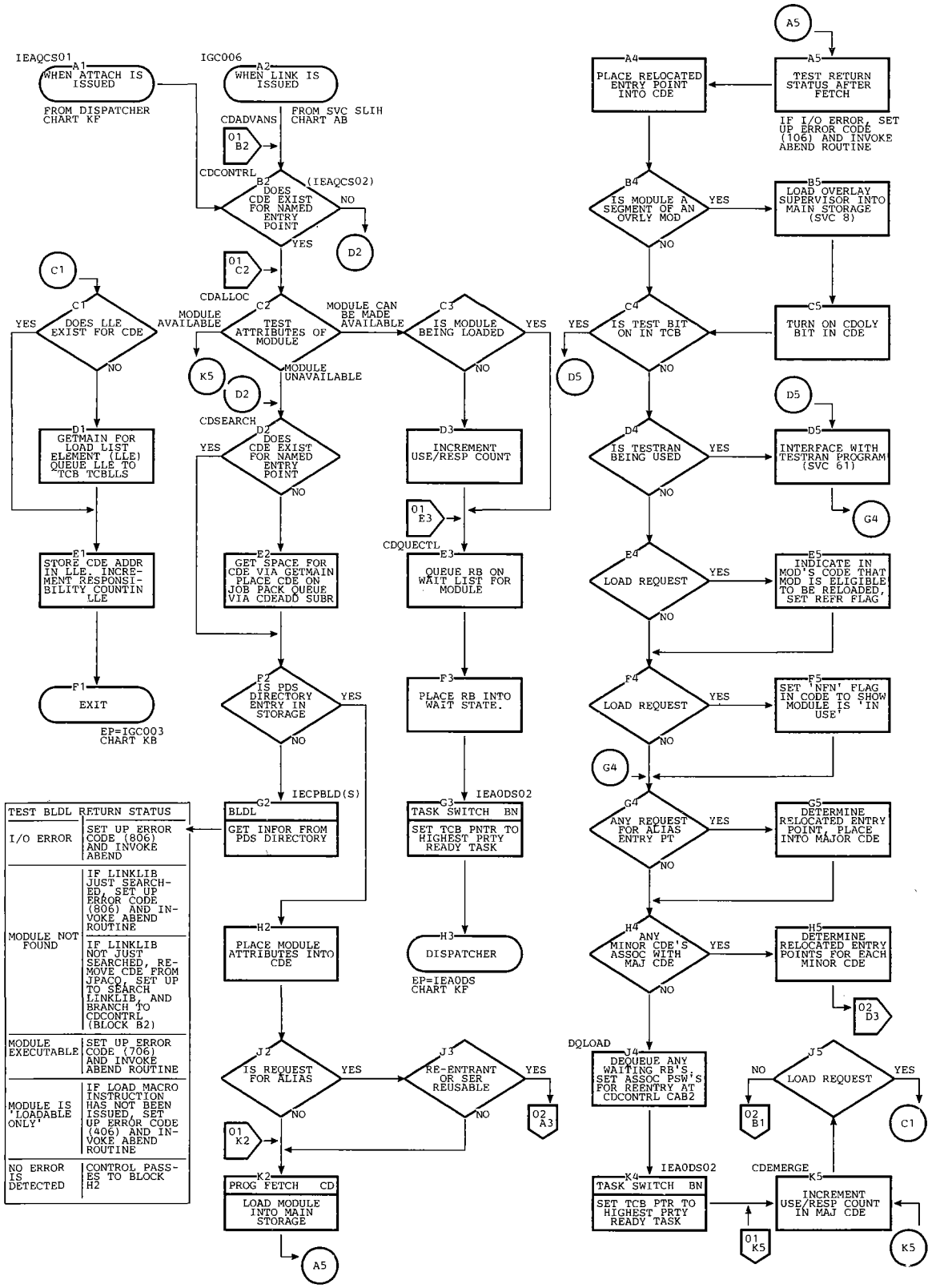


Chart CA. Link, Load, XCTL, and SYNCH Processing (Page 2 of 3)

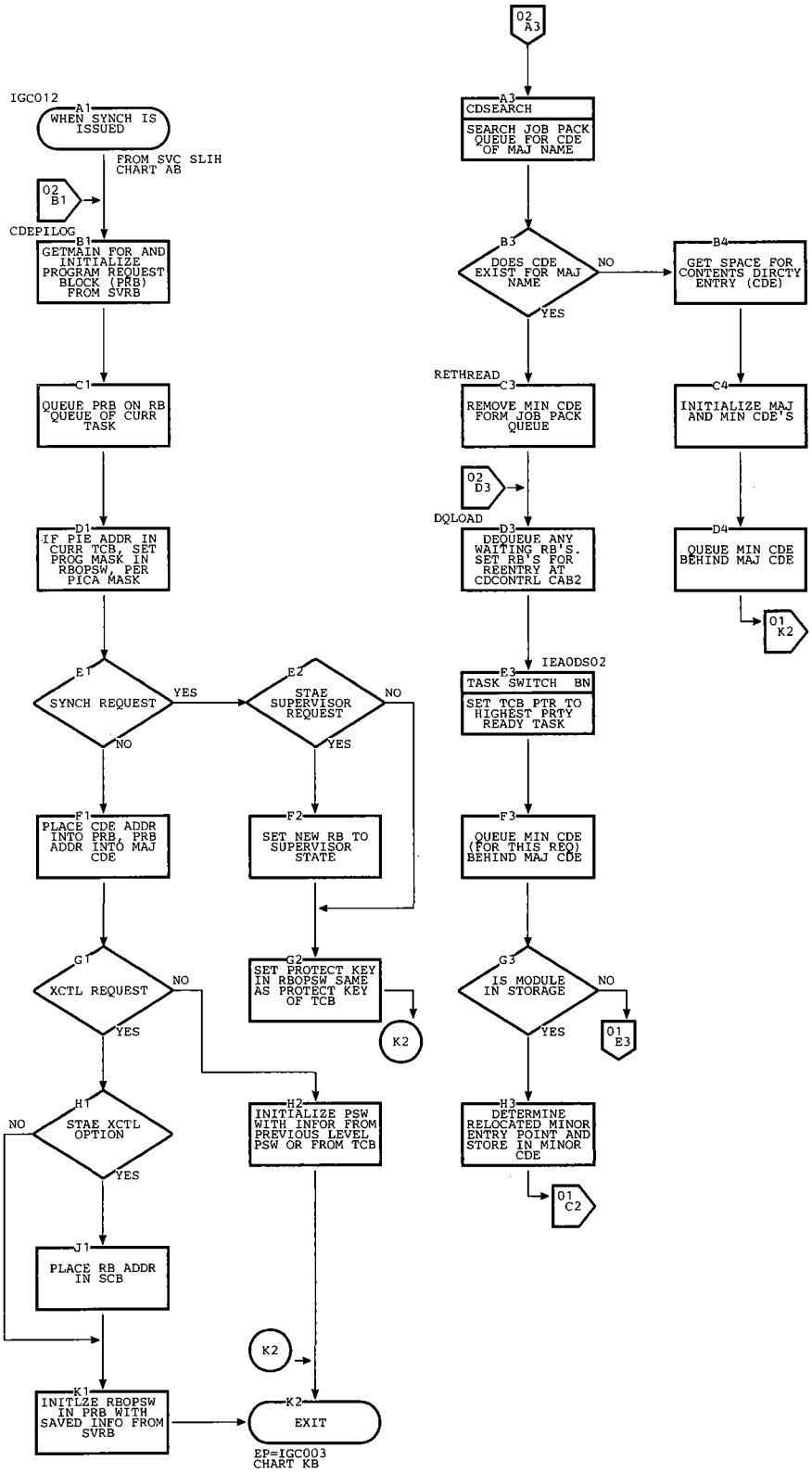


Chart CA. Link, Load, XCTL, and SYNCH Processing (Page 3 of 3)

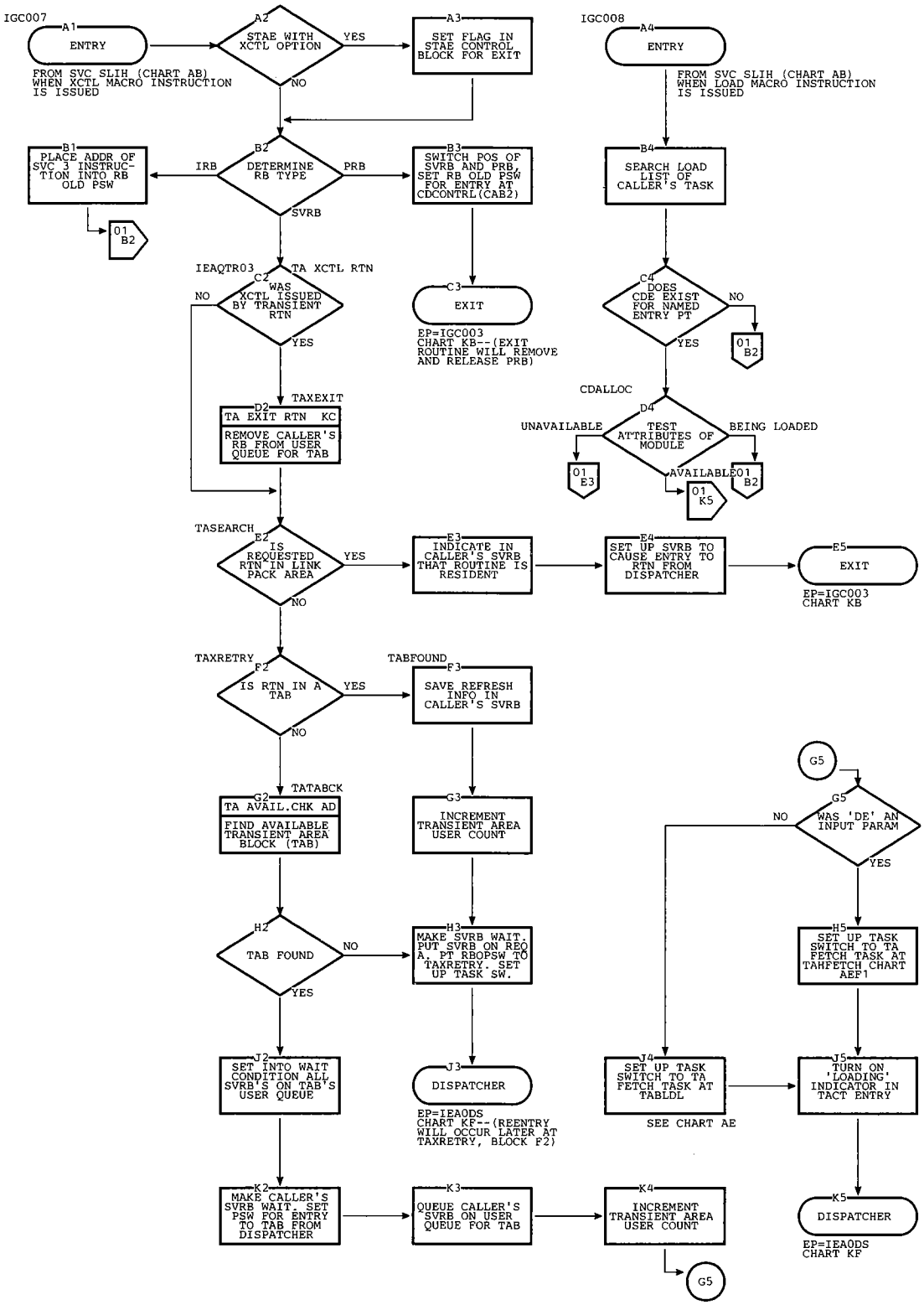


Chart CB. Identify Routine (Page 1 of 2)

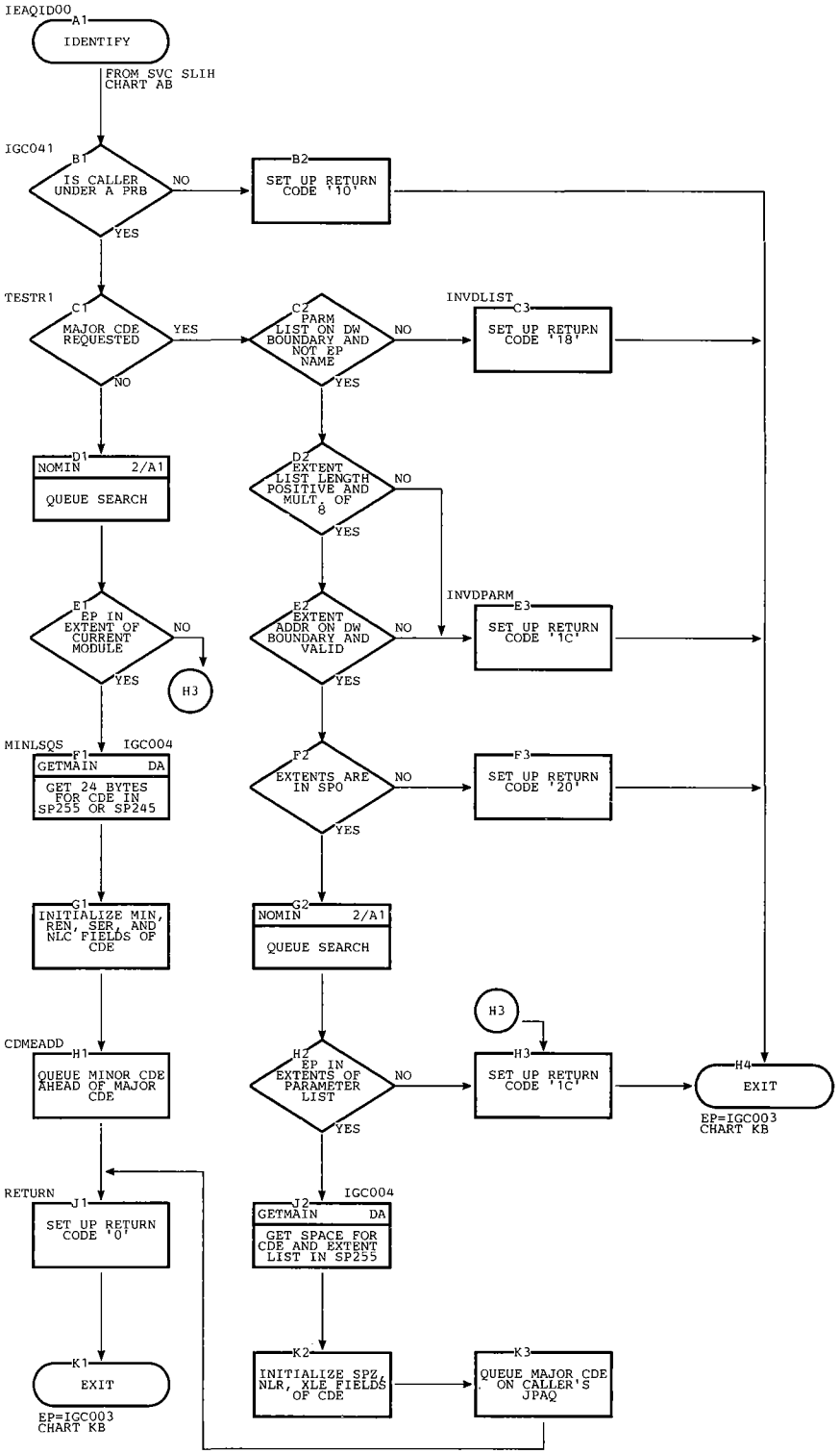


Chart CB. Identify Routine (Page 2 of 2)

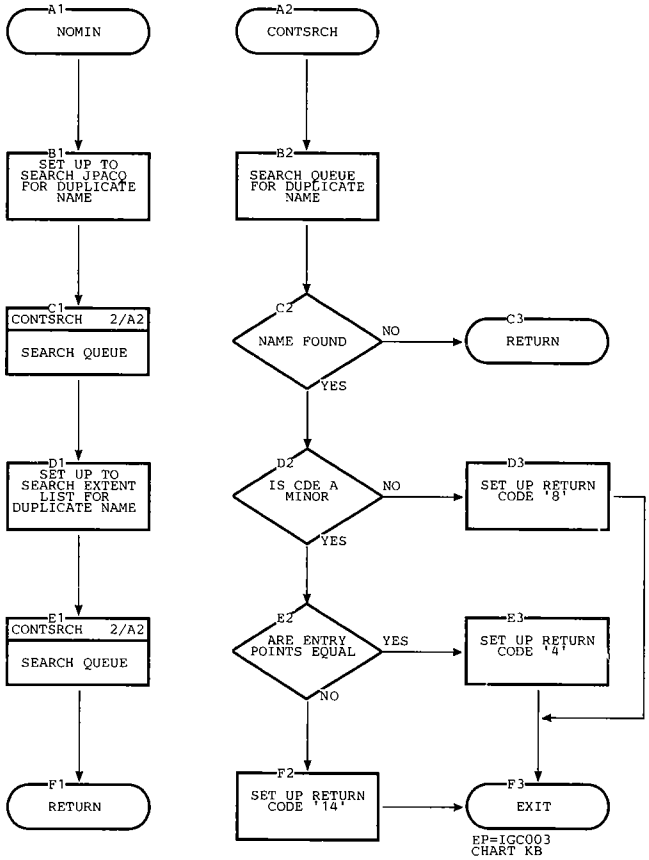


Chart CC. Delete Routine

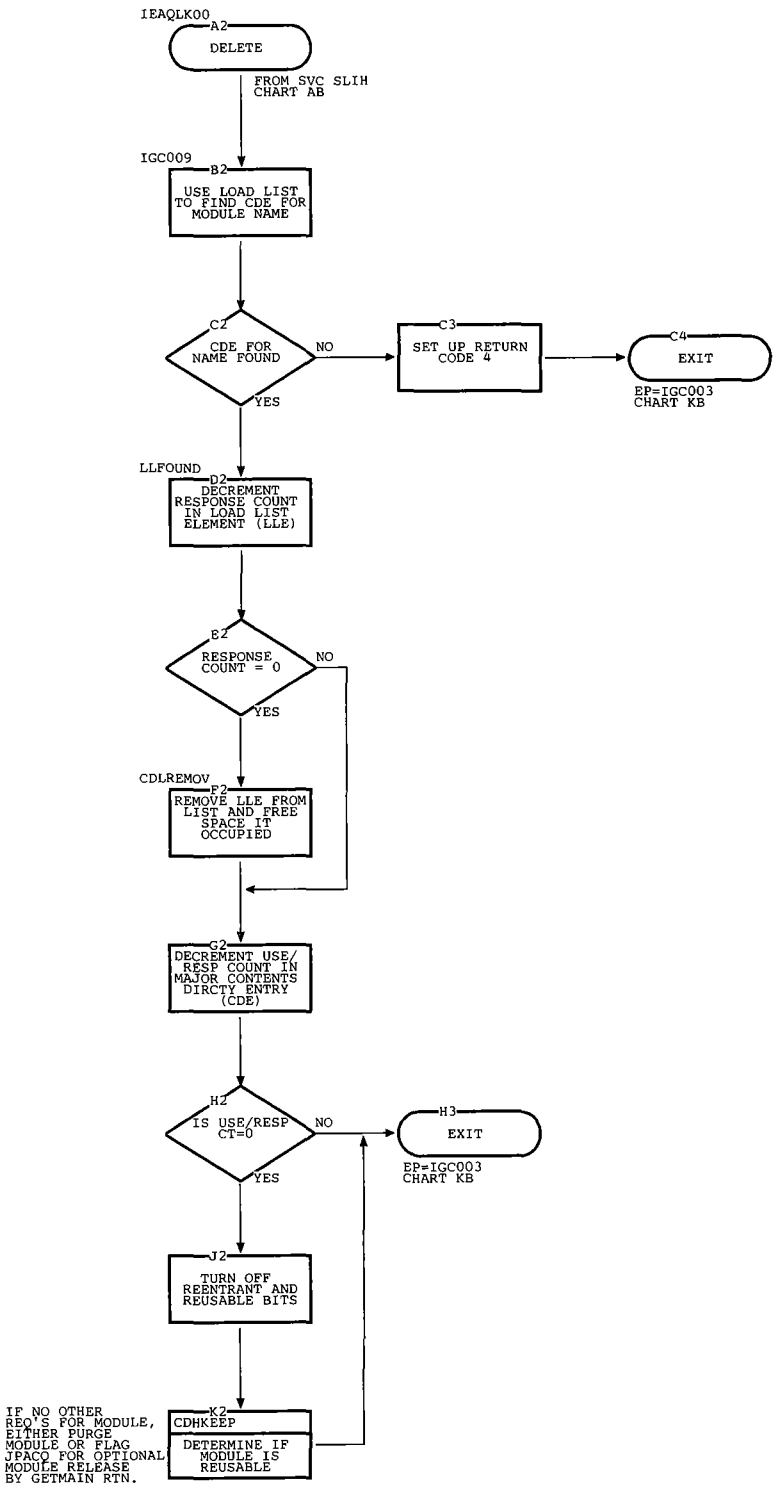


Chart CD. Program Fetch Routine (Page 1 of 3)

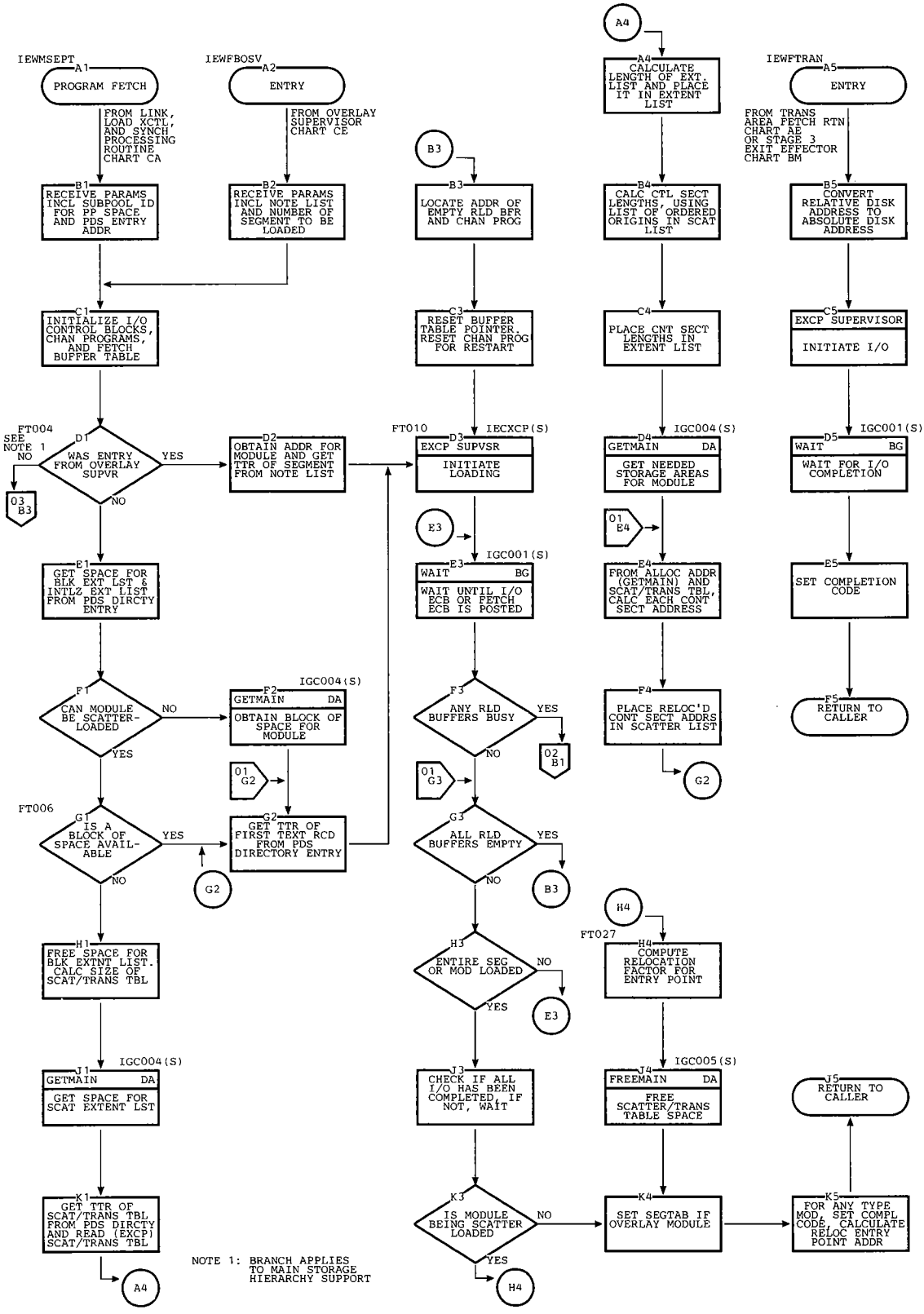


Chart CD. Program Fetch Routine -- PCI and Channel End Appendages (Page 2 of 3)

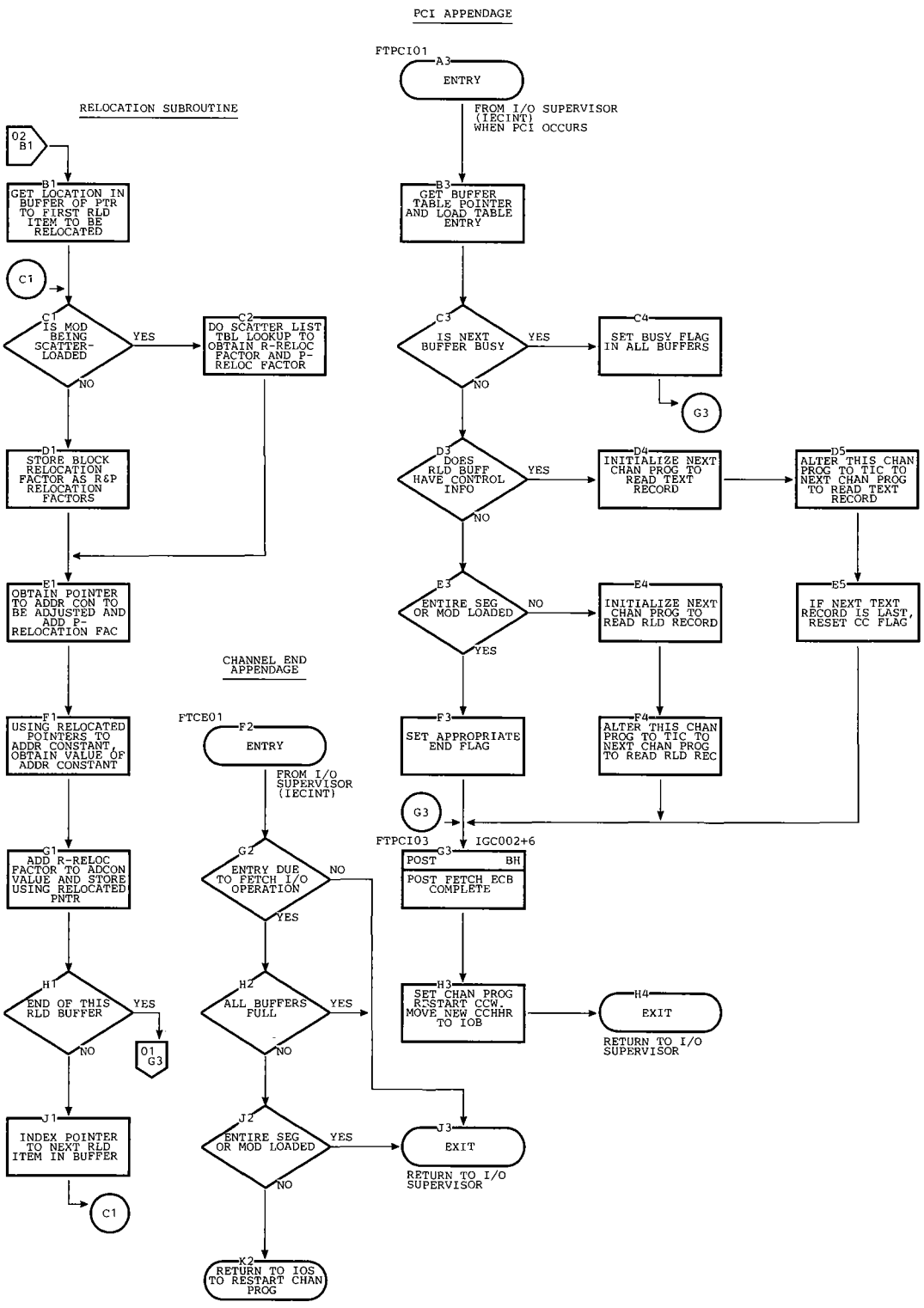


Chart CD. Program Fetch Routine -- with Main Storage Hierarchies (Page 3 of 3)

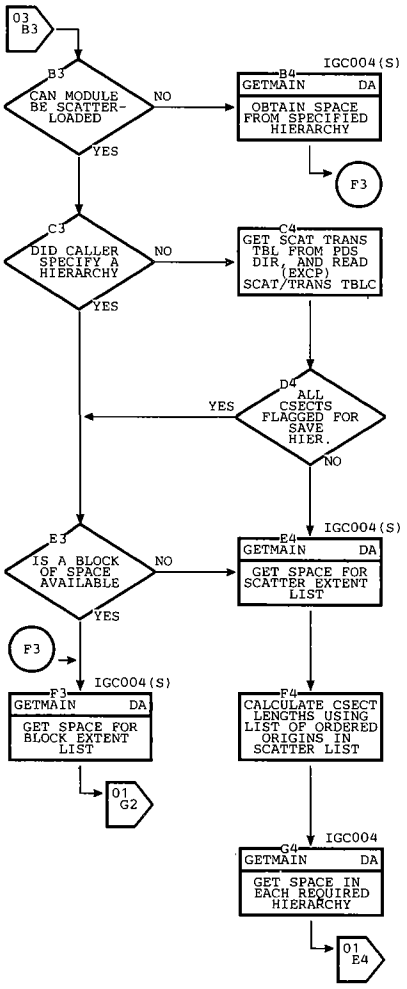


Chart CE. Overlay Supervision

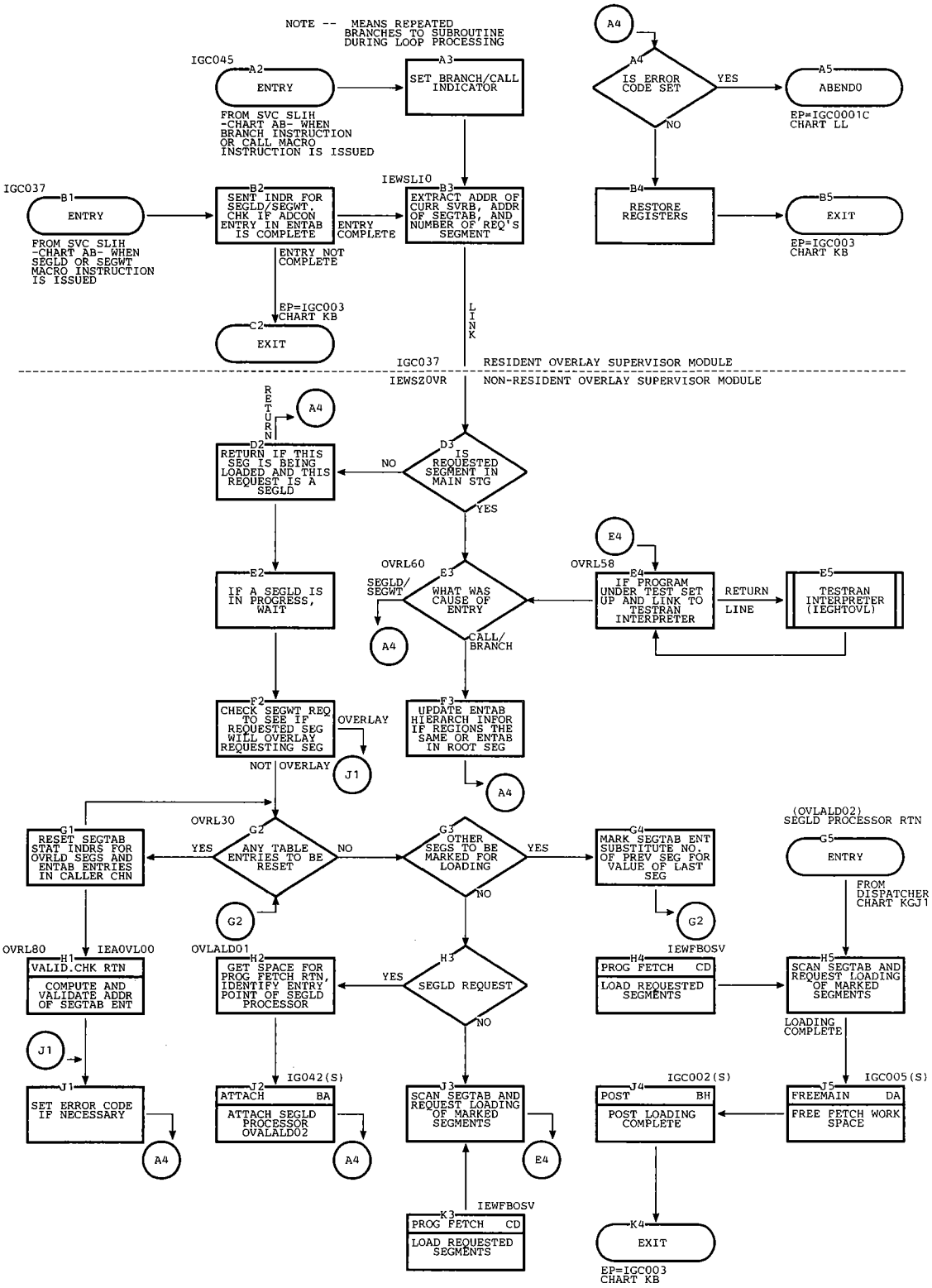


Chart DA. GETMAIN/FREEMAIN Routine (Page 1 of 3)

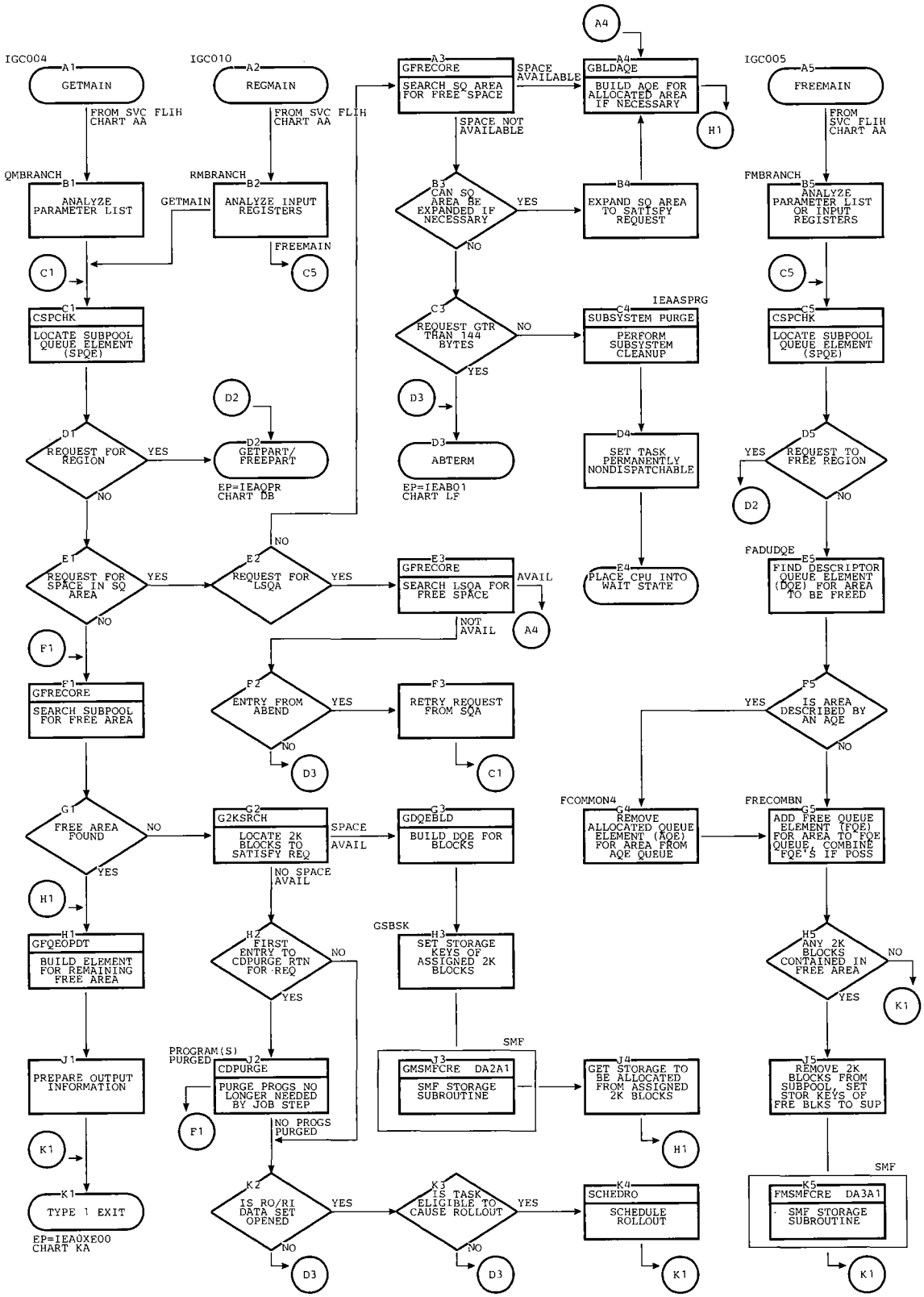


Chart DA. GETMAIN/FREEMAIN -- SMF Storage Routine (Page 2 of 3)

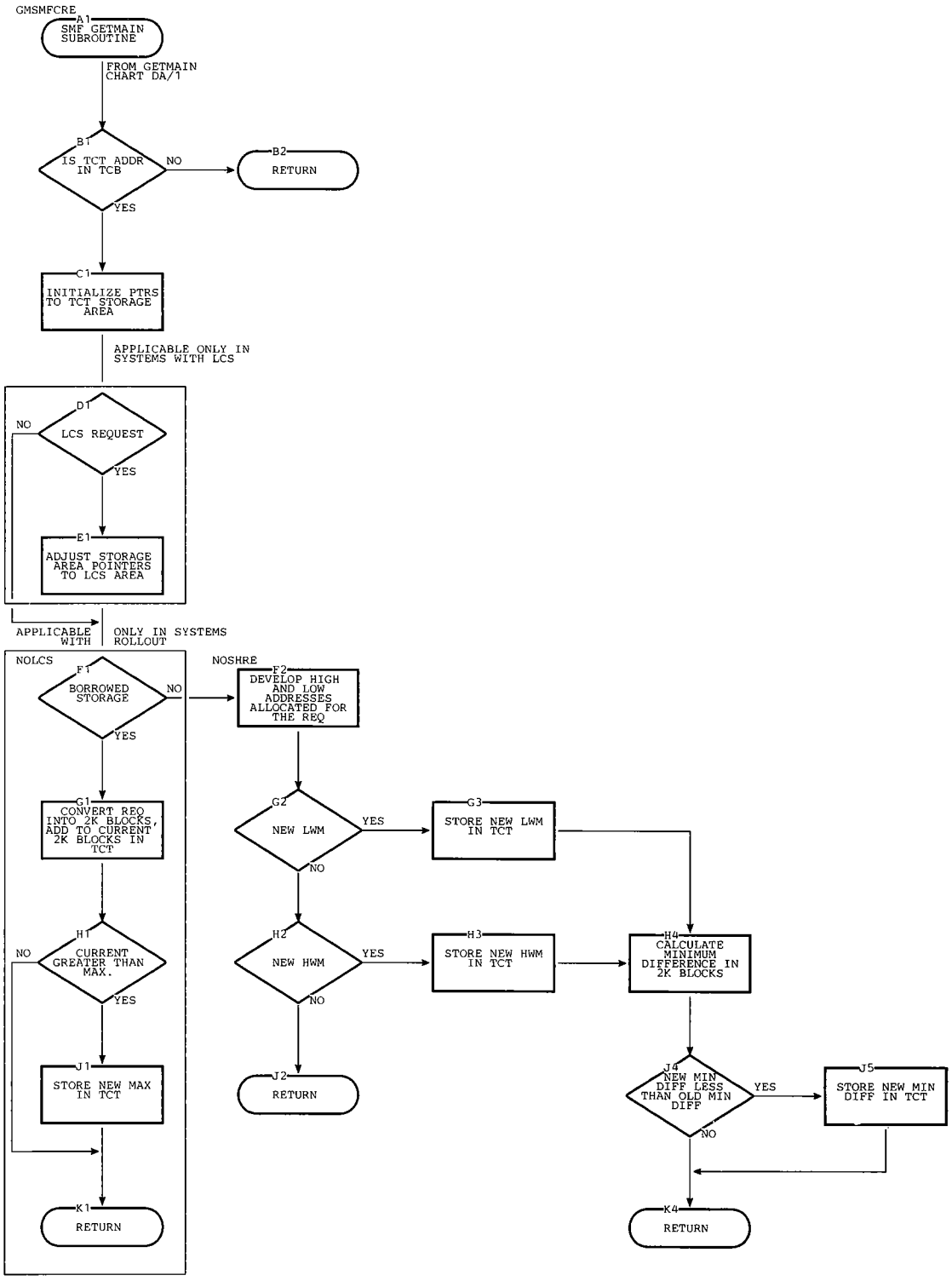


Chart DA. GETMAIN/FREEMAIN -- SMF Storage Routine (Page 3 of 3)

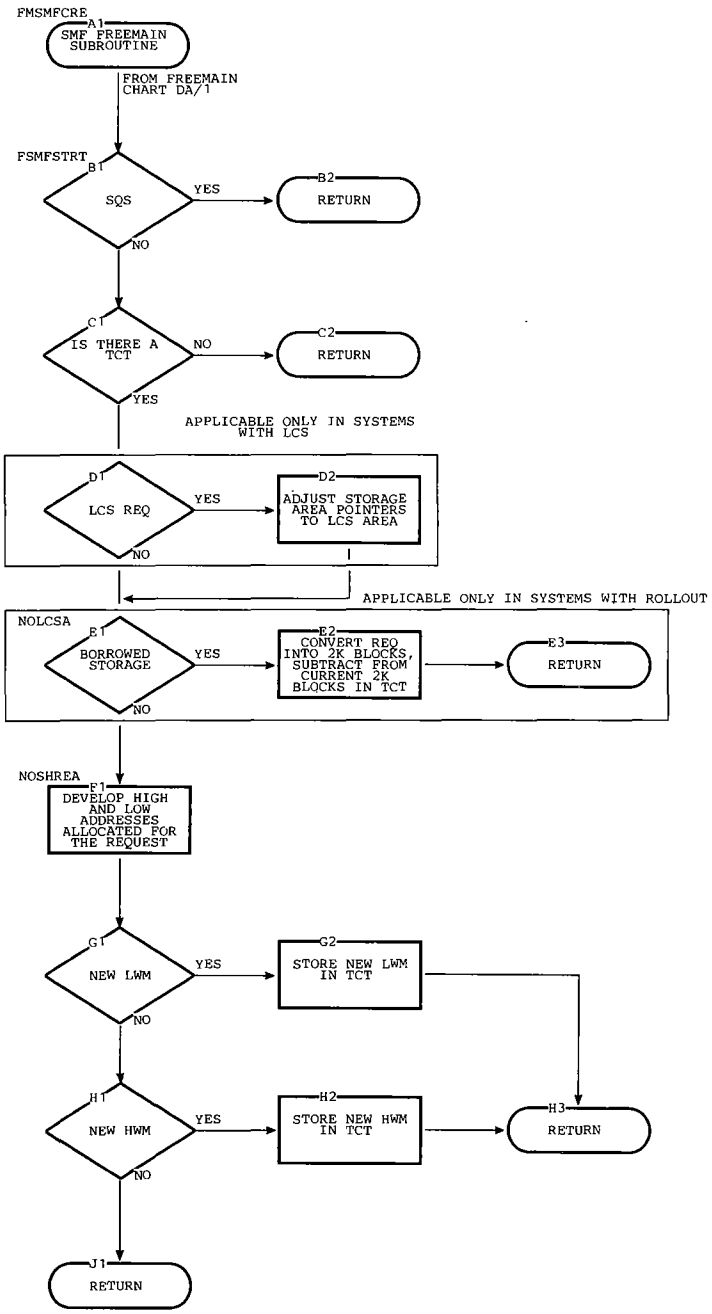


Chart DB. GETPART/FREEPART Routine (Page 1 of 2)

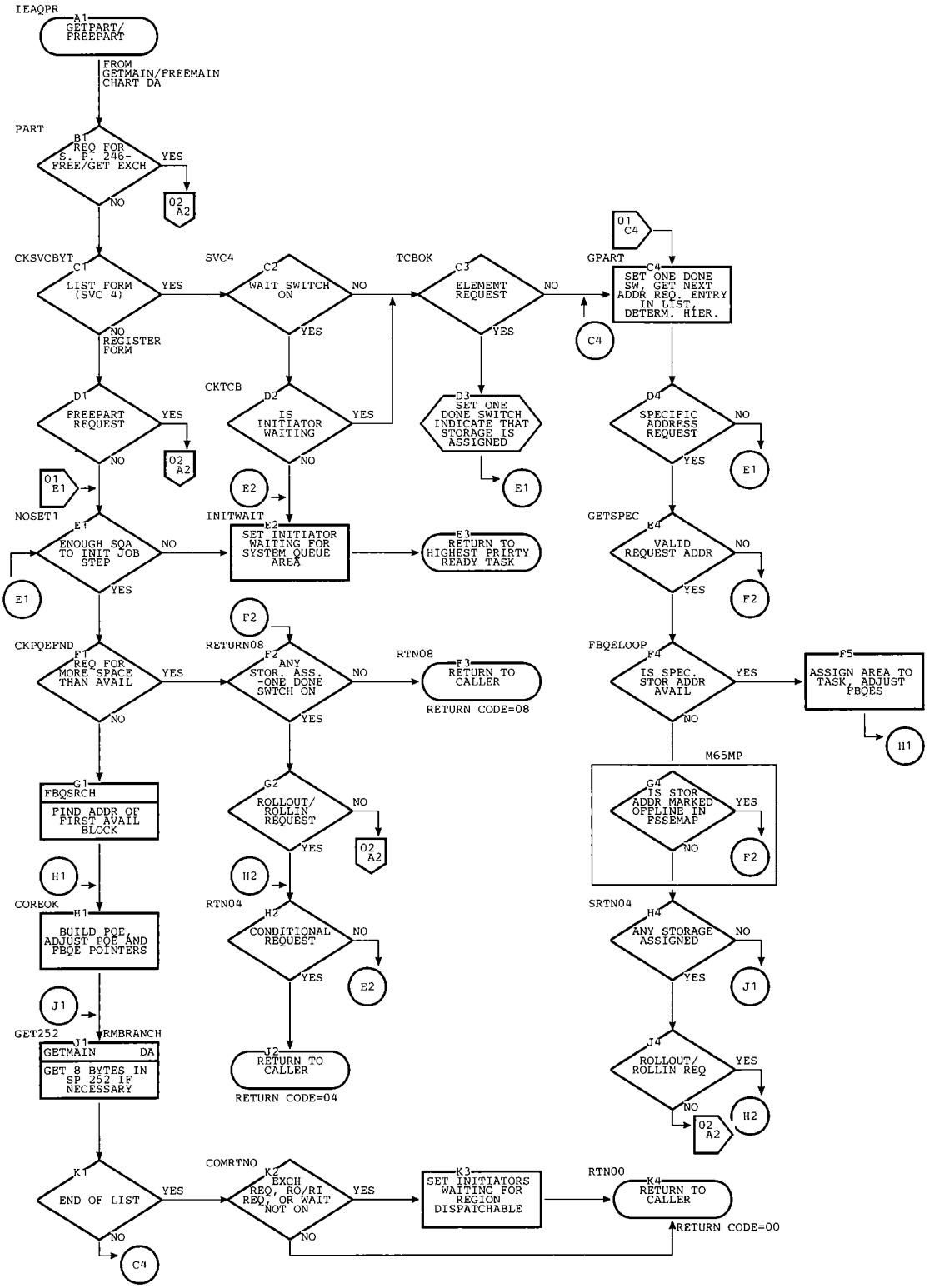


Chart DB. GETPART/FREEPART Routine (Page 2 of 2)

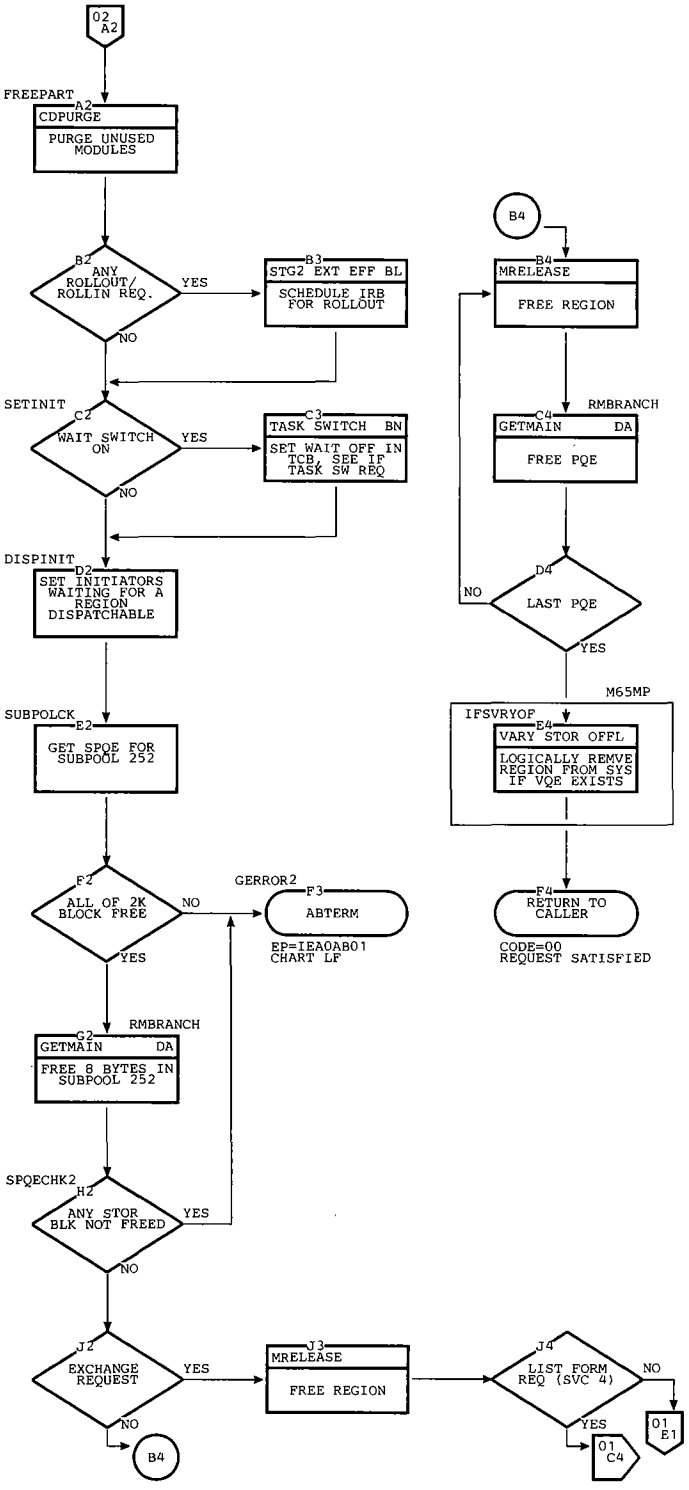


Chart DC. Rollout-Rollin Criterion Routine (Page 1 of 2)

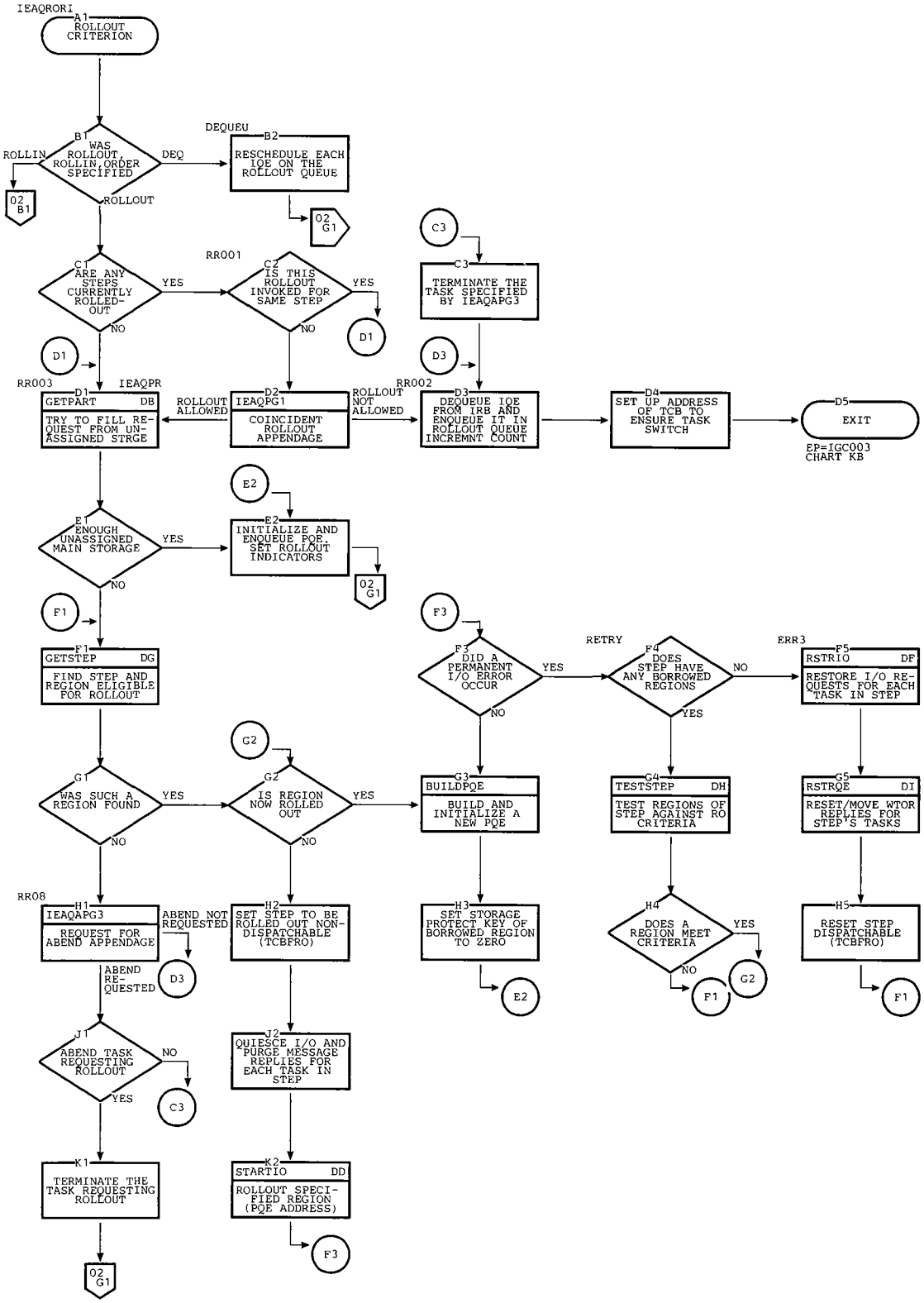


Chart DC. Rollout-Rollin Criterion Routine (Page 2 of 2)

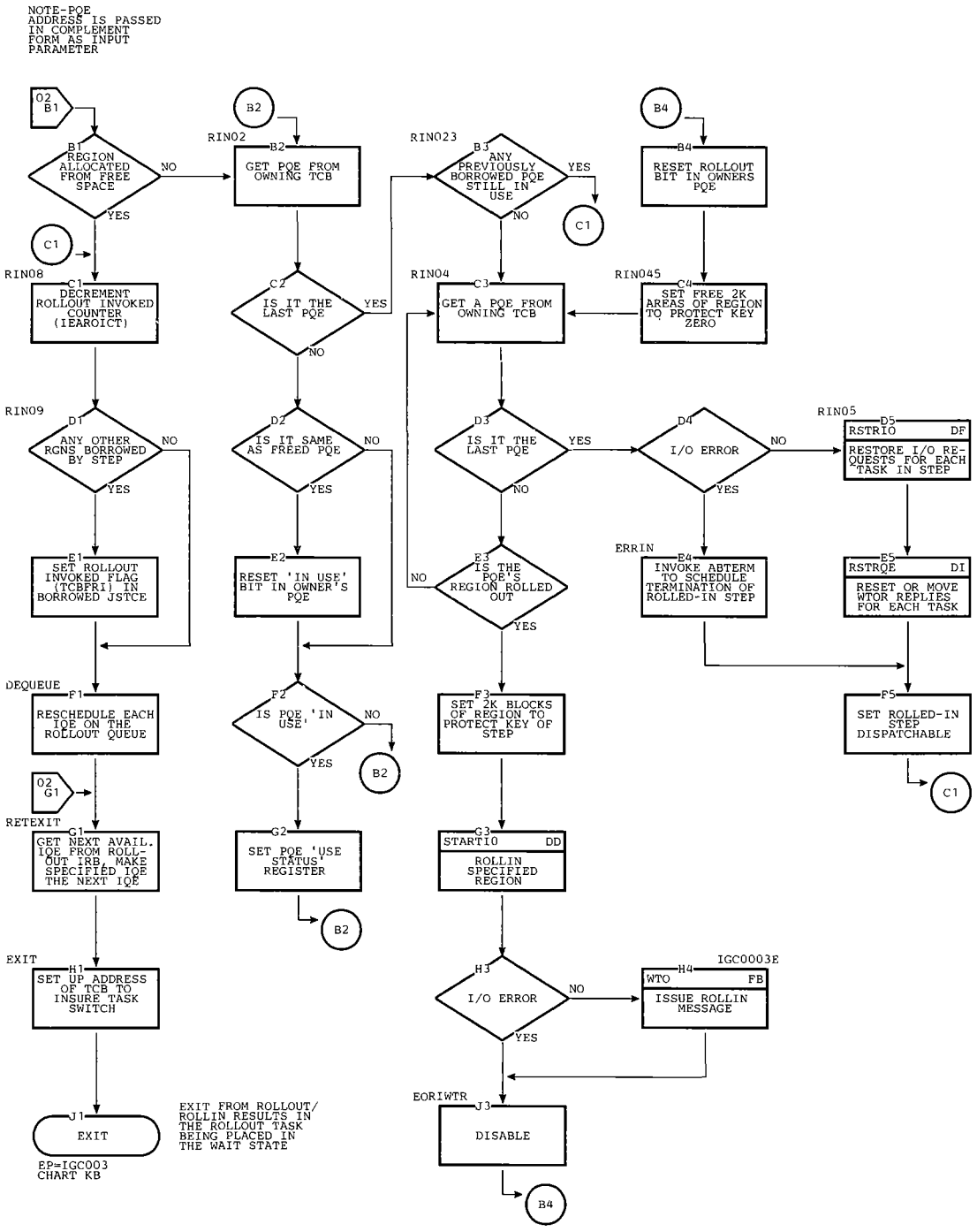


Chart DD. Rollout/Rollin I/O Routine

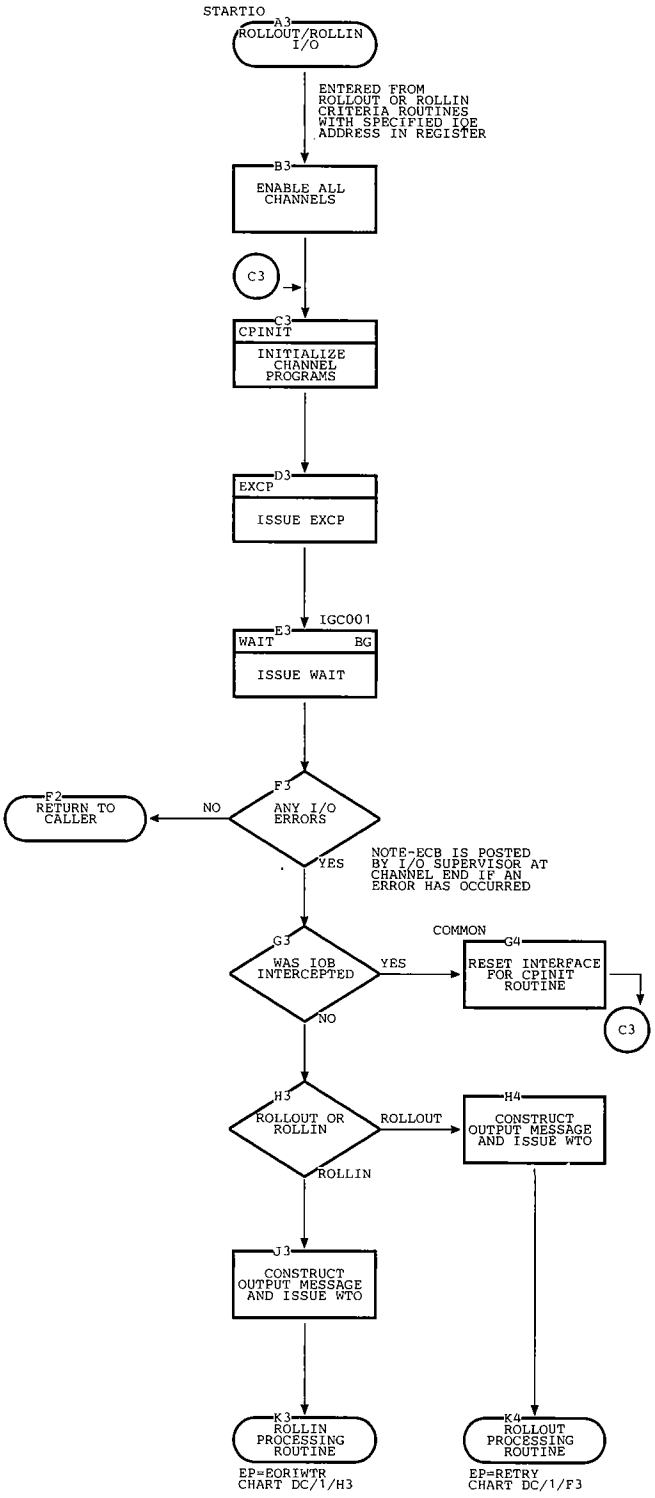


Chart DE. SVC Purge Interface

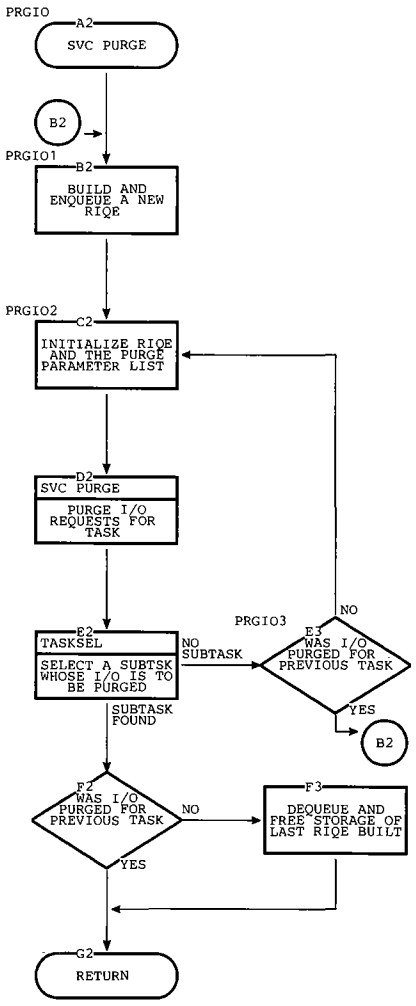


Chart DF. SVC Restore Interface

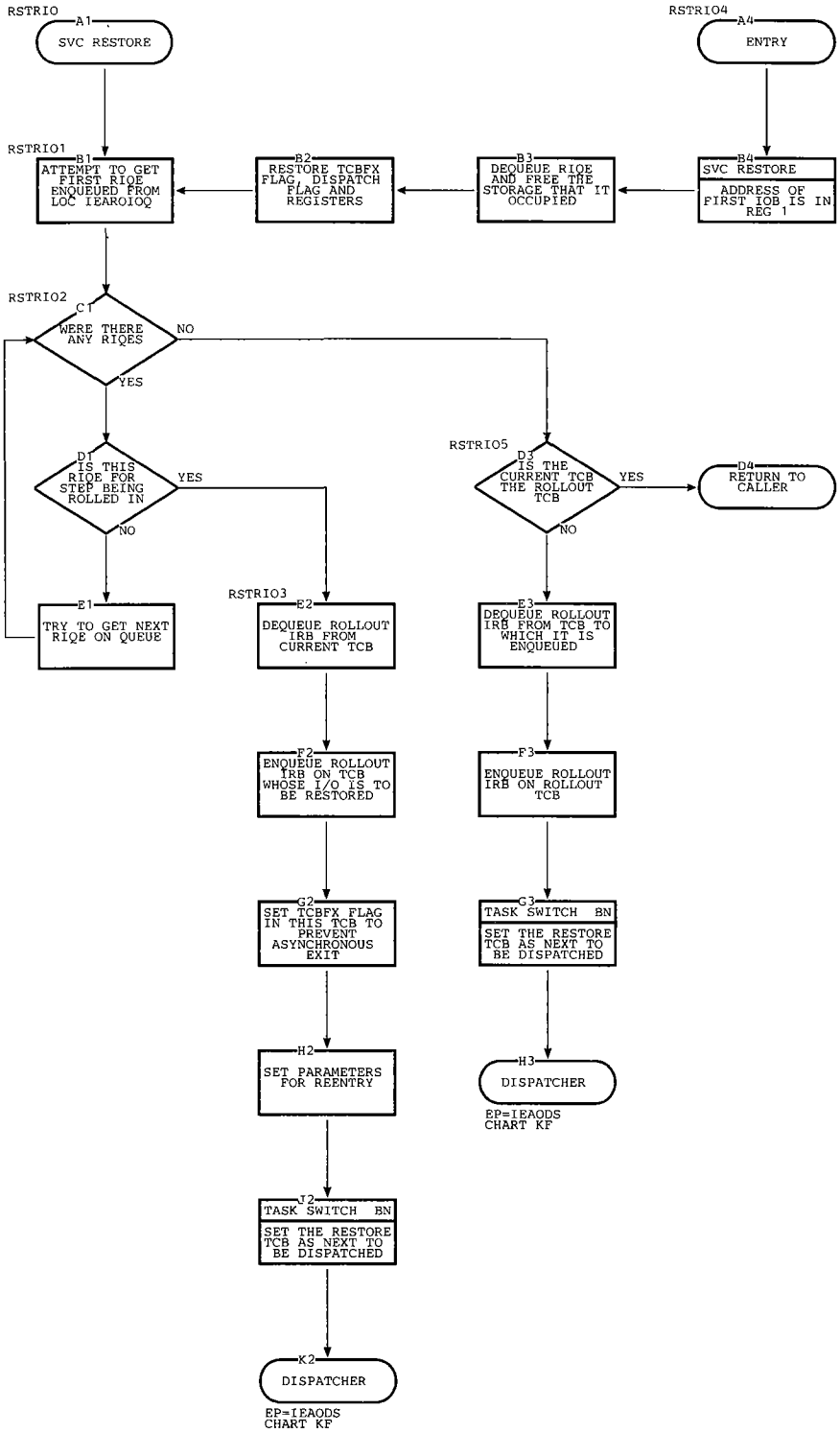


Chart DG. Rollout/Rollin GETSTEP Routine

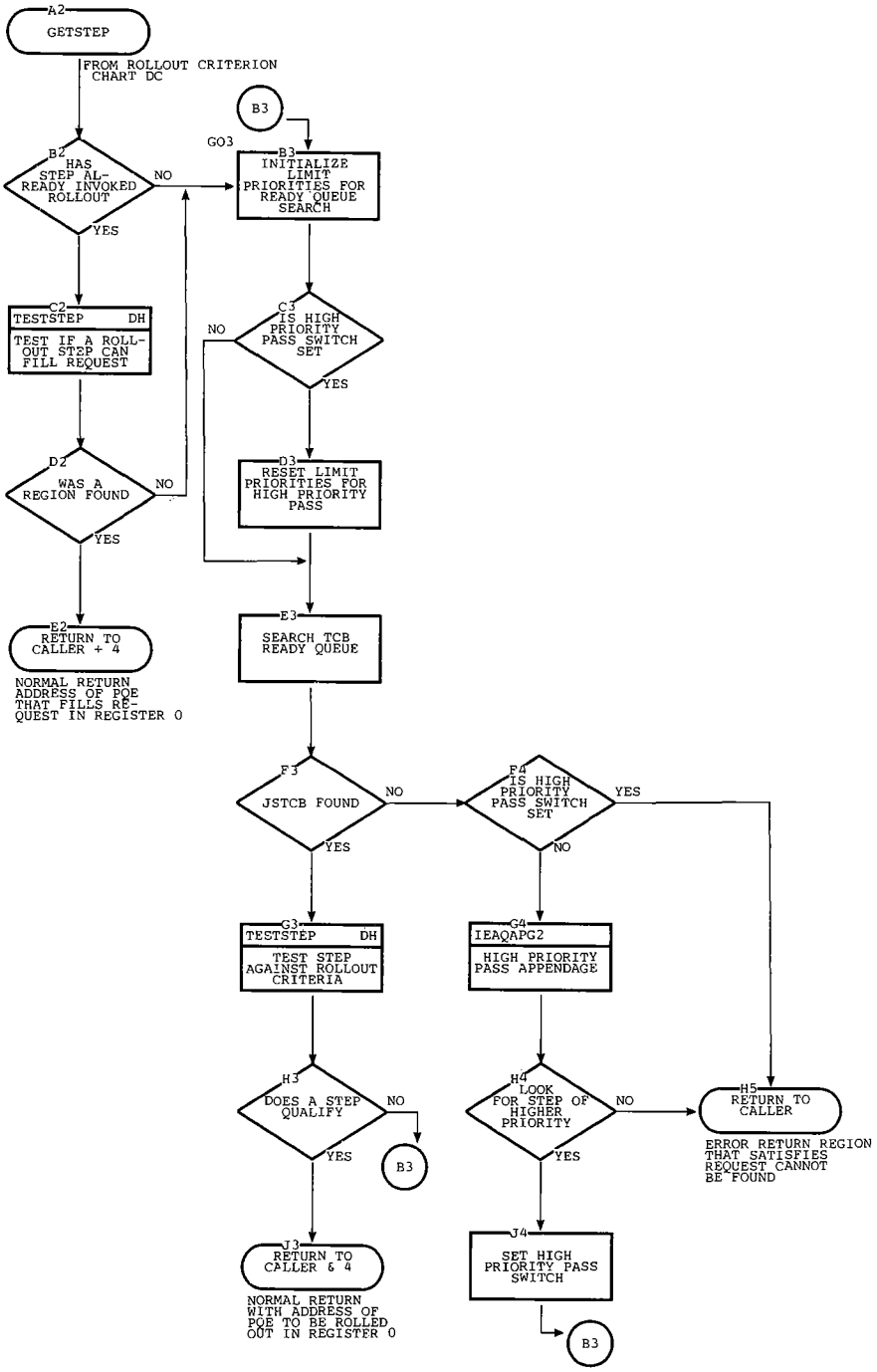


Chart DH. Rollout/Rollin TESTSTEP Routine

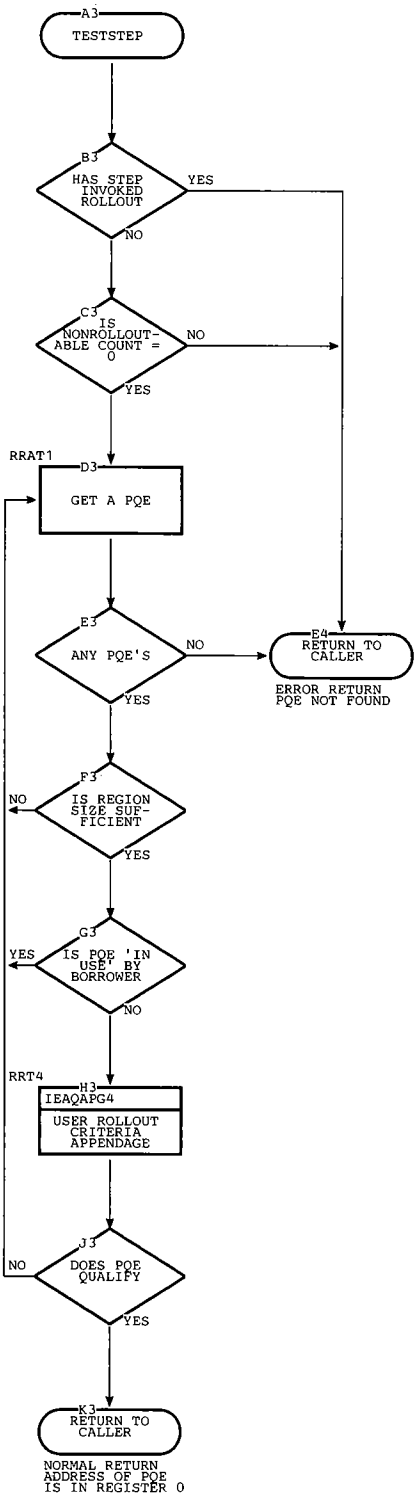


Chart DI. Rollout/Rollin Reply Restore Routine

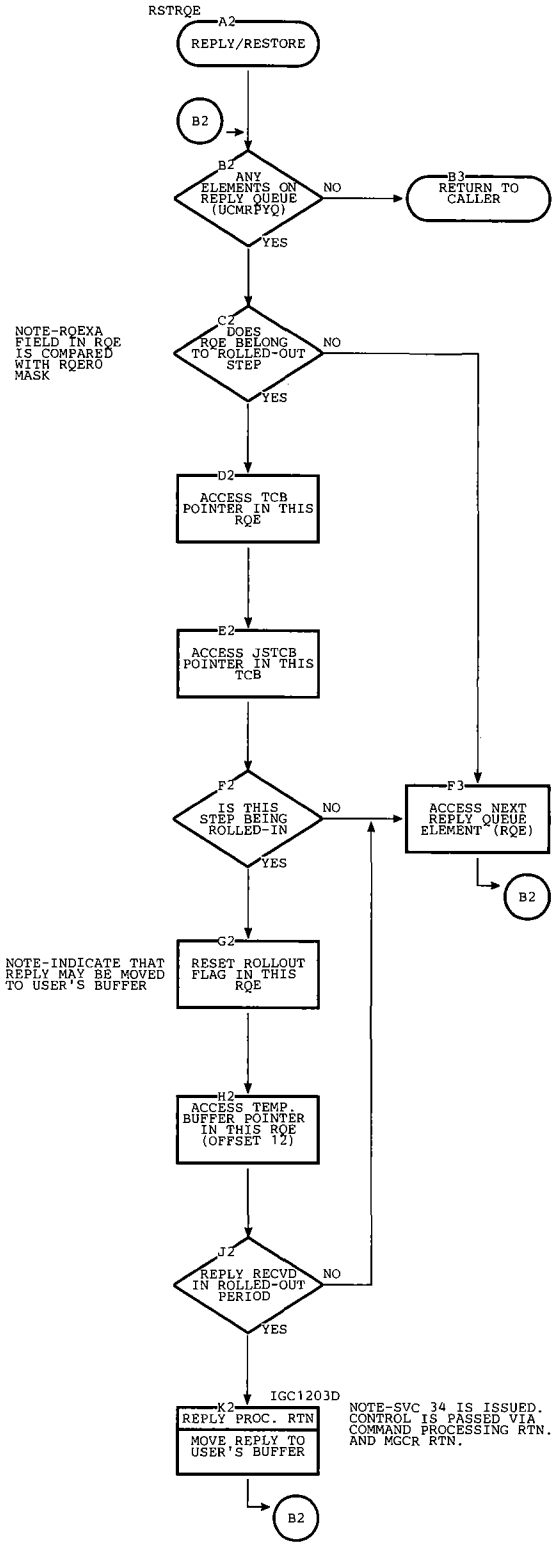


Chart EA. TIME Routine

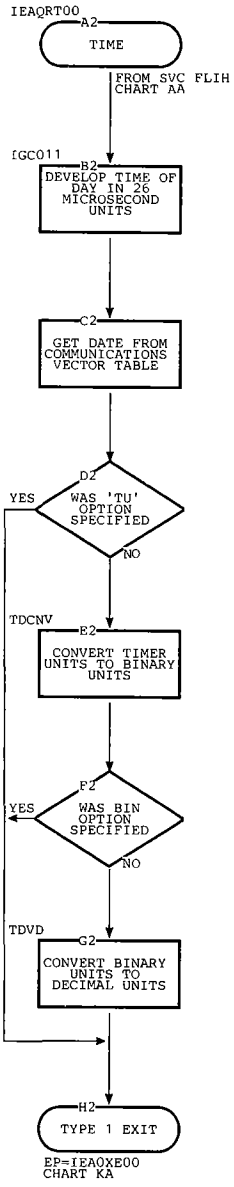


Chart EB. STIMER Routine

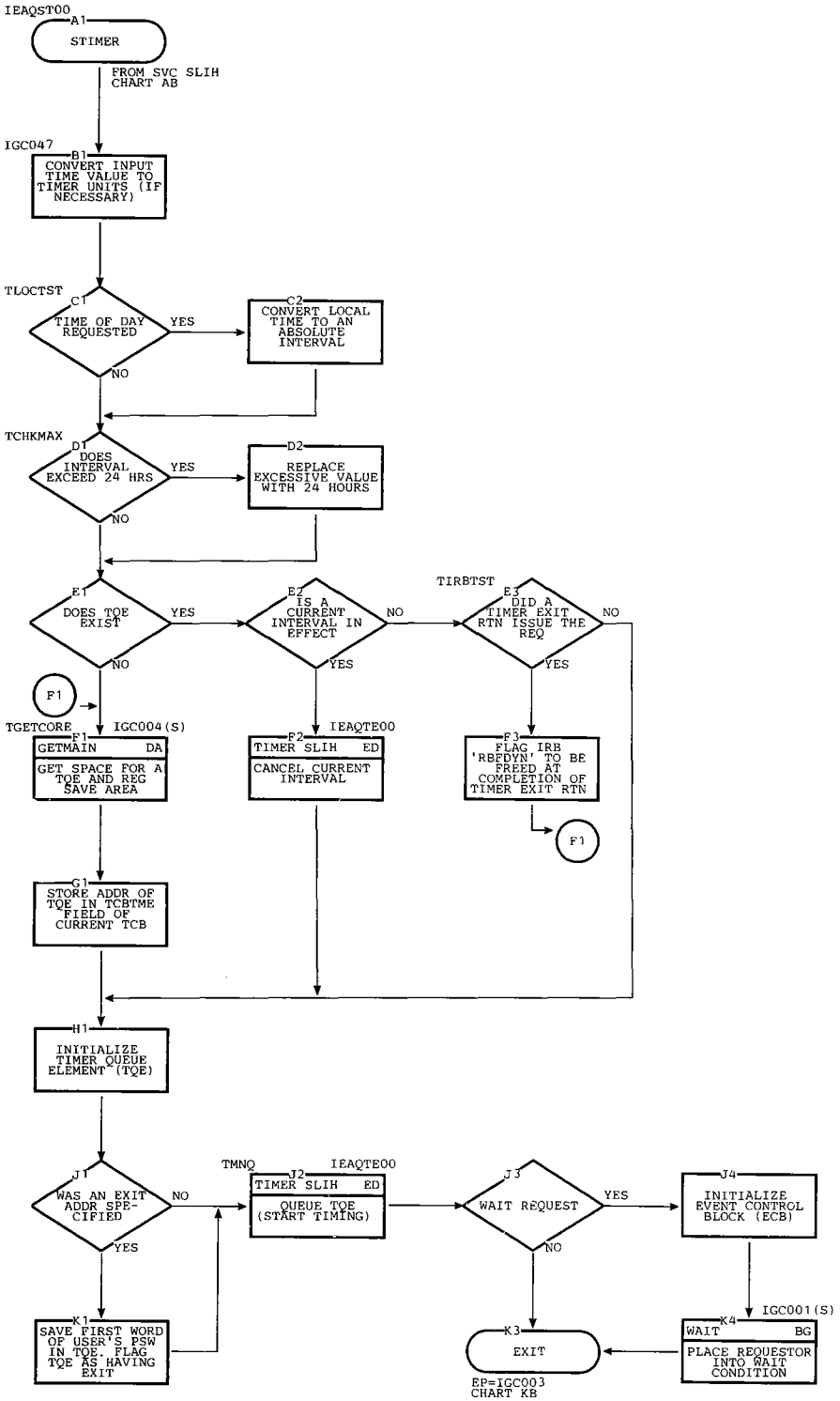


Chart EC. TTIMER Routine

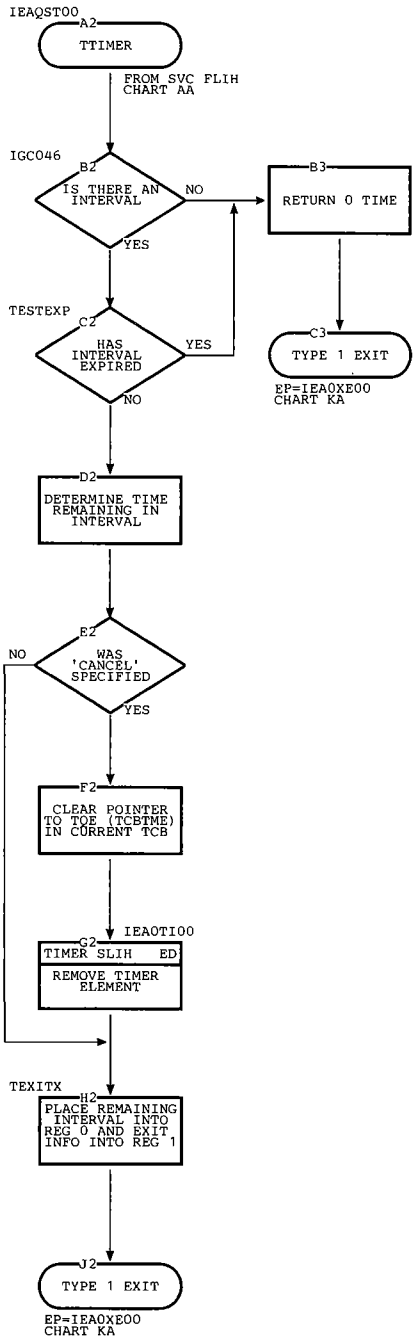


Chart ED. Timer Second-Level Interruption Handler (Page 1 of 2)

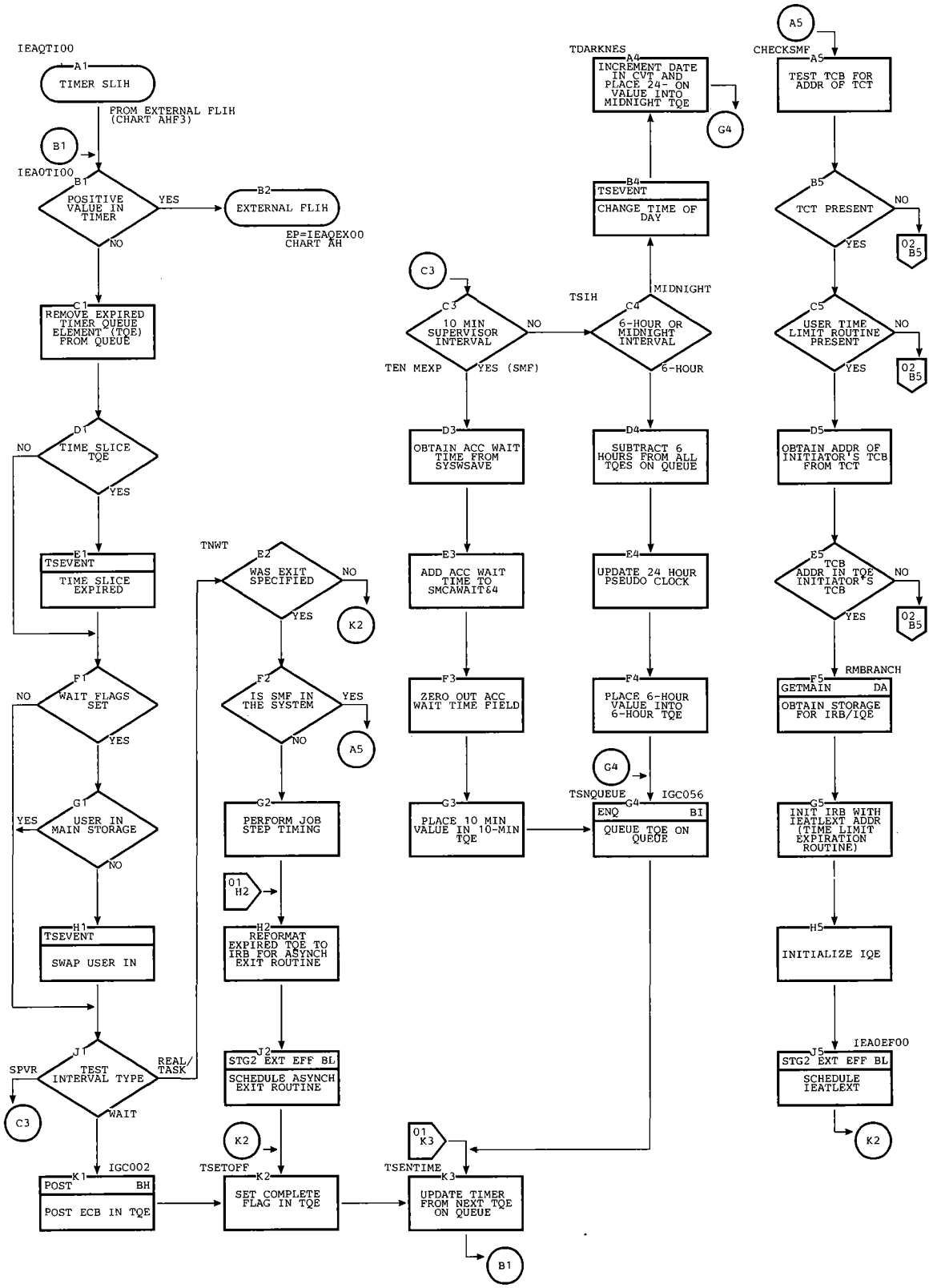


Chart ED. Timer Second-Level Interruption Handler -- Dequeue and Enqueue Subroutines (Page 2 of 2)

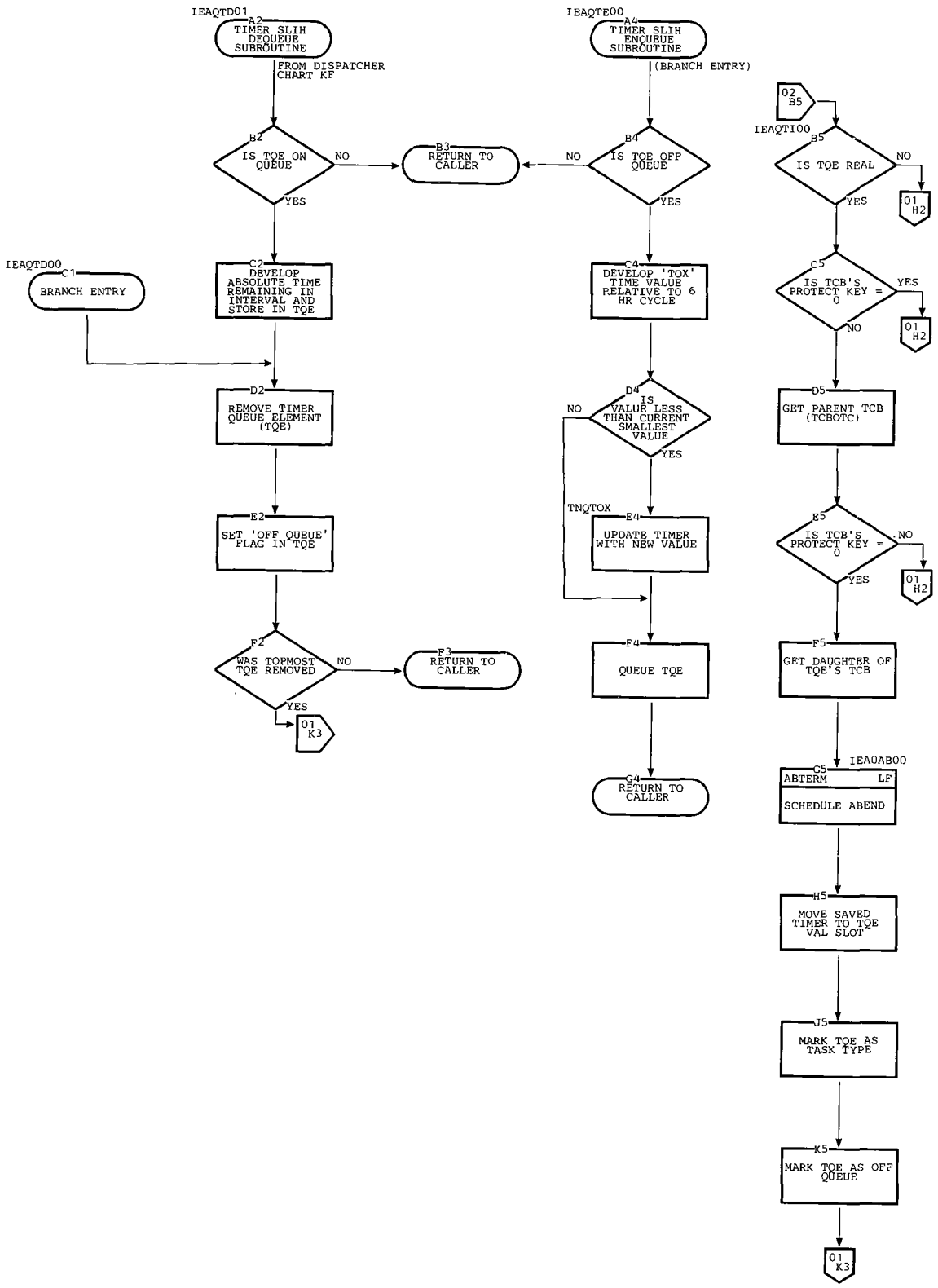


Chart EE. TIME Routine with System/370 Time-of-Day Clock

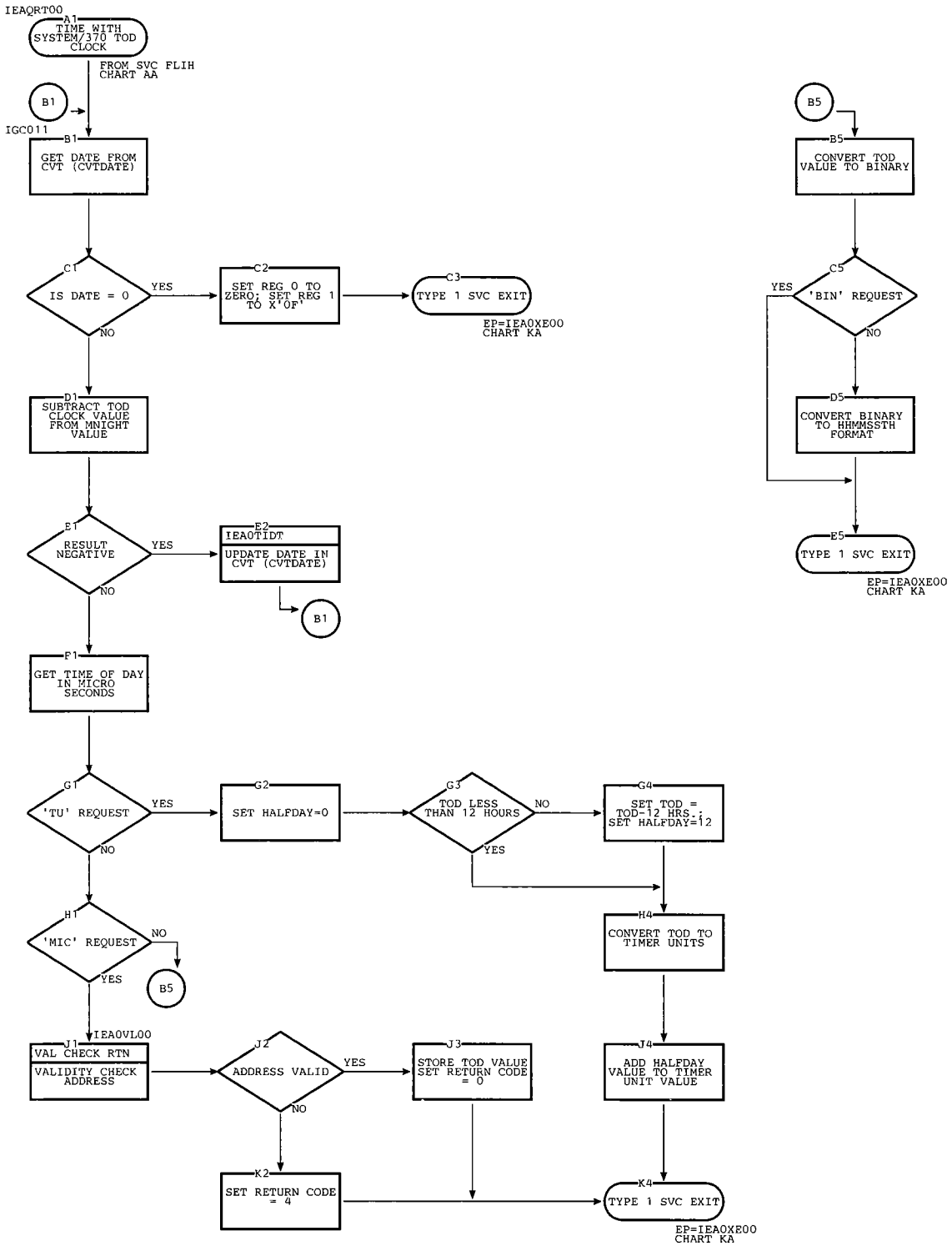


Chart EF. STIMER Routine with System/370 Time-of-Day Clock

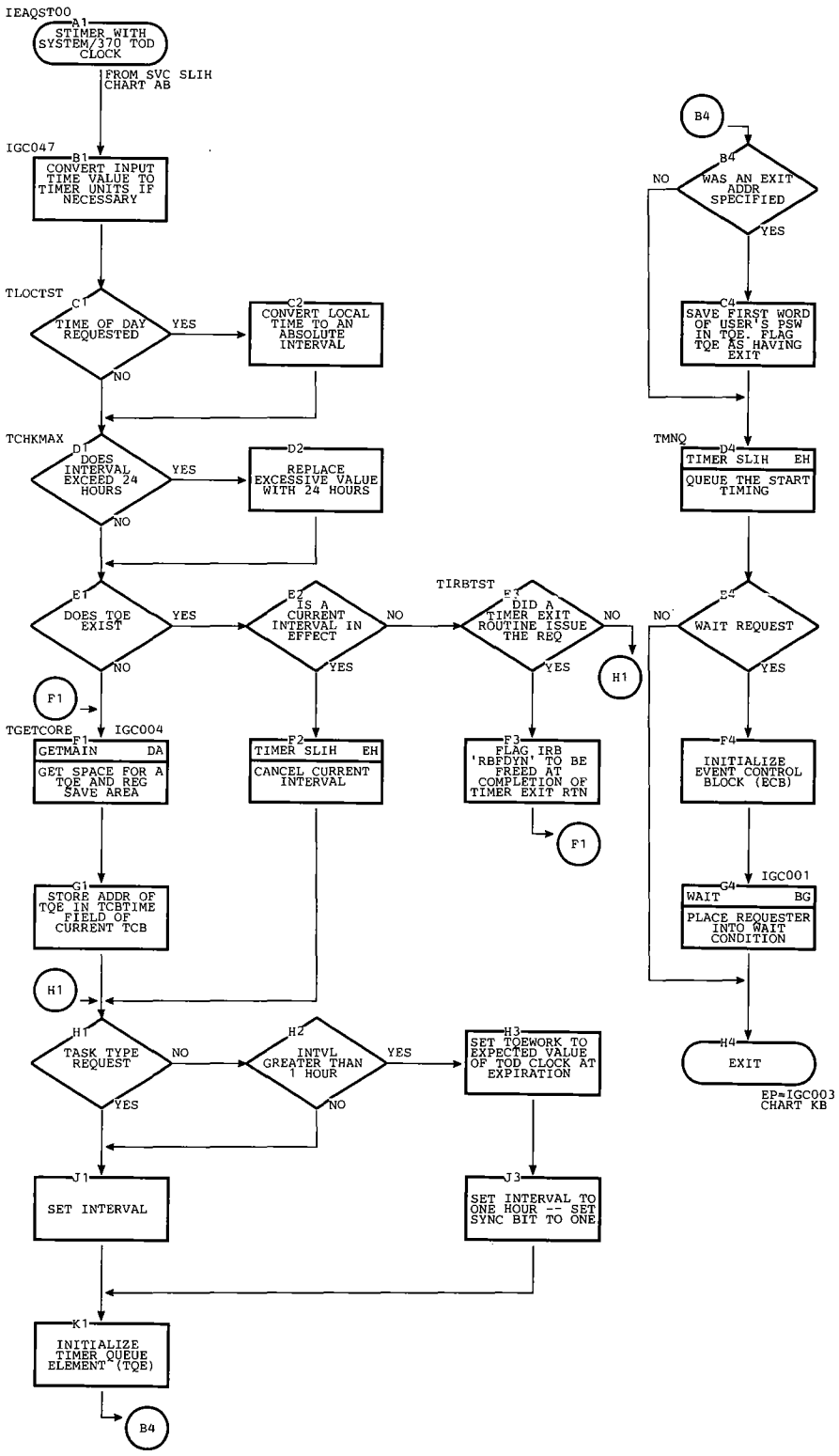


Chart EG. TTIMER Routine with System/370 Time-of-Day Clock

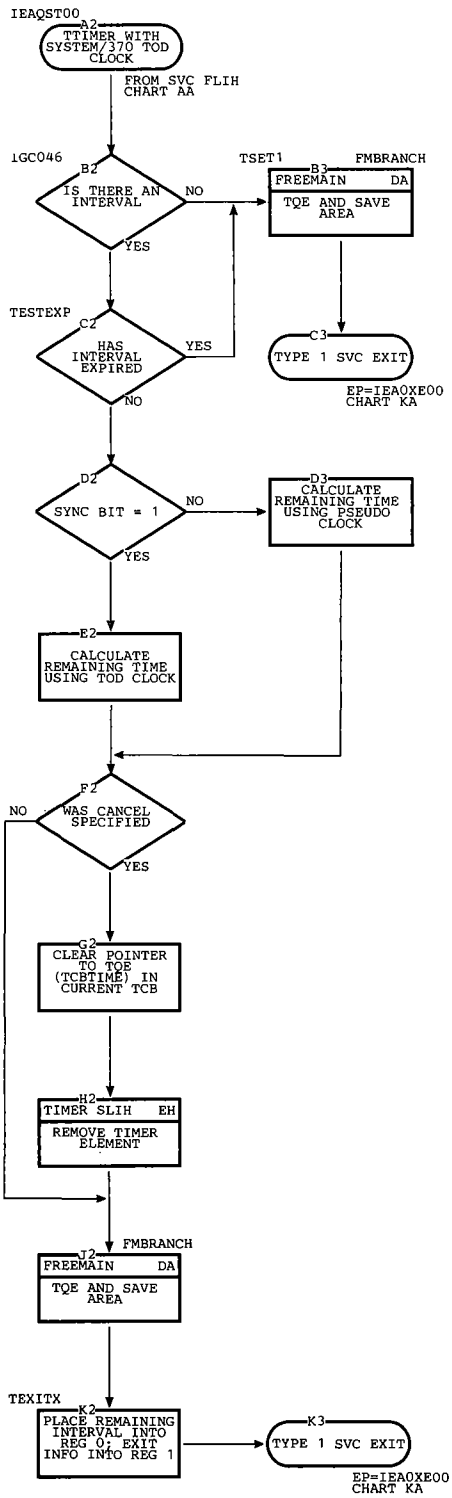


Chart EH. Timer Second-Level Interruption Handler with System/370 Time-of-Day Clock

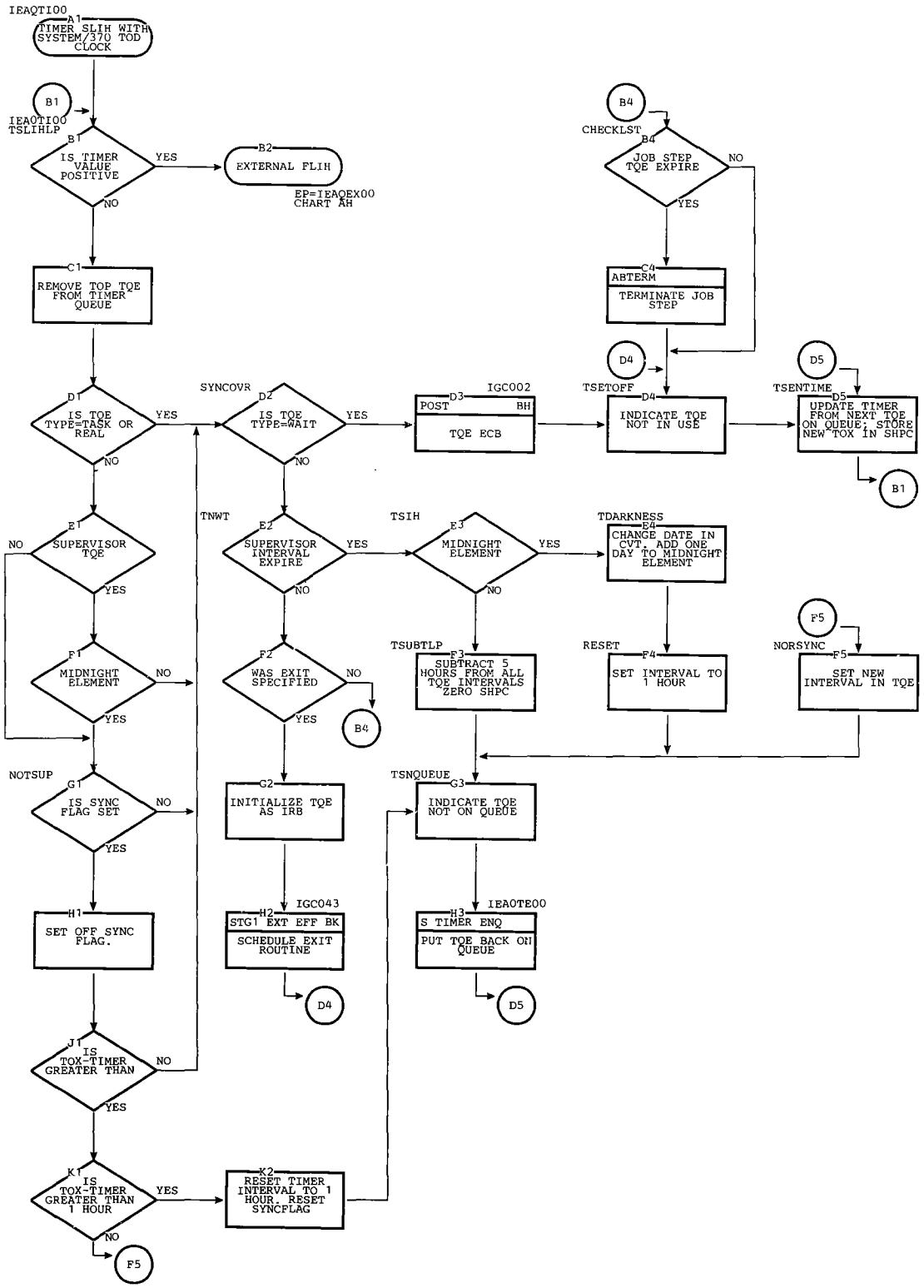


Chart FA. External Interruption and I/O Attention Handlers

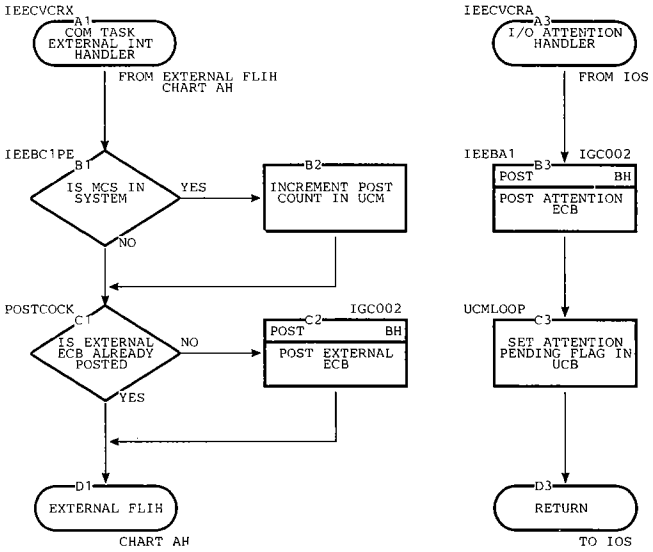


Chart FB. Write-To-Operator

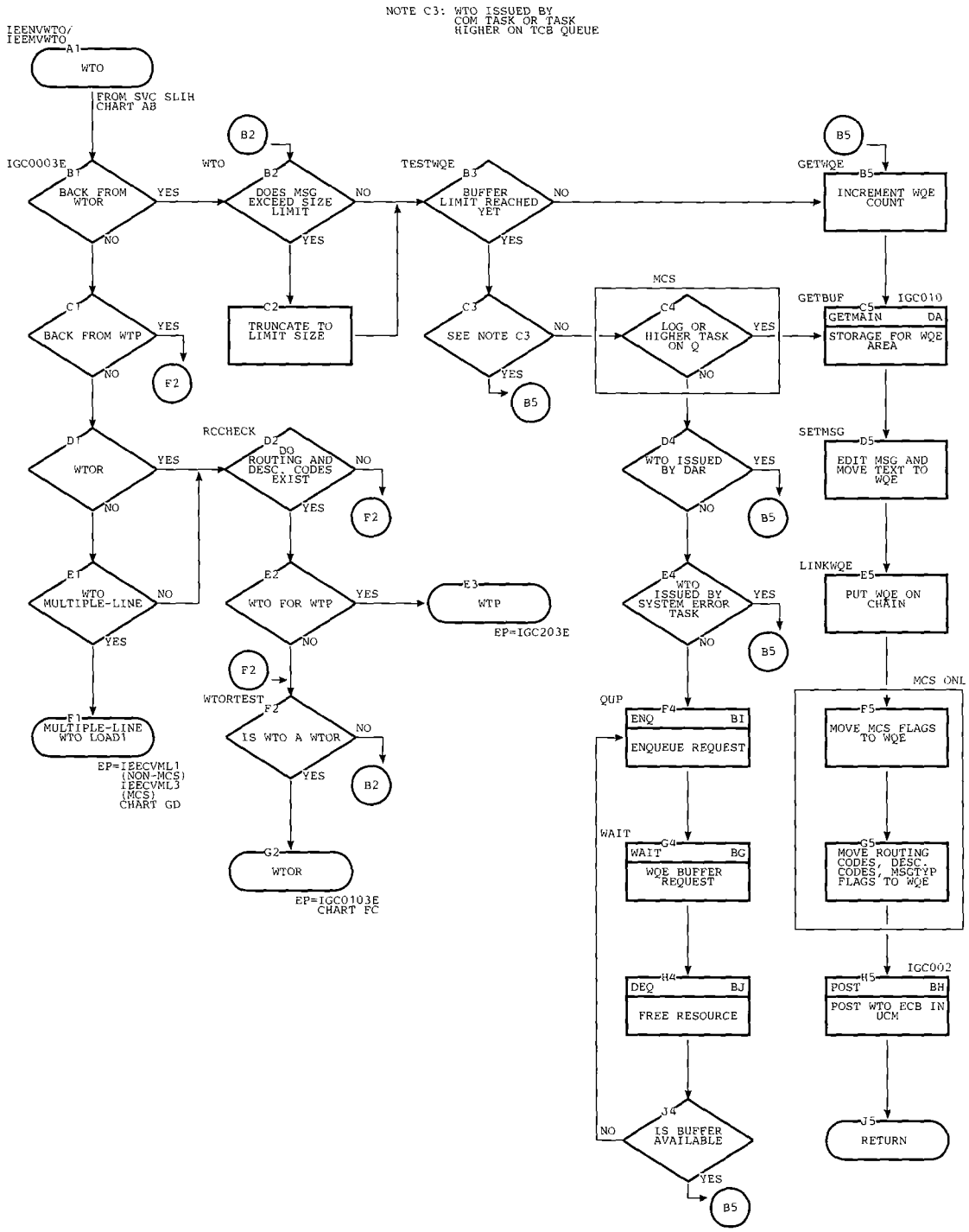


Chart FC. Write-To-Operator with Reply

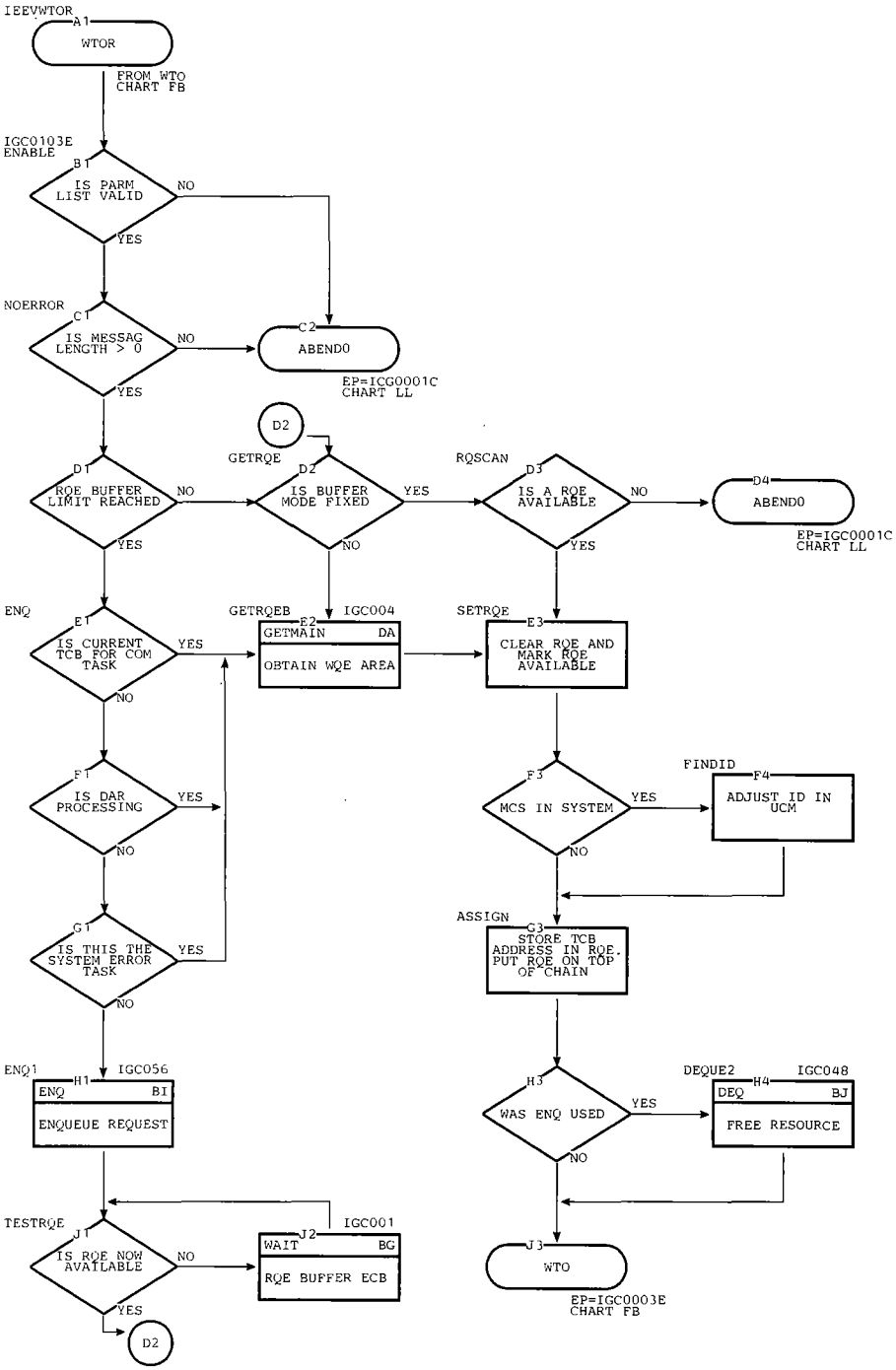


Chart FD. Communications Task Initialization Routine (Page 1 of 5)

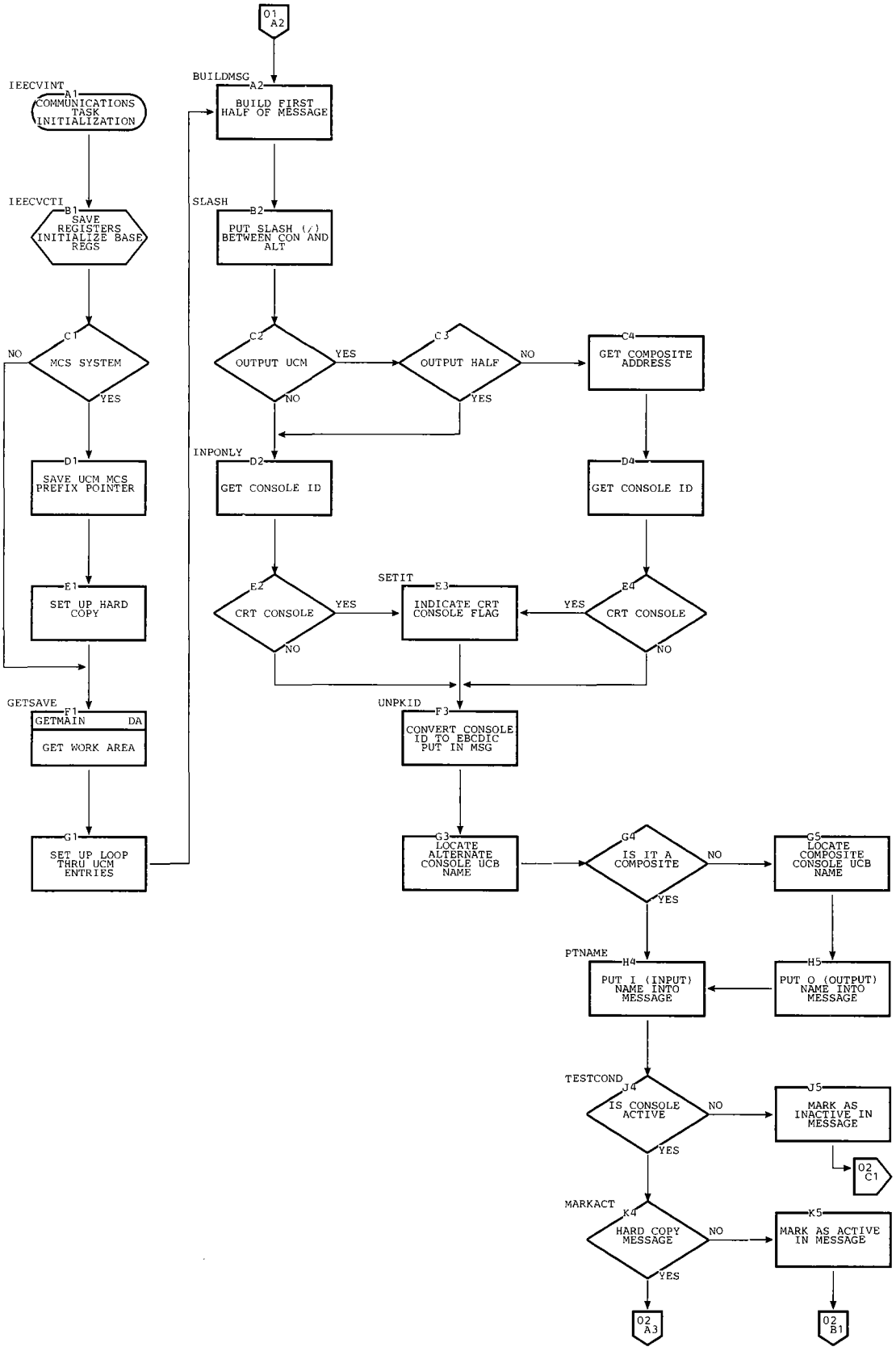


Chart FD. Communications Task Initialization Routine (Page 2 of 5)

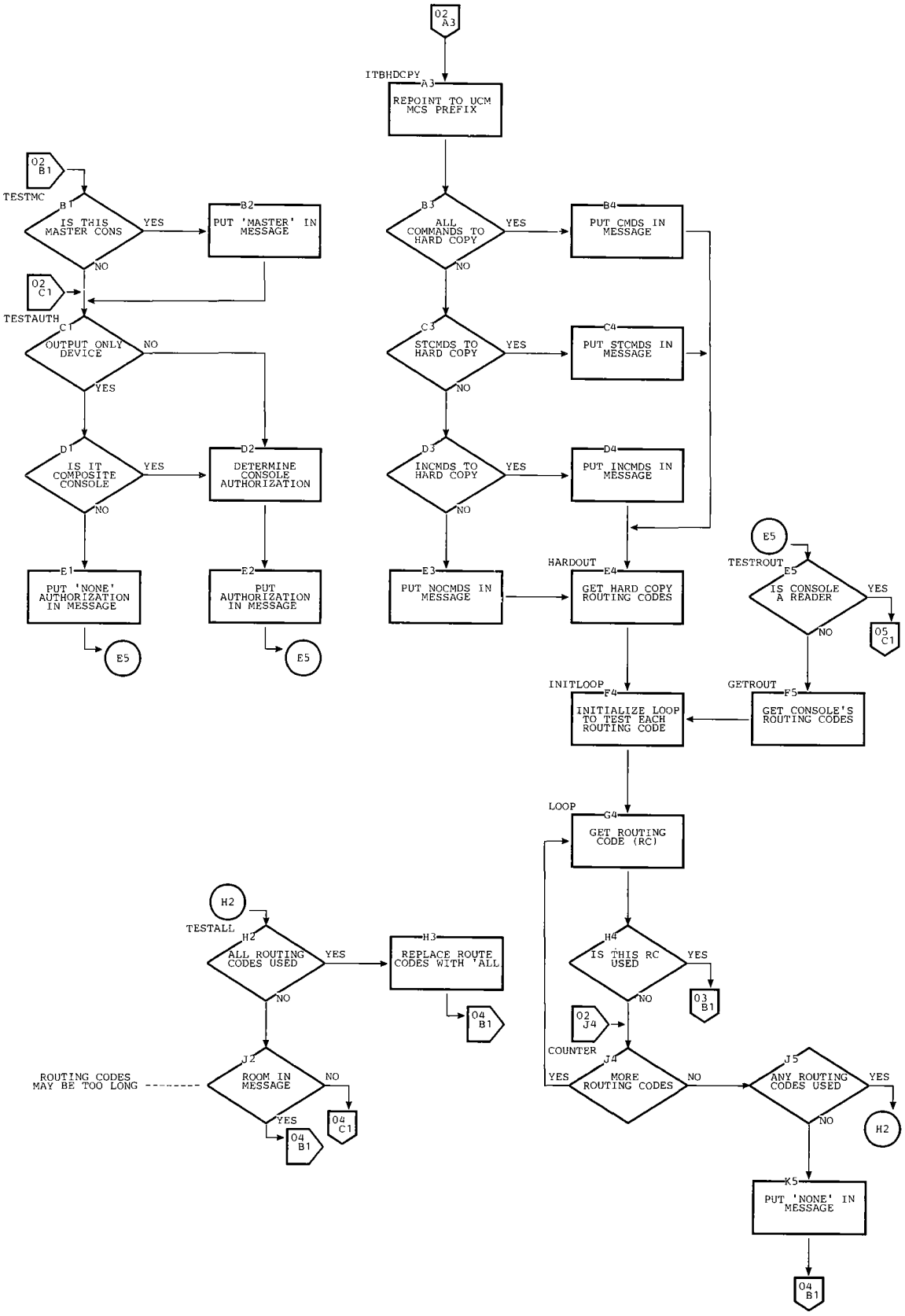


Chart FD. Communications Task Initialization Routine (Page 3 of 5)

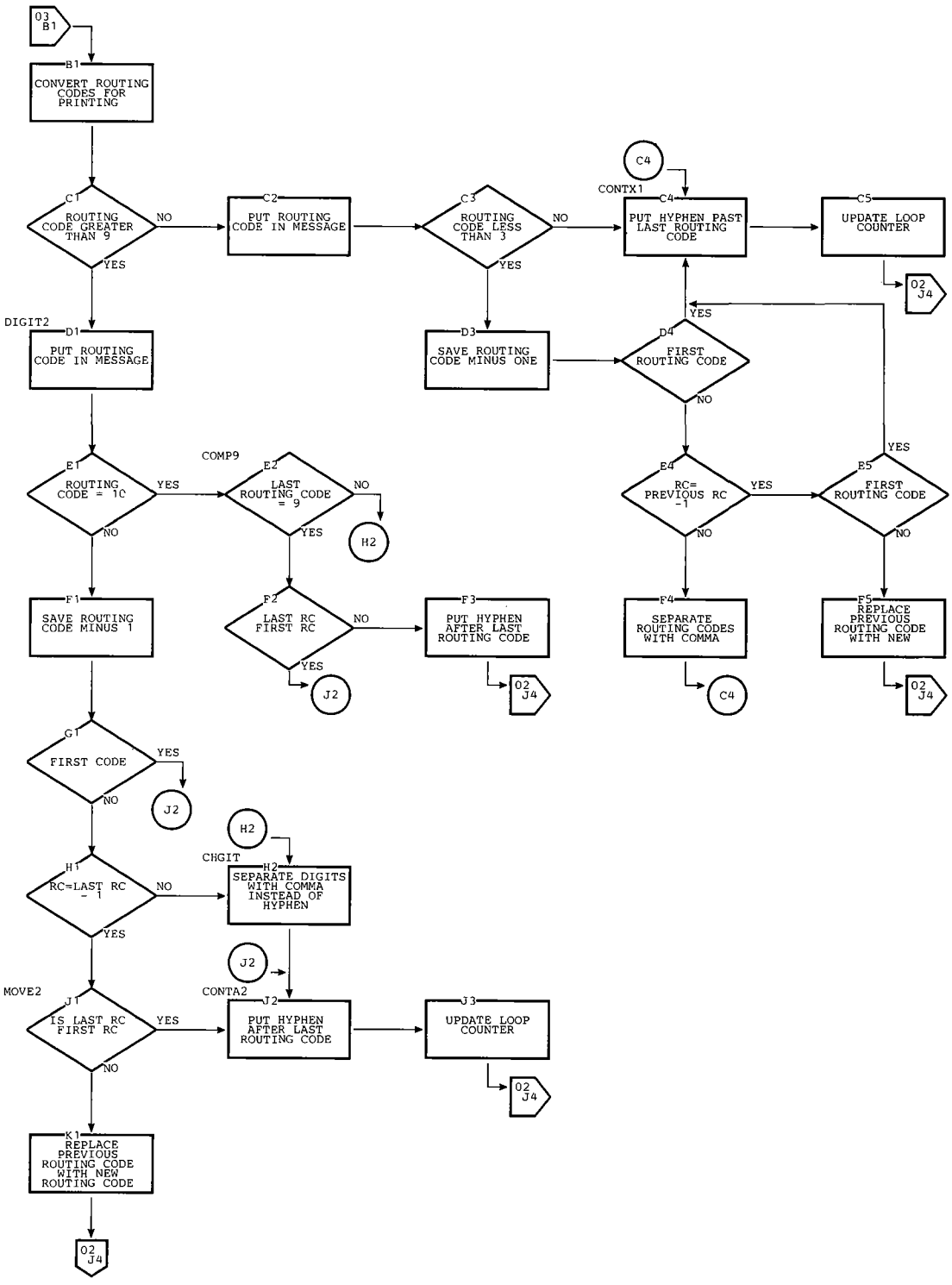


Chart FD. Communications Task Initialization Routine (Page 4 of 5)

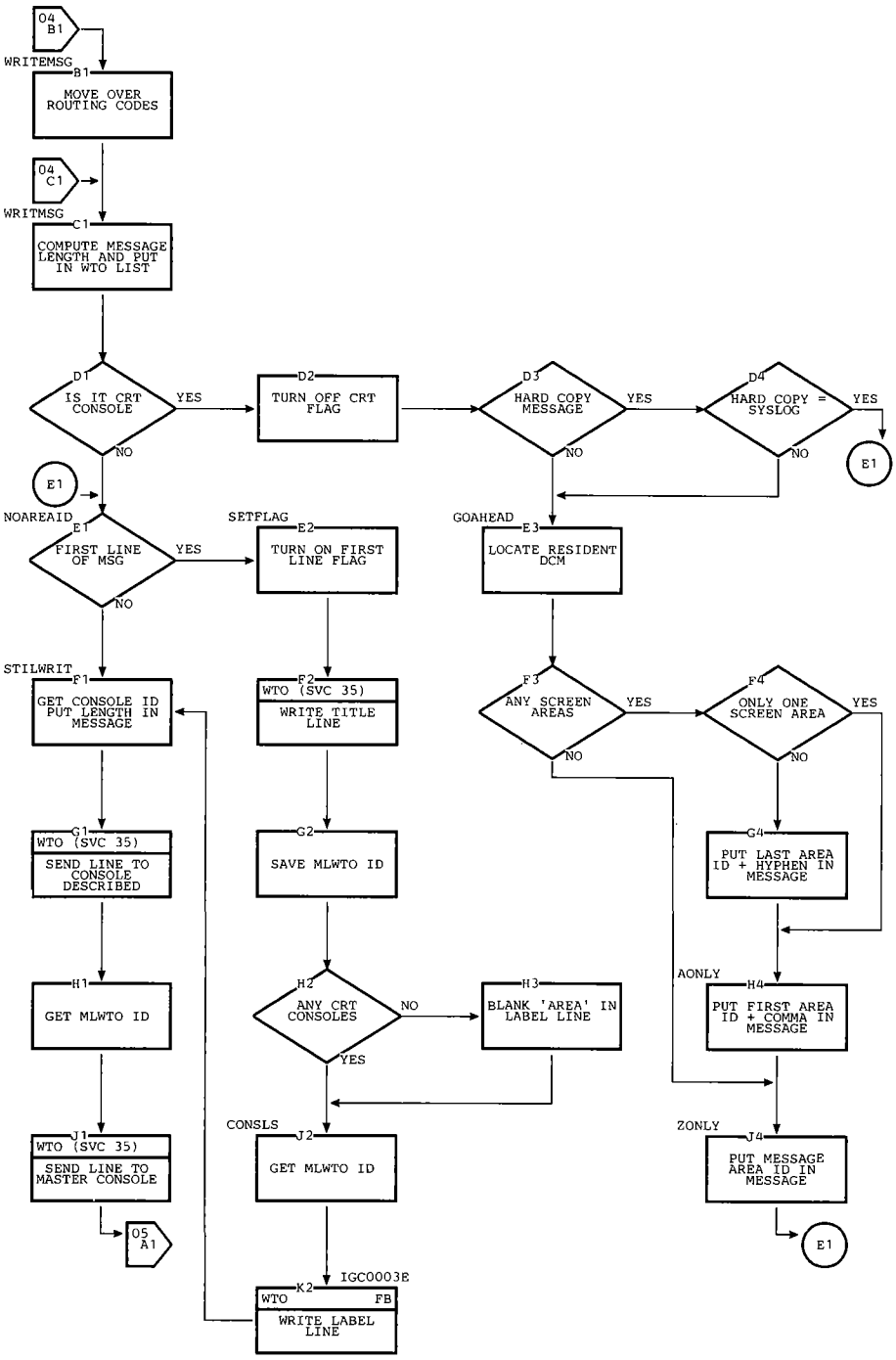


Chart FD. Communications Task Initialization Routine (Page 5 of 5)

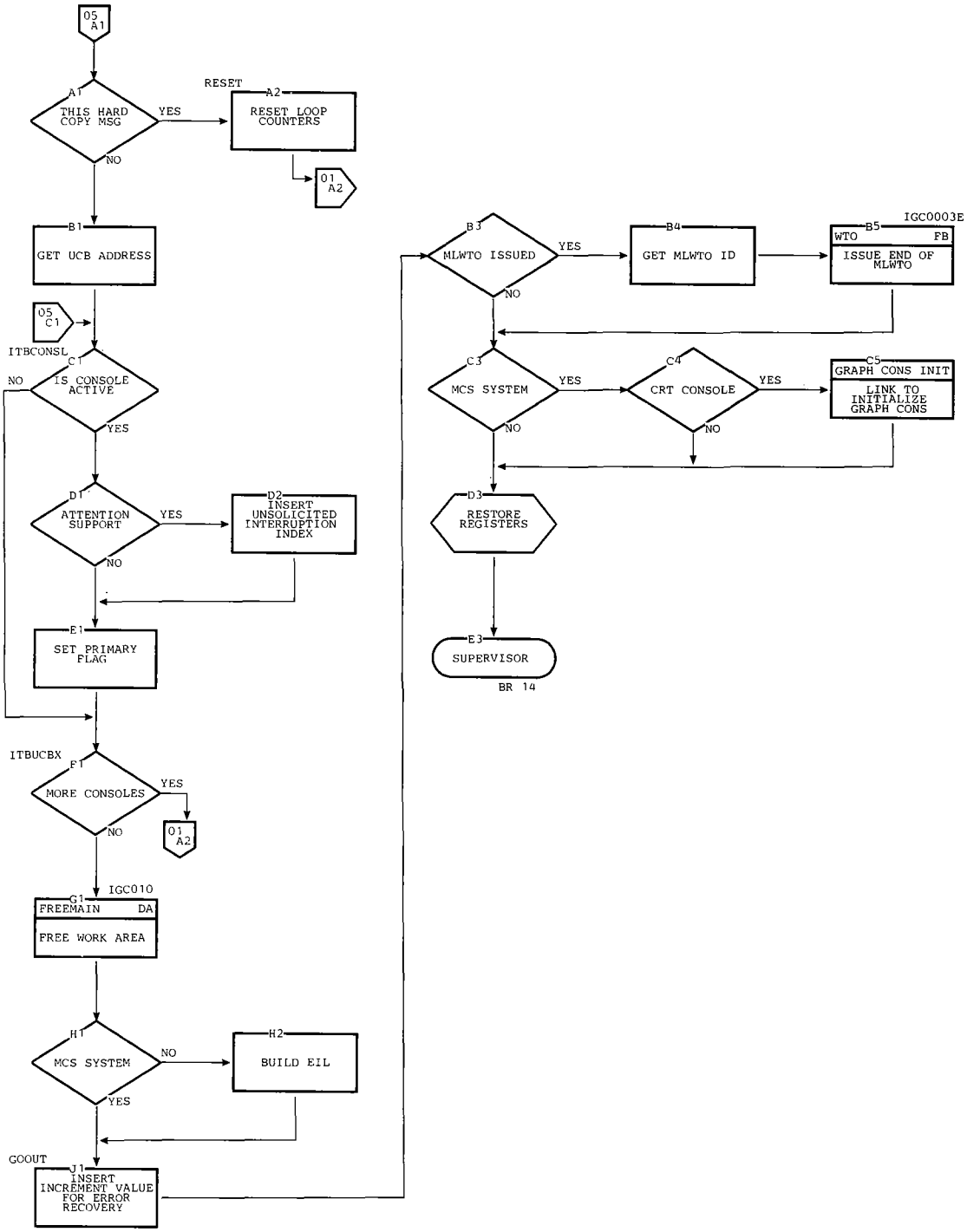


Chart FE. Graphic Console Initialization Routine

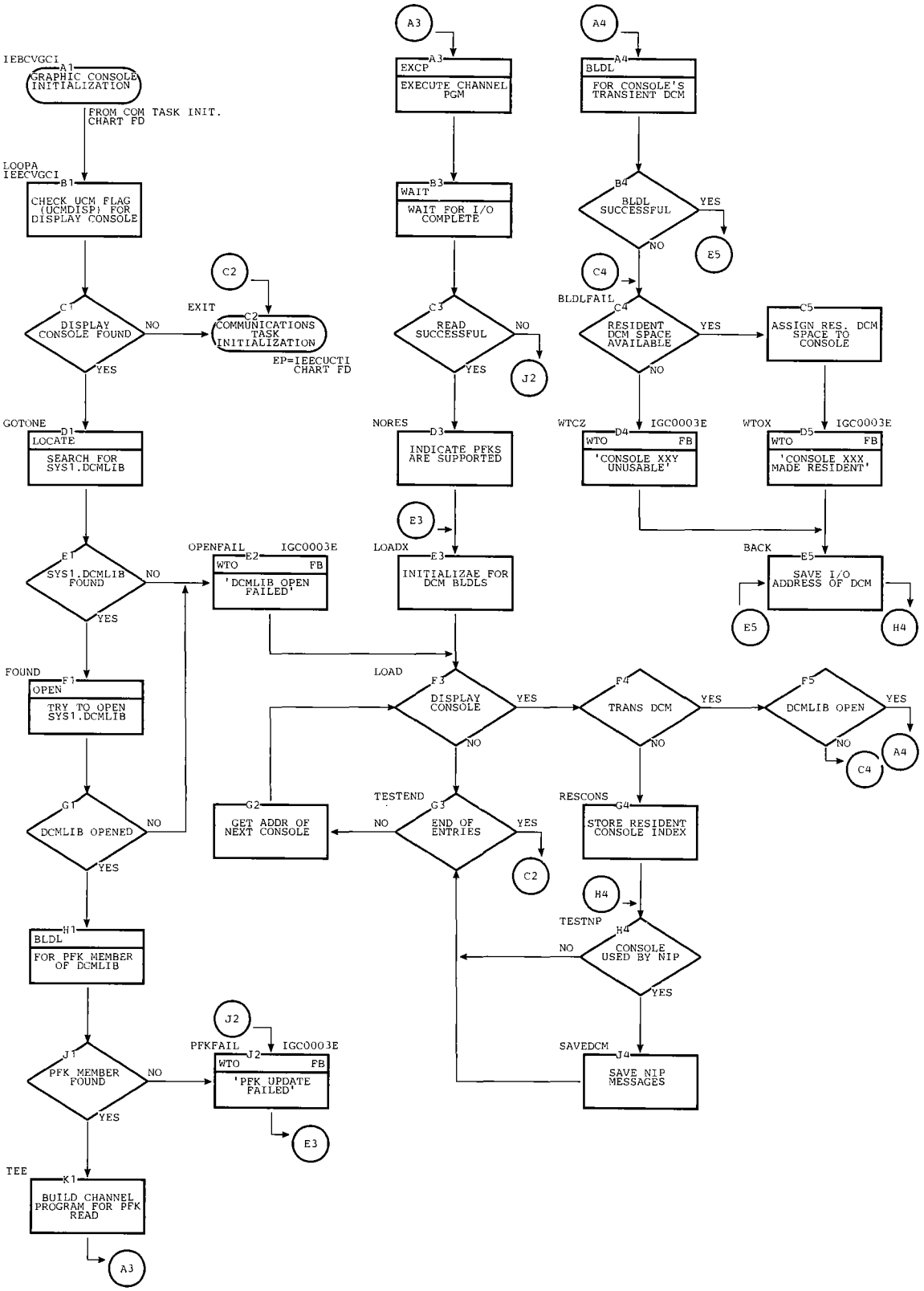


Chart FF. Wait -- Communications Task without Multiple Console Support

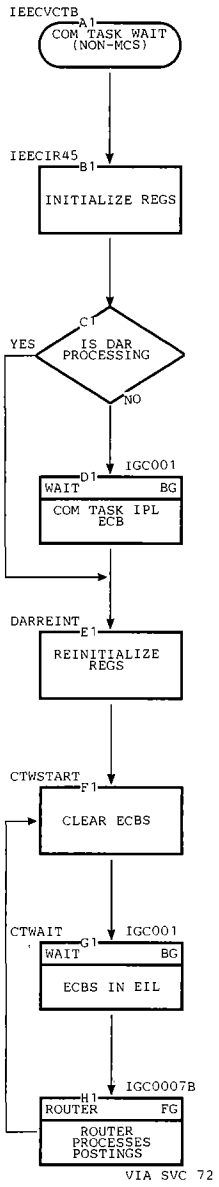


Chart FG. Router -- Communications Task without Multiple Console Support

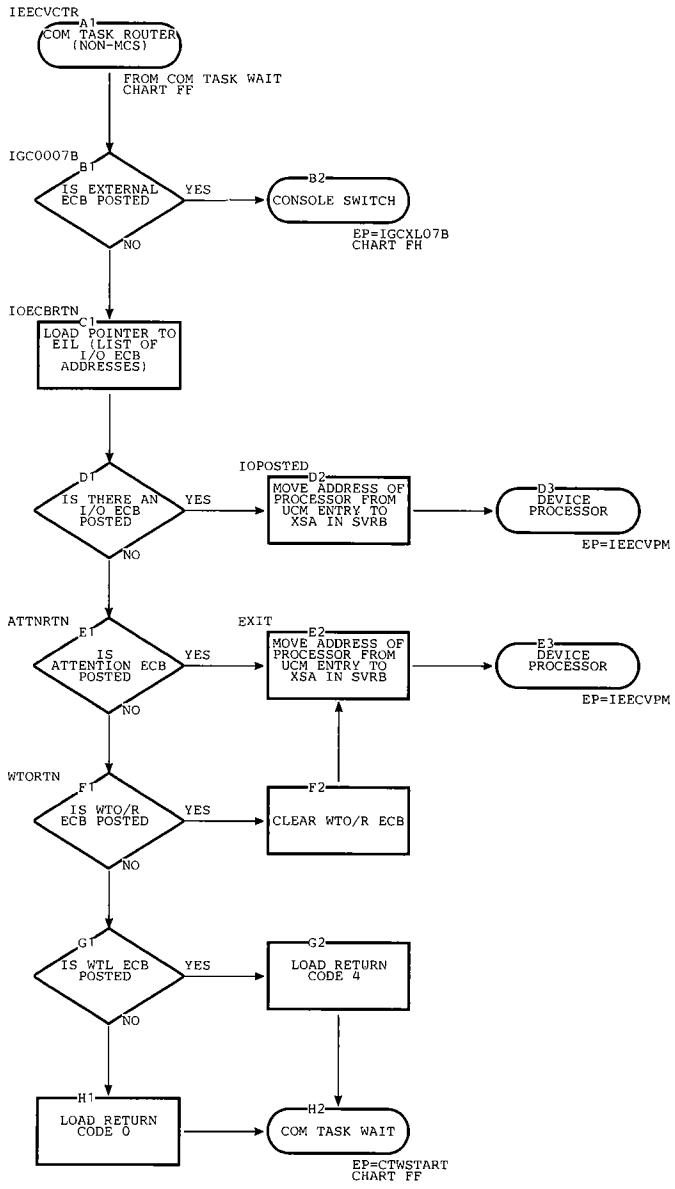


Chart FH. Console Switch -- Communications Task without Multiple Console Support

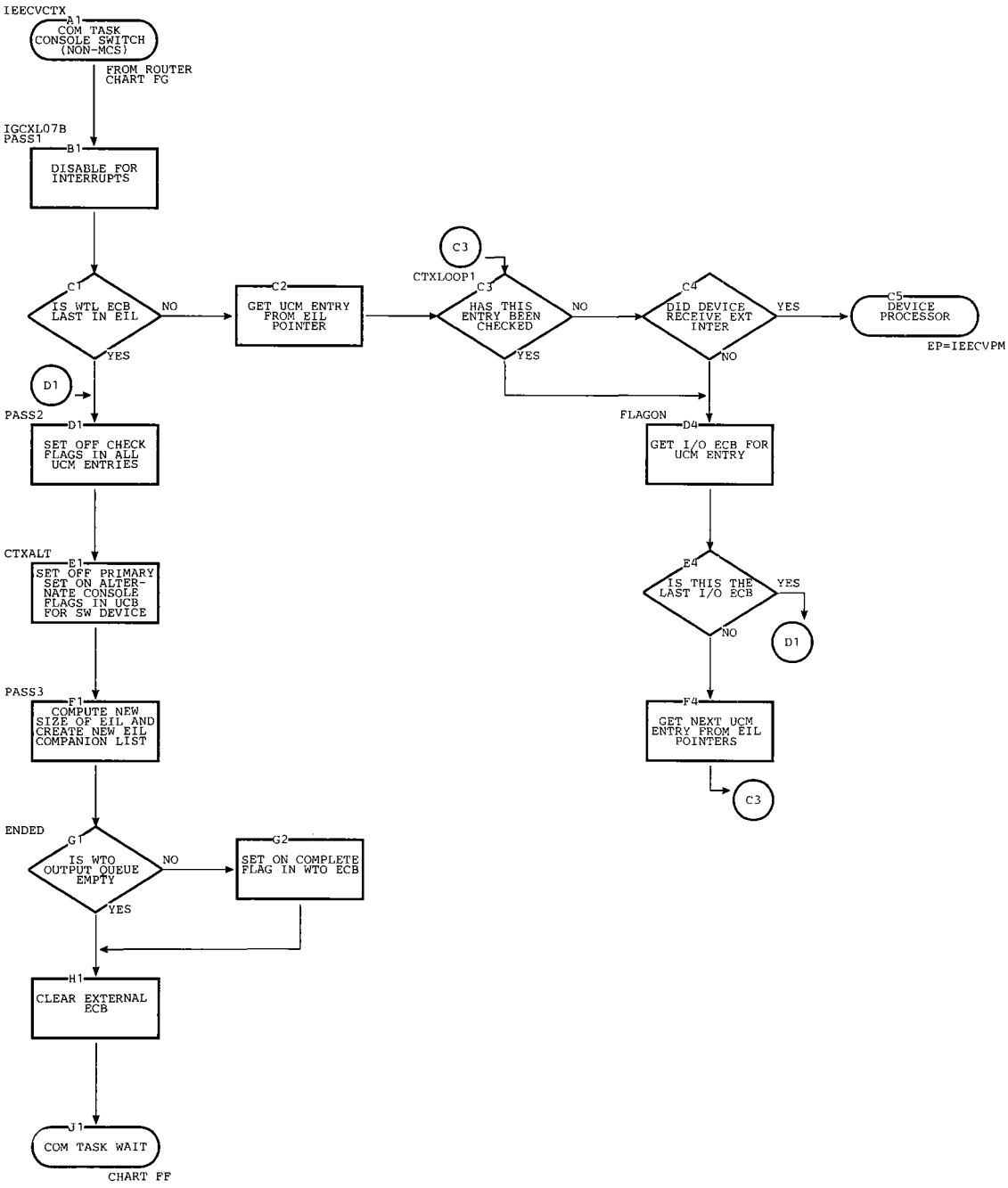


Chart FI. Router -- Communications Task with Multiple Console Support

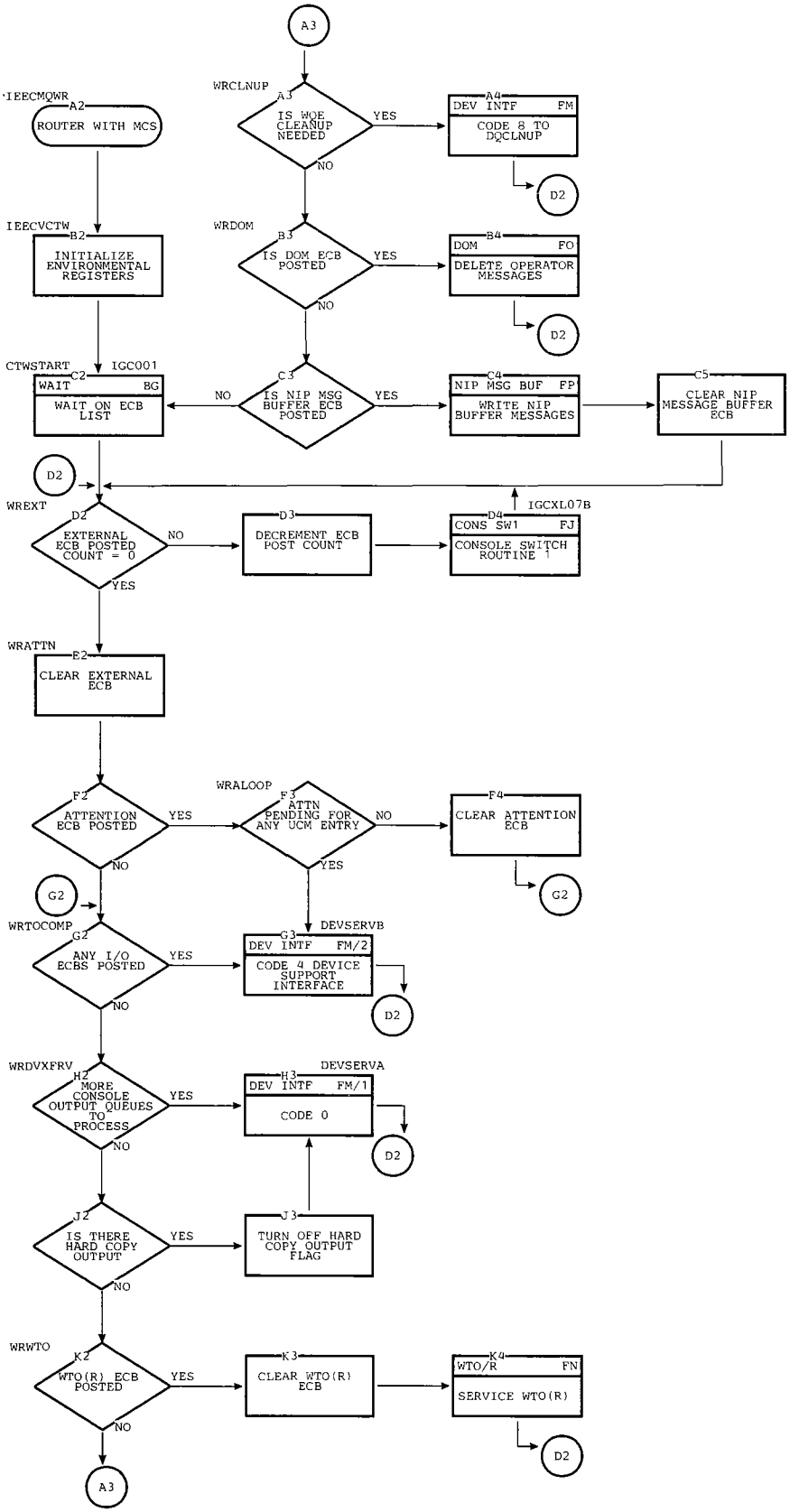


Chart FJ. Console Switch Load 1 -- Communications Task with Multiple Console Support (Page 1 of 3)

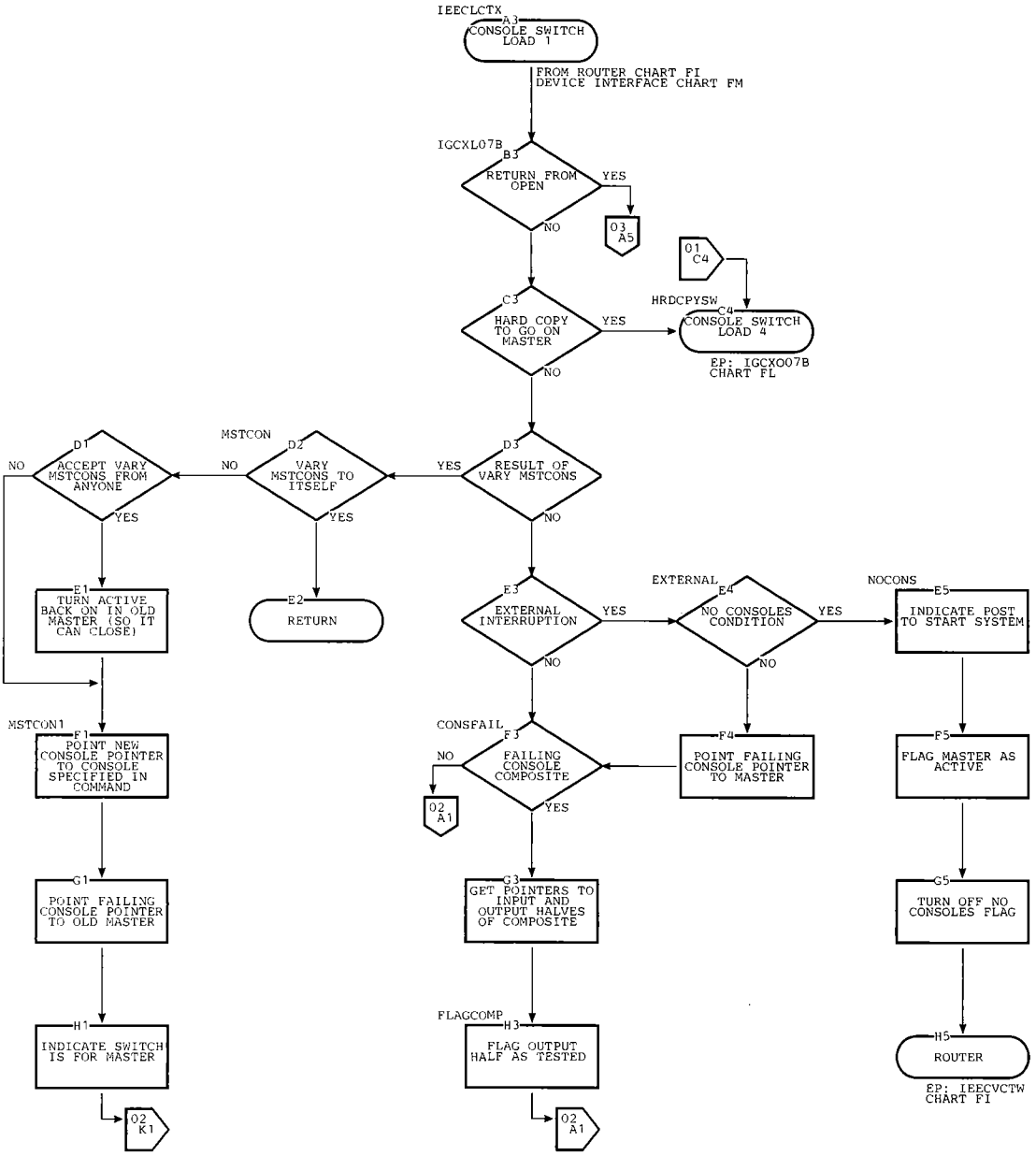


Chart FJ. Console Switch Load 1 -- Communications Task with Multiple Console Support (Page 2 of 3)

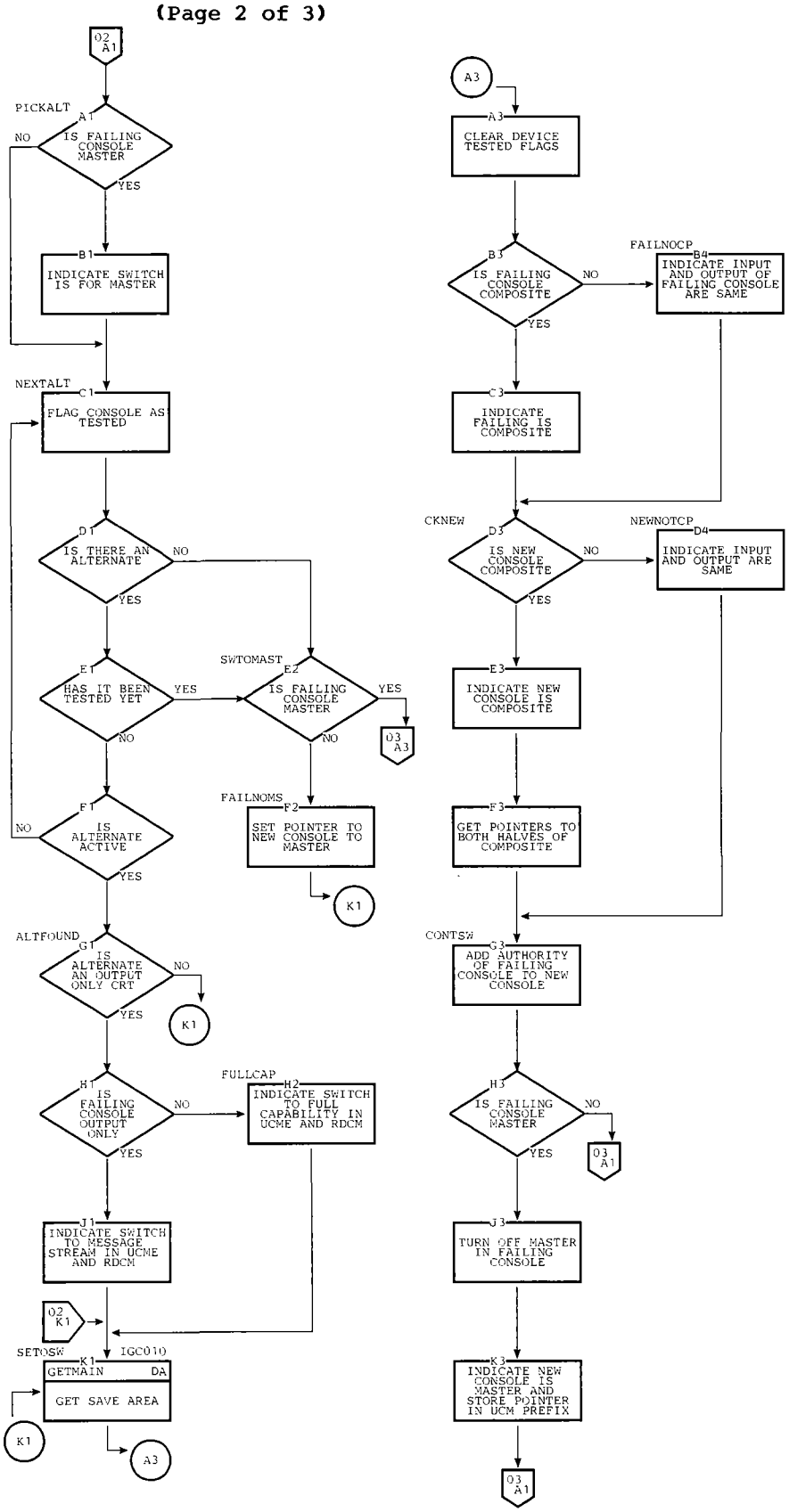


Chart FJ. Console Switch Load 1 -- Communications Task with Multiple Console Support (Page 3 of 3)

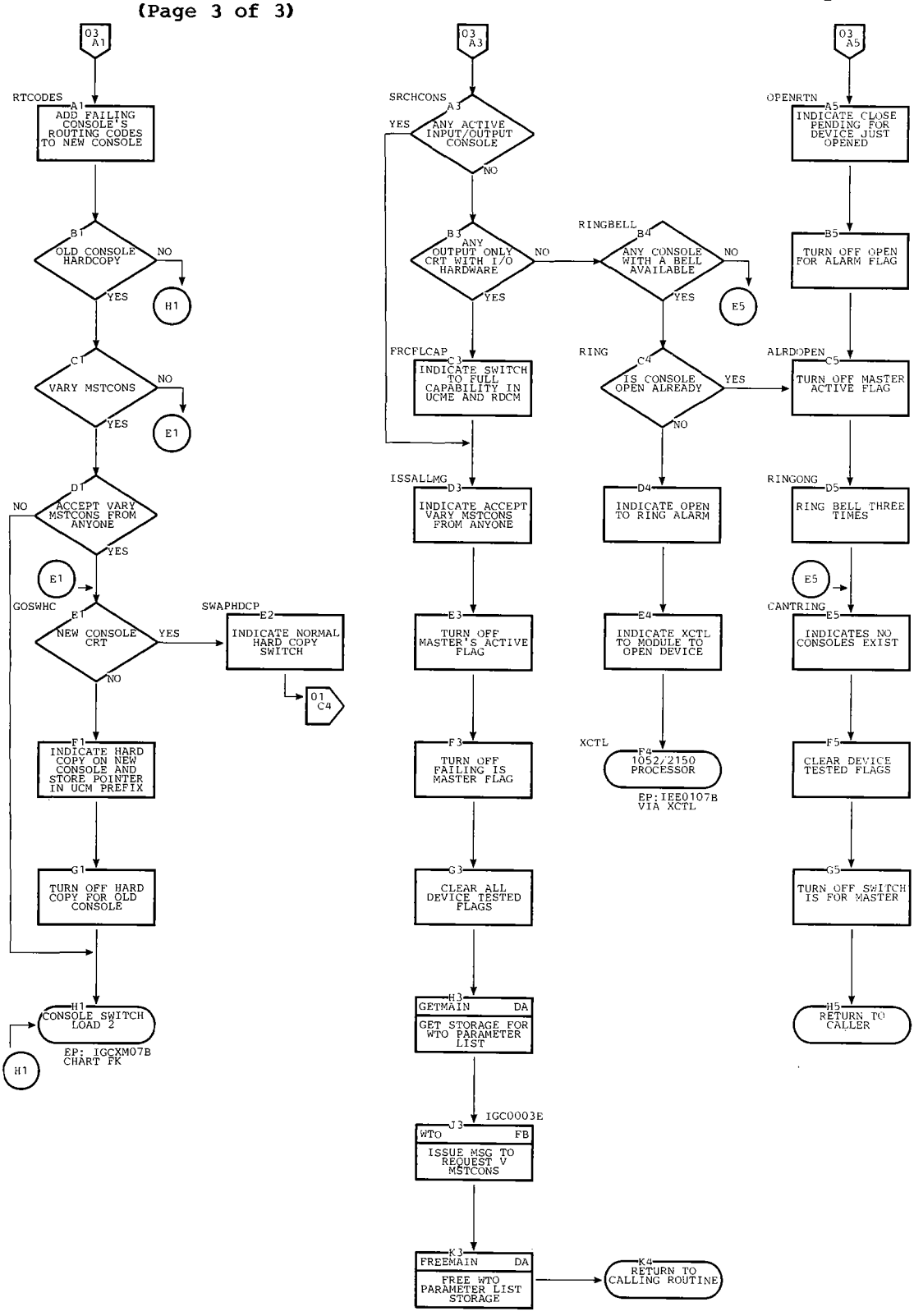


Chart FK. Console Switch Loads 2 and 3 -- Communications Task with Multiple Console Support (Page 1 of 2)

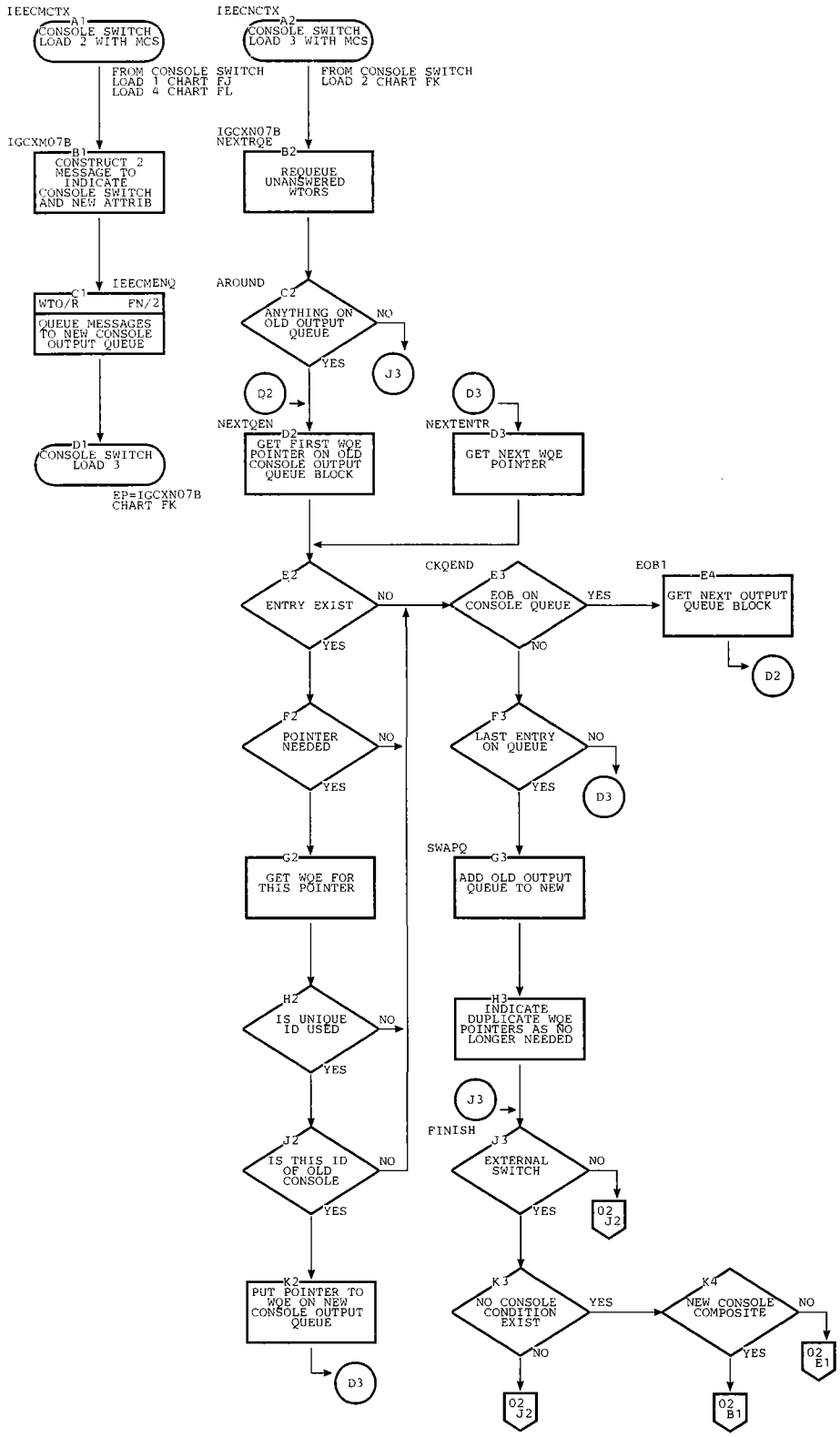


Chart FK. Console Switch Load 3 -- Communications Task with Multiple Console Support (Page 2 of 2)

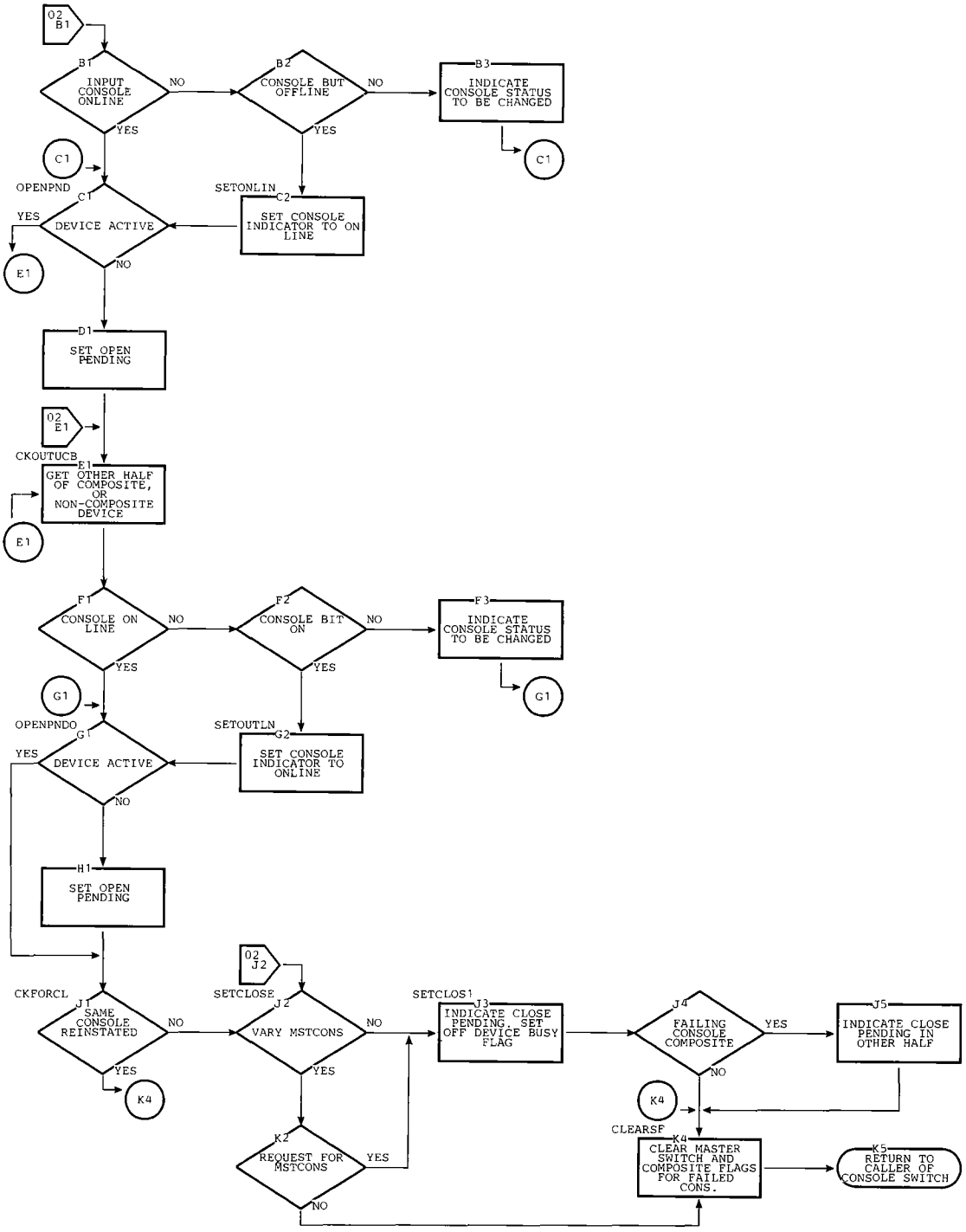


Chart FL. Console Switch Load 4 -- Communications Task with Multiple Console Support

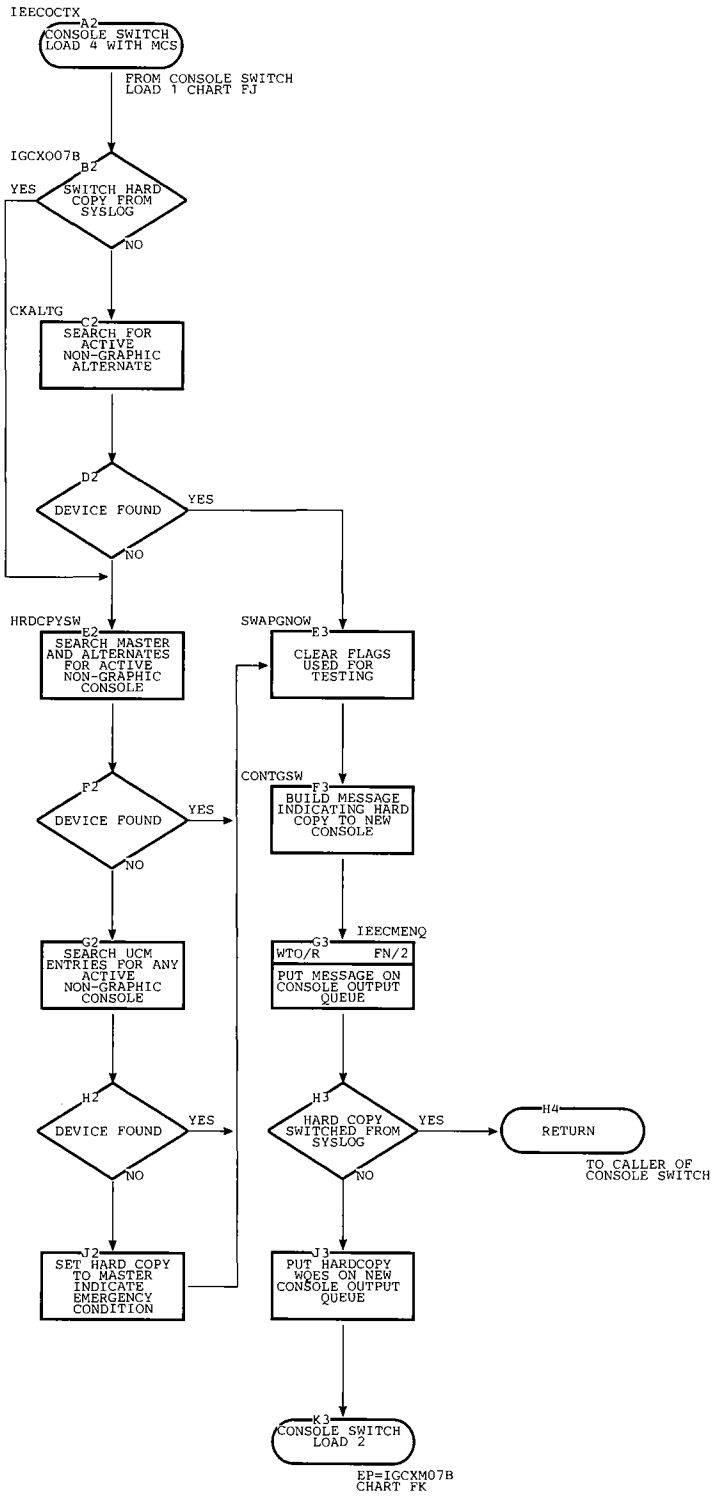


Chart FM. Device Interface -- Communications Task with Multiple Console Support (Page 1 of 3)

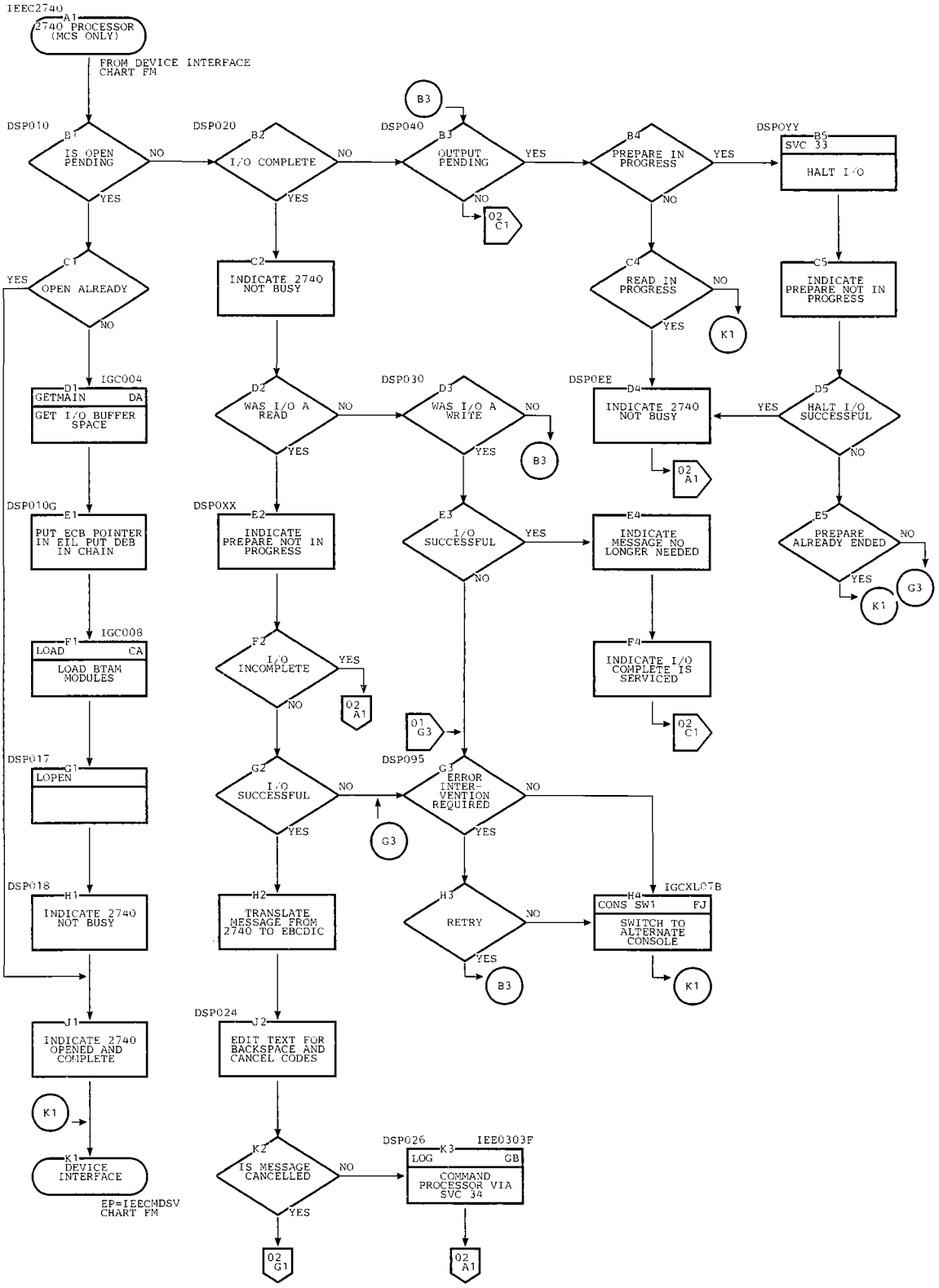


Chart FM. Device Interface -- Communications Task with Multiple Console Support (Page 2 of 3)

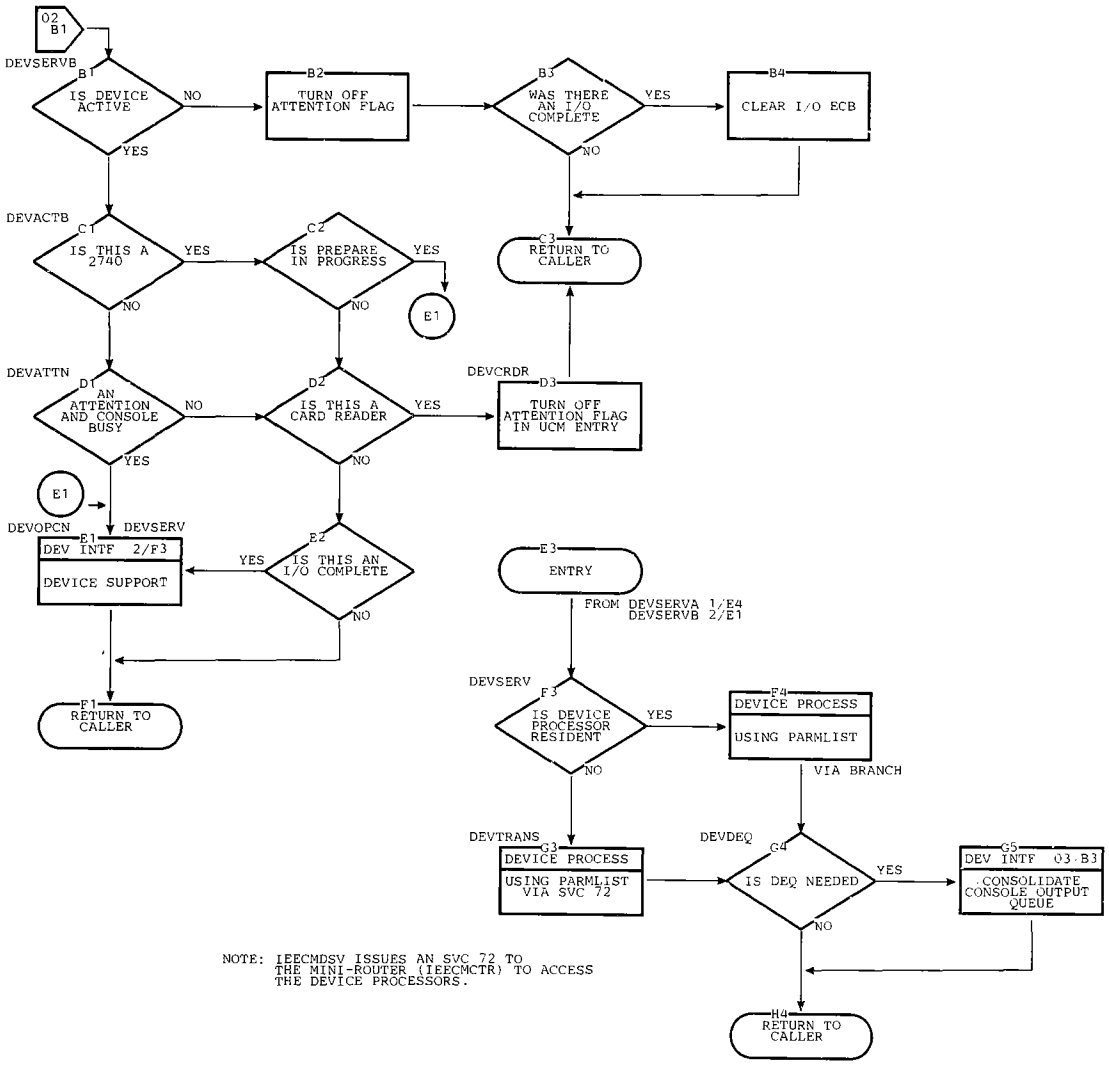


Chart FM. Device Interface -- Communications Task with Multiple Console Support (Page 3 of 3)

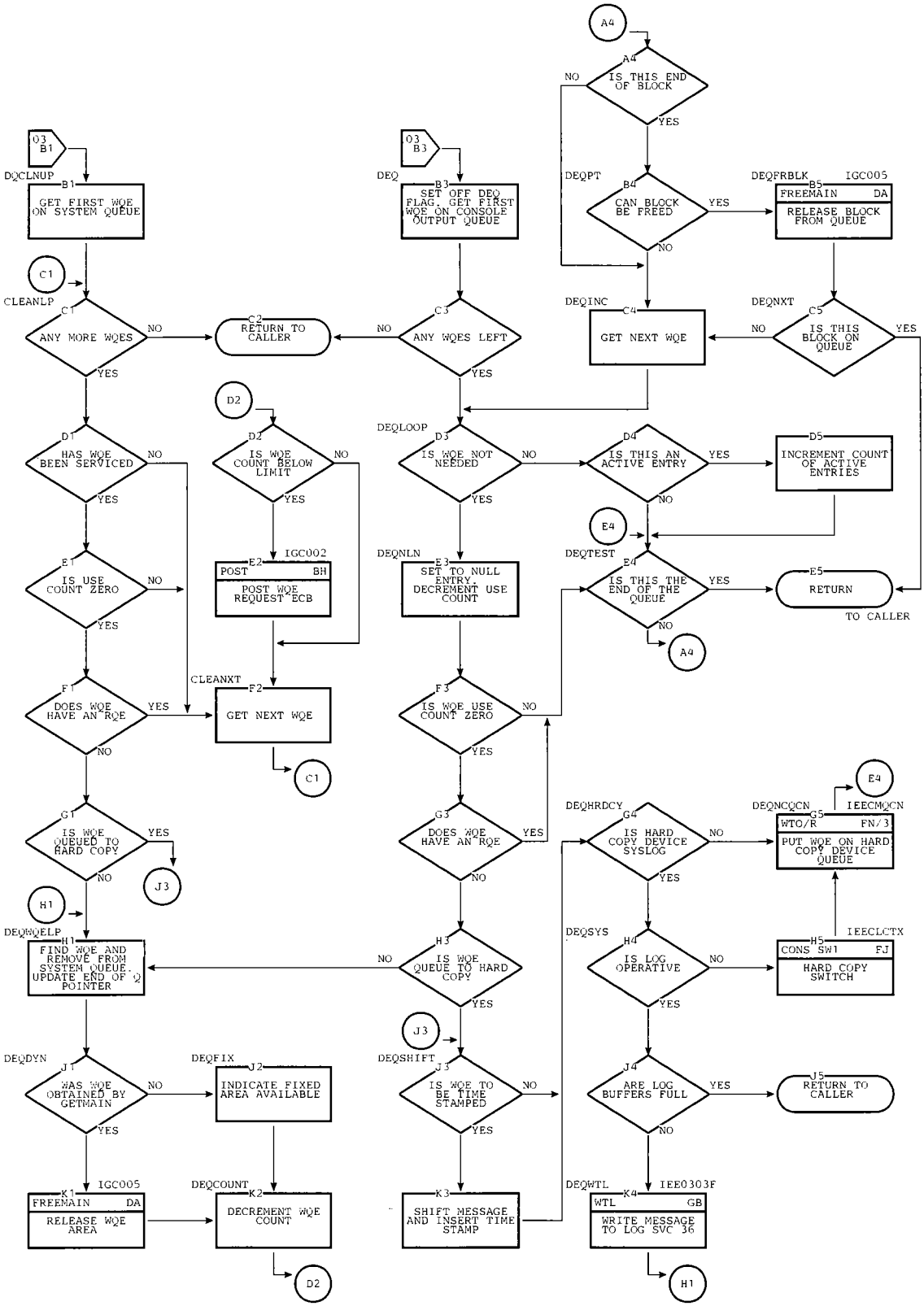


Chart FN. WTO/R Processor -- Communications Task with Multiple Console Support (Page 1 of 3)

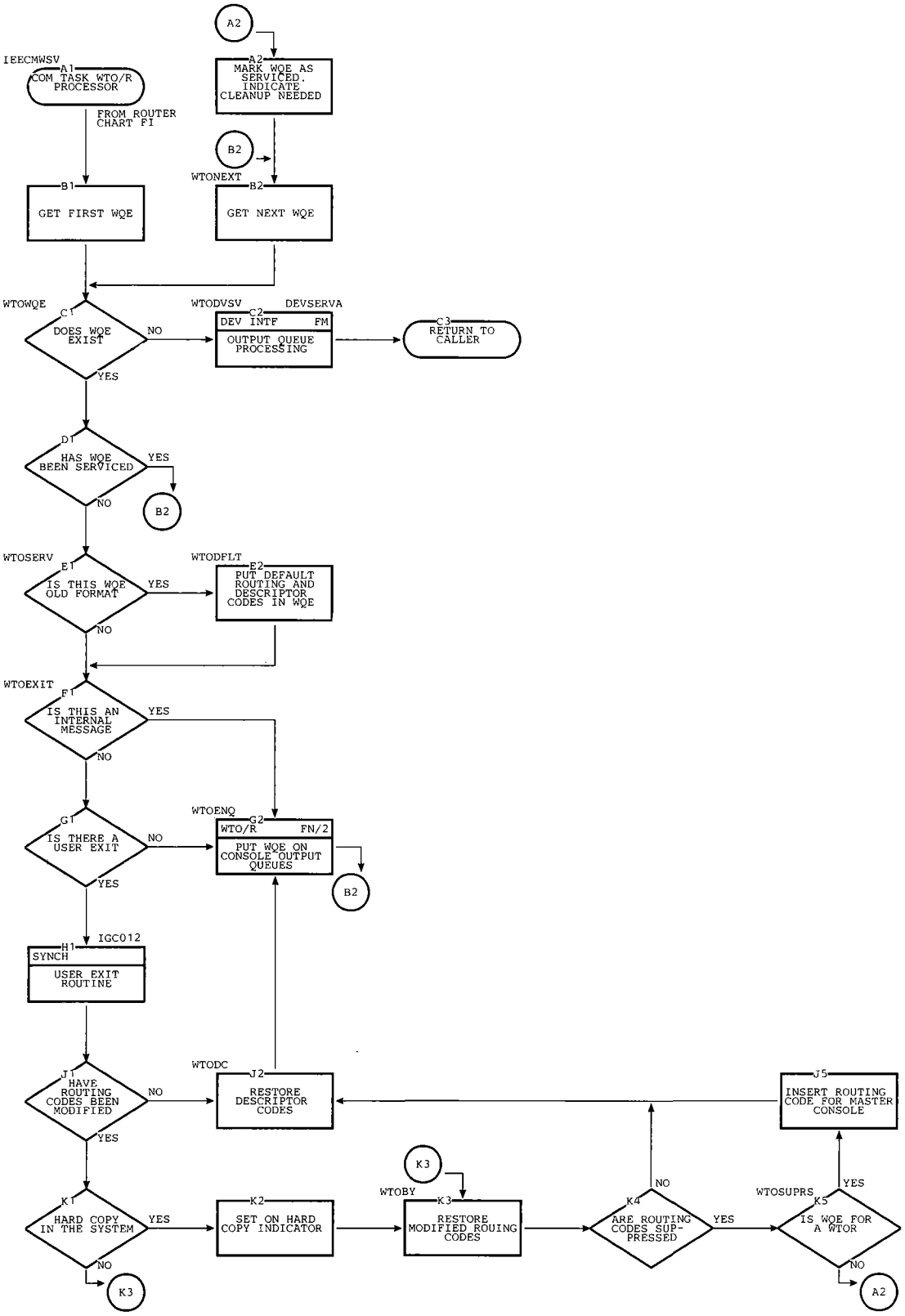


Chart FN. WTO/R Processor -- Communications Task with Multiple Console Support (Page 2 of 3)

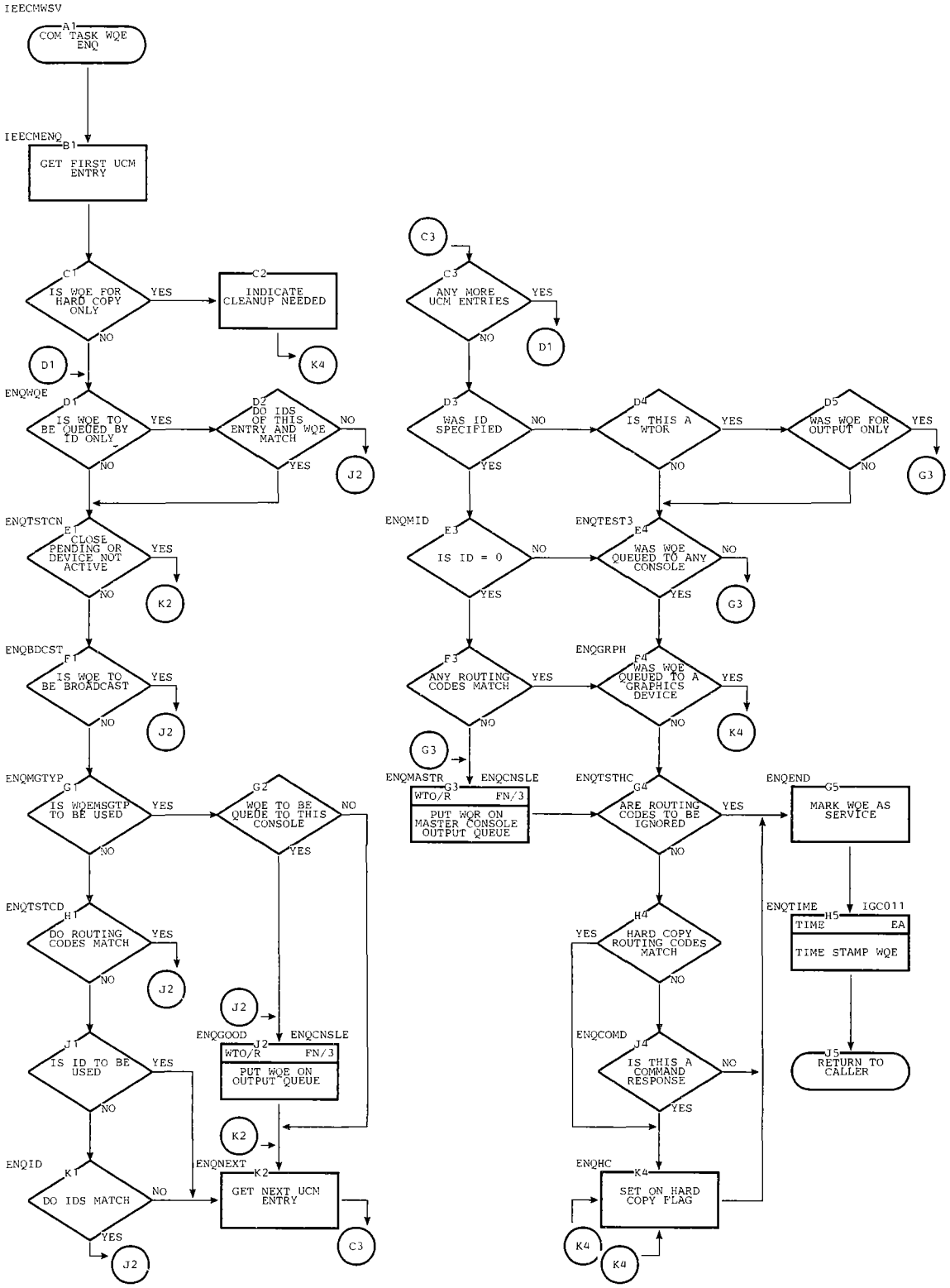


Chart FN. WTO/R Processor -- Communications Task with Multiple Console Support (Page 3 of 3)

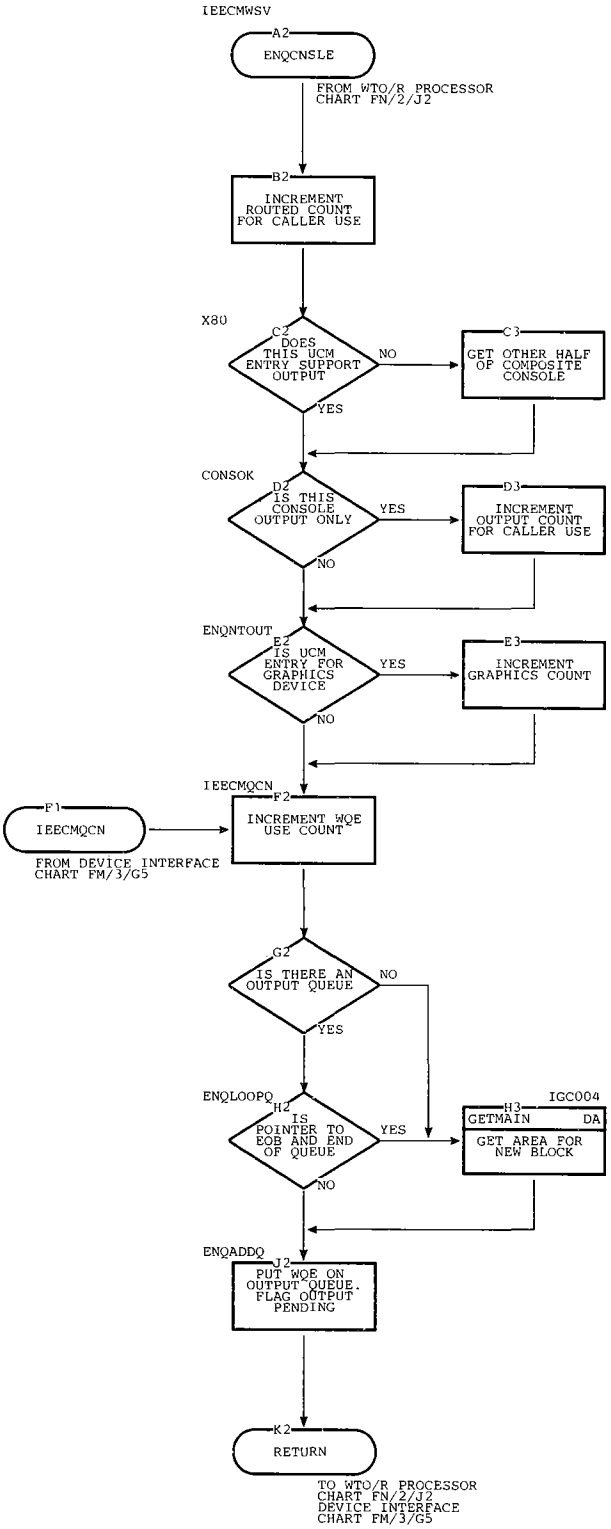


Chart FO. Delete-Operator-Message (DOM) Processor -- Communications Task with Multiple Console Support

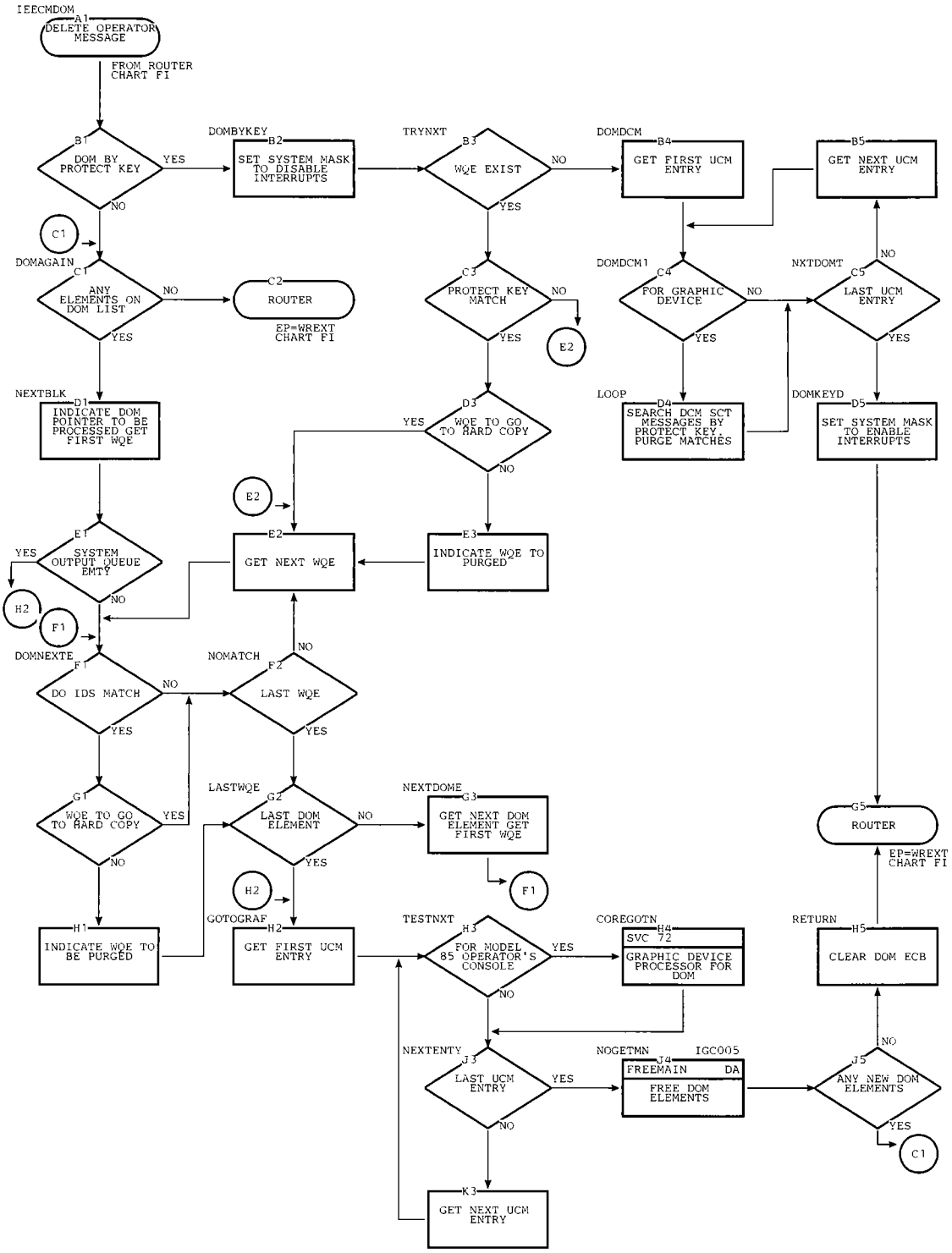


Chart FP. NIP Message Buffer Writer -- Communications Task with Multiple Console Support

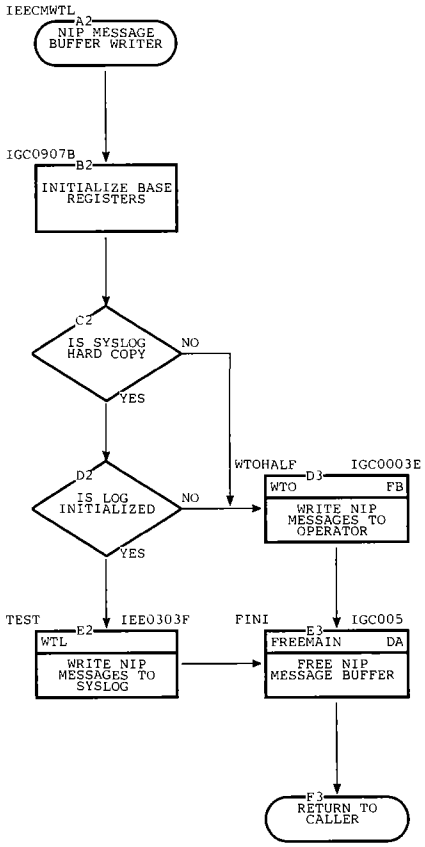


Chart FQ. 1052 Processor 1 -- Communications Task without Multiple Console Support (Page 1 of 2)

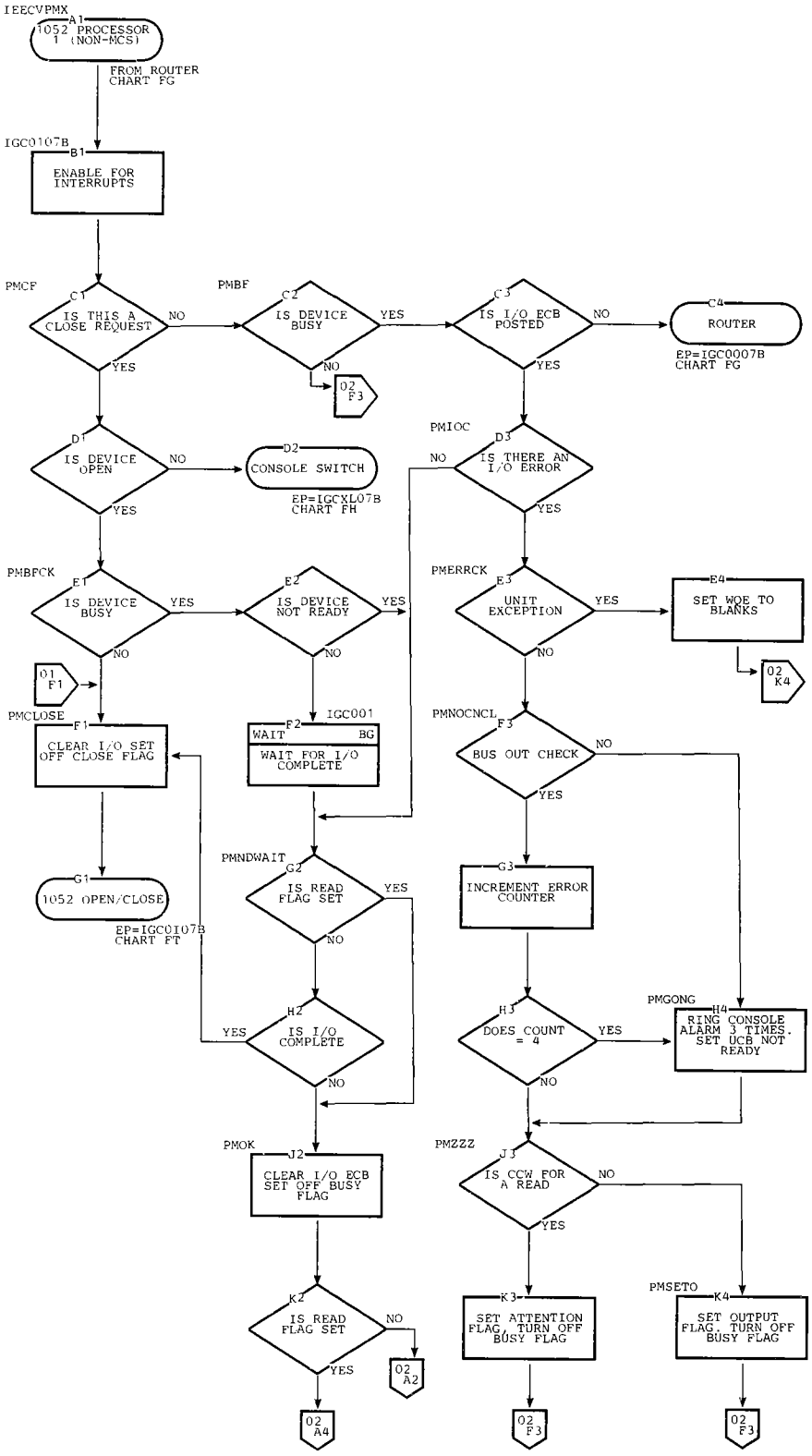


Chart FQ. 1052 Processor 1 -- Communications Task without Multiple Console Support (Page 2 of 2)

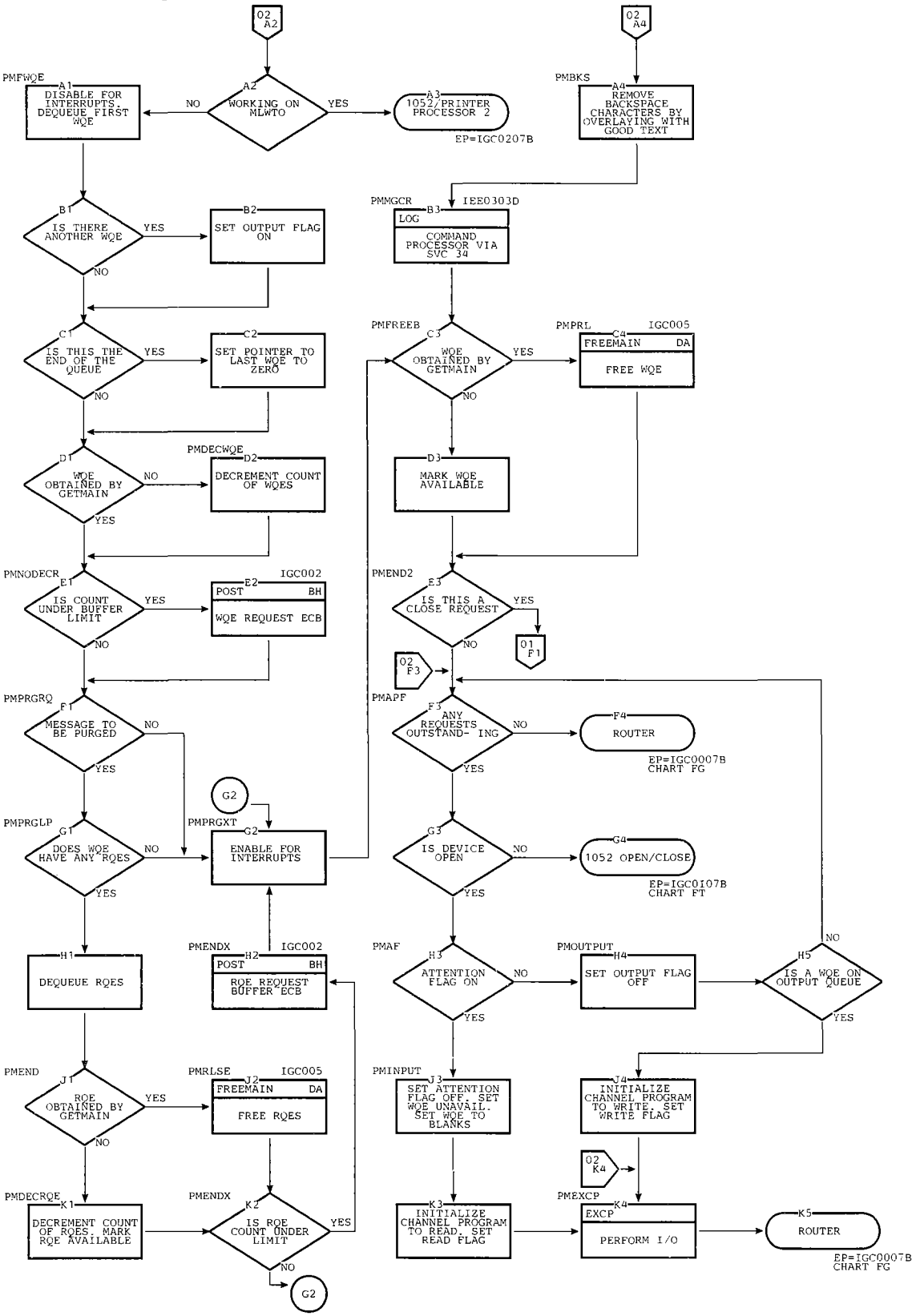


Chart FR. 1052 Processor 1 -- Communications Task with Multiple Console Support (Page 1 of 2)

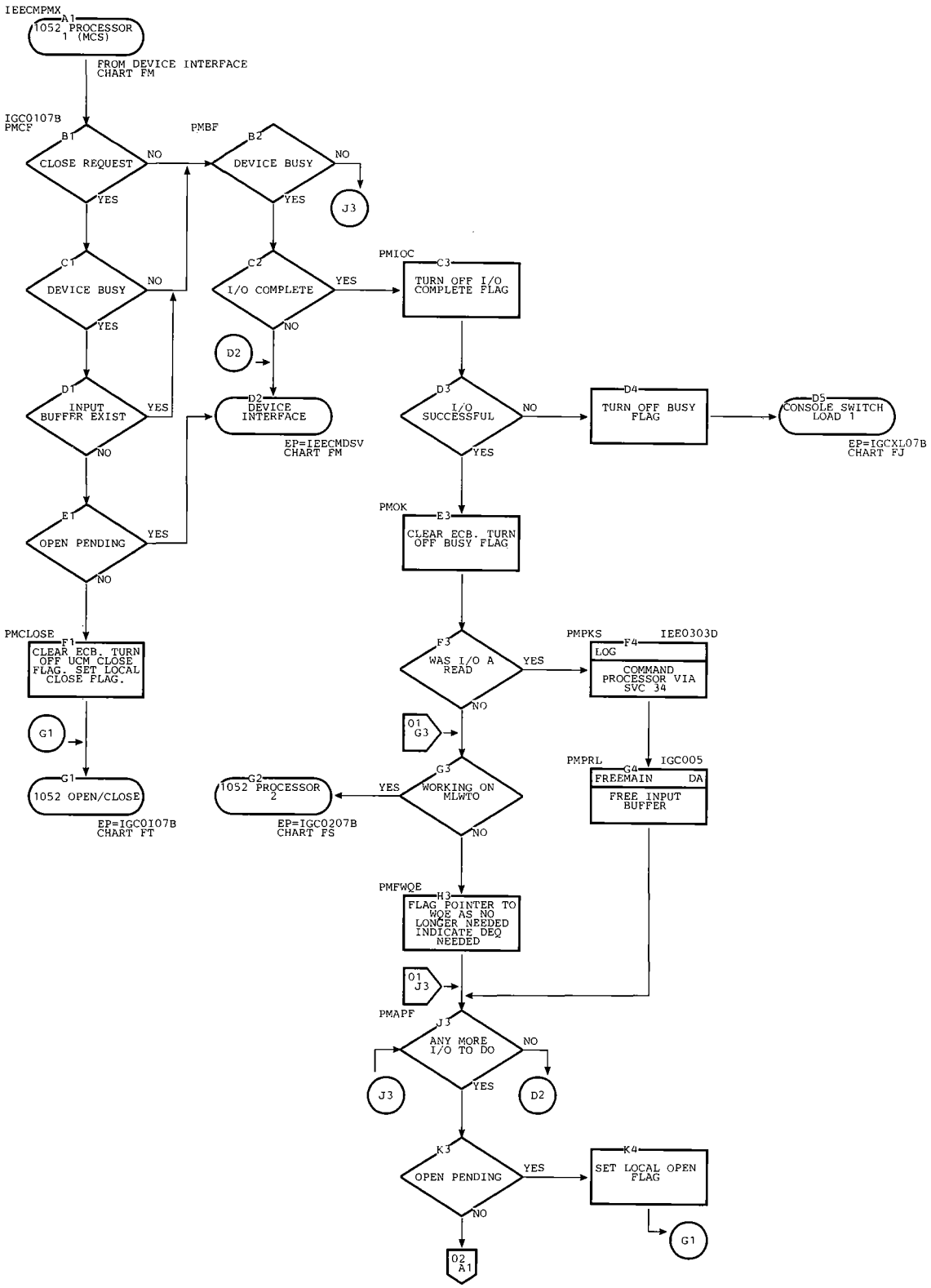


Chart FR. 1052 Processor 1 -- Communications Task with Multiple Console Support (Page 2 of 2)

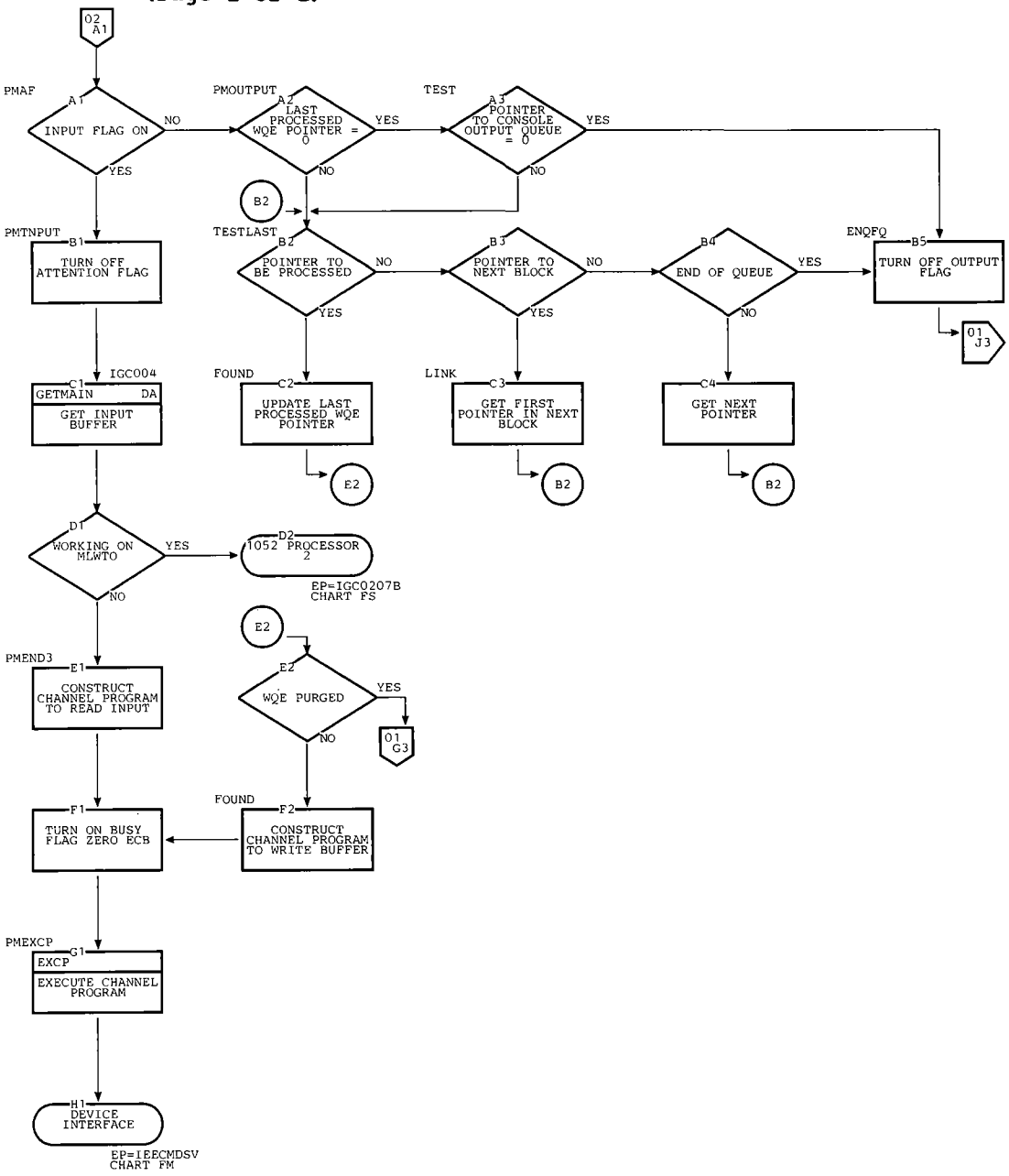


Chart FS. 1052 Processor 2 -- Communications Task with Multiple Console Support (Page 1 of 2)

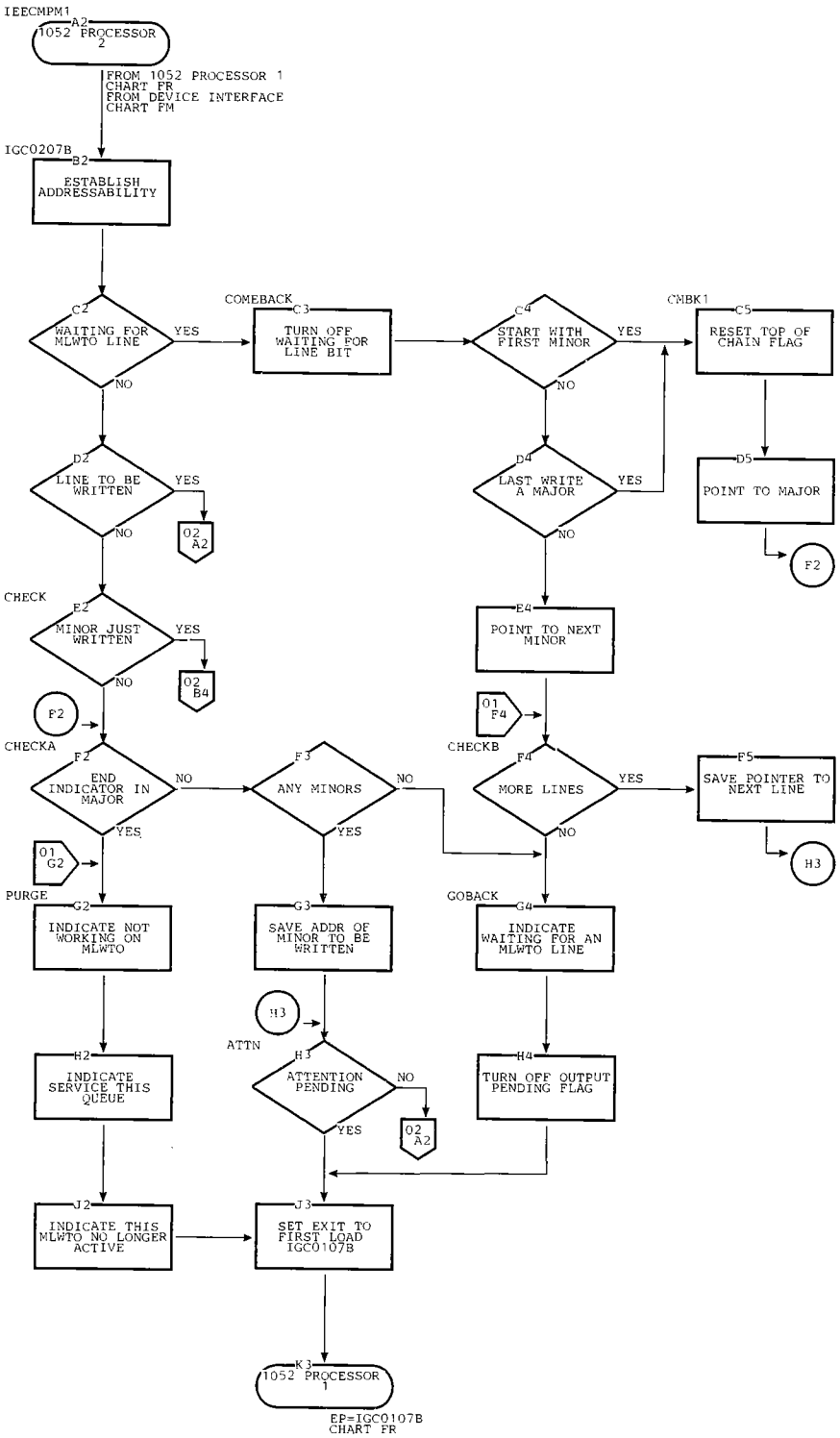


Chart FS. 1052 Processor 2 -- Communications Task with Multiple Console Support (Page 2 of 2)

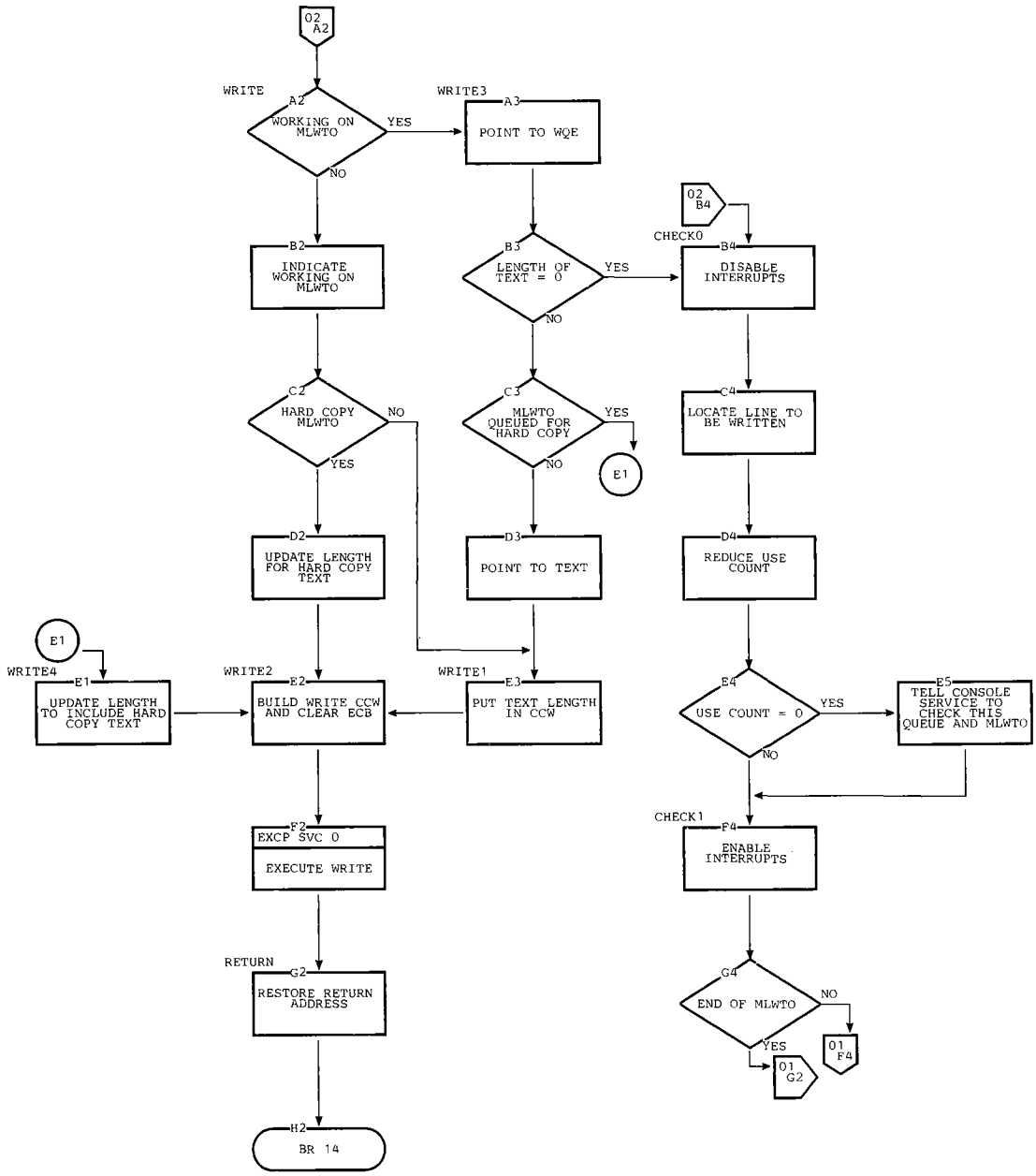


Chart FT. 1052 Open/Close

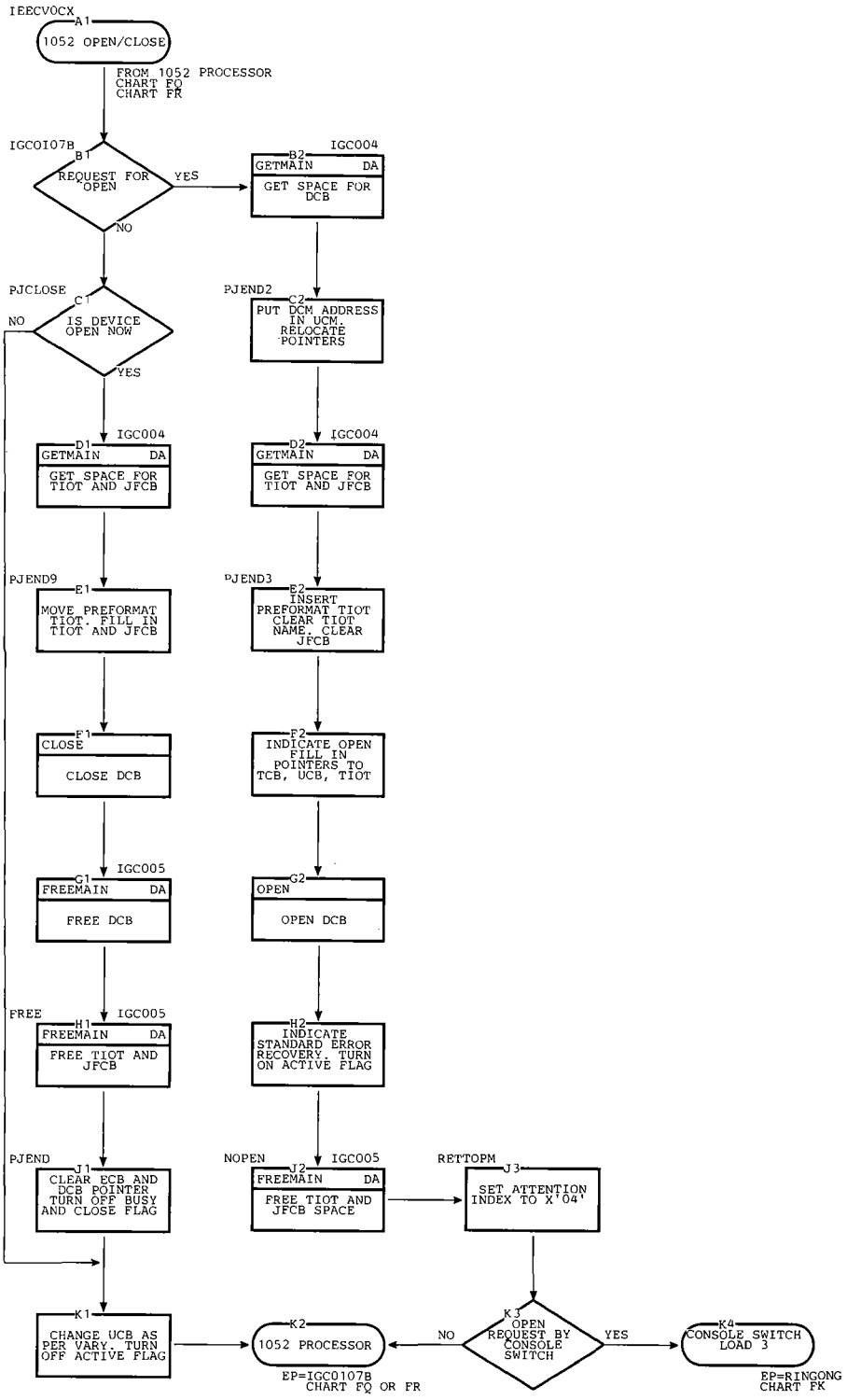


Chart FU. 2540 Processor -- Communications Task without Multiple Console Support

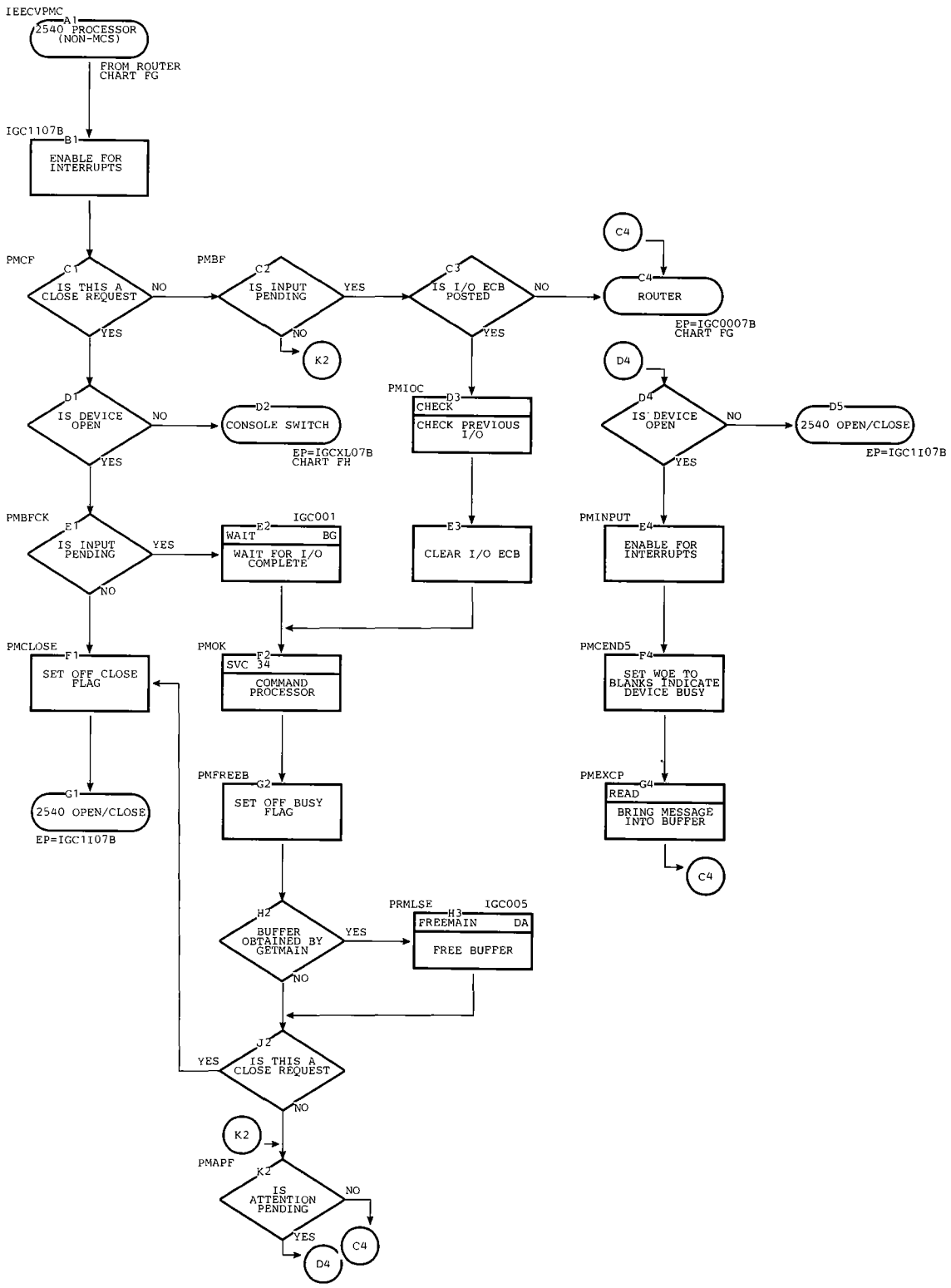


Chart FV. 2540 Processor -- Communications Task with Multiple Console Support

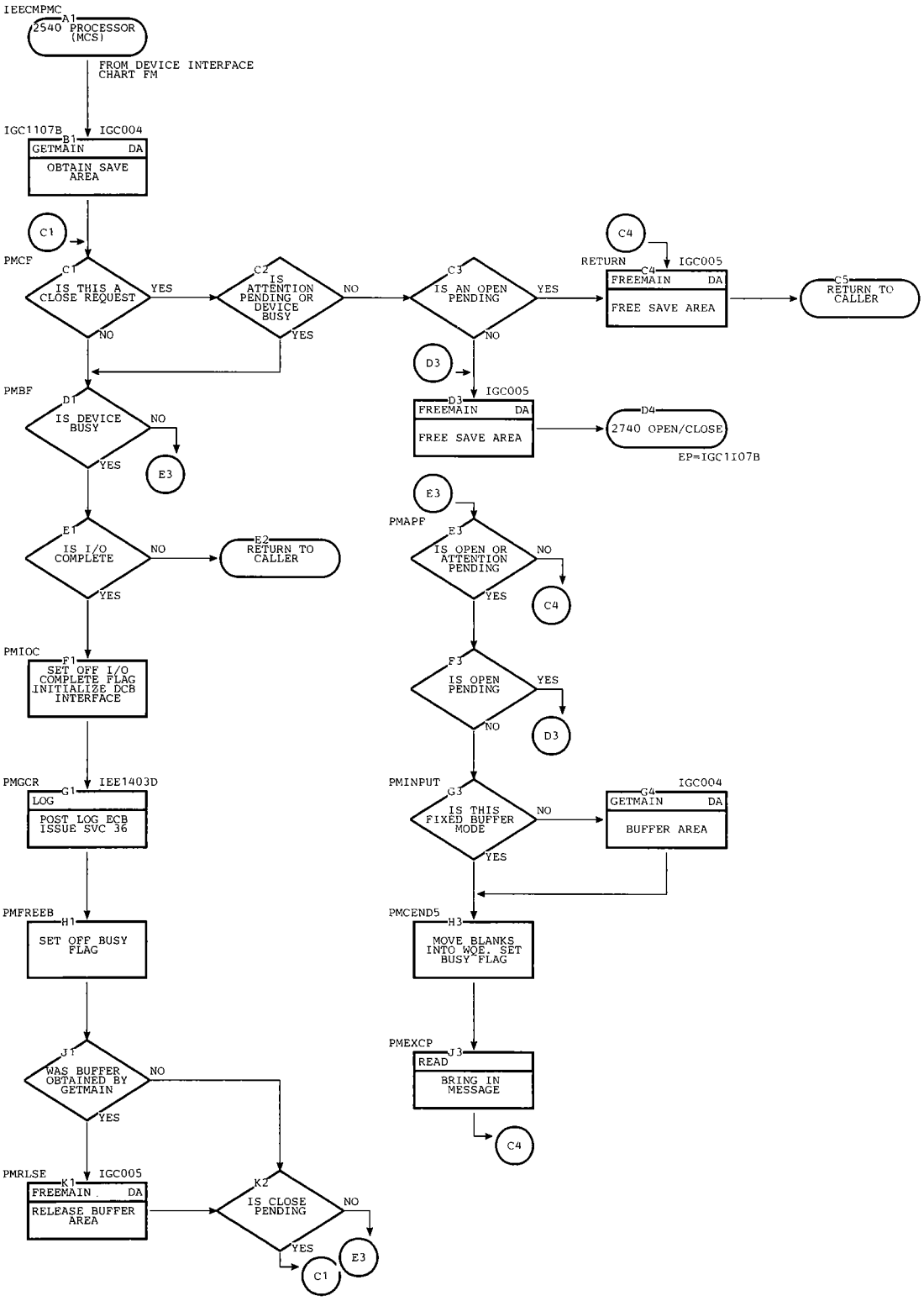


Chart FW. 2740 Processor -- Communications Task with Multiple Console Support (Page 1 of 2)

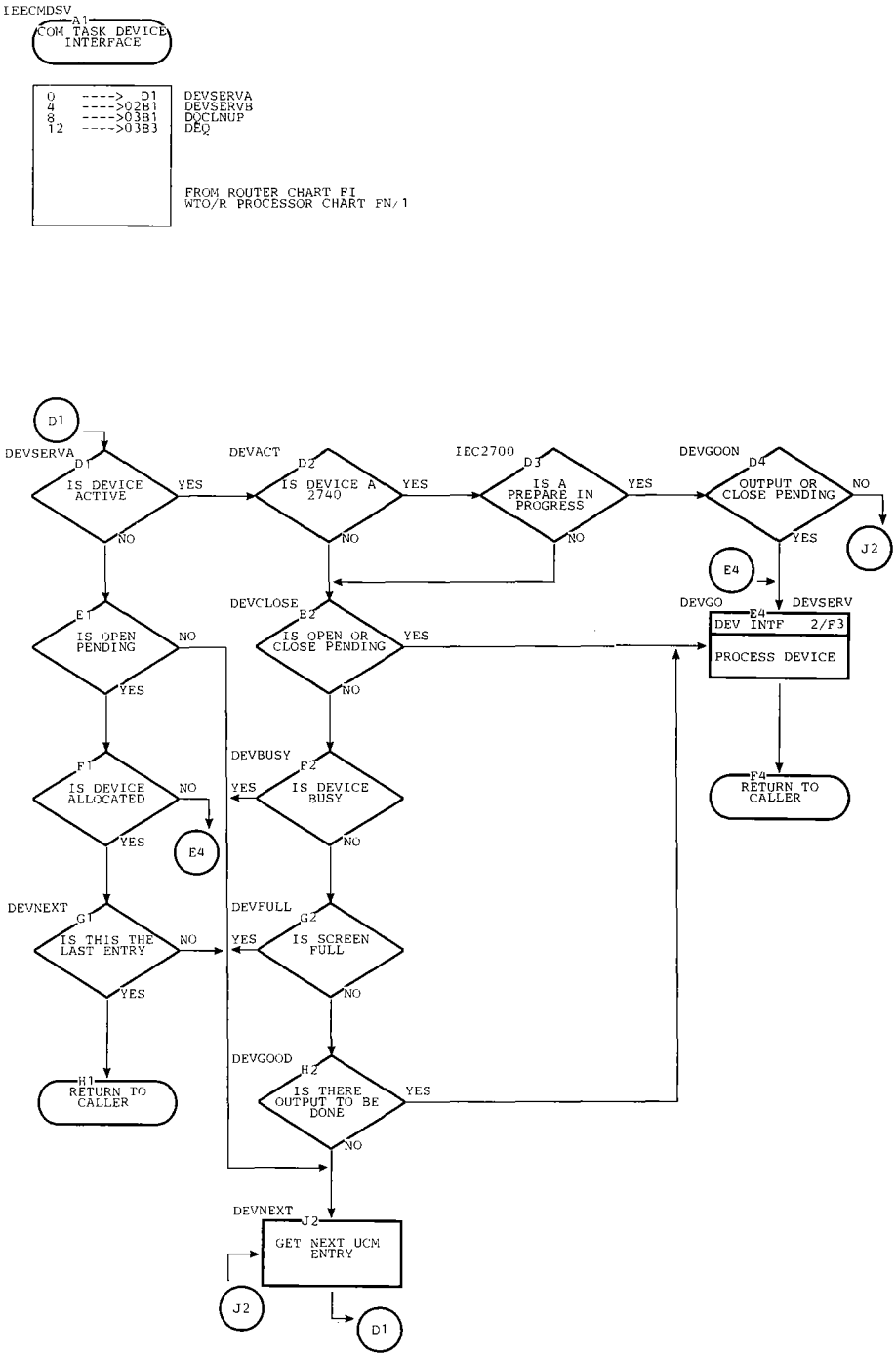


Chart FW. 2740 Processor -- Communications Task with Multiple Console Support (Page 2 of 2)

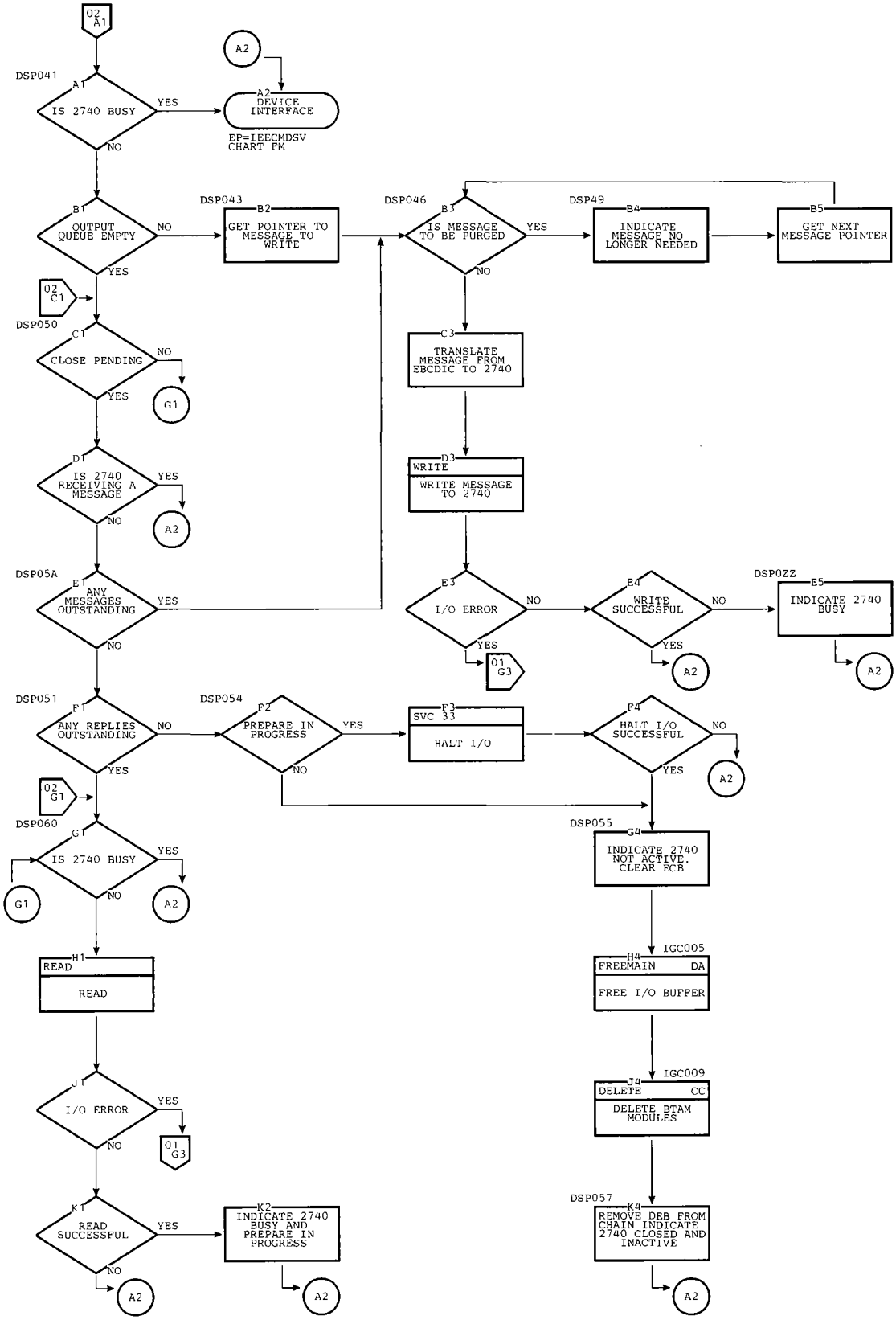


Chart GA. Log Writer

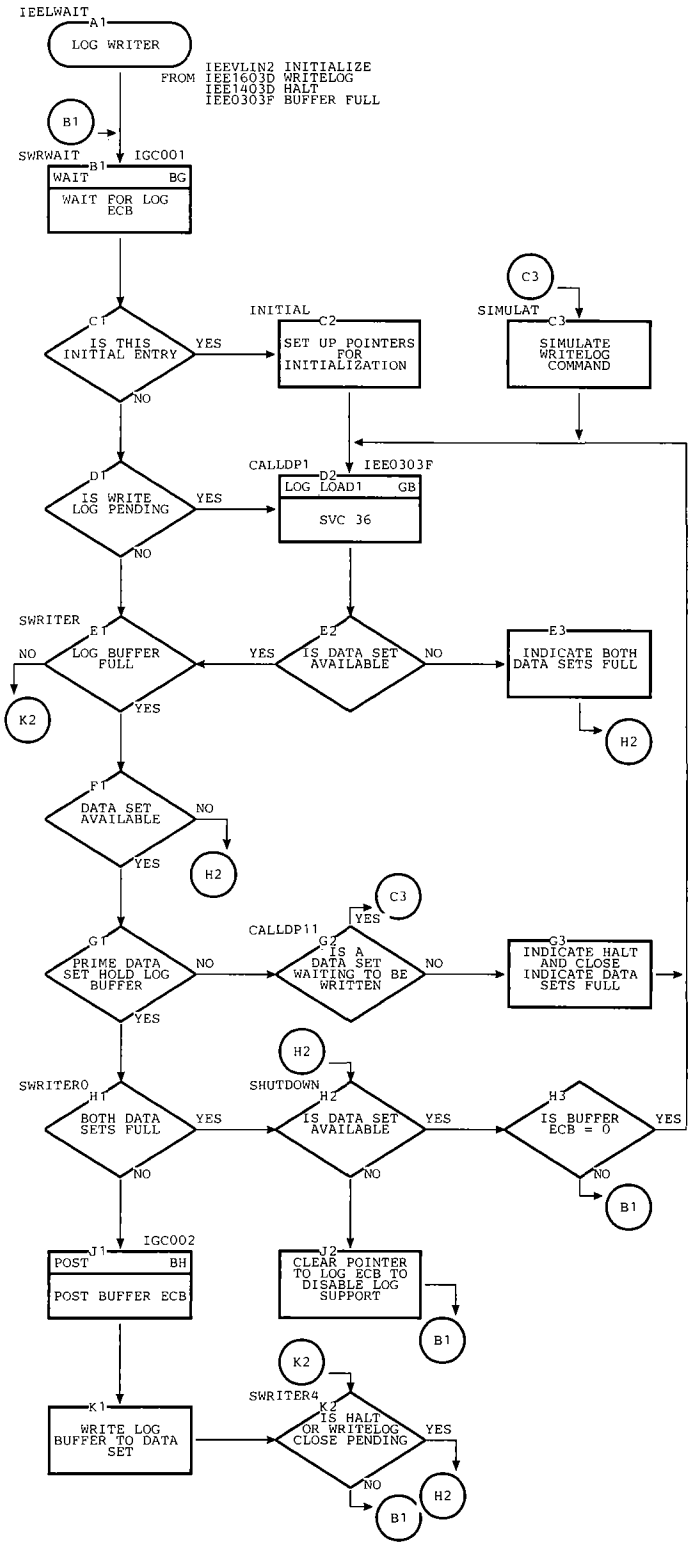


Chart GB. LOG, Write-to-Log -- Load 1

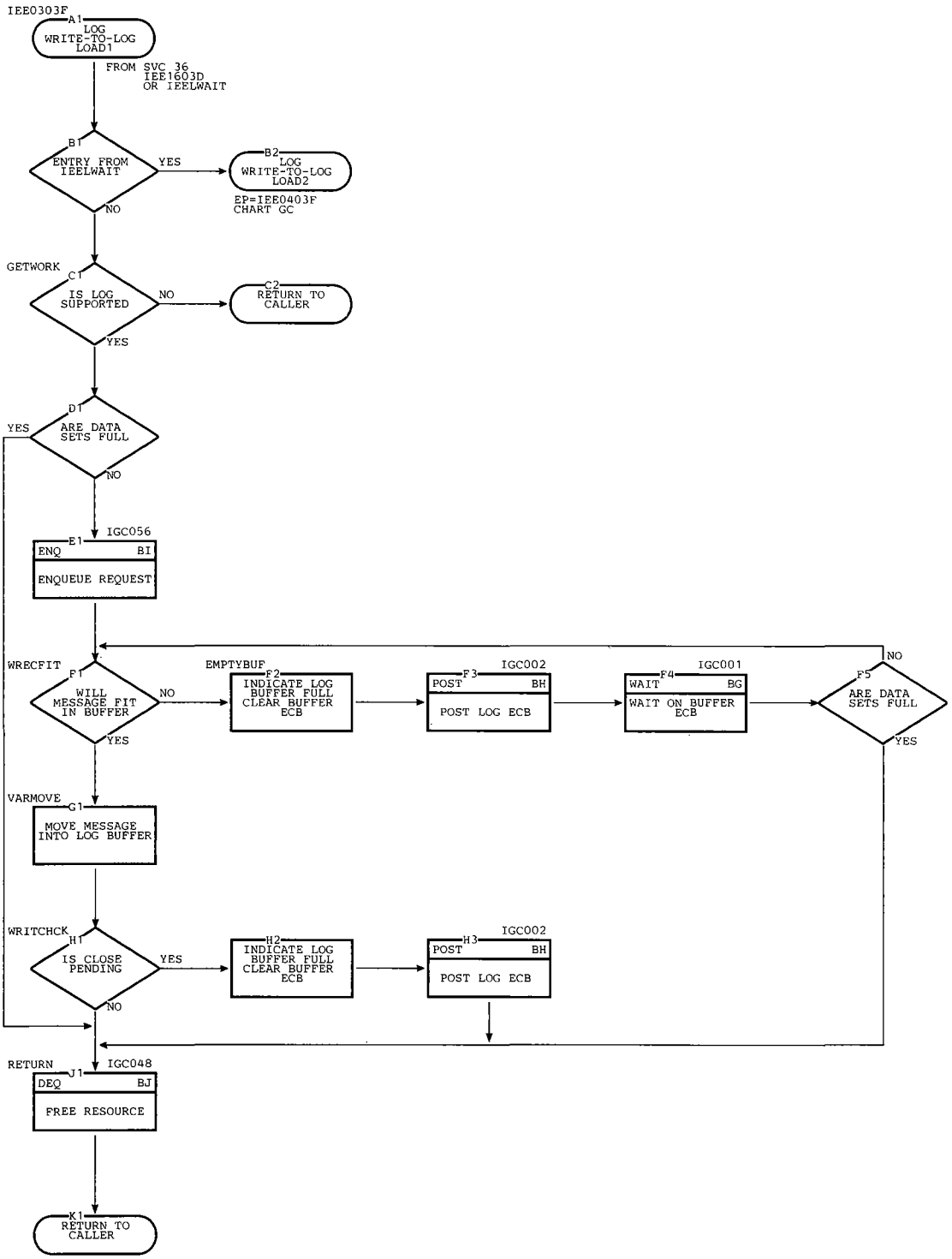


Chart GC. LOG, Write-to-Log -- Load 2

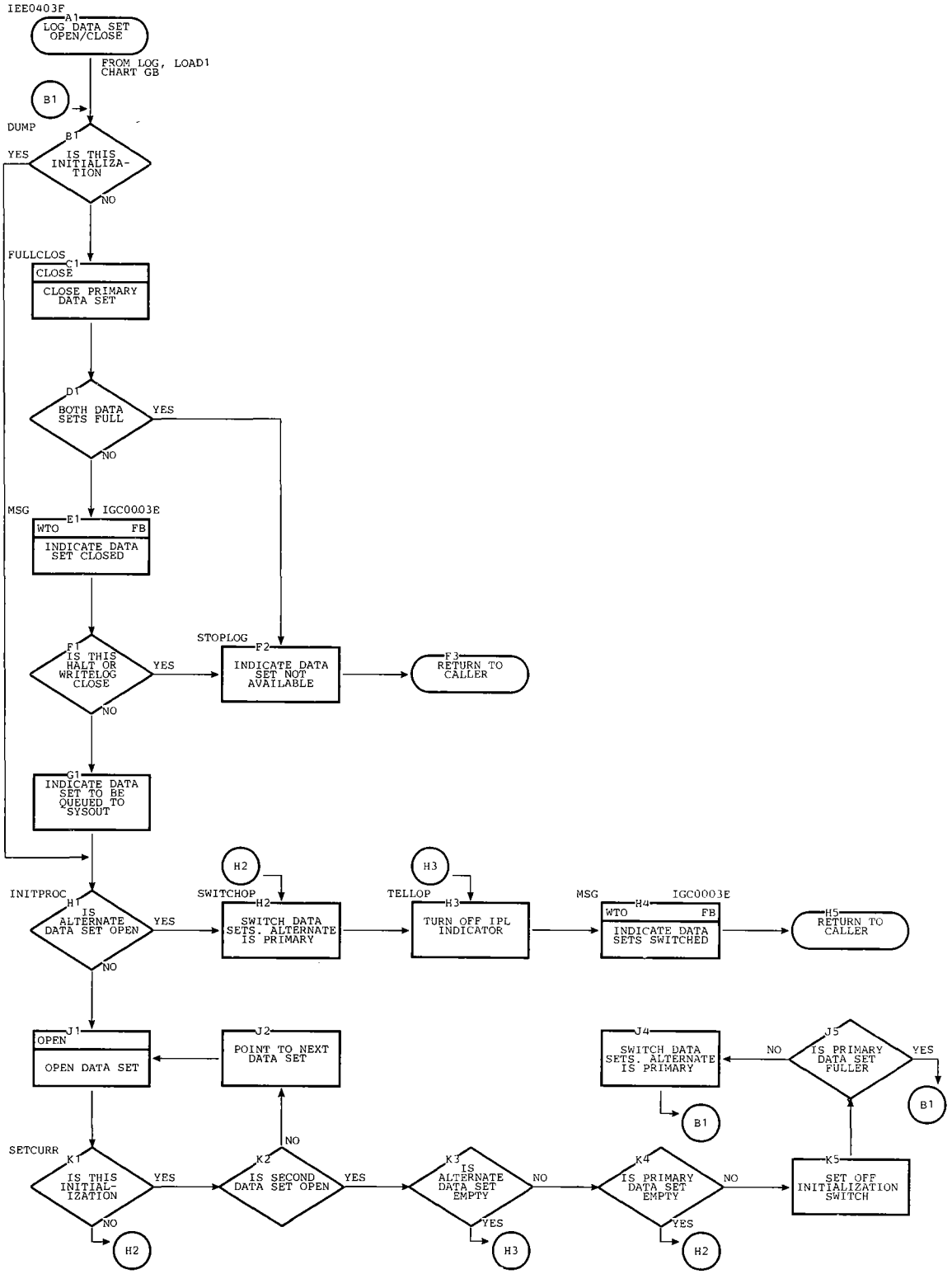


Chart GD. Multiple-Line Write-to-Operator -- Load 1

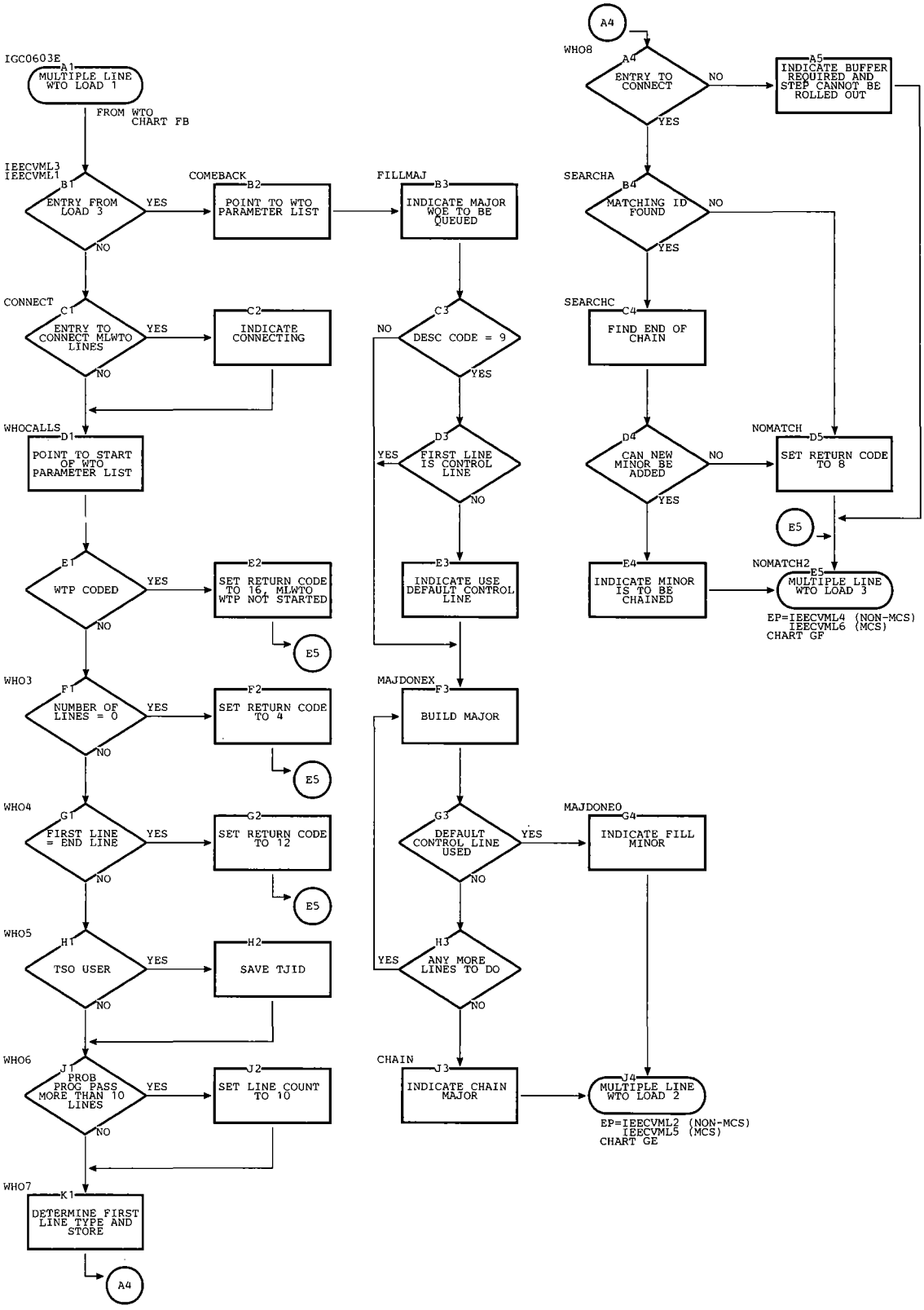


Chart GE. Multiple-Line Write-to-Operator -- Load 2

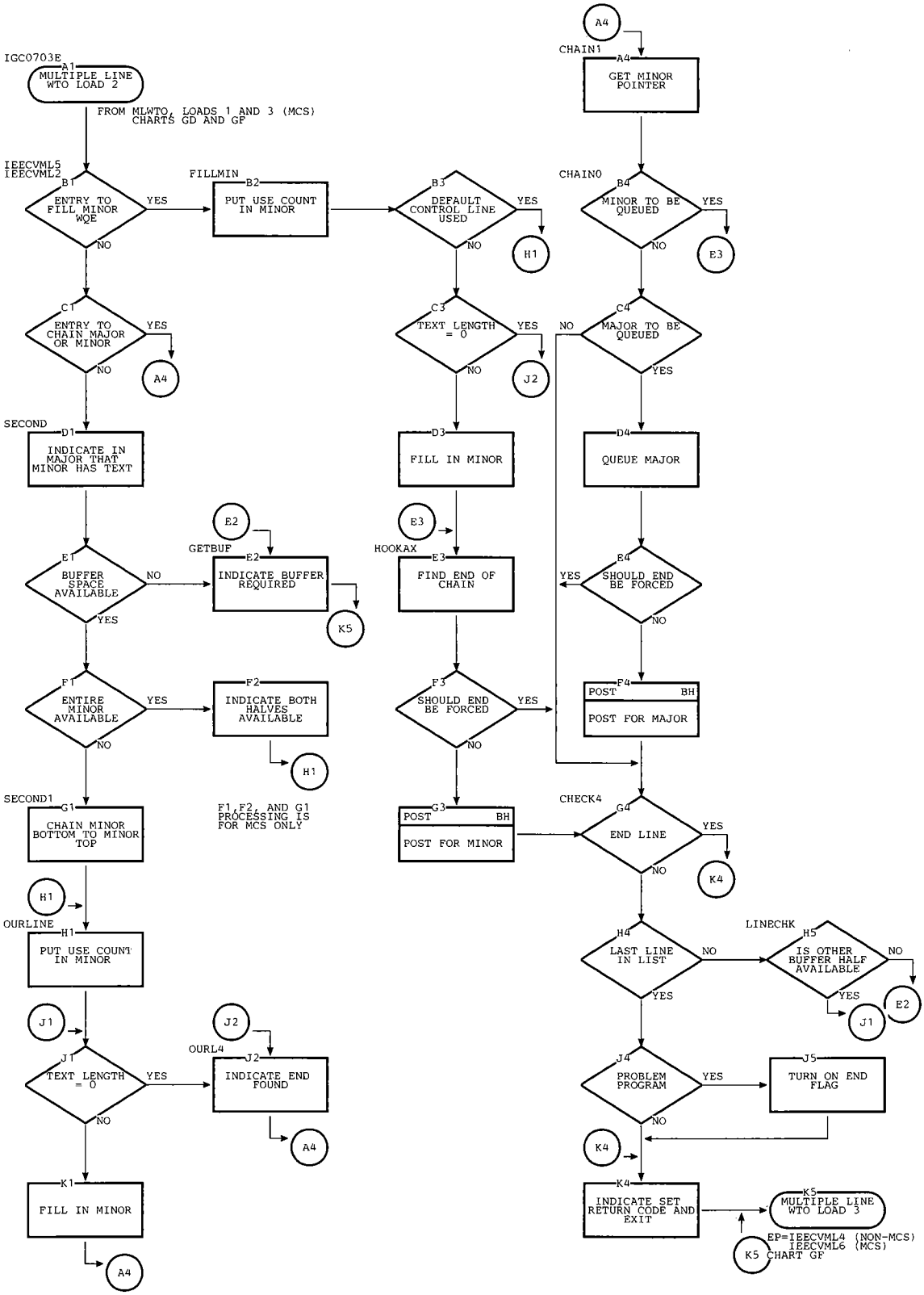


Chart GF. Multiple-Line Write-to-Operator -- Load 3

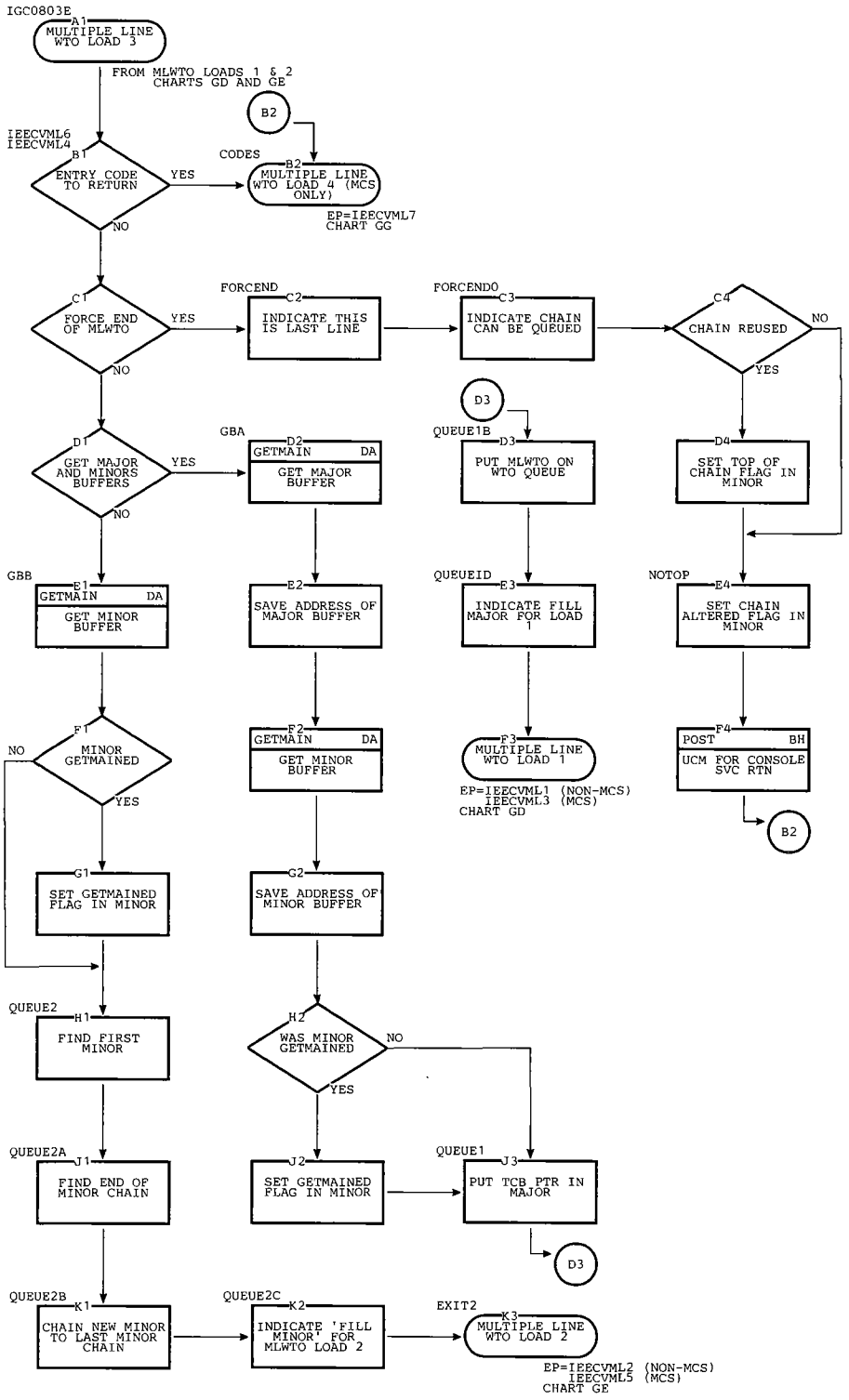


Chart GG. Multiple-Line Write-to-Operator -- Load 4

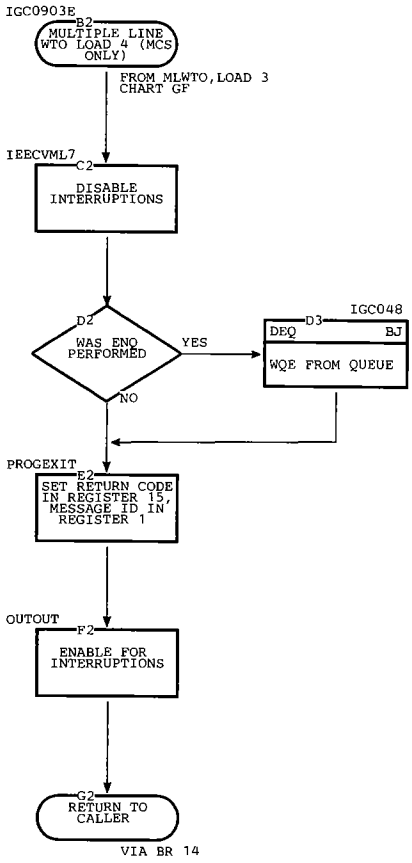


Chart HA. DIDOCS Processor 0, Load 1 (Page 1 of 2)

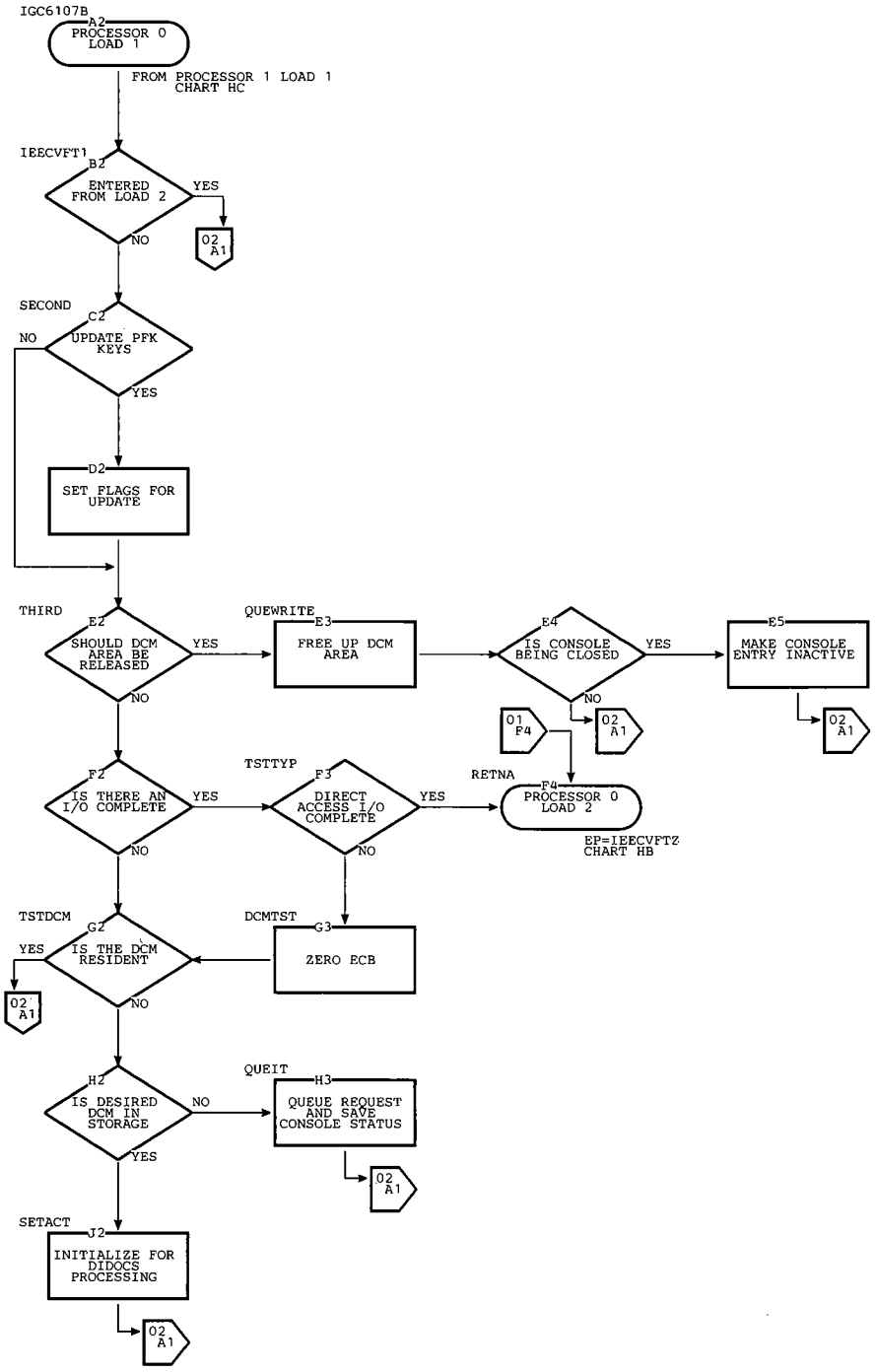


Chart HA. DIDOCS Processor 0, Load 1 (Page 2 of 2)

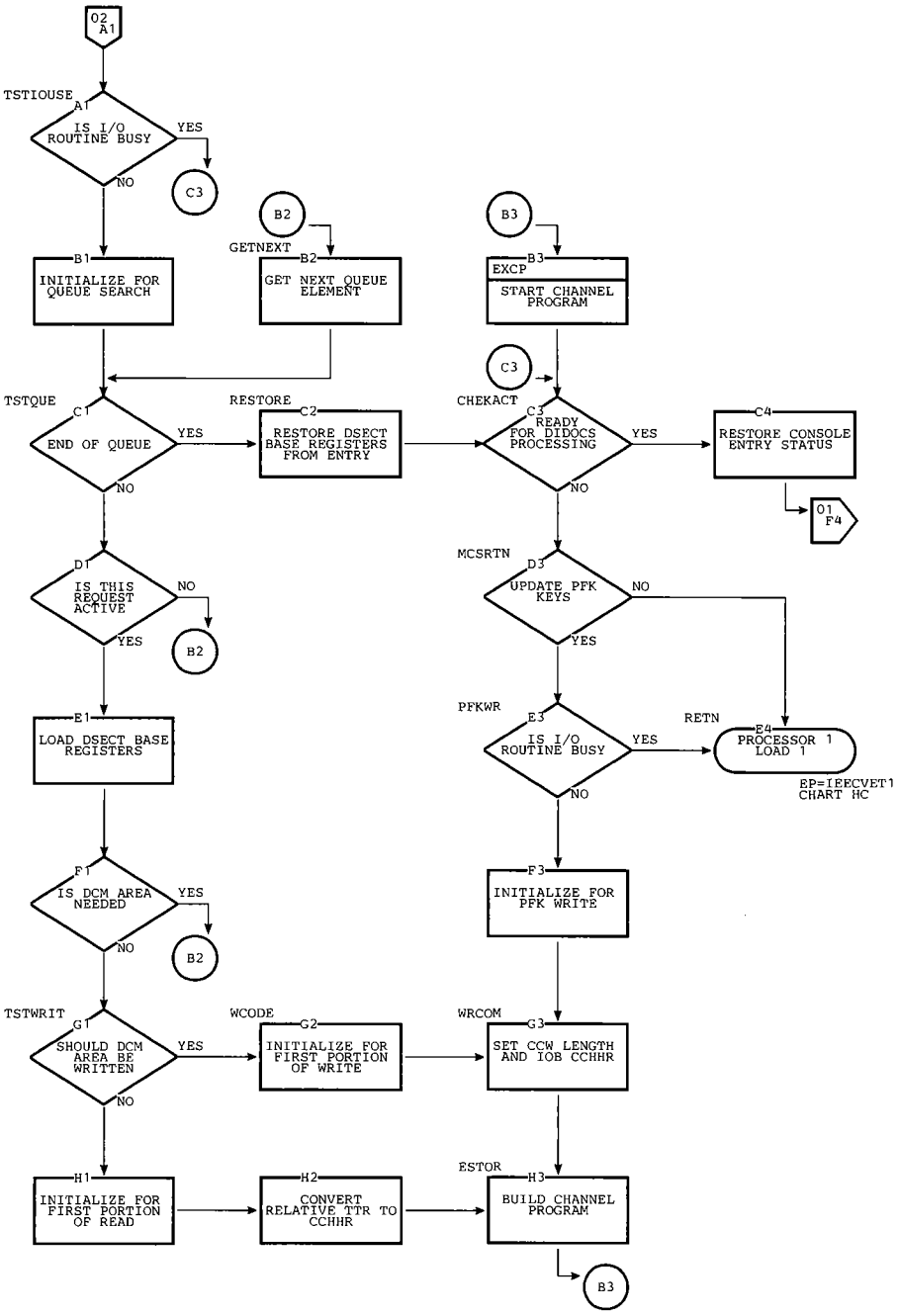


Chart HB. DIDOCS Processor 0, Load 2 (Page 1 of 2)

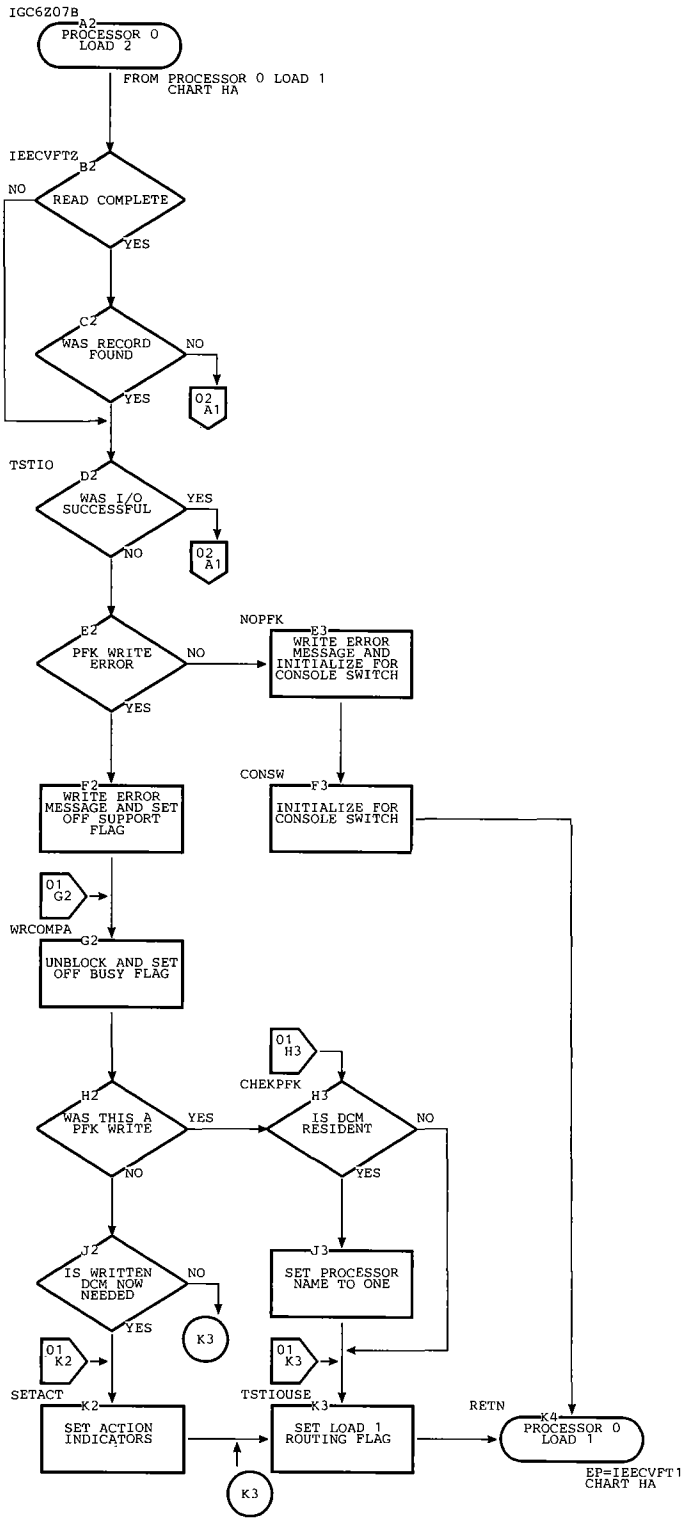


Chart HB. DIDICS Processor 0, Load 2 (Page 2 of 2)

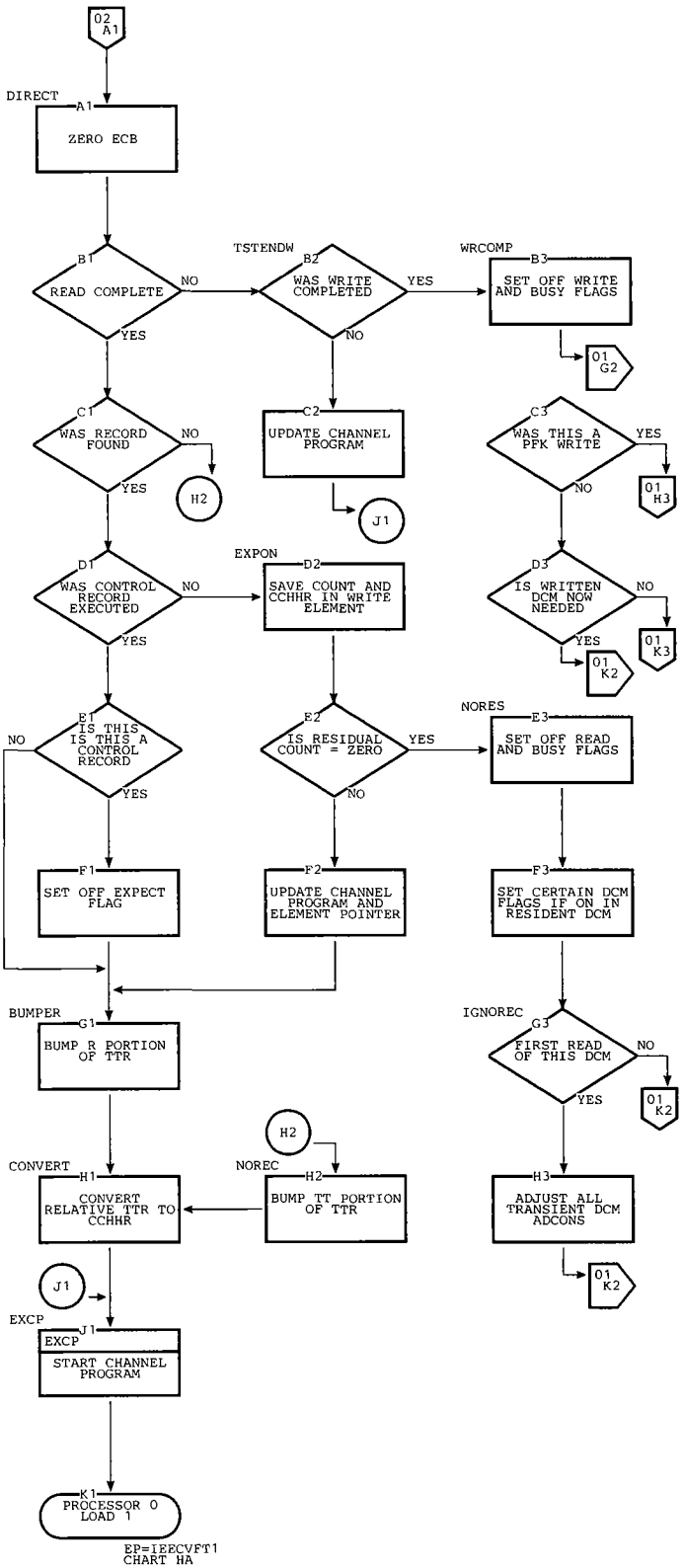


Chart HC. DIDOCS Processor 1, Load 1 (Page 1 of 2)

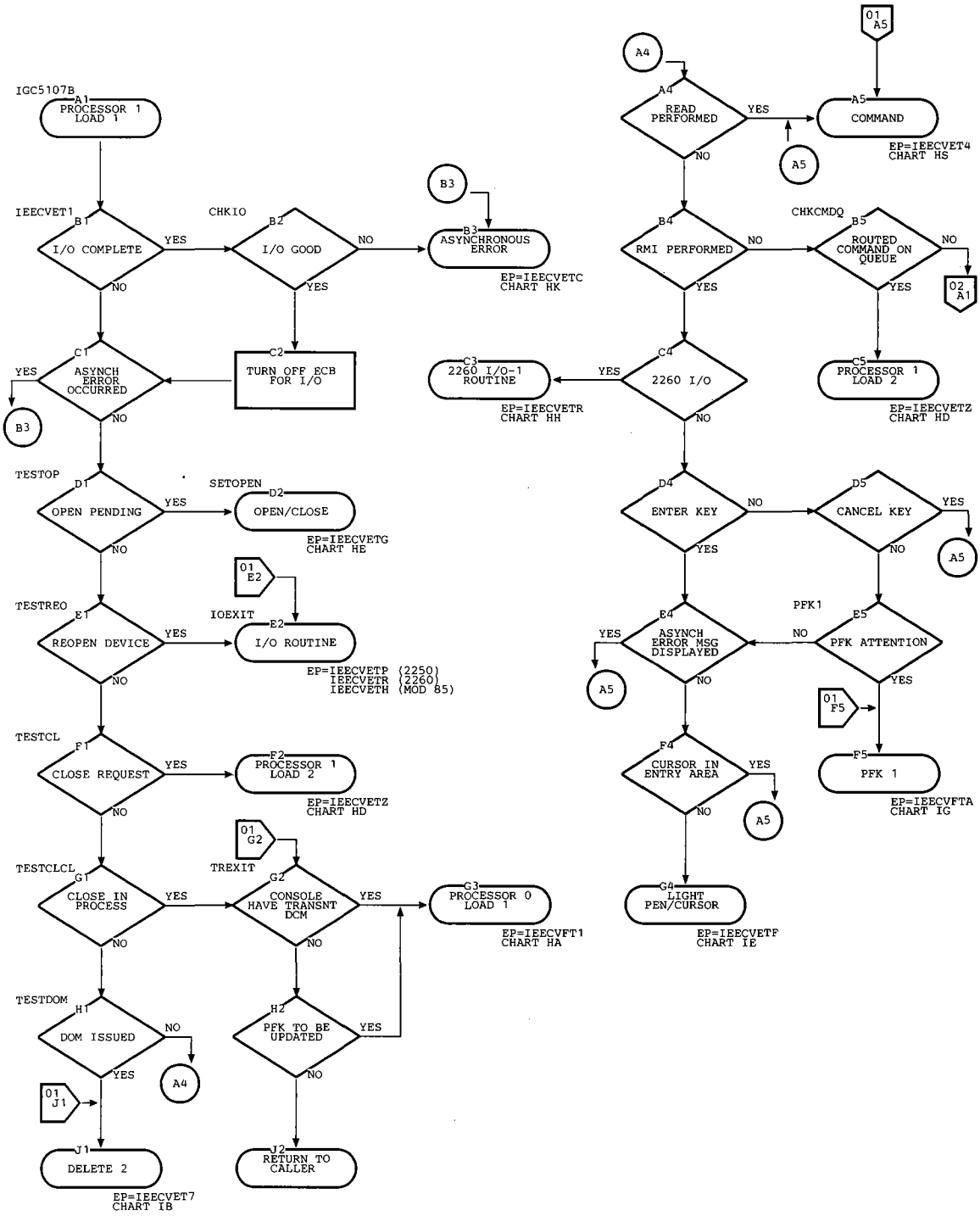


Chart HC. DIDOCS Processor 1, Load 1 (Page 2 of 2)

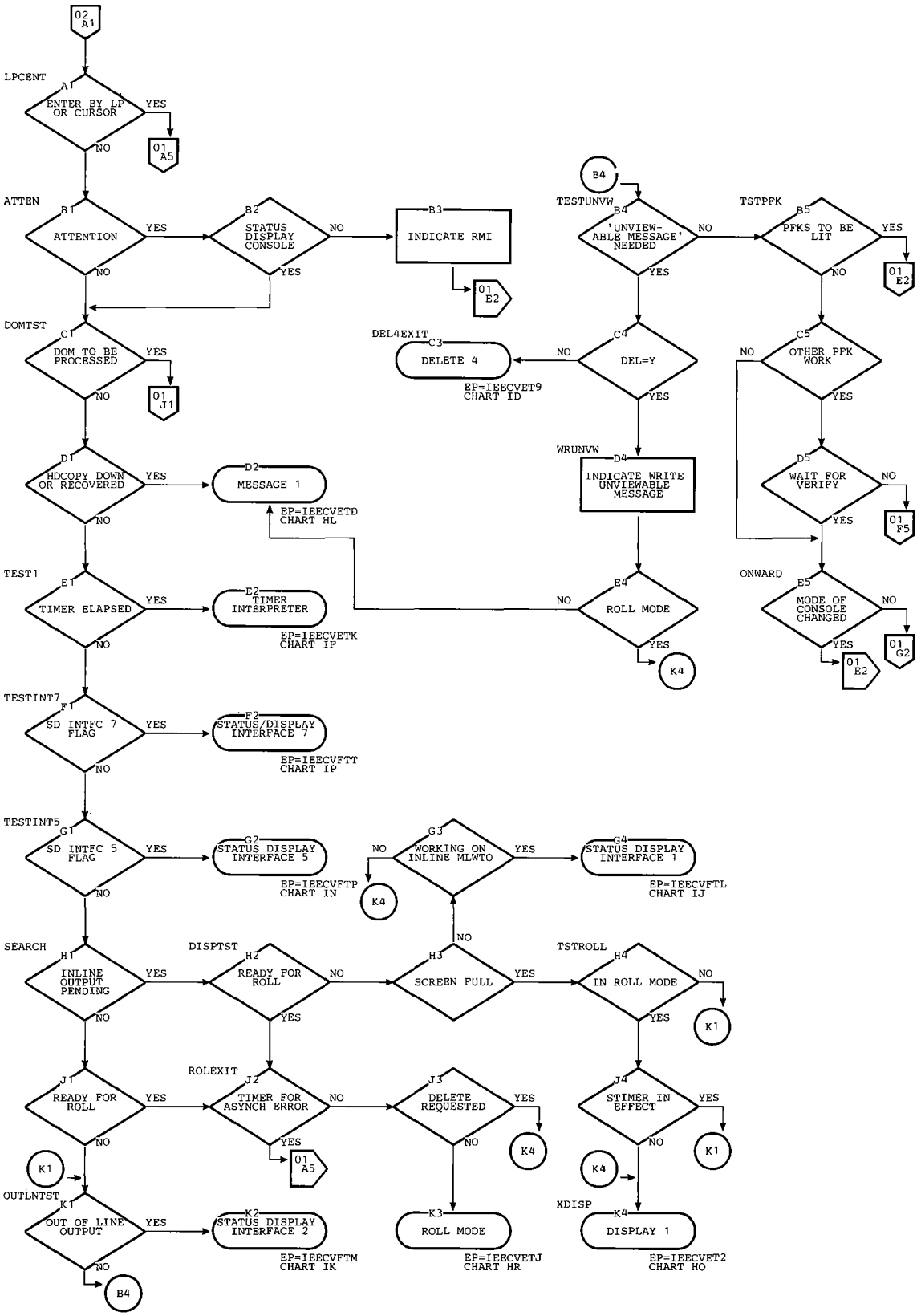


Chart HD. DIDOCS Processor 1, Load 2

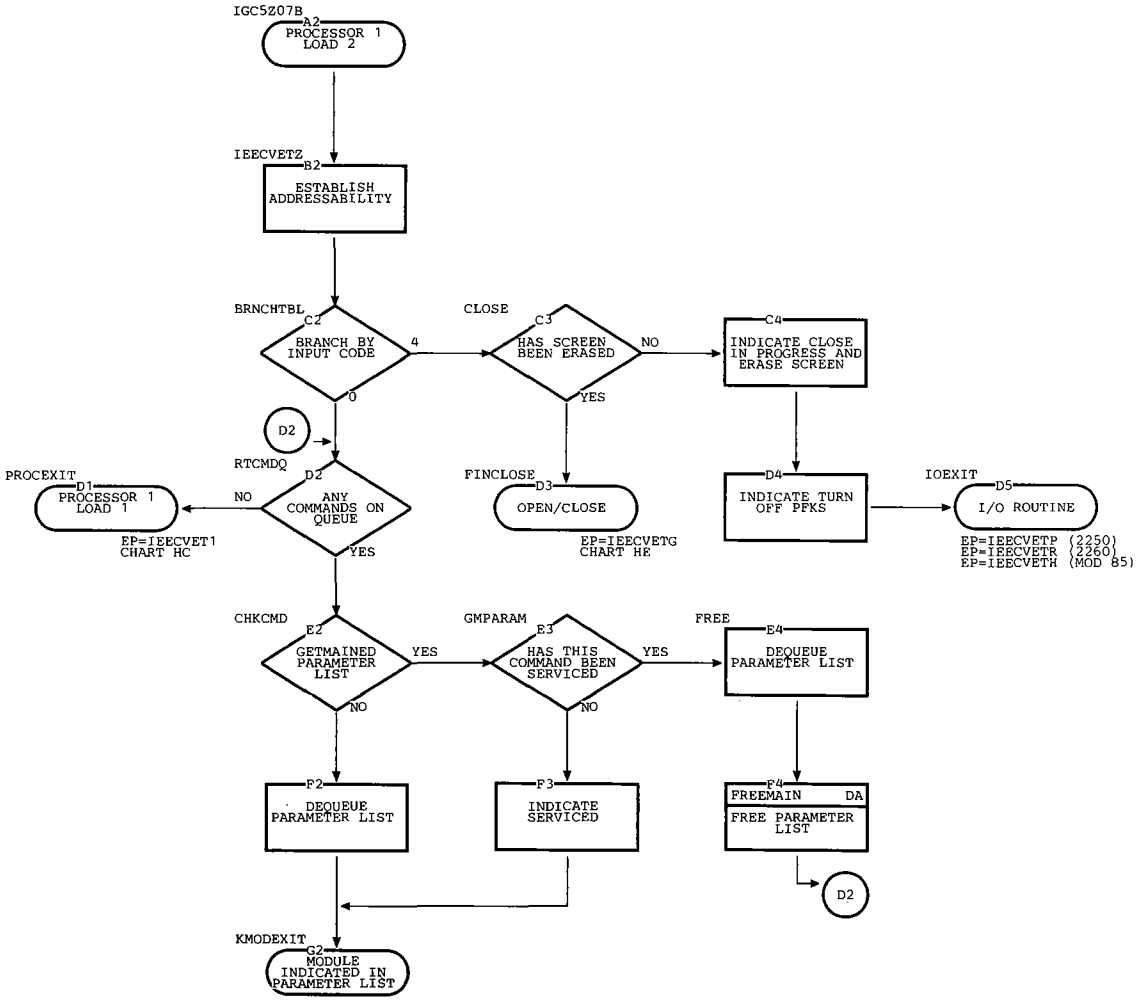


Chart HE. DIDOCS Open/Close Routine

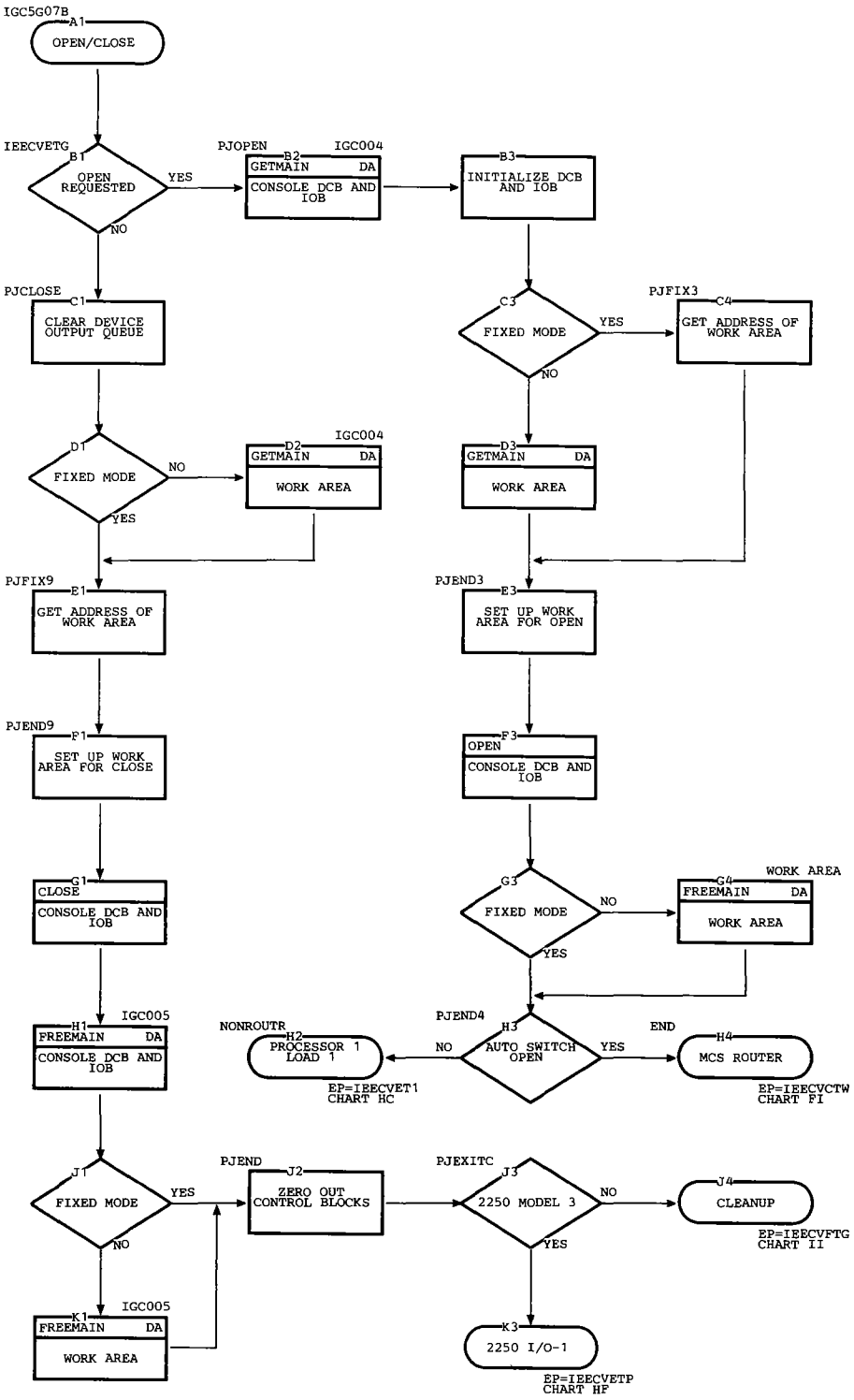


Chart HF. DIDOCS 2250 I/O-1 Routine (Page 1 of 2)

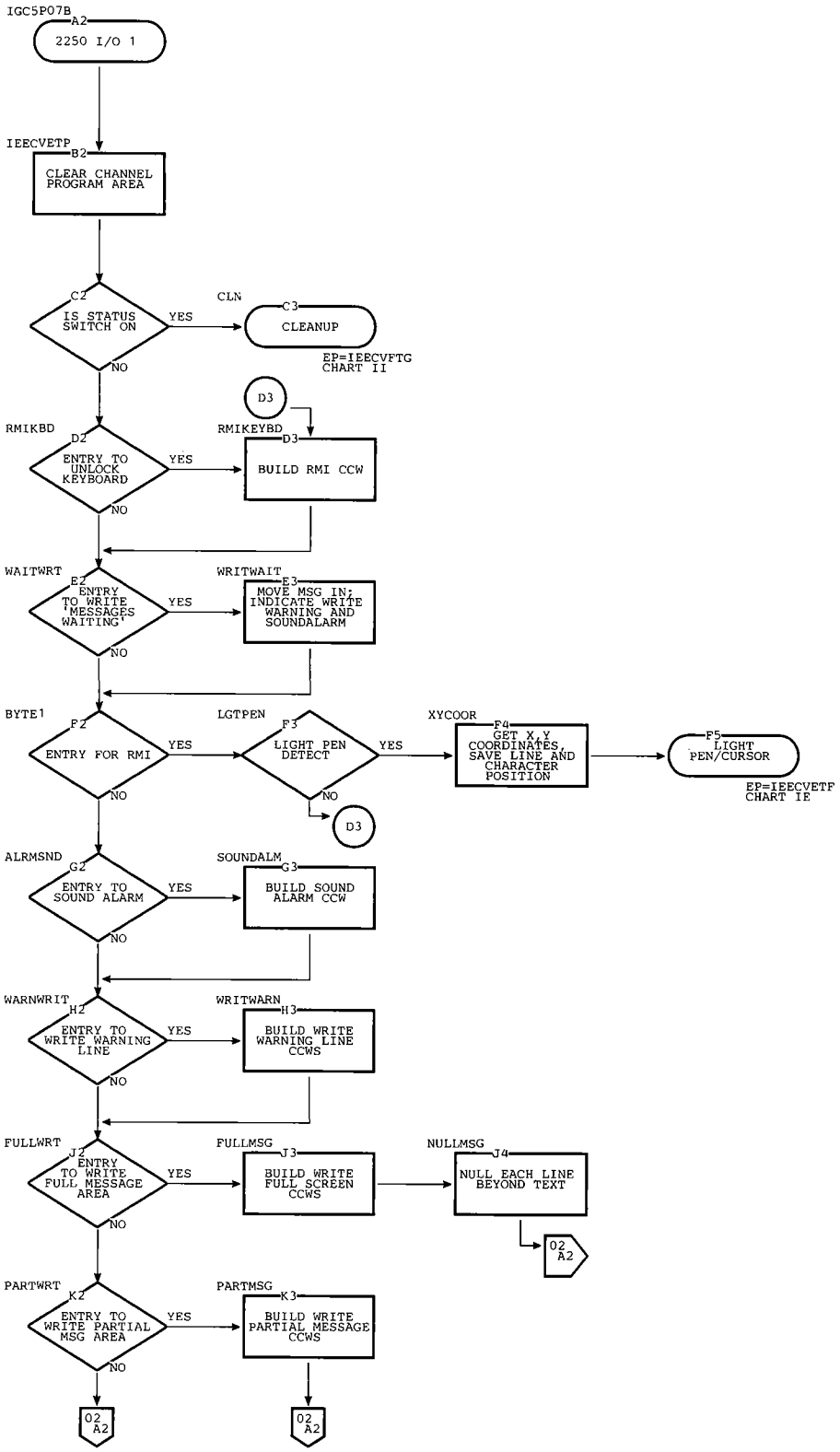


Chart HF. DIDOCS 2250 I/O-1 Routine (Page 2 of 2)

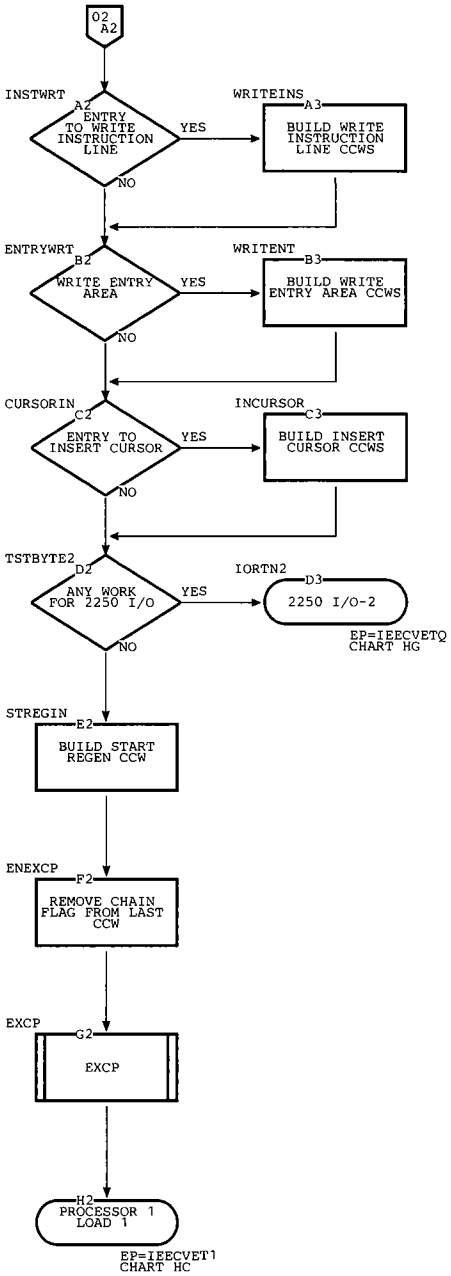


Chart HG. DIDOCS 2250 I/O-2 Routine

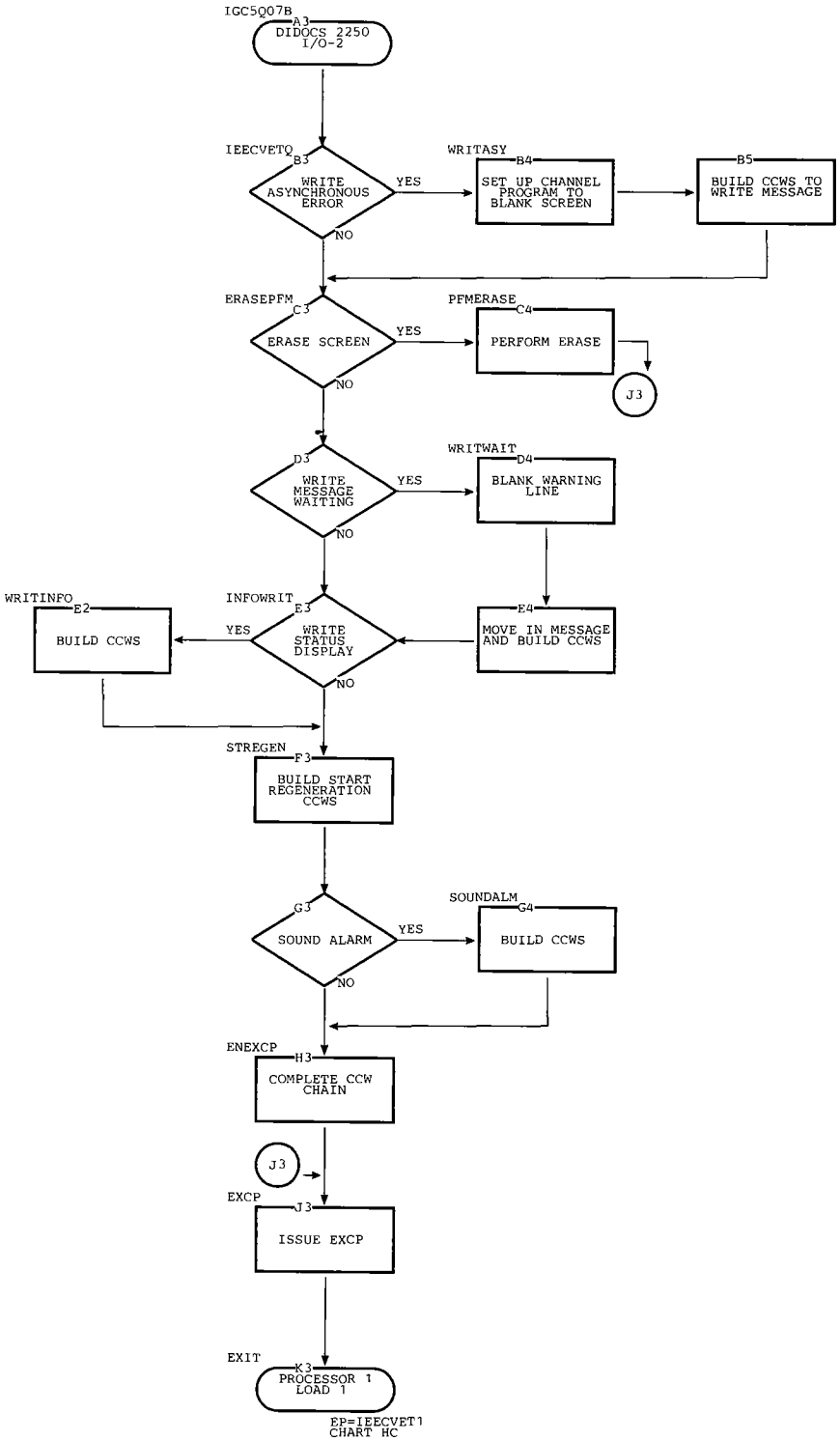


Chart HH. DIDOCS 2260 I/O-1 Routine

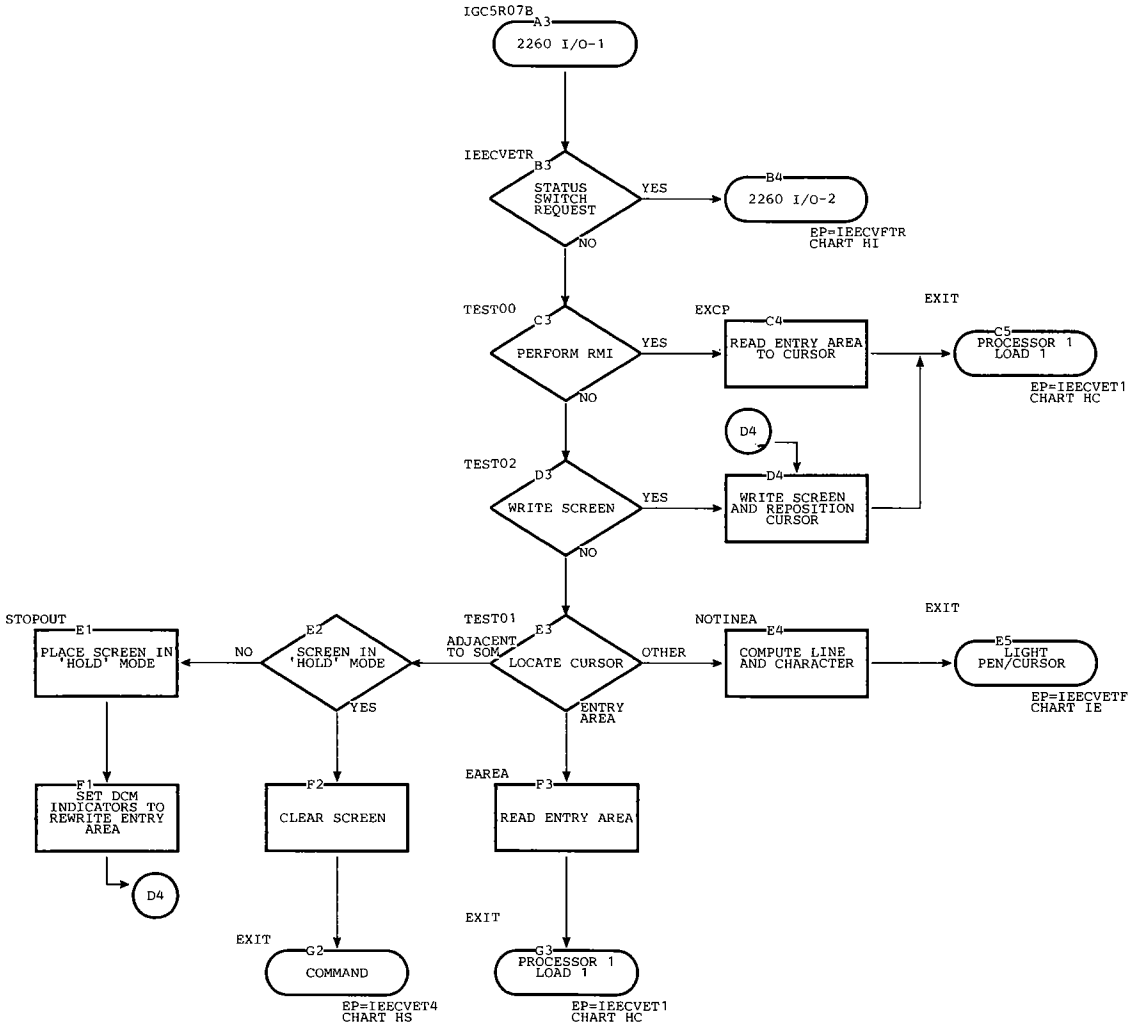


Chart HI. DIDOCS 2260 I/O-2 Routine

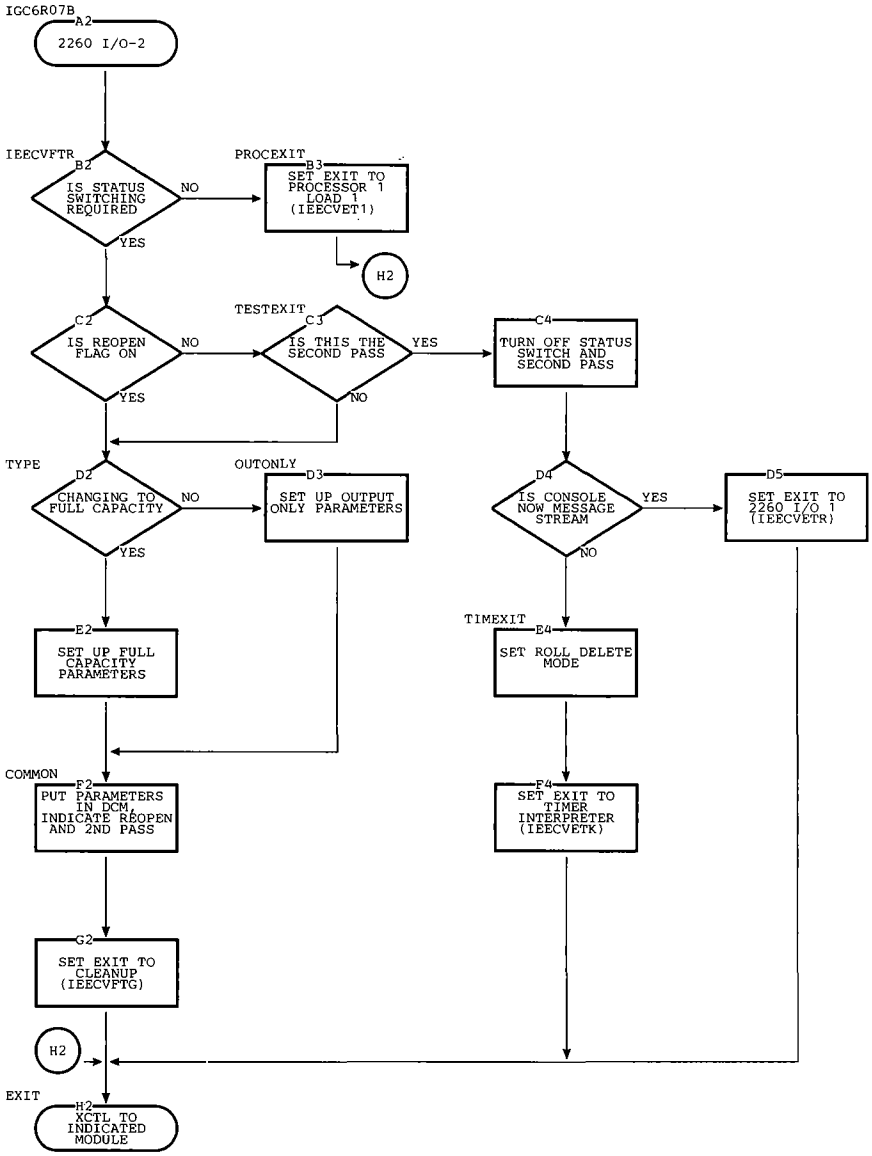


Chart HJ. DIDOCS Model 85 I/O Routine

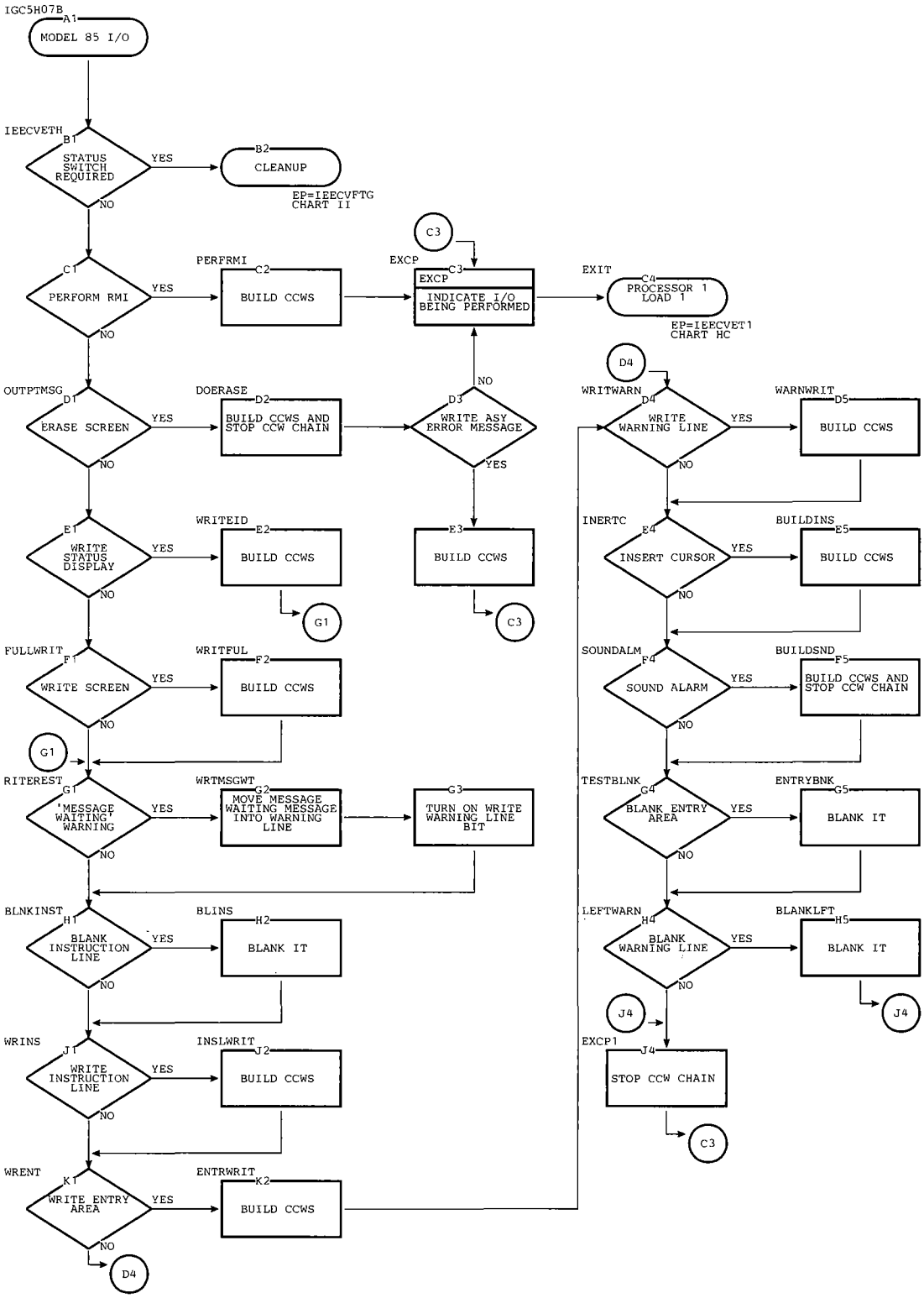


Chart HK. DIDOCS Asynchronous Error Routine (Page 1 of 3)

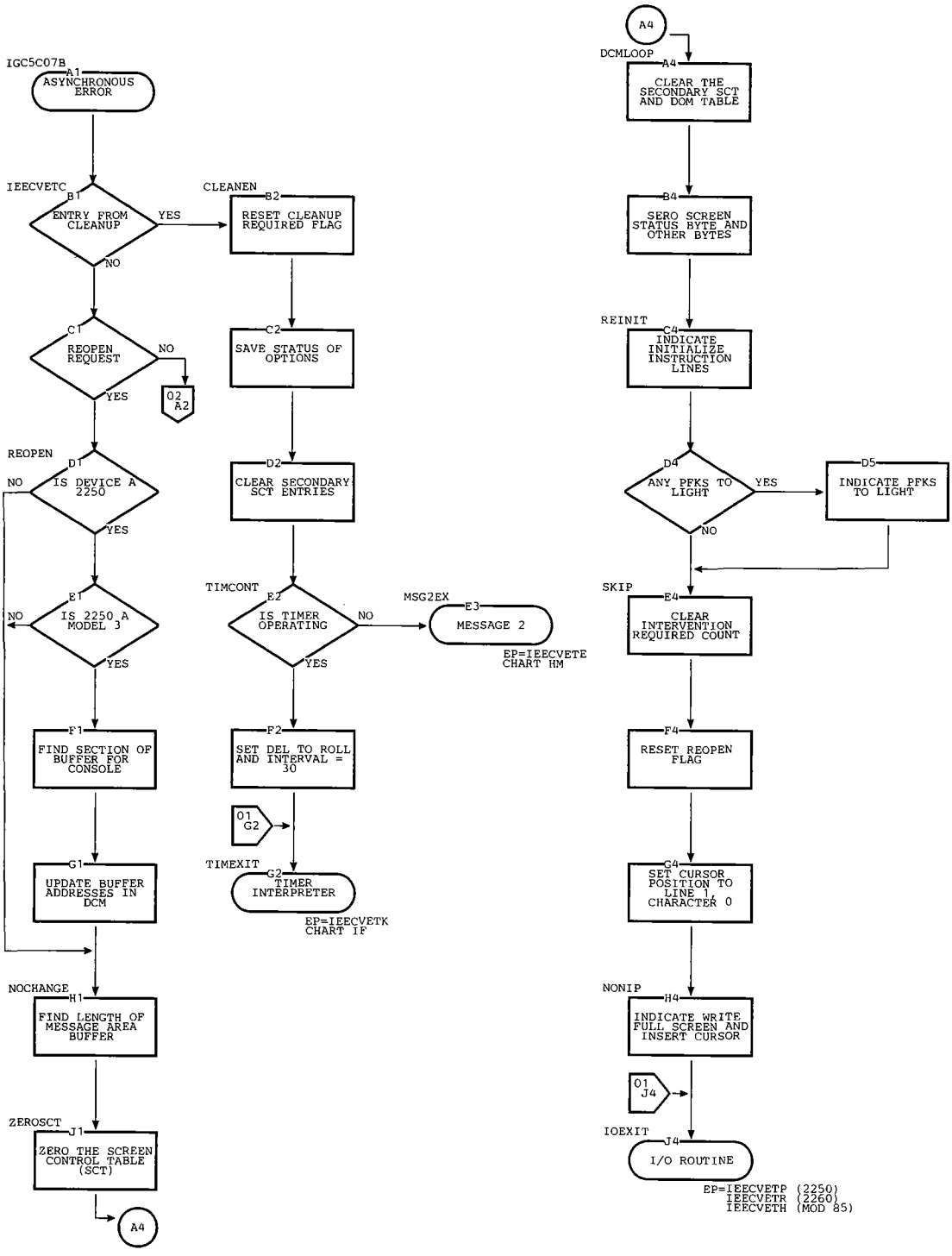


Chart HK. DIDOCS Asynchronous Error Routine (Page 2 of 3)

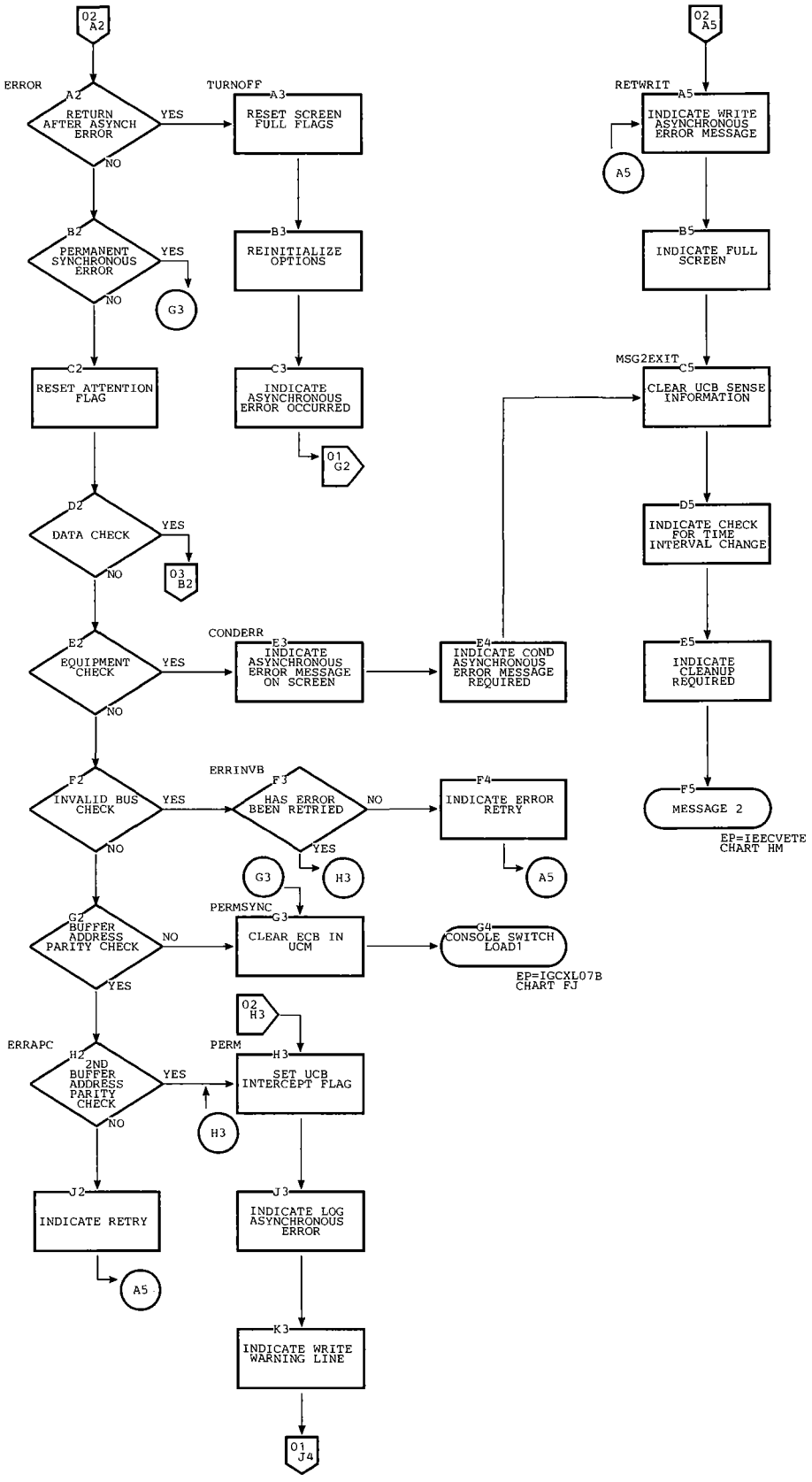


Chart HK. DIDOCS Asynchronous Error Routine (Page 3 of 3)

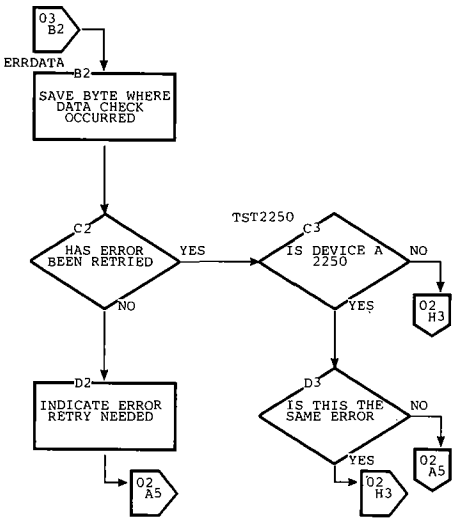


Chart HL. DIDOCS Message 1 Routine

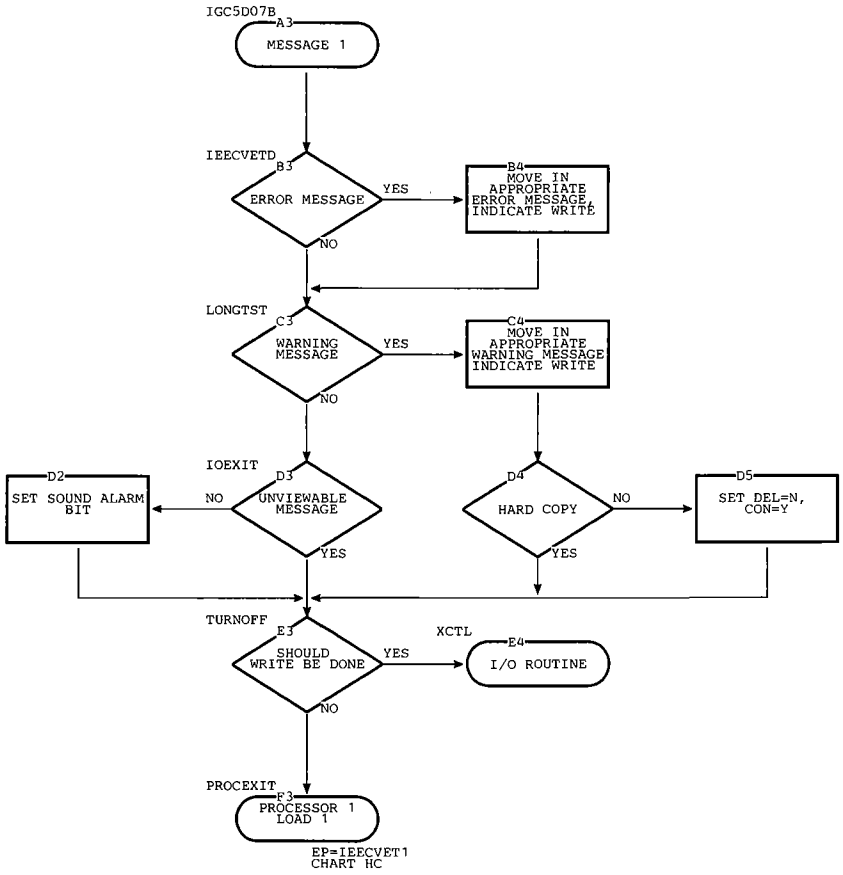


Chart HM. DIDOCS Message 2 Routine (Page 1 of 2)

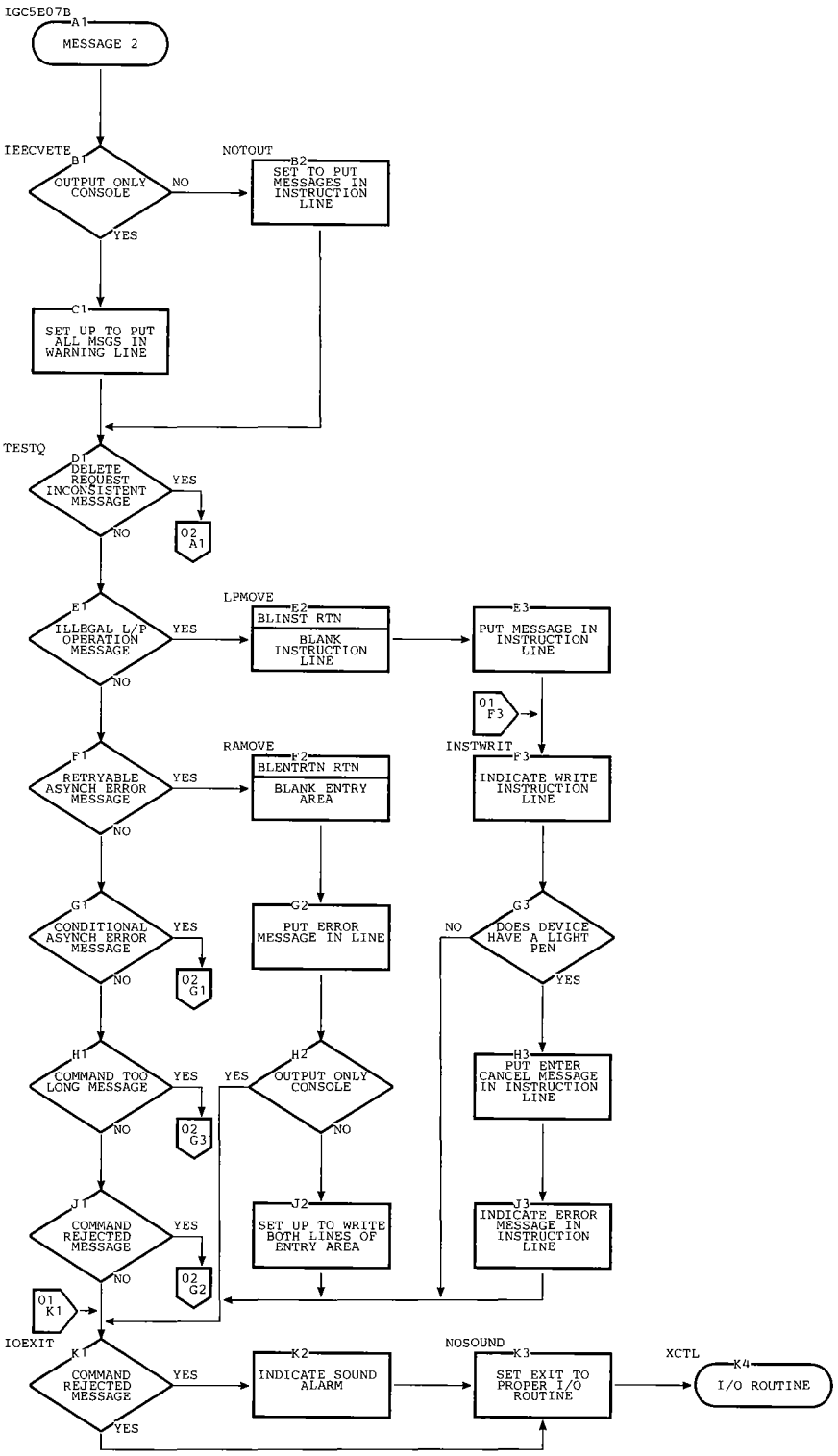


Chart HM. DIDOCS Message 2 Routine (Page 2 of 2)

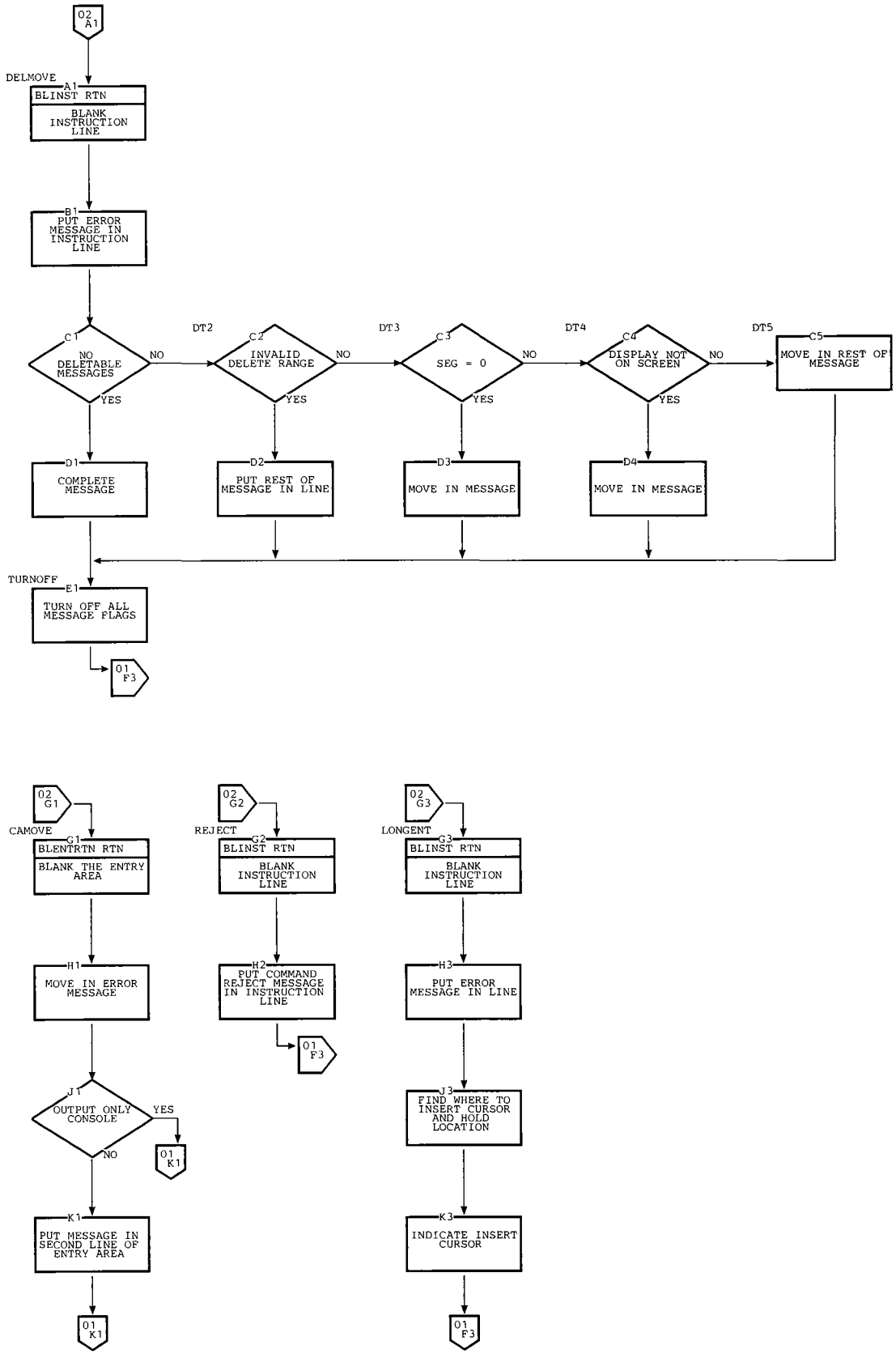


Chart HN. DIDOCS Message 3 Routine

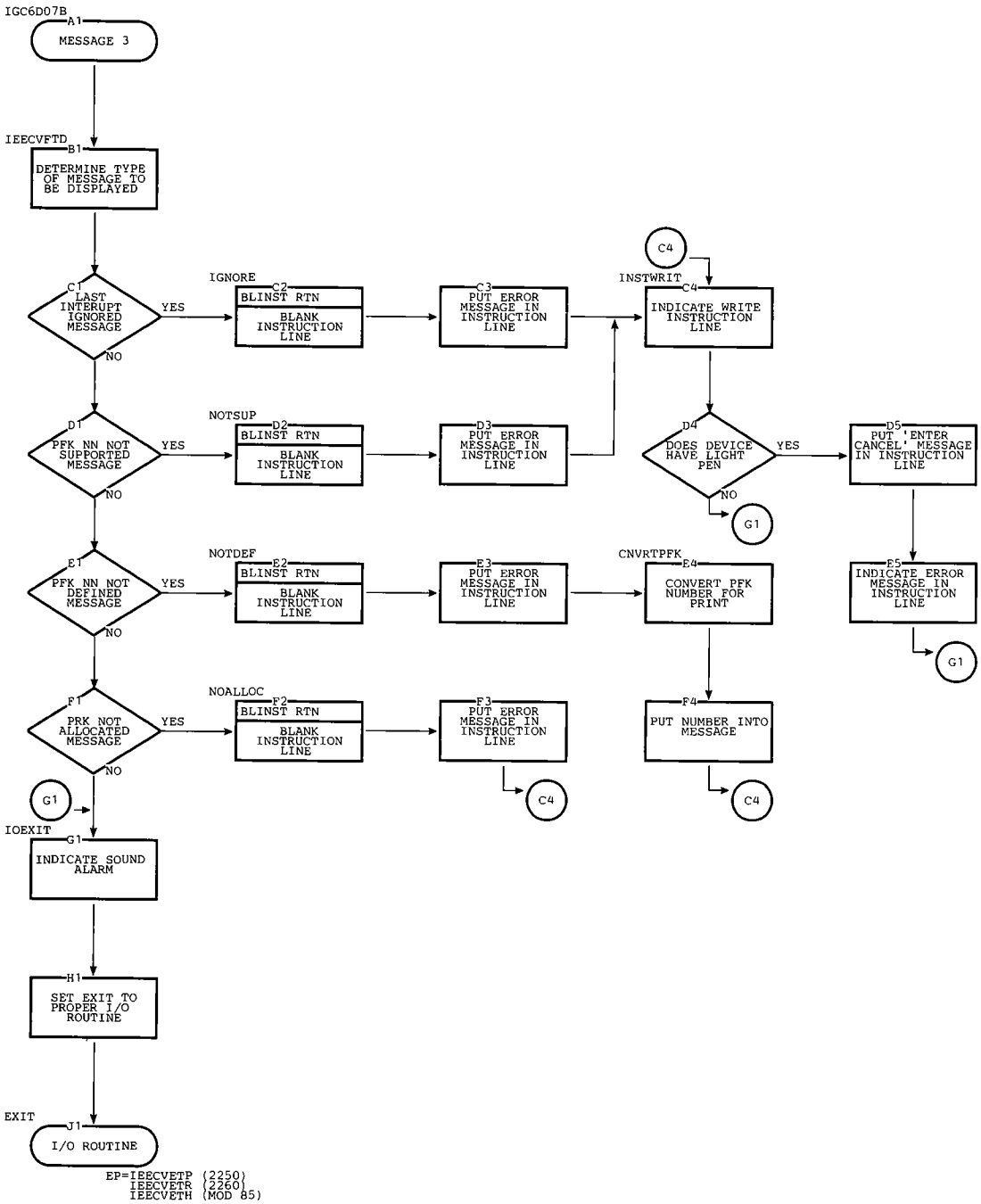


Chart HO. DIDOCS Display 1 Routine (Page 1 of 2)

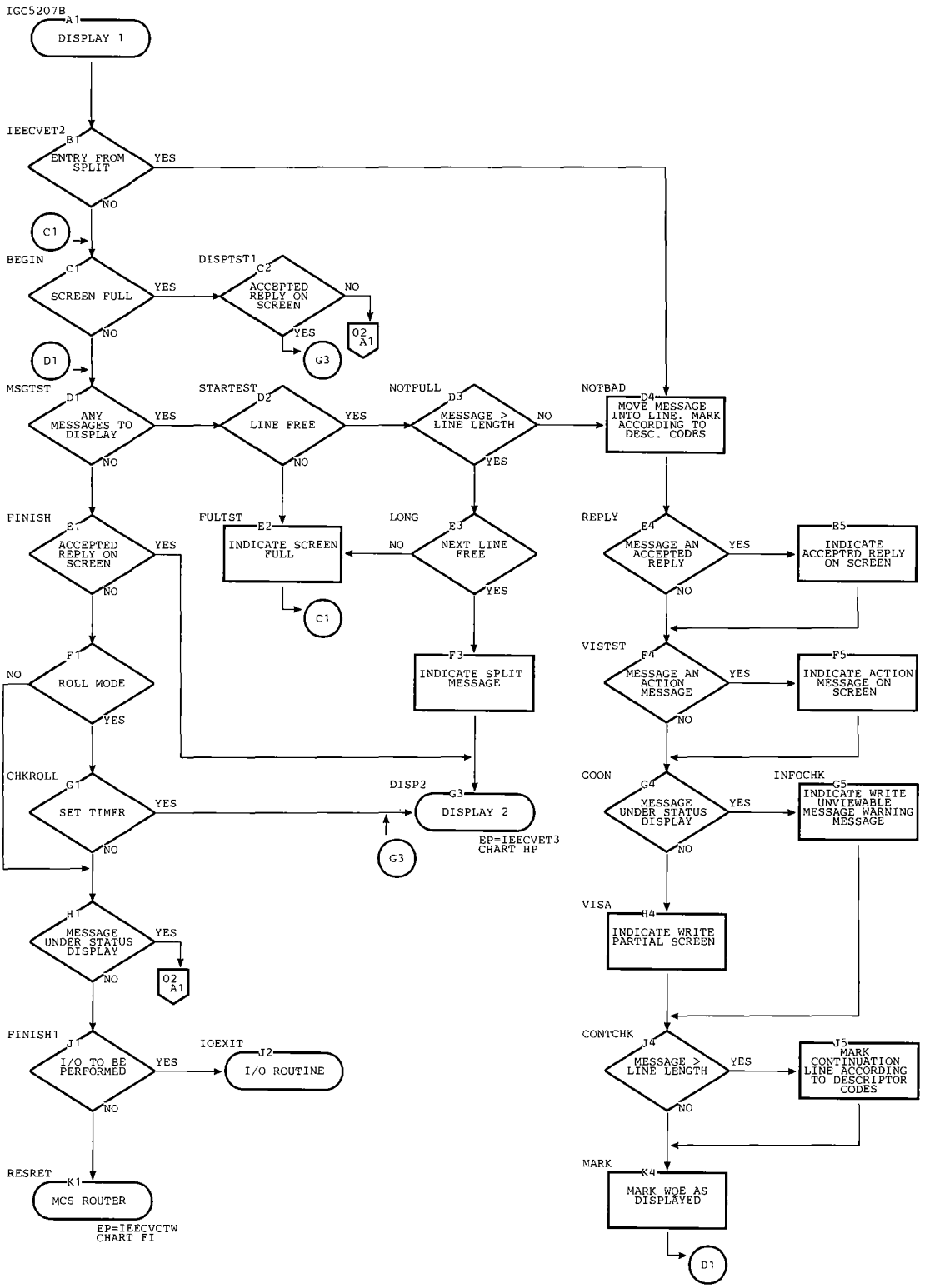


Chart HO. DIDOCS Display 1 Routine (Page 2 of 2)

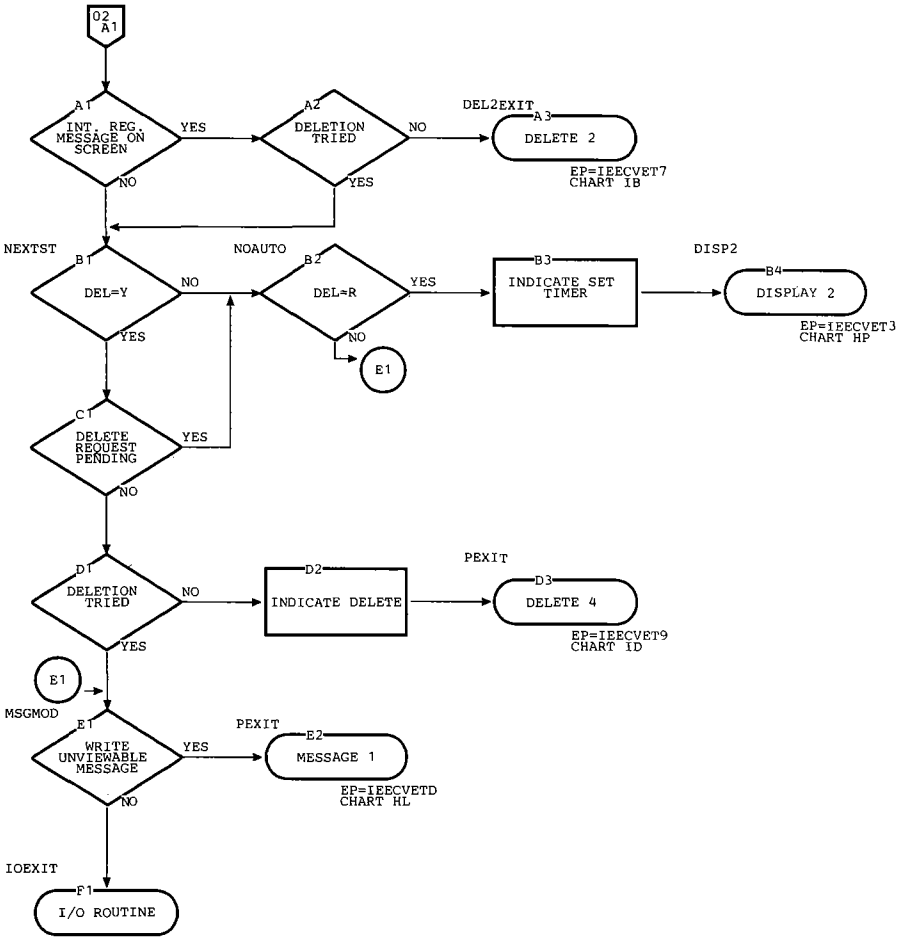


Chart HP. DIDOCS Display 2 Routine (Page 1 of 2)

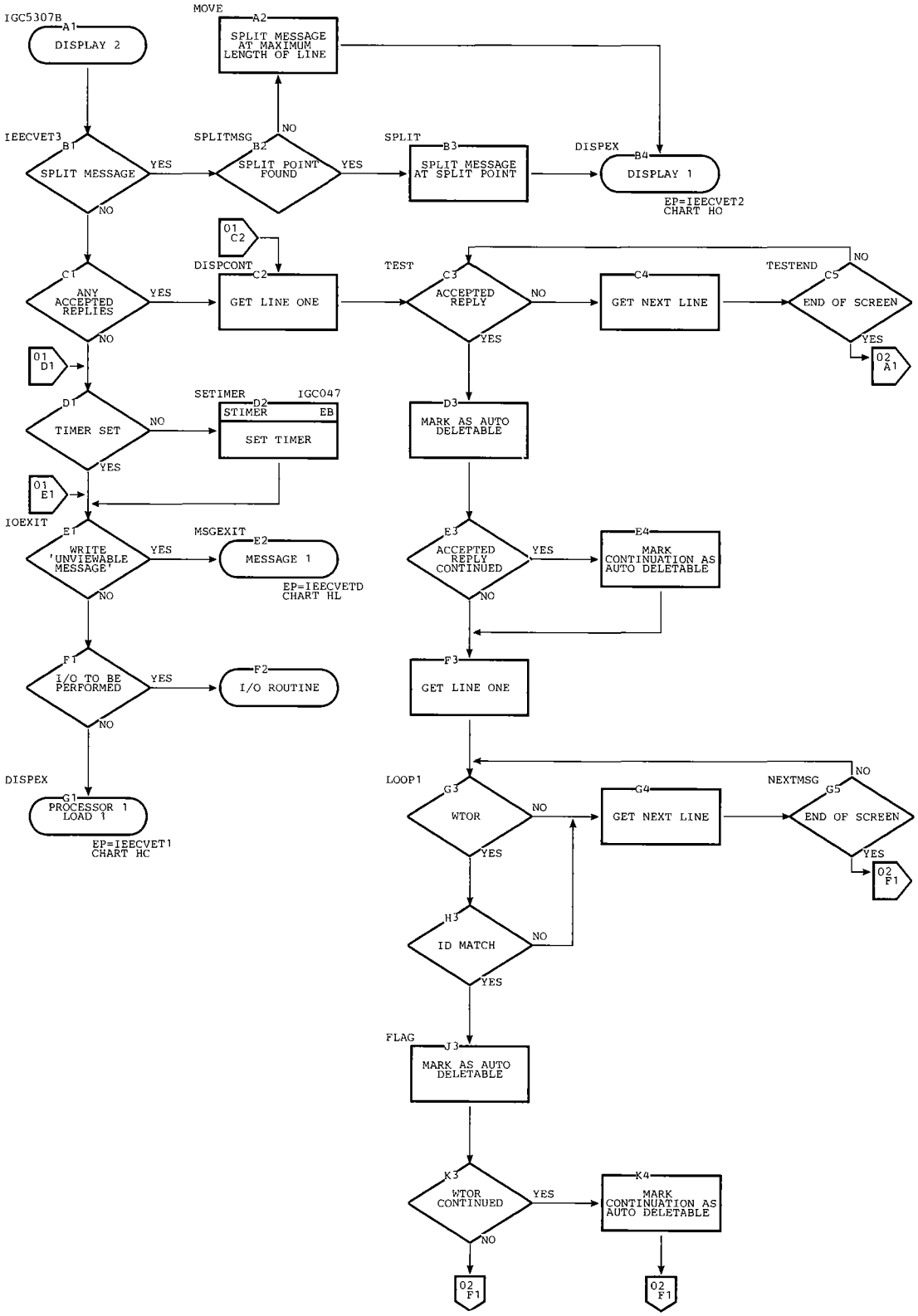
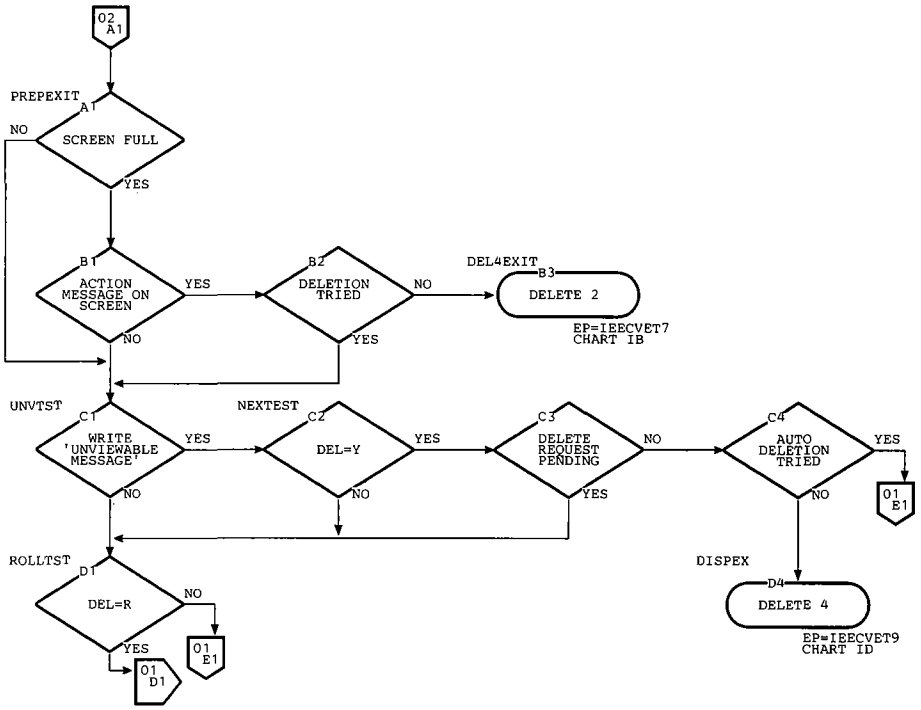


Chart HP. DIDOCS Display 2 Routine (Page 2 of 2)



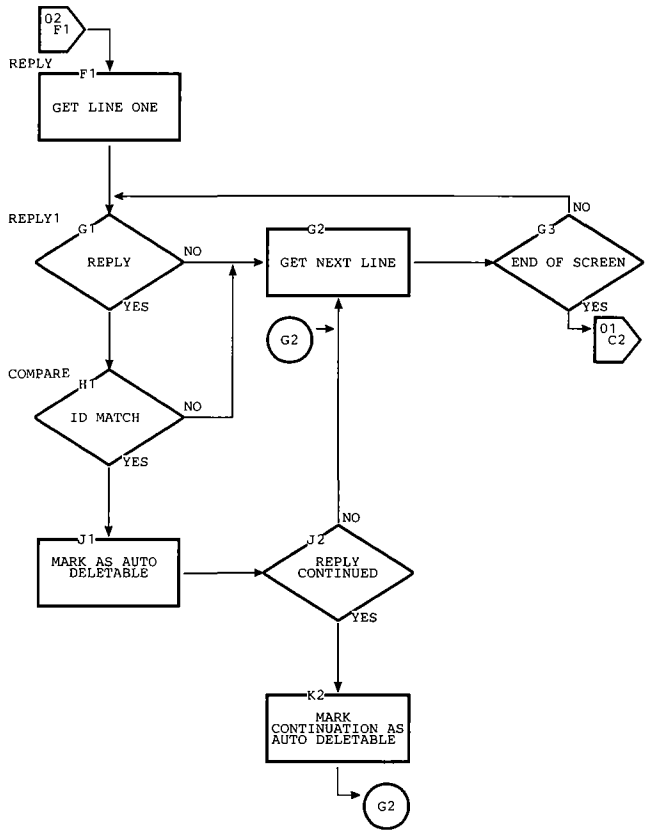


CHART HQ. DIDOCS Display 3 Routine (Page 1 of 2)

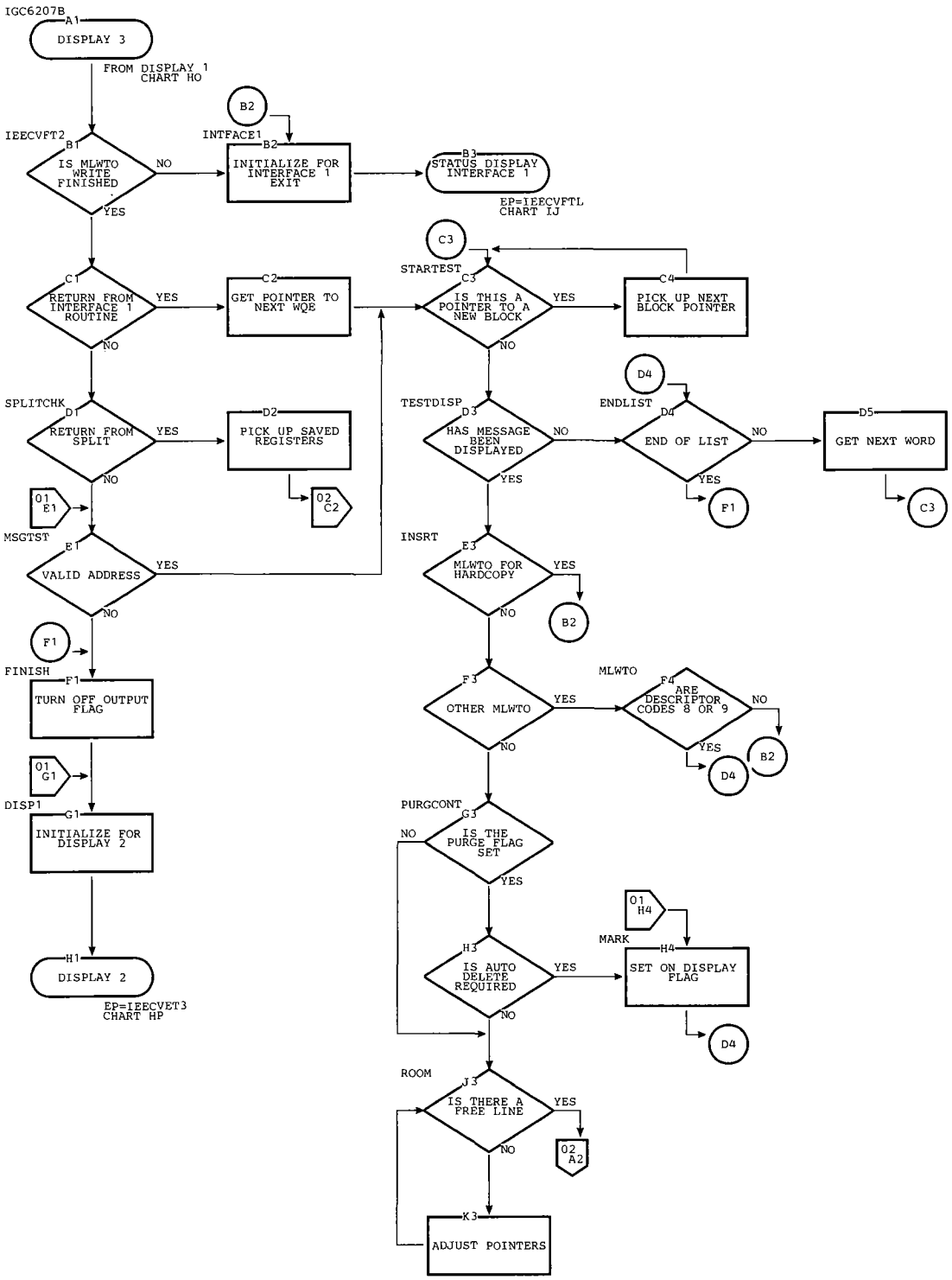


Chart HQ. DIDOCS Display 3 Routine (Page 2 of 2)

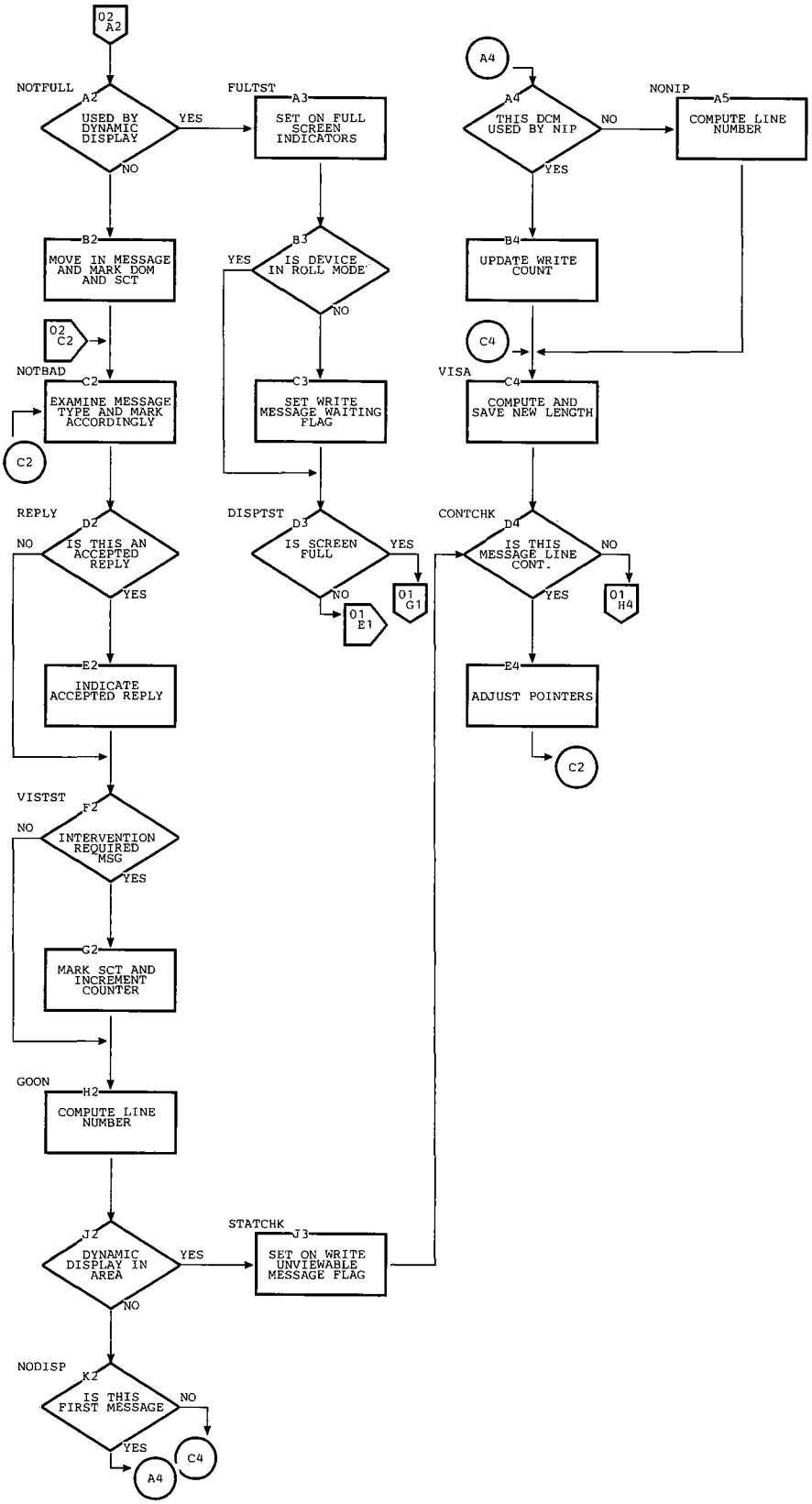


Chart HR. DIDOCS Roll Mode Routine

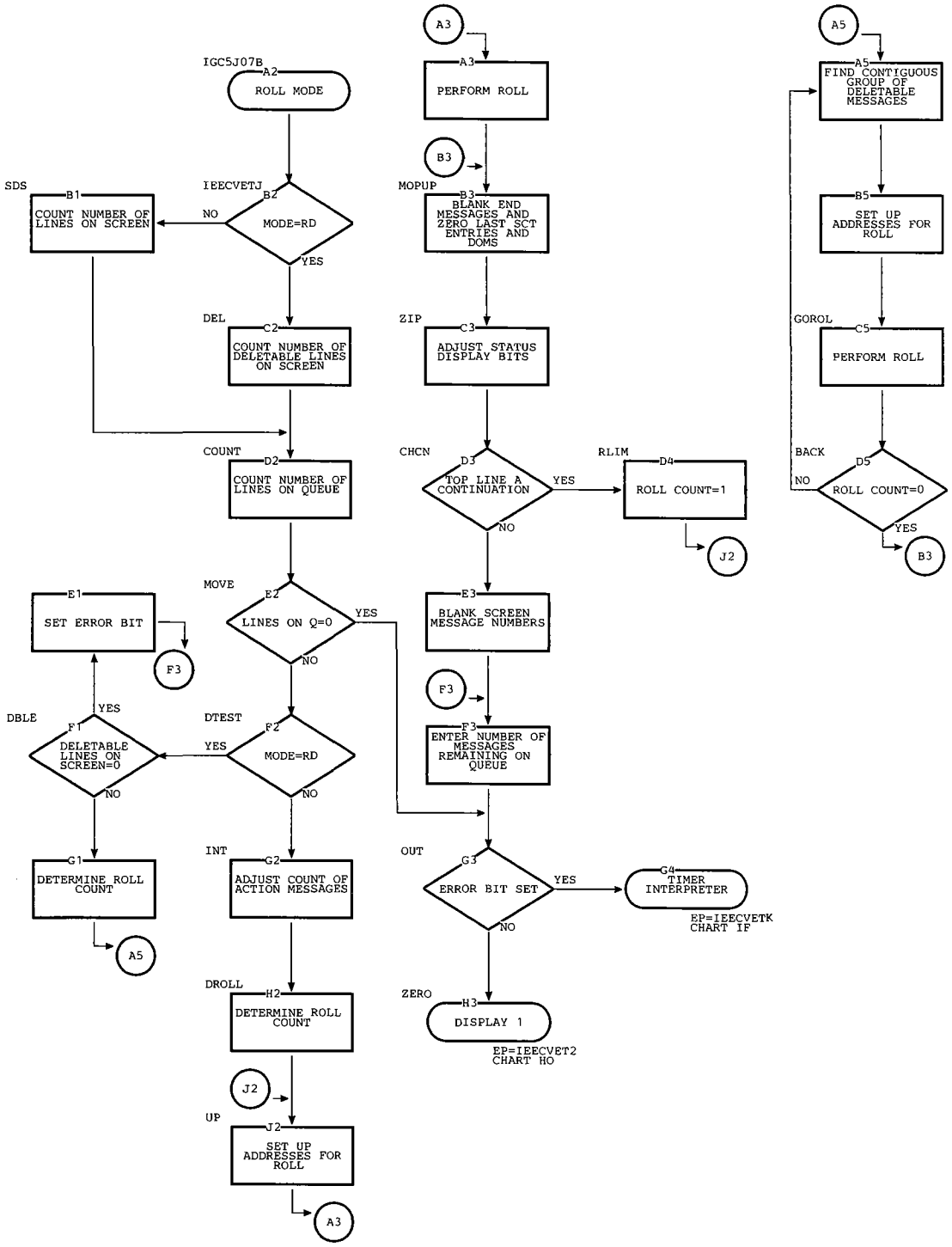


Chart HS. DIDOCS Command Routine

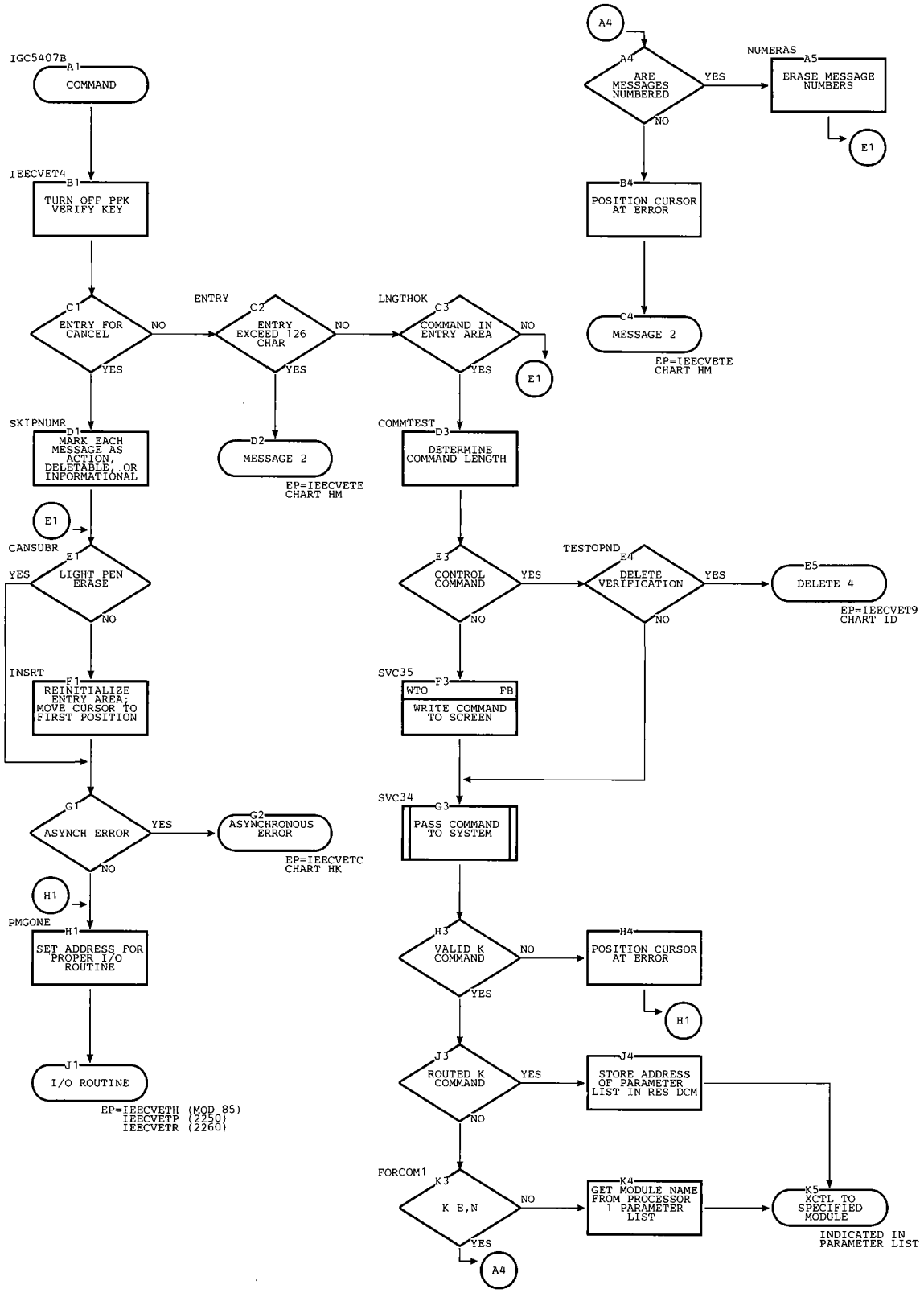


Chart HT. DIDOCS Options Routine (Page 1 of 2)

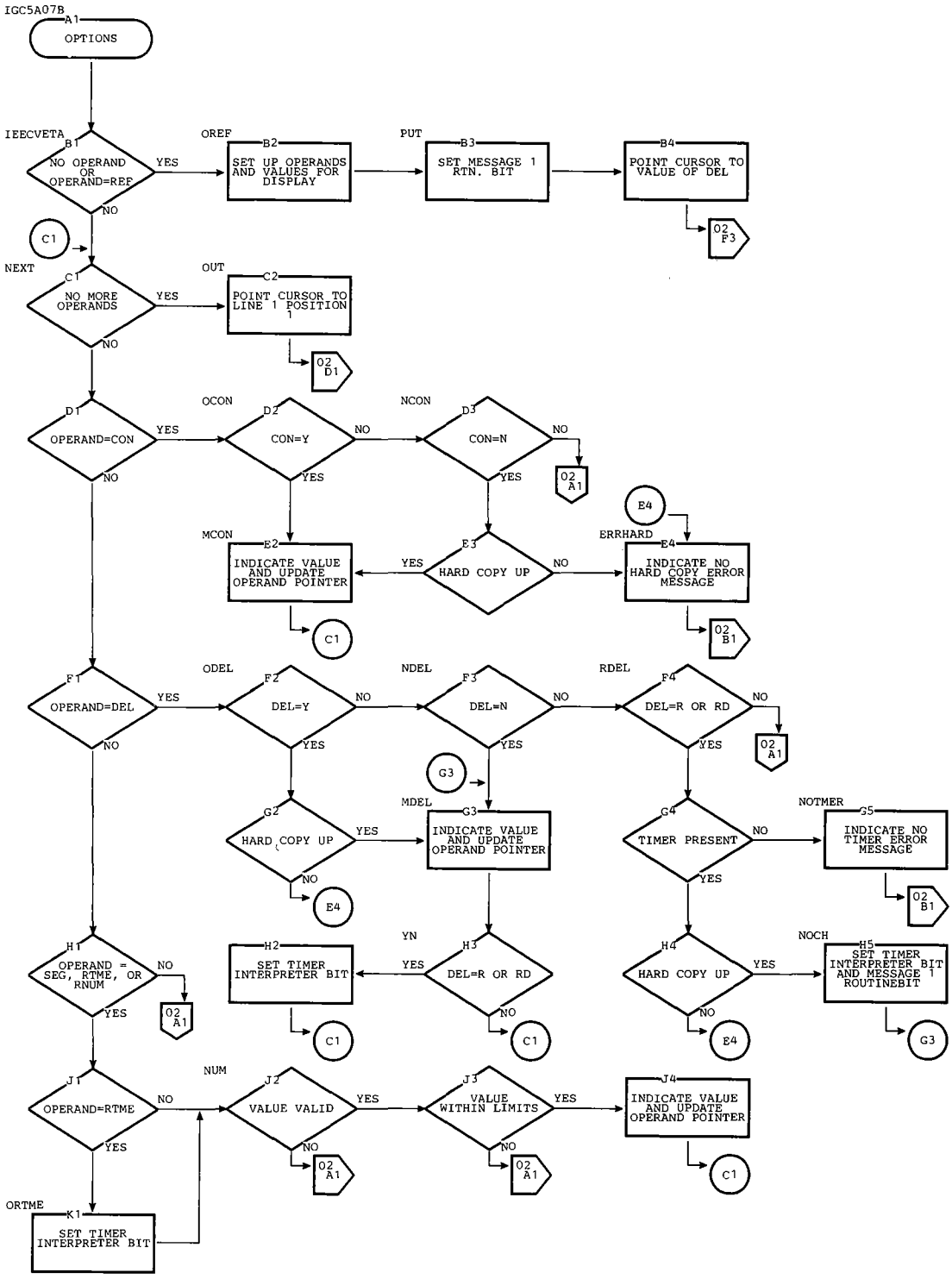


Chart HT. DIDOCS Options Routine (Page 2 of 2)

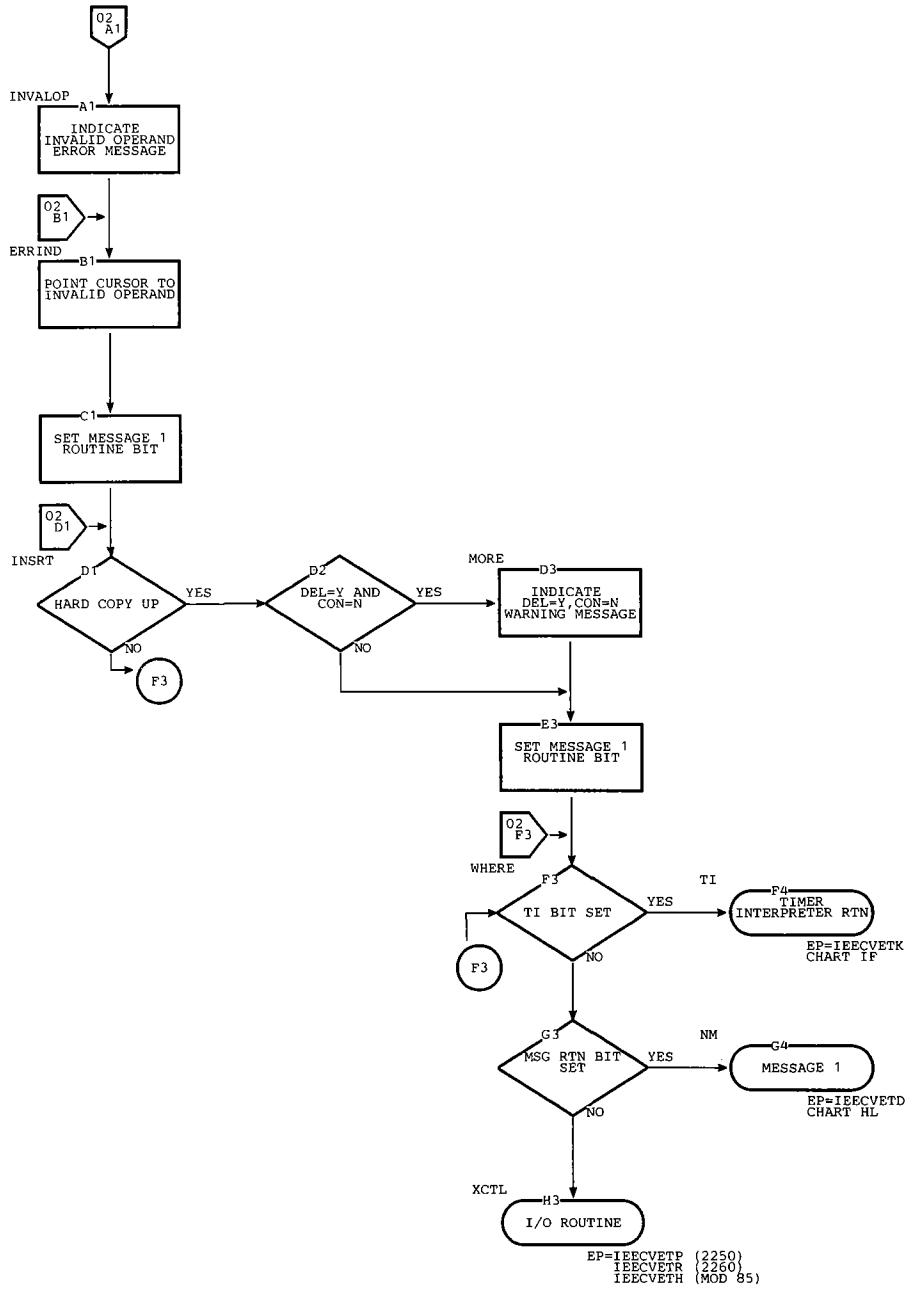


Chart IA. DIDOCS Delete 1 Routine (Page 1 of 2)

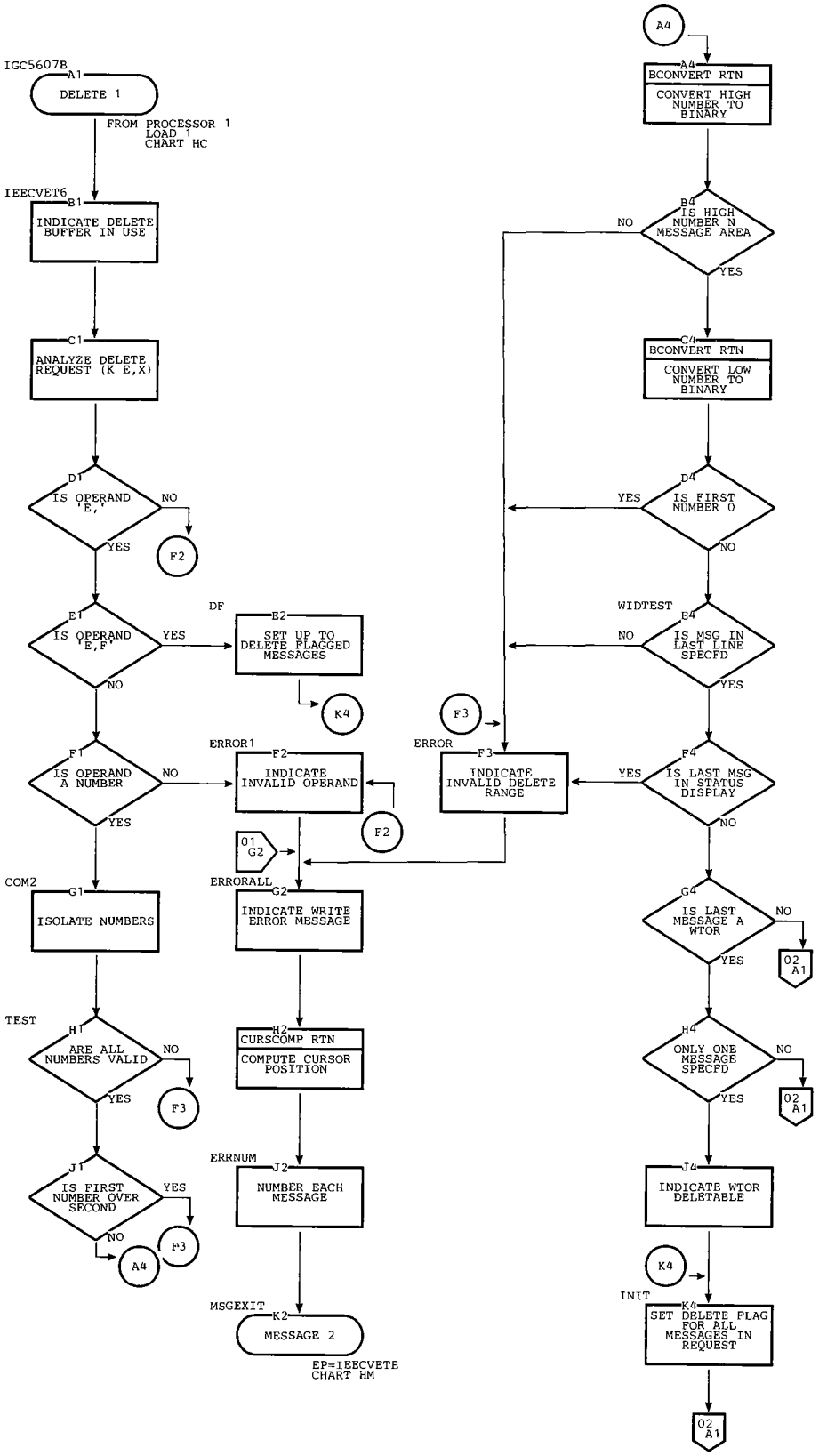


Chart IA. DIDOCS Delete 1 Routine (Page 2 of 2)

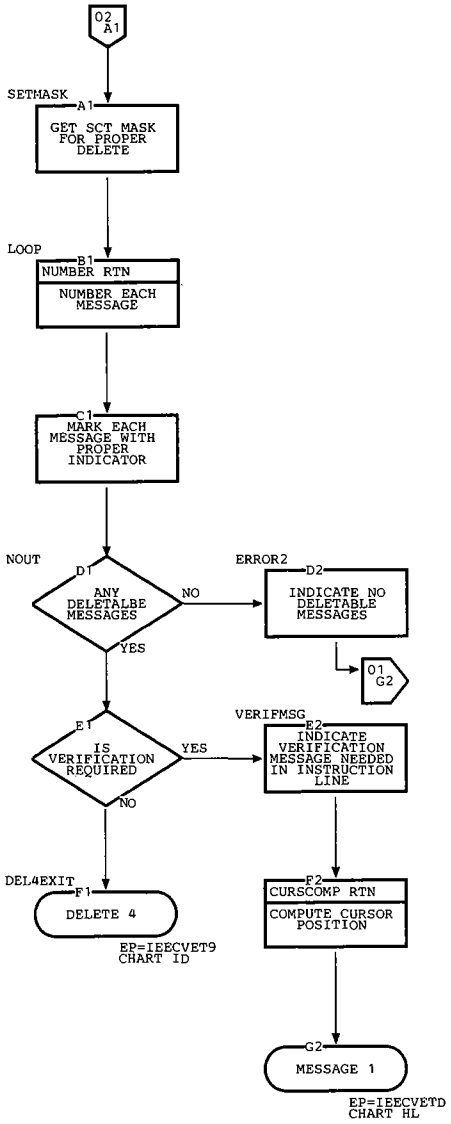


Chart IB. DIDOCS Delete 2 Routine

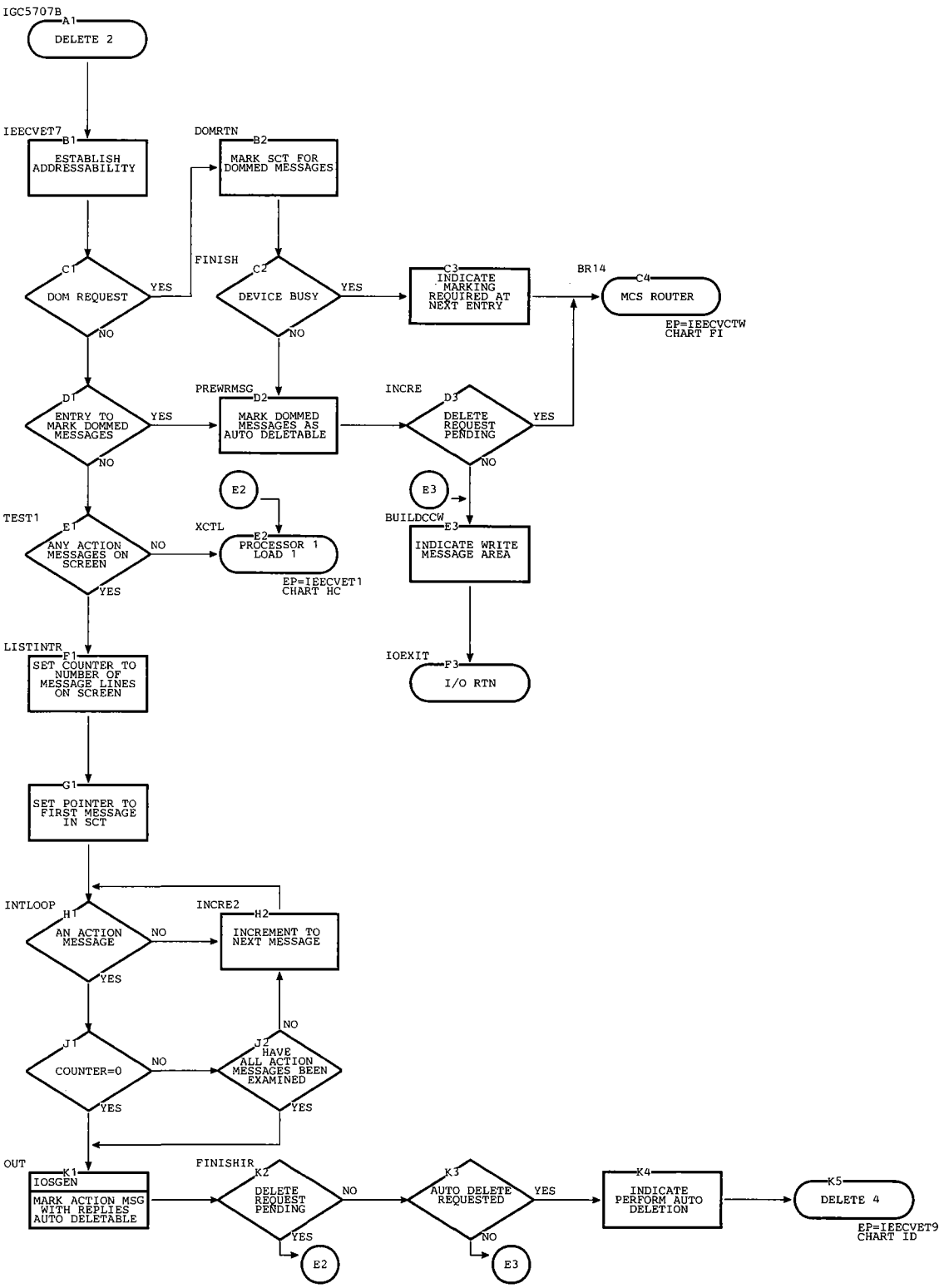


Chart IC. DIDOCS Delete 3 Routine (Page 1 of 2)

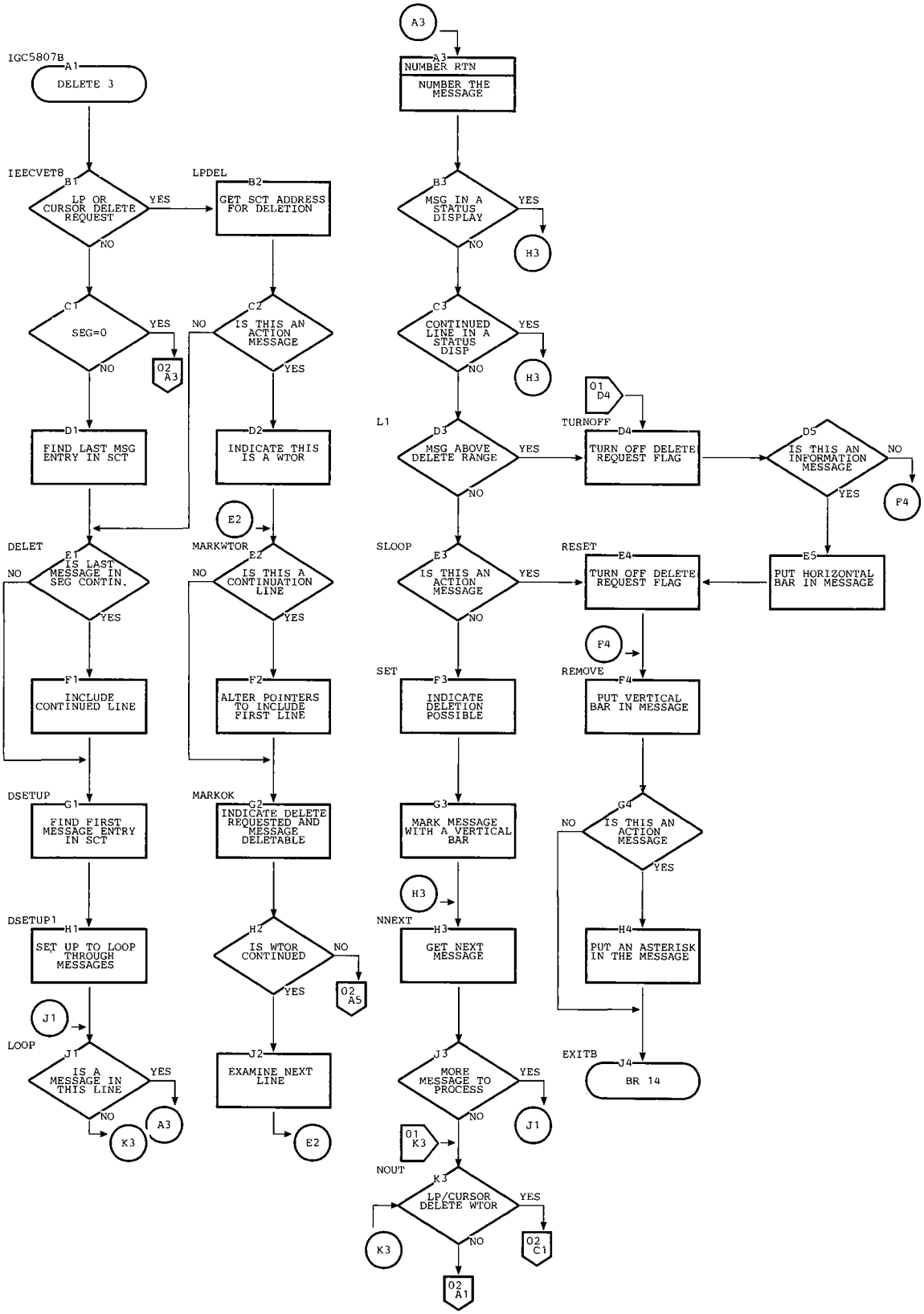


Chart IC. DIDOCS Delete 3 Routine (Page 2 of 2)

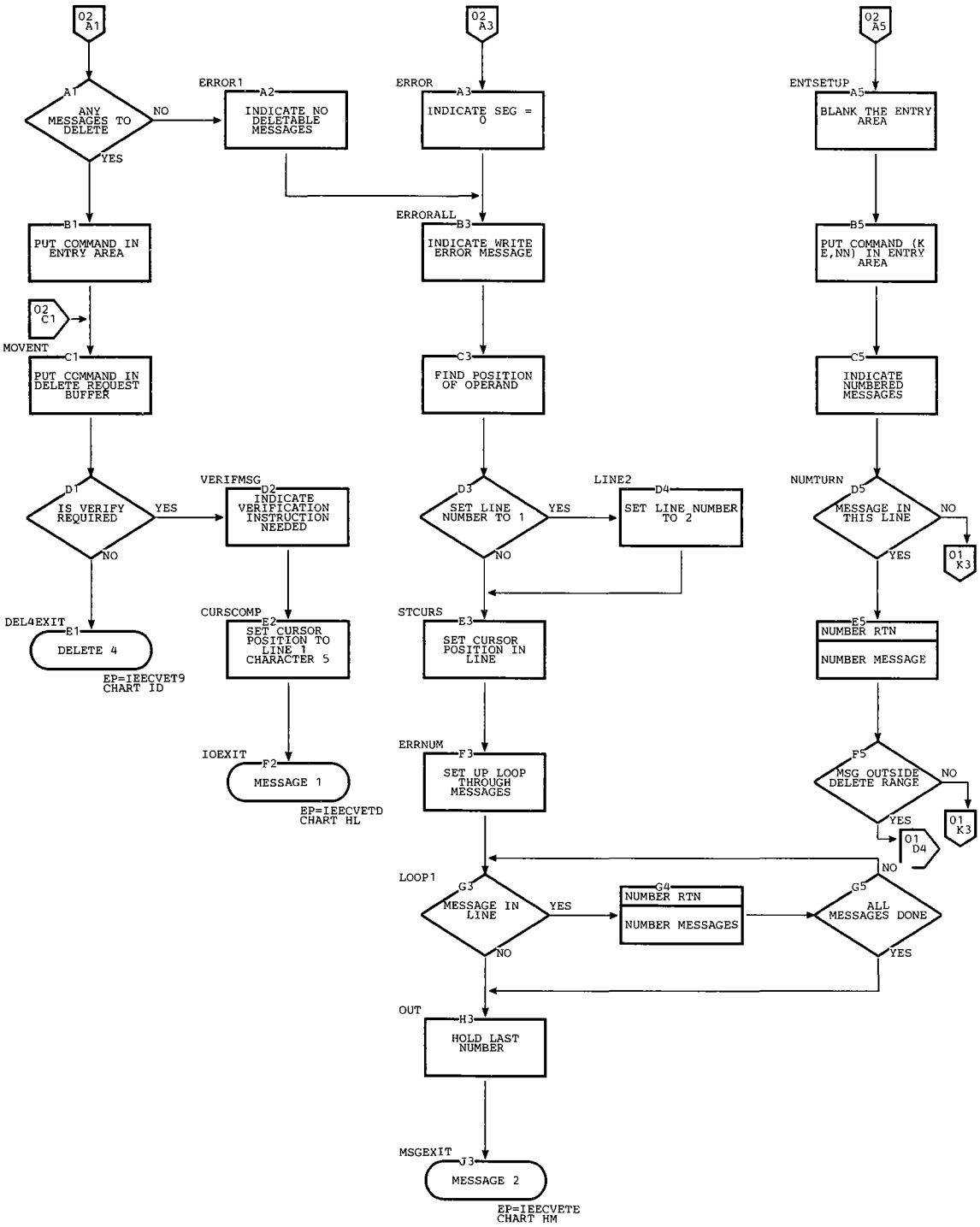


Chart ID. DIDOCS Delete 4 Routine

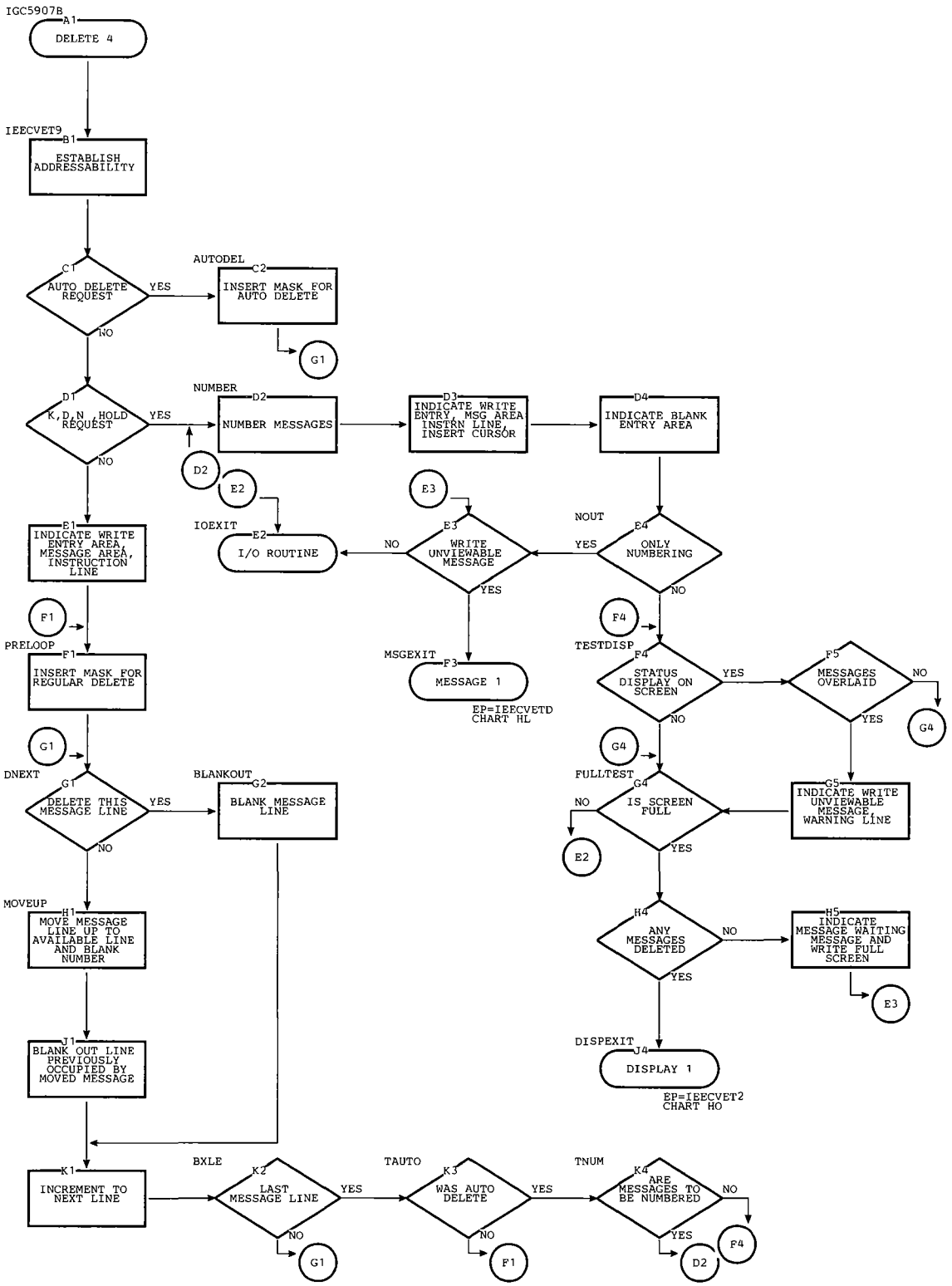


Chart IE. DIDOCS Light Pen/Cursor Routine

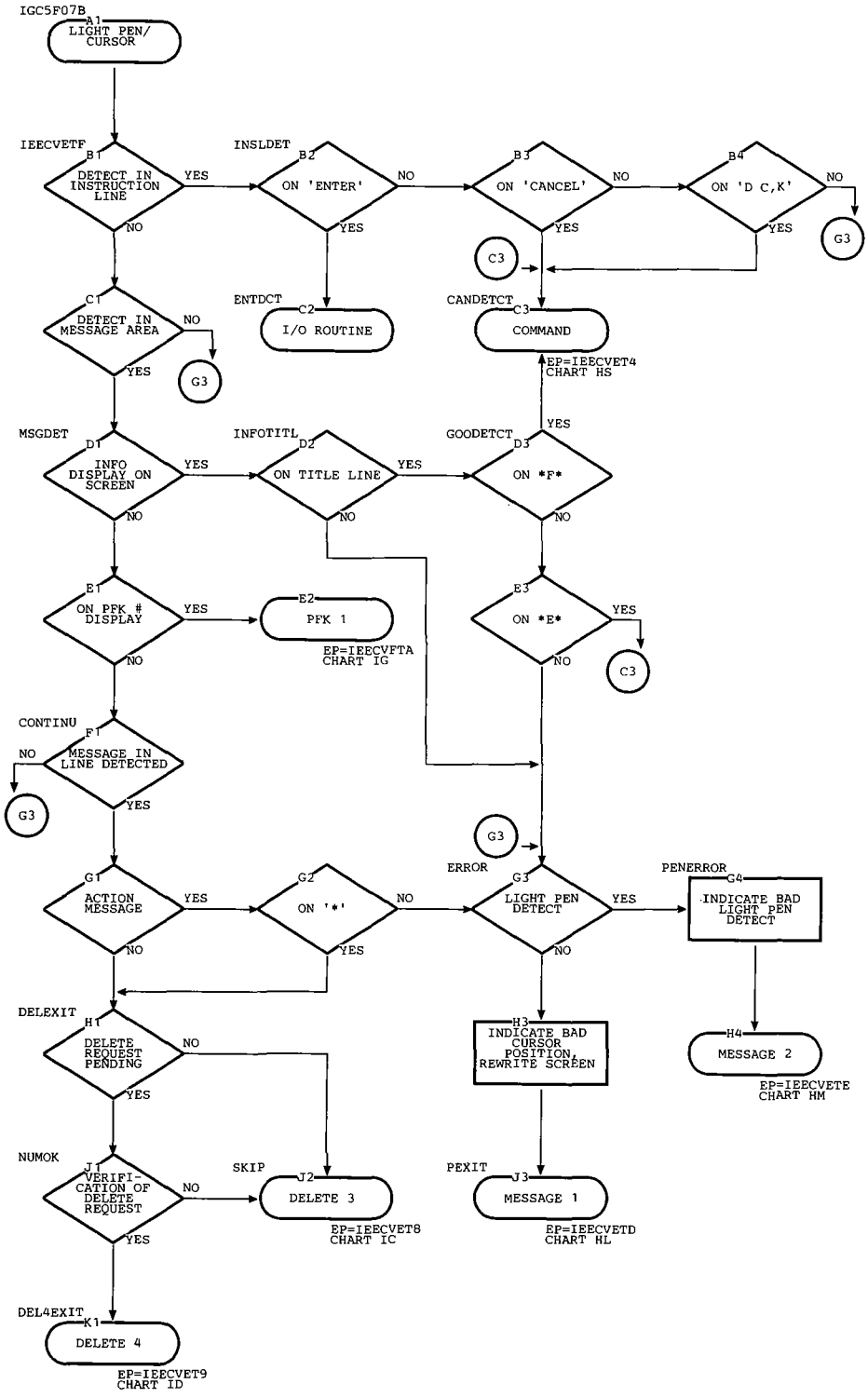


Chart IF. DIDOCS Timer Interpreter (Page 1 of 2)

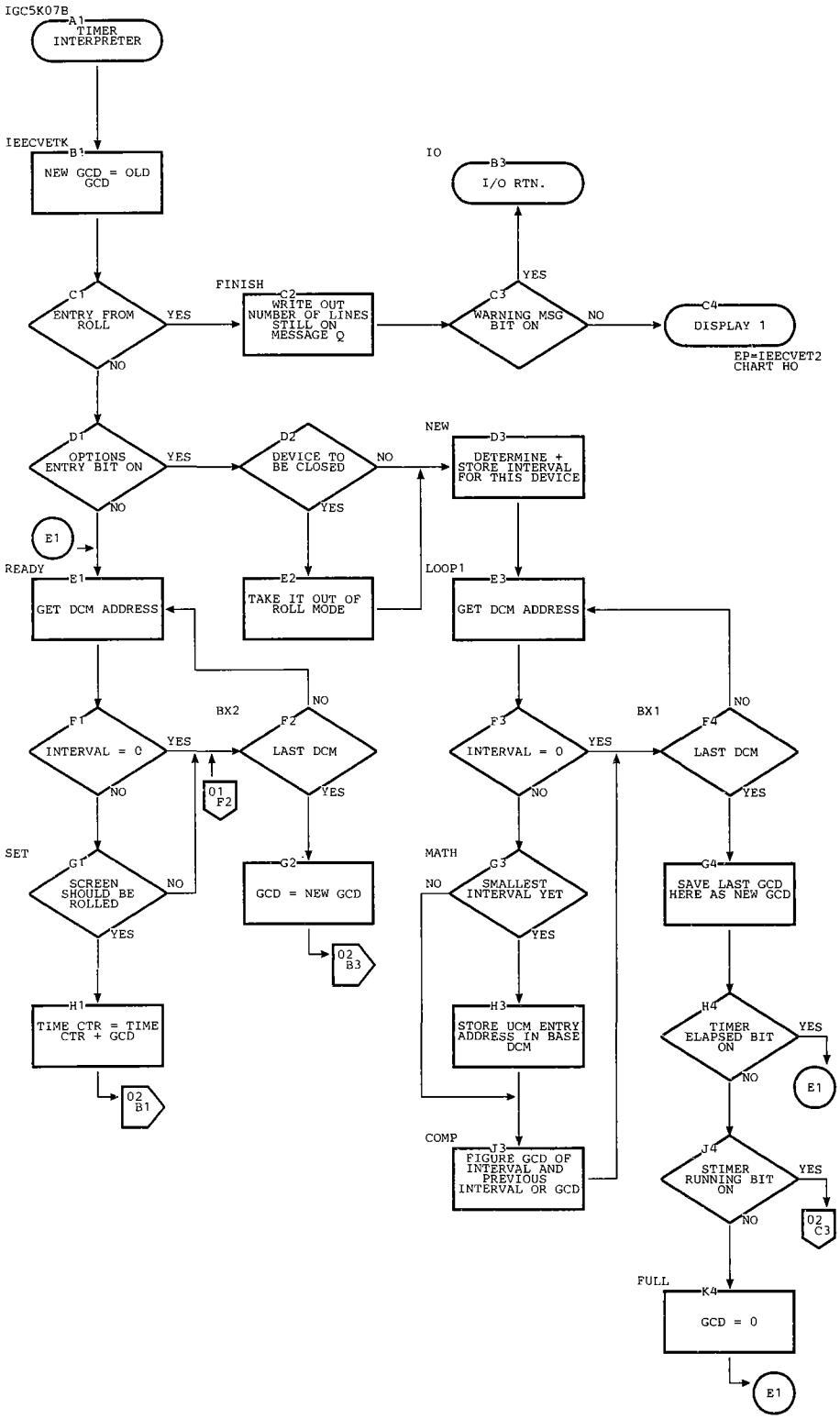


Chart IF. DIDOCS Timer Interpreter (Page 2 of 2)

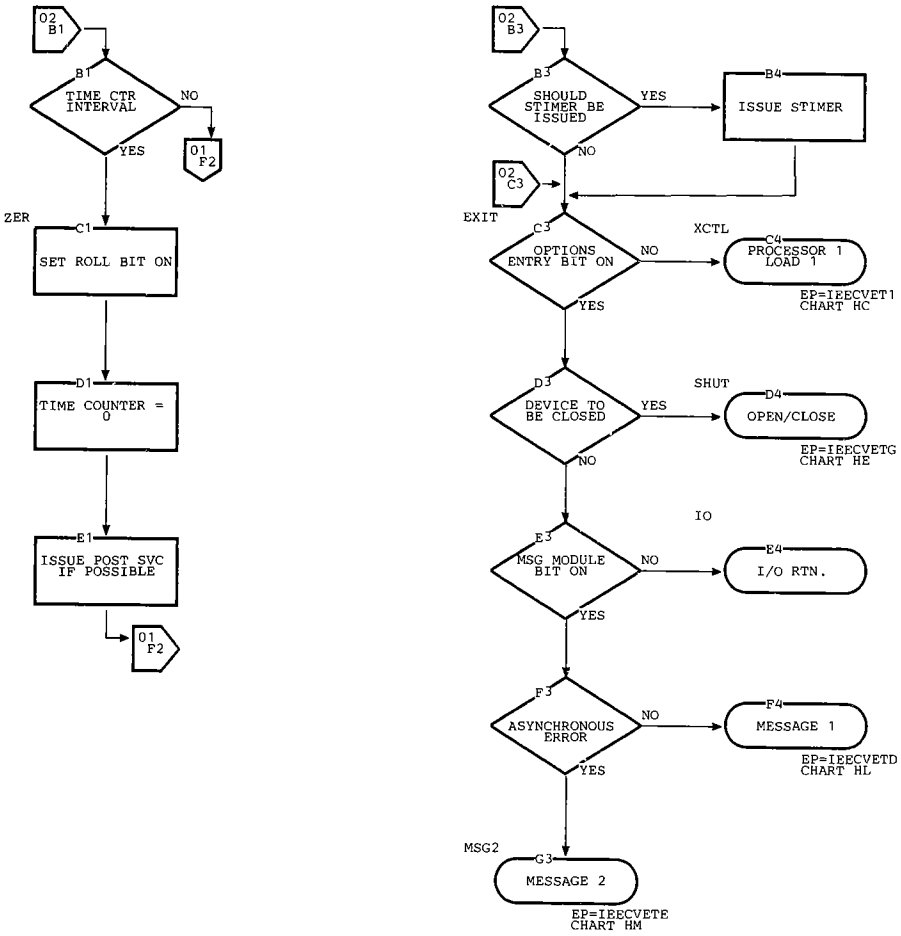


Chart IG. DIDOCS PFK 1 Routine (Page 1 of 2)

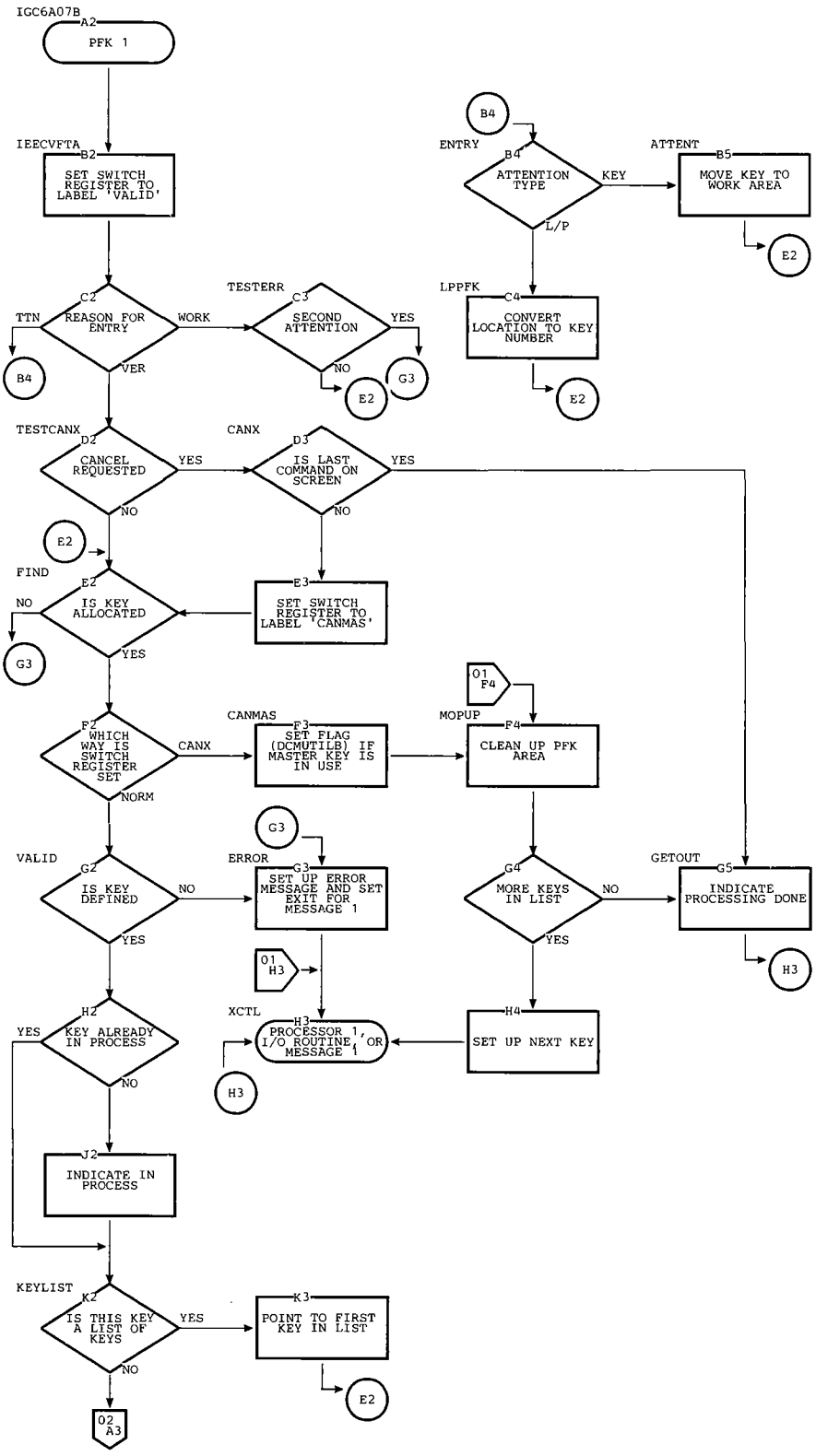


Chart IG. DIDOCS PFK 1 Routine (Page 2 of 2)

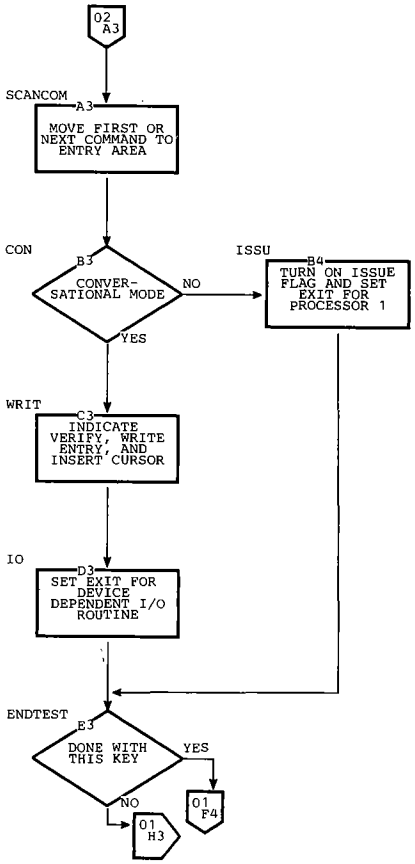


Chart IH. DIDOCS PFK 2 Routine

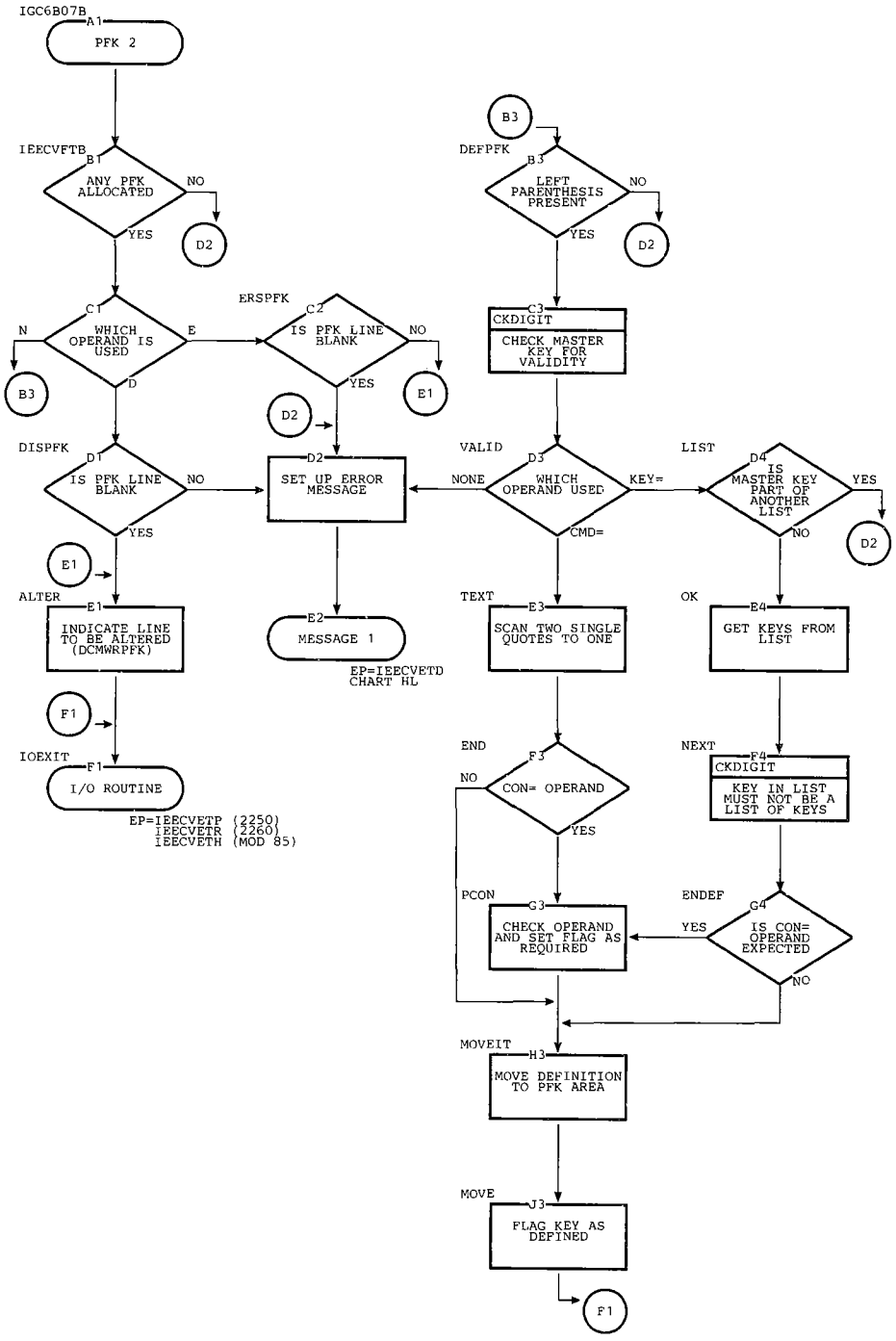


Chart II. DIDOCS Cleanup (Page 1 of 2)

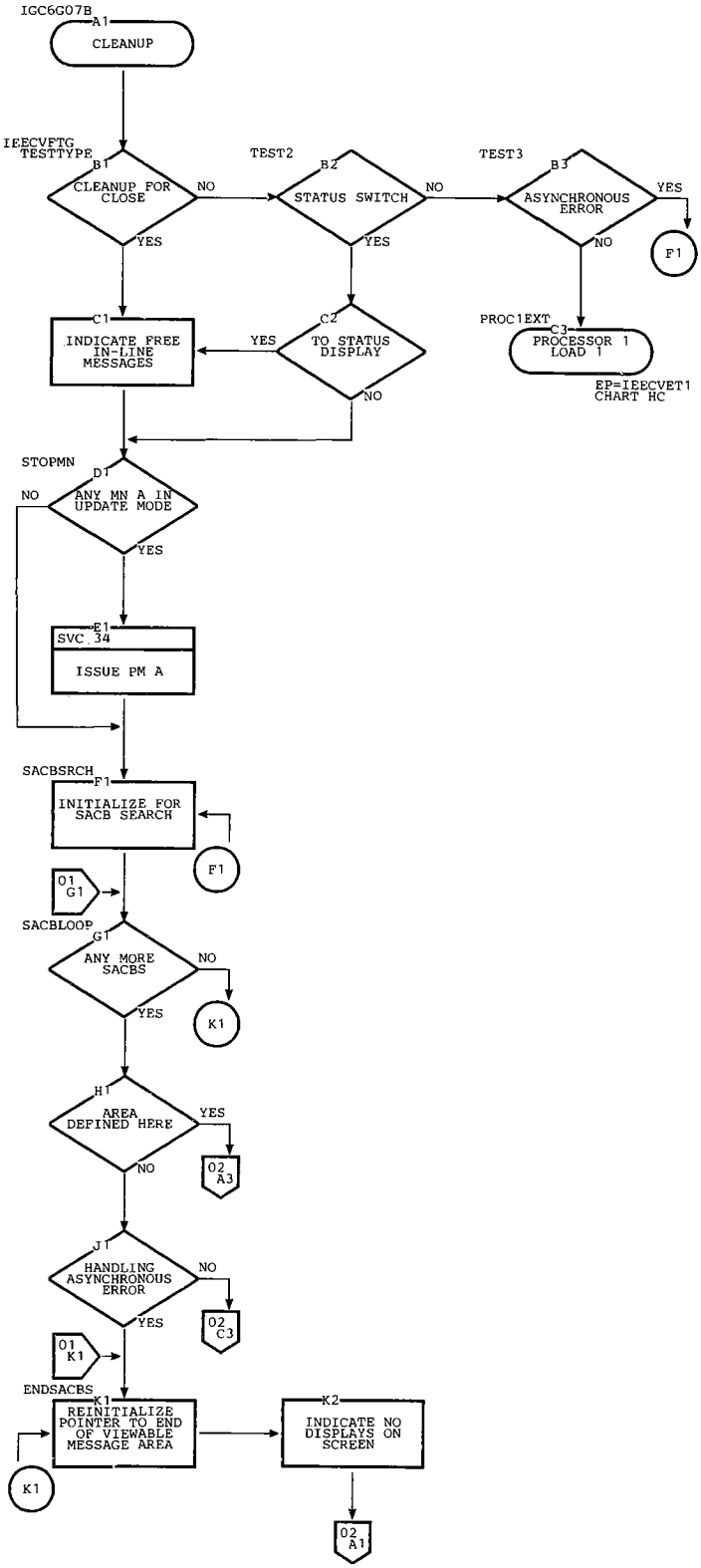


Chart II. DIDOCS Cleanup (Page 2 of 2)

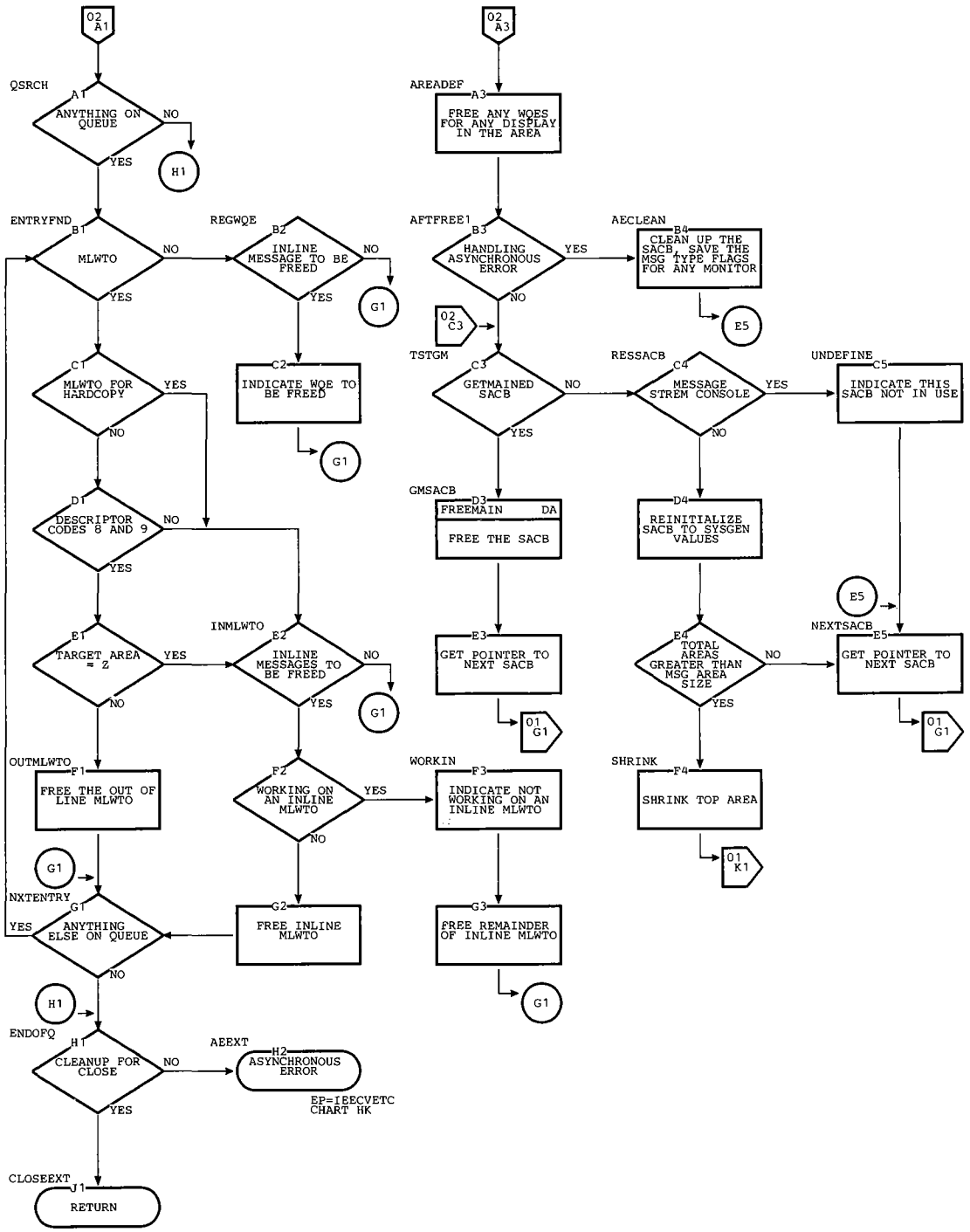


Chart IJ. Status Display Interface 1 Routine

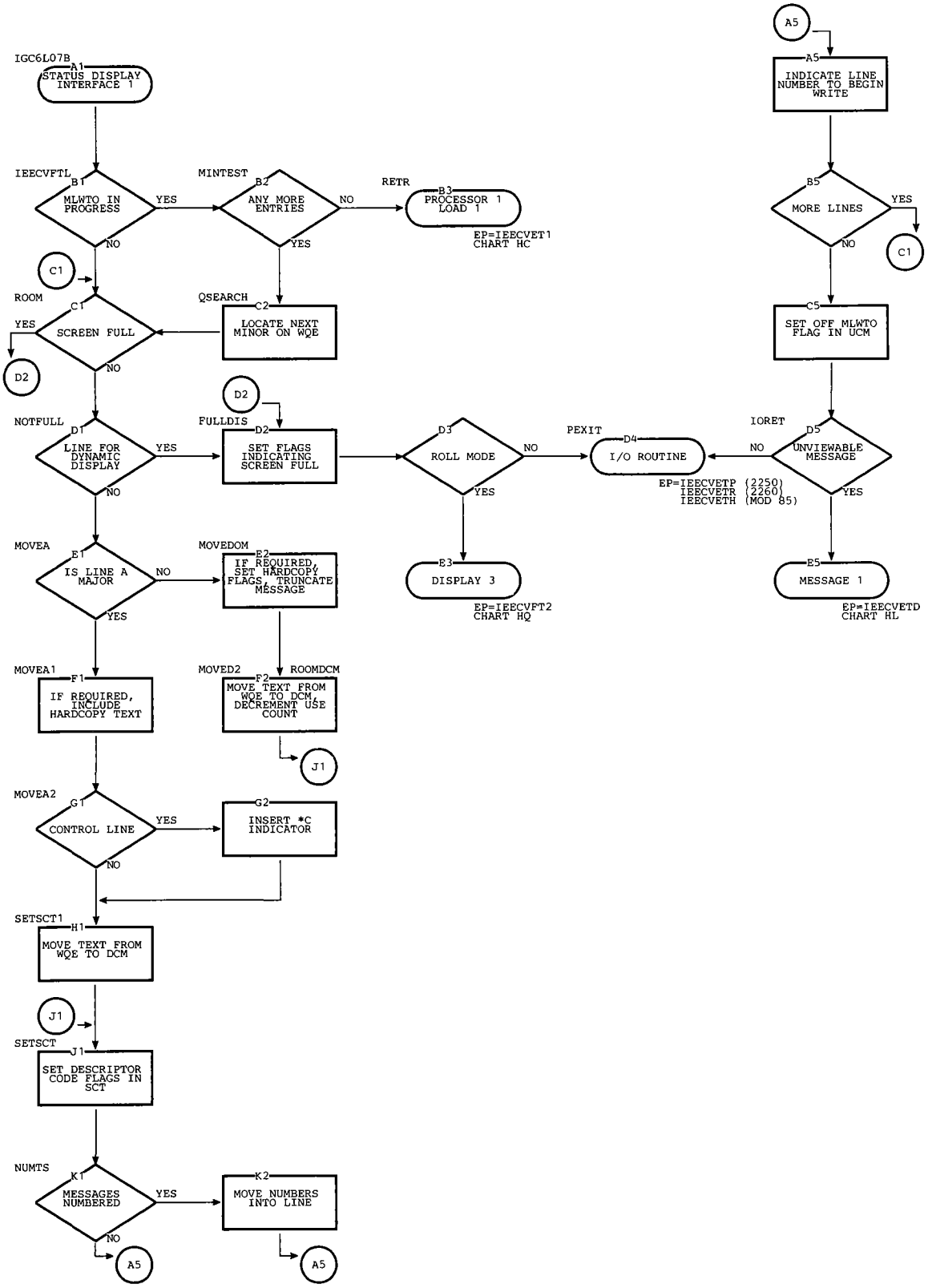


Chart IK. Status Display Interface 2 Routine (Page 1 of 2)

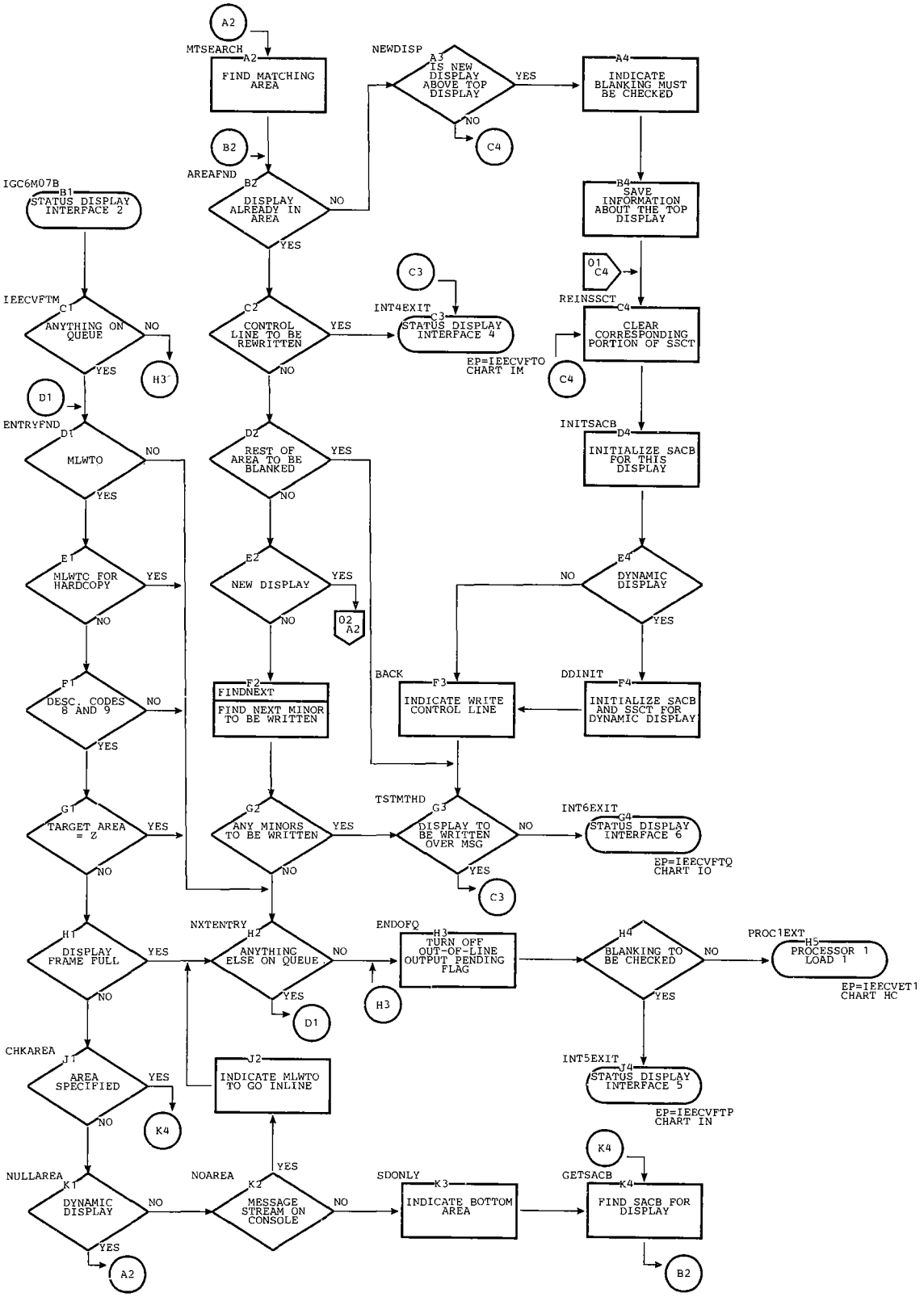


Chart IK. Status Display Interface 2 Routine (Page 2 of 2)

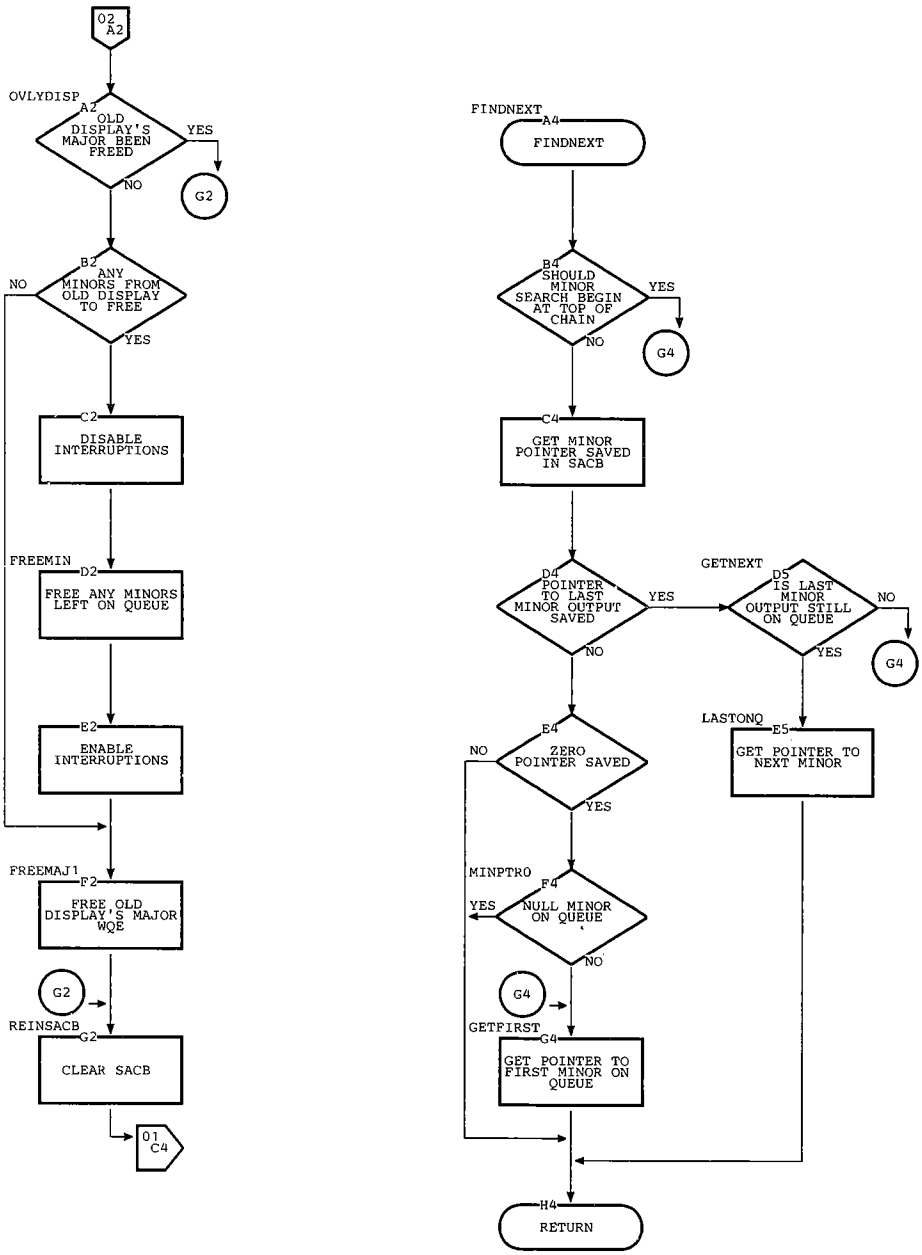


Chart IL. Status Display Interface 3 Routine (Page 1 of 2)

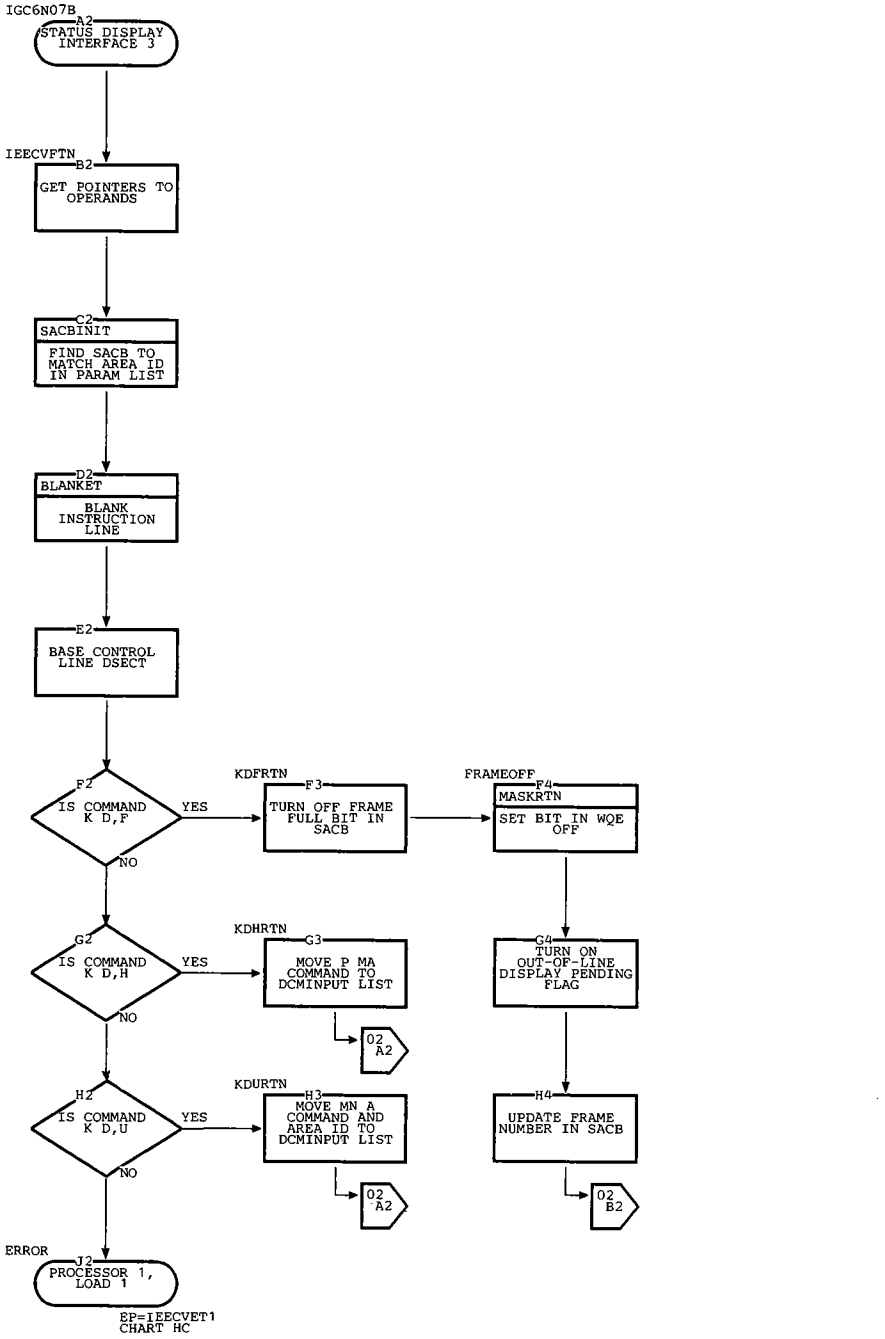


Chart IL. Status Display Interface 3 Routine (Page 2 of 2)

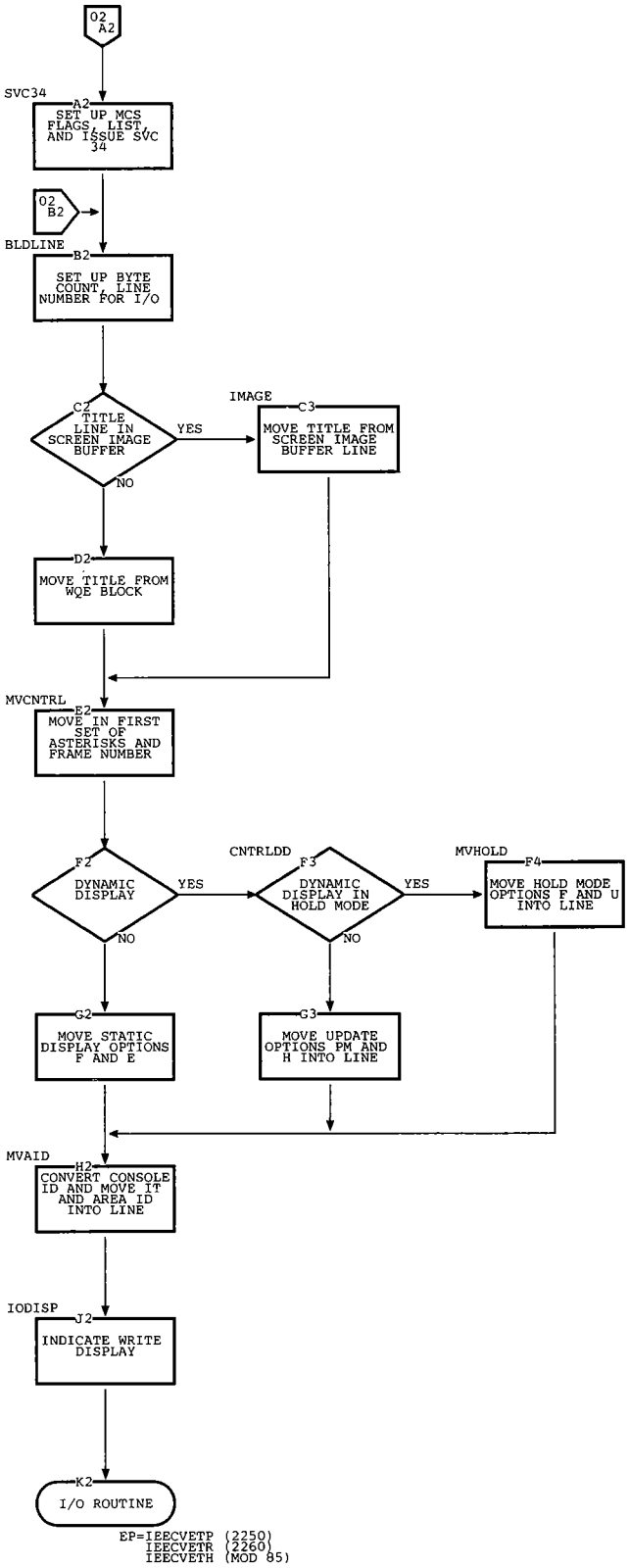


Chart IM. Status Display Interface 4 Routine

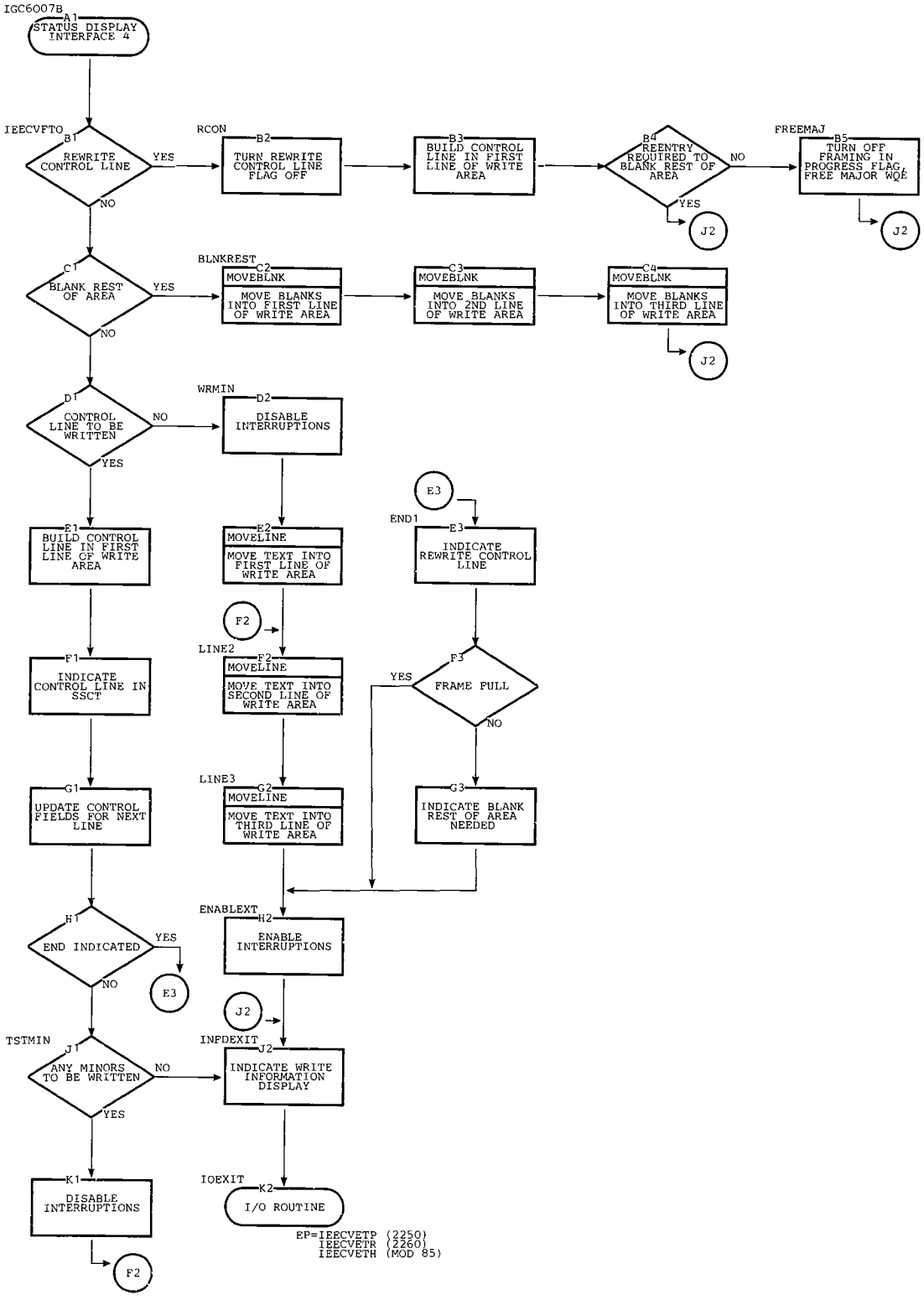


Chart IN. Status Display Interface 5 Routine (Page 1 of 3)

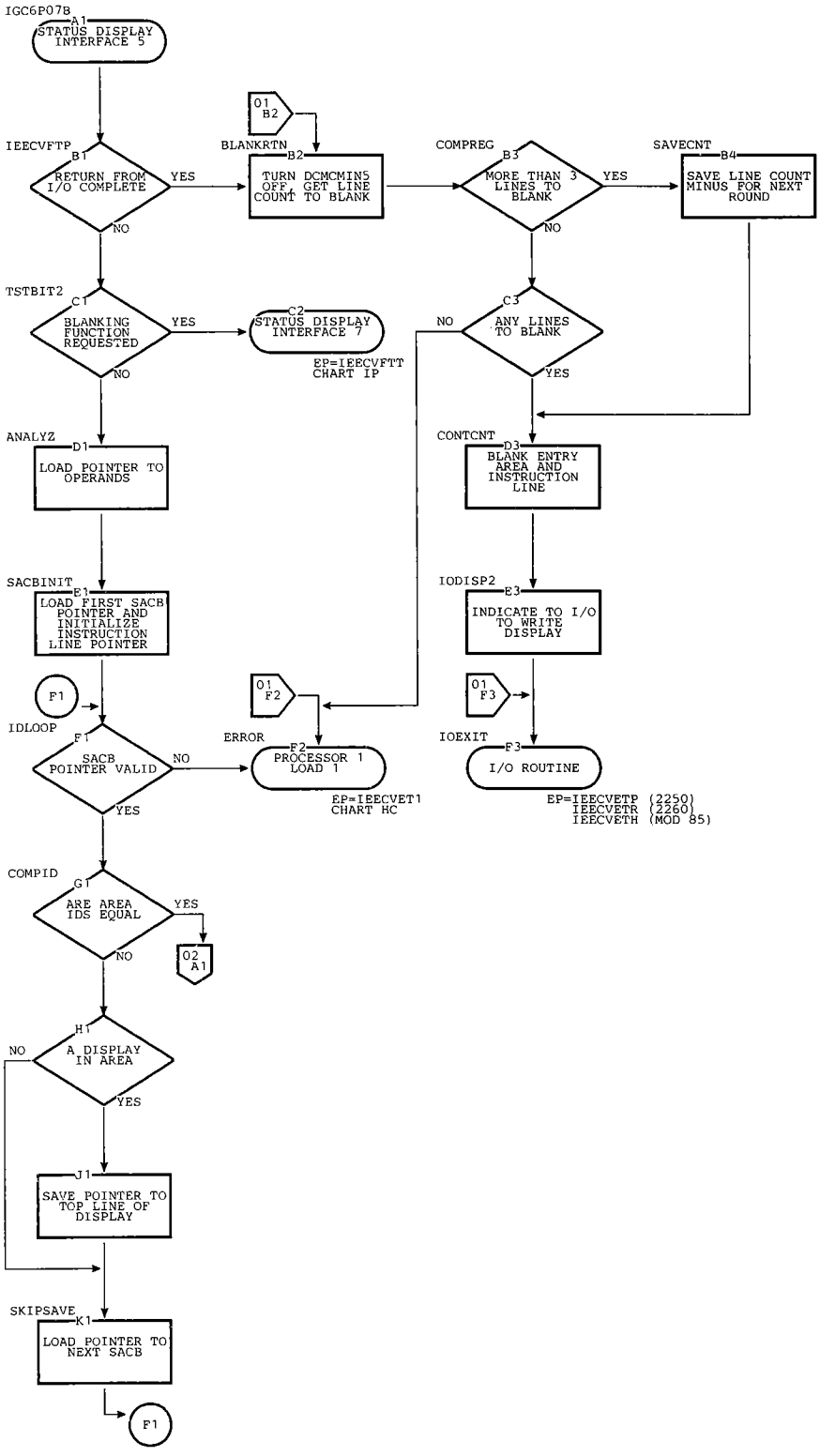


Chart IN. Status Display Interface 5 Routine (Page 2 of 3)

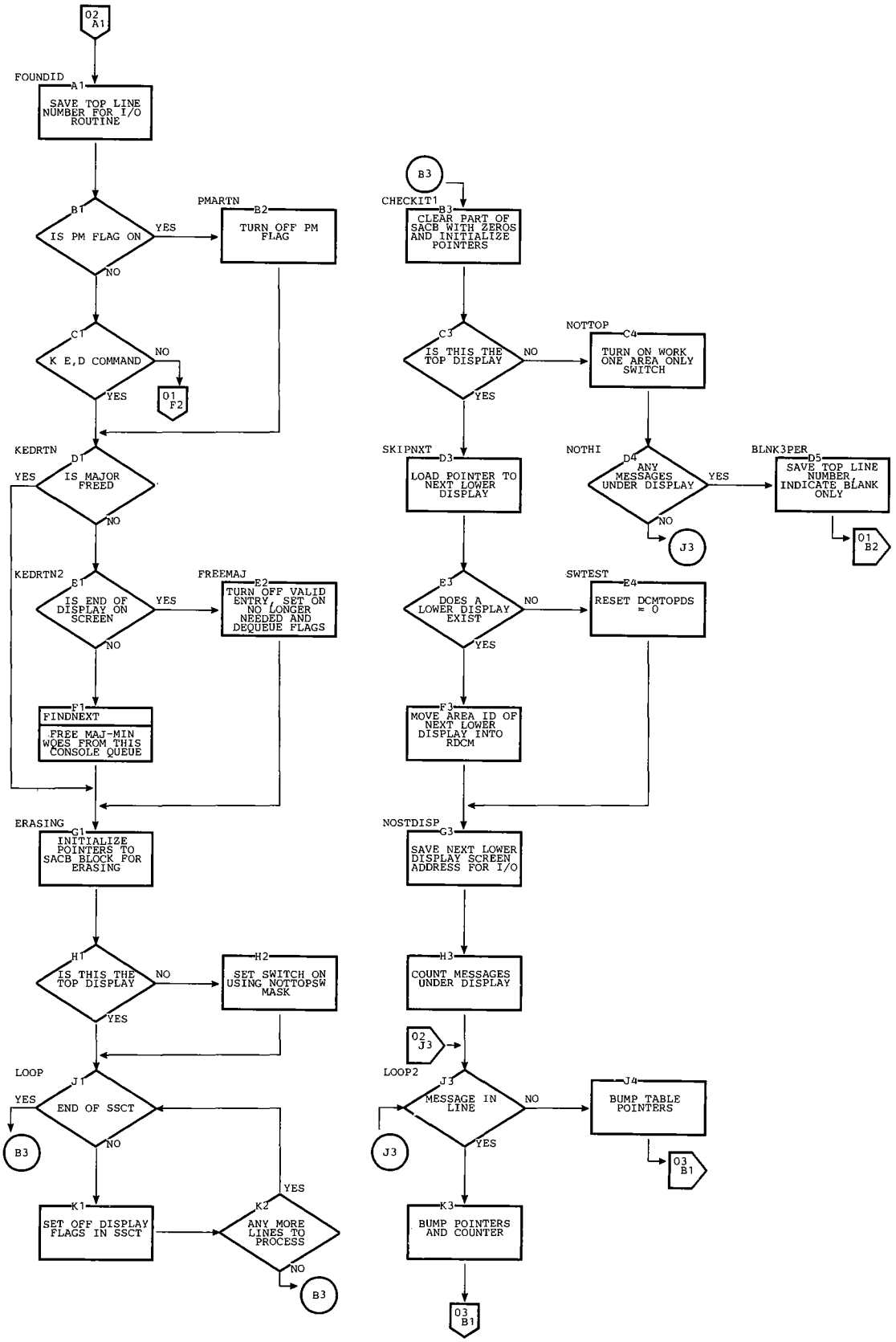


Chart IN. Status Display Interface 5 Routine (Page 3 of 3)

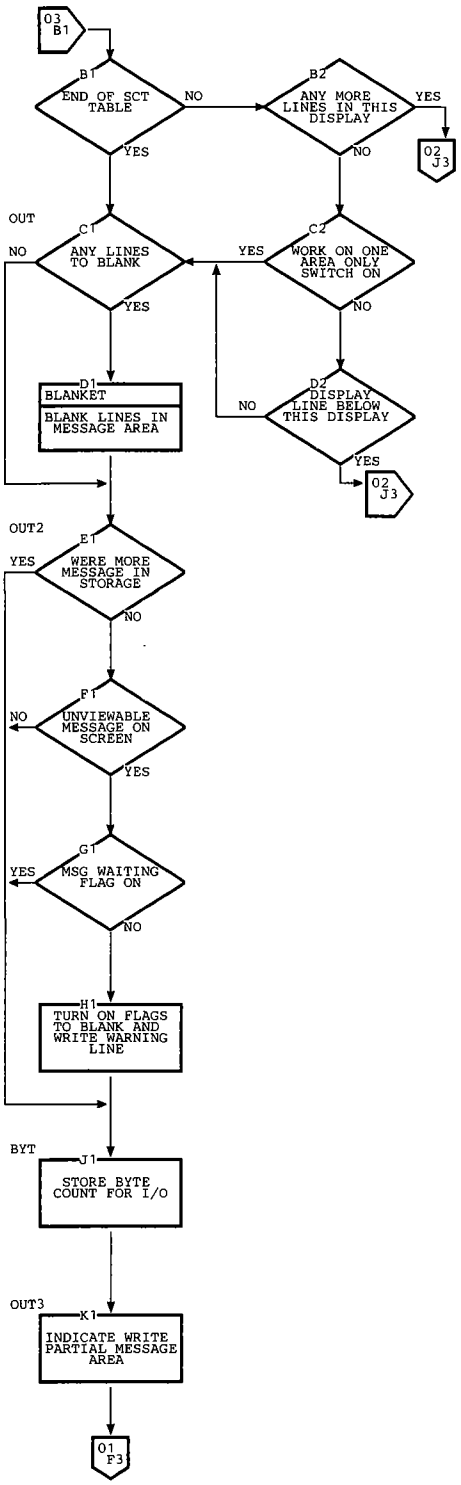


Chart IO. Status Display Interface 6 Routine (Page 1 of 2)

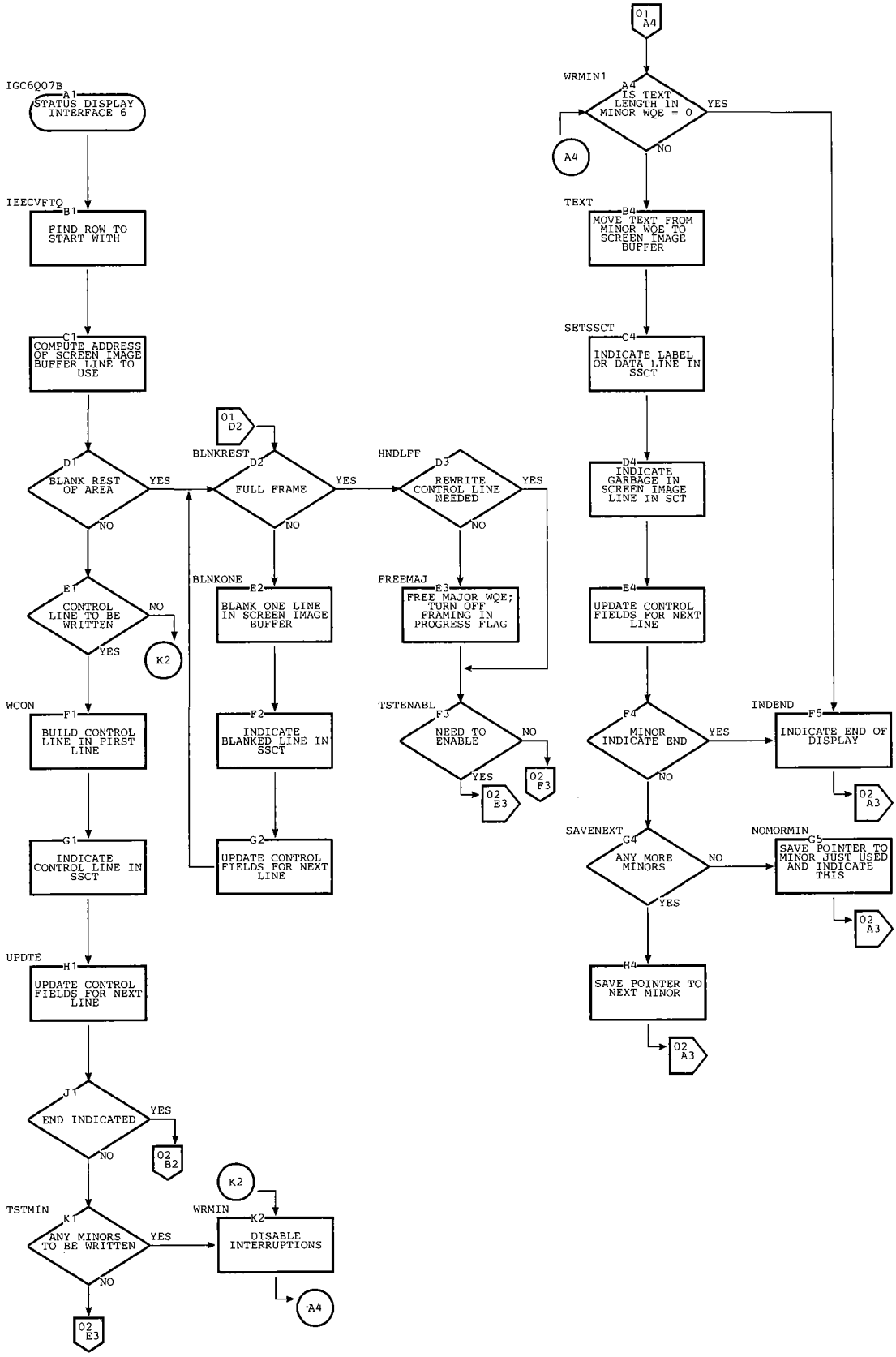


Chart IO. Status Display Interface 6 Routine (Page 2 of 2)

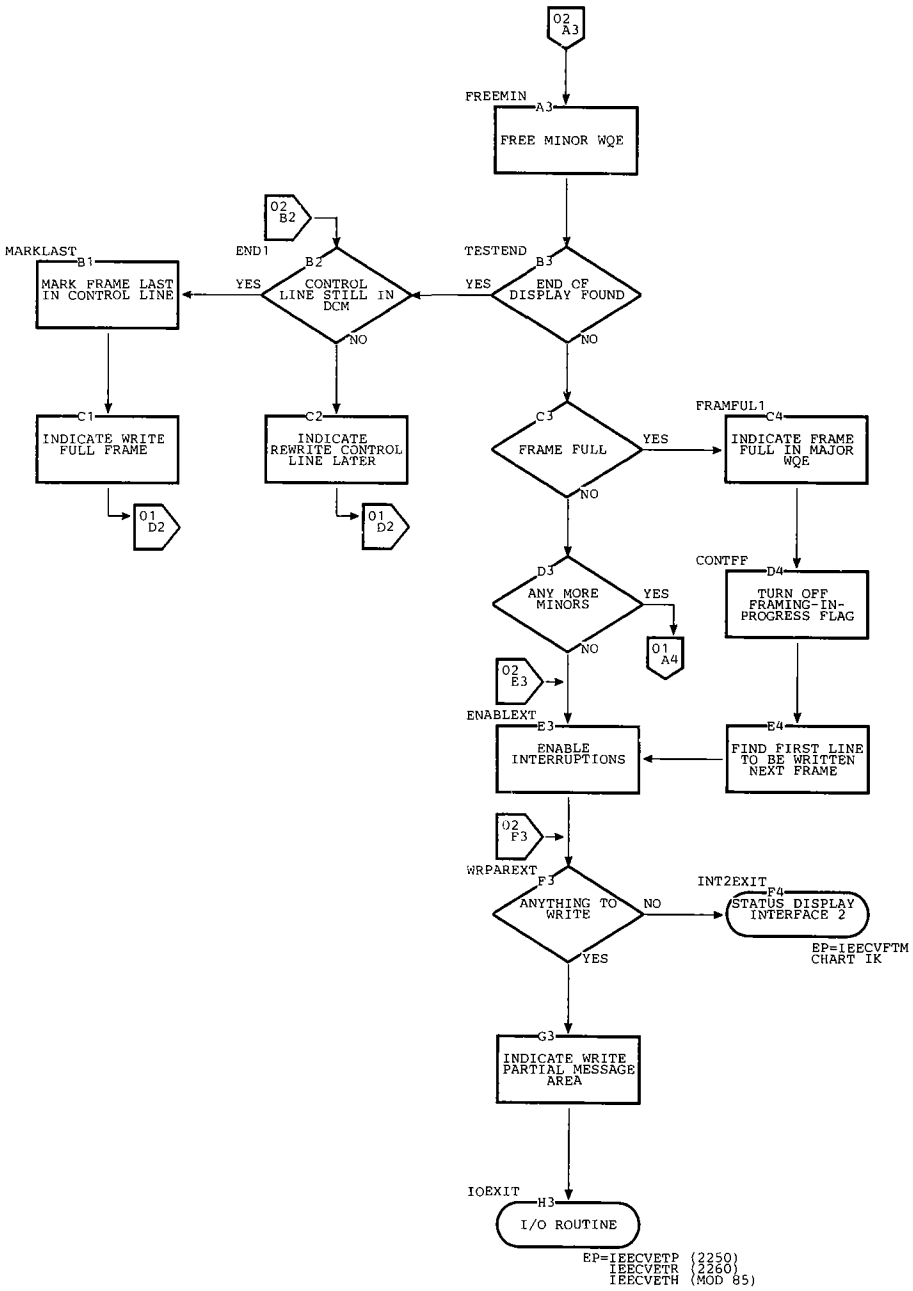


Chart IP. Status Display Interface 7 Routine

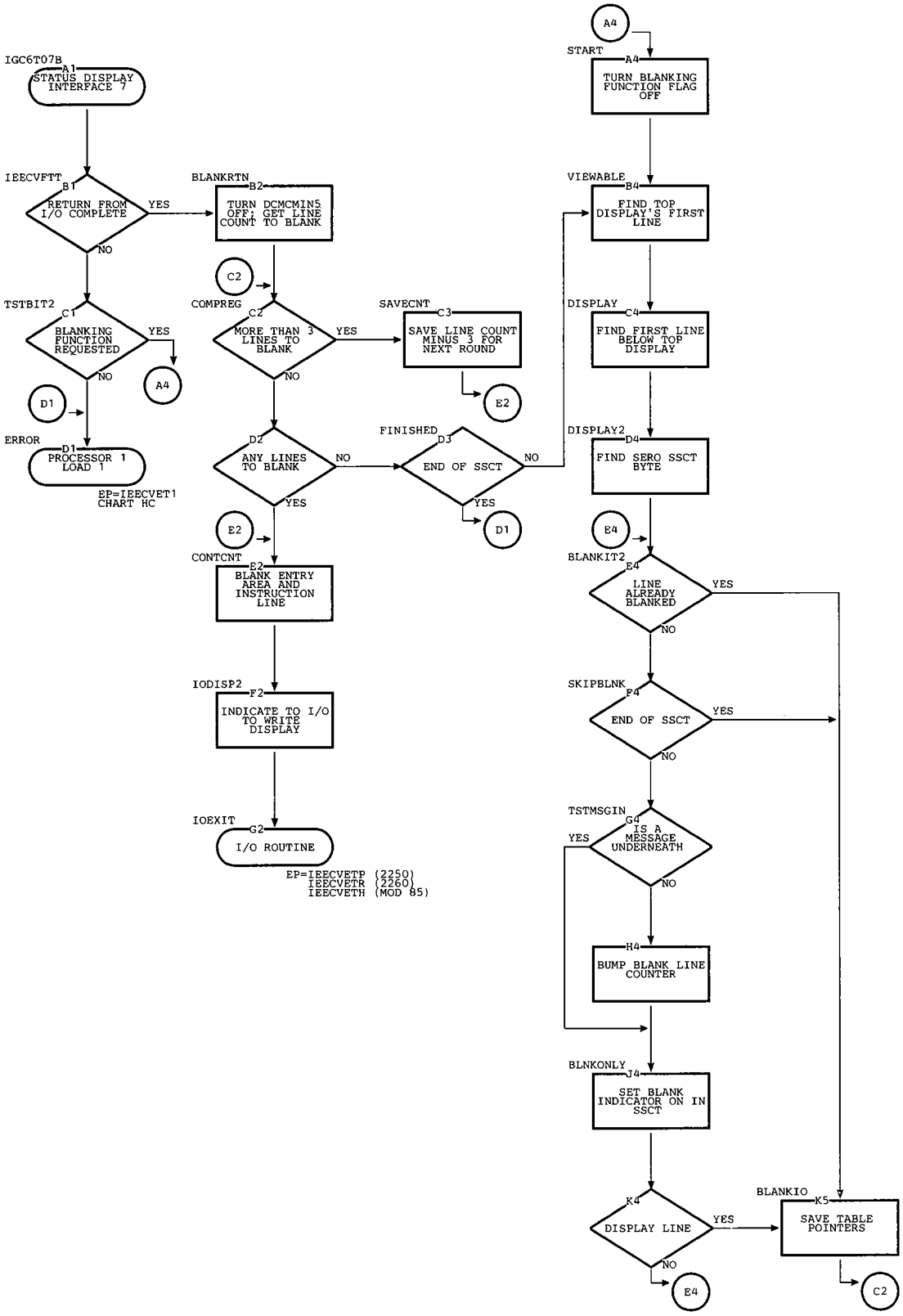


Chart JA. Checkpoint Housekeeping 1 Routine

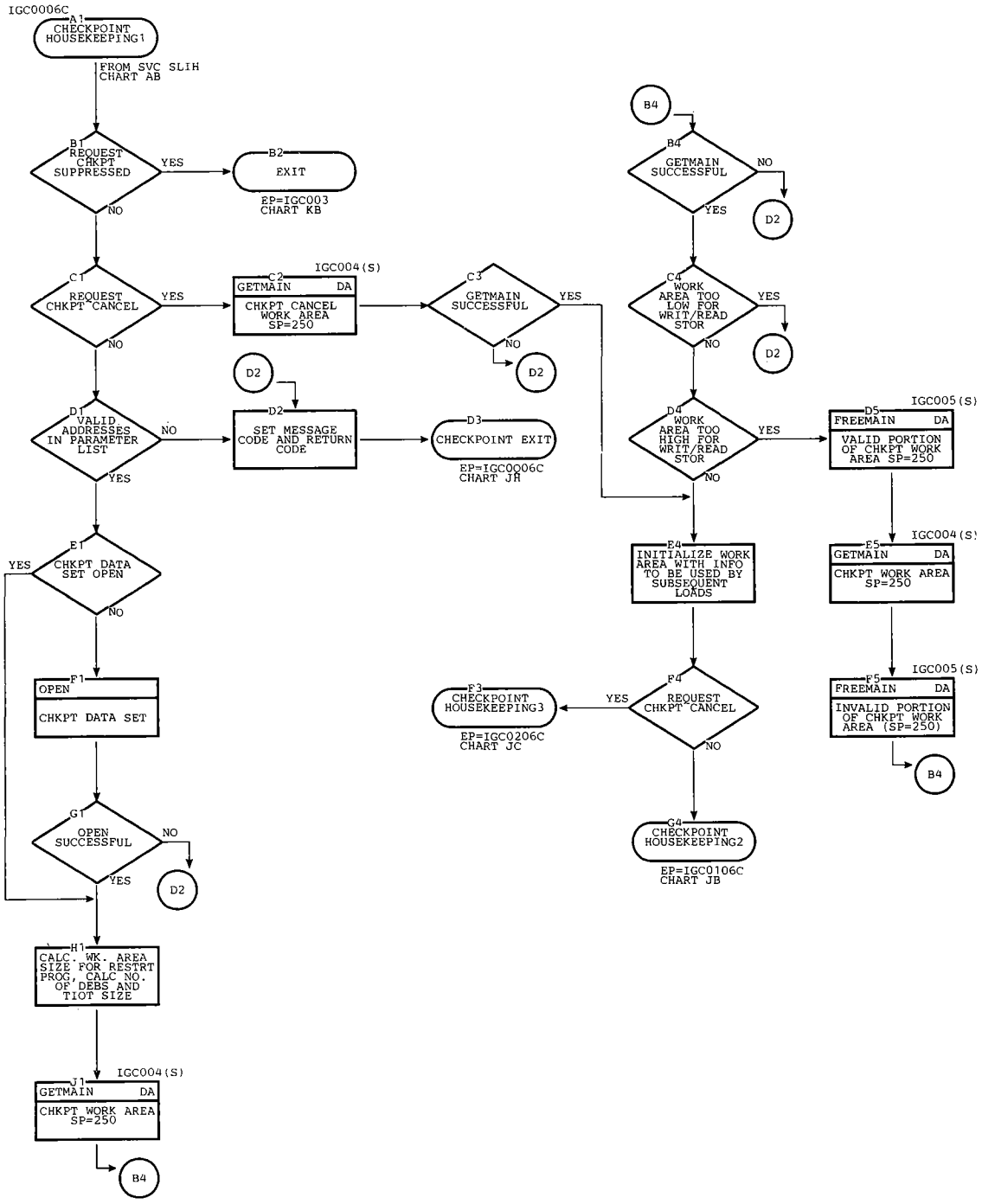


Chart JB. Checkpoint Housekeeping 2 Routine

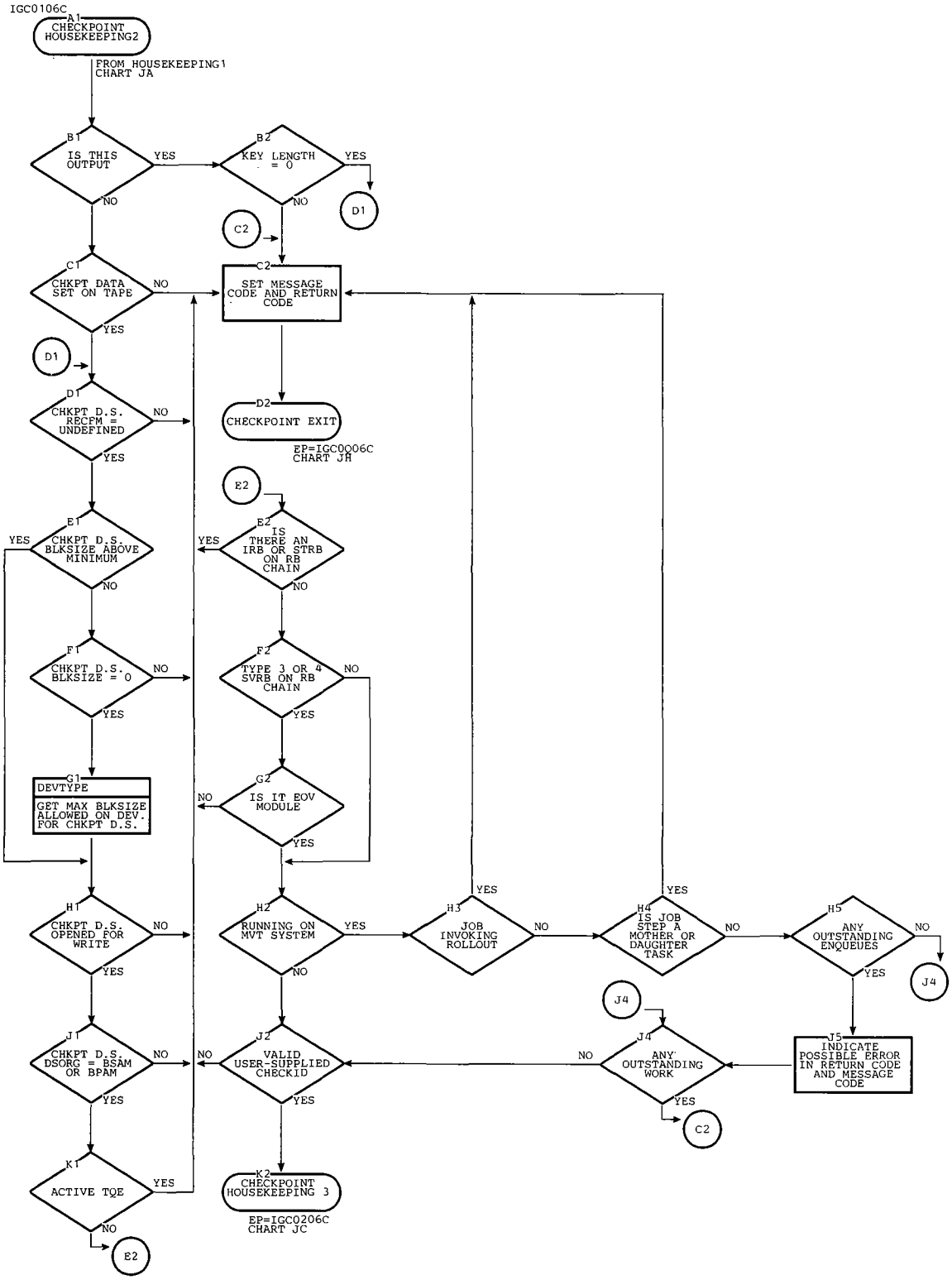


Chart JC. Checkpoint Housekeeping 3 Routine

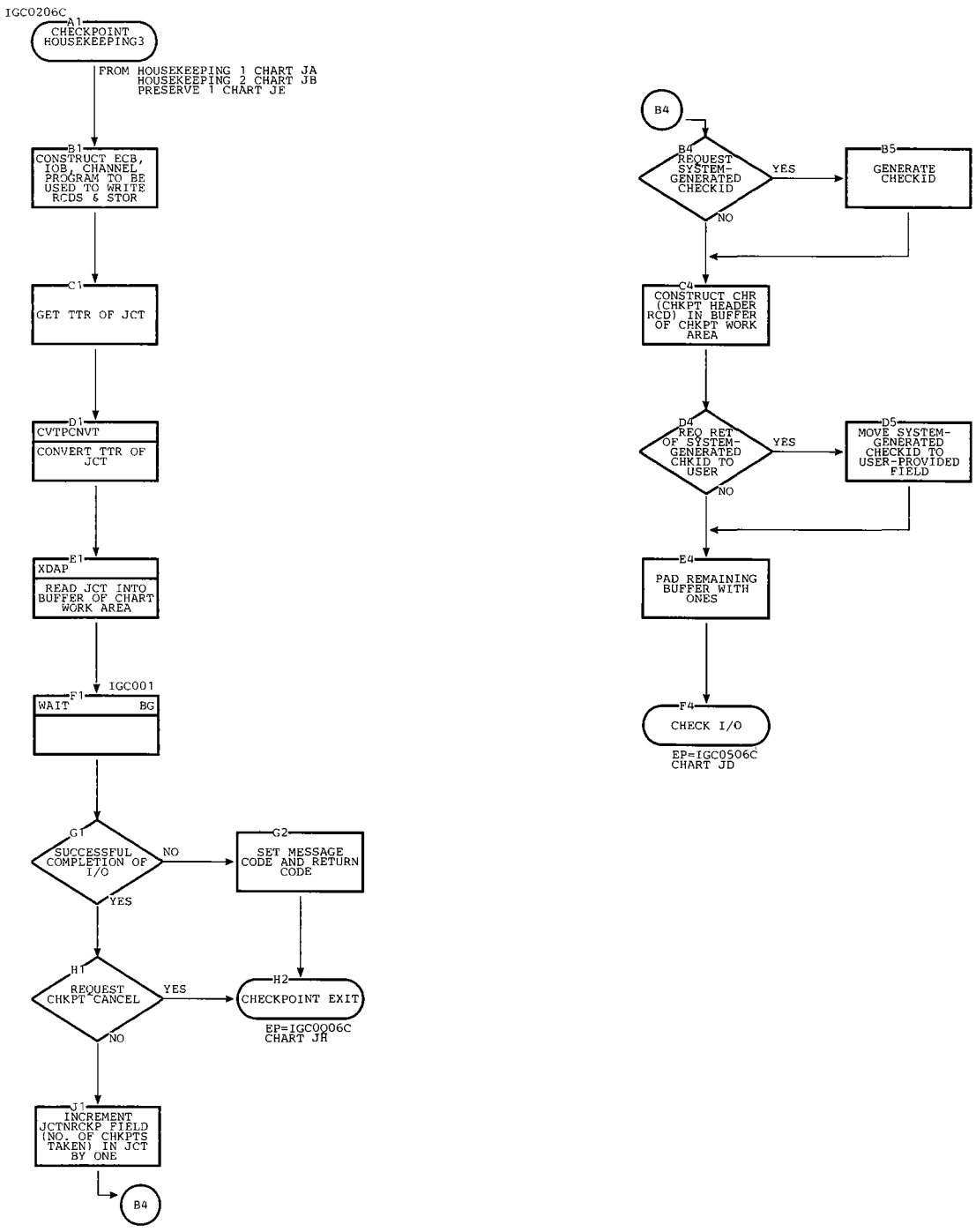


Chart JD. Checkpoint Check I/O Routine

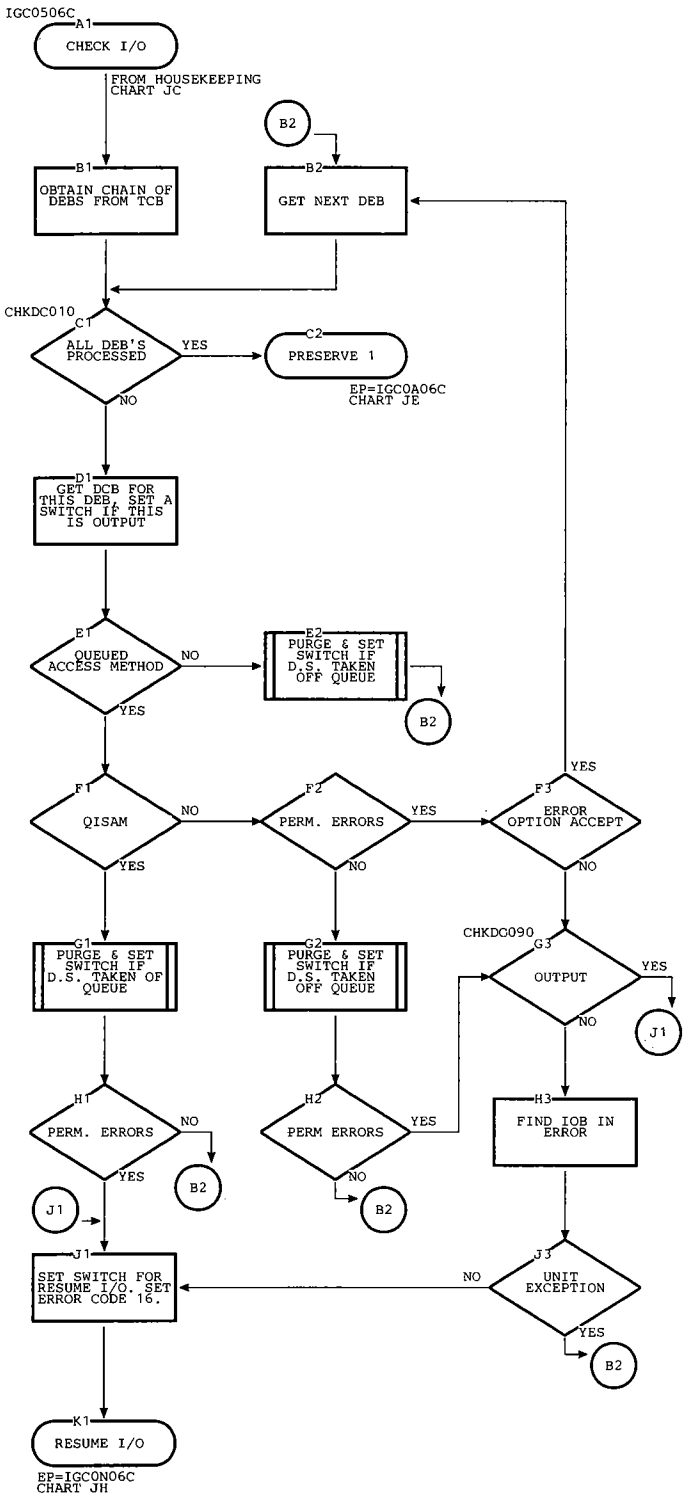


Chart JE. Checkpoint Preserve 1 and 2 Routines

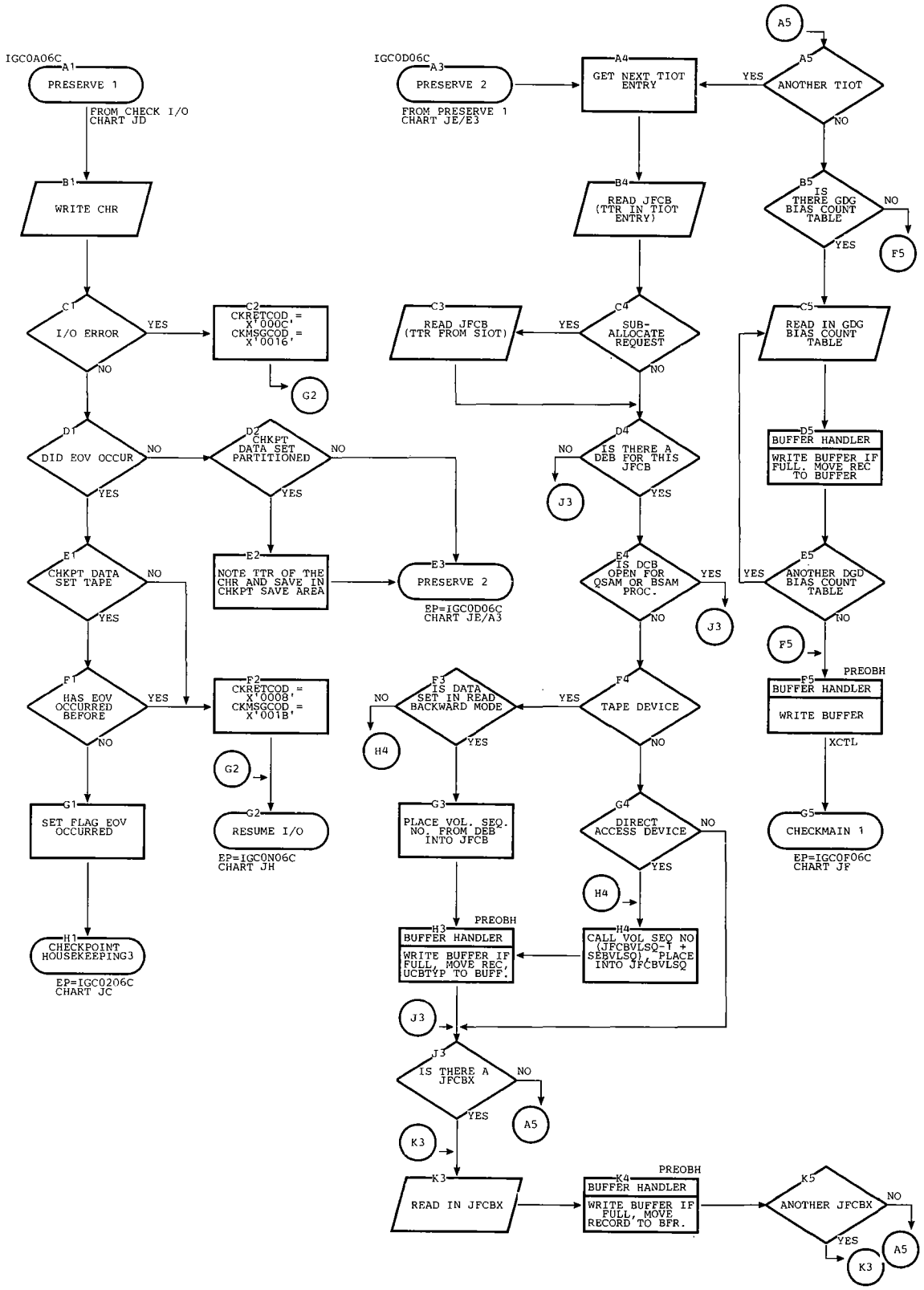


Chart JF. Checkpoint Checkmain 1 and 2 Routines (Page 1 of 2)

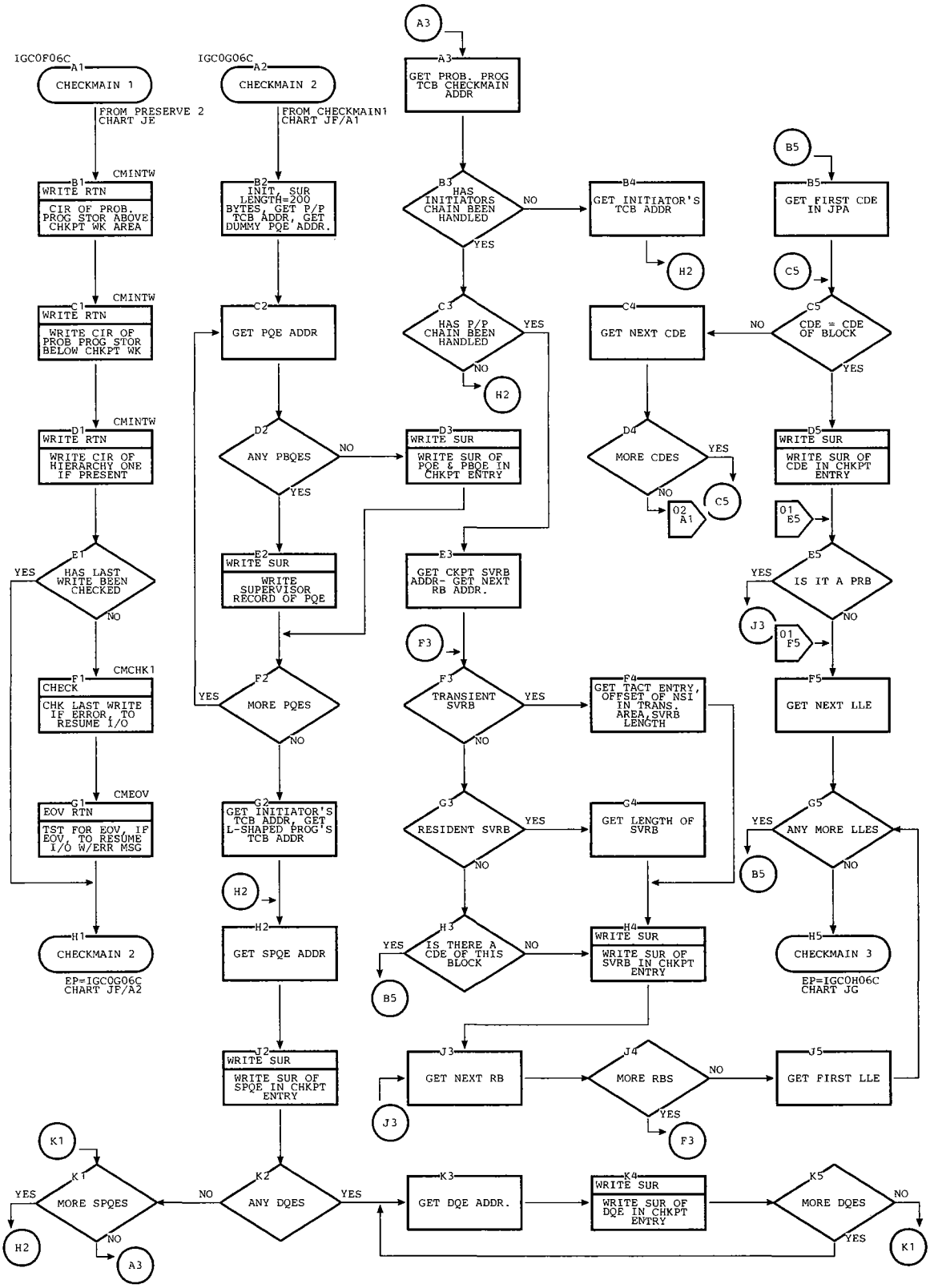


Chart JF. Checkpoint Checkmain 2 Routine (Page 2 of 2)

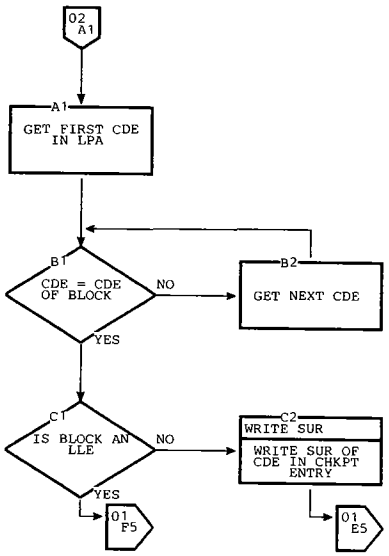


Chart JG. Checkpoint Checkmain 3 Routine

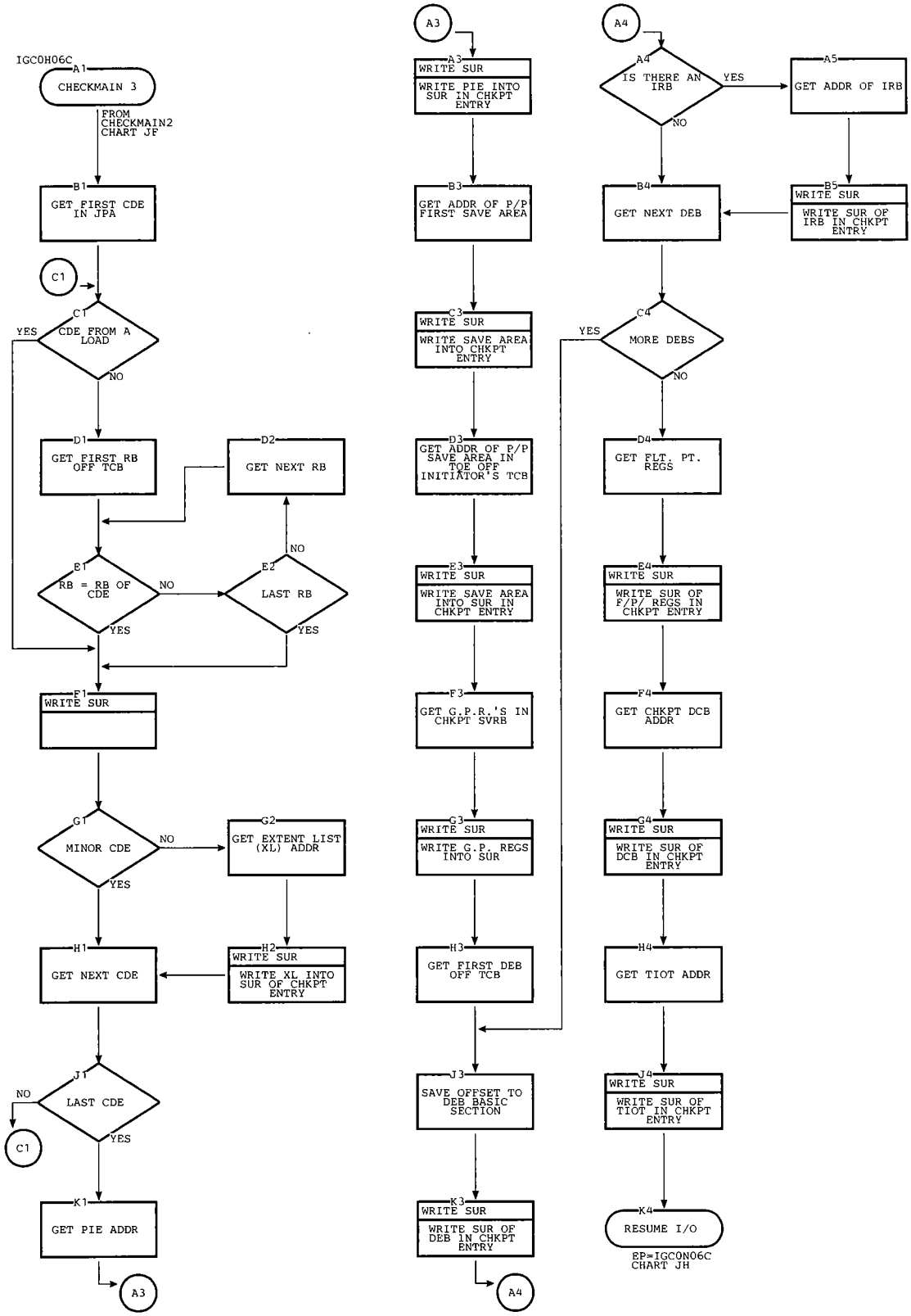


Chart JH. Checkpoint Resume I/O and Exit Routines

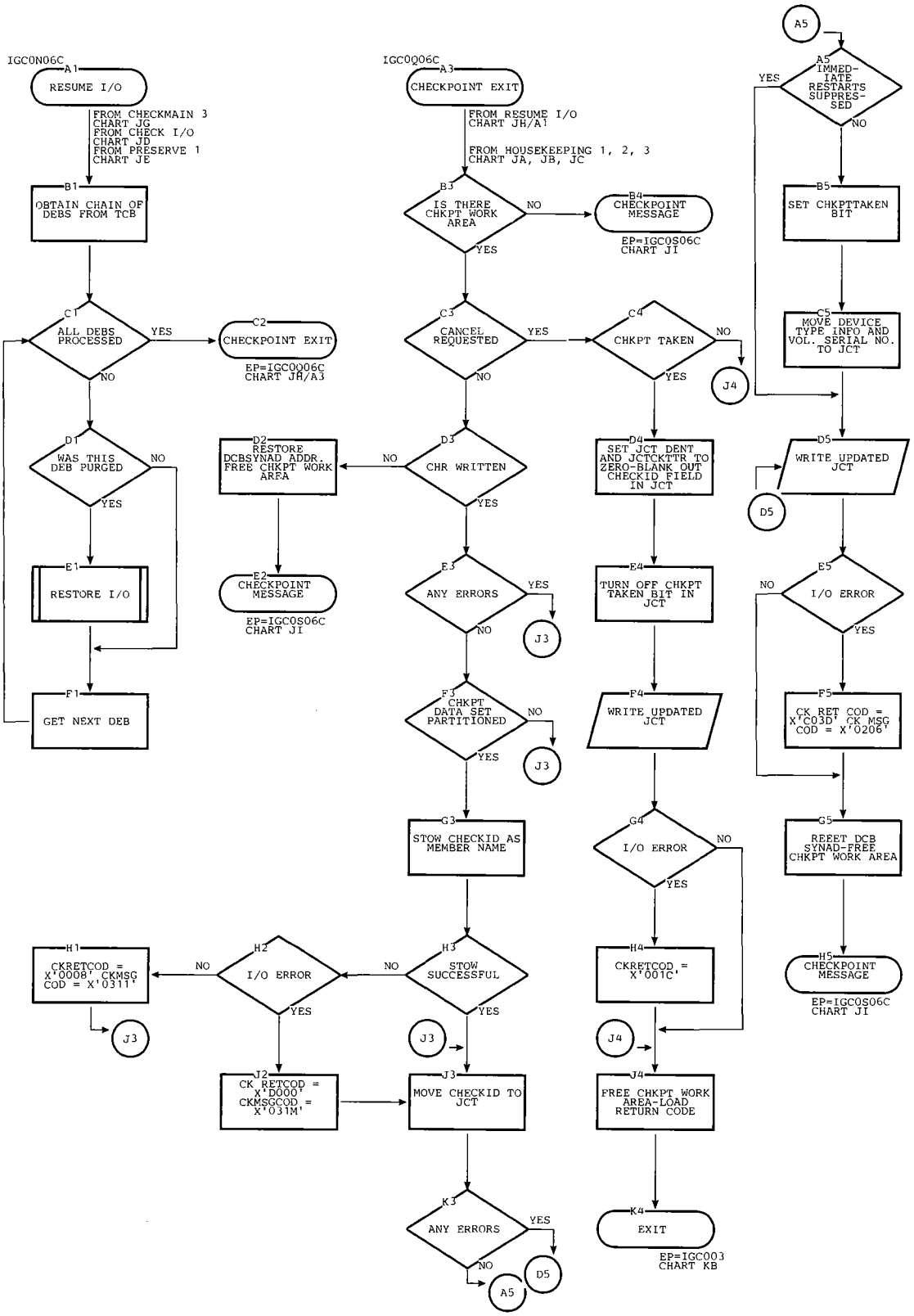


Chart JI. Checkpoint Message Routine

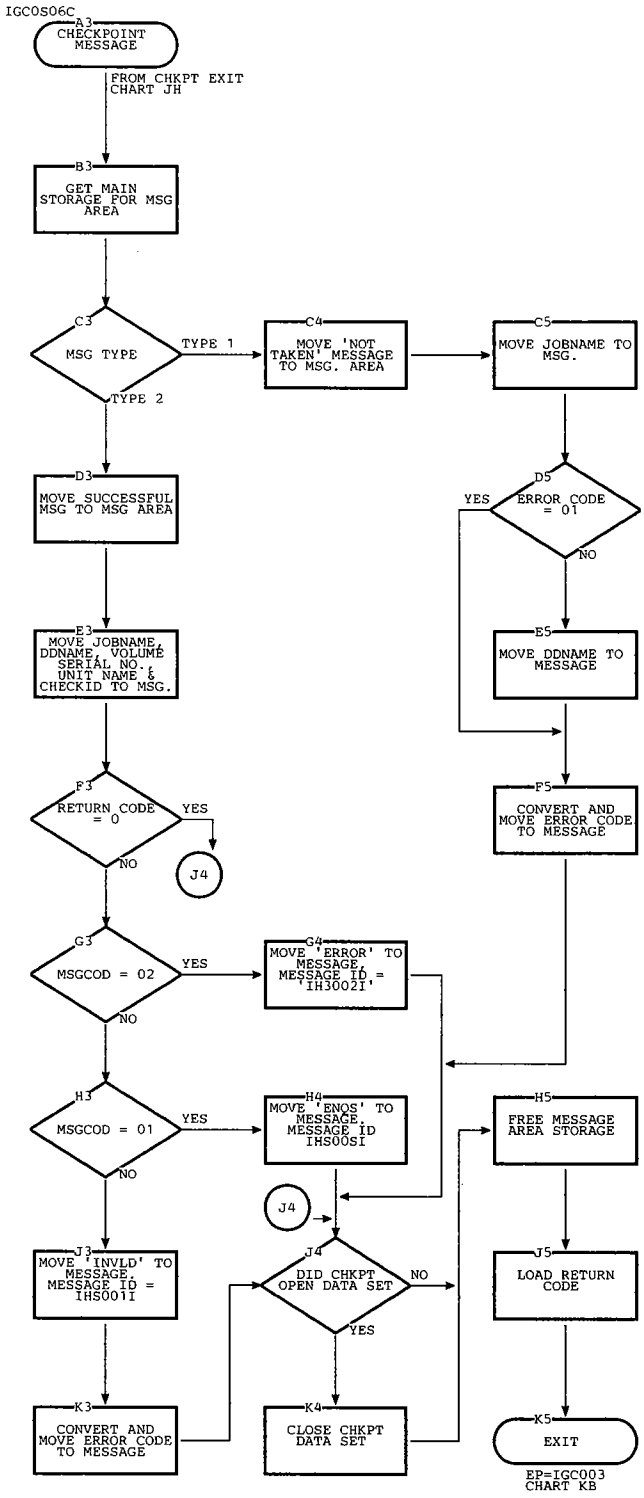


Chart JJ. Restart Housekeeping 1 and 2 Routines

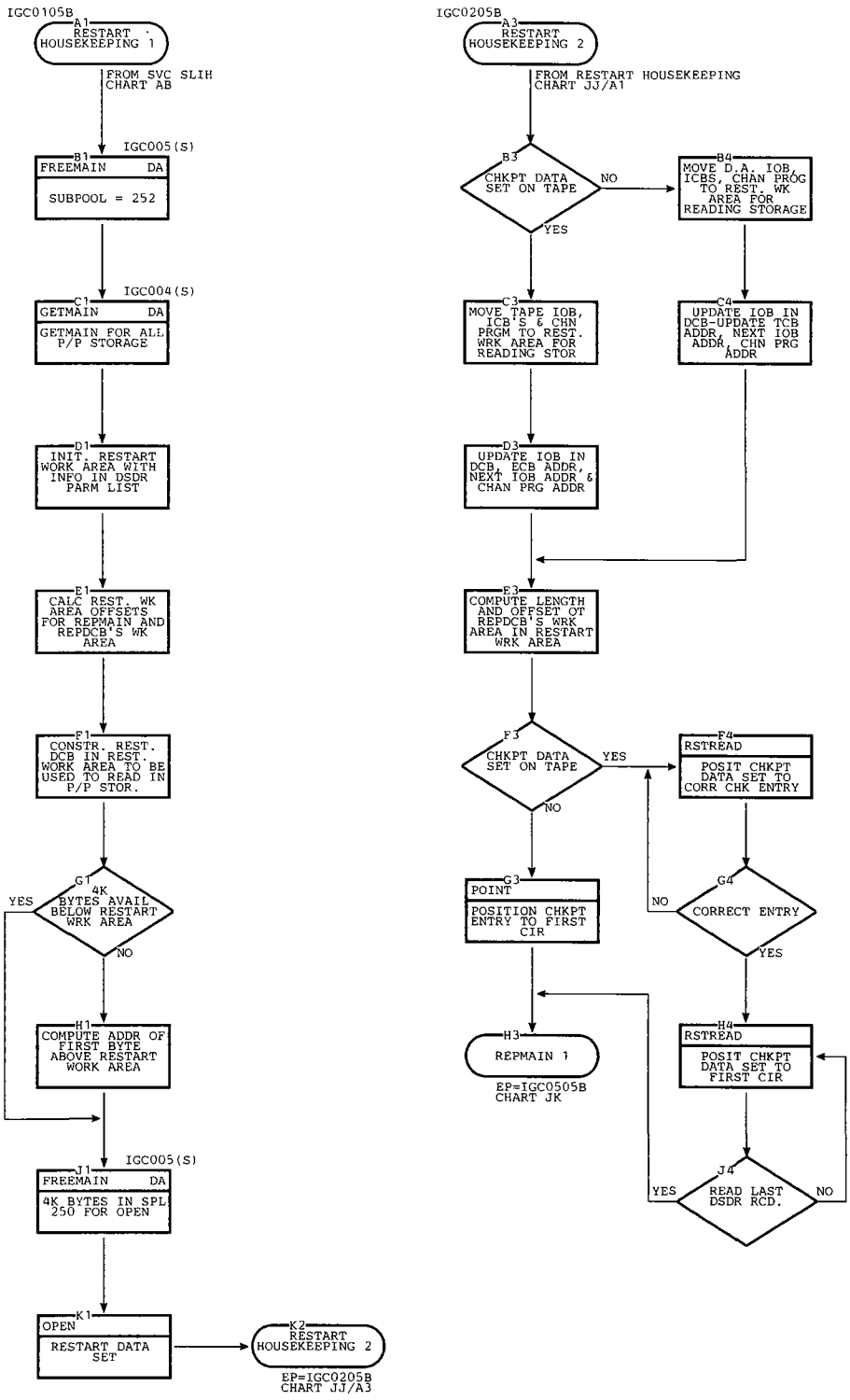


Chart JK. Restart Repmain 1 Routine

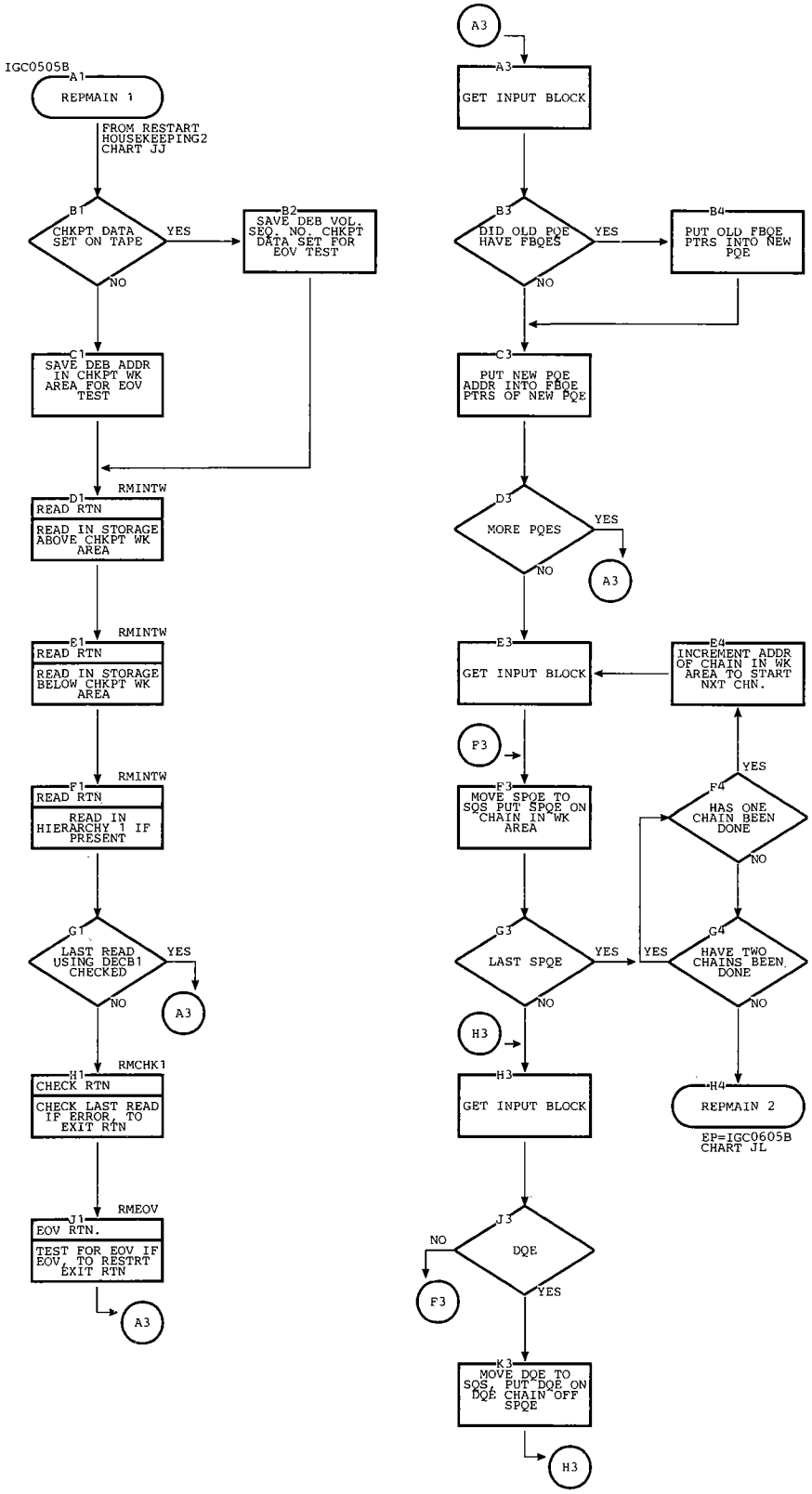


Chart JL. Restart Repmain 2 Routine

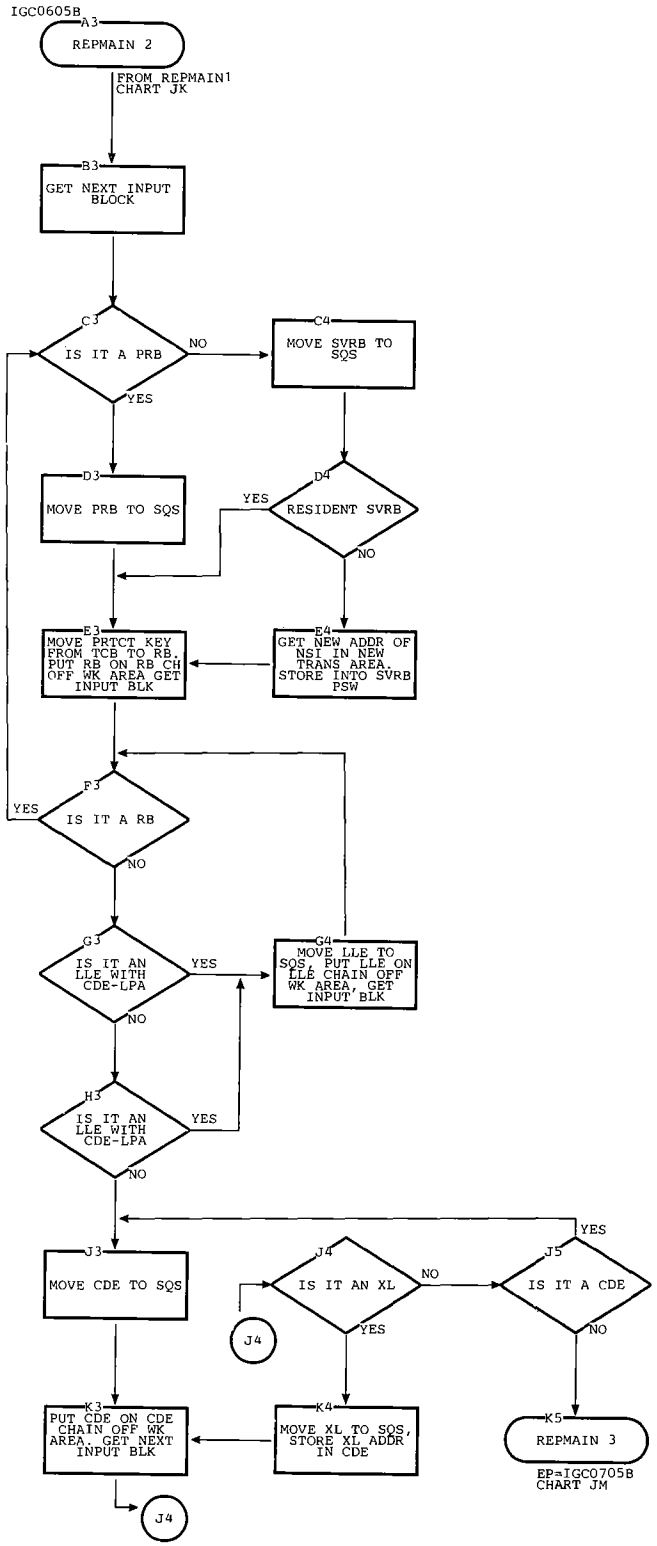


Chart JM. Restart Repmain 3 and 4 Routines

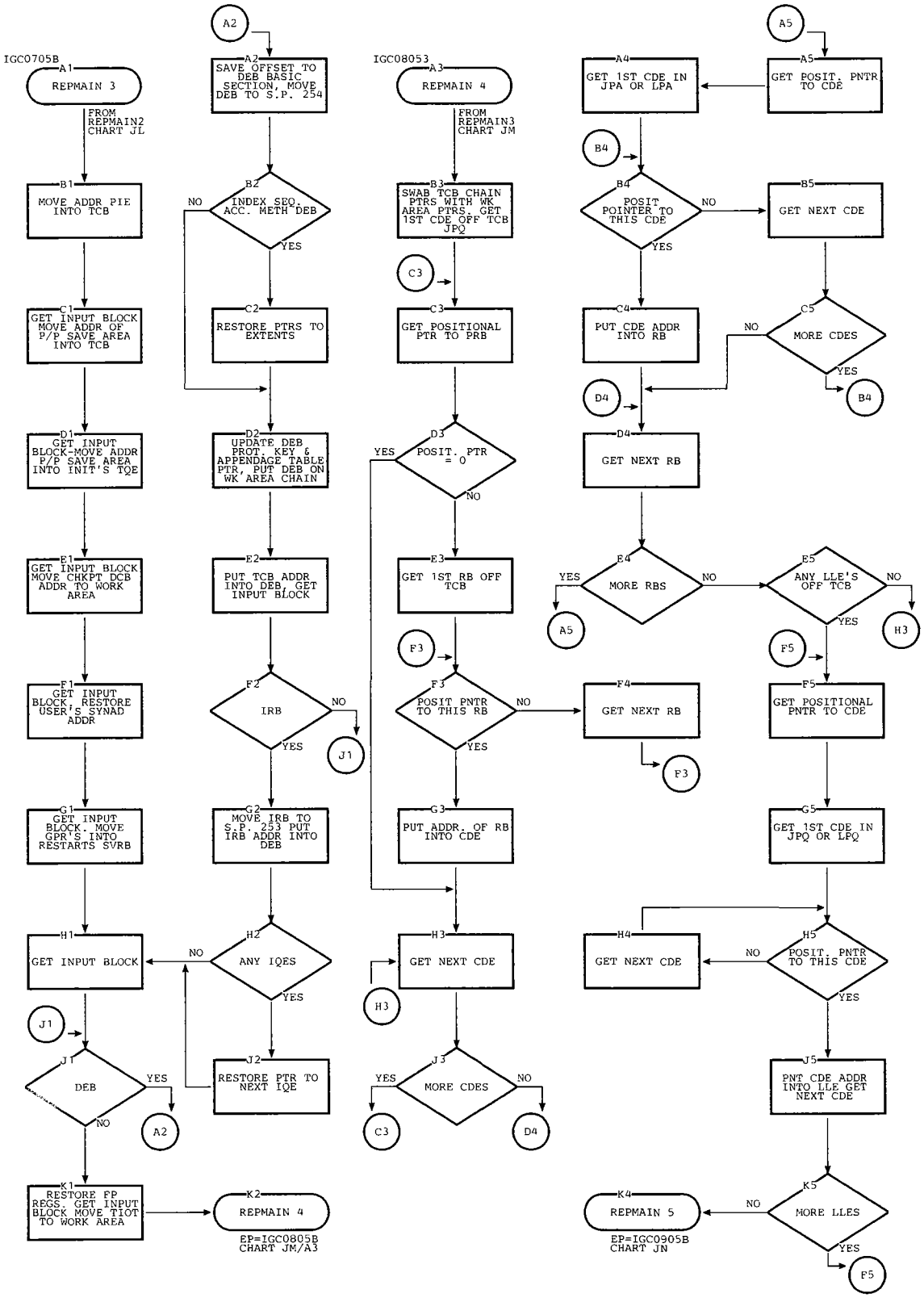


Chart JN. Restart Repmain 5 Routine (Page 1 of 2)

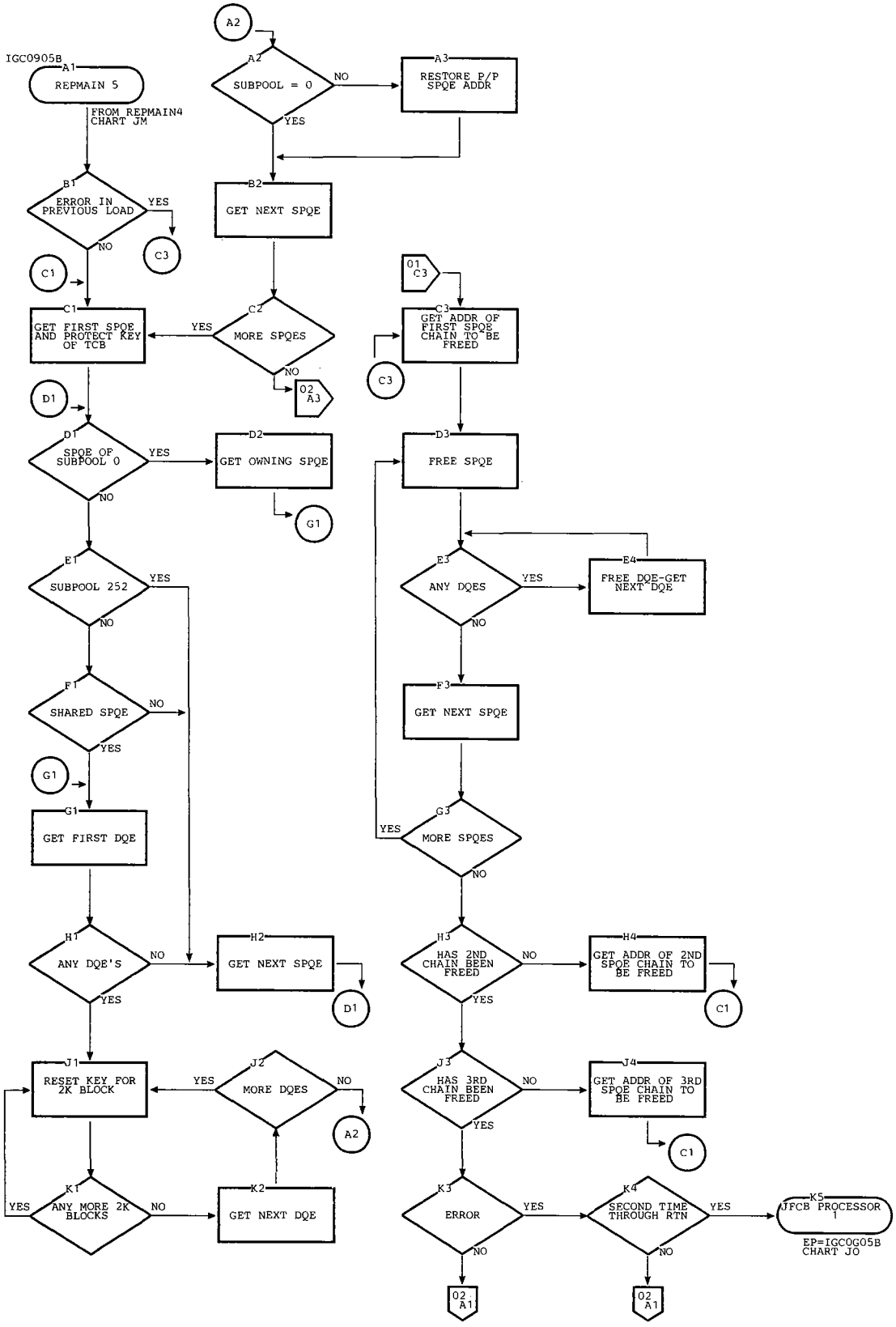


Chart JN. Restart Repmain 5 Routine (Page 2 of 2)

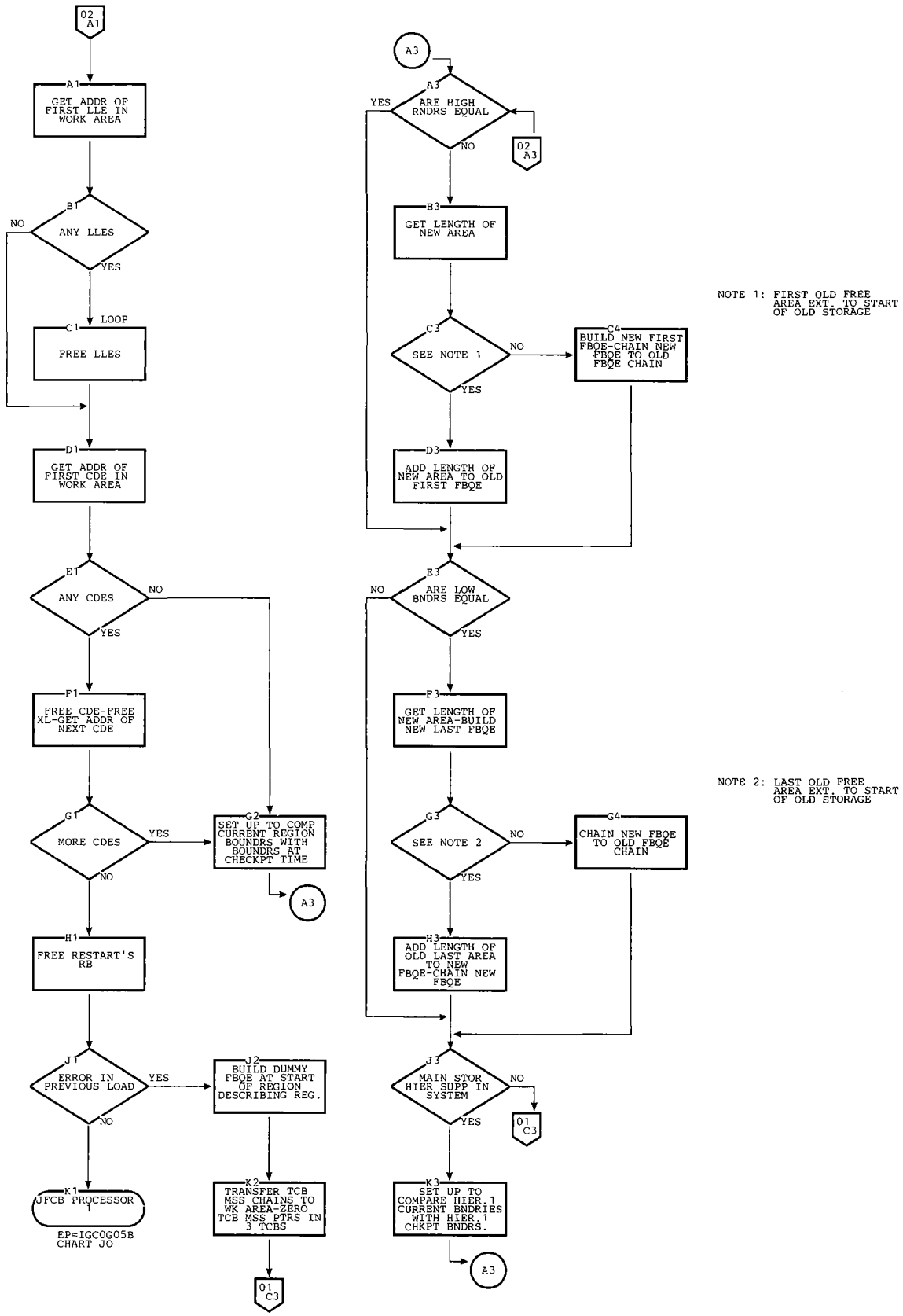


Chart JO. Restart JFCB Processors 1, 1A, and 2

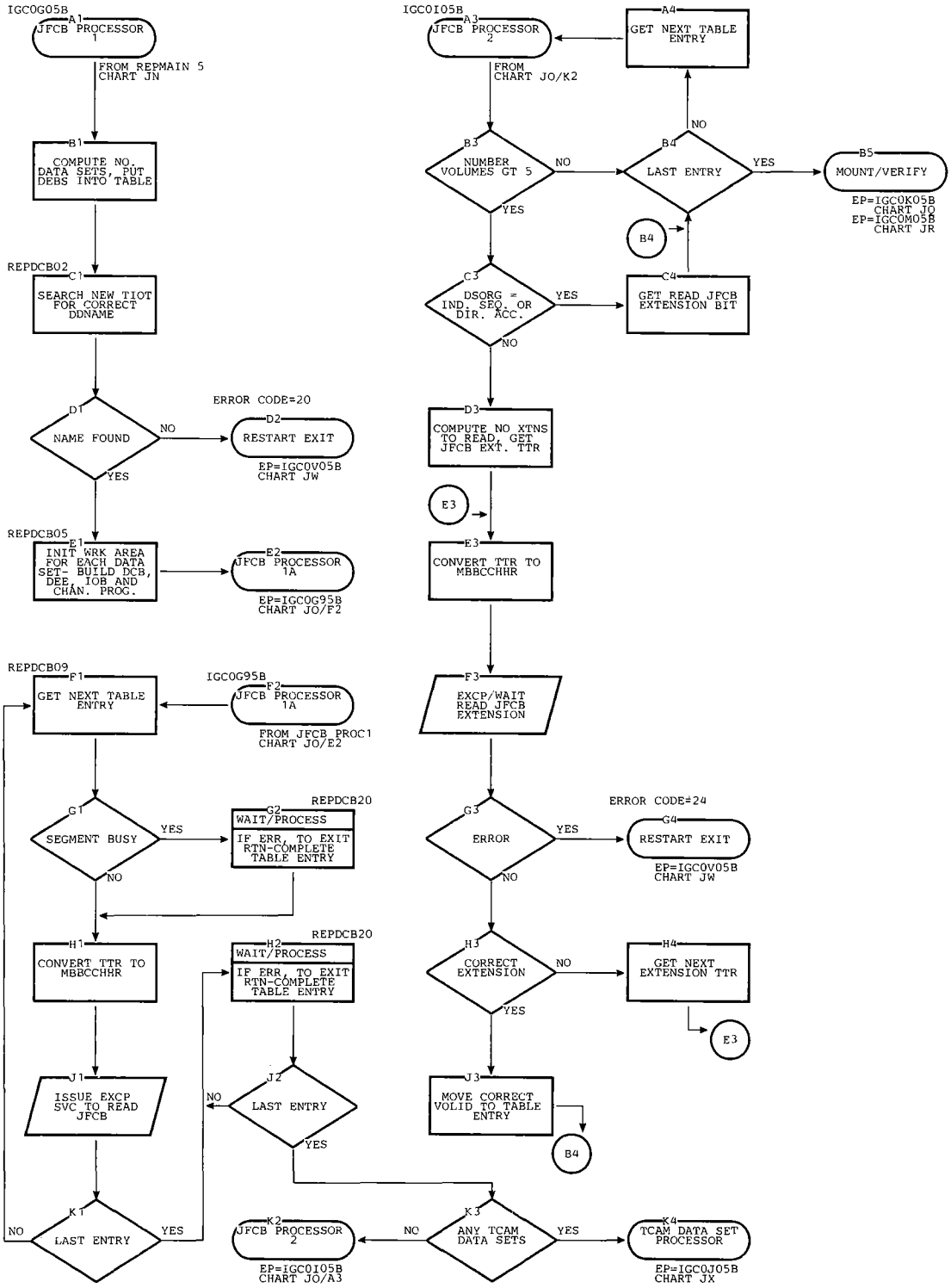


Chart JP. Restart Dummy Data Set Processor

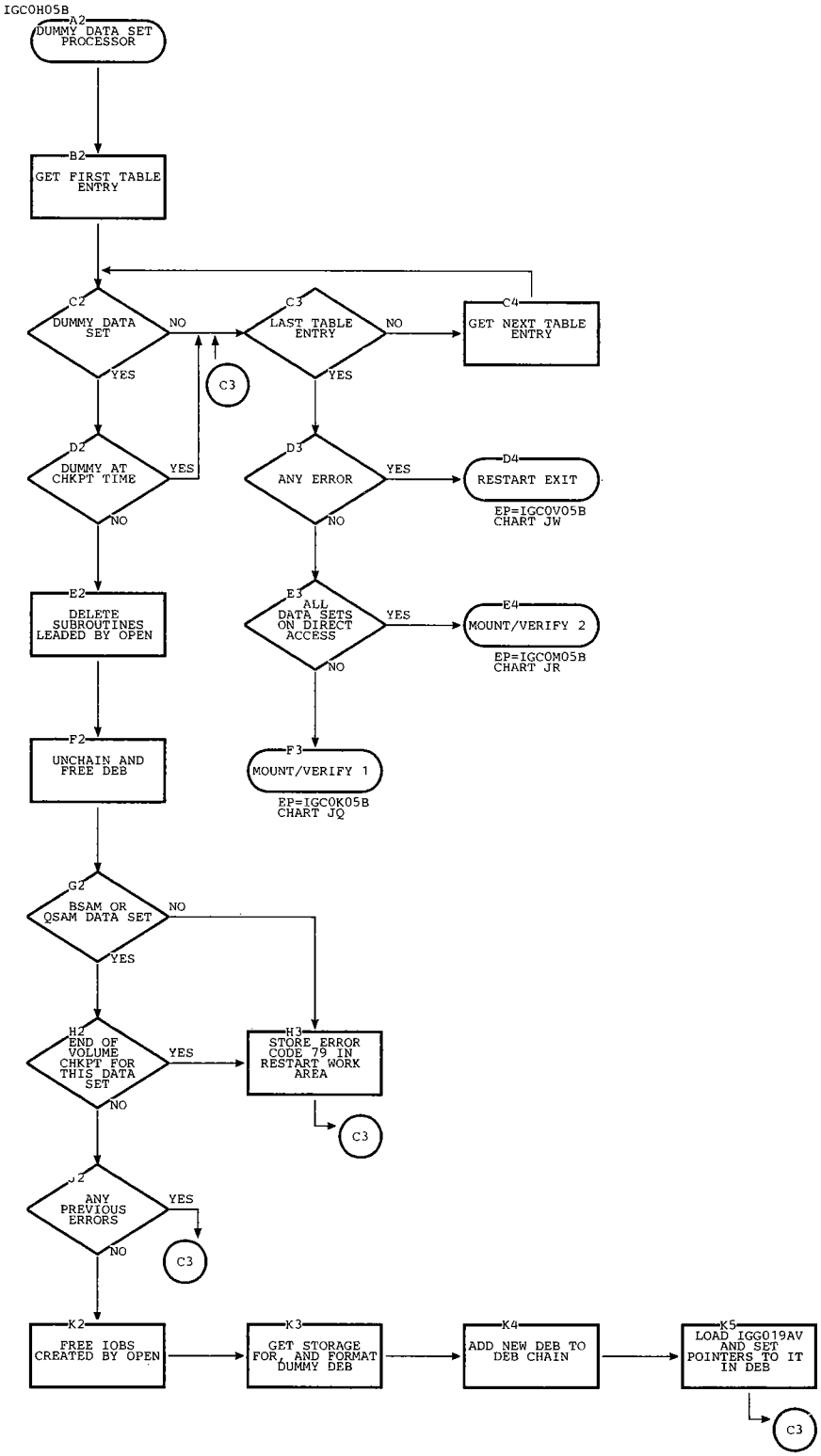


Chart JQ. Restart Mount Verify 1 Routine (Non-Direct Access)

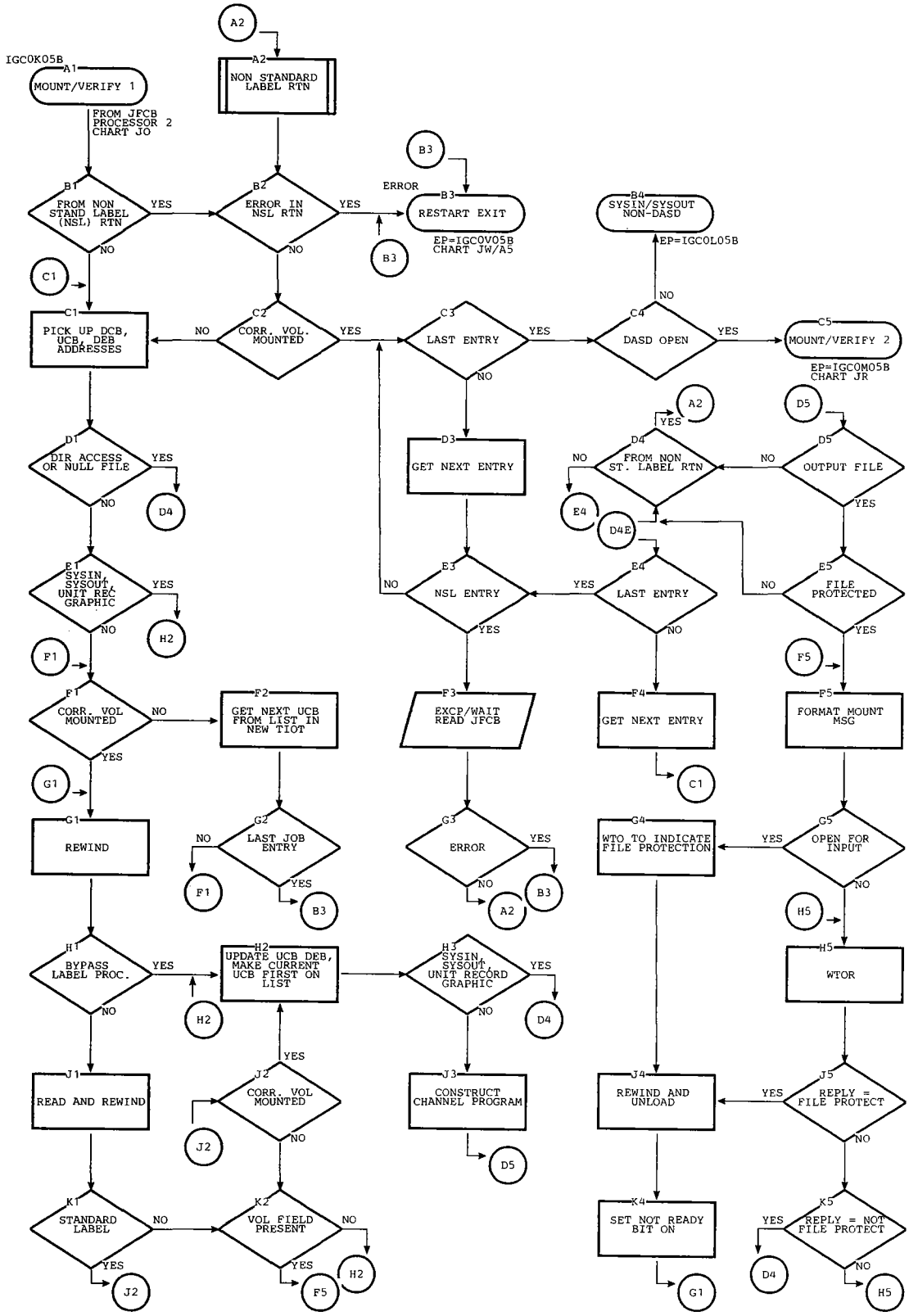


Chart JR. Restart Mount Verify 2 Routine (Direct Access)

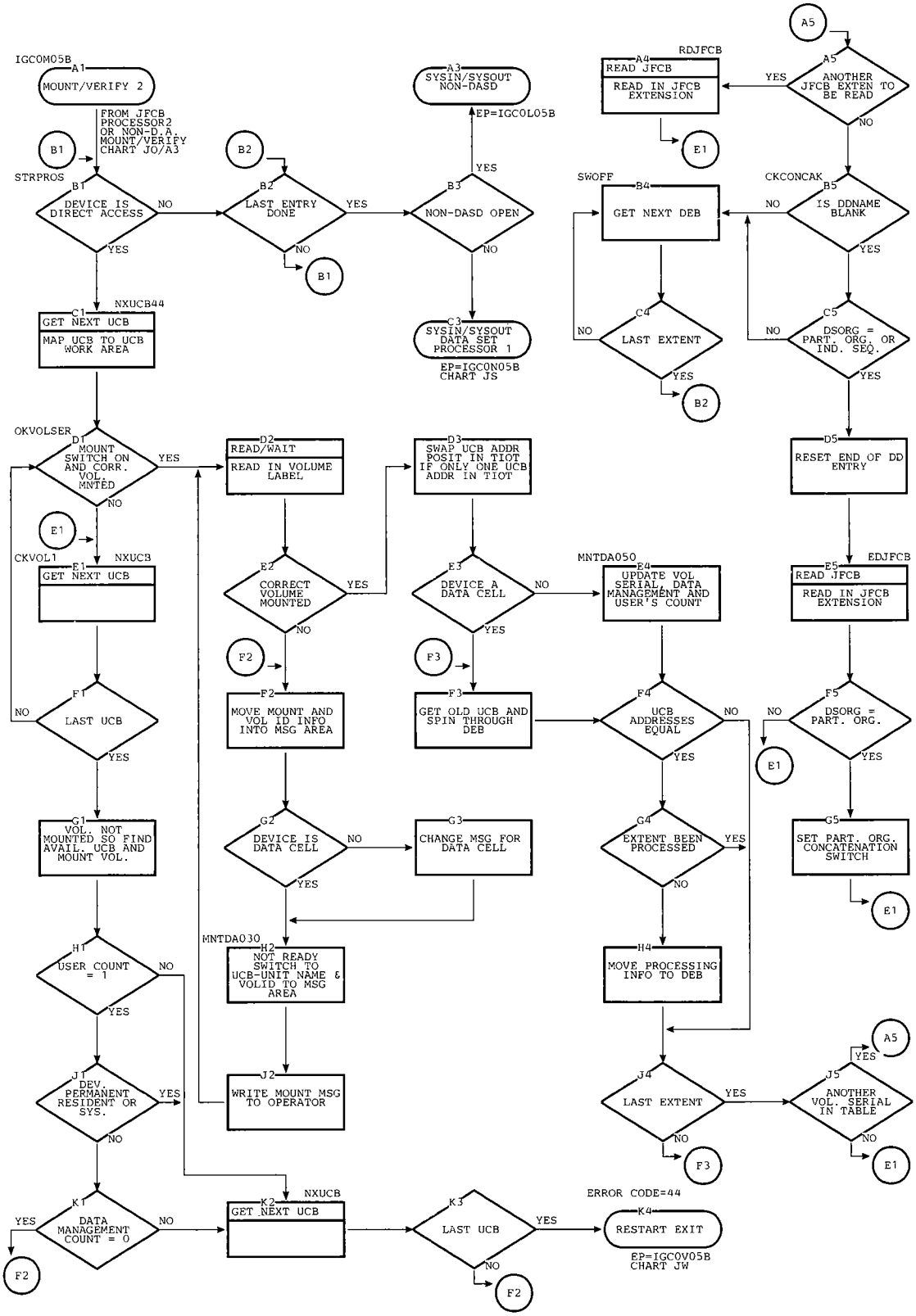


Chart JS. Restart SYSIN/SYSOUT Data Set Processors 1 and 2 (Non-Direct Access)

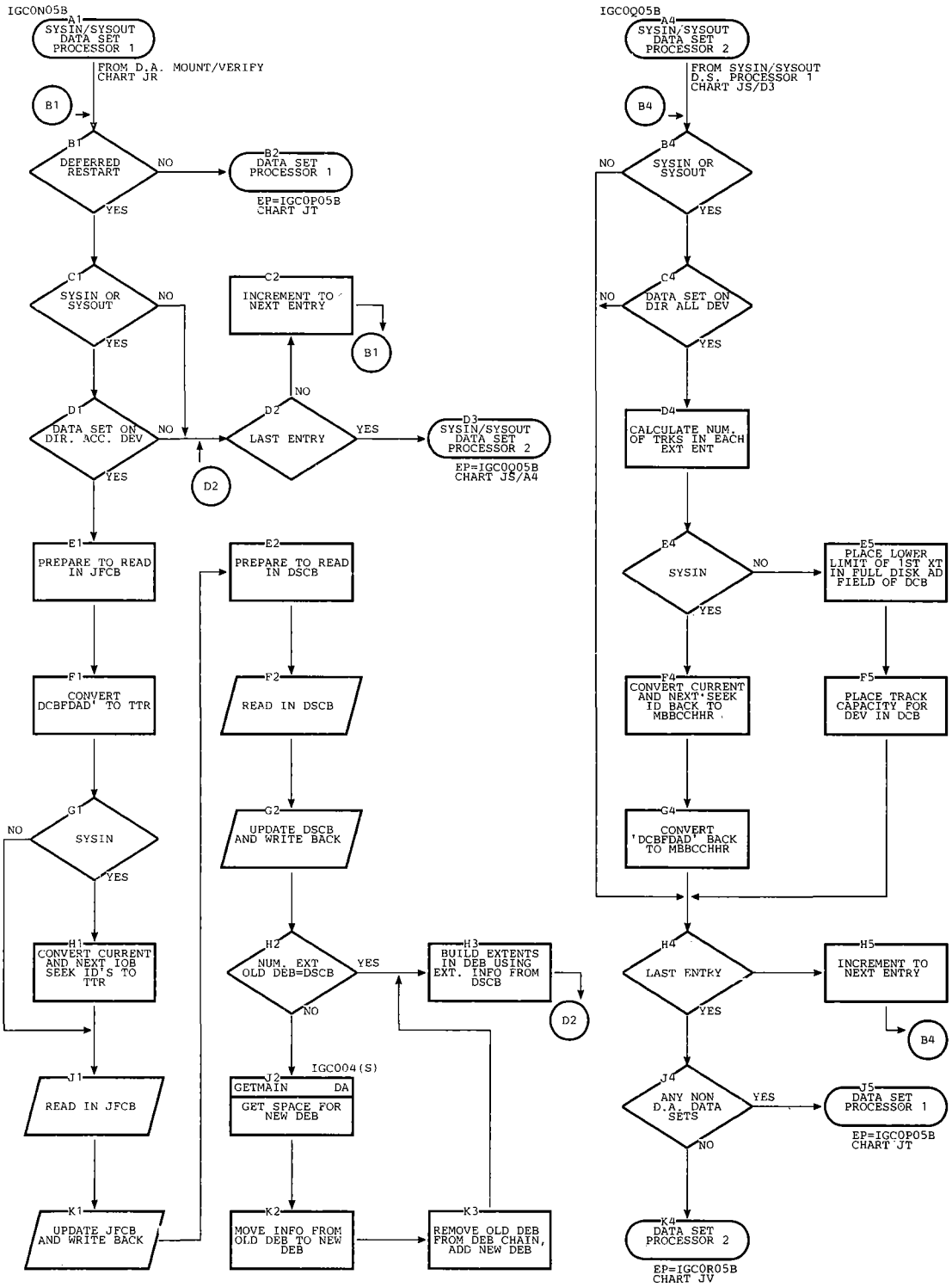


Chart JT. Restart Data Set Processor 1 (Non-Direct Access)

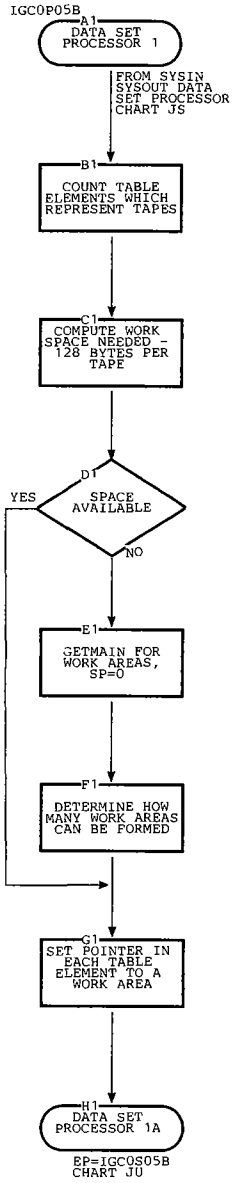


Chart JU. Restart Data Set Processor 1A -- Non-Direct Access (Page 1 of 2)

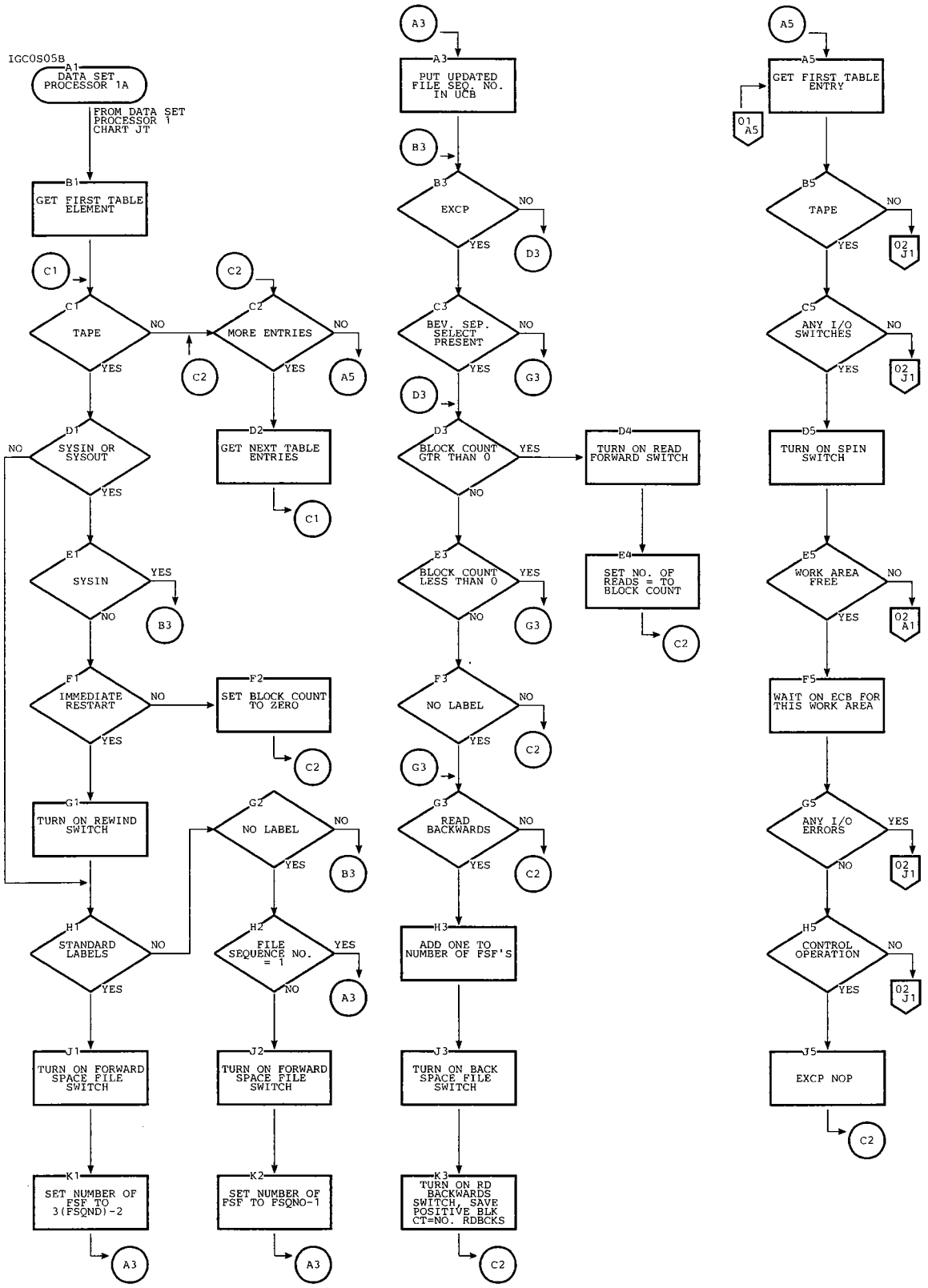


Chart JU. Restart Data Set Processor 1A -- Non-Direct Access (Page 2 of 2)

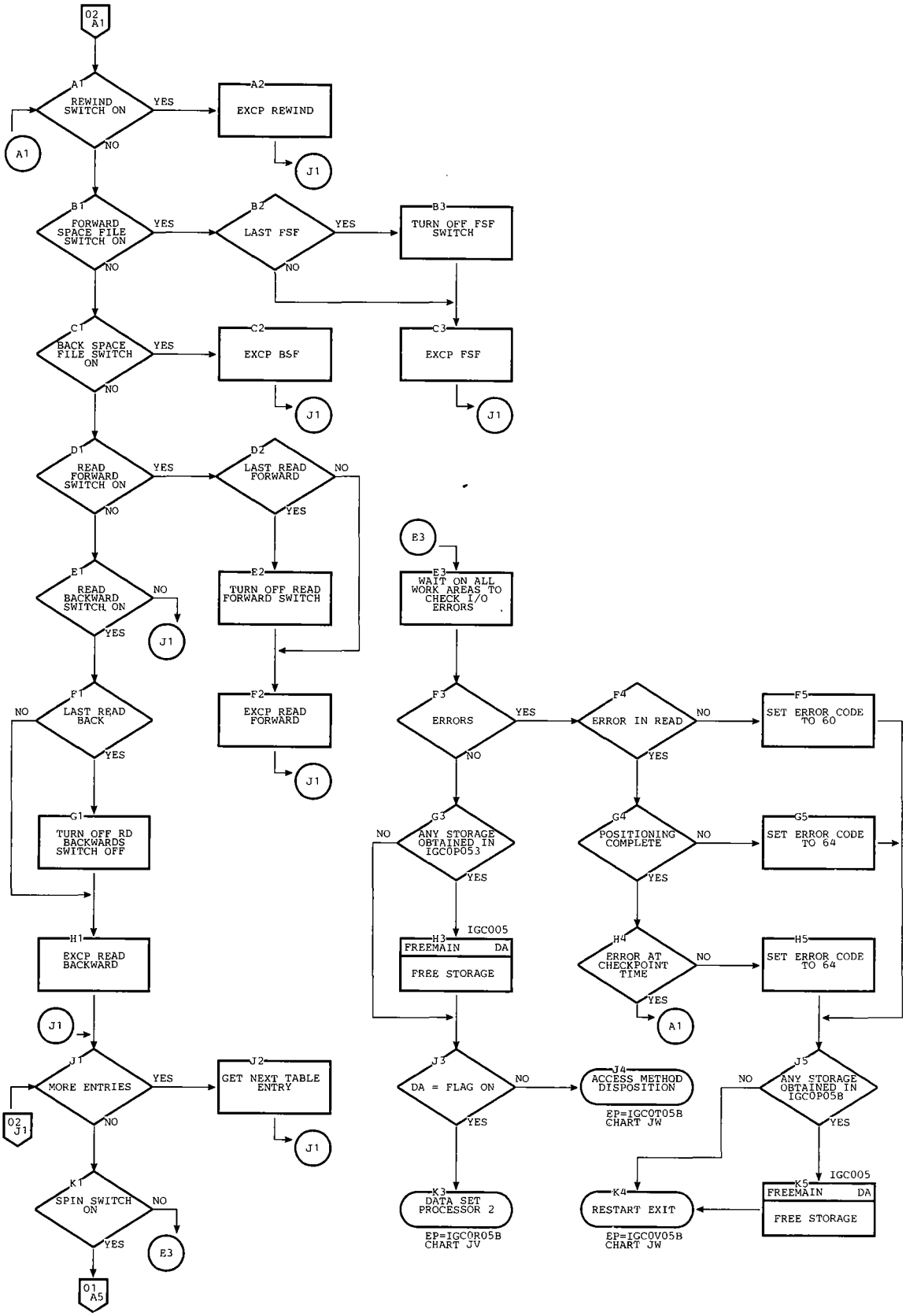


Chart JV. Restart Data Set Processor 2 (Direct Access)

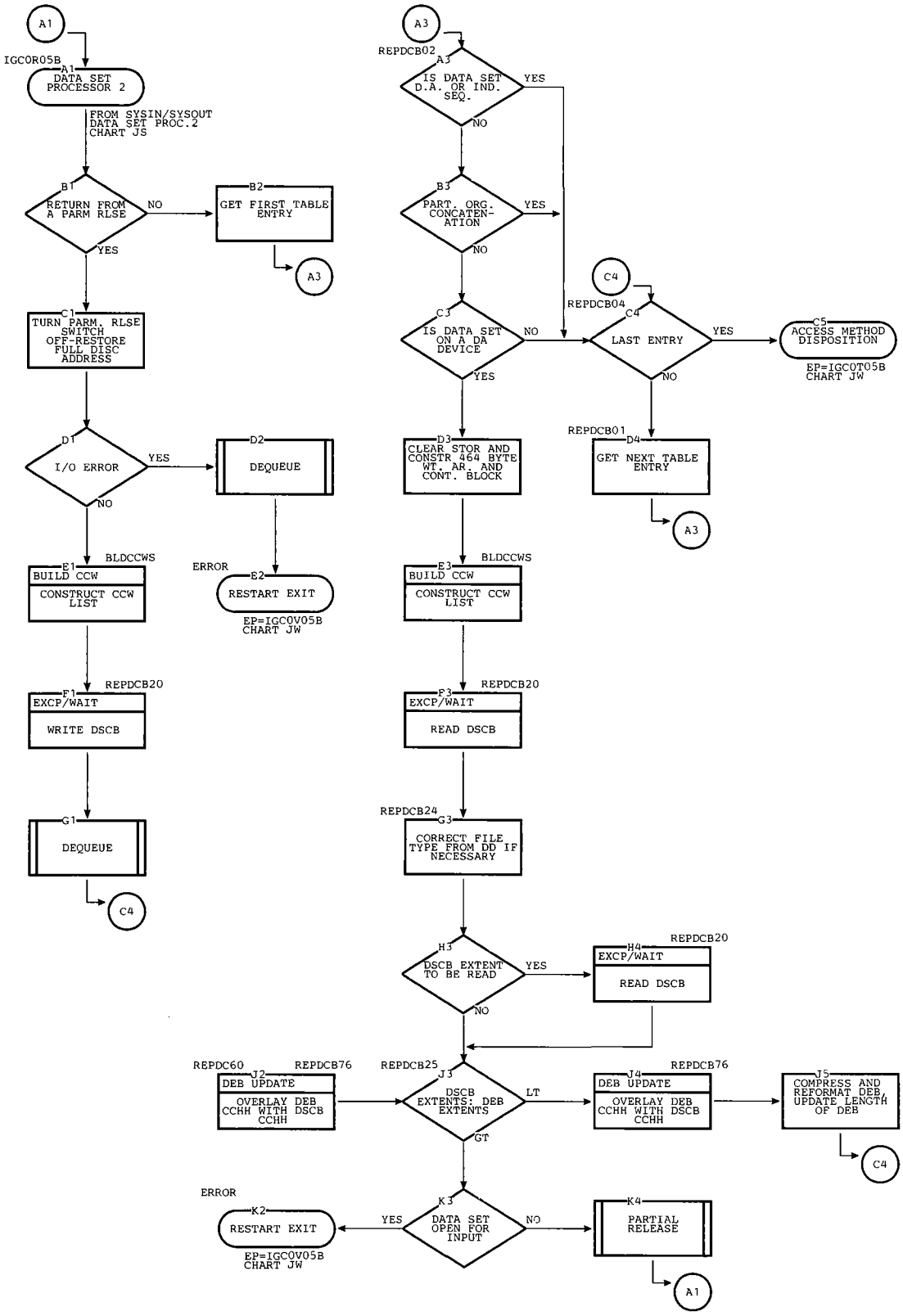


Chart JW. Restart Access Method Disposition and Exit

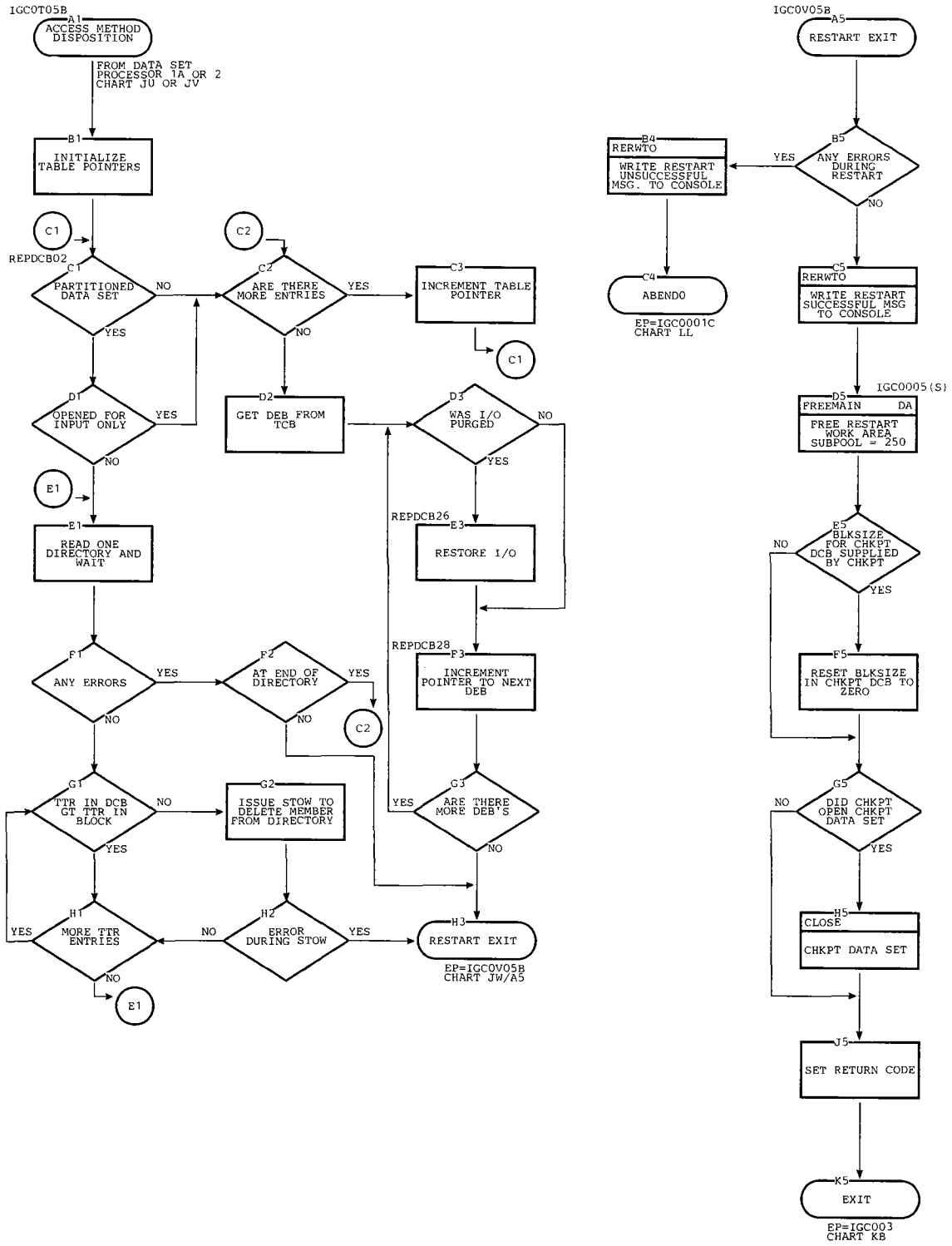


Chart JX. TCAM Data Set Processor (Page 1 of 2)

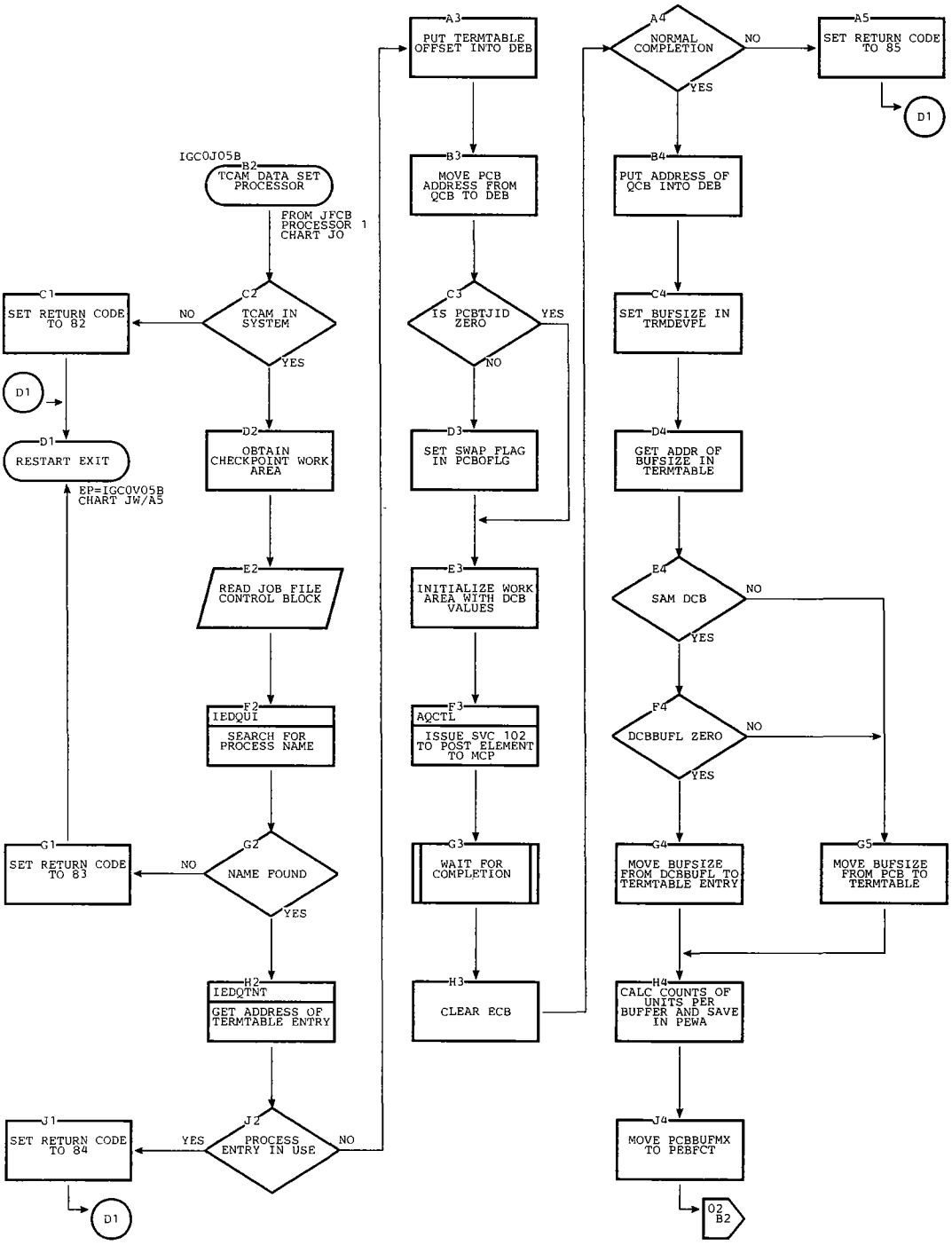


Chart JX. TCAM Data Set Processor (Page 2 of 2)

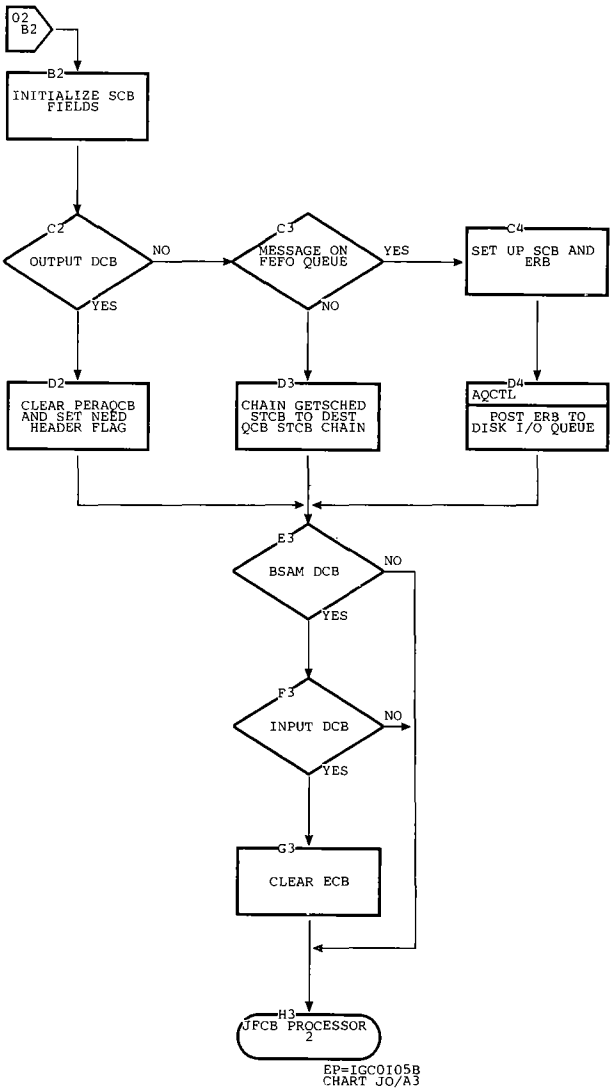


Chart JY. DOS Tape Data Set Processor (Page 1 of 2)

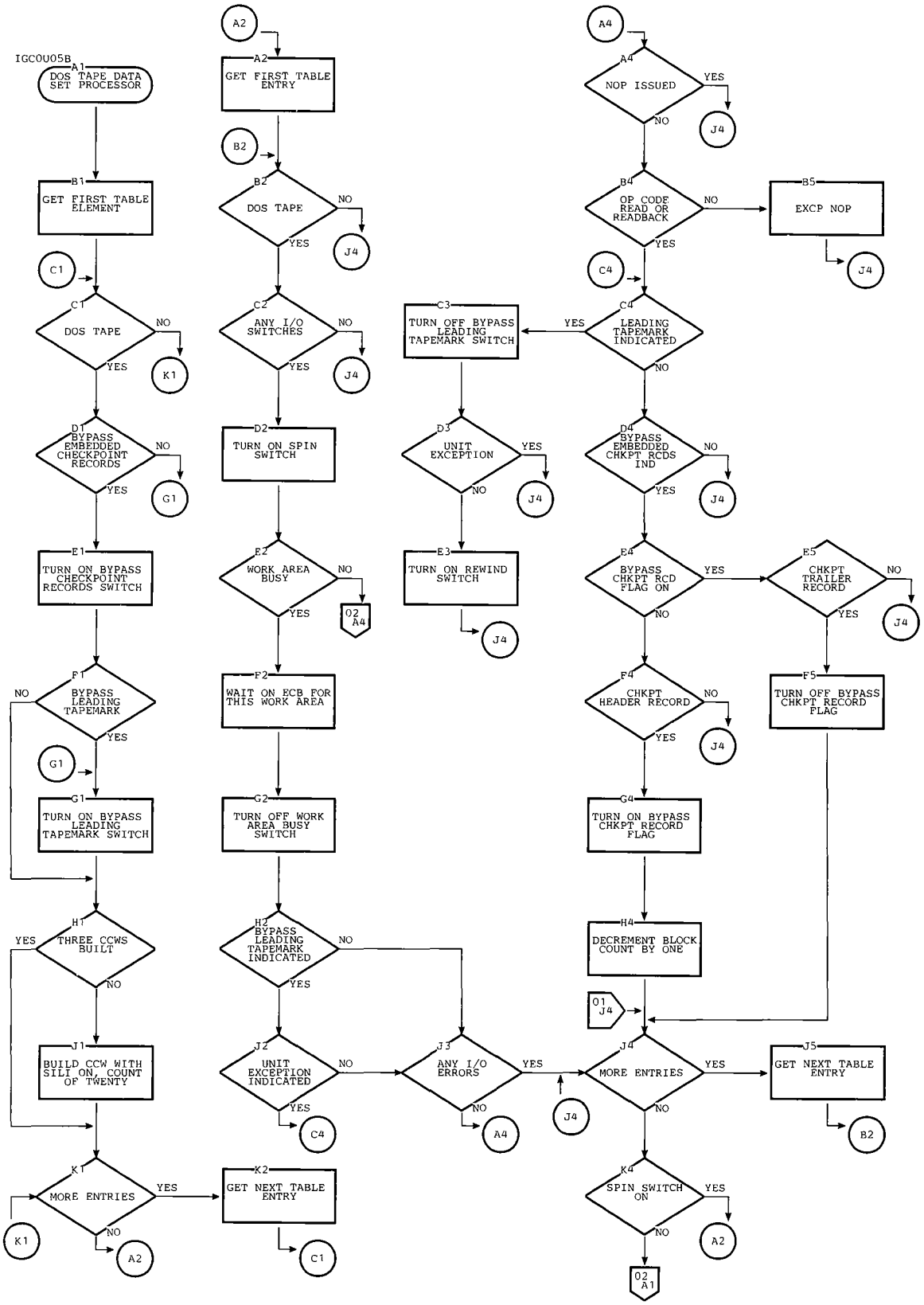


Chart JY. DOS Tape Data Set Processor (Page 2 of 2)

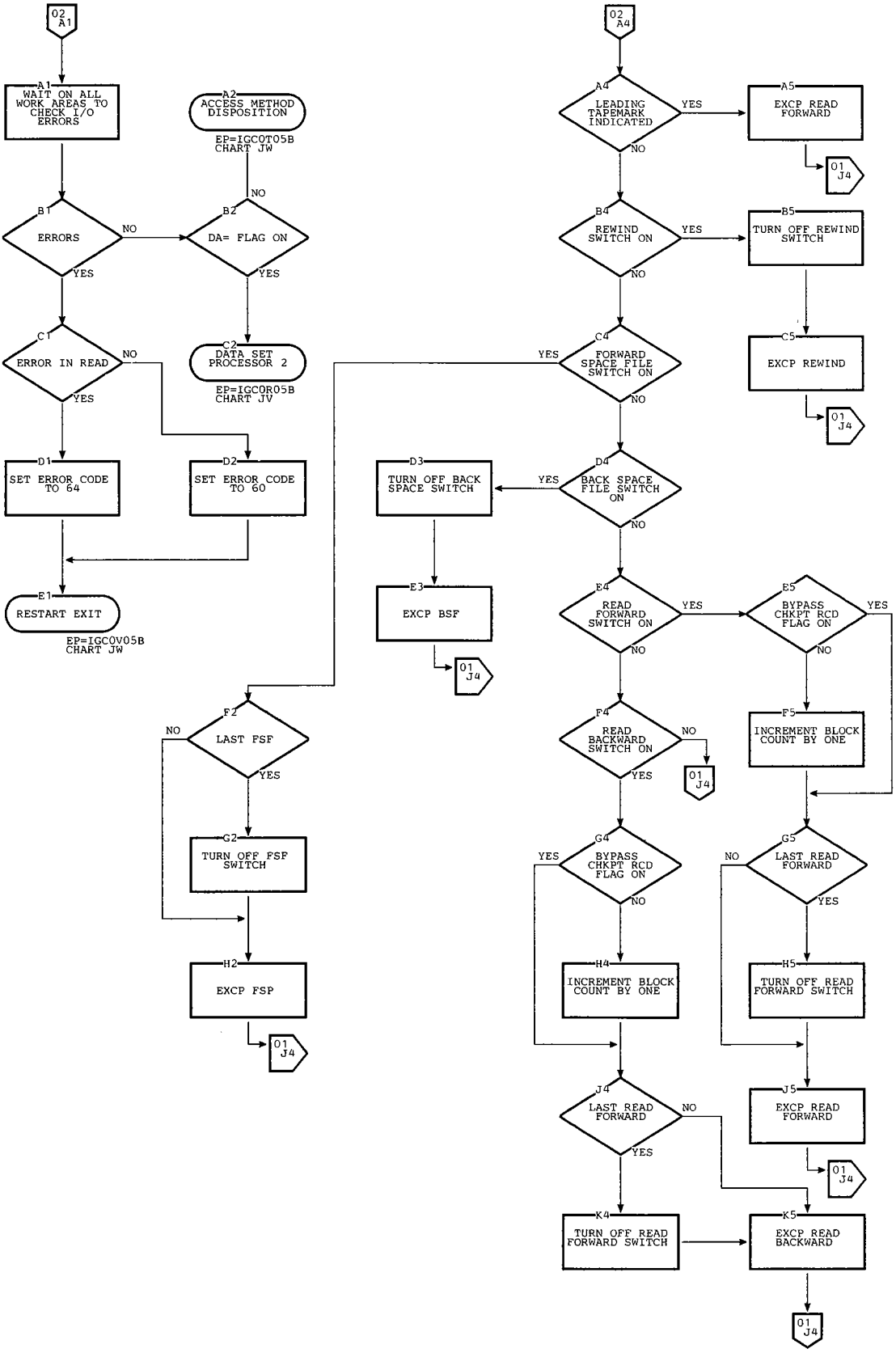


Chart KA. Type-1 SVC Exit

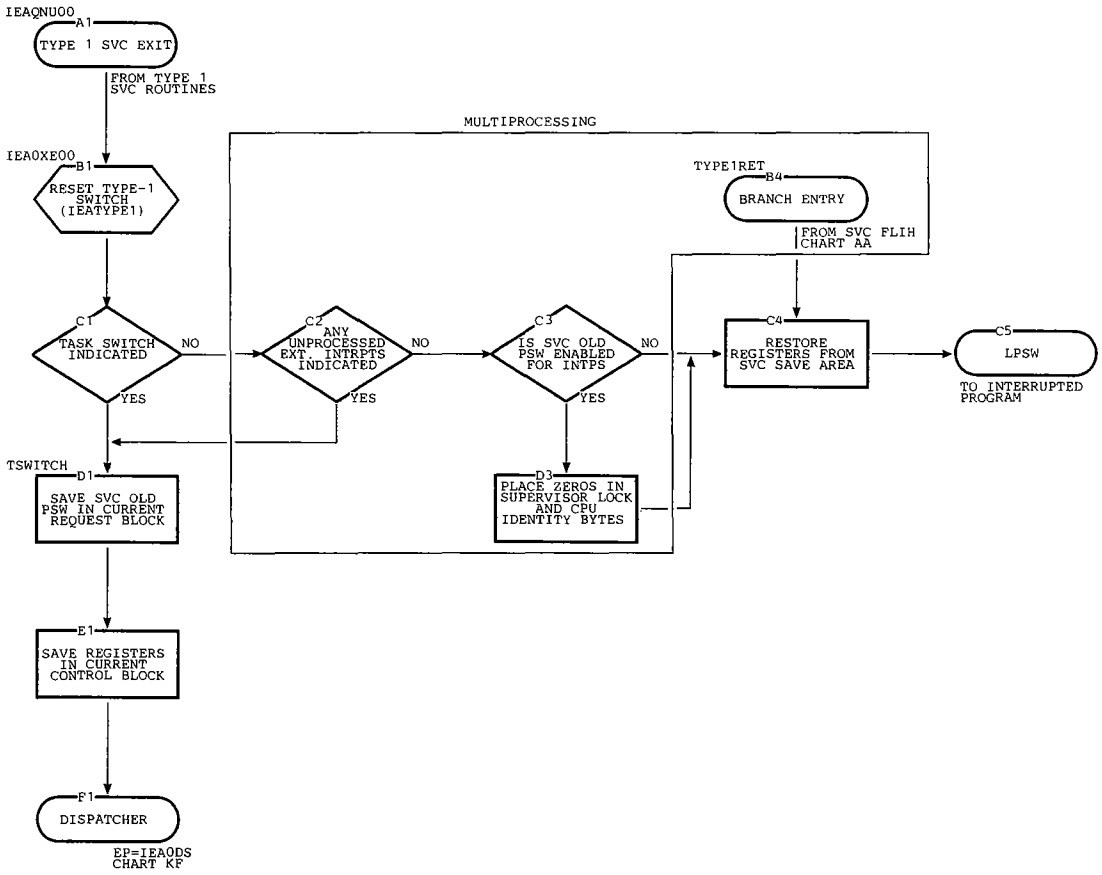


Chart KB. Exit Routine (Page 1 of 3)

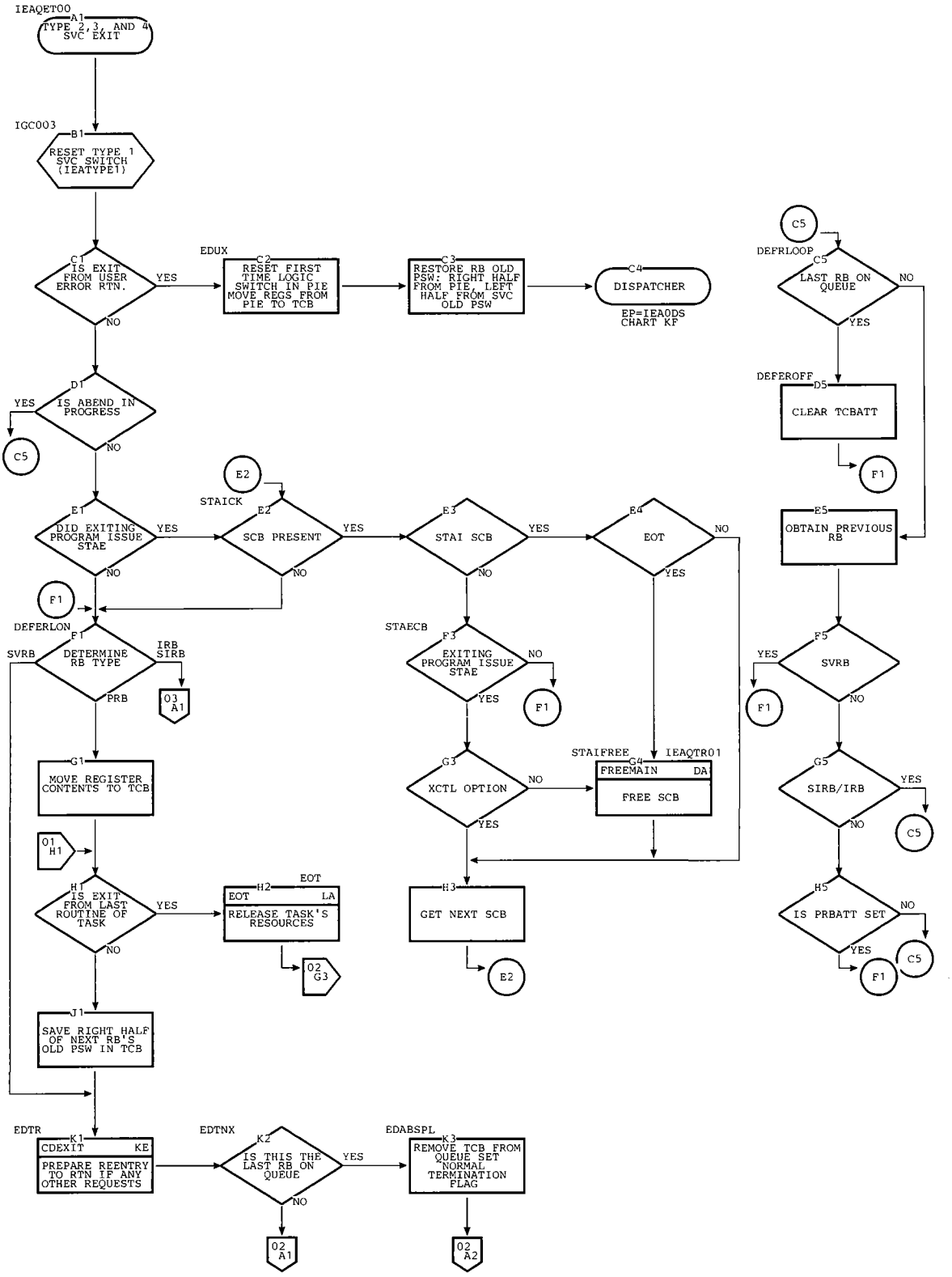


Chart KB. Exit Routine (Page 2 of 3)

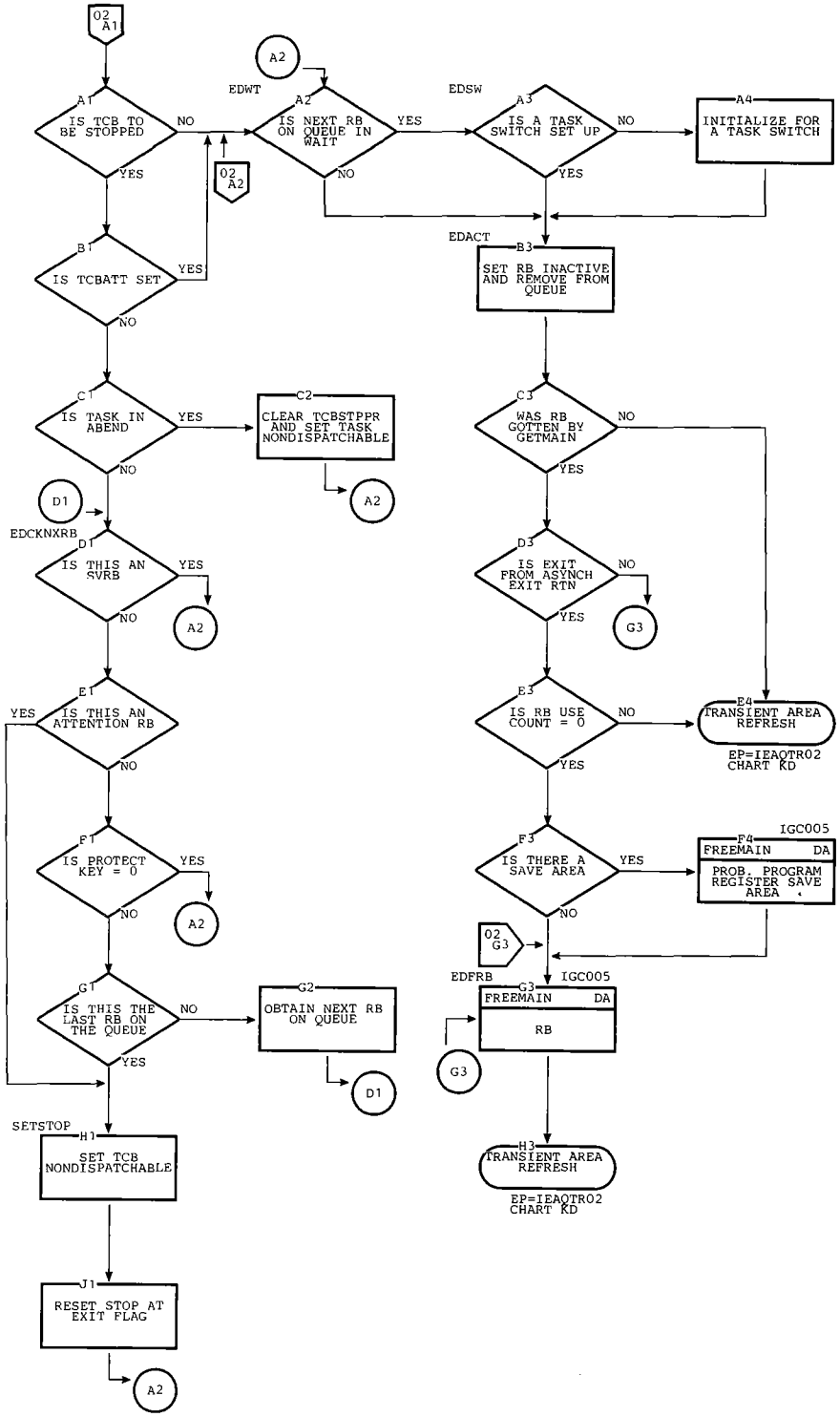


Chart KB. Exit Routine (Page 3 of 3)

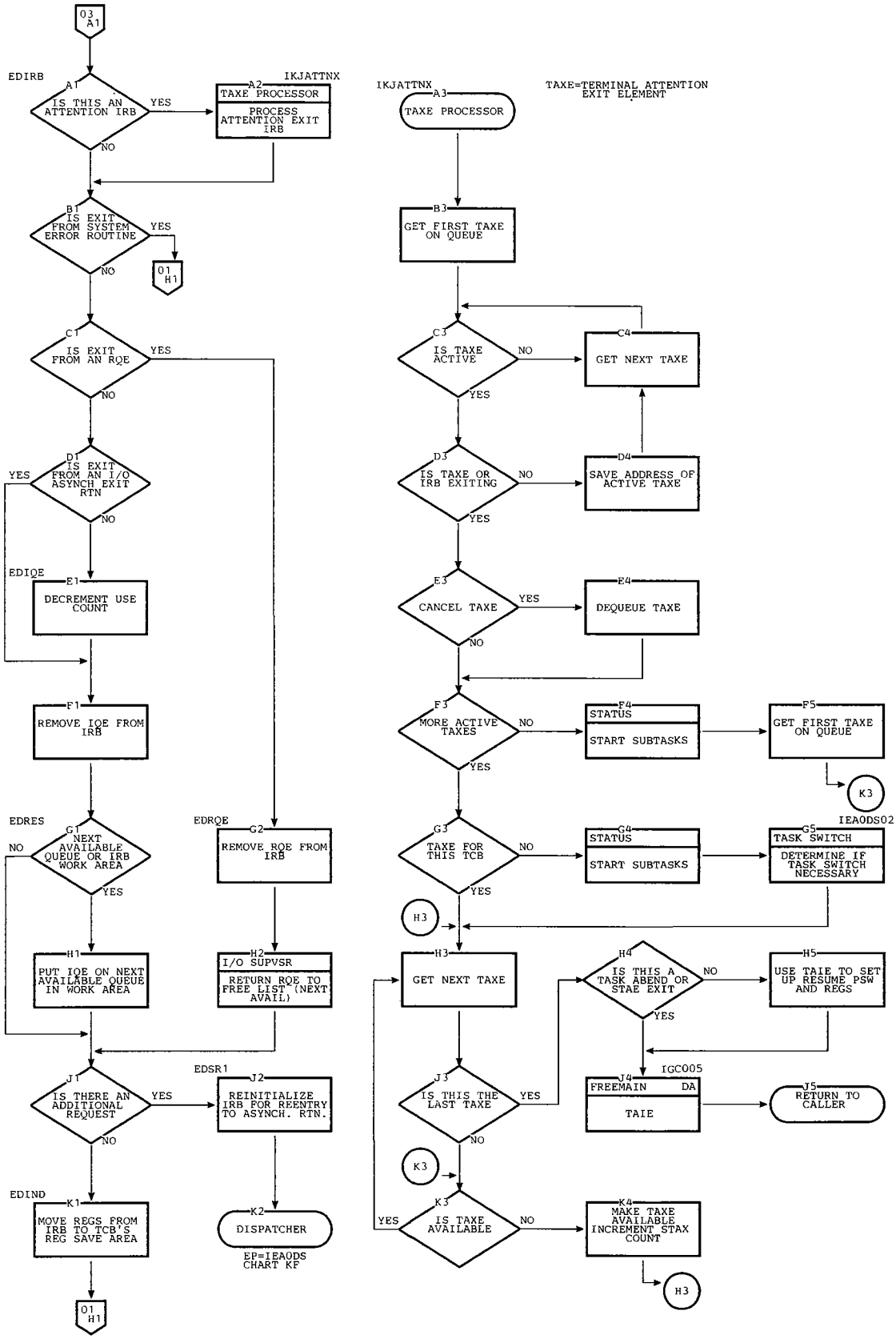


Chart KC. Transient Area Exit Routine

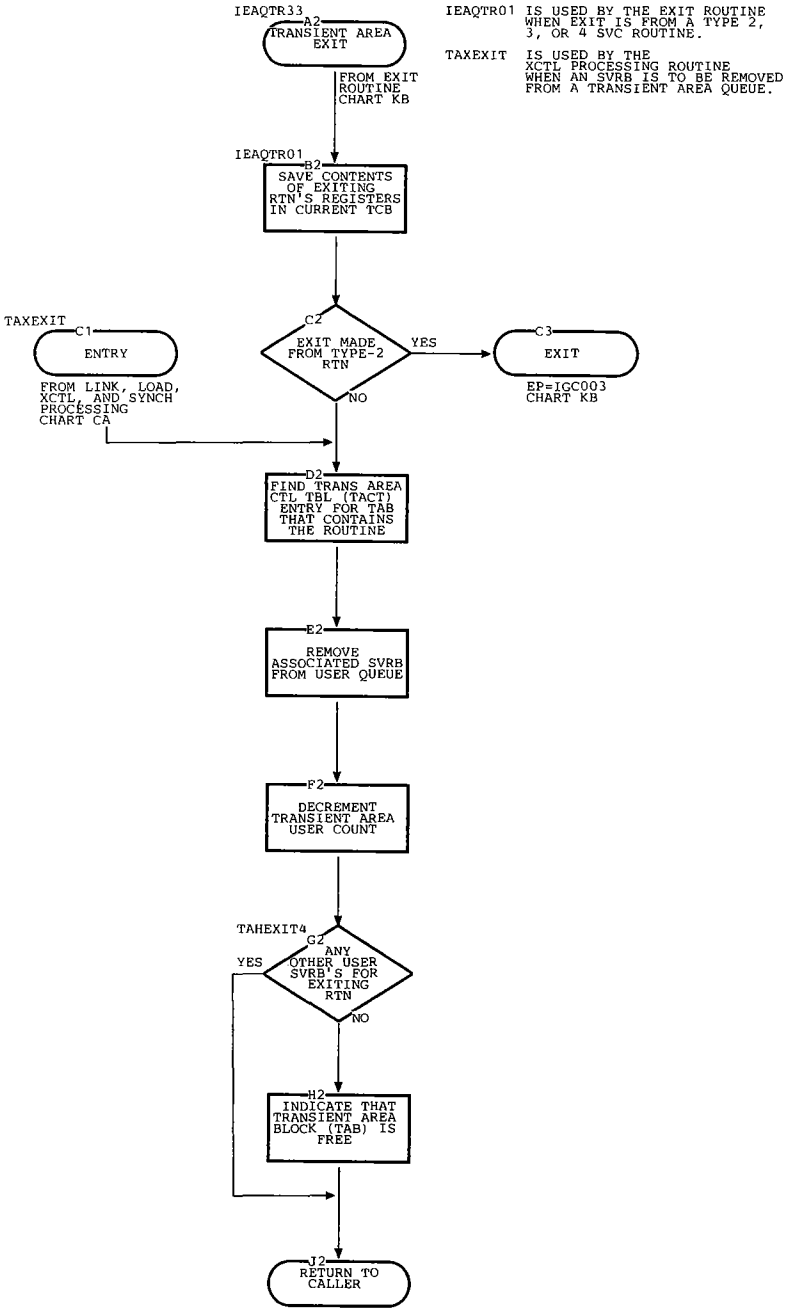


Chart KD. Transient Area Refresh Routine

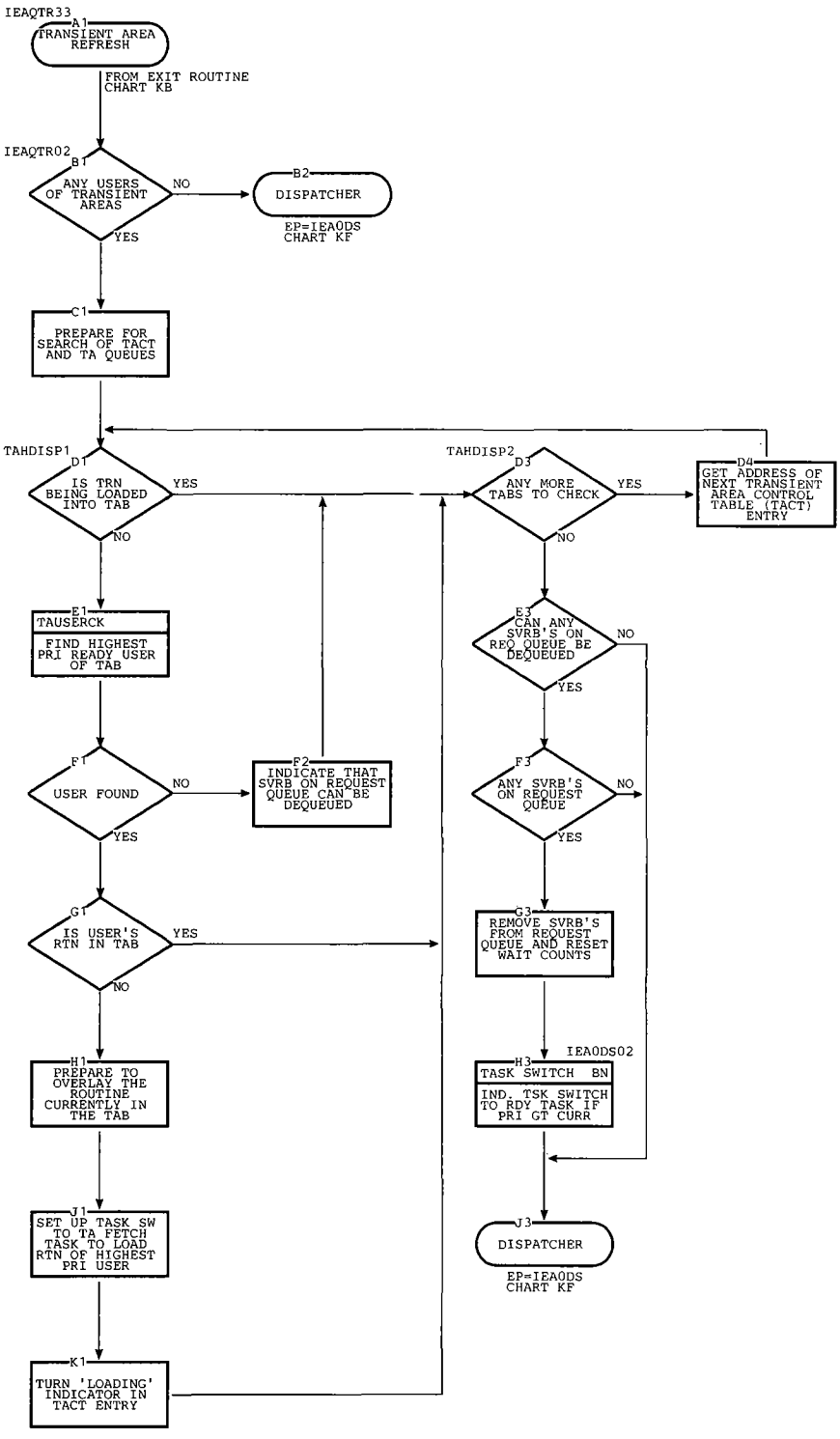


Chart KE. CDEXIT and CDDESTRY Subroutines

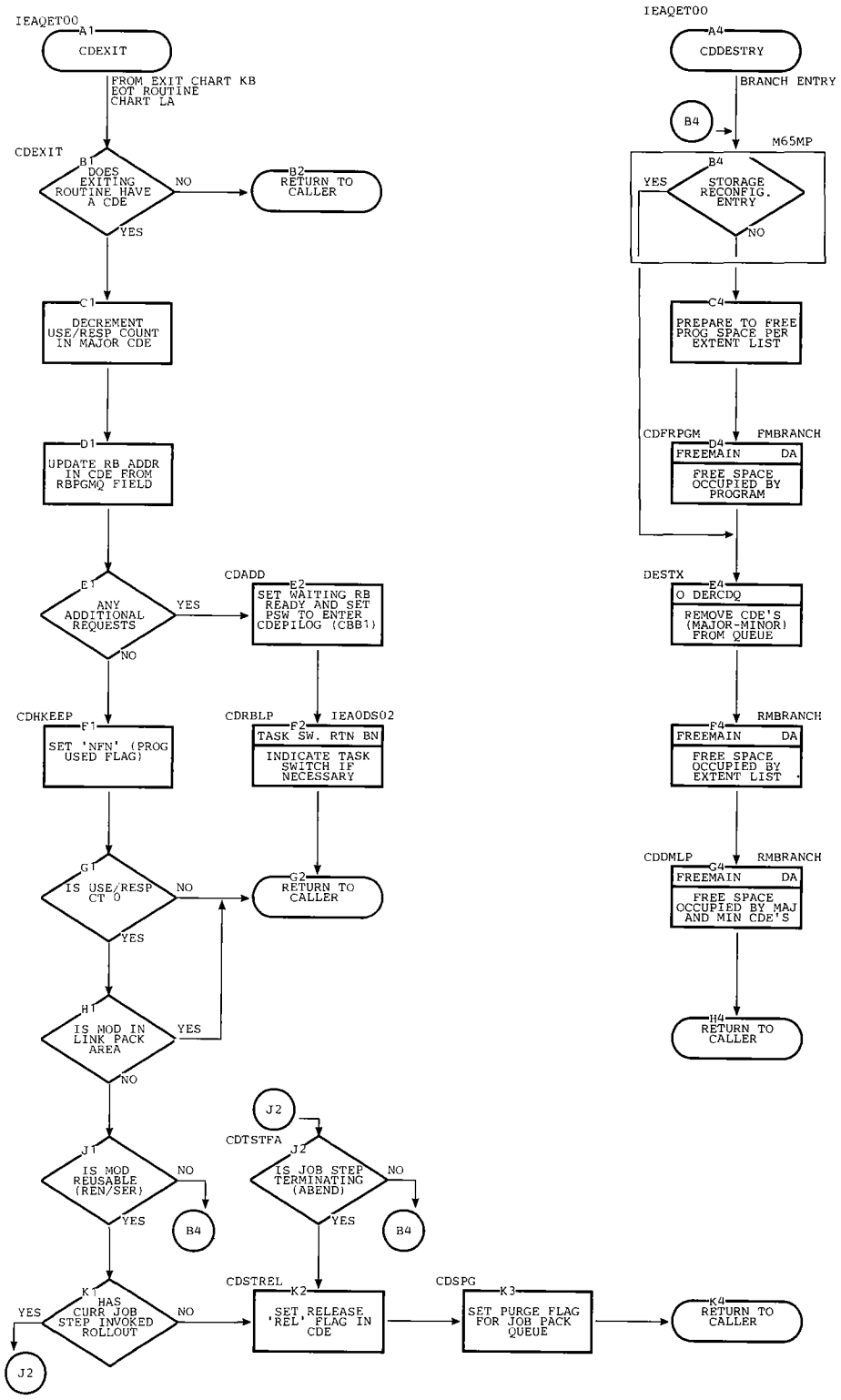


Chart KF. Dispatcher -- Uniprocessing

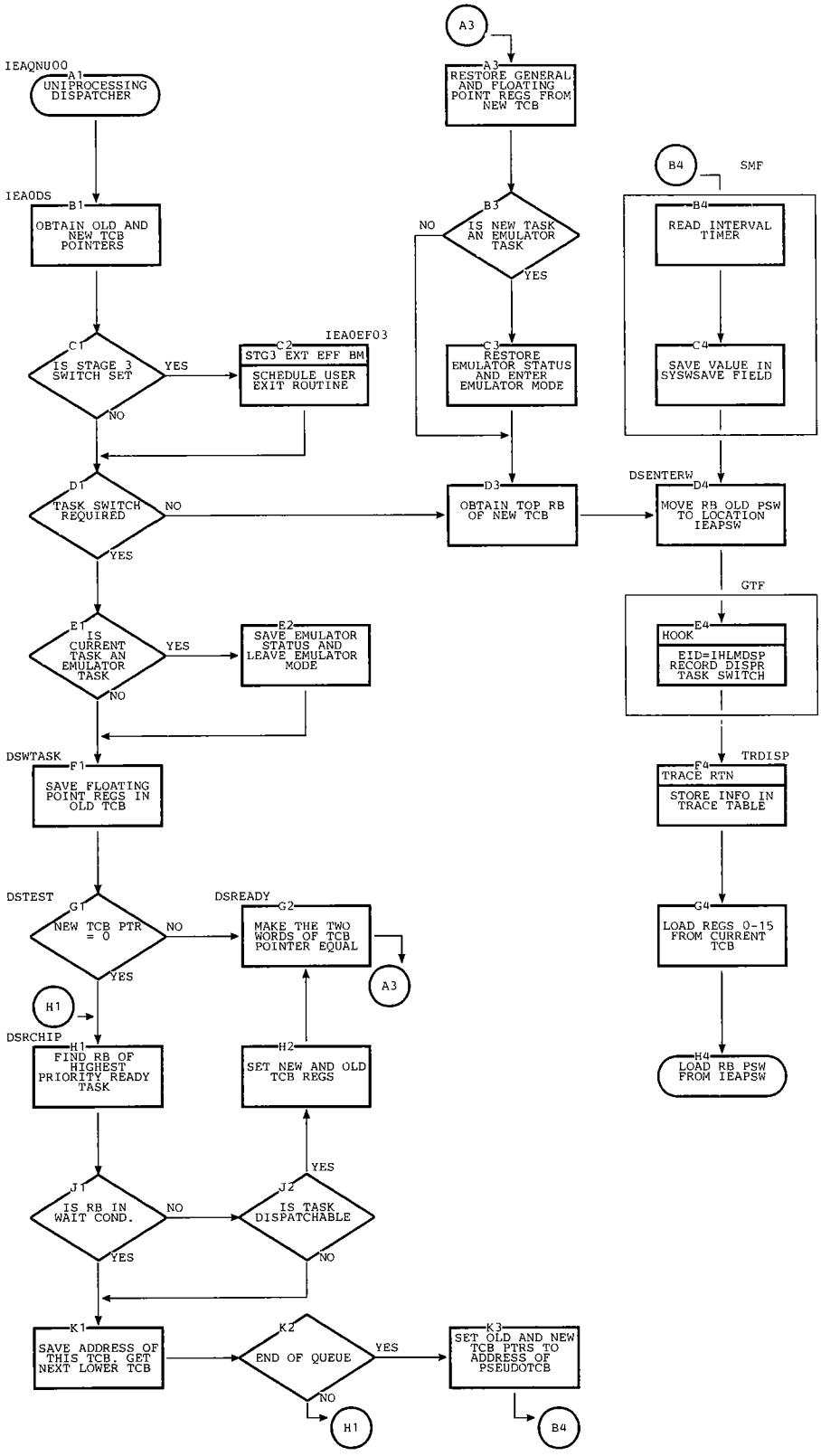


Chart KG. Dispatcher with Job Step and Task Timing (Page 1 of 2)

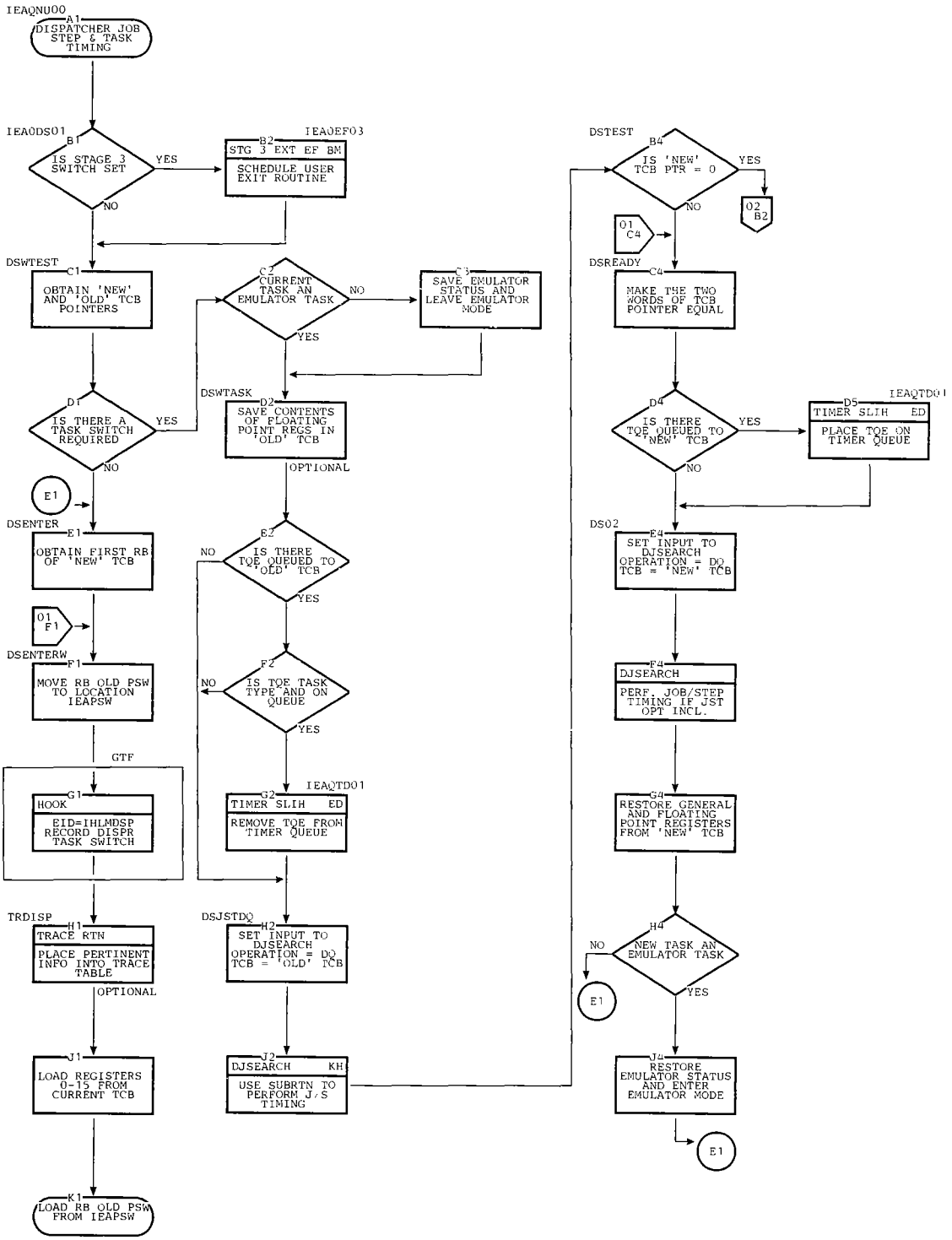


Chart KG. Dispatcher with Job Step and Task Timing (Page 2 of 2)

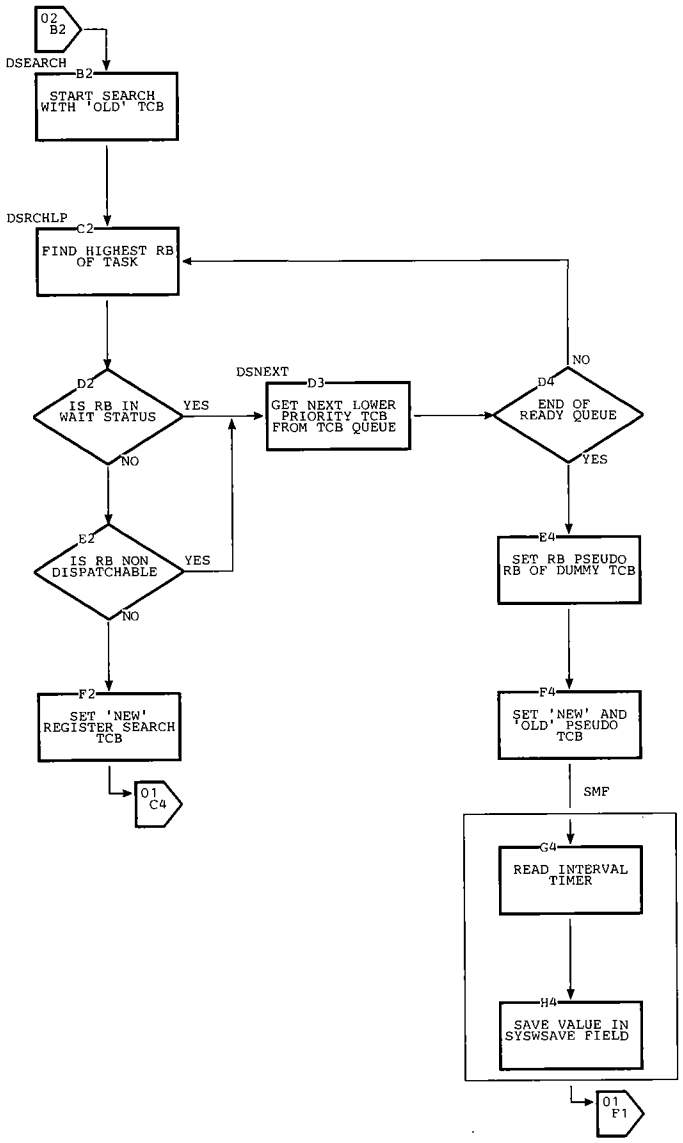


Chart KH. DJSEARCH Subroutine

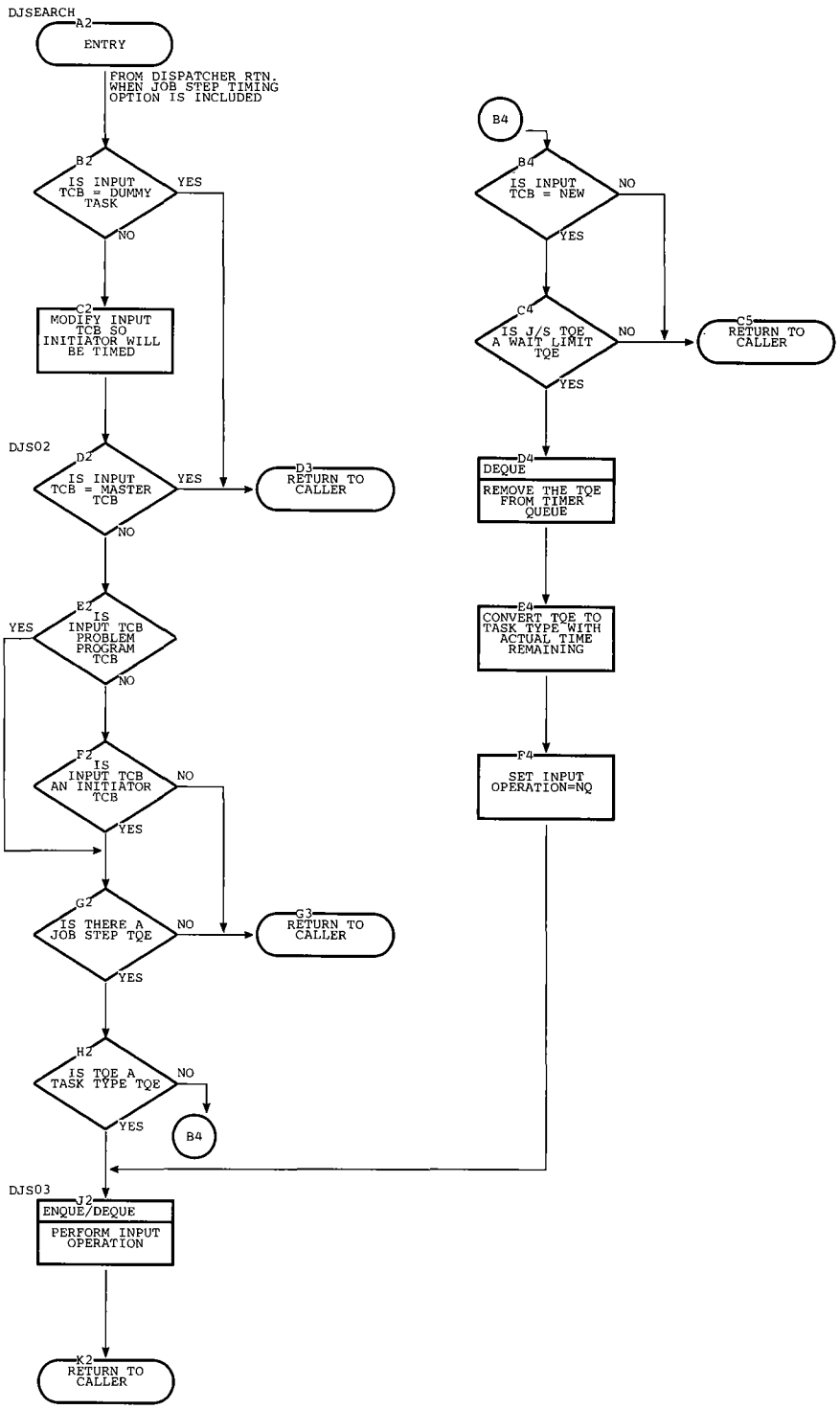


Chart KI. Dispatcher with Time Slicing (Page 1 of 3)

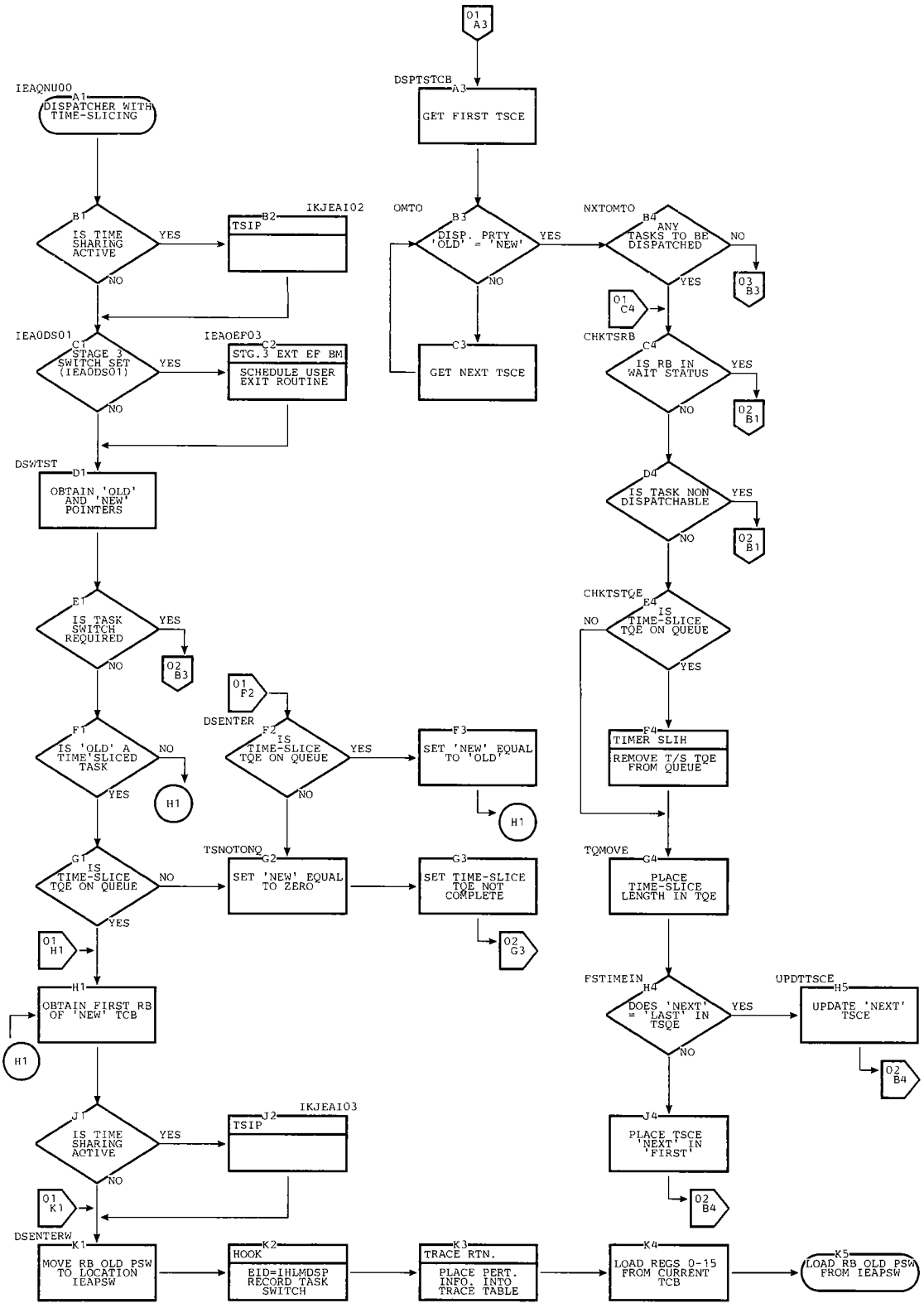


Chart KI. Dispatcher with Time Slicing (Page 2 of 3)

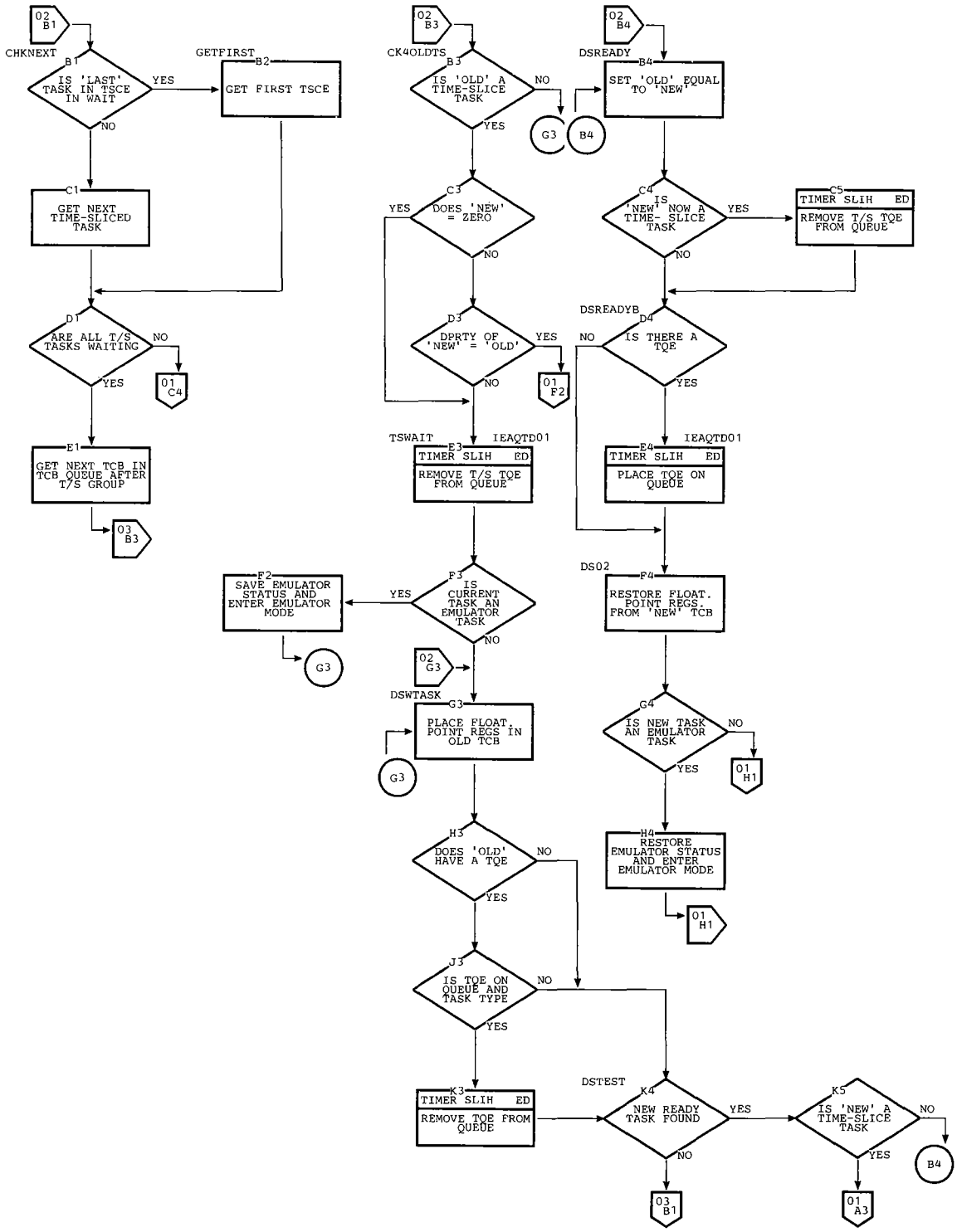


Chart KI. Dispatcher with Time Slicing (Page 3 of 3)

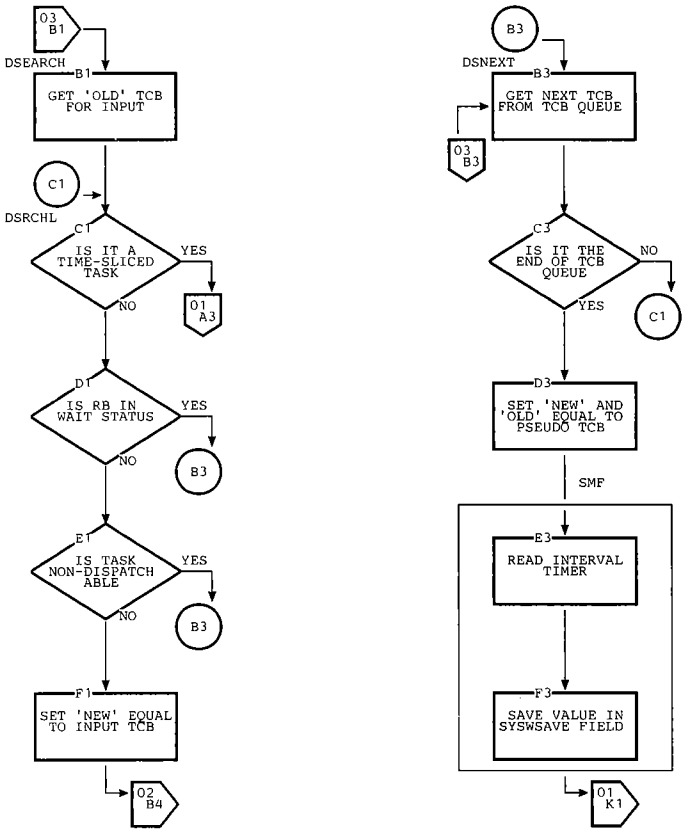


Chart KJ. Dispatcher with Multiprocessing (Page 1 of 2)

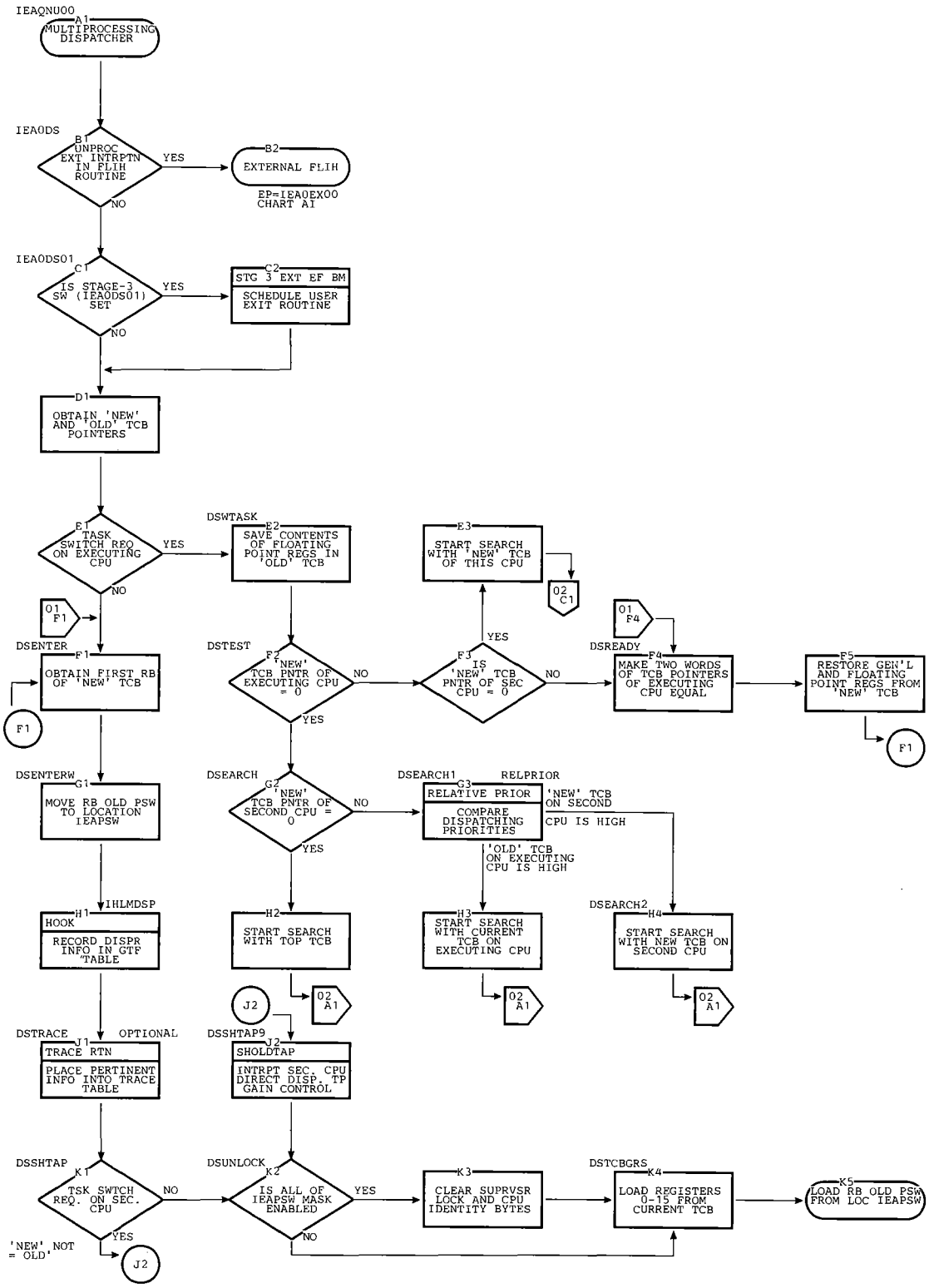


Chart KJ. Dispatcher with Multiprocessing (Page 2 of 2)

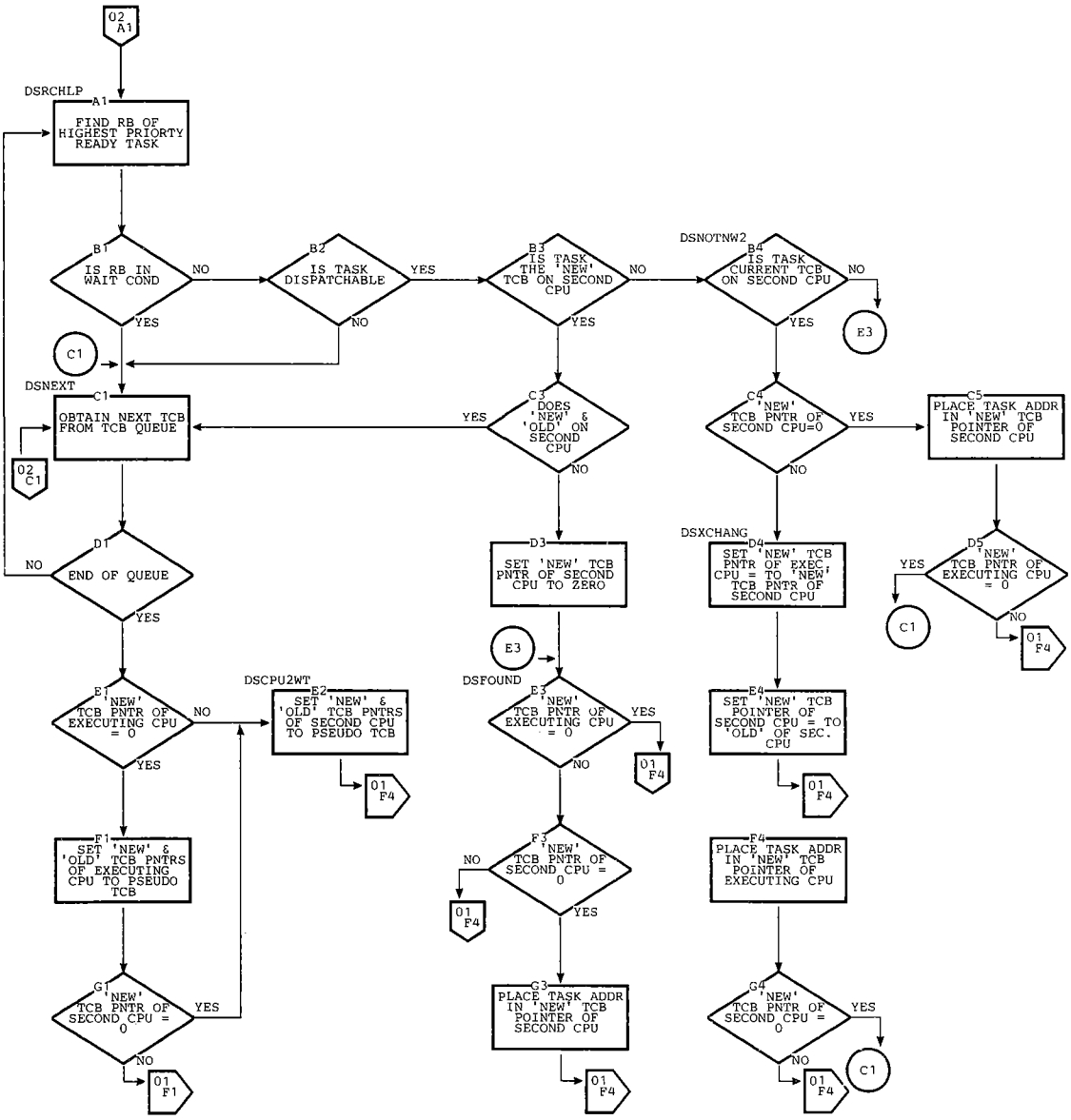


Chart KK. DJSEARCH Subroutine with Multiprocessing

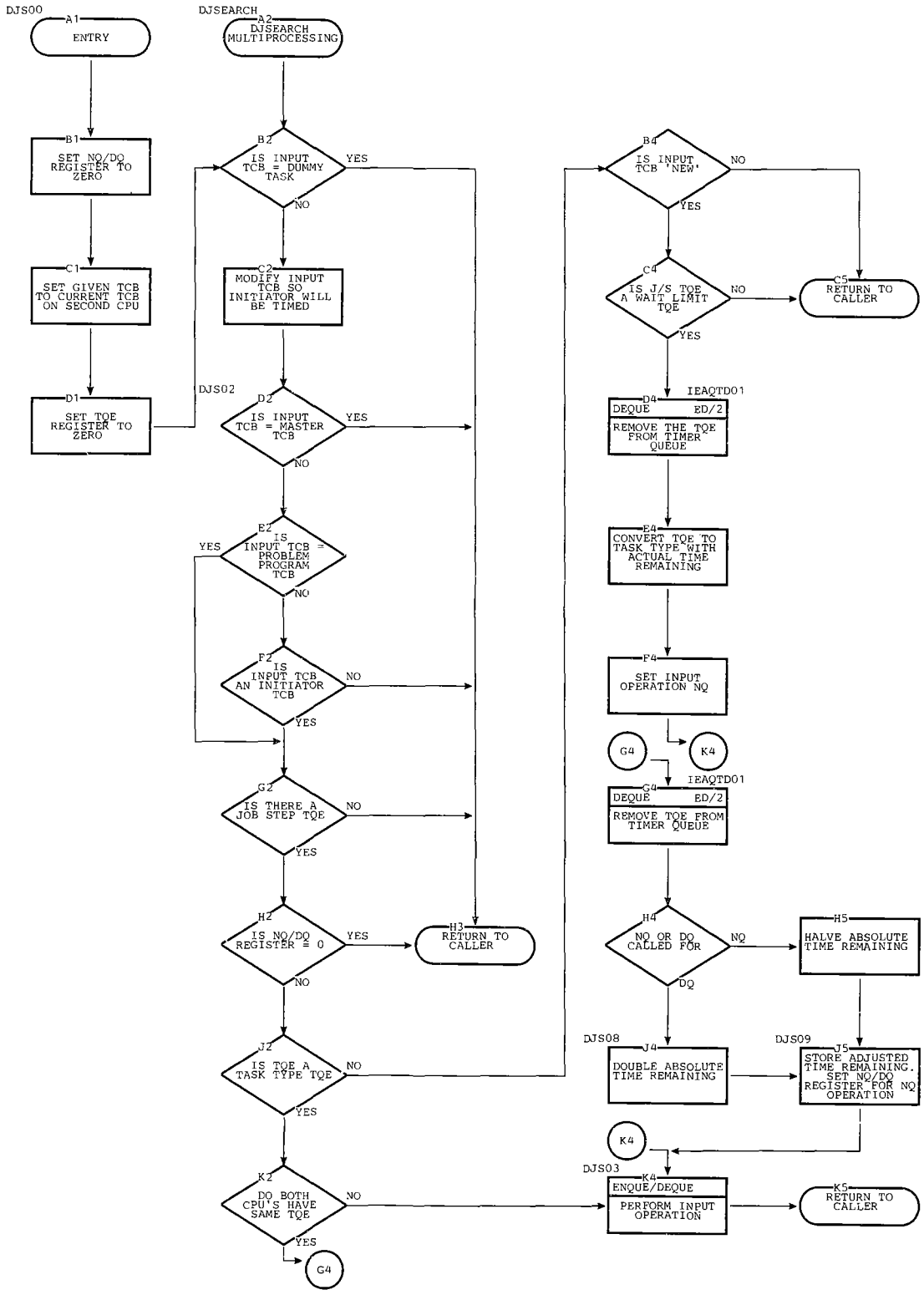


Chart KL. Dispatcher with Multiprocessing and Time Slicing (Page 1 of 9)

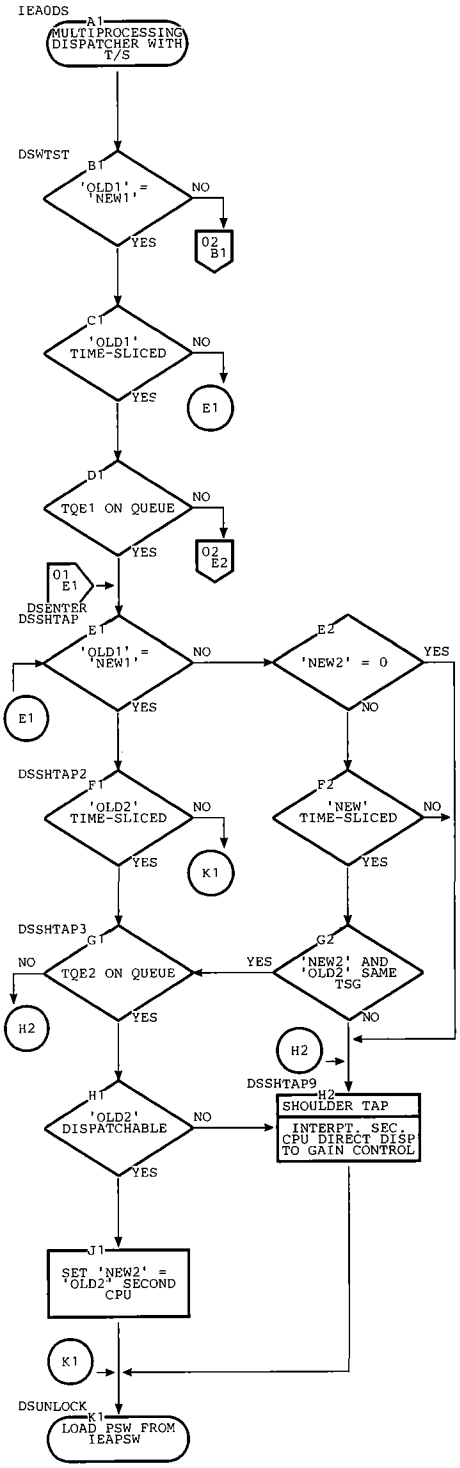


Chart KL. Dispatcher with Multiprocessing and Time Slicing (Page 2 of 9)

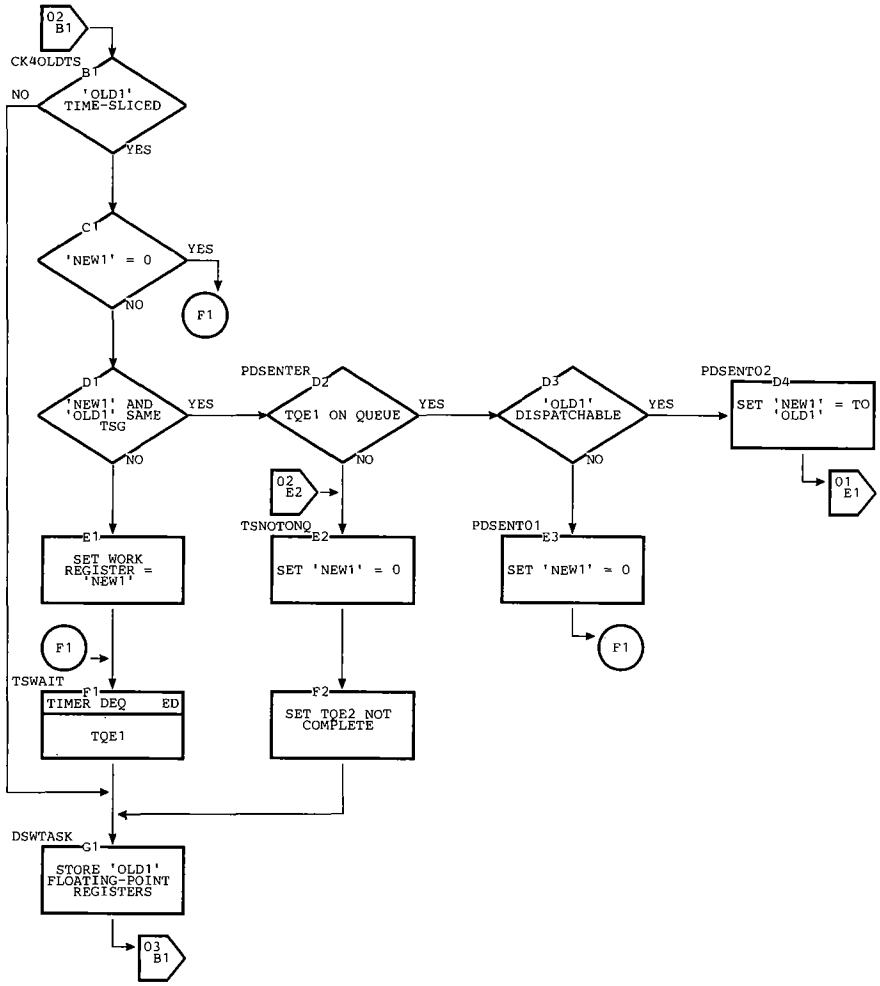


Chart KL. Dispatcher with Multiprocessing and Time Slicing (Page 3 of 9)

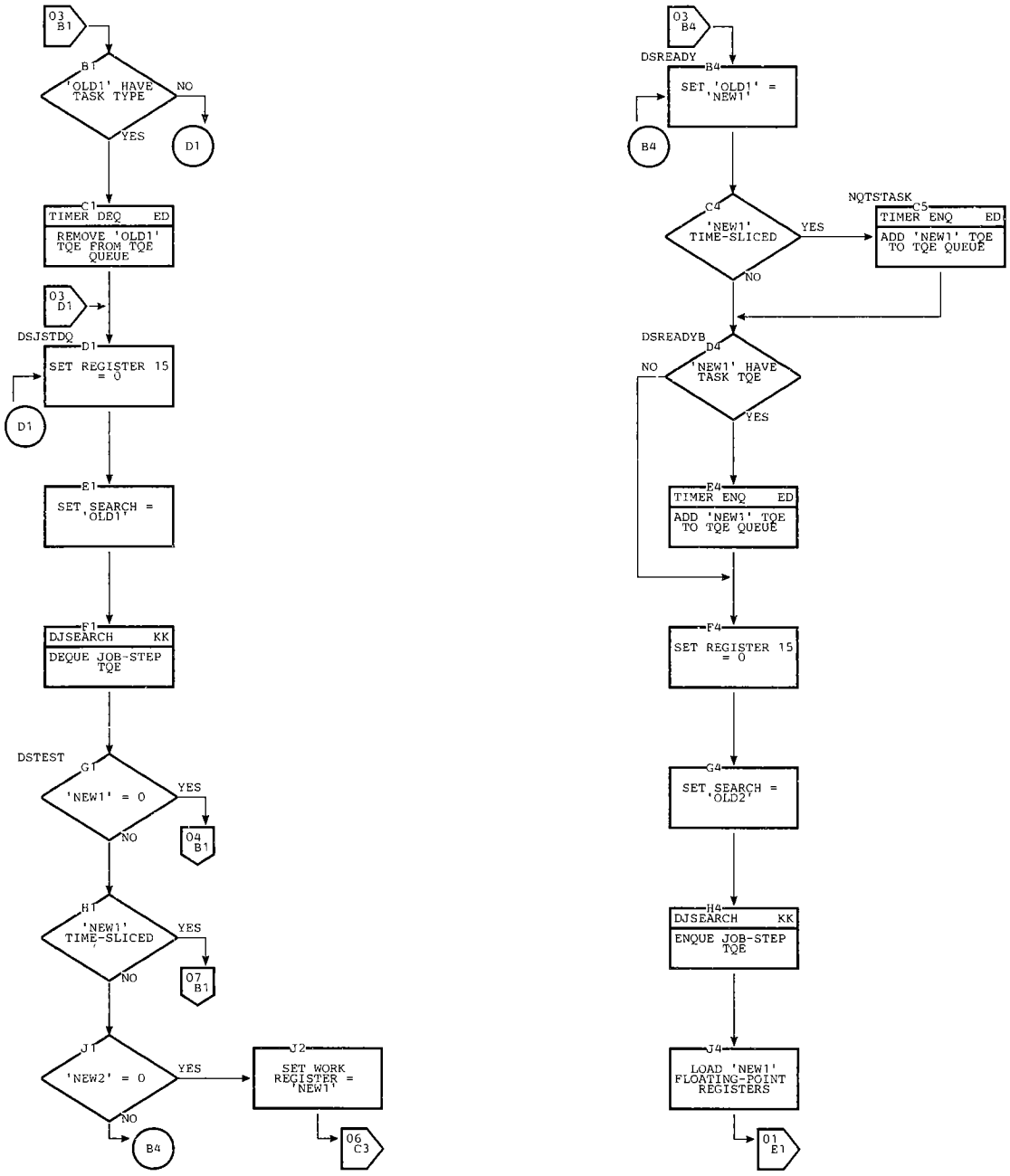


Chart KL. Dispatcher with Multiprocessing and Time Slicing (Page 4 of 9)

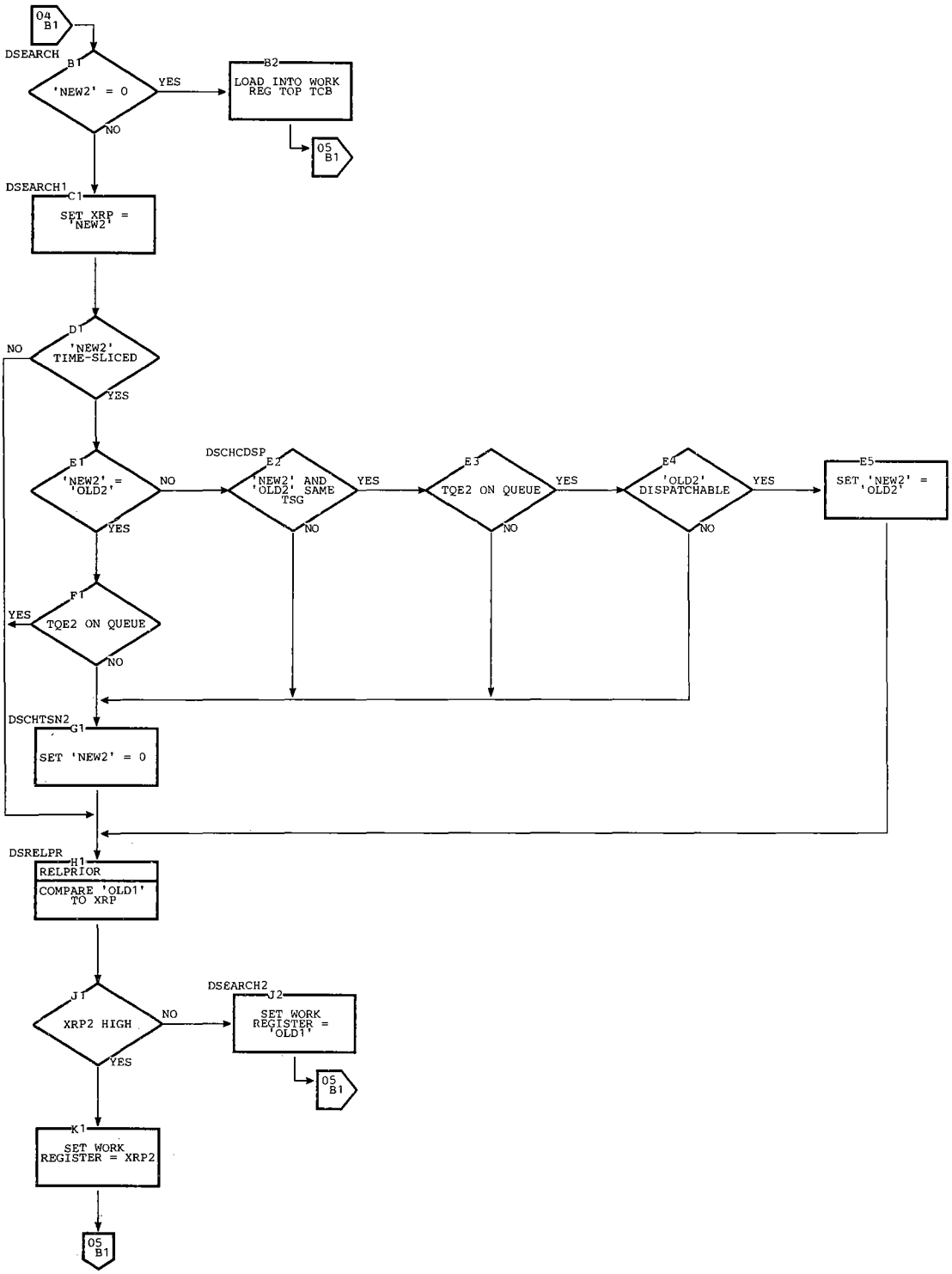


Chart KL. Dispatcher with Multiprocessing and Time Slicing (Page 5 of 9)

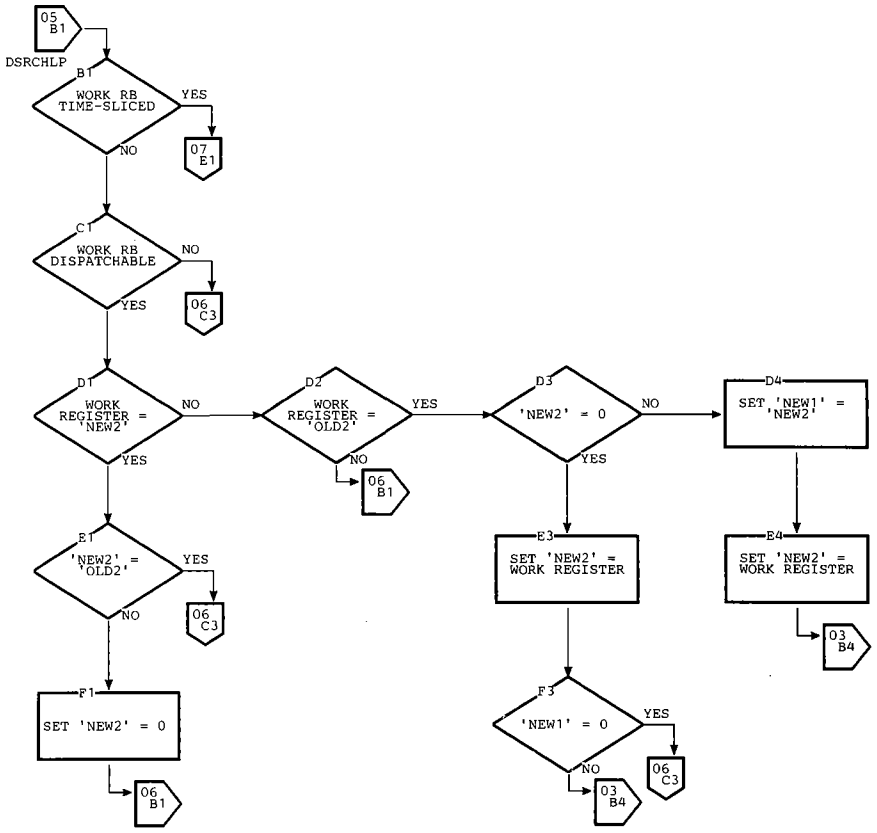


Chart KL. Dispatcher with Multiprocessing and Time Slicing (Page 6 of 9)

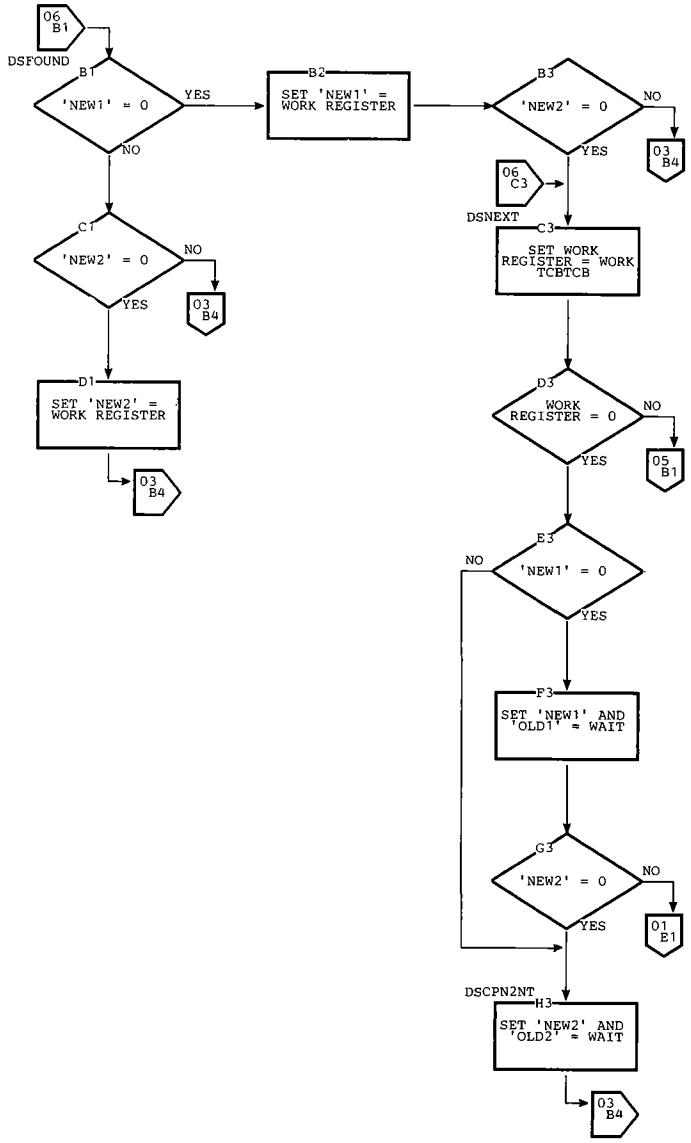


Chart KL. Dispatcher with Multiprocessing and Time Slicing (Page 7 of 9)

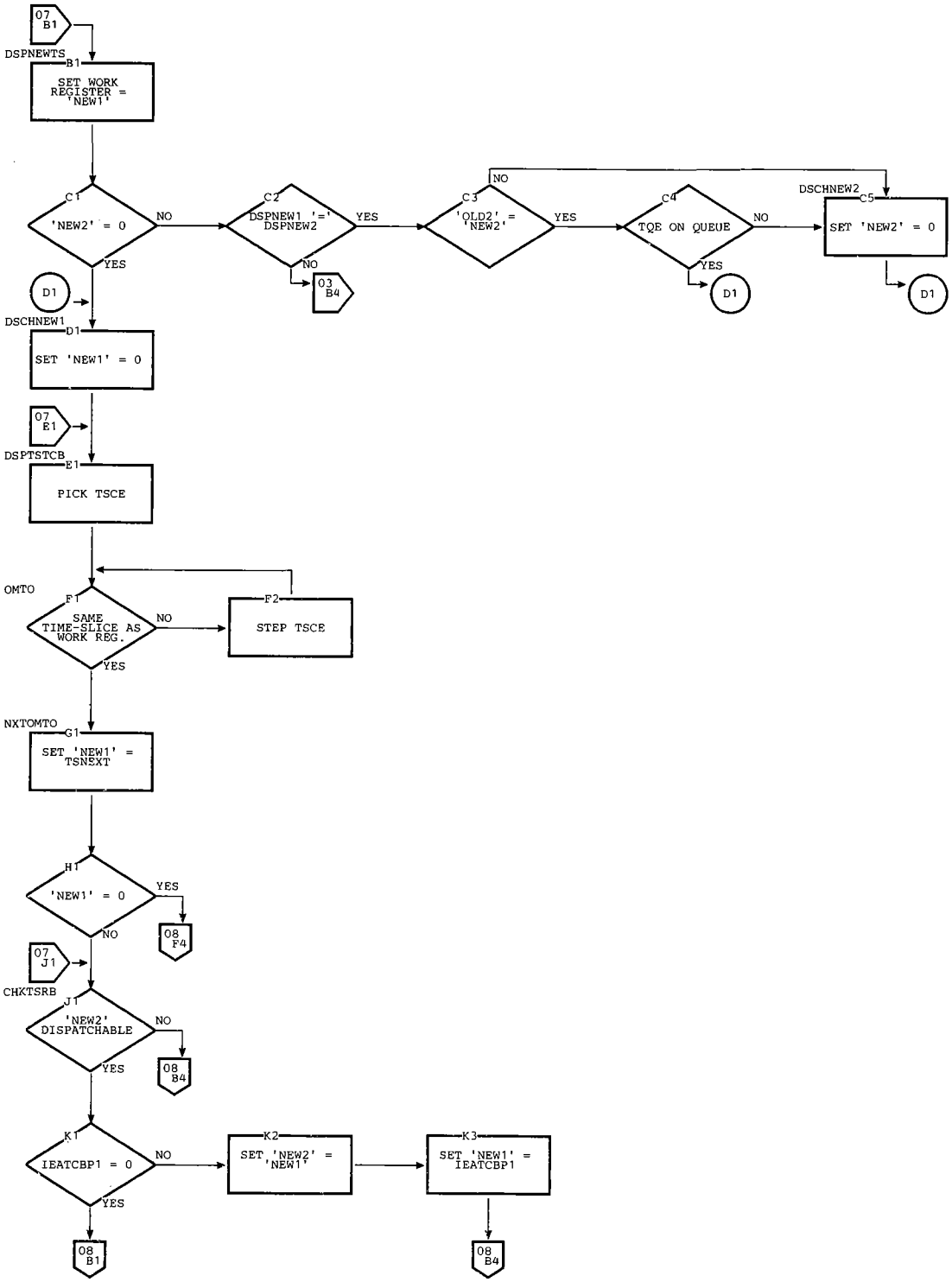


Chart KL. Dispatcher with Multiprocessing and Time Slicing (Page 8 of 9)

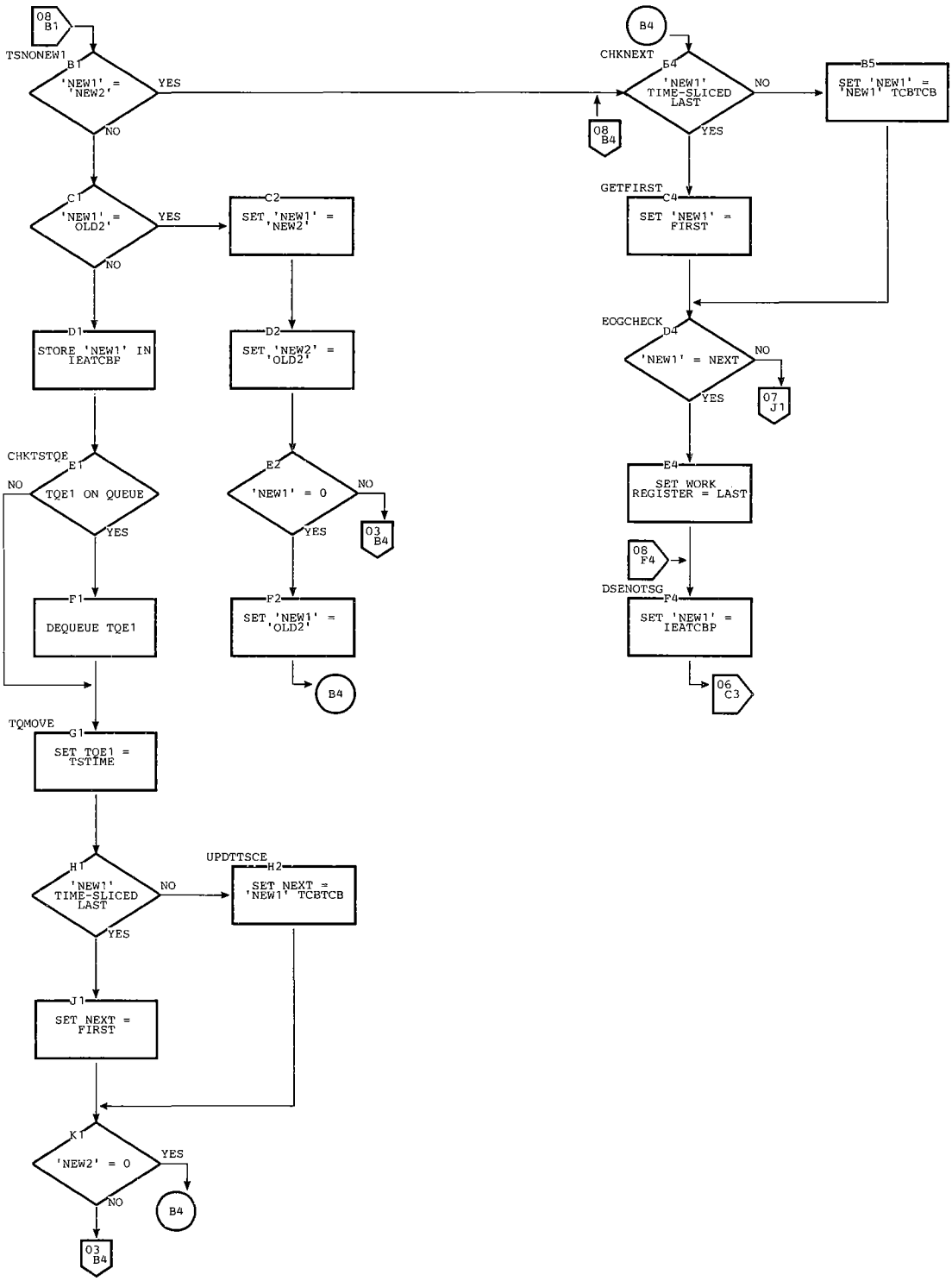


Chart KL. Dispatcher with Multiprocessing and Time Slicing (Page 9 of 9)

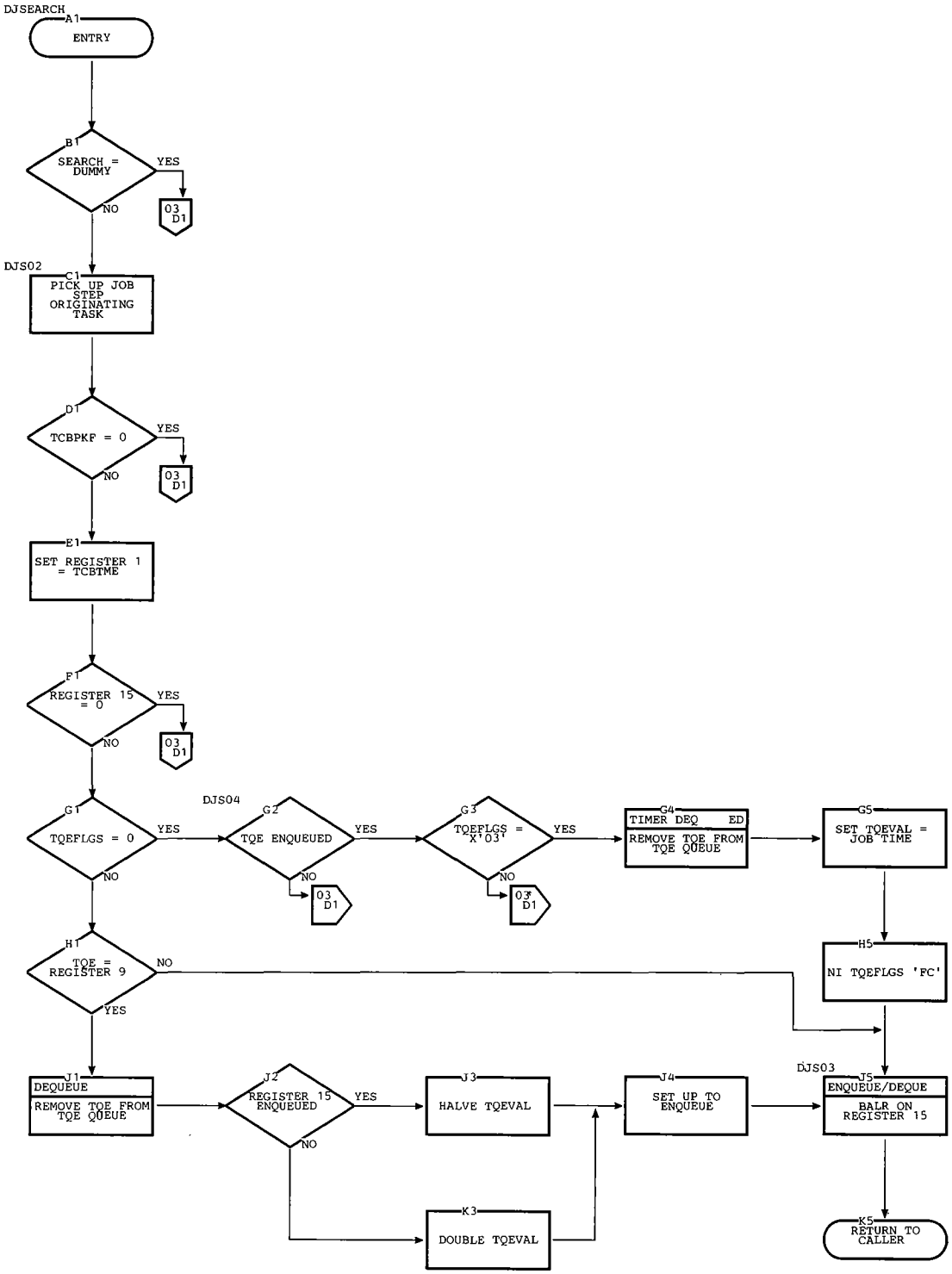


Chart LA. End-Of-Task (EOT)

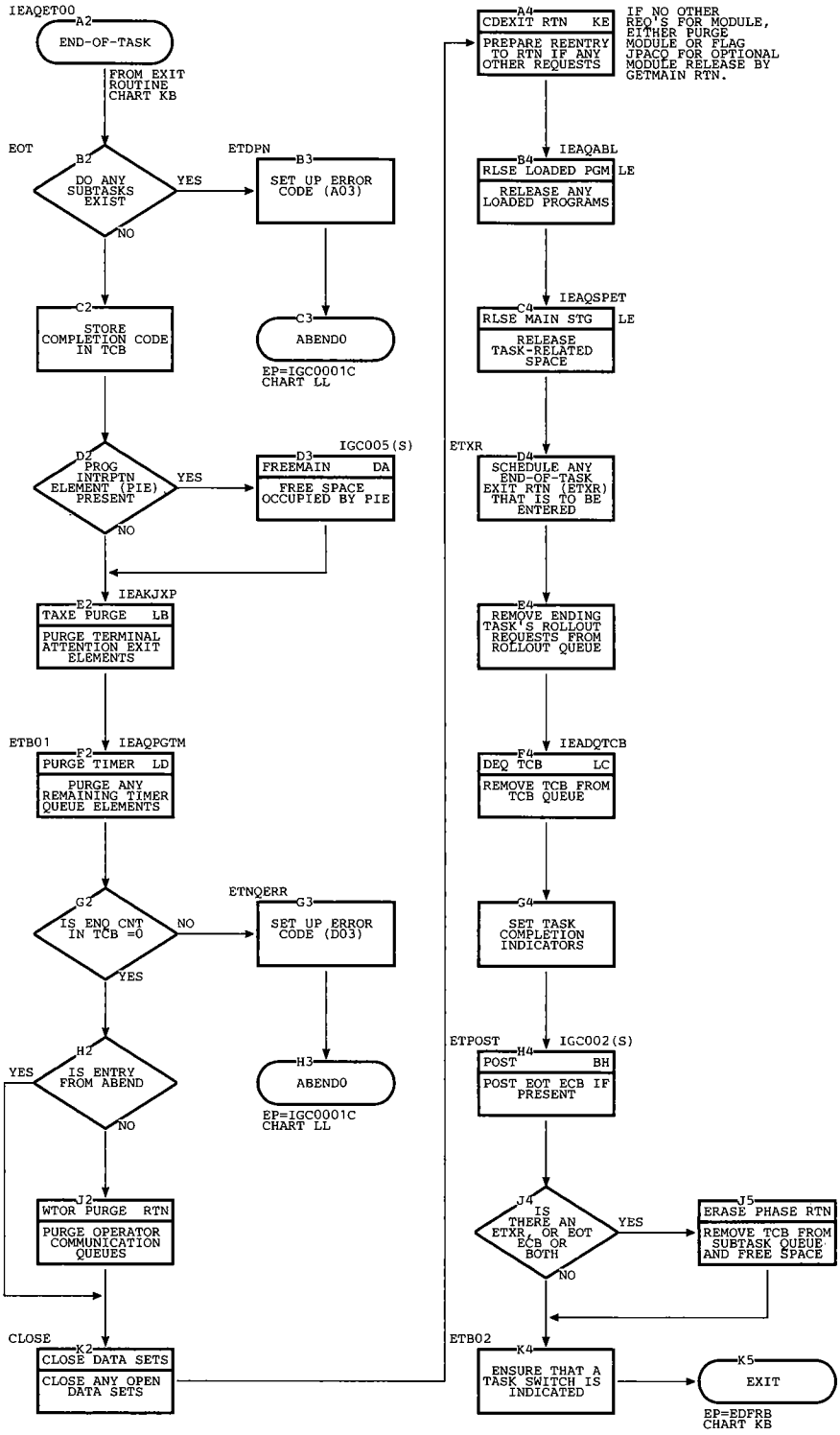


Chart LB. Terminal Attention Exit Element Purge

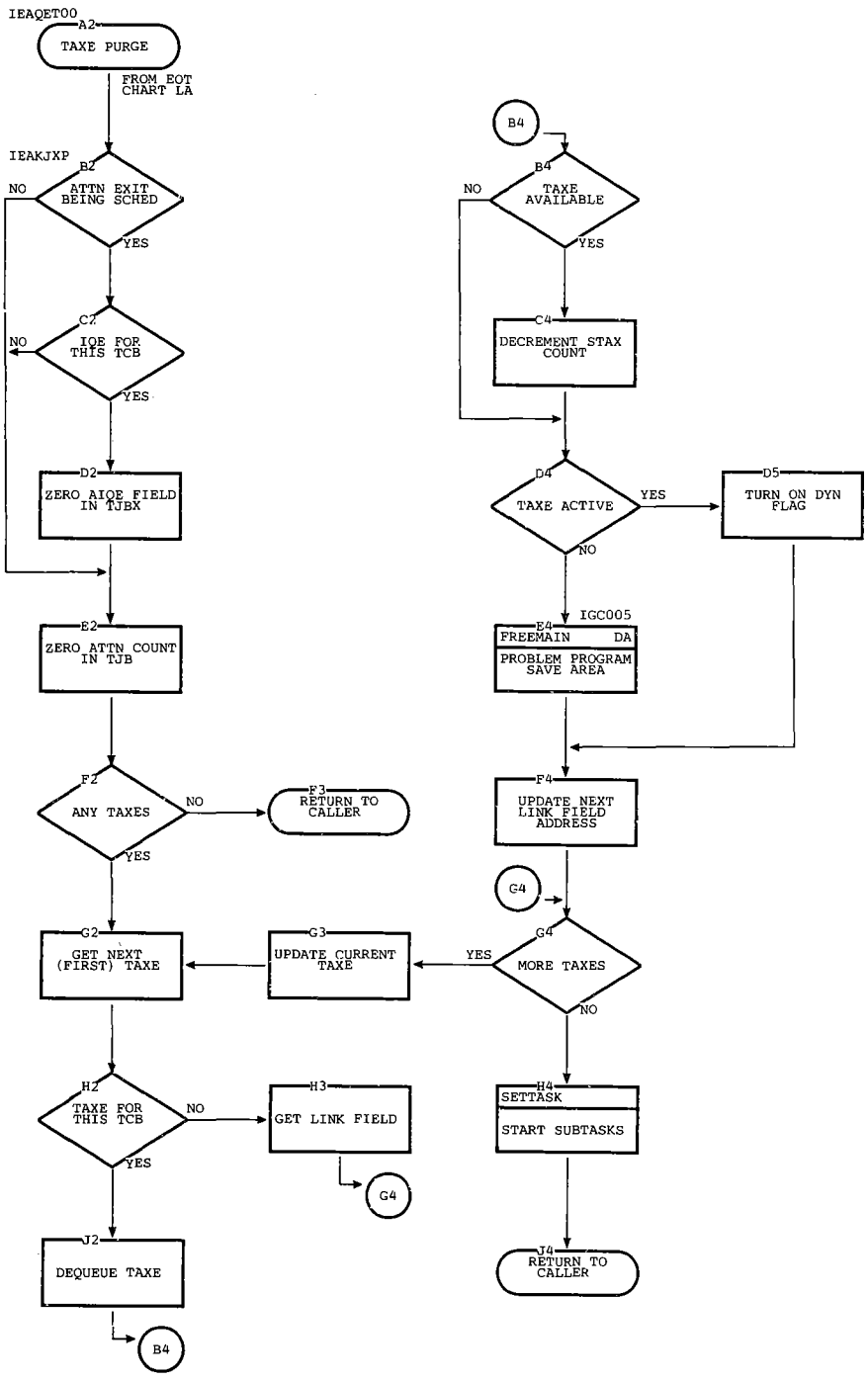


Chart LC. TCB Dequeue

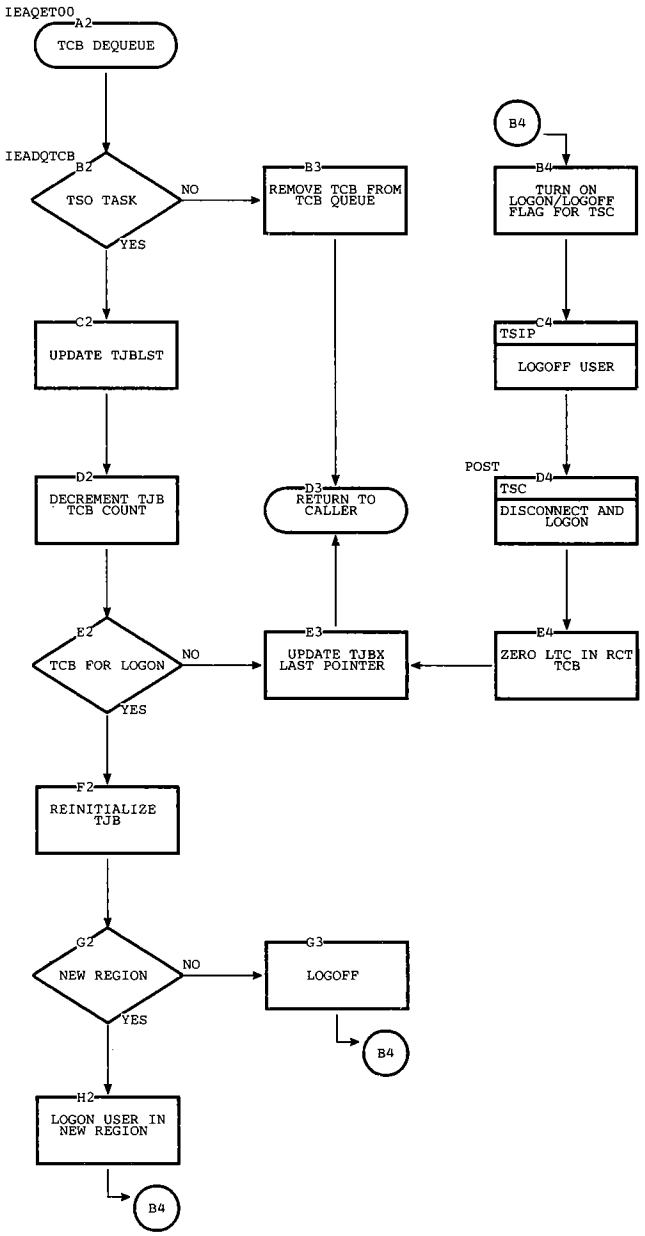


Chart LD. Purge Timer

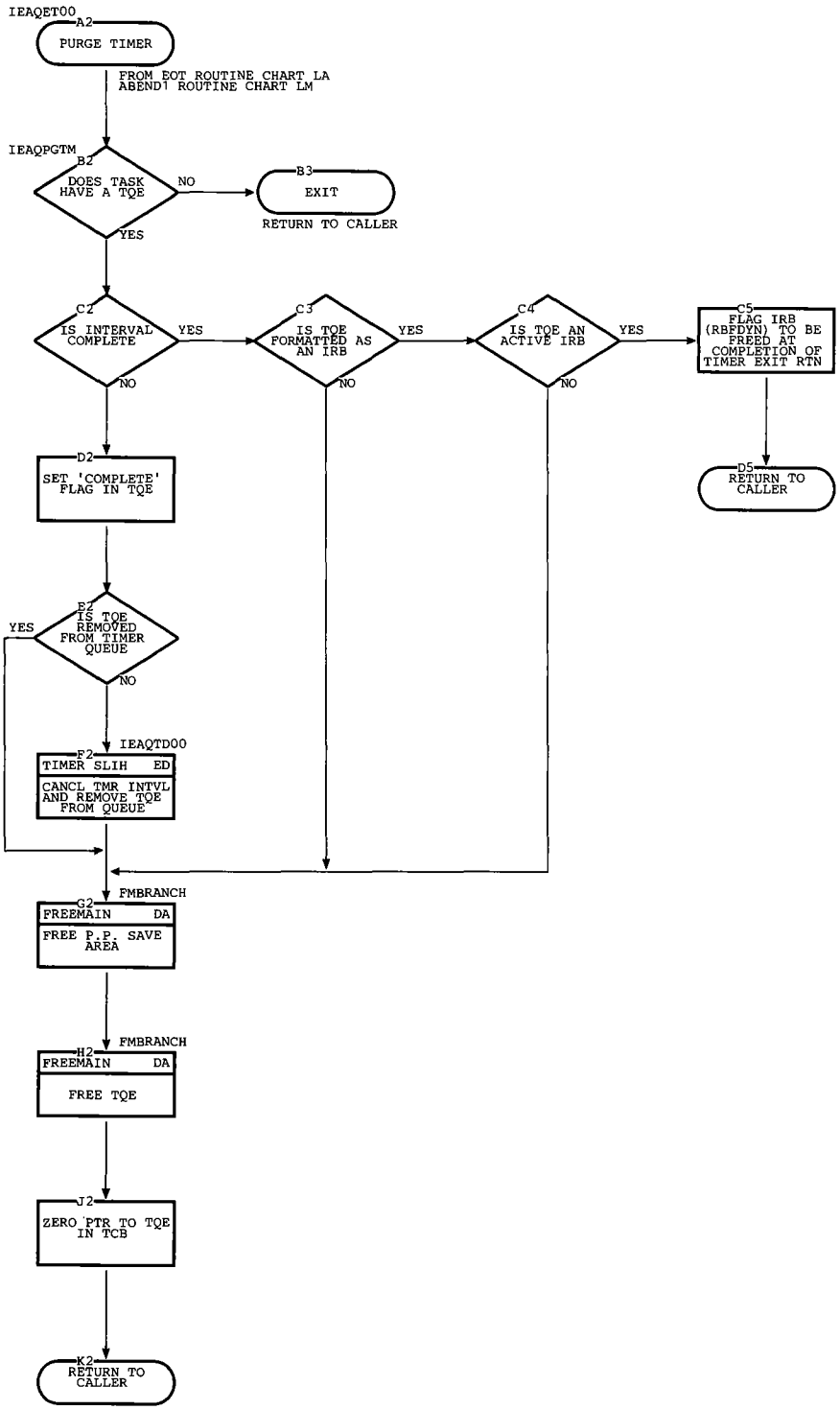


Chart LE. Release Main Storage and Release Loaded Programs

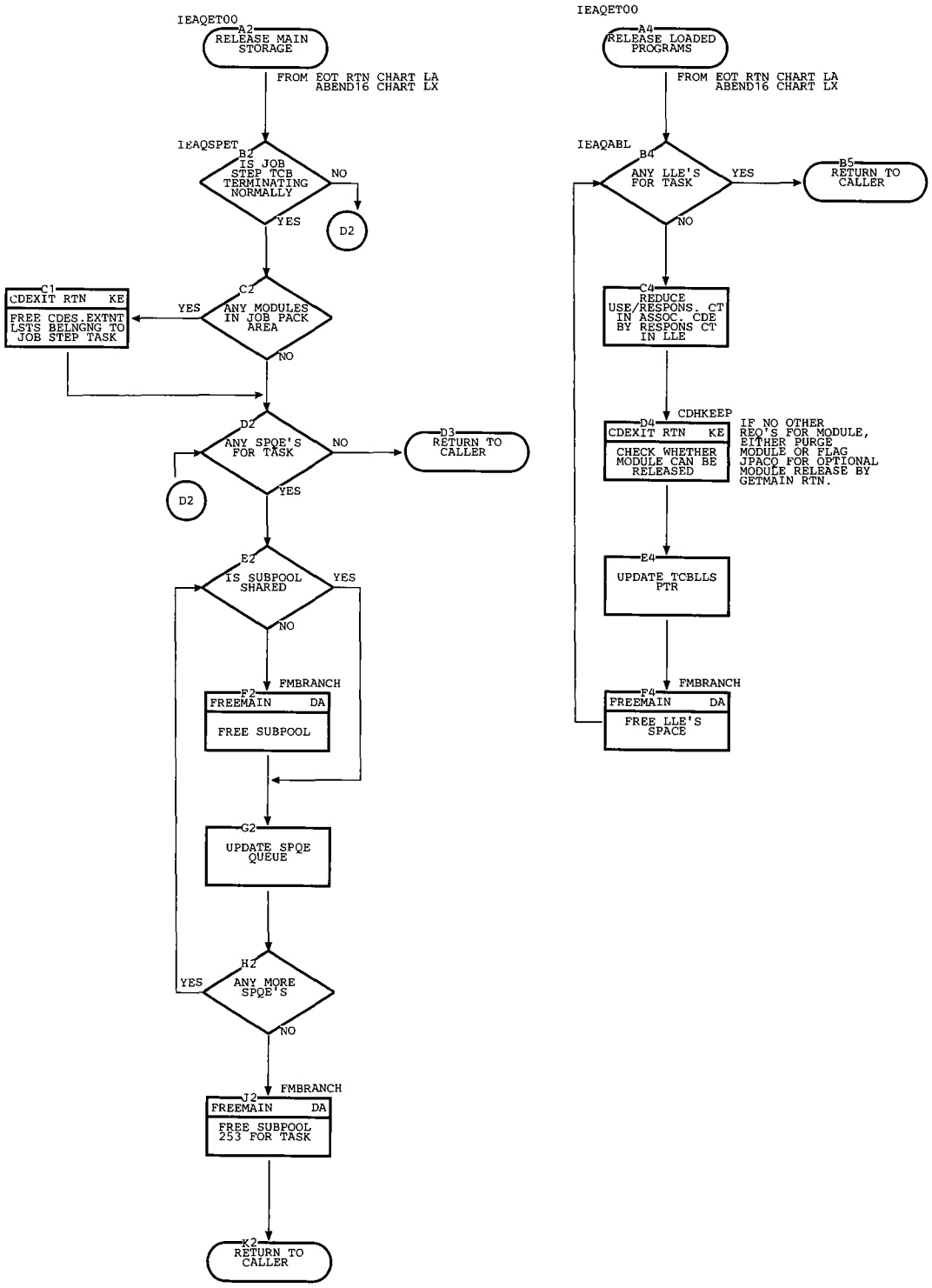


Chart LF. ABTERM

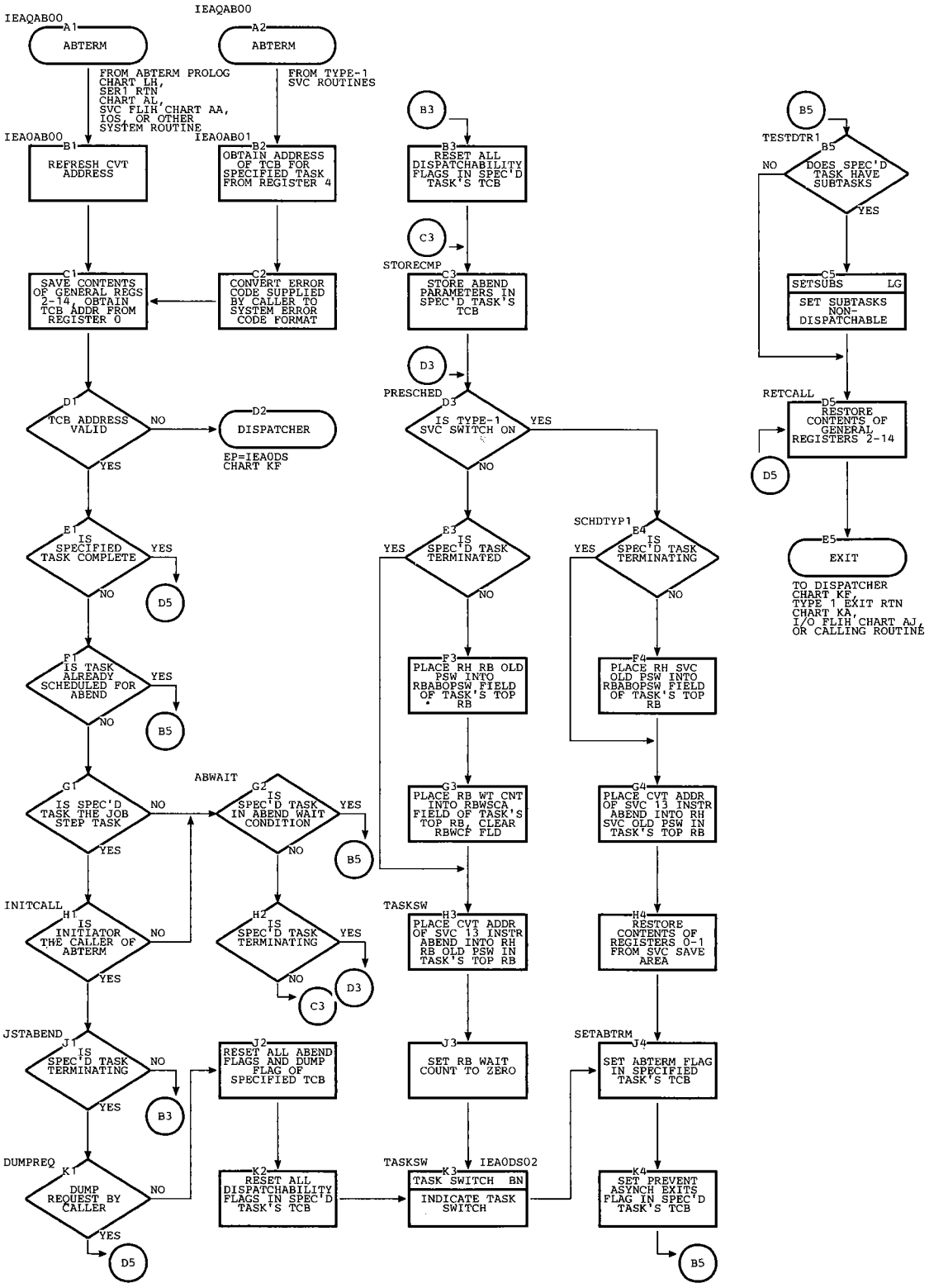


Chart LG. ABTERM Setsubs Subroutine

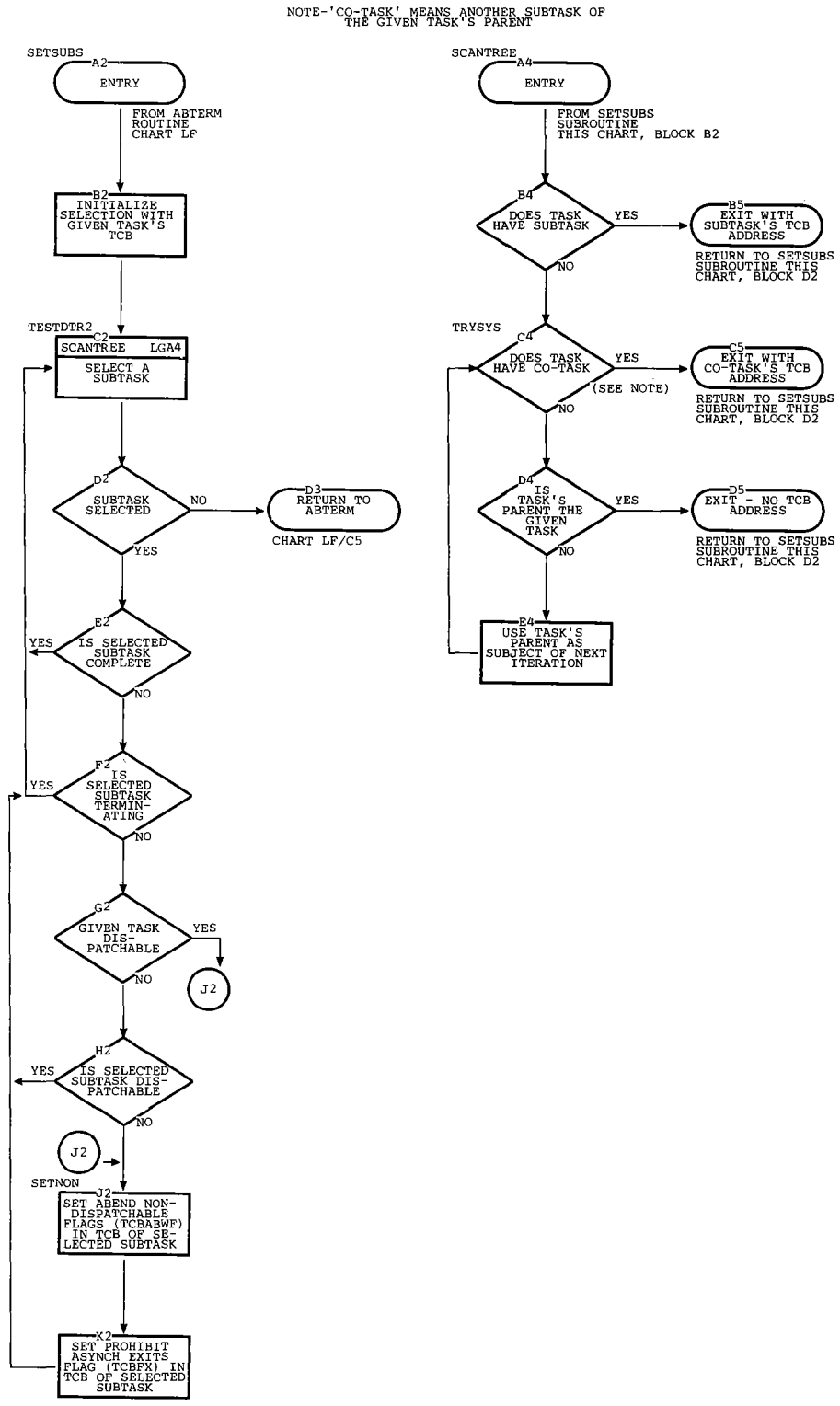


Chart LH. ABTERM Prologue

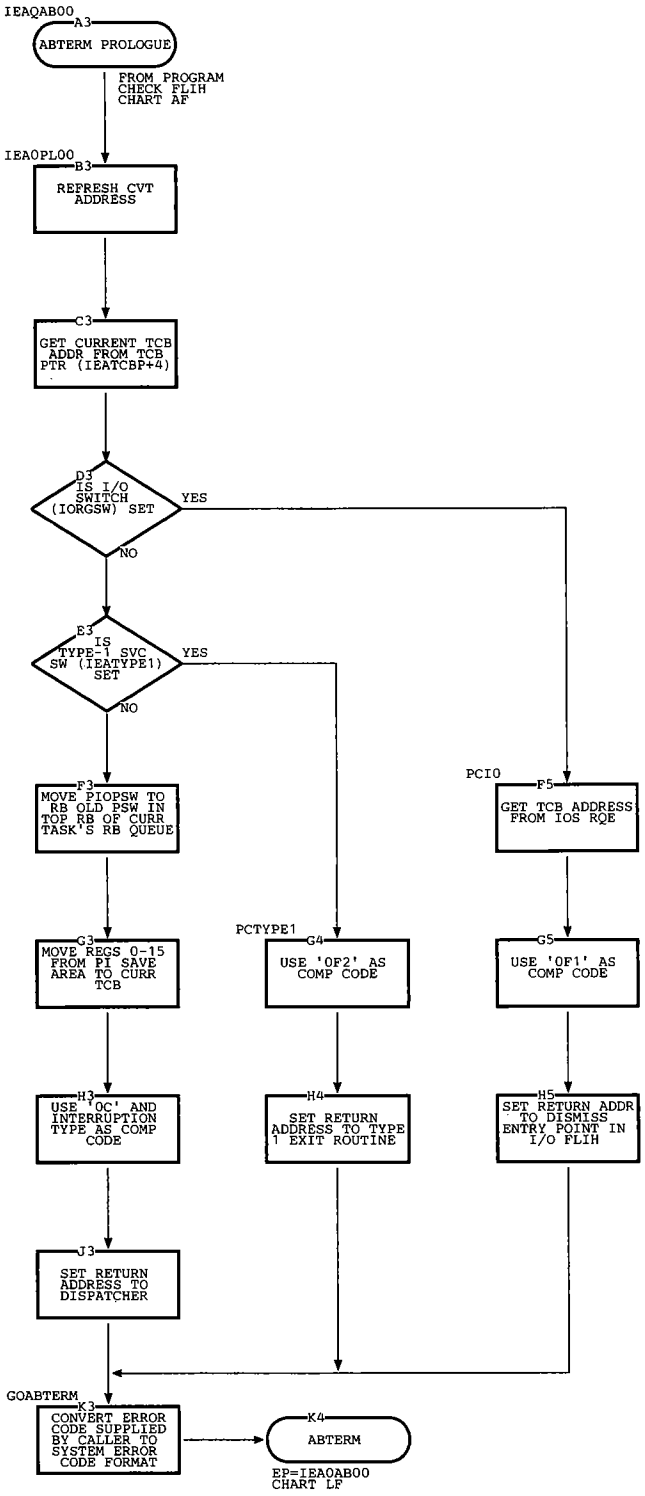


Chart LI. ABDUMP Modules (Page 1 of 2)

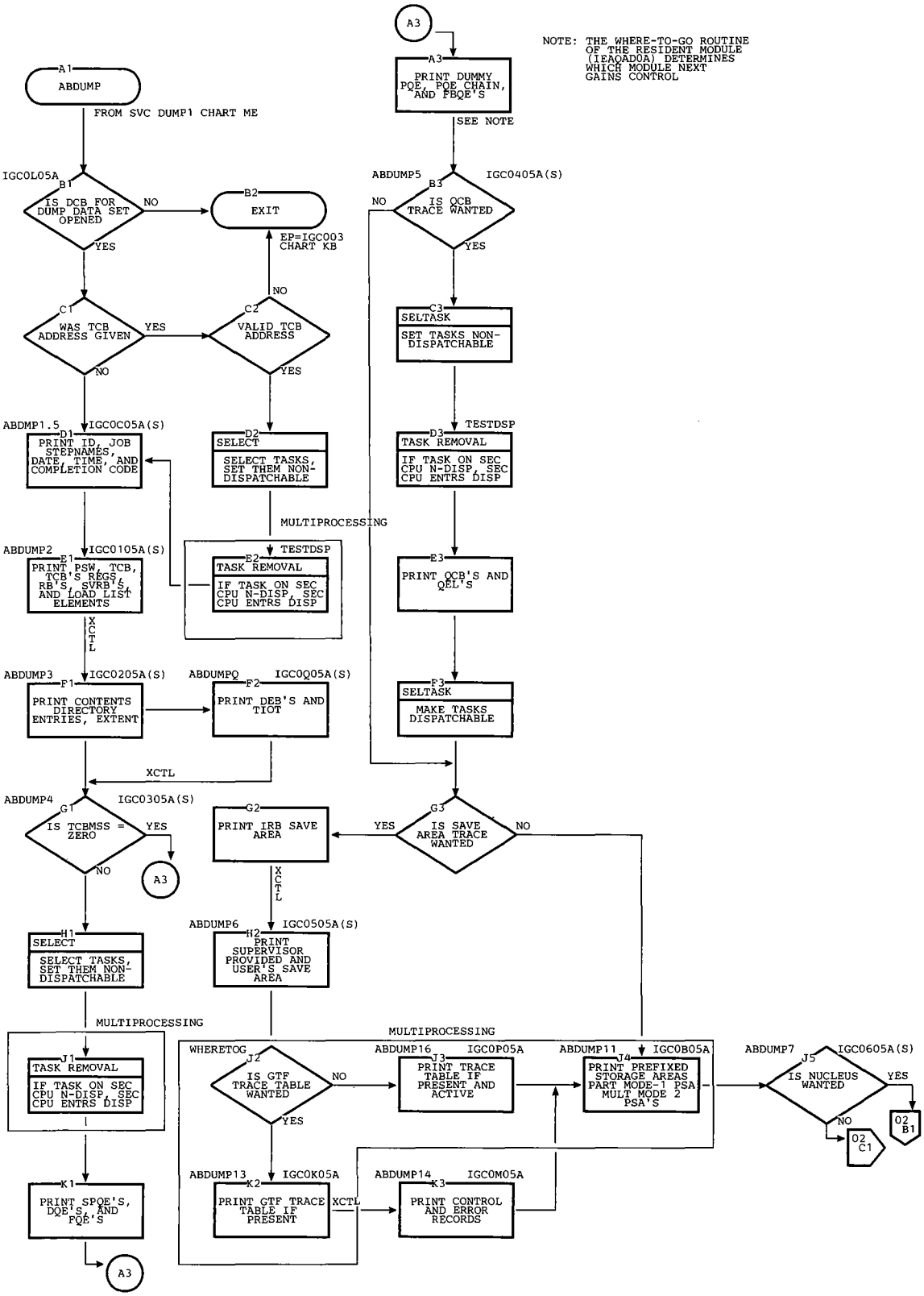


Chart LI. ABDUMP Modules (Page 2 of 2)

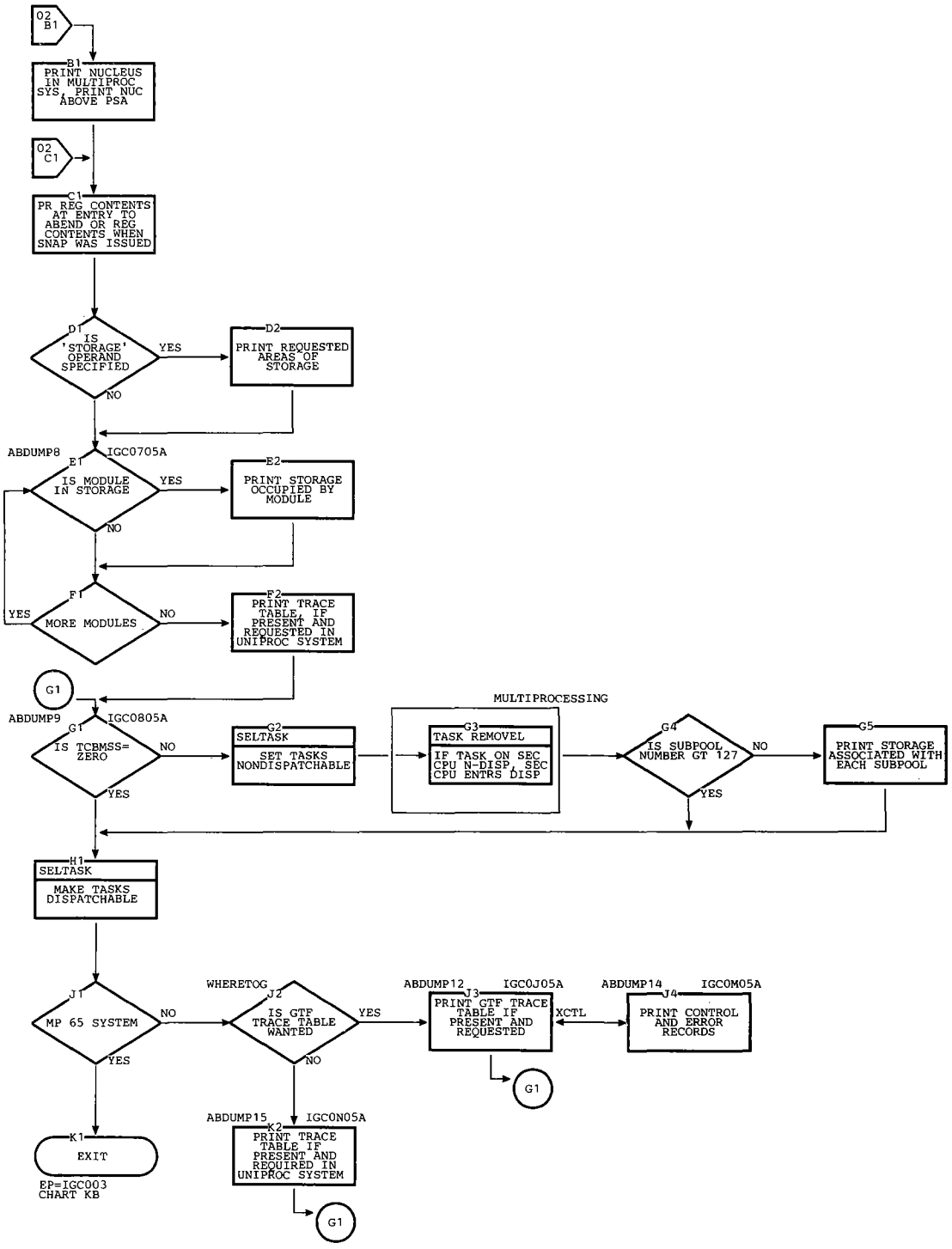


Chart LJ. TCAM ABDUMP Modules

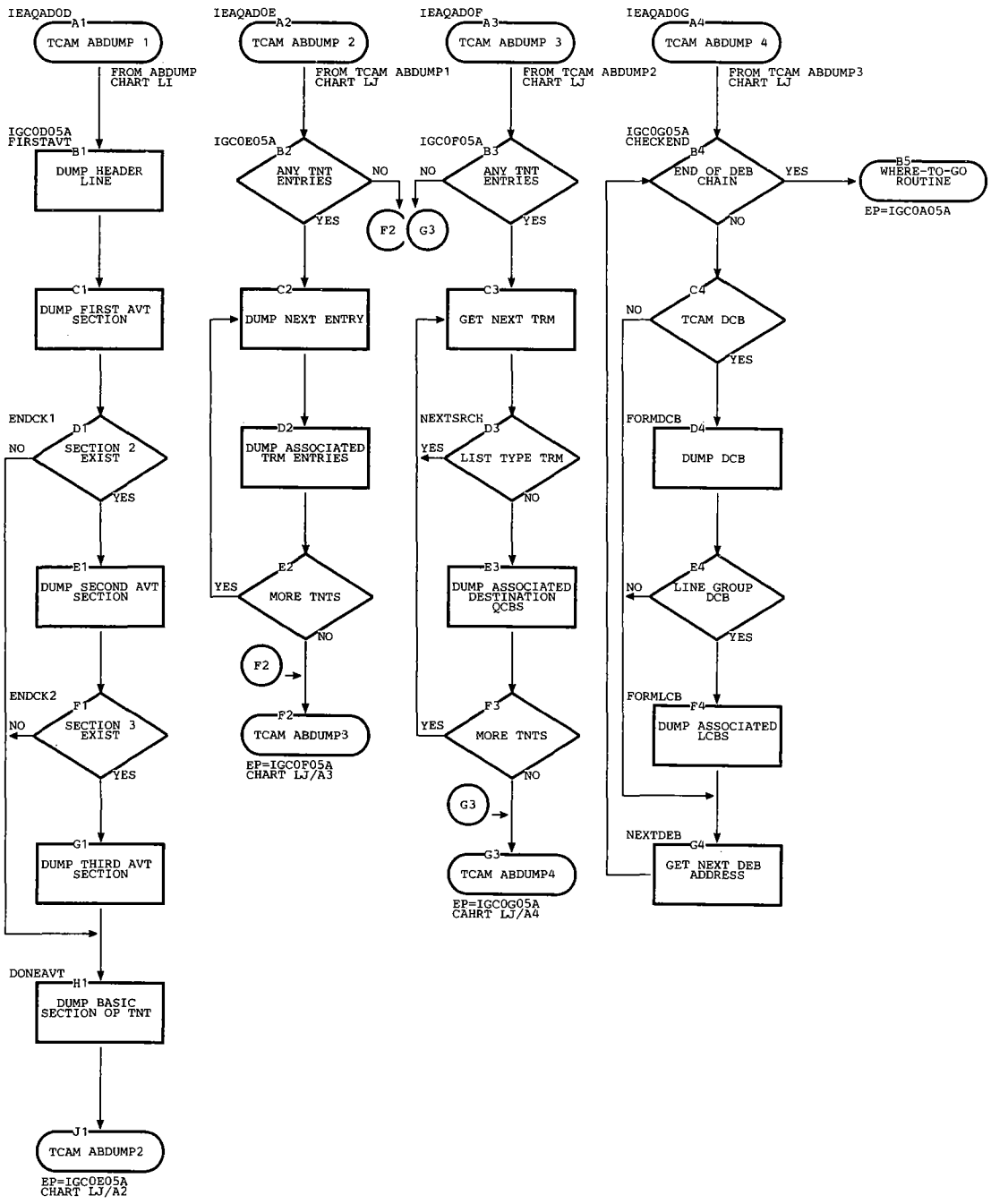


Chart LK. Abnormal Termination Modular Overview

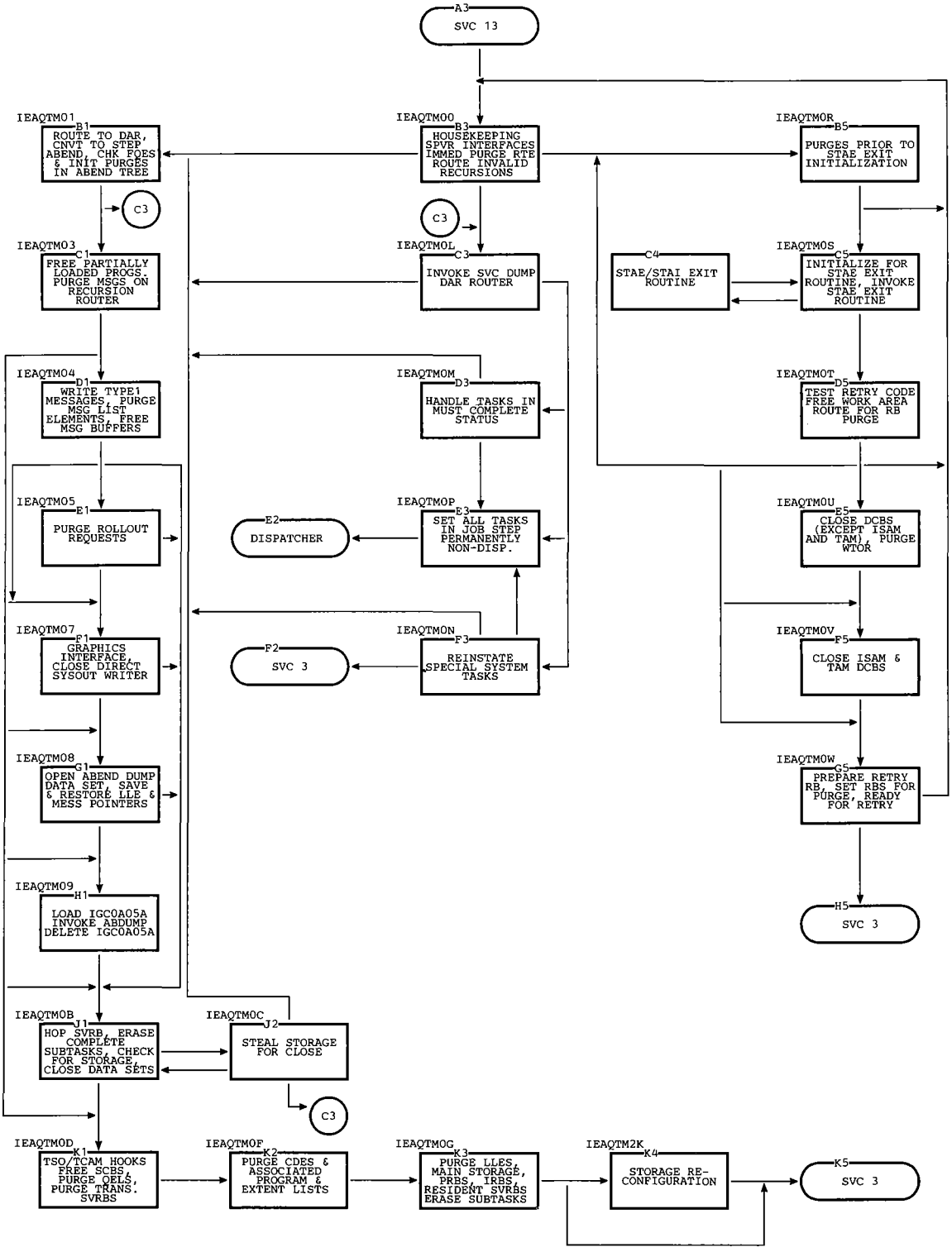


Chart LL. ABEND0 (Page 1 of 3)

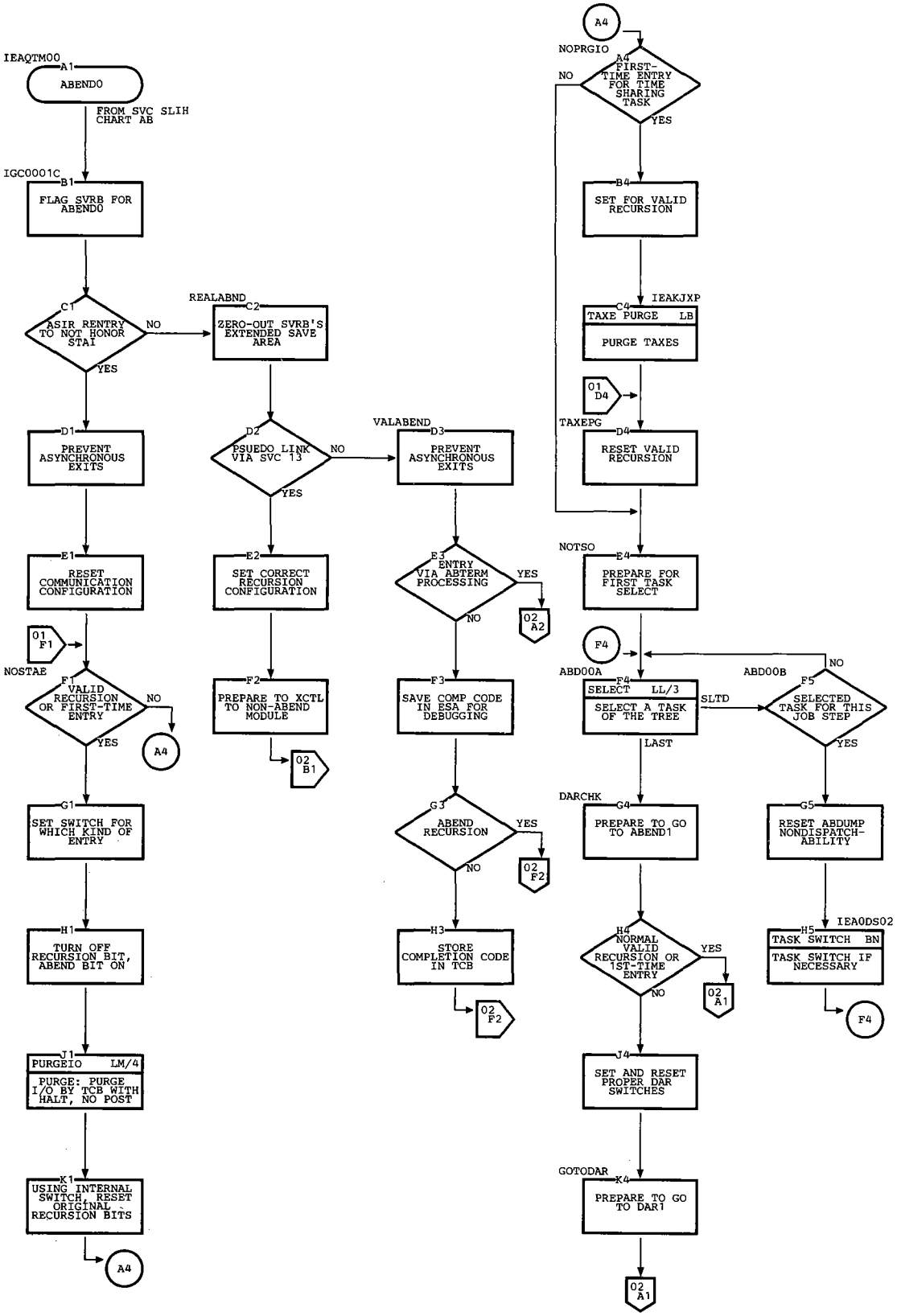


Chart LL. ABEND0 (Page 2 of 3)

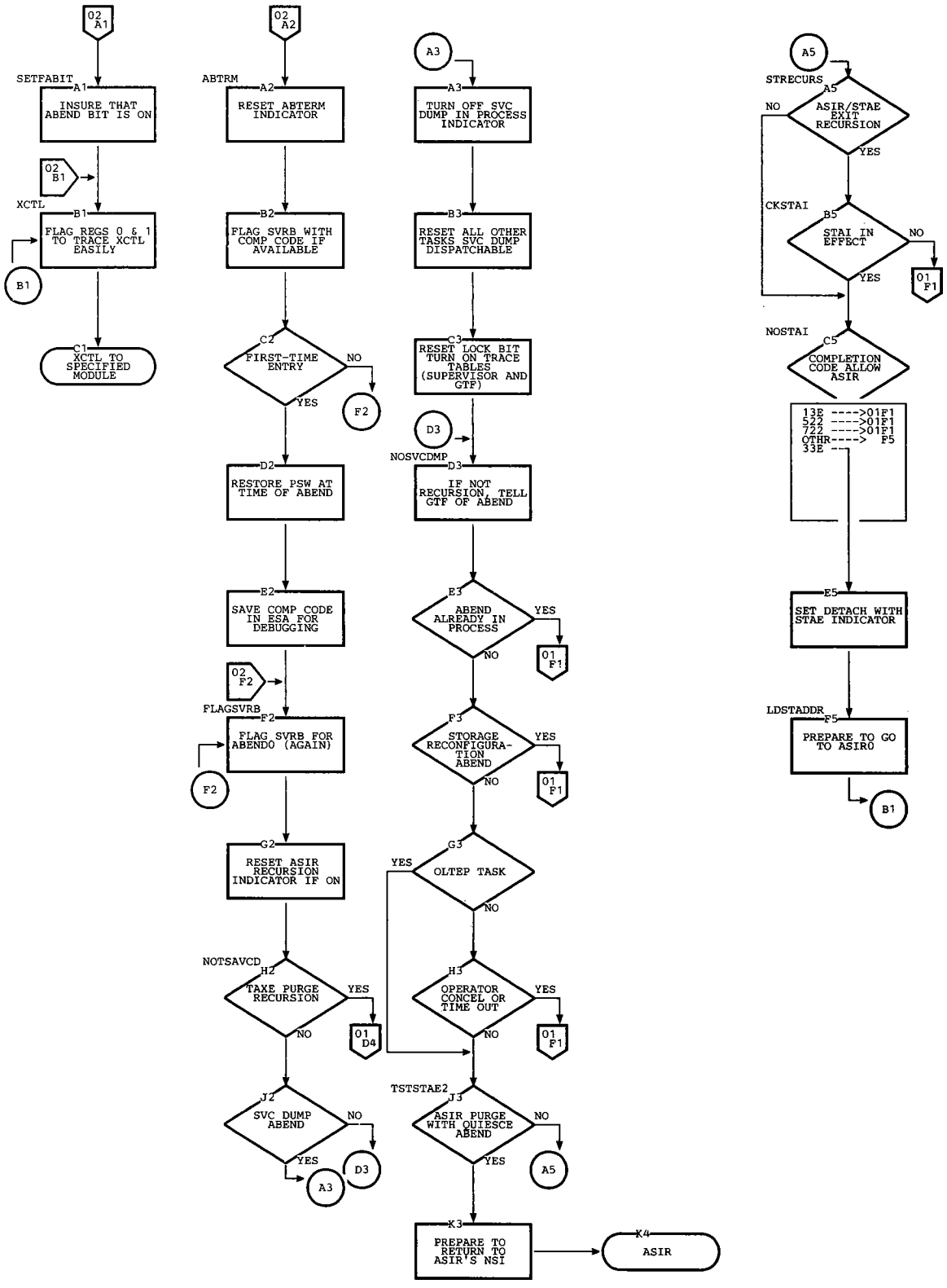


Chart LL. ABEND0 (Page 3 of 3)

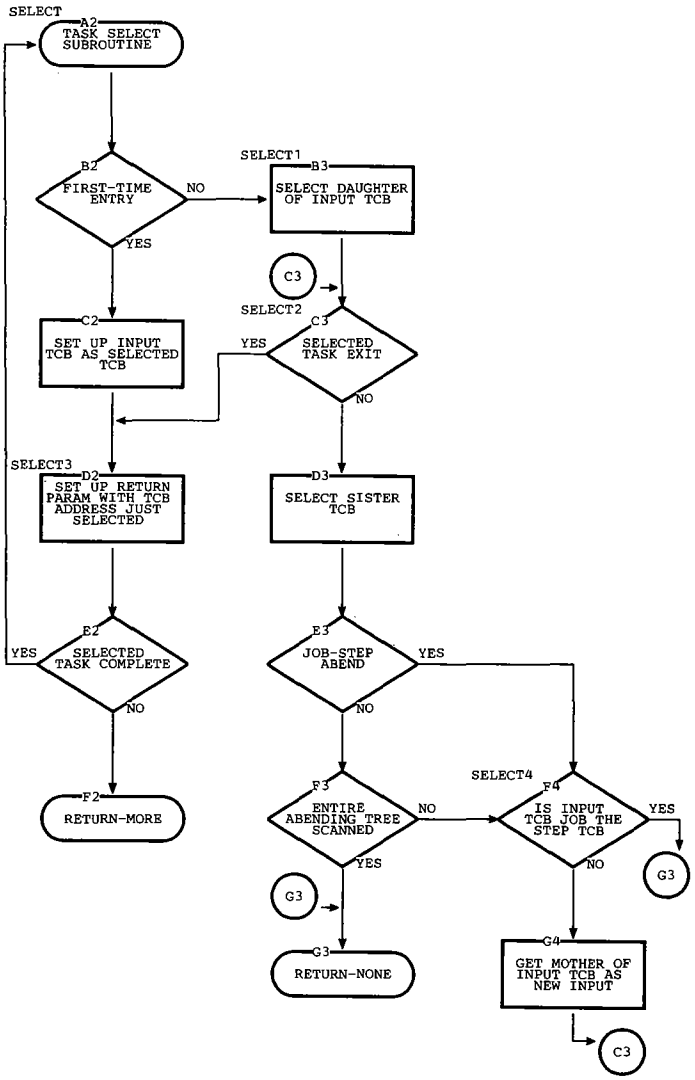


Chart LM. ABEND1 (Page 1 of 5)

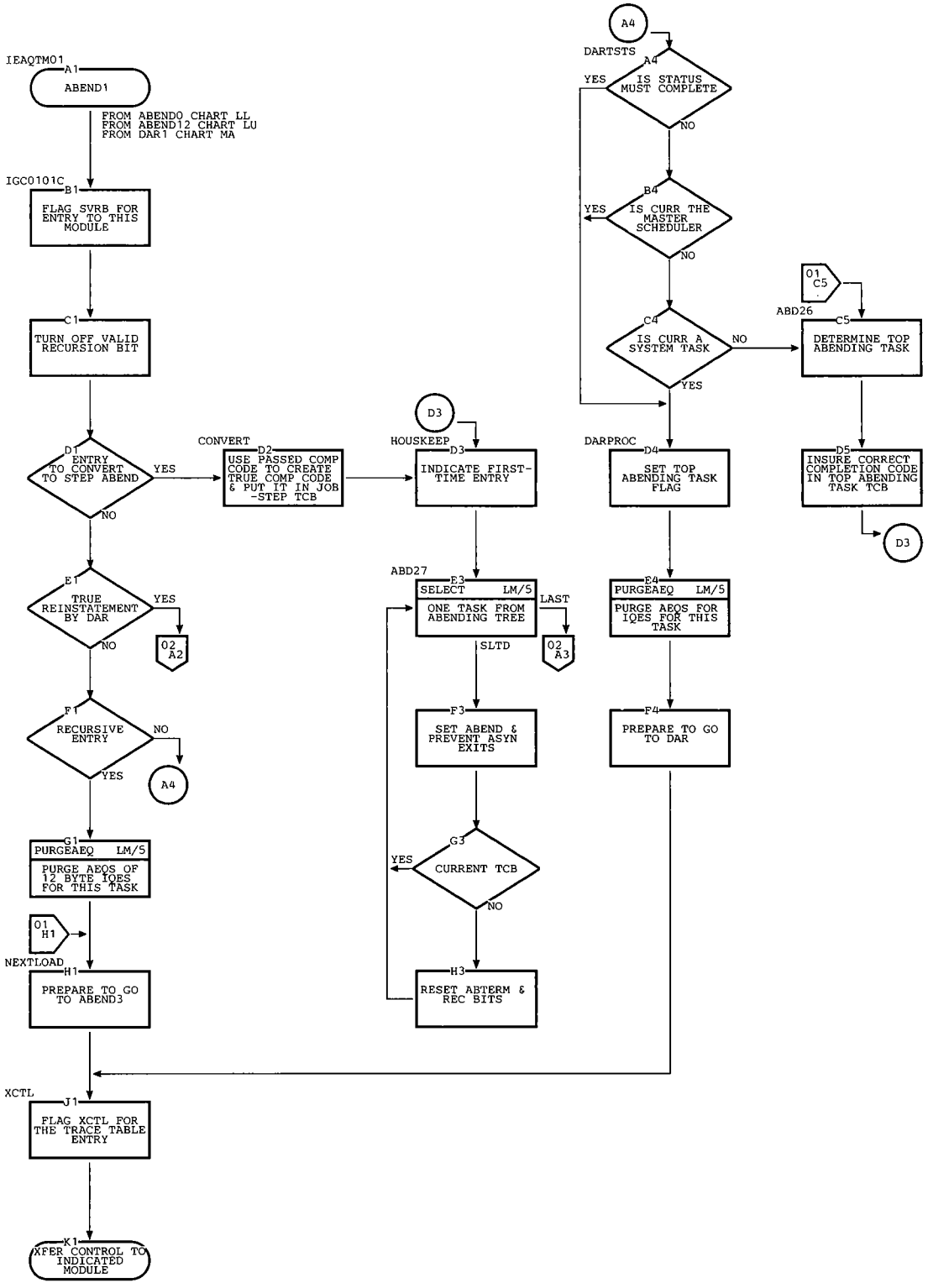


Chart LM. ABEND1 (Page 2 of 5)

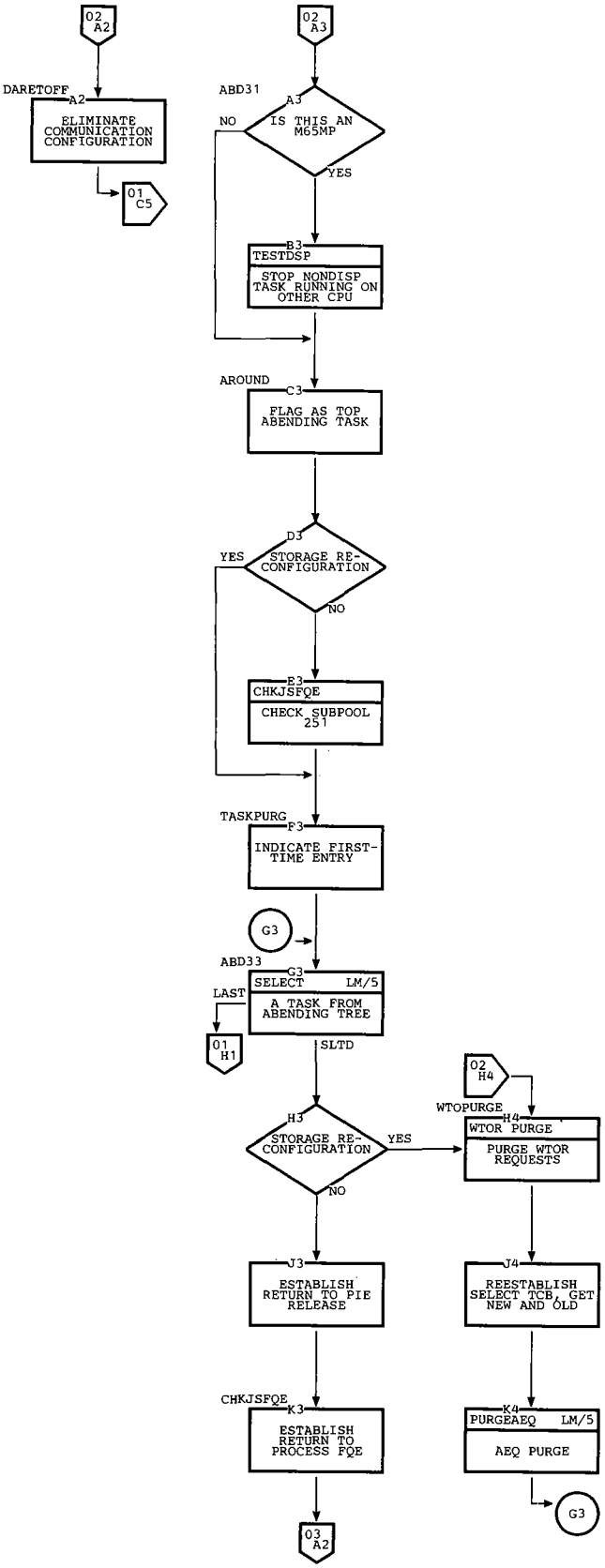
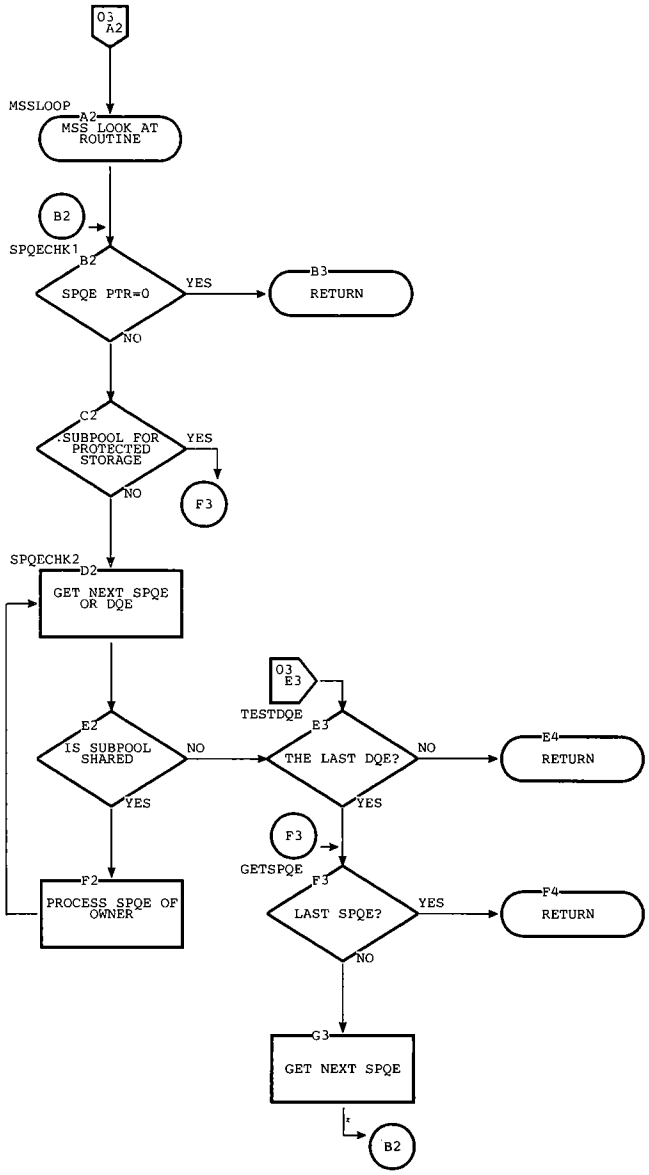


Chart LM. ABEND1 (Page 3 of 5)



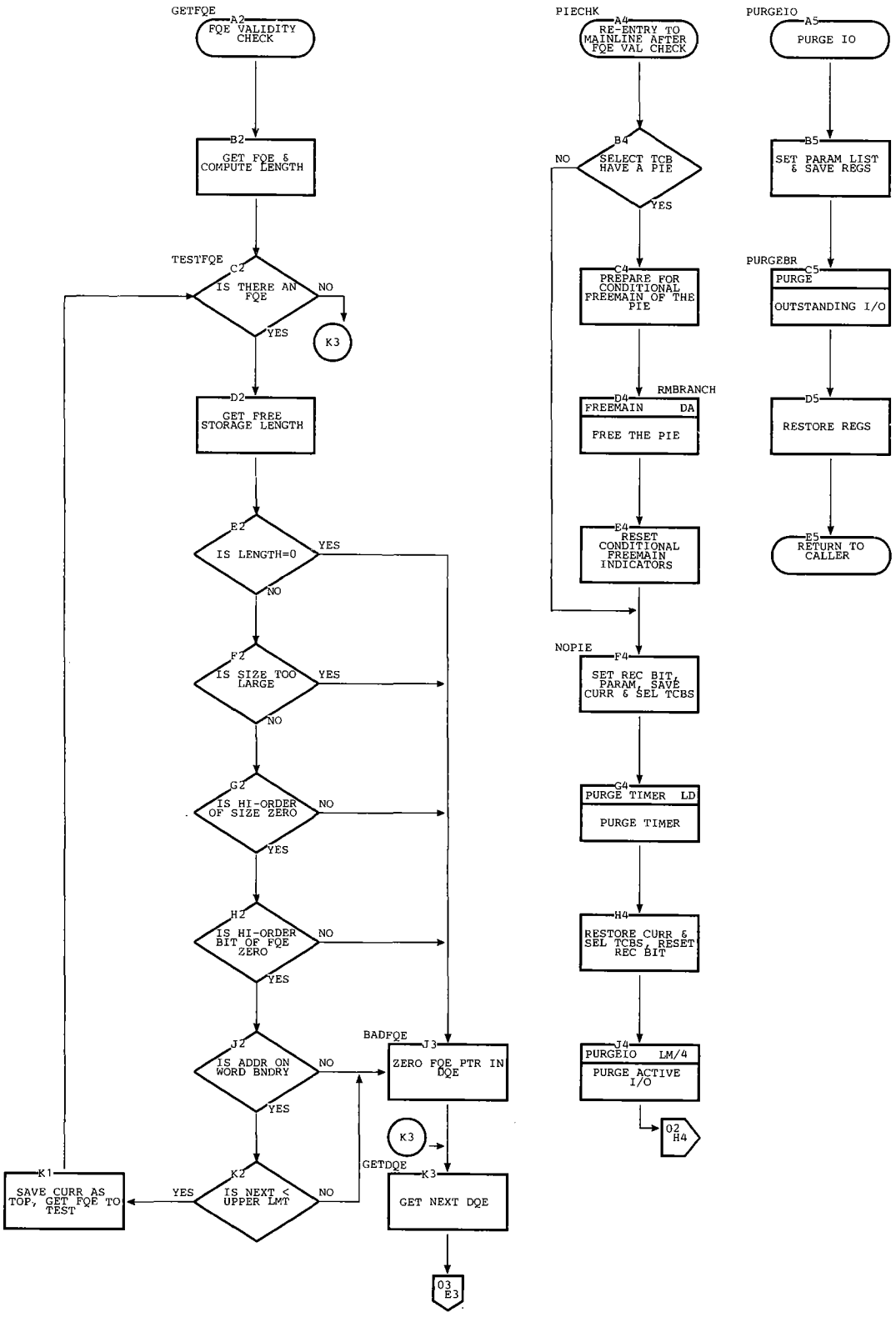

GETFQE
A2
FQE VALIDITY CHECK
B2
GET FQE & COMPUTE LENGTH
TESTFQE
C2
IS THERE AN FQE
NO
YES
K3
D2
GET FREE STORAGE LENGTH
E2
IS LENGTH=0
YES
NO
F2
IS SIZE TOO LARGE
YES
NO
G2
IS HI-ORDER OF SIZE ZERO
NO
YES
H2
IS HI-ORDER BIT OF FQE ZERO
NO
YES
J2
IS ADDR ON WORD BNDRY
NO
YES
BADFQE
J3
ZERO FQE PTR IN DQE
K3
GETDQE
K3
GET NEXT DQE
K2
IS NEXT < UPPER LMT
YES
NO
K1
SAVE CURR AS TOP, GET FQE TO TEST
03 E3
PIECHK
A4
RE-ENTRY TO MAINLINE AFTER FQE VAL CHECK
B4
SELECT TCB HAVE A PIE
NO
YES
C4
PREPARE FOR CONDITIONAL FREEMAIN OF THE PIE
RMBRANCH
D4
FREEMAIN DA
FREE THE PIE
E4
RESET CONDITIONAL FREEMAIN INDICATORS
NOPIE
F4
SET REC BIT, PARAM, SAVE CURR & SEL TCBS
C4
PURGE TIMER LD
PURGE TIMER
H4
RESTORE CURR & SEL TCBS, RESET REC BIT
J4
PURGEIO LM/4
PURGE ACTIVE I/O
02 H4
PURGEIO
A5
PURGE IO
B5
SET PARAM LIST & SAVE REGS
PURGEBR
C5
PURGE
OUTSTANDING I/O
D5
RESTORE REGS
E5
RETURN TO CALLER

Chart LM. ABEND1 (Page 5 of 5)

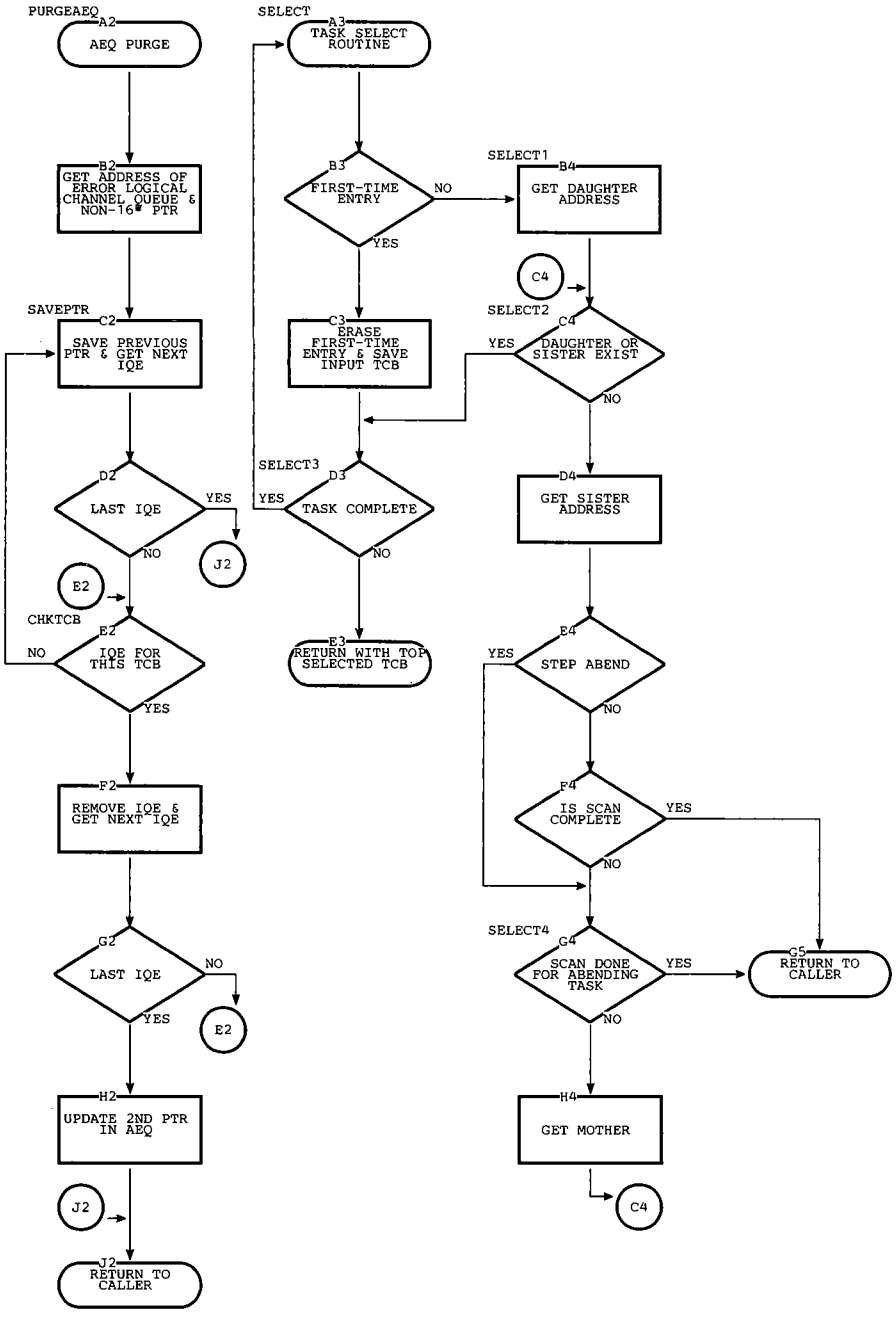


Chart LN. ABEND3 (Page 1 of 2)

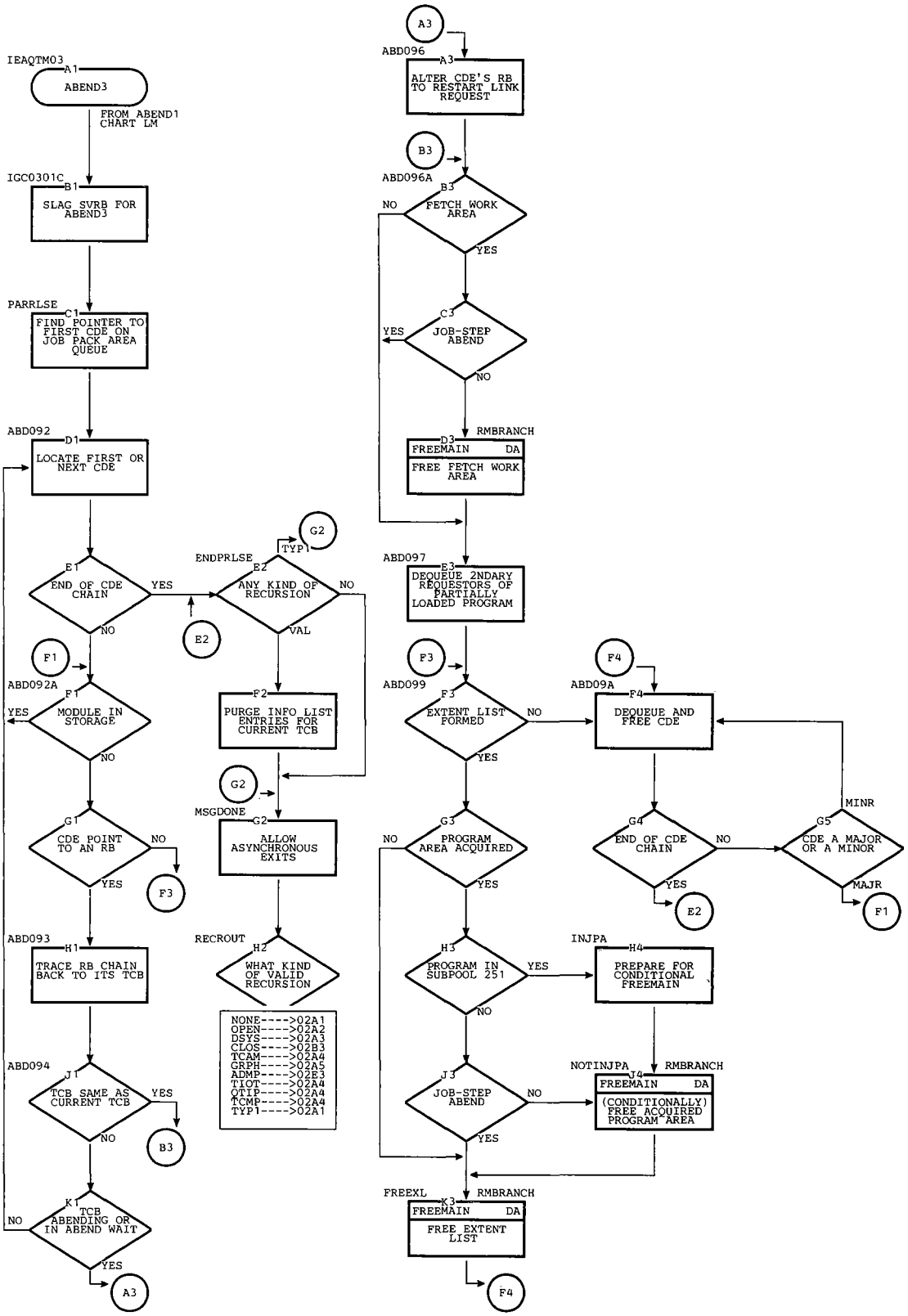


Chart LN. ABEND3 (Page 2 of 2)

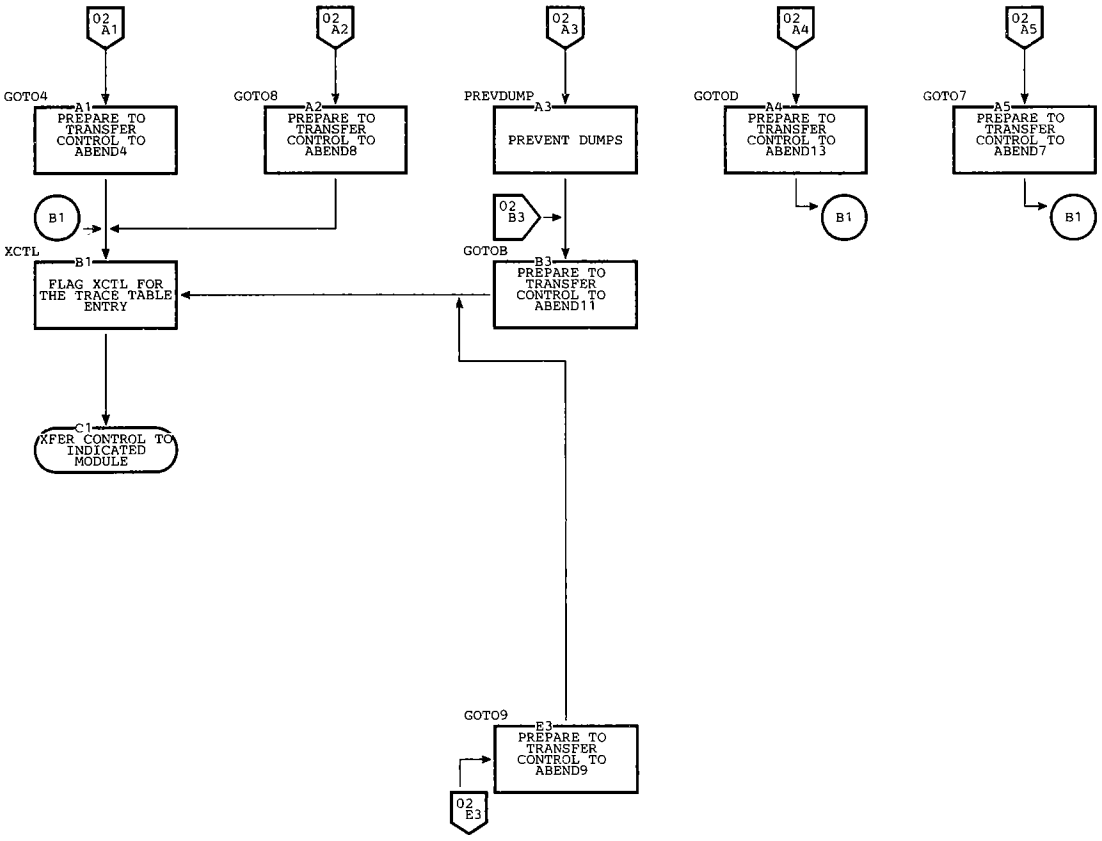


Chart LO. ABEND4 (Page 1 of 2)

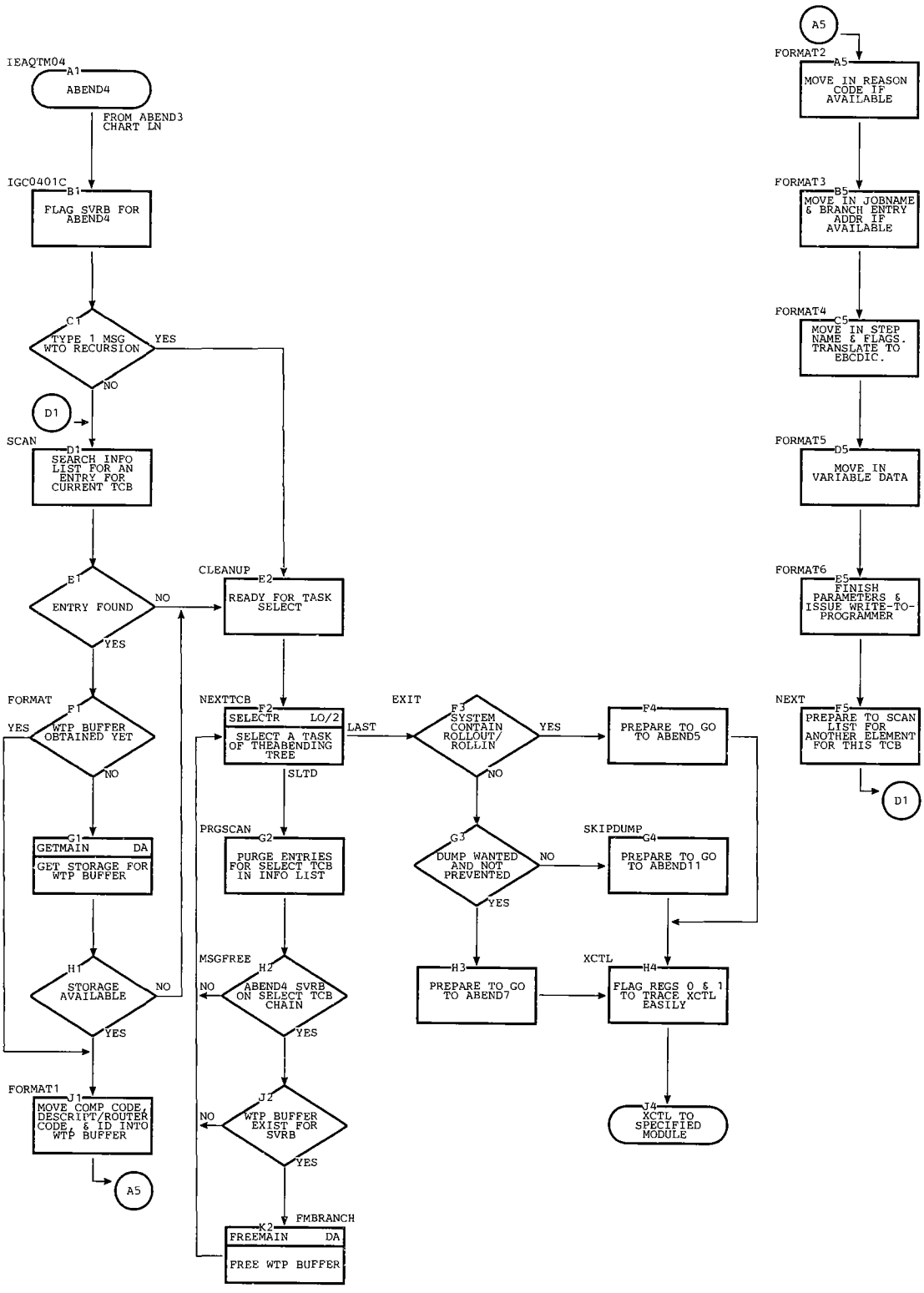


Chart LO. ABEND4 (Page 2 of 2)

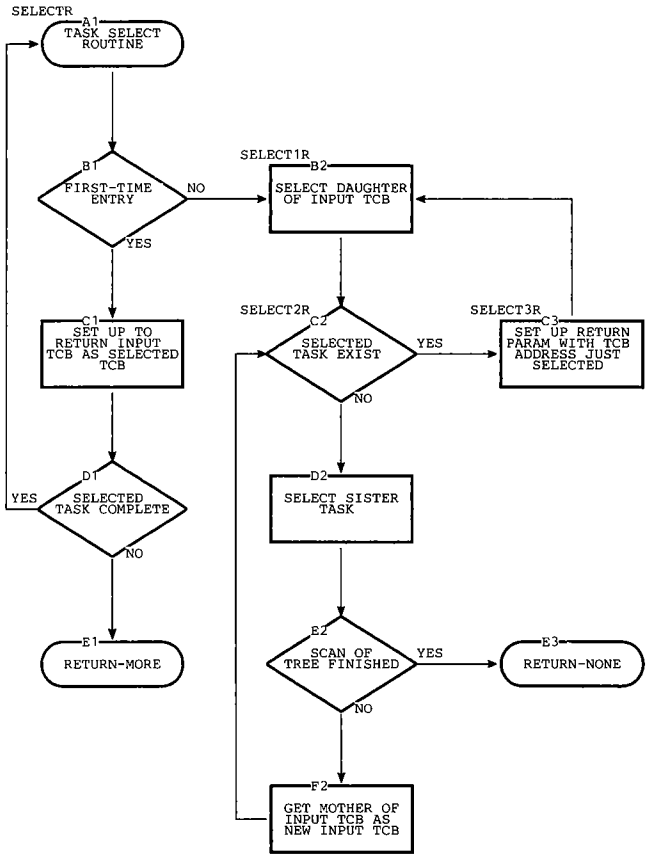


Chart LP. ABEND5 (Page 1 of 2)

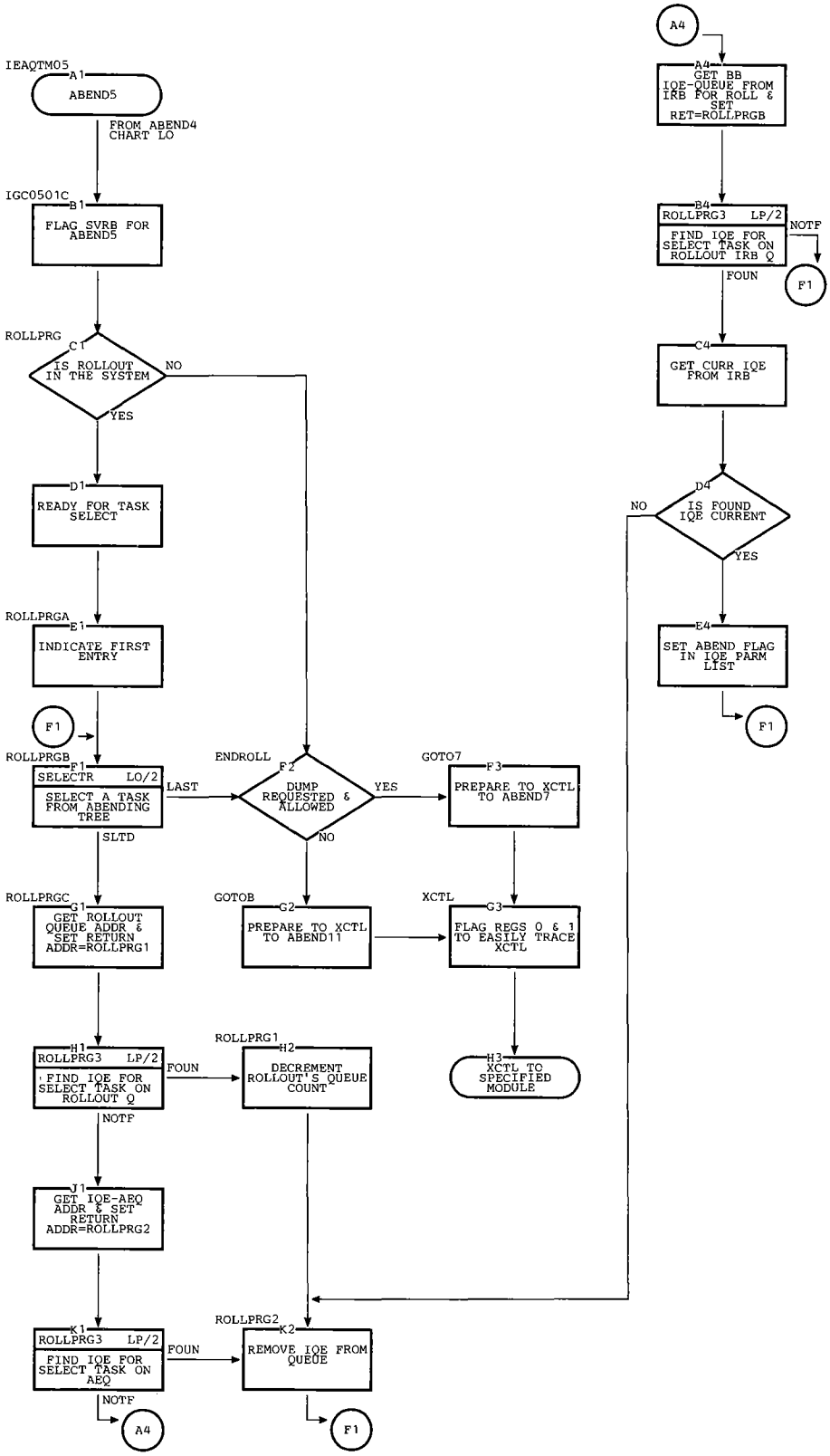


Chart LP. ABEND5 (Page 2 of 2)

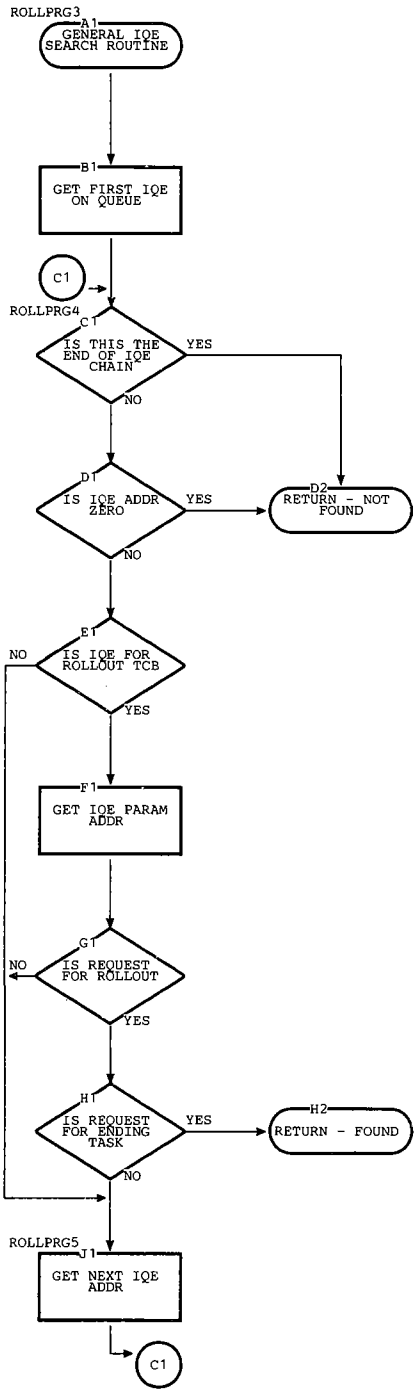


Chart LQ. ABEND7 (Page 1 of 2)

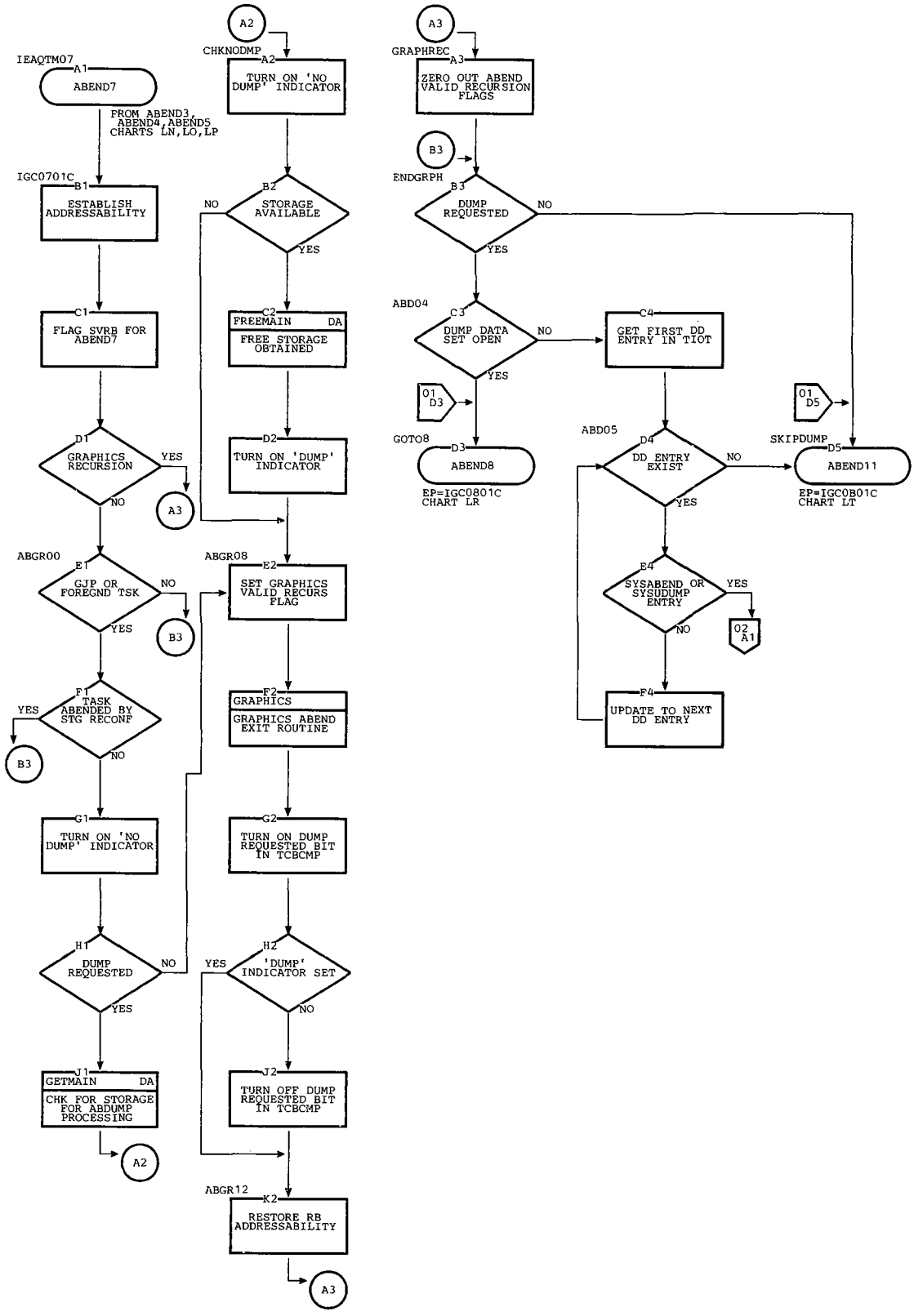


Chart LQ. ABEND7 (Page 2 of 2)

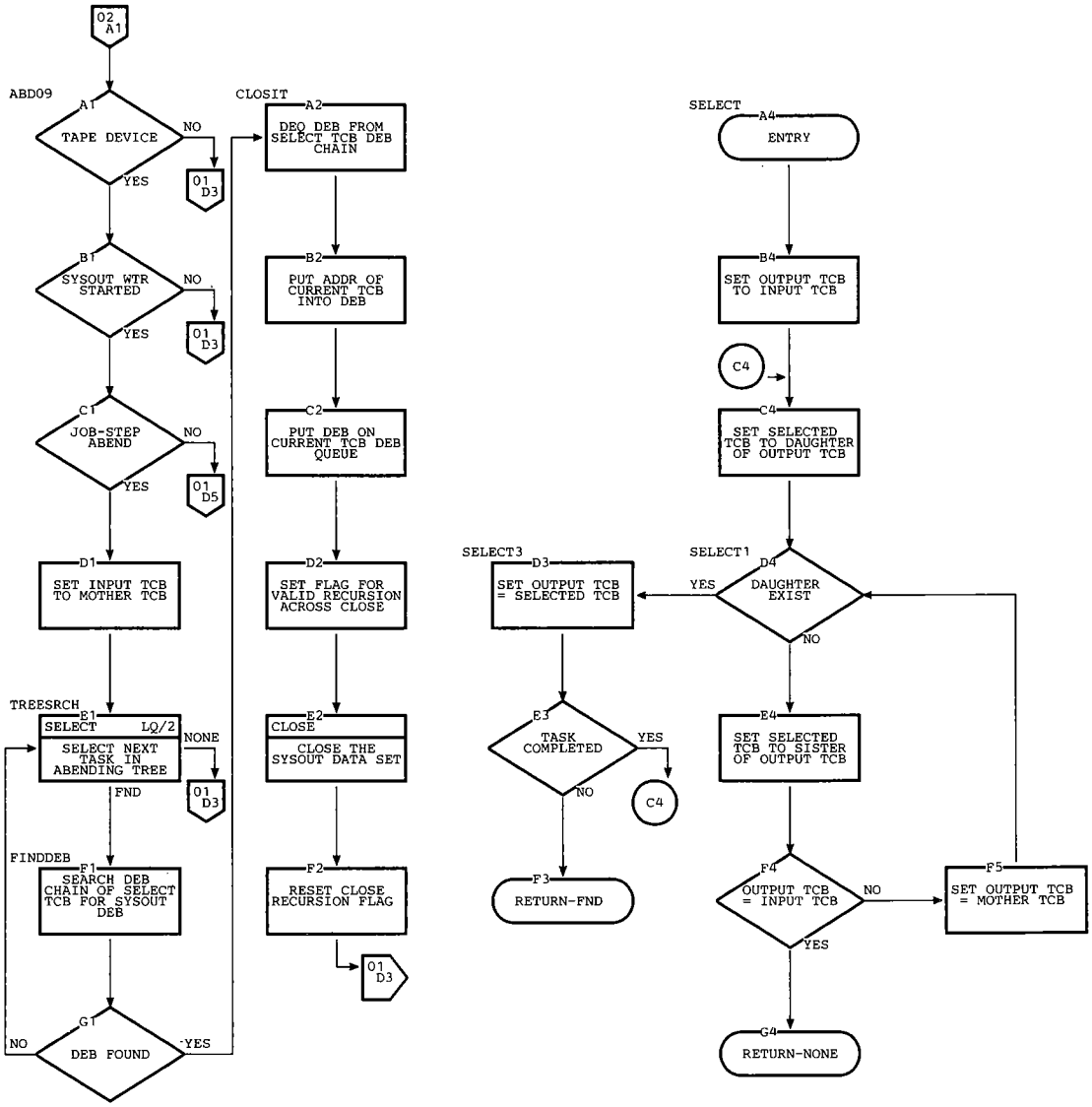


Chart LR. ABEND8 (Page 1 of 4)

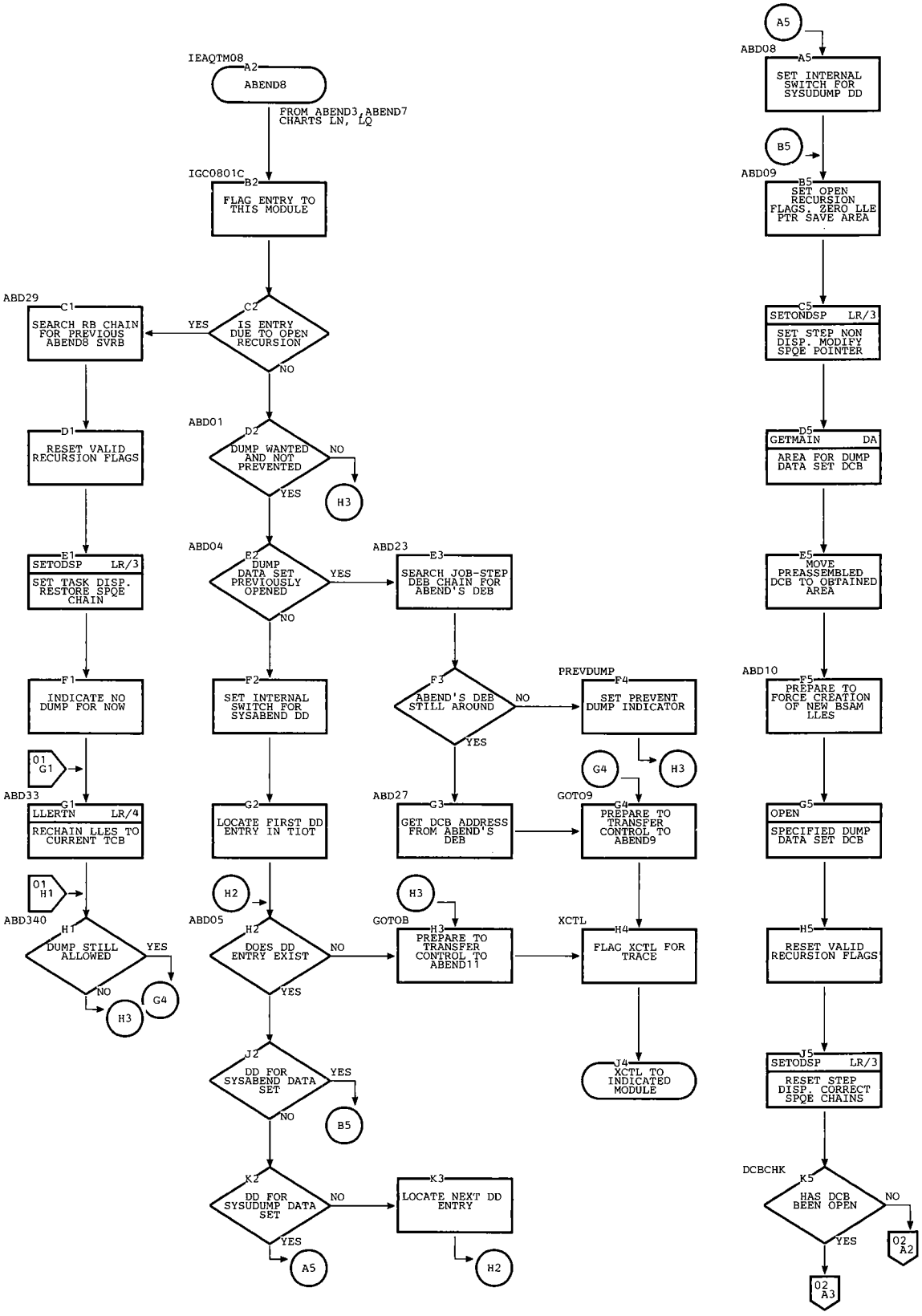


Chart LR. ABEND8 (Page 2 of 4)

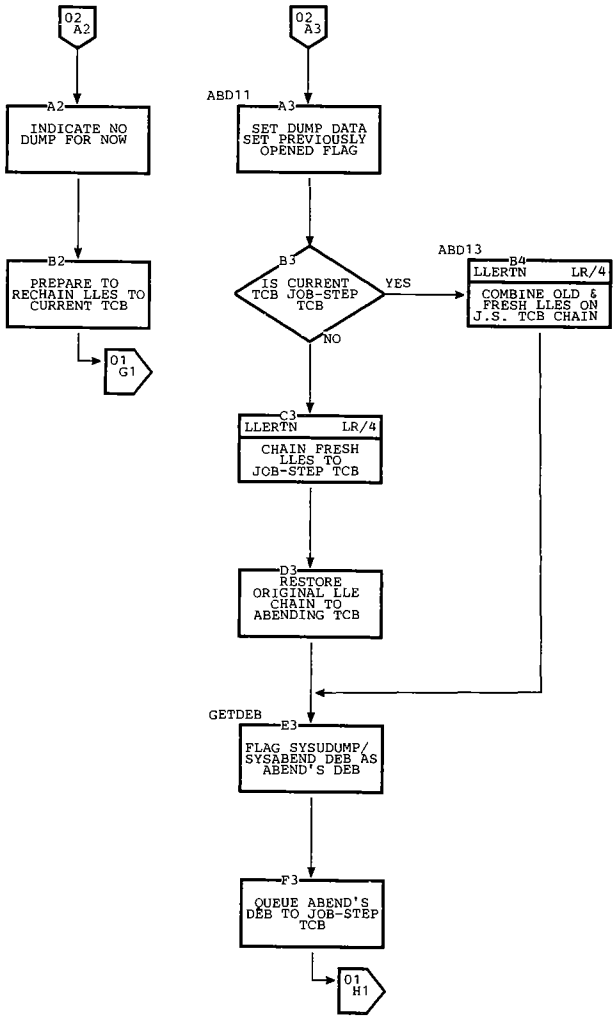


Chart LR. ABEND8 (Page 3 of 4)

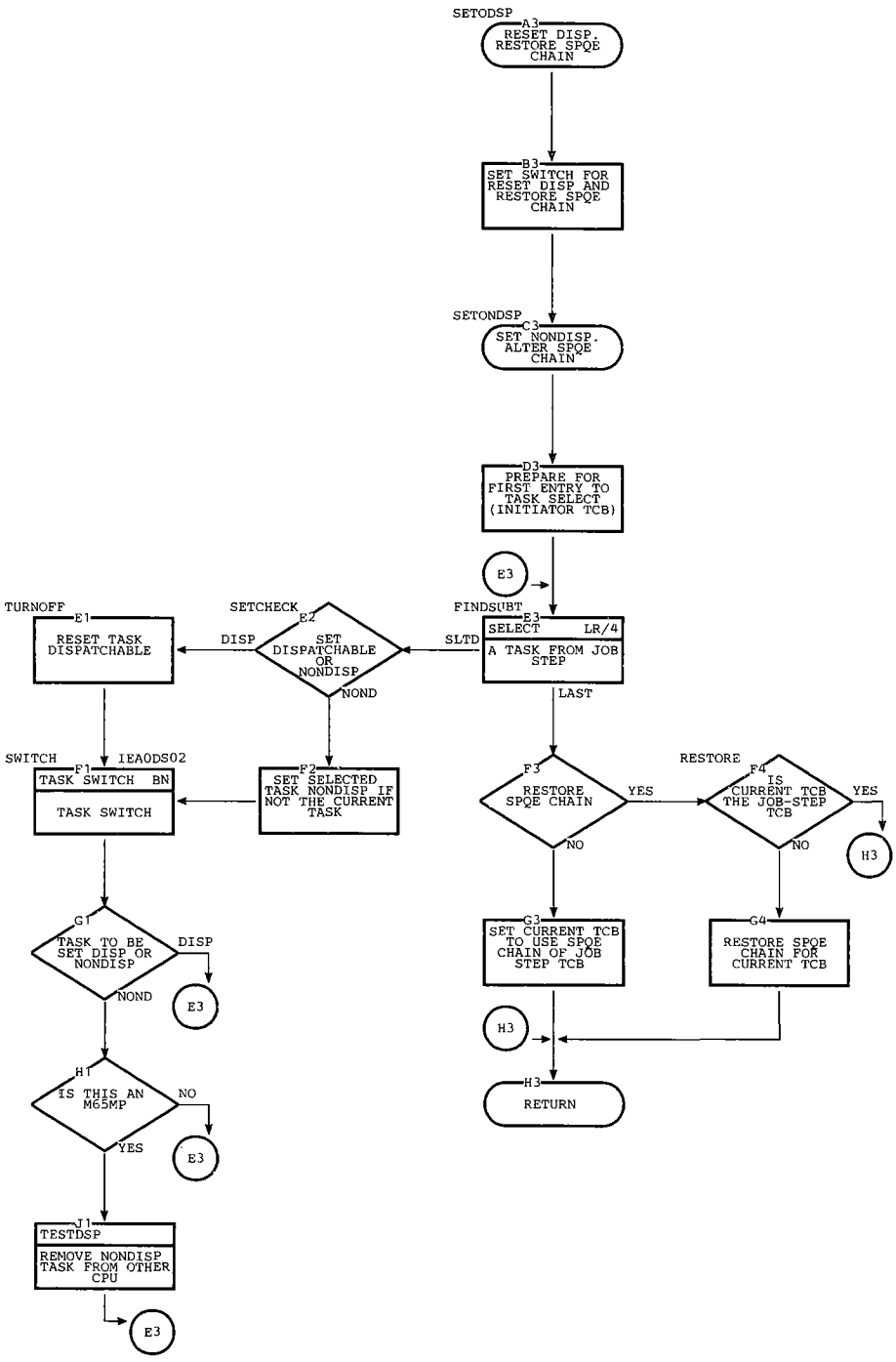


Chart LR. ABEND8 (Page 4 of 4)

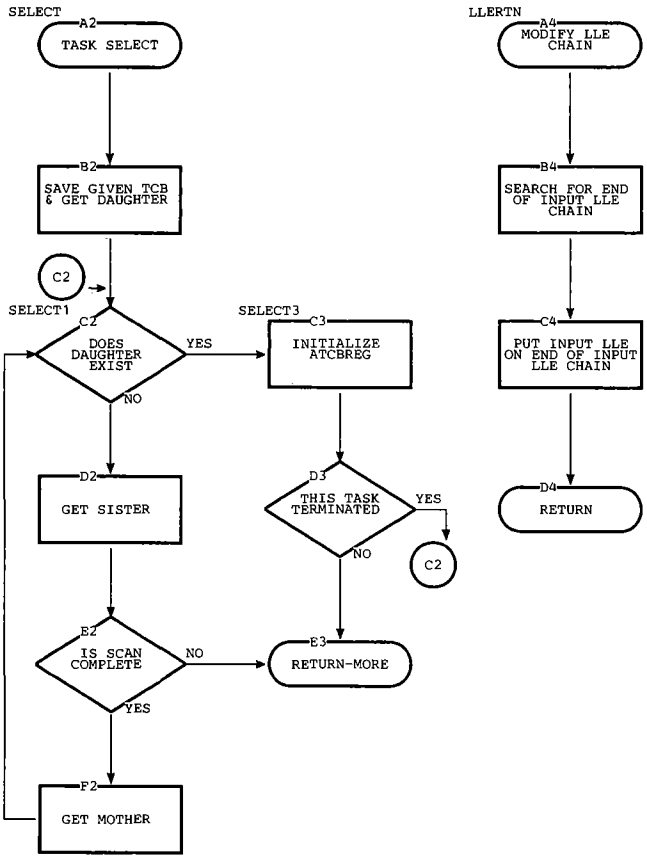


Chart LS. ABEND9 (Page 1 of 2)

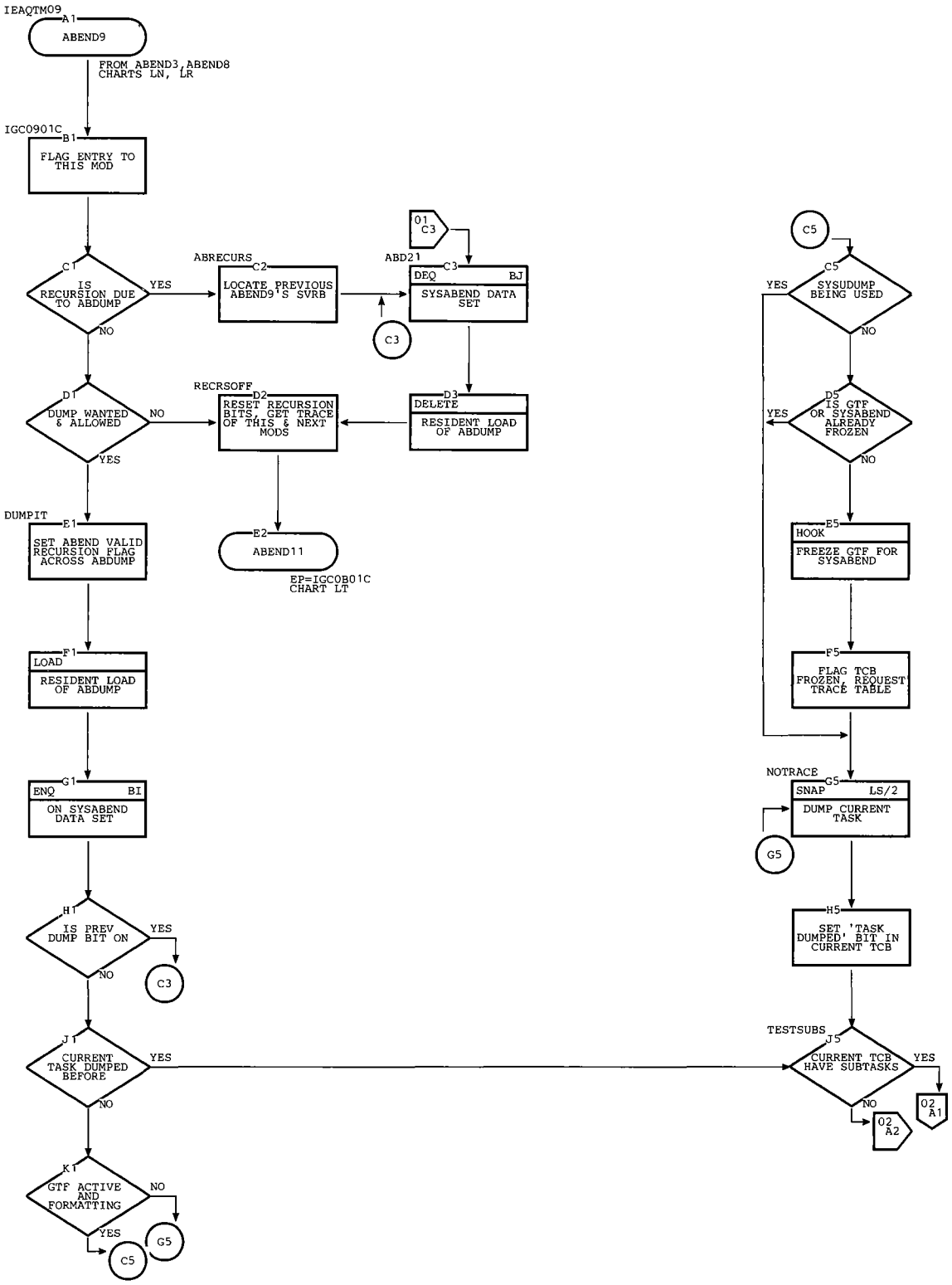


Chart LS. ABEND9 (Page 2 of 2)

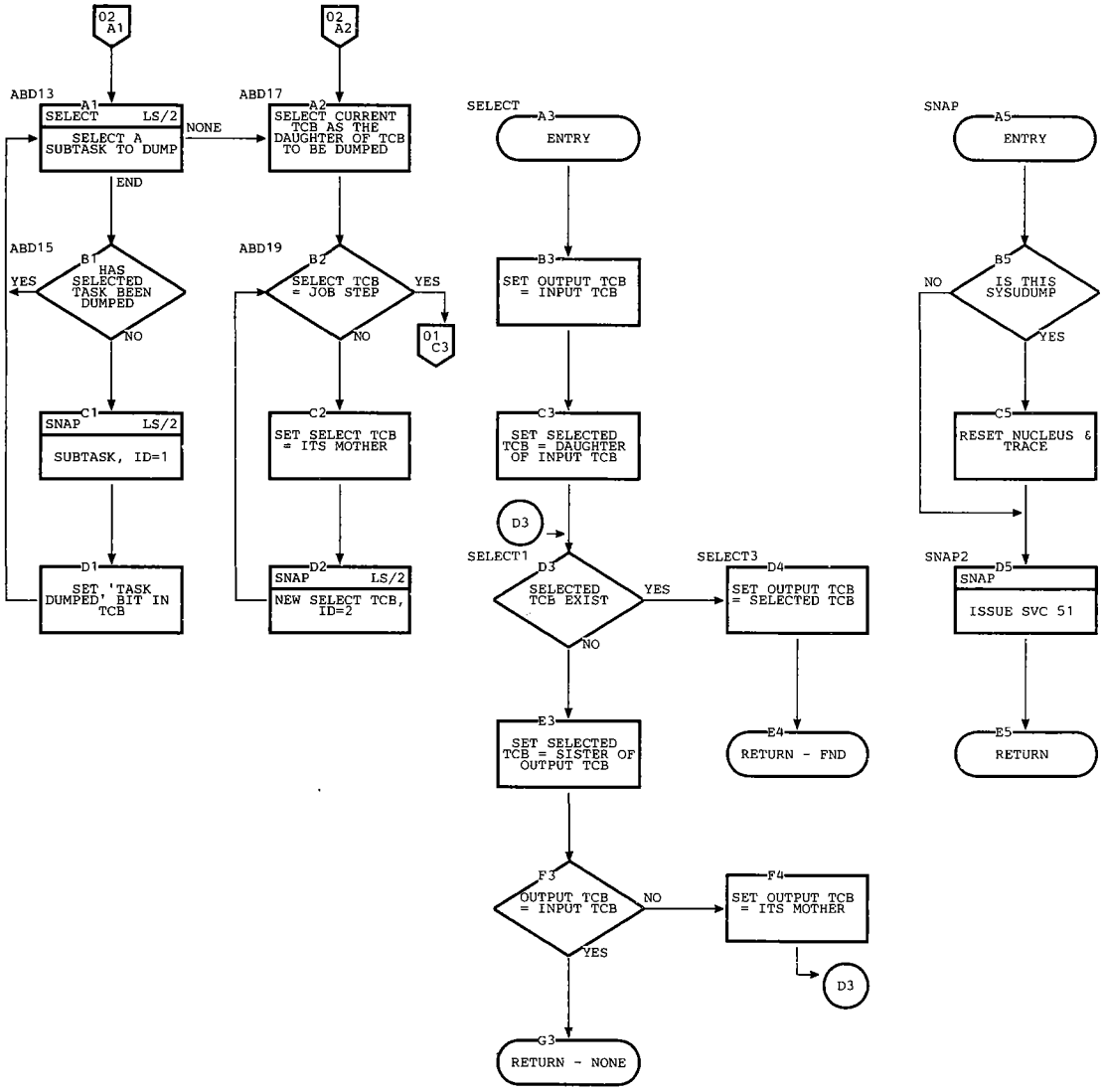


Chart LT. ABEND11 (Page 1 of 4)

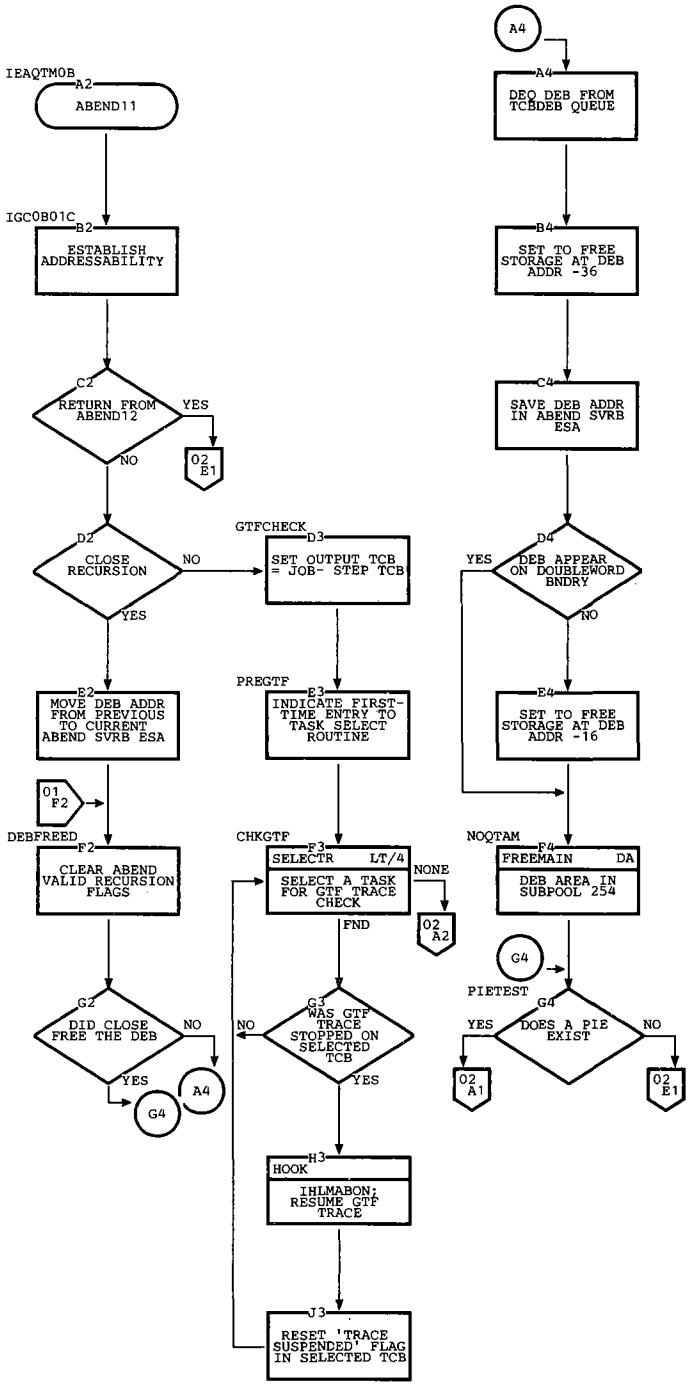


Chart LT. ABEND11 (Page 2 of 4)

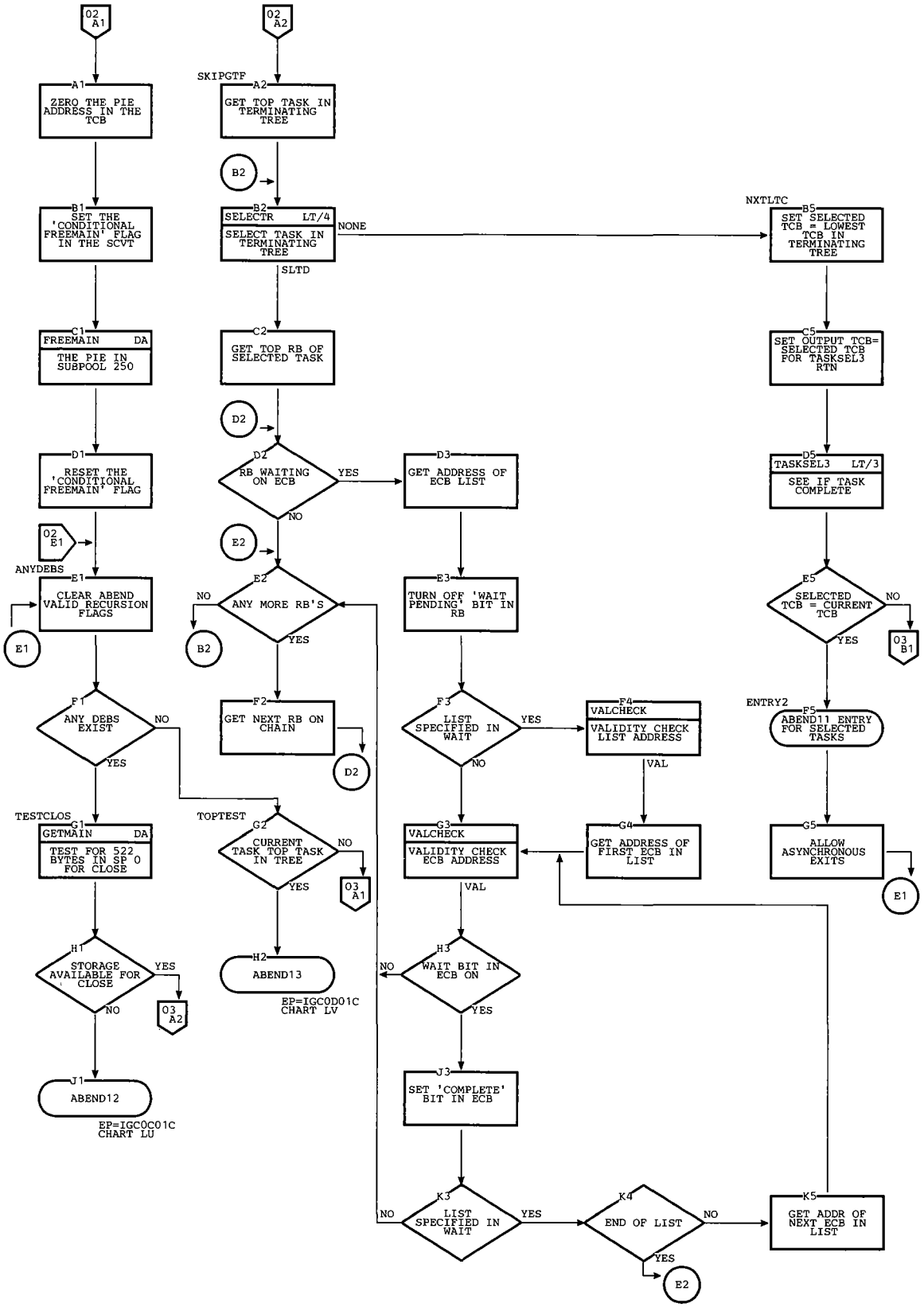


Chart LT. ABEND11 (Page 3 of 4)

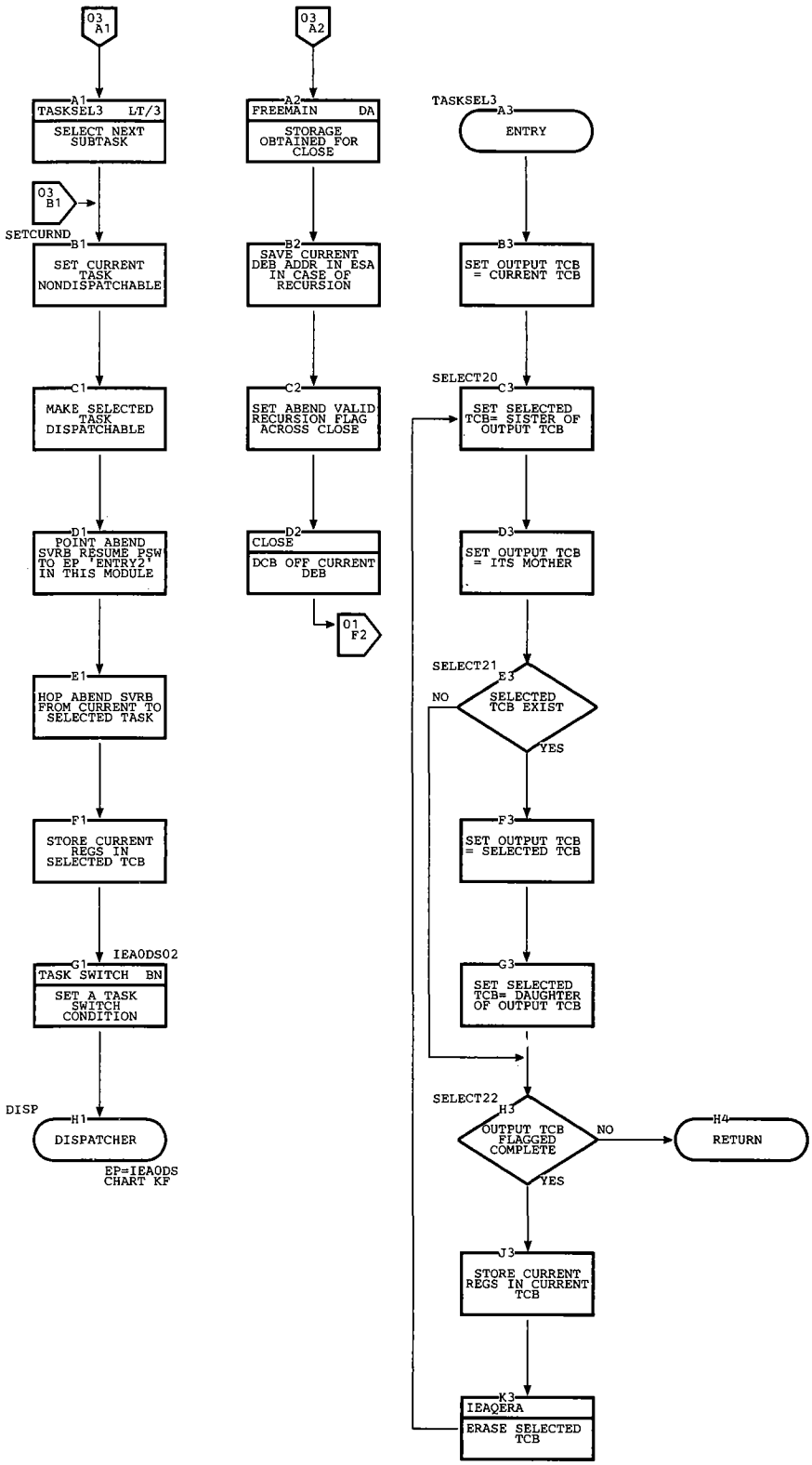


Chart LT. ABEND11 (Page 4 of 4)

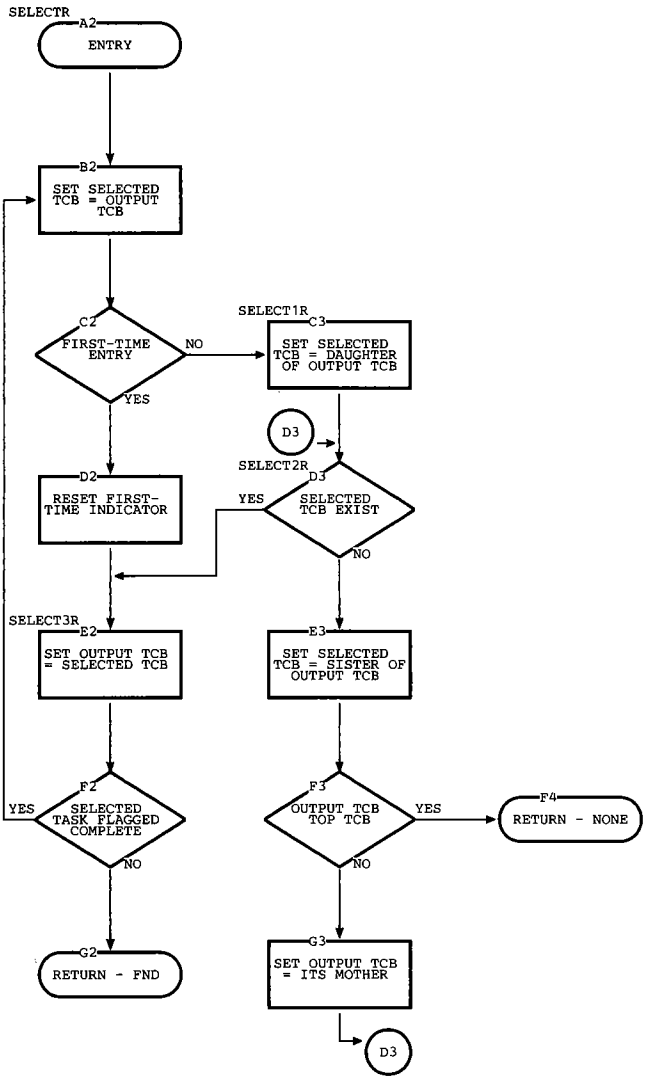
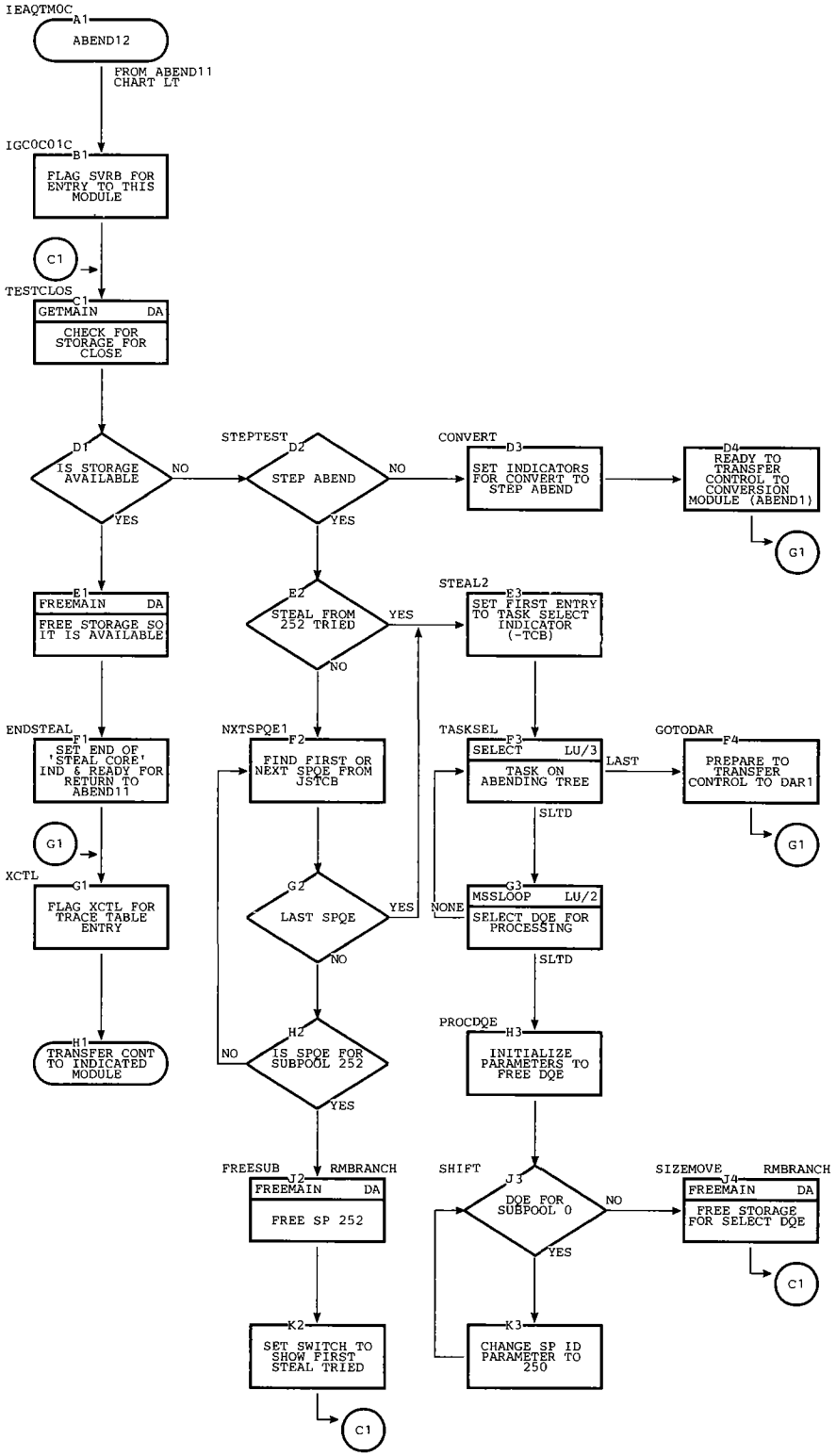


Chart LU. ABEND12 (Page 1 of 3)



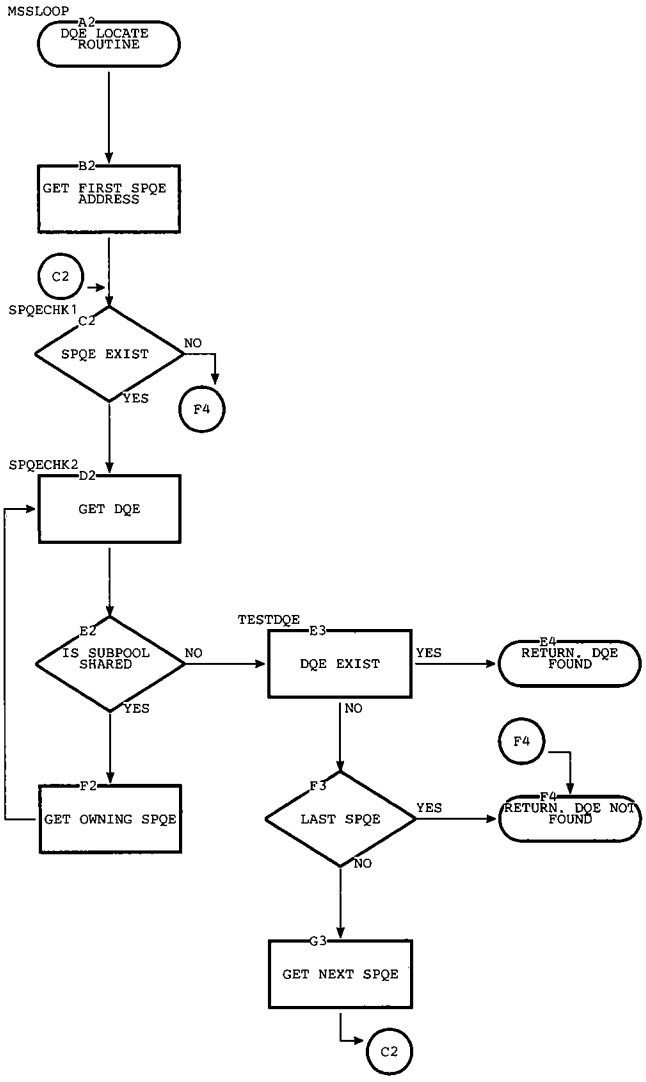

MSSLOOP
A2
DQE LOCATE ROUTINE
B2
GET FIRST SPQE ADDRESS
C2
SPQECHK1
C2
SPQE EXIST
NO
YES
F4
SPQECHK2
D2
GET DQE
E2
IS SUBPOOL SHARED
NO
YES
TESTDQE
E3
DQE EXIST
YES
E4
RETURN, DQE FOUND
NO
F4
F2
GET OWNING SPQE
F3
LAST SPQE
YES
F4
RETURN, DQE NOT FOUND
NO
G3
GET NEXT SPQE
C2

Chart LU. ABEND12 (Page 3 of 3)

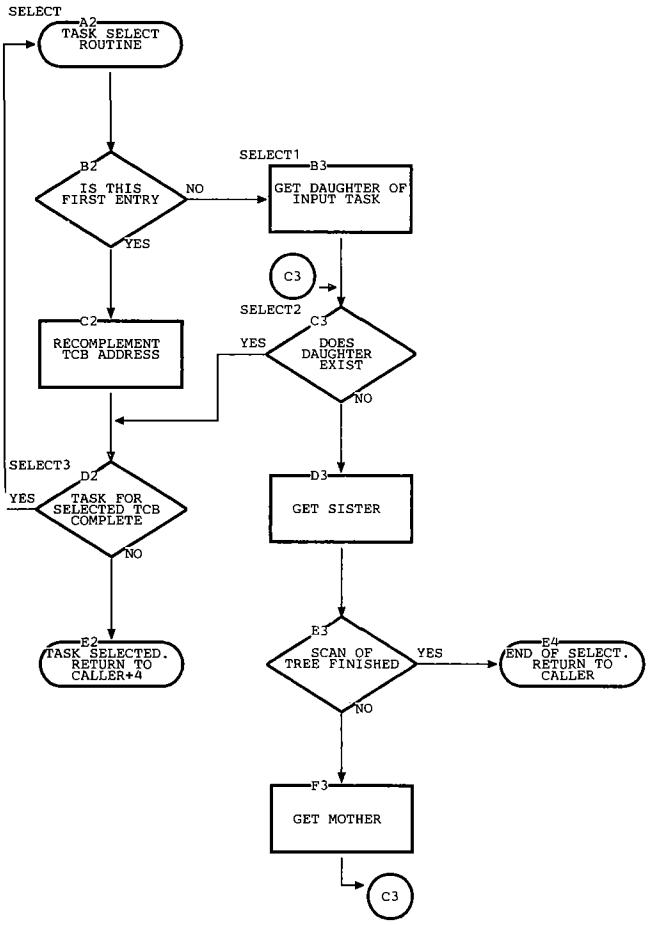


Chart LV. ABEND13 (Page 1 of 3)

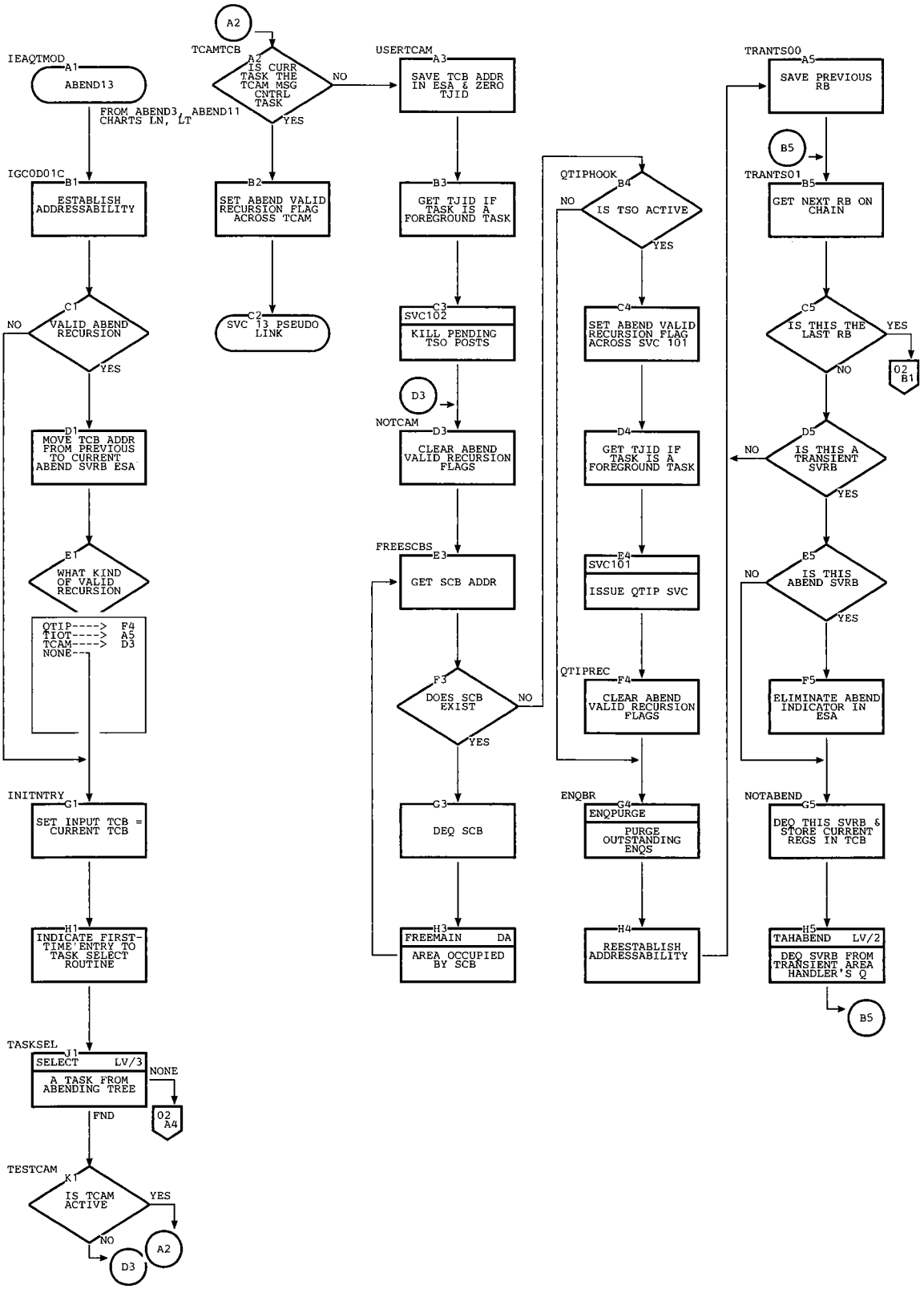


Chart LV. ABEND13 (Page 2 of 3)

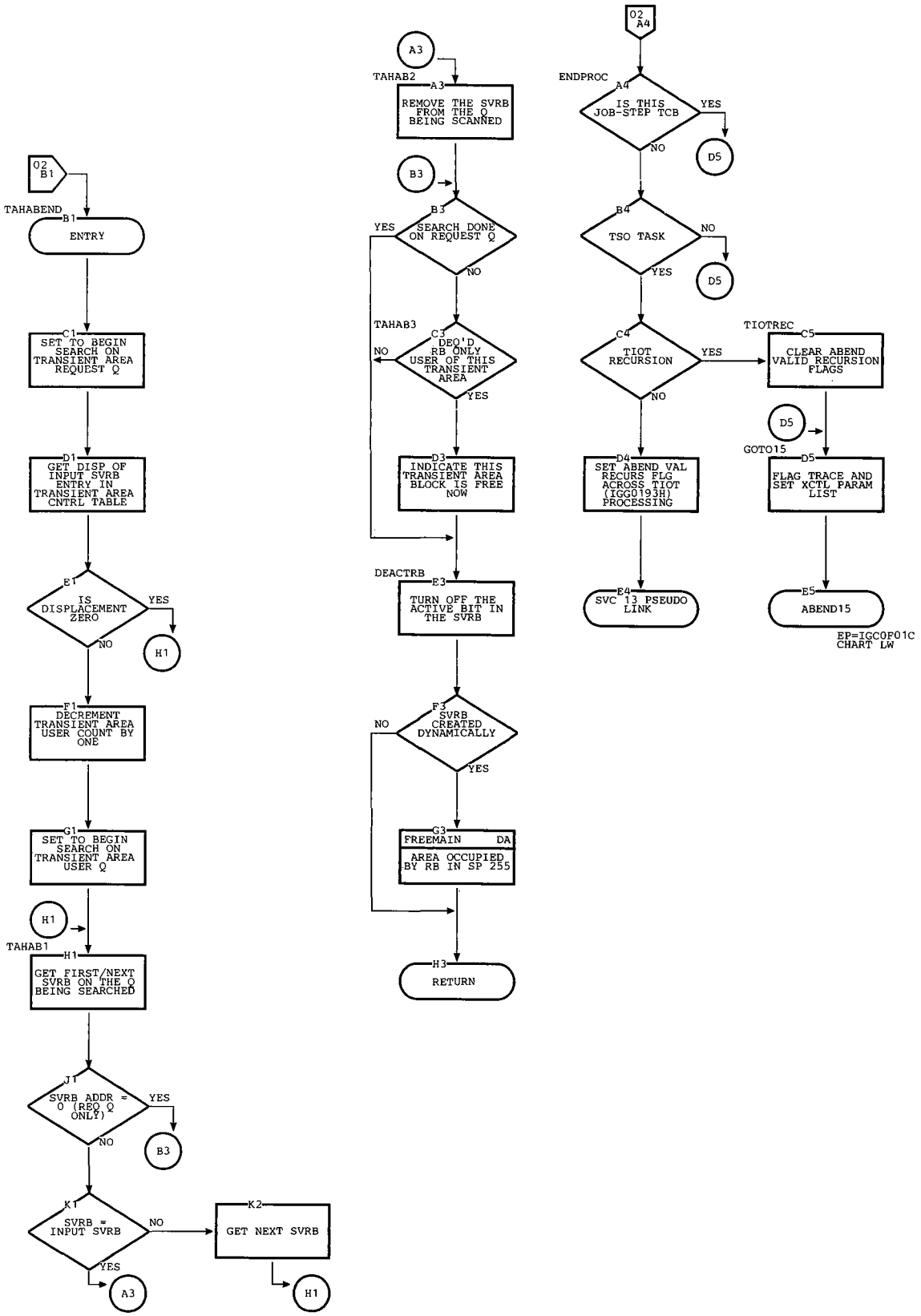


Chart LV. ABEND13 (Page 3 of 3)

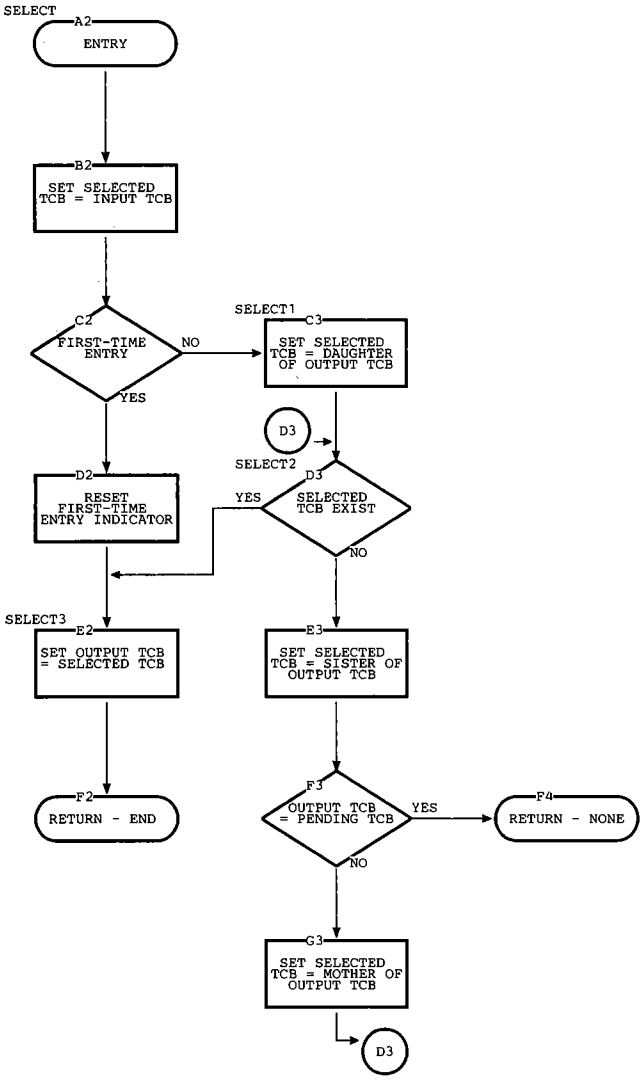


Chart LW. ABEND15 (Page 1 of 5)

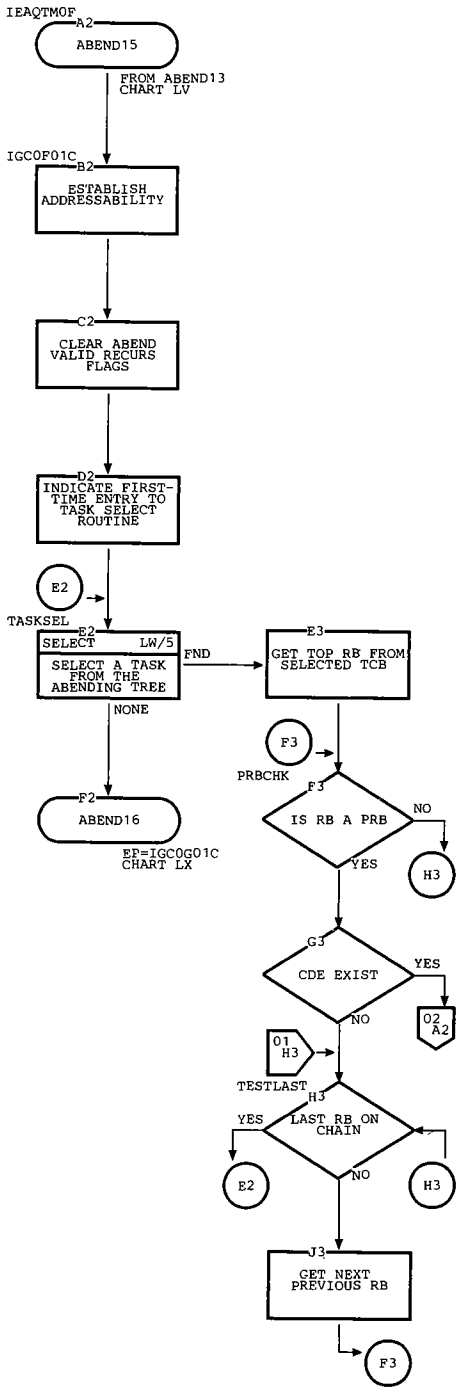


Chart LW. ABEND15 (Page 2 of 5)

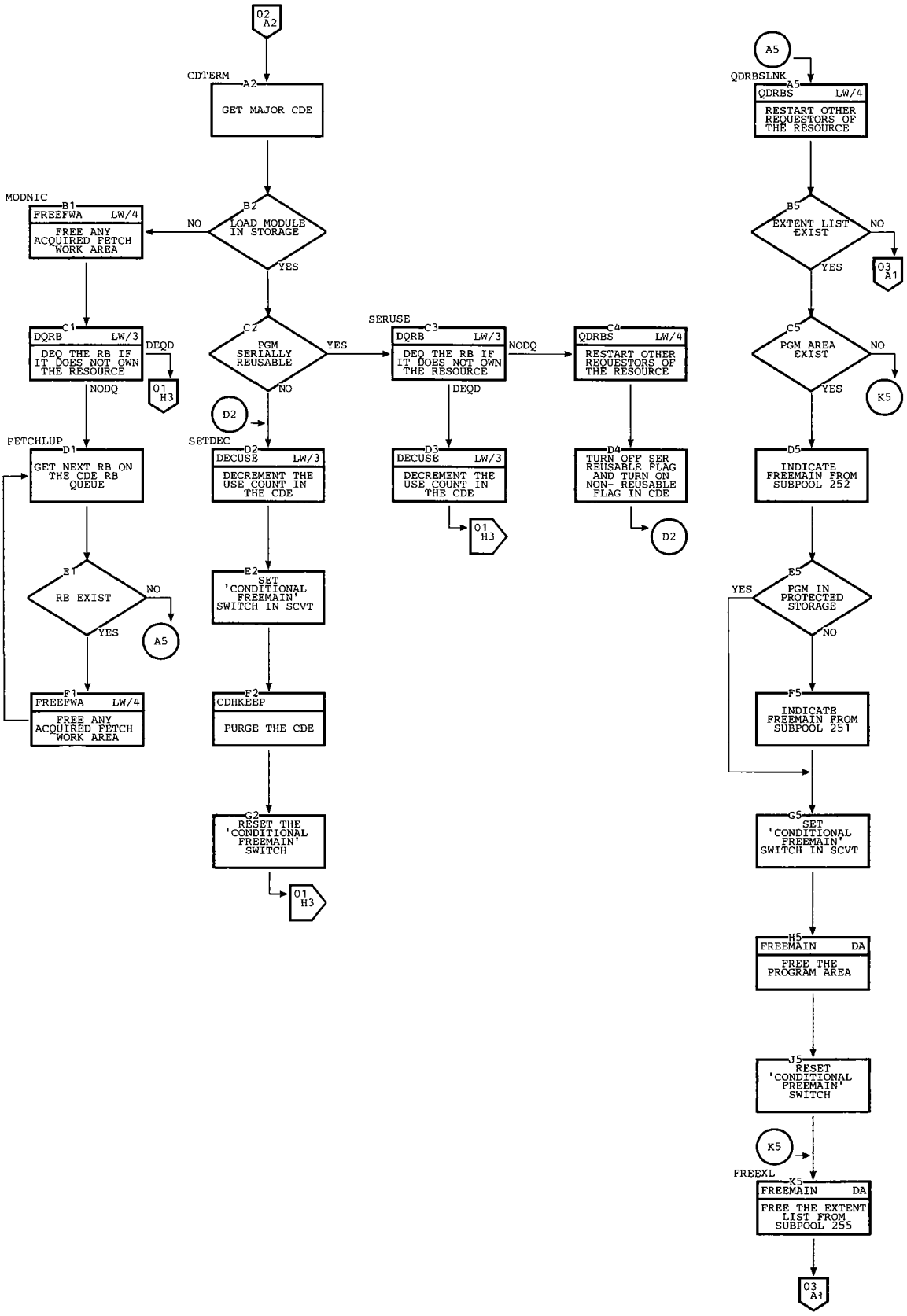


Chart LW. ABEND15 (Page 3 of 5)

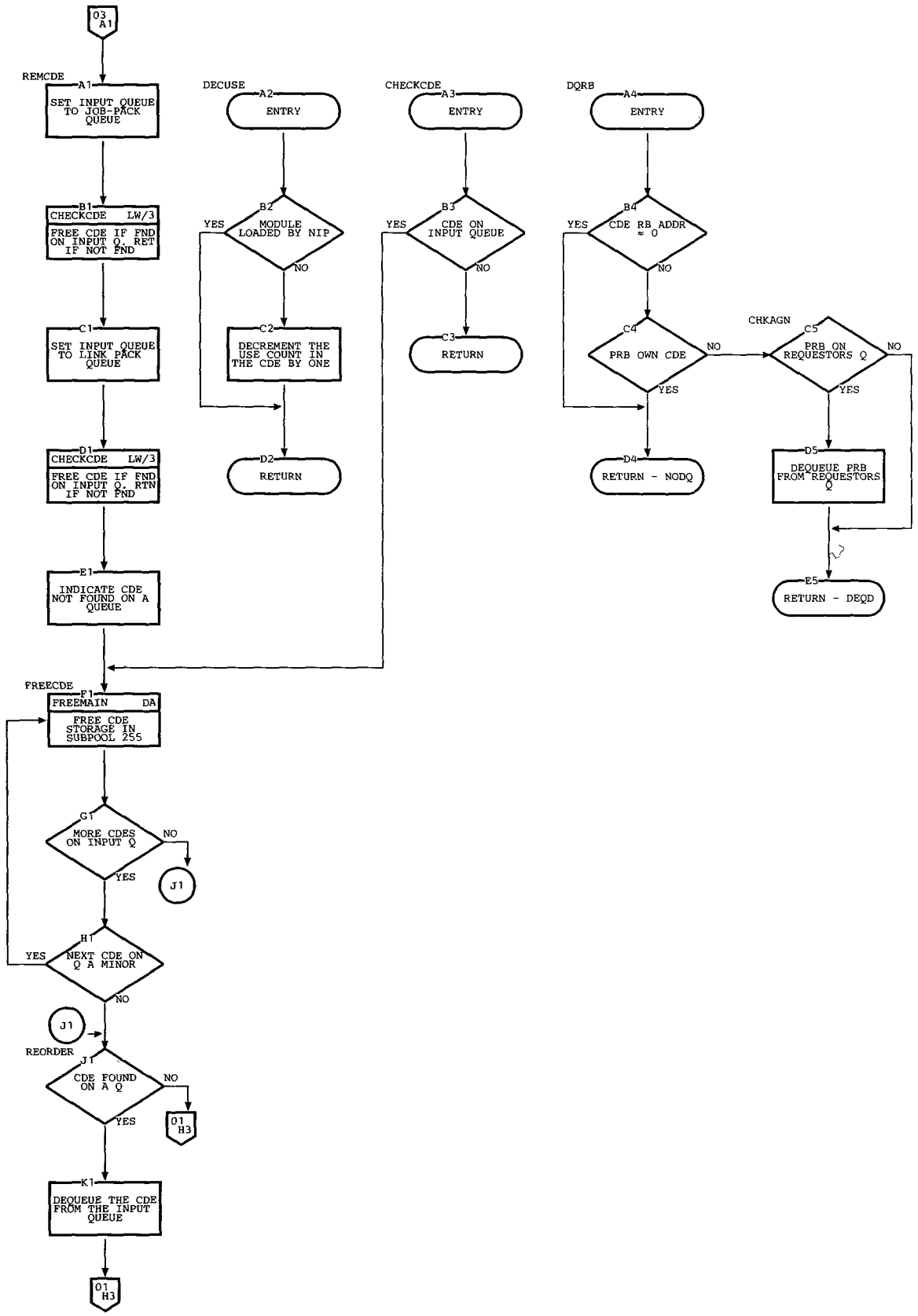


Chart LW. ABEND15 (Page 4 of 5)

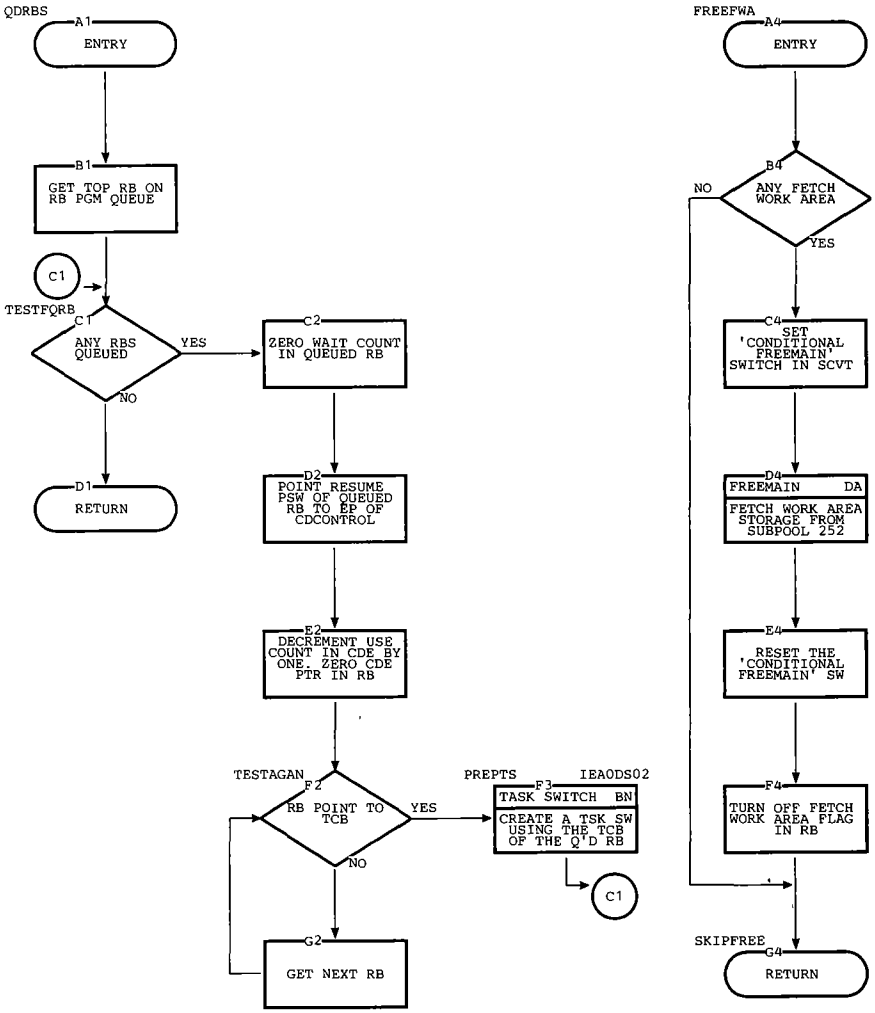


Chart LW. ABEND15 (Page 5 of 5)

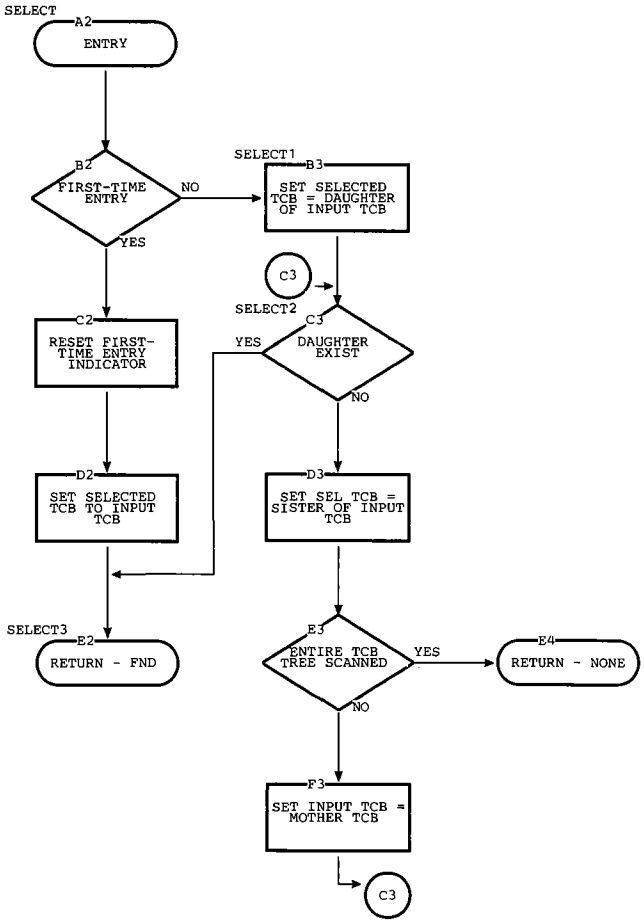


Chart LX. ABEND16 (Page 1 of 2)

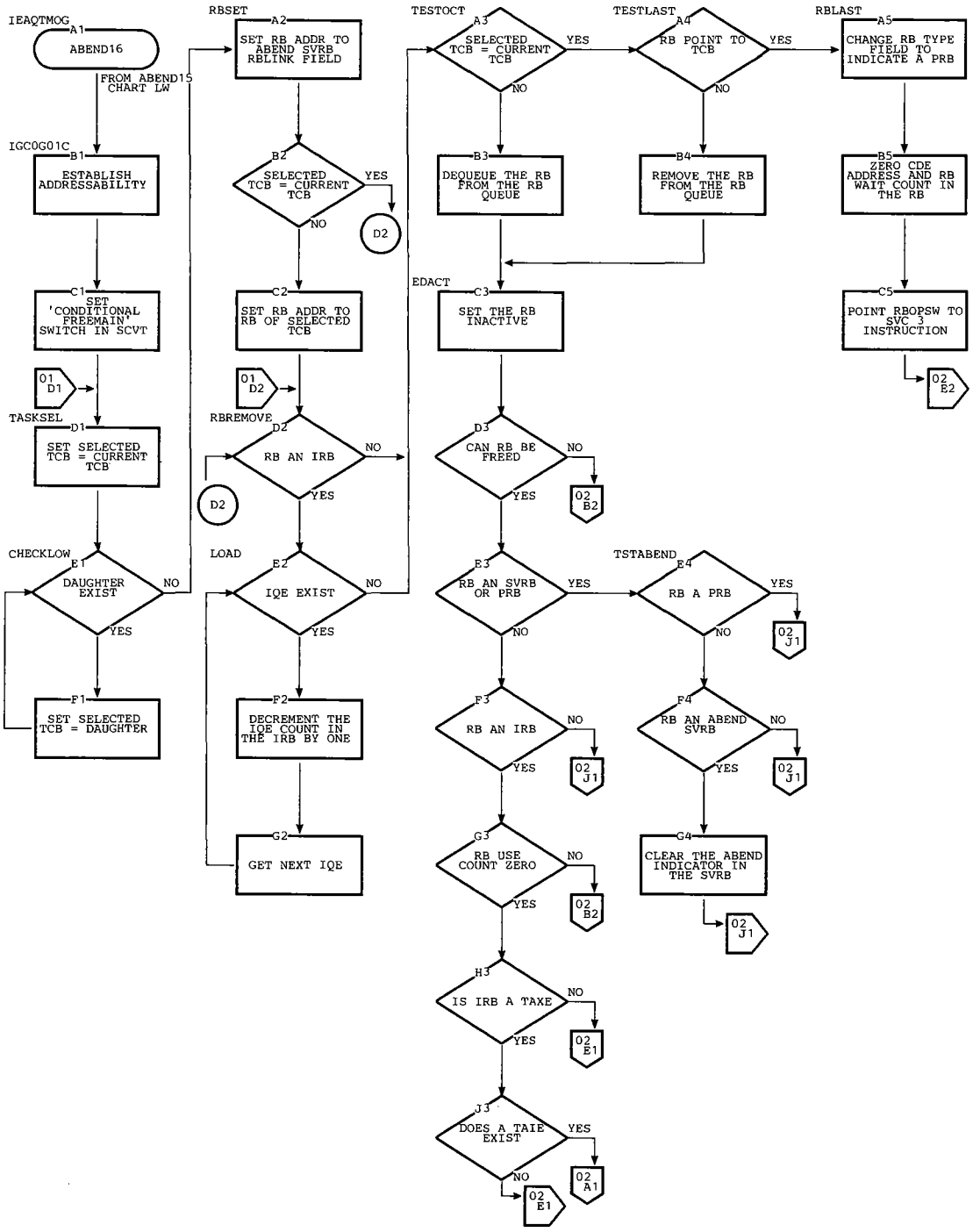


Chart LX. ABEND16 (Page 2 of 2)

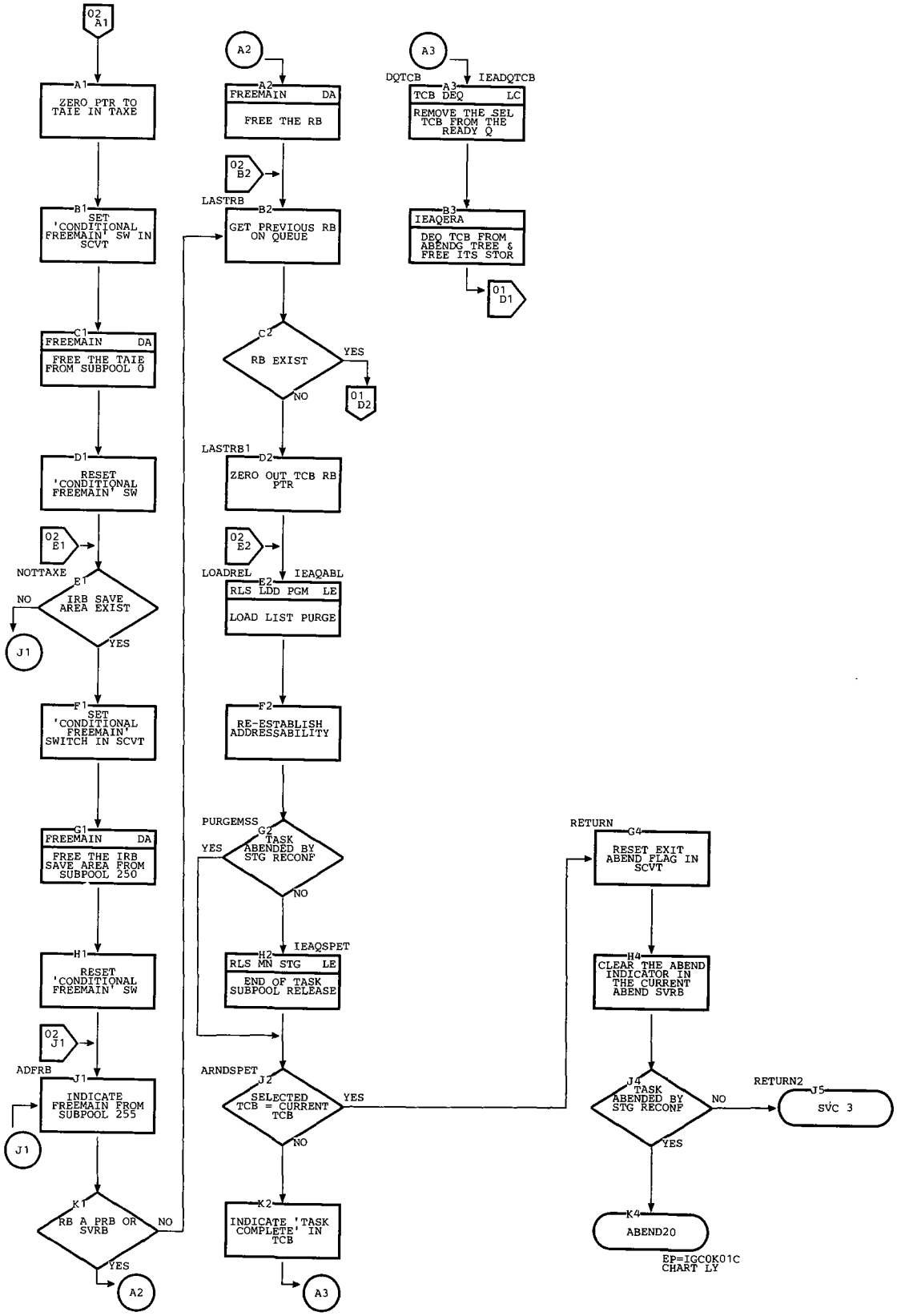


Chart LY. ABEND20

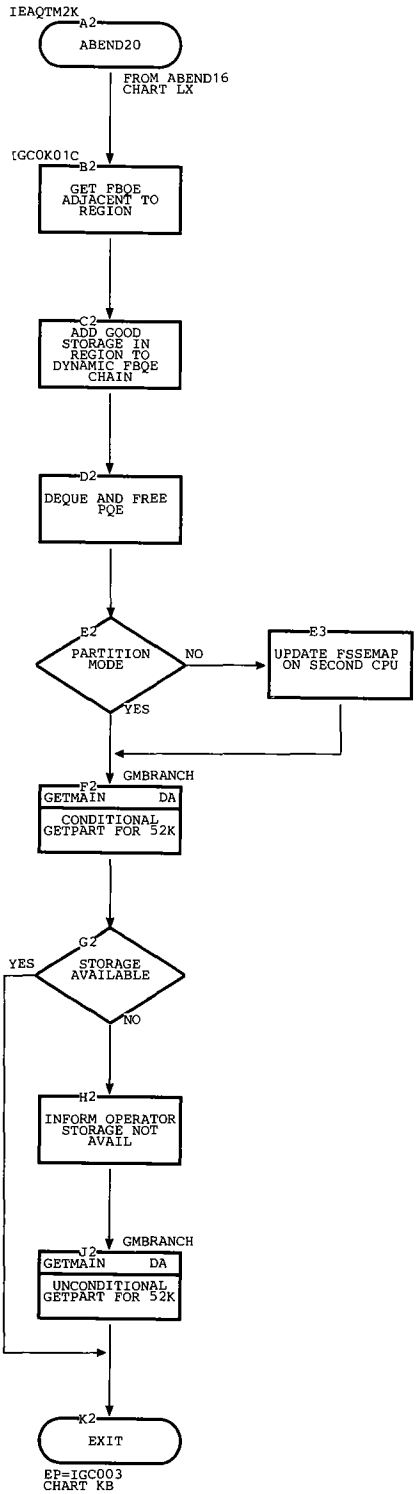


Chart MA. DAR1

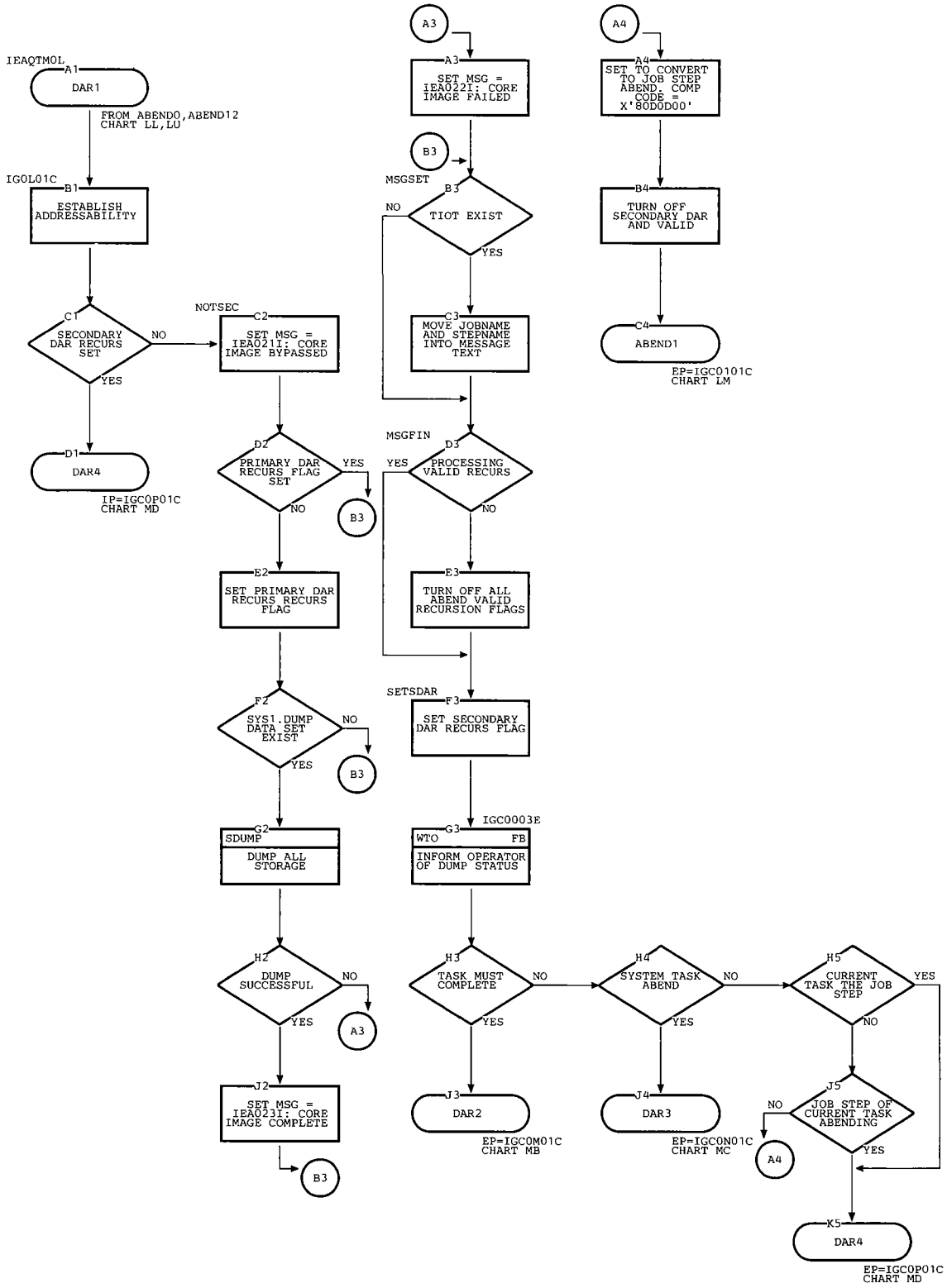


Chart MB. DAR2

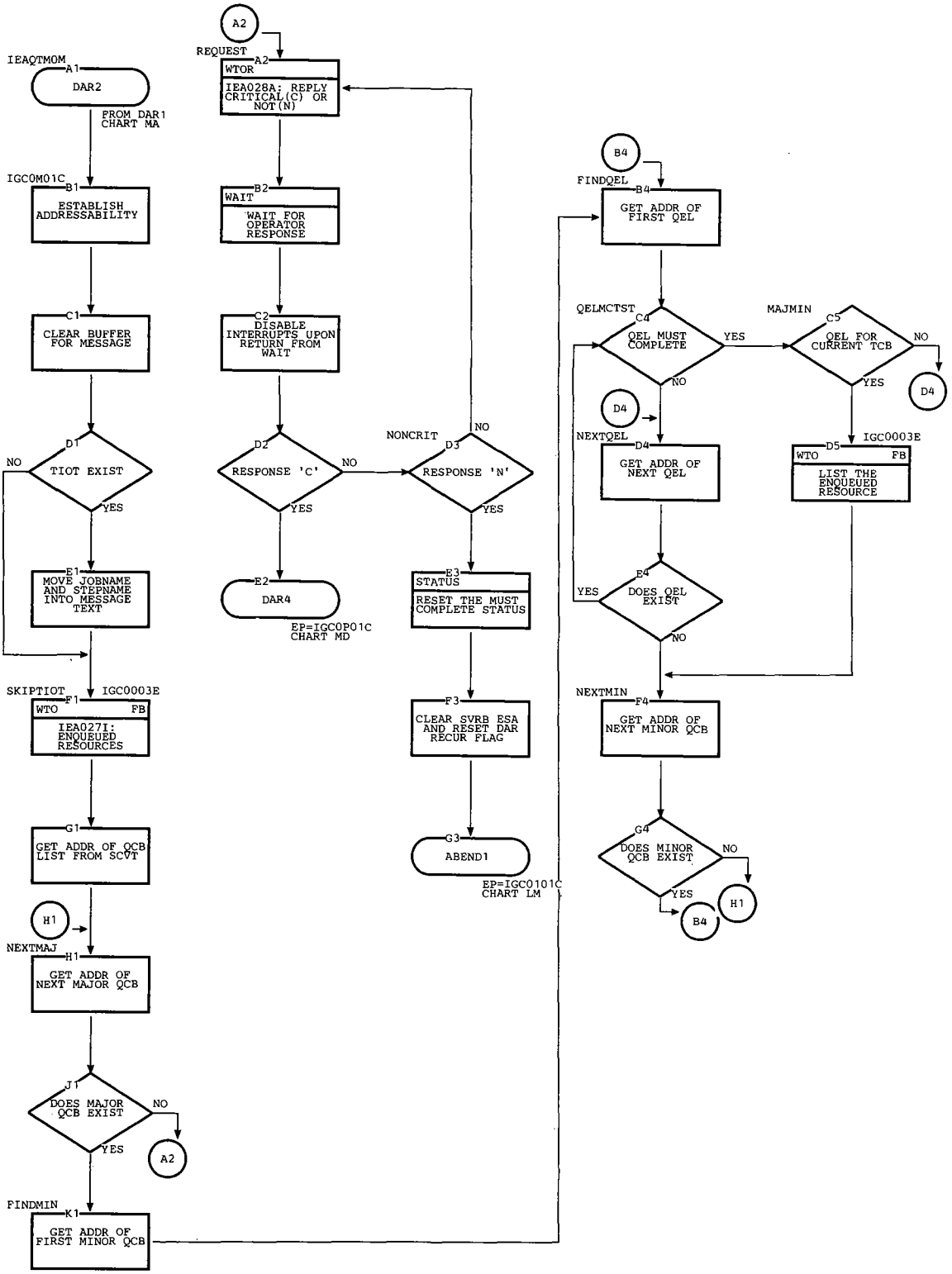


Chart MC. DAR3 (Page 1 of 3)

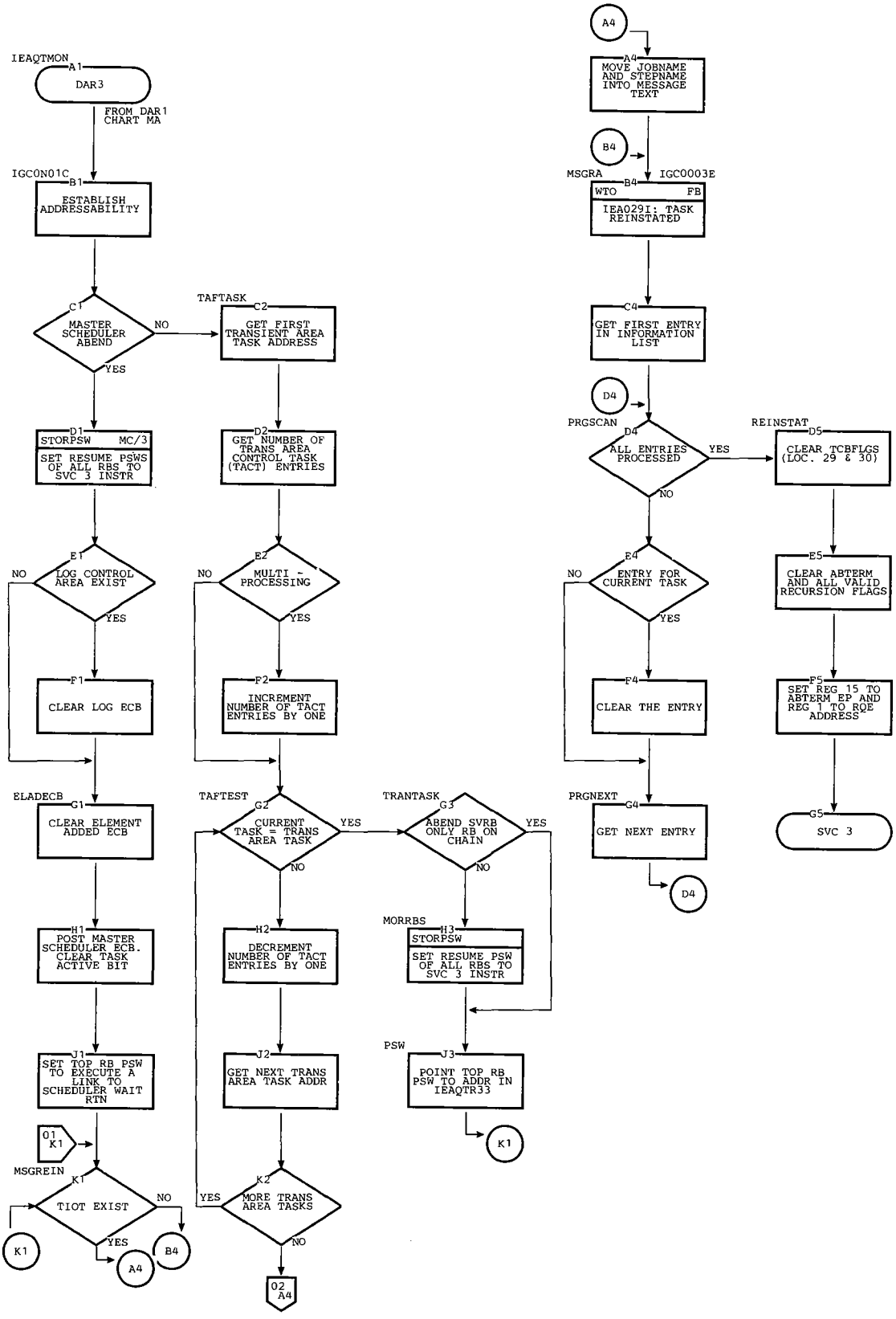


Chart MC. DAR3 (Page 2 of 3)

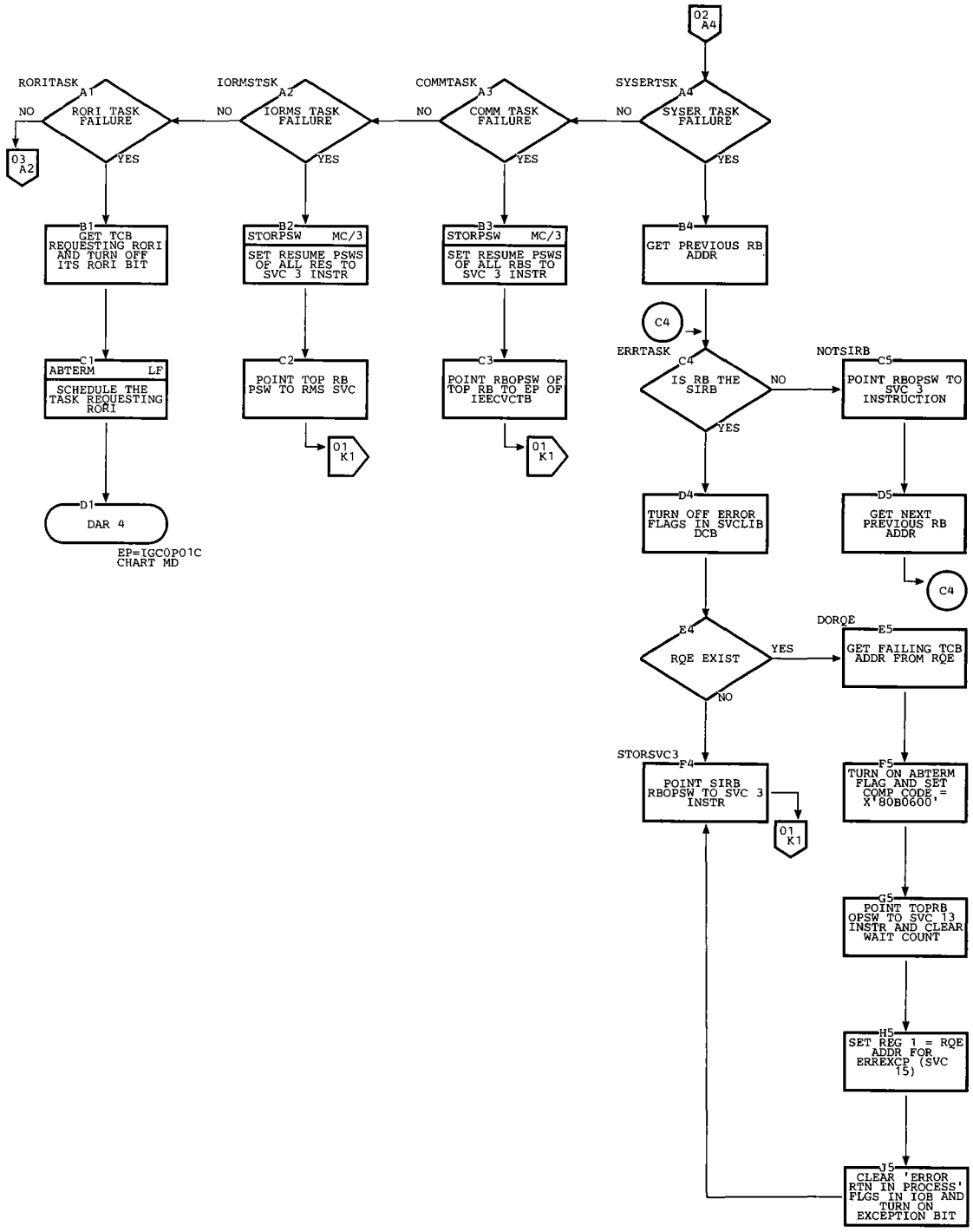
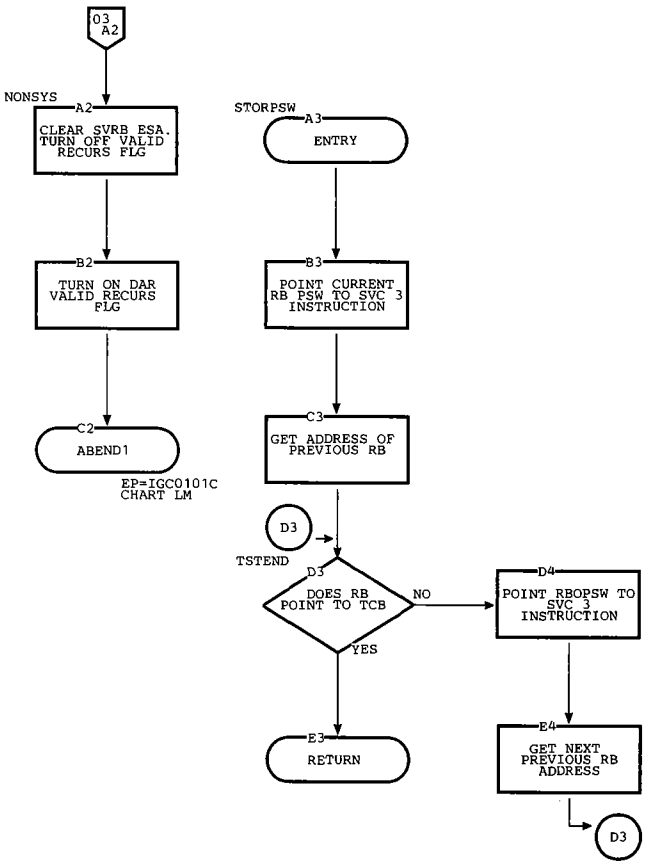


Chart MC. DAR3 (Page 3 of 3)



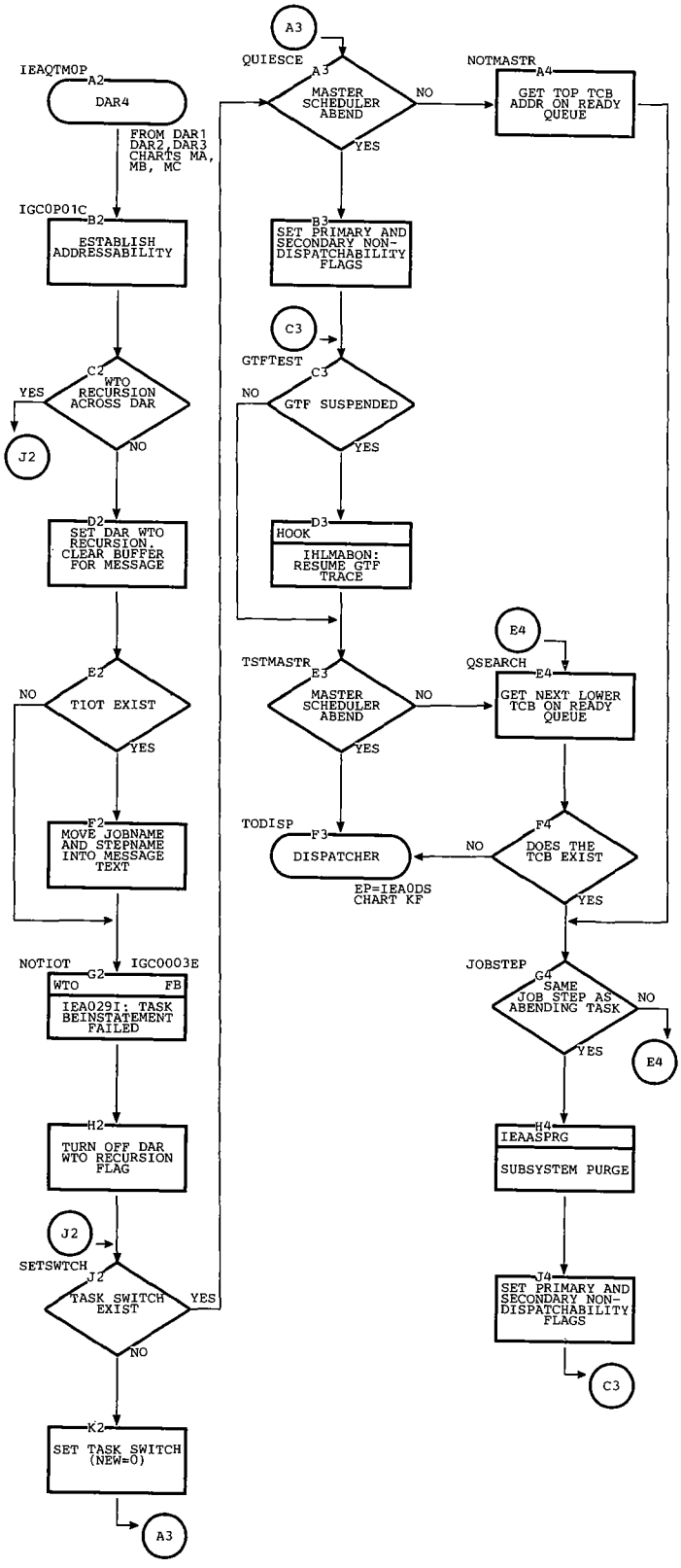

IEAQTM0P
A2
DAR4
FROM DAR1 DAR2 DAR3 CHARTS MA, MB, MC
IGC0P01C
B2
ESTABLISH ADDRESSABILITY
C2
WTO RECURSION ACROSS DAR
YES
NO
J2
D2
SET DAR WTO RECURSION, CLEAR BUFFER FOR MESSAGE
E2
TIOT EXIST
NO
YES
F2
MOVE JOBNAME AND STEPNAME INTO MESSAGE TEXT
NOTIOT
IGC0003E
G2
WTO FB
IEA029I: TASK BEINSTATEMENT FAILED
H2
TURN OFF DAR WTO RECURSION FLAG
J2
SETSWTCH
J2
TASK SWITCH EXIST
YES
NO
K2
SET TASK SWITCH (NEW=0)
A3
A3
QUIESCE
A3
MASTER SCHEDULER ABEND
NO
YES
NOTMASTR
A4
GET TOP TCB ADDR ON READY QUEUE
B3
SET PRIMARY AND SECONDARY NON-DISPATCHABILITY FLAGS
C3
GTFTEST
C3
GTF SUSPENDED
NO
YES
D3
HOOK
IHLMABON: RESUME GTF TRACE
E4
TSTMASTR
E3
MASTER SCHEDULER ABEND
NO
YES
QSEARCH
E4
GET NEXT LOWER TCB ON READY QUEUE
TODISP
F3
DISPATCHER
EP=IEA0DS
CHART KF
F4
DOES THE TCB EXIST
NO
YES
JOBSTEP
G4
SAME JOB STEP AS ABENDING TASK
NO
YES
E4
H4
IEAASPRG
SUBSYSTEM PURGE
J4
SET PRIMARY AND SECONDARY NON-DISPATCHABILITY FLAGS
C3

Chart ME. SVC DUMP 1 (Page 1 of 2)

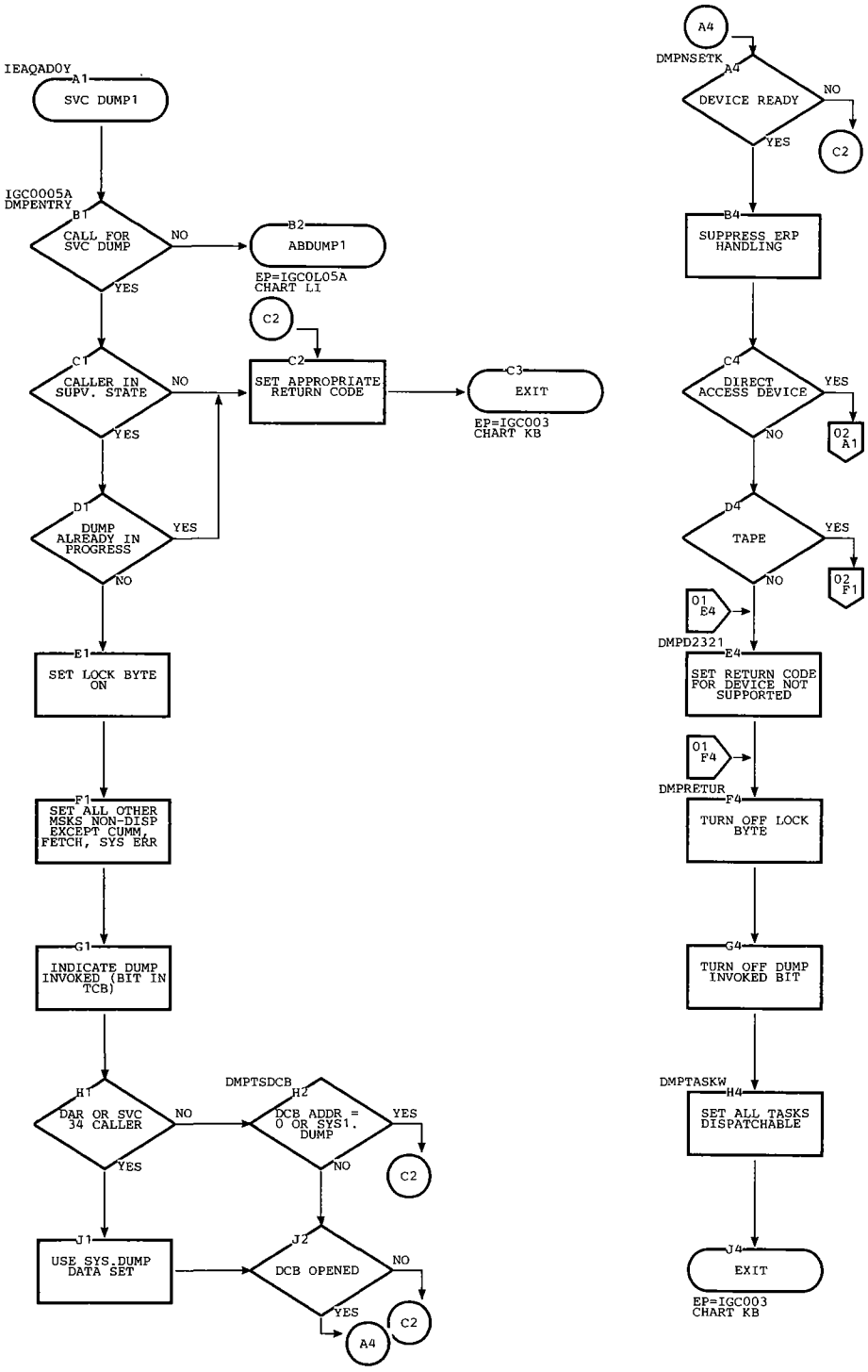


Chart ME. SVC DUMP 1 (Page 2 of 2)

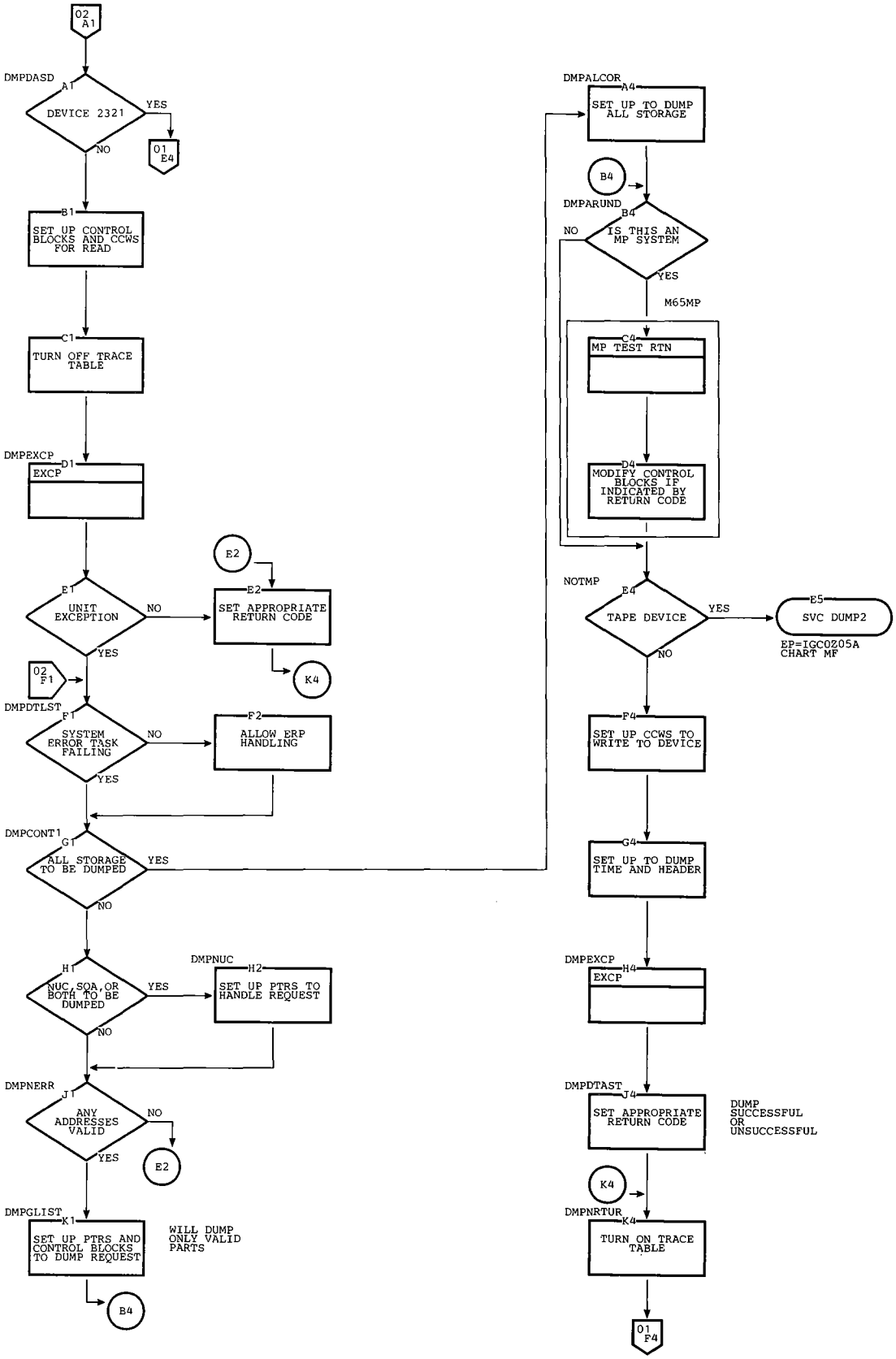


Chart MF. SVC DUMP 2

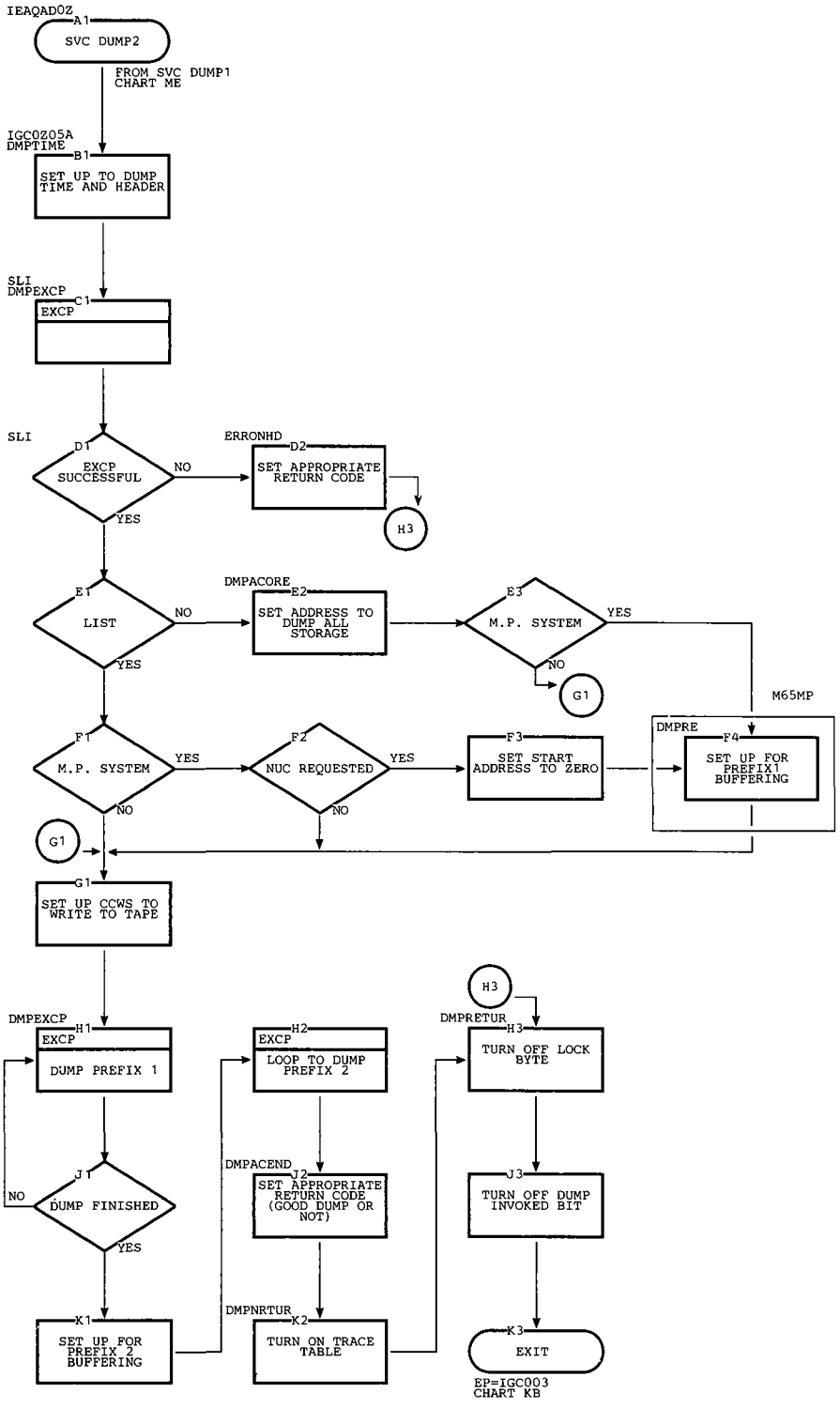


Chart MG. Models 91 and 195 Simulator Control Routine

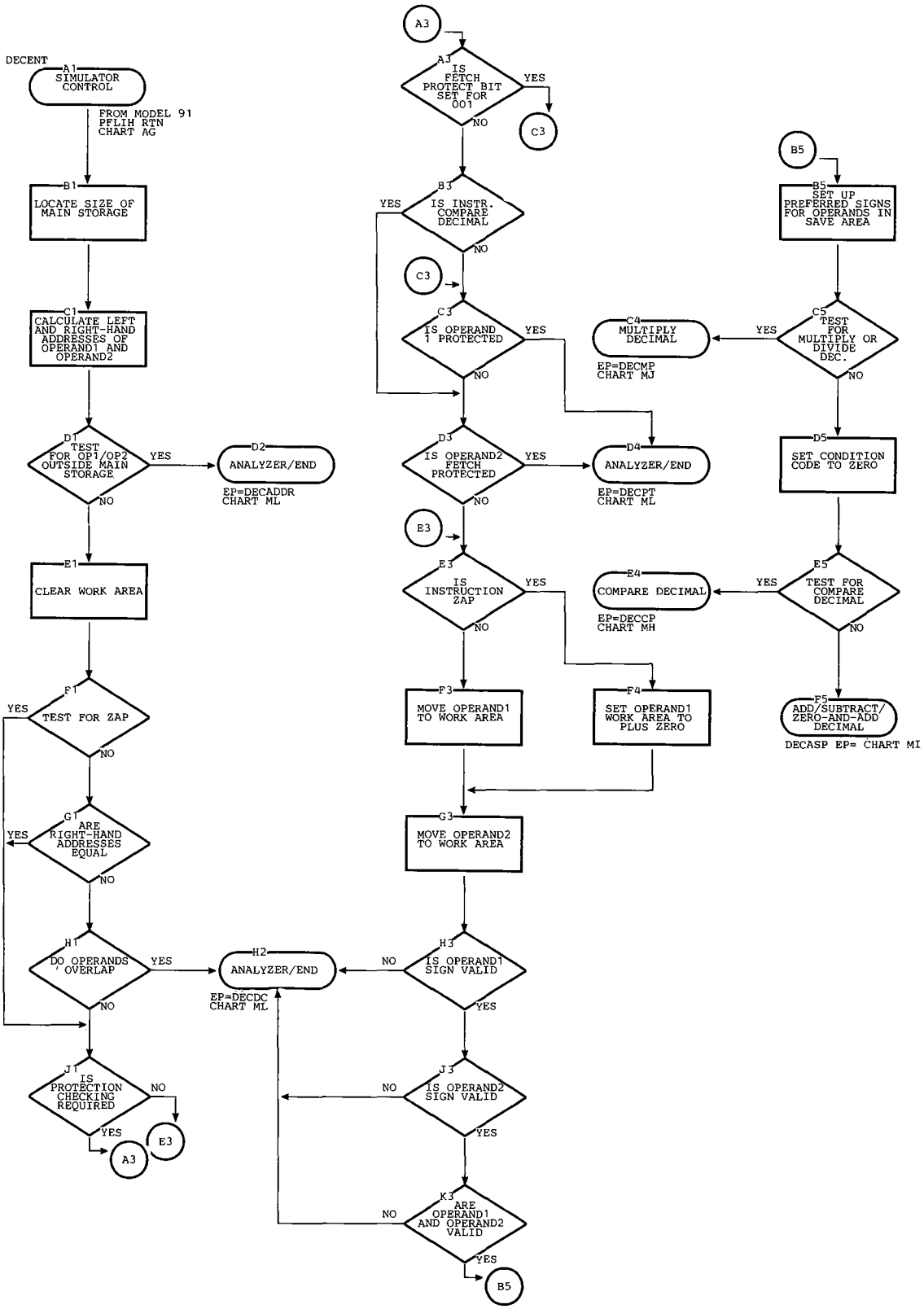


Chart MH. Models 91 and 195 Compare Decimal Routine

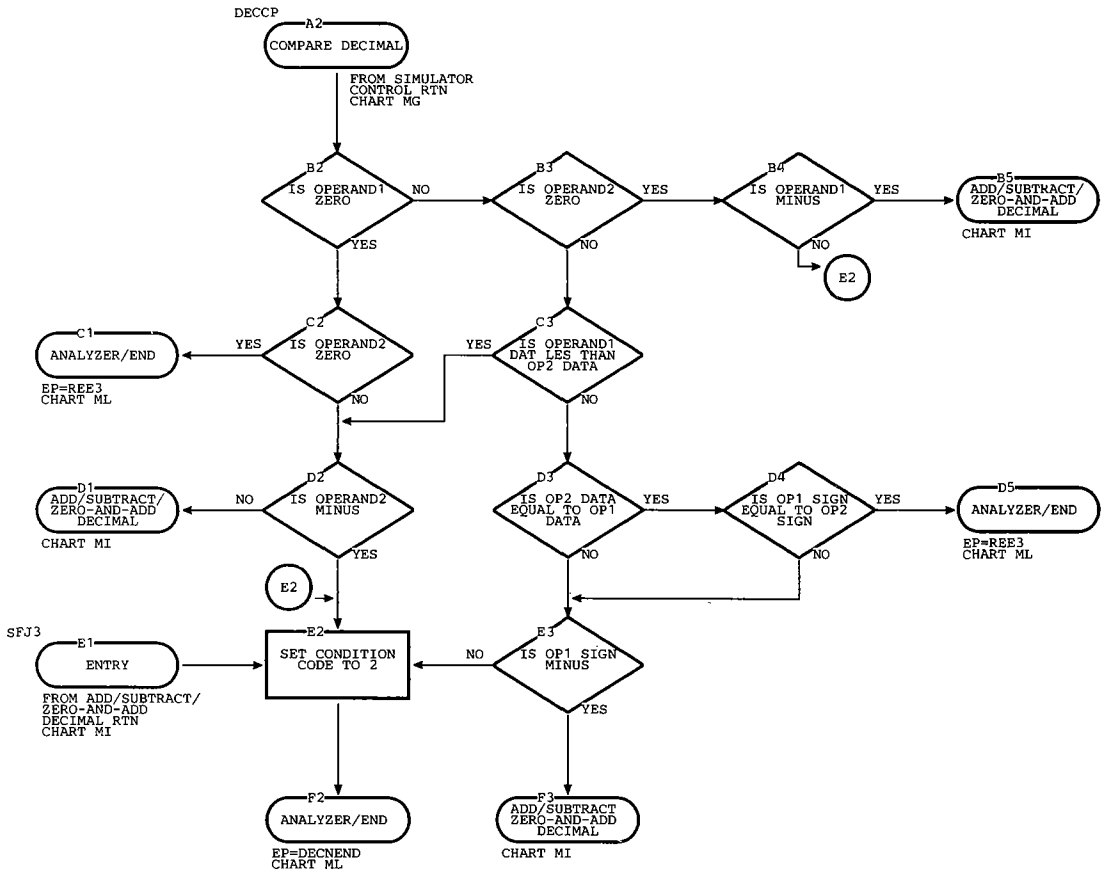


Chart MI. Models 91 and 195 Add/Subtract/Zero-and-Add Decimal Routine (Page 1 of 2)

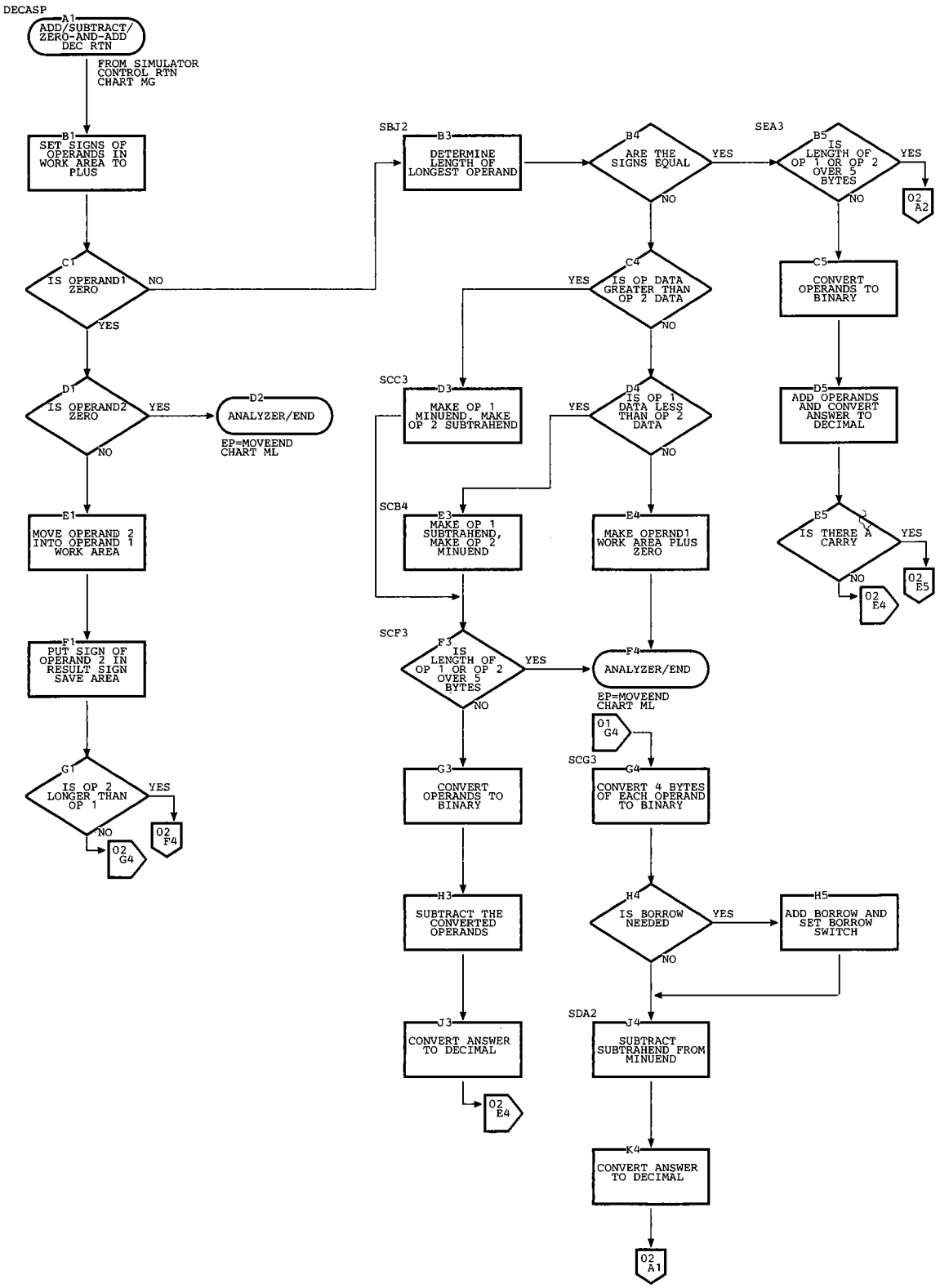


Chart MI. Models 91 and 195 Add/Subtract/Zero-and-Add Decimal Routine (Page 2 of 2)

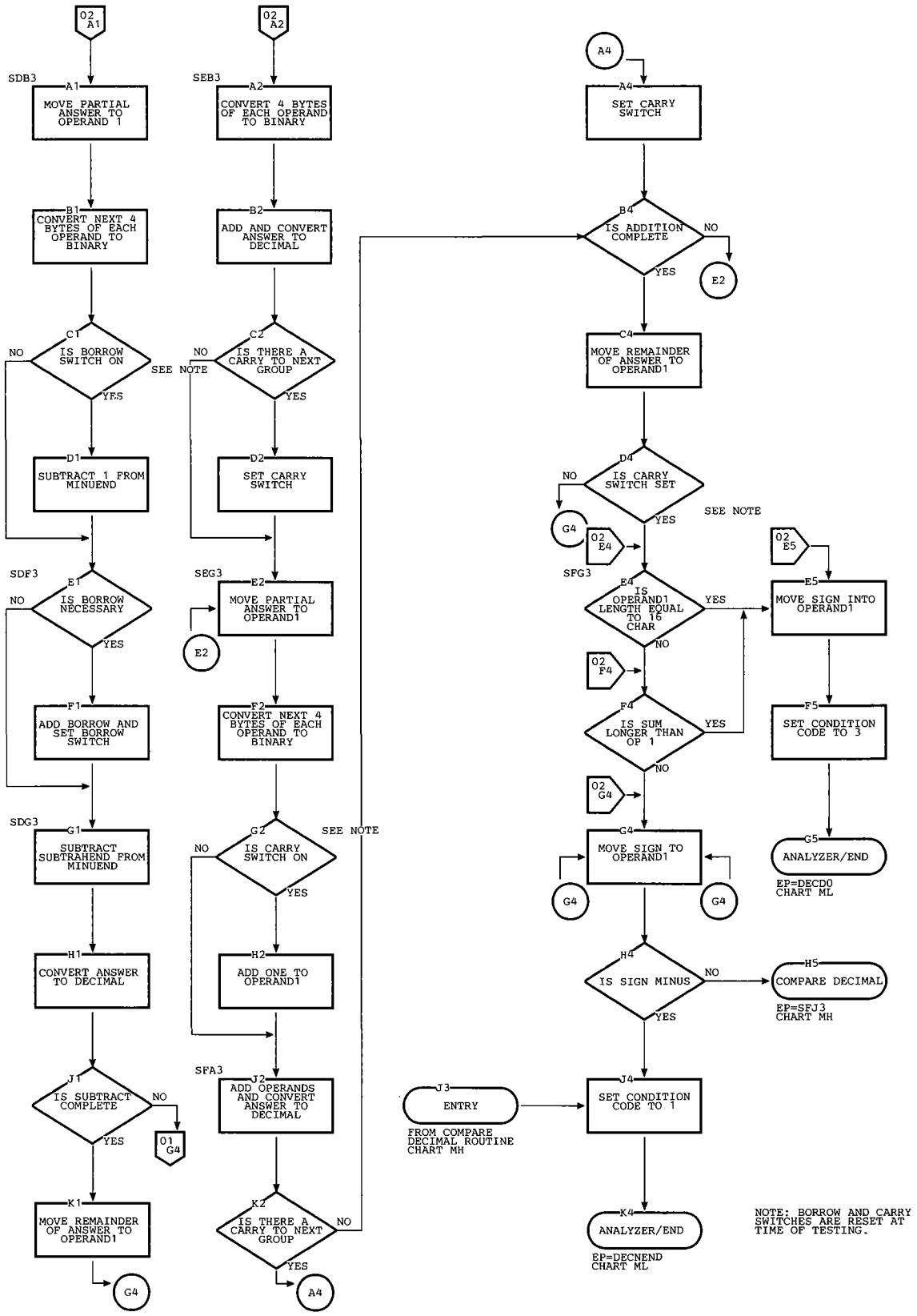


Chart MJ. Models 91 and 195 Multiply Decimal Routine

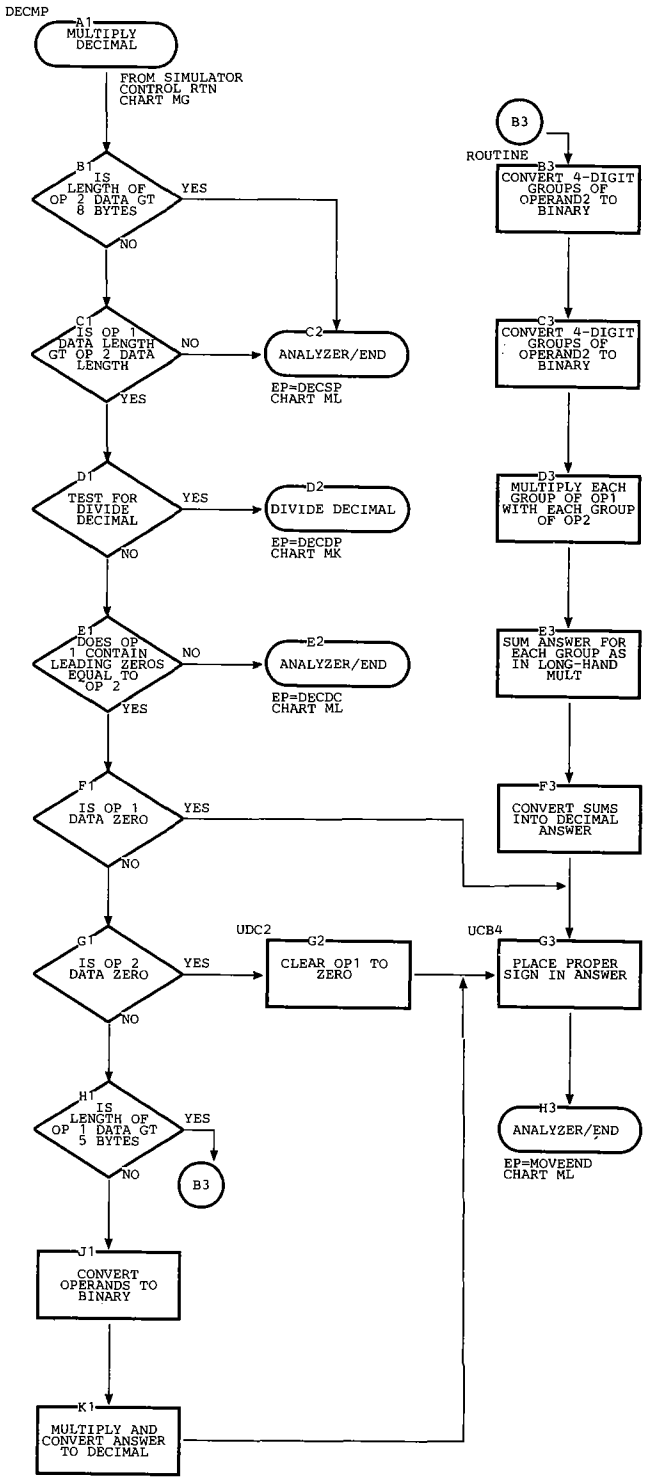


Chart MK. Models 91 and 195 Divide Decimal Routine

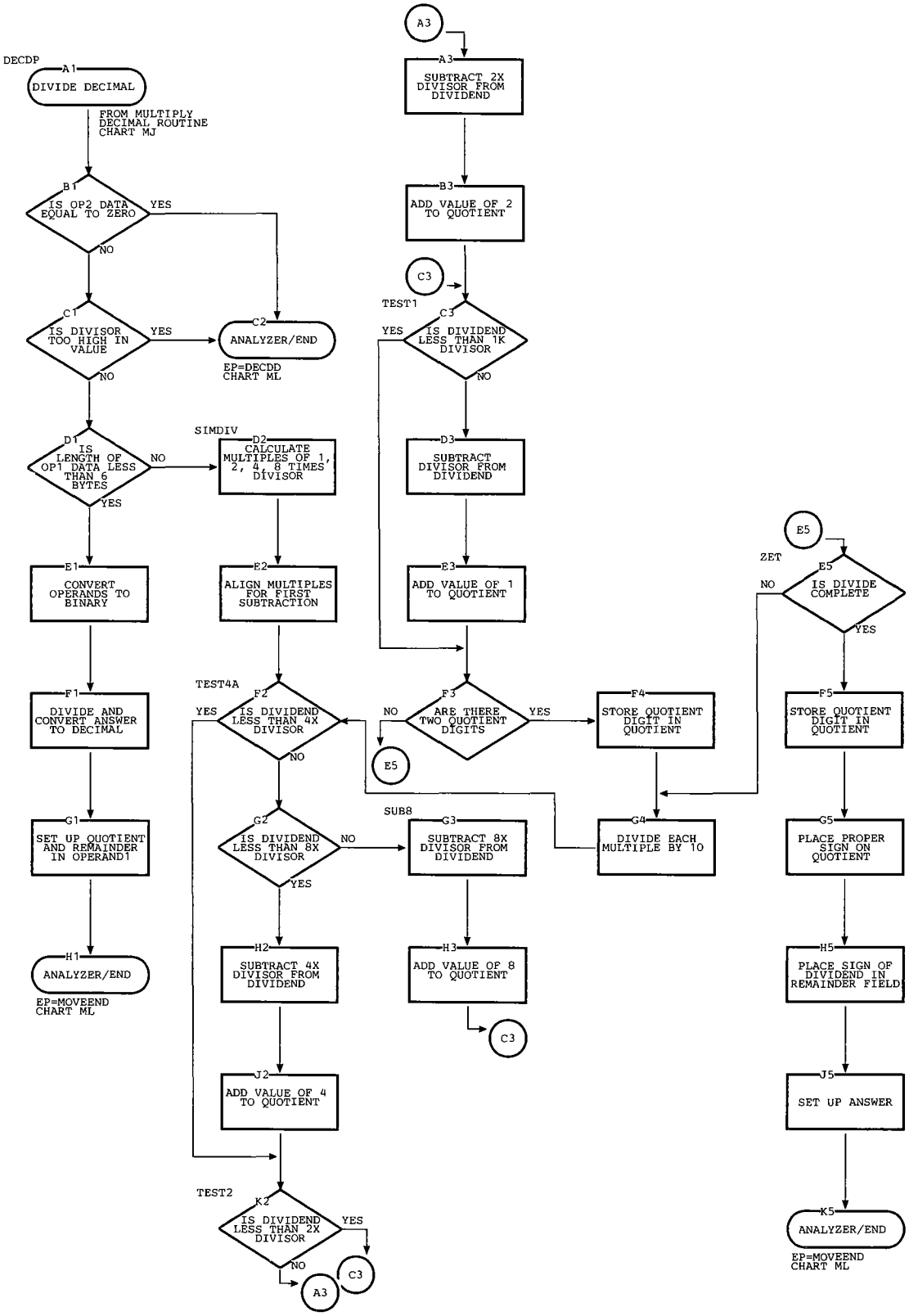


Chart ML. Models 91 and 195 Analyzer/End Routine

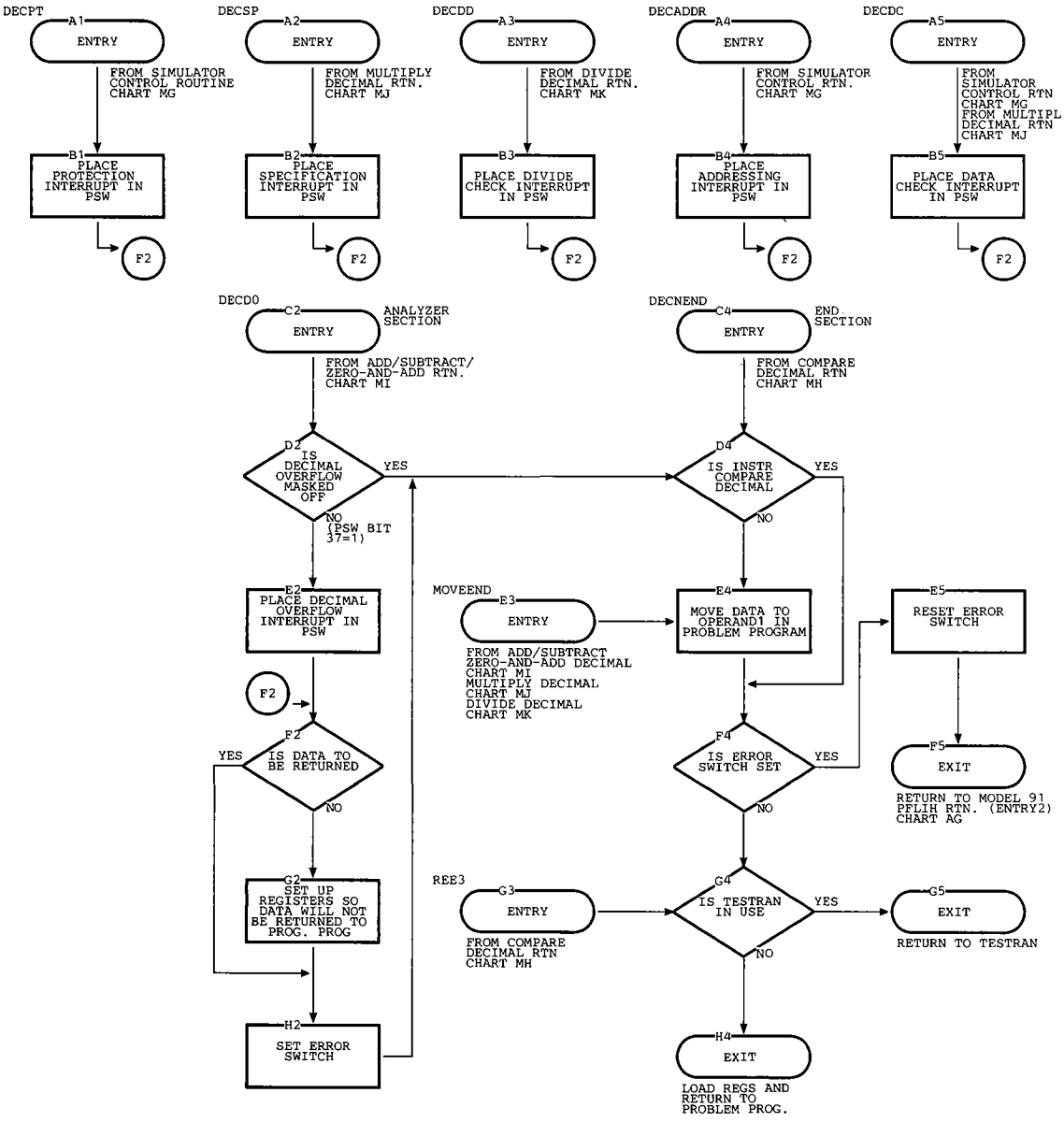


Chart MM. System Management Facility EXCP Counter Routine (Page 1 of 2)

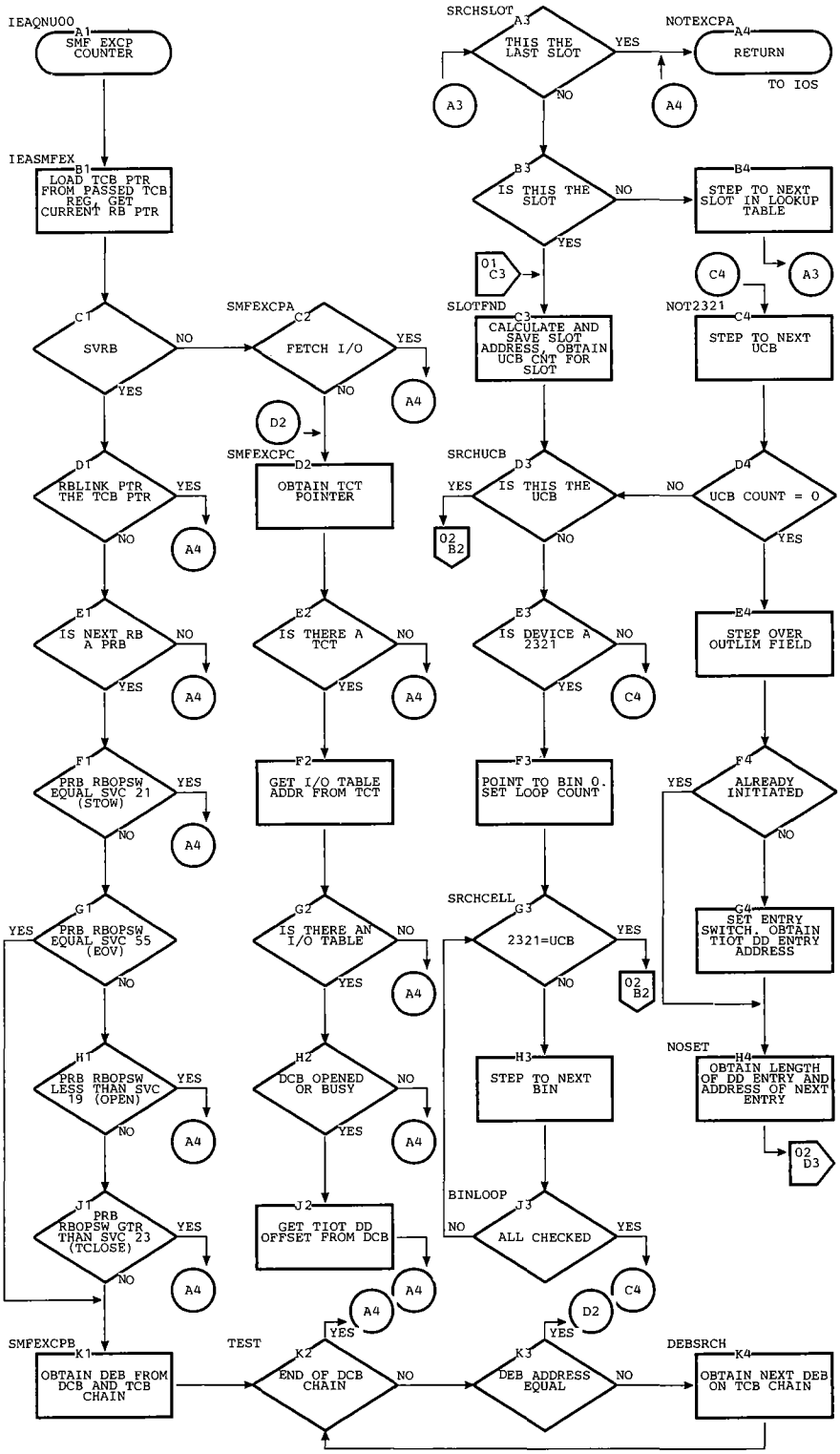


Chart MM. System Management Facility EXCP Counter Routine (Page 2 of 2)

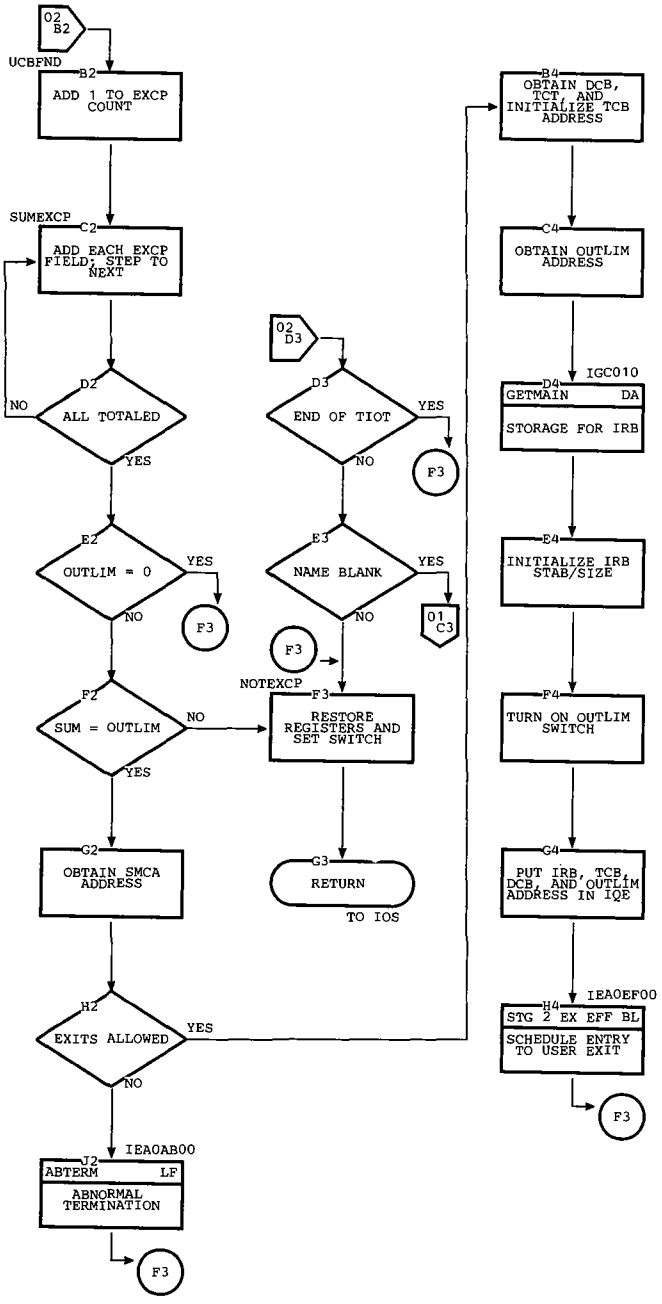


Chart MN. System Management Facility Time/Output Limit Expiration Routine

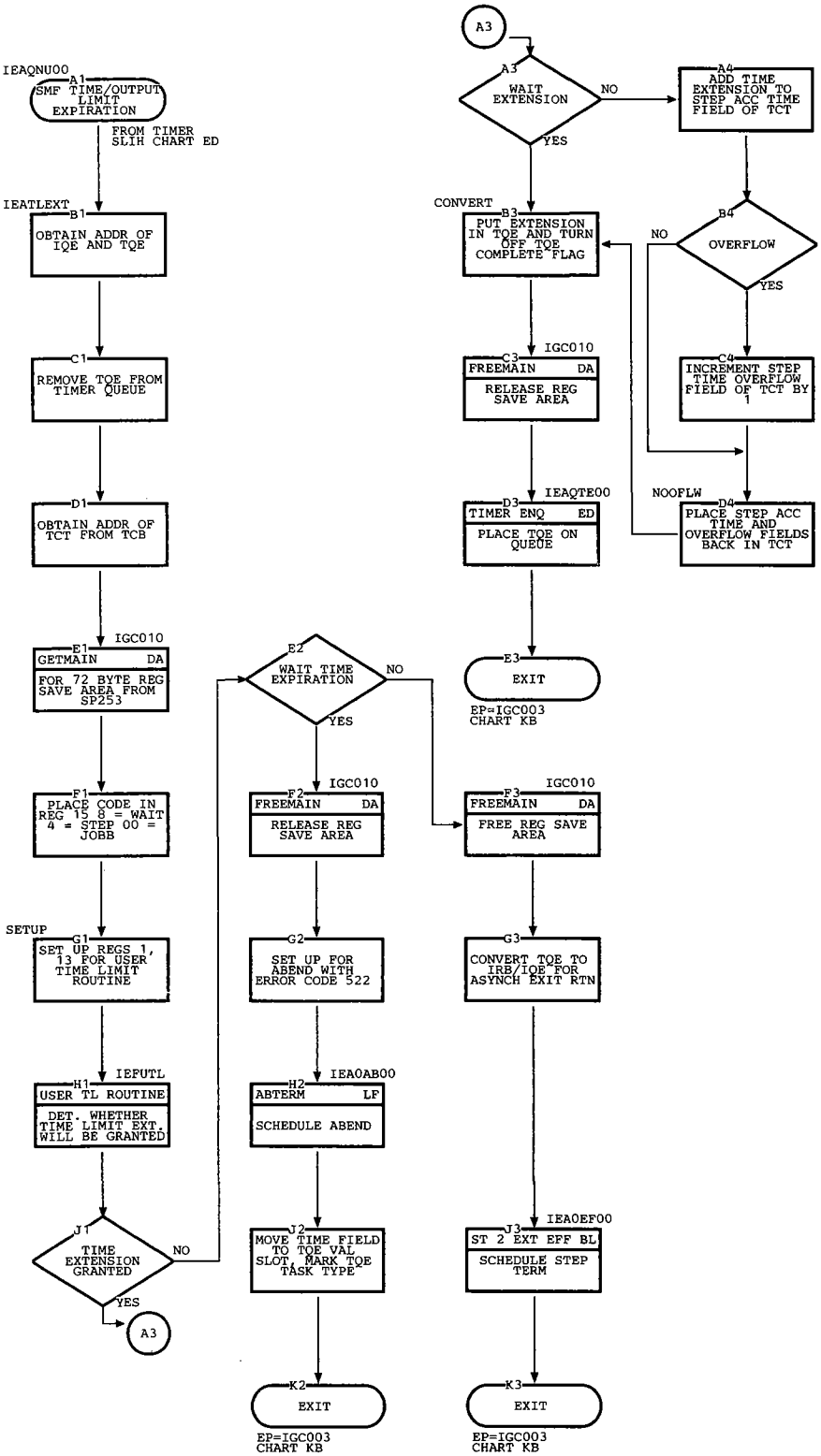
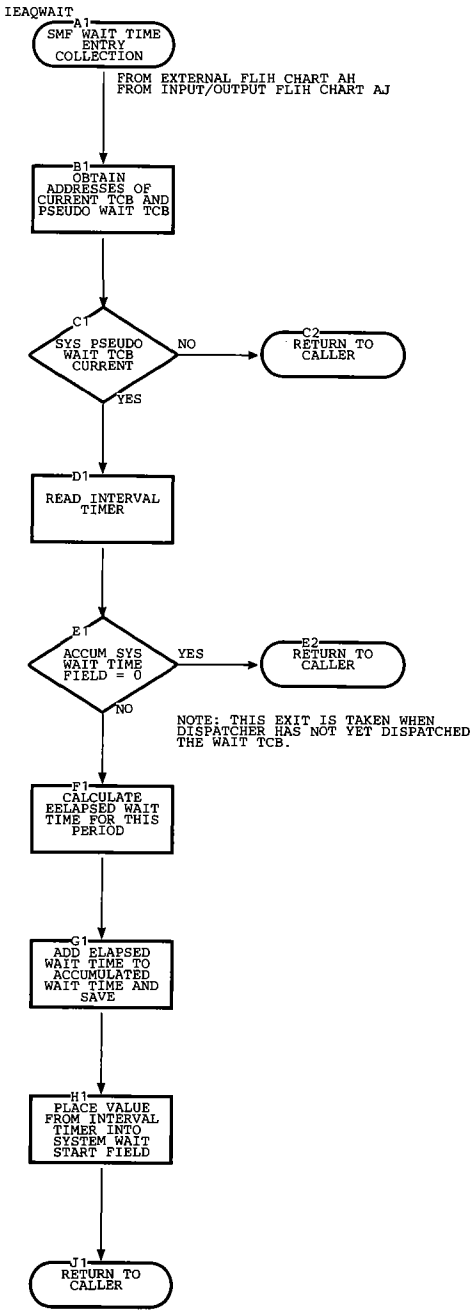


Chart MO. System Management Facility Wait Time Collection Routine



This section is designed to help the reader 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.

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.
CDADVANS	Contents Supervision common subroutines (search phase)	IEAQLK00	IEAQLK00	CS	CA	NUC			
CDCONTRL also called IEAQCS02	Contents Supervision common subroutines (search phase)	IEAQLK00	IEAQLK00	CS	CA	NUC			
CDDESTRY	CDEXIT routine	IEAQET00	IGC003	EP,TP	KE	NUC			
CDEPILOG	Contents Supervision common subroutines (scheduling phase)	IEAQLK00	IEAQLK00	CS	CA	NUC			
CDEXIT	CDEXIT routine	IEAQET00	IGC003	EP	KE	NUC			
CDHKEEP	CDEXIT routine	IEAQET00	IGC003	EP,TP	KE	NUC			
EOT	End-of-Task (EOT) routine	IEAQET00	IGC003	TP	LA	NUC			
ERFETCH	Stage 3 Exit Effector	IEAQNU00	IEAQNU00	TS	BM	NUC			
FLASH	First CPU Signal routine	IEAQFX00	IEAQFX00	TS		NUC			
FMBRANCH	FREEMAIN routine (branch e.p.)	IEAQGM00	IEAQGM00	MSS	DA	NUC	1		
FMSMFCRE	SMF Storage routine	IEAQGM00	IEASMFGF	SF	DA	NUC			
FTCE01	Program Fetch Channel-End Appendage routine	IEWFETCH	IEWFETCH	CS	CD	NUC			
FTPCI01	Program Fetch PCI Appendage routine	IEWFETCH	IEWFETCH	CS	CD	NUC			
GETIQE	GETMAIN routine (branch e.p. to the GETIQE subroutine)	IEAQPRT0	IEAQPRT0	CS	DA	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		
GMSMFCRE	SMF Storage routine	IEAQGM00	IEASMFGF	SF	DA	NUC			
IBMORG	SVC Table (start of IBM-assigned SVC numbers)	IEAQBK00	IEAQBK00			NUC			
IEAASPRG	Subsystem Purge	IEAASPRG	IEAASPRG	TP		NUC			
IEABEND	Secondary Communications Vector Table (used by the ABEND routine)	IEAQET00	IGC003			NUC			
IEACVT	Communications Vector Table	IEAQBK00	IEAQBK00			NUC			
IEADQTCB	Dequeue TCB routine	IEAQET00	IGC003	TP	LC	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			

*Routine discussed in I/O Supervisor PLM, GY28-6676.

Figure 14-1. Module Directory (Part 1 of 17) (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.
IEAERWA	I/O Supervisor transient area	IEAQBK00	IEAQBK00			NUC			
IEAMCH00	SER0 routine (resident module)	IFBSR000	IFBSR000	IH	AK	NUC			
	SER1 routine for System/360, model 40	IFBSR340	IFBSR340	IH	AL	LINK			
	SER1 routine for System/360, model 50	IFBSR350	IFBSR350	IH	AL	LINK			
	SER1 routine for System/360, model 65	IFBSR365	IFBSR365	IH	AL	LINK			
	SER1 routine for System/360, model 75	IFBSR375	IFBSR375	IH	AL	LINK			
	SER1 routine for System/360, model 91	IFBSR395	IFBSR395	IH	AM	LINK			
	SER1 routine for System/360, model 195	IFBSR3A5	IFBSR3A5	IH	AM	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	BH	NUC	1		
IEAOPT02	Post routine (branch e.p. from the I/O Supervisor and from supervisor routines)	IEAQSY50	IGC001	TS	BH	NUC	1		
IEAQABL	Release Loaded Programs routine	IEAQET00	IGC003	TP	LE	NUC			
IEAQCS01	Contents Supervision common subroutines (e.p. for the ATTACH macro instruction)	IEAQLK00	IEAQLK00	CS	CA	NUC			
IEAQCS02 also called CDCONTRL	Contents Supervision common subroutines (search phase)	IEAQLK00	IEAQLK00	CS	CA	NUC			
IEAQCS03 also called CDEPILOG	Contents Supervision common subroutines (scheduling phase)	IEAQLK00	IEAQLK00	CS	CA	NUC			
IEAQEQ01	ENQ/DEQ Purge routine	IEAQENQ2	IGC048	TP		NUC			
IEAQERA	Erase Phase routine	IEAQET00	IGC003	TP		NUC			
IEAQEX00	External First-Level Interruption Handler	IEAQNU00	IEAQNU00	IH	AH,AI	NUC			
IEAQIO00	I/O First-Level Interruption Handler	IEAQNU00	IEAQNU00	IH	AJ	NUC			
IEAQLPAQ	Link pack area queue	IEAQBK00	IEAQBK00			NUC			
IEAQPGTM	Purge Timer routine	IEAQET00	IGC003	TP	LD	NUC			
IEAQPK00	Program Check First-Level Interruption Handler	IEAQNU00	IEAQNU00	IH	AF,AG	NUC			

Figure 14-1. Module Directory (Part 2 of 17) (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.
IEAQQCB0	Origin of QCB queues	IEAQENQ2	IGC048			NUC			
IEAQRORI	Rollout/Rollin module	IEAQRORI	IEAQRORI	MSS	DC-DI	NUC			
IEAQSC00	SVC First-Level Interruption Handler	IEAQNU00	IEAQNU00	IH	AA	NUC			
IEAQSPET	Release Main Storage routine	IEAQET00	IGC003	TP	LE	NUC			
IEAQTAQ IEAQTAQ1	Transient Area Control table (TACT)	IEAQBK00	IEAQBK00			NUC			
IEAQTD00	Timer Second-Level Interruption Handler (branch e.p.)	IEAQTI00	IEAQTI00	TMS	ED,EH	NUC			
IEAQTD01	Timer Second-Level Interruption Handler (e.p. for Dispatcher)	IEAQTI00	IEAQTI00	TMS	ED	NUC			
IEAQTE00	Timer Second-Level Interruption Handler (branch e.p.)	IEAQTI00	IEAQTI00	TMS	ED	NUC			
IEAQTR00	SVC Second-Level Interruption Handler	IEAQTR33	IEAQTR00	IH	AB	NUC			
IEAQTR01	Transient Area Exit routine	IEAQTR33	IEAQTR00	EP	KC	NUC			
IEAQTR02	Transient Area Refresh routine	IEAQTR33	IEAQTR00	EP	KD	NUC			
IEAQTR03	Transient Area XCTL routine	IEAQTR33	IEAQTR00	CS	CA	NUC			
IEAQWAIT	SMF Wait Time Collection routine	IEAQNU00	IEAQNU00	SF	MO	NUC			
IEASMFEX	SMF EXCP Counter routine	IEAQNU00	IEAQNU00	SF	MM	NUC			
IEATCB1	Transient Area Fetch TCBs	IEAQBK00	IEAQBK00			NUC			
IEATCB2		IEAQBK00	IEAQBK00			NUC			
IEATCBn		IEAQBK00	IEAQBK00			NUC			
IEATCBn +1		IEAQBK00	IEAQBK00			NUC			
IEATLEXT	SMF Time/Output Limit Expiration routine	IEAQTI01	IEAQTI00	SF	MN	NUC			
	SMF Time/Output Limit Expiration routine with Model 65 Multiprocessing	IEAQTIM1	IEAQTI00	SF	MN	NUC			
IEATYPE1	Type-1 SVC Switch	IEAQNU00	IEAQNU00			NUC			
IEAXDS00	Decimal Simulator routine (Models 91 and 195)	IEAXDS00	IEAXDS00	SF	MG-ML				
IEAXKALL	Extended Precision Floating Point Simulator for System/360 models	IEAXPALL	IEAXKALL	SF		LINK			
IEAXKDXR	Extended Precision Floating Point Simulator for System/360 Models 85, 195 and System/370 models	IEAXPDXR	IEAXKDXR	SF		LINK			
IEAXPALL	Extended Precision Floating Point Simulator for System/360 models	IEAXPALL	IEAXPALL	SF		LINK			

Figure 14-1. Module Directory (Part 3 of 17) (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.
IEAXPDXR	Extended Precision Floating Point Simulator for System/360 Models 85, 195, and System/370 models	IEAXPDXR	IEAXPDXR	SF		LINK			
IEAXPSIM	Extended Precision Floating Point Simulator CPU Determination	IEAXPSIM	IEAXPSIM	SF		LINK			
IEA0AB00 IEA0AB01	ABTERM routine	IEAQAB00	IEAQAB00	TP	LF	NUC			
IEA0DS	Dispatcher	IEAQNU00	IEAQNU00	EP	KF-KL	NUC			
IEA0DS02	Task Switching routine	IEAQNU00	IEAQNU00	TS	BN,BO	NUC			
IEA0EF00	Stage 2 Exit Effector	IEAQNU00	IEAQNU00	TS	BL	NUC			
IEA0EF03	Stage 3 Exit Effector	IEAQNU00	IEAQNU00	TS	BM	NUC			
IEA0PL00	ABTERM Prologue routine	IEAQAB00	IEAQAB00	TP	LH	NUC			
IEA0TI00	Timer Second-Level Interruption Handler (e.p. for External First-Level Interruption Handler)	IEAQTI00	IEAQTI00	TMS	ED,EH	NUC			
IEA0VL00	Validity Check routine	IEAQNU00	IEAQNU00	TS		NUC			
IEA0XE00	Type-1 Exit routine	IEAQNU00	IEAQNU00	EP	KA	NUC			
IECINT	I/O Interruption Supervisor in the I/O Supervisor	IEAQFX00 (IEAQFX and IECIOS macros)	IEAQFX00	CS		NUC			
IECPBLDL	BLDL routine	IECPFIND or IECPFND1 (depends on SYSGEN option)	IGC018			NUC	2	BLDL	SVC 18
IECXCP	EXCP Supervisor in the I/O Supervisor	IEAQFX00 (IEAQFX and IECIOS macros)	IEAQFX00	CS		NUC	1	EXCP	SVC 0
IECXTLER	Stage 3 Exit Effector	IEAQNU00	IEAQNU00	TS	BM	NUC			
IEEBA1	Attention routine	IEECVCRA	IEEBA1	CCSL	FA	NUC			
IEEBC1PE	Communications Task External Interruption Handler	IEECVCRX	IEEBC1PE	CCSL	FA	NUC			
IEECIR45	Communications Task Wait routine	IEECVCTB	IEECIR45	CCSL	FF	NUC			
IEECMDOM	Communications Task DOM Service Module (MCS)	IEECMDOM	IEECMDOM	CCSL	FO	NUC	2	DOM	SVC 87
IEECMDSV	Communications Task Device Interface Module (MCS)	IEECMDSV	IEECMDSV	CCSL	FM	NUC			

Figure 14-1. Module Directory (Part 4 of 17) (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.
IEECMENQ	Communications Task Enqueue Subroutine	IEECMWSV	IEECMWSV	CCSL	FN				
IEECMQCN	Communication Task Console Output Block Creation	IEECMWSV	IEECMWSV	CCSL	FN				
IEECMWSV	Communications Task WTO(R) Service Module (MCS)	IEECMWSV	IEECMWSV	CCSL	FN	NUC			
IEECVCTI	Communications Task Initialization routine	IEECVINT	IEECVCTI	CCSL	FD	NUC			
IEECVCTW	Communications Task Router Module (MCS)	IEECMQWR	IEECVCTW	CCSL	FI	NUC			
IEECVETA	DIDOCS Options routine	IGC5A07B		CCSL	HT	SVC	4		
IEECVETC	DIDOCS Asynchronous Error routine	IGC5C07B		CCSL	HK	SVC	4		
IEECVETD	DIDOCS Message 1 routine	IGC5D07B	IEECVETD	CCSL	HL	SVC	4		
IEECVETE	DIDOCS Message 2 routine	IGC5E07B		CCSL	HM	SVC	4		
IEECVETF	DIDOCS Light Pen/Cursor Service routine	IGC5F07B		CCSL	IE	SVC	4		
IEECVETG	DIDOCS Open/Close routine	IGC5G07B		CCSL	HE	SVC	4		
IEECVETH	DIDOCS Model 85 I/O routine	IGC5H07B		CCSL	HJ	SVC	4		
IEECVETJ	DIDOCS Roll Mode routine	IGC5J07B		CCSL	HR	SVC	4		
IEECVETK	DIDOCS Timer Interpreter routine	IGC5K07B		CCSL	IF	SVC	4		
IEECVETP	DIDOCS 2250 I/O-1 routine	IHF5P07B		CCSL	HF	SVC	4		
IEECVETQ	DIDOCS 2250 I/O-2 routine	IGC5Q07B		CCSL	HG	SVC	4		
IEECVETR	DIDOCS 2260 I/O-1 routine	IGC5R07B		CCSL	HH	SVC	4		
IEECVETZ	DIDOCS Processor 1, Load 2	IGC5Z07B	IEECVETZ	CCSL	HD	SVC	4		SVC 72
IEECVET1	DIDOCS Processor 1, Load 1	IGC5107B		CCSL	HC	SVC	4		
IEECVET2	DIDOCS Display 1 routine	IGC5207B		CCSL	HO	SVC	4		
IEECVET3	DIDOCS Display 2 routine	IGC5307B		CCSL	HP	SVC	4		
IEECVET4	DIDOCS Command routine	IGC5407B		CCSL	HS	SVC	4		
IEECVET6	DIDOCS Delete 1 routine	IGC5607B		CCSL	IA	SVC	4		
IEECVET7	DIDOCS Delete 2 routine	IGC5707B		CCSL	IB	SVC	4		
IEECVET8	DIDOCS Delete 3 routine	IGC5807B		CCSL	IC	SVC	4		
IEECVET9	DIDOCS Delete 4 routine	IGC5907B		CCSL	ID	SVC	4		
IEECVFTA	PFK 1 routine	IGC6A07B	IEECVFTA	CCSL	IG	SVC	4		SVC 72
IEECVFTB	PFK 2 routine	IGC6B07B	IEECVFTB	CCSL	IH	SVC	4		SVC 72
IEECVFTD	DIDOCS Message 3 routine	IGC6D07B	IEECVFTD	CCSL	HN	SVC	4		SVC 72
IEECVFTG	DIDOCS Cleanup routine	IGC6G07B	IEECVFTG	CCSL	II	SVC	4		SVC 72

Figure 14-1. Module Directory (Part 5 of 17) (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.
IEECVFTL	Status Display Interface 1	IGC6L07B	IEECVFTL	CCSL	IJ	SVC	4		SVC 72
IEECVFTM	Status Display Interface 2	IGC6M07B	IEECVFTM	CCSL	IK	SVC	4		SVC 72
IEECVFTN	Status Display Interface 3	IGC6N07B	IEECVFTN	CCSL	IL	SVC	4		SVC 72
IEECVFTO	Status Display Interface 4	IGC6O07B	IEECVFTO	CCSL	IM	SVC	4		SVC 72
IEECVFTP	Status Display Interface 5	IGC6P07B	IEECVFTP	CSSL	IN	SVC	4		SVC 72
IEECVFTQ	Status Display Interface 6	IGC6Q07B	IEECVFTQ	CCSL	IO	SVC	4		SVC 72
IEECVFTR	DIDOCS 2260 I/O-2 routine	IGC6R07B	IEECVFTR	CCSL	HI	SVC	4		SVC 72
IEECVFTT	Status Display Interface 7	IGC6T07B	IEECVFTT	CCSL	IP	SVC	4		SVC 72
IEECVFTZ	DIDOCS Processor 0, Load 2	IGC6Z07B	IEECVFTZ	CCSL	HB	SVC	4		SVC 72
IEECVFT1	DIDOCS Processor 0, Load 1	IGC6107B	IEECVFT1	CCSL	HA	SVC	4		SVC 72
IEECVFT2	DIDOCS Display 3 routine	IGC6207B	IEECVFT2	CCSL	HQ	SVC	4		SVC 72
IEECVGCI	Graphic Console Initialization	IEECVGCI	IEECVGCI	CCSL	FE	LINK			
IEECVML1	MLWTO (Non-MCS) Load 1	IGC0603E	IEECVML1	CCSL	GD	SVC	4	WTO	SVC 35
IEECVML2	MLWTO (Non-MCS) Load 2	IGC0703E	IEECVML2	CCSL	GE	SVC	4	WTO	SVC 35
IEECVML3	MLWTO (MCS) Load 1	IGC0603E	IEECVML3	CCSL	GD	SVC	4	WTO	SVC 35
IEECVML4	MLWTO (Non-MCS) Load 3	IGC0803E	IEECVML4	CCSL	GF	SVC	4	WTO	SVC 35
IEECVML5	MLWTO (MCS) Load 2	IGC0703E	IEECVML5	CCSL	GE	SVC	4	WTO	SVC 35
IEECVML6	MLWTO (MCS) Load 3	IGC0803E	IEECVML6	CCSL	GF	SVC	4	WTO	SVC 35
IEECVML7	MLWTO (MCS) Load 4	IGC0903E	IEECVML7	CCSL	GG	SVC	4	WTO	SVC 35
IEECVPRG	WTOR Purge routine (also called the Reply Purge routine)	IEECVED2	IEECVPRG	TP		NUC			
IEECVUCM	Communications Task Unit Control Module	IEECVUCM	IEECVUCM	CCSL	(Fig. 7-1, 7-2)	NUC			
IEEC2740	2740 Processor Module	IEEC2740	IEEC2740	CCSL	FW	SVC	4		SVC 72
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. 7-10)	LINK			
IEEVLDSP	WRITELOG Dispatch routine	IEEVLDSP	IEEVLDSP	CCSL	(Fig. 7-10)	LINK			
IEEVLIN	WRITELOG Log Initialization routine	IEEVLIN	IEEVLIN1	CCSL	(Fig. 7-10)	LINK			
IEEVLOGJ	Job File Control Blocks (JFCBs) for log data sets	IEEVLOGJ	IEEVLOGJ	CCSL	(Fig. 7-10)	LINK			
IEEVLOPN	WRITELOG Open Device routine	IEEVLOPN	IEEVLOPN	CCSL	(Fig. 7-10)	LINK			
IEEVLOUT	WRITELOG Available Log Data Set routine	IEEVLOUT	IEEVLOUT	CCSL	(Fig. 7-10)	LINK			

Figure 14-1. Module Directory (Part 6 of 17) (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.
IEEVLWTR	WRITELOG Log Writer routine	IEEVLWTR	IEEVLWTR	CCSL	(Fig. 7-10)	LINK			
IEEVRFRX	Communications Task Misc. Look-up Services routine	IEEVRFRX	IEEVRFRX	CCSL		LINK			
IEEVWAIT	WRITELOG Master Wait routine	IEEVWAIT	IEEVWAIT	CCSL	(Fig. 7-10)	LINK			
IEE0303F	Write-to-Log routine	IGC0303F	IEE0303F	CCSL	GB	SVC	4	WTL	SVC 36
IEE0403F	Log Data Set Open/Close	IEE0403F	IEE0403F	CCSL	GC	SVC	4		SVC 36
IEE1A03D	Communications Task MCS Reply Processor routine	IEE1A03D	IEE1A03D	CCSL		SVC			
IEE1B03D	Communications Task Error Message routine	IEE1B03D	IEE1B03D	CCSL		SVC			
IEE1203D	Communications Task Reply	IEE1203D	IEE1203D	CCSL		SVC			
IEE1403D	HALT and WRITELOG CLOSE routine	IGC1403D	IEE1403D	CCSL	(Fig. 7-10)	SVC	4		SVC 34
IEE1603D	LOG and WRITELOG routine	IEE1603D	IEE1603D	CCSL	(Fig. 7-10)	SVC	4		SVC 34
IEGHTOVL	TESTRAN Interpreter	IEGTTRNO	IEGHTOVL	CS		LINK			
IEWFBOSV	Program Fetch routine (e.p. from the Overlay Supervisor)	IEWFETCH	IEWFETCH	CS	CD	NUC			
IEWFTRAN	Program Fetch routine (e.p. from the TA Fetch routine)	IEWFETCH	IEWFETCH	CS	CD	NUC			
IEWMSEPT	Program Fetch routine (e.p. from the common subroutines of Contents Supervision)	IEWFETCH	IEWFETCH	CS	CD	NUC			
IEWSZOVR	Overlay Supervisor (non-resident module)	IEWSWOVR	IEWSWOVR	CS	CE	LINK			
IFBSER00	SER0 routine (System/360, Model 40)	IFBSR040	IFBSR040	IH	AK	LINK			
IFBSER00	SER0 routine (System/360, model 50)	IFBSR050	IFBSR050	IH	AK	LINK			
IFBSER00	SER0 routine (System/360, model 65)	IFBSR065	IFBSR065	IH	AK	LINK			
IFBSER00	SER0 routine (System/360, model 75)	IFBSR075	IFBSR075	IH	AK	LINK			
IGCXI07B	Communications Task External Processor	IEECVCTX	IEECVCTX	CCSL	FH	SVC	4		SVC 72
IGCXL07B	Communications Task Console Switch (load 1) (MCS)	IEECLCTX	IEECLCTX	CCSL	FJ	SVC	4		SVC 72
IGCXM07B	Communications Task Console Switch (load 2) (MCS)	IEECMCTX	IEECMCTX	CCSL	FK	SVC	4		SVC 72
IGCXN07B	Communications Task Console Switch (load 3) (MCS)	IEECNCTX	IEECNCTX	CCSL	FK	SVC	4		SVC 72

Figure 14-1. Module Directory (Part 7 of 17) (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.
IGCXO07B	Communications Task Console Switch (load 4) (MCS)	IEECOCTX	IEECOCTX	CCSL	FL	SVC	4		SVC 72
IGC0A05A	ABDUMP routine ("resident" Module)	IEAQAD0A	IGC0A05A	TP	LI	SVC	4		SVC 51
IGC0A06C	Preserve 1	IGC0A06C	IGC0A06C	CR	JE	SVC	4		SVC 63
IGC0B01C	ABEND routine (ABEND11)	IEAQTM0B	IGC0B01C	TP	LT	SVC	4		SVC 13
IGC0B05A	ABDUMP routine (ABDUMP11)	IEAQAD0B	IGC0B05A	TP	LI	SVC	4		
IGC0C01C	ABEND routine (ABEND12)	IEAQTM0C	IGC0C01C	TP	LU	SVC	4		SVC 13
IGC0C05A	ABDUMP routine (ABDUMP1.5)	IEAQAD0C	IGC0C05A	TP	LI	SVC	4		
IGC0D01C	ABEND routine (ABEND13)	IEAQTM0D	IGC0D01C	TP	LV	SVC	4		SVC 13
IGC0D05A	TCAM ABDUMP 1	IEAQAD0D	IGC0D05A	TP	LJ	SVC	4		
IGC0D06C	Preserve 2	IGC0D06C	IGC0D06C	CR	JE	SVC	4		SVC 63
IGC0E05A	TCAM ABDUMP 2	IEAQAD0E	IGC0E05A	TP	LJ	SVC	4		
IGC0F01C	ABEND routine (ABEND15)	IEAQTM0F	IGC0F01C	TP	LW	SVC	4		SVC 13
IGC0F05A	TCAM ABDUMP 3	IEAQAD0F	IGC0F05A	TP	LJ	SVC	4		
IGC0F06C	Checkmain 1	IGC0F06C	IGC0F06C	CR	JF	SVC	4		SVC 63
IGC0G01C	ABEND routine (ABEND16)	IEAQTM0G	IGC0G01C	TP	LX	SVC	4		SVC 13
IGC0G05A	TCAM ABDUMP 4	IEAQAD0G	IGC0G05A	TP	LJ	SVC	4		
IGC0G05B	REP I/O-JFCB Processor 1	IGC0G05B	IGC0G05B	CR	JO	SVC	4		SVC 52
IGC0G06C	Checkmain 2	IGC0G06C	IGC0G06C	CR	JF	SVC	4		SVC 63
IGC0G95B	REP I/O-JFCB Processor 1A	IGC0G95B	IGC0G95B	CR	JO	SVC	4		SVC 52
IGC0H05A	ABDUMP routine (ABDUMPH)	IEAQAD0H	IGC0H05A	TP		SVC	4		
IGC0H05B	Dummy Data Set Processor	IGC0H05B	IGC0H05B	CR	JP	SVC	4		SVC 52
IGC0H06C	Checkmain 3	IGC0H06C	IGC0H06C	CR	JG	SVC	4		SVC 63
IGC0I05A	ABDUMP routine (ABDUMPI)	IEAQAD0I	IGC0I05A	TP		SVC	4		
IGC0I05B	REP I/O-JFCB Processor 2	IGC0I05B	IGC0I05B	CR	JO	SVC	4		SVC 52
IGC0I07B	Communications Task 1052 Open/Close	IEECVOCX	IEECVOC	CCSL	FT	SVC	4		SVC 72
IGC0J05A	ABDUMP routine (ABDUMP12)	IEAQAD0J	IGC0J05A	TP	LI	SVC	4		SVC 51
IGC0J05B	TCAM Data Set Processor	IGC0J05B	IGC0J05B	CR	JX	SVC	4		SVC 52
IGC0K01C	ABEND routine (ABEND20)	IEAQTM2K	IGC0K01C	TP	LY	SVC	4		
IGC0K05A	ABDUMP routine (ABDUMP13)	IEAQAD0K	IGC0K05A	TP	LI	SVC	4		SVC 51
IGC0K05B	REP I/O-Mount/Verify 1 routine	IGC0K05B	IGC0K05B	CR	JQ	SVC	4		SVC 52
IGC0L01C	DAR routine (DAR1)	IEAQTM0L	IGC0L01C	TP	MA	SVC	4		
IGC0L05A	ABDUMP routine (ABDUMP1)	IEAQAD00	IGC0L05A	TP	LI	SVC	4		SVC 51

Figure 14-1. Module Directory (Part 8 of 17) (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.
IGC0L05B	SYSIN/SYSOUT Non-DASD Data Set Processor	IGC0L05B	IGC0L05B	CR		SVC	4		SVC 52
IGC0M01C	DAR routine (DAR2)	IEAQTM0M	IGC0M01C	TP	MB	SVC	4		
IGC0M05A	ABDUMP routine (ABDUMP14)	IEAQAD0M	IGC0M05A	TP	LI	SVC	4		SVC 51
IGC0M05B	REP I/O-Mount/Verify 2 routine	IGC0M05B	IGC0M05B	CR	JR	SVC	4		SVC 52
IGC0N01C	DAR routine (DAR3)	IEAQTM0N	IGC0N01C	TP	MC	SVC	4		
IGC0N05A	ABDUMP routine (ABDUMP15)	IEAQAD0N	IGC0N05A	TP	LI	SVC	4		SVC 51
IGC0N05B	REP I/O-SYSIN/SYSOUT Data Set Processor 1	IGC0N05B	IGC0N05B	CR	JS	SVC	4		SVC 52
IGC0N06C	Resume I/O routine	IGC0N06C	IGC0N06C	CR	JH	SVC	4		SVC 63
IGC0P01C	DAR routine (DAR4)	IEAQTM0P	IGC0P01C	TP	MD	SVC	4		
IGC0P05A	ABDUMP routine (ABDUMP16)	IEAQAD0P	IGC0P05A	TP	LI	SVC	4		SVC 51
IGC0P05B	REP I/O-Data Set Processor 1	IGC0P05B	IGC0P05B	CR	JT	SVC	4		SVC 52
IGC0Q05A	ABDUMP routine (ABDUMPQ)	IEAQAD0Q	IGC0Q05A	TP	LI	SVC	4		SVC 51
IGC0Q05B	REP I/O-SYSIN/SYSOUT Data Set Processor 2	IGC0Q05B	IGC0Q05B	CR	JS	SVC	4		SVC 52
IGC0Q06C	Checkpoint Exit routine	IGC0Q06C	IGC0Q06C	CR	JH	SVC	4		SVC 63
IGC0R01C	ABEND/STAE Interface 0 routine	IEAQTM0R	IGC0R01C	TP	BQ	SVC	4		
IGC0R05B	REP I/O-Data Set Processor 2	IGC0R05B	IGC0R05B	CR	JV	SVC	4		SVC 52
IGC0S01C	ABEND/STAE Interface 1 routine	IEAQTM0S	IGC0S01C	TP	BR	SVC	4		
IGC0S05B	REP I/O-Data Set Processor 1A	IGC0S05B	IGC0S05B	CR	JU	SVC	4		SVC 52
IGC0S06C	Checkpoint Message Module	IGC0S06C	IGC0S06C	CR	JI	SVC	4		SVC 63
IGC0T01C	ABEND/STAE Interface 2 routine	IEAQTM0T	IGC0T01C	TP	BS	SVC	4		
IGC0T05B	REP I/O-Access Method-Disposition routine	IGC0T05B	IGC0T05B	CR	JW	SVC	4		SVC 52
IGC0U01C	ABEND/STAE Interface 3 routine	IEAQTM0U	IGC0U01C	TP	BT	SVC	4		
IGC0U05B	DOS Tape Data Set Processor	IGC0U05B	IGC0U05B	CR	JY	SVC	4		SVC 52
IGC0V01C	ABEND/STAE Interface 4 routine	IEAQTM0V	IGC0V01C	TP	BU	SVC	4		
IGC0V05B	Restart Exit routine	IGC0V05B	IGC0V05B	CR	JW	SVC	4		SVC 52
IGC0W01C	ABEND/STAE Interface 5 routine	IEAQTM0W	IGC0W01C	TP	BV	SVC	4		
IGC0W05B	ISAM and BDAM Data Set Processor	IGC0W05B	IGC0W05B	CR		SVC	4		SVC 52
IGC0Z05A	SVC DUMP 2	IEAQAD0Z	IGC0Z05A	TP	MF	SVC	4		
IGC0001C	ABEND routine (ABEND0)	IEAQTM00	IGC0001C	TP	LL	SVC	4	ABEND	SVC 13
IGC0003E	WTO (MCS)	IEEMVWTO	IGC0003E	CCSL	FB	SVC	4	WTO	SVC 35
IGC0003E	WTO (Non-MCS)	IEENVWTO	IGC0003E	CCSL	FB	SVC	4	WTO	SVC 35
IGC0005A	SVC DUMP 1	IEAQAD0Y	IGC0005A	TP	ME	SVC	4	SNAP	SVC 51

Figure 14-1. Module Directory (Part 9 of 17) (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 Routinedule		
				Section	Chart ID		Type	Macro Instruction	SVC Instr.
IGC00060	STAE Service routine	IEAAST00	IGC00060	TP	BP	SVC	3	STAE	SVC 60
IGC0006C	Checkpoint Housekeeping 1	IGC0006C	IGC0006C	CR	JA	SVC	4	CHKPT	SVC 63
IGC0007B	Communications Task Router routine	IEECVCTR	IEECVCTR	CCSL	FG	SVC	4		SVC 72
IGC0007B	Communications Task Mini-Router routine (MCS)	IEECMCTR	IEECMCTR	CCSL	(Fig. 7-3)	SVC	4		SVC 72
IGC0008E	DDR SVC Router, Initiator, Terminator	IGC0008E	IGC0008E	*		SVC	4		SVC 85
IGC0101C	ABEND routine (ABEND1)	IEAQTM01	IGC0101C	TP	LM	SVC	4		SVC 13
IGC0103E	Write-to-Operator Reply routine	IEEVWTOR	IGC0103E	CCSL	FC	SVC	4	WTOR	SVC 35
IGC0105A	ABDUMP routine (ABDUMP2)	IEAQAD01	IGC0105A	TP	LI	SVC	4		SVC 51
IGC0105B	Restart Housekeeping 1	IGC0105B	IGC0105B	CR	JJ	SVC	4		SVC 52
IGC0106C	Checkpoint Housekeeping 2	IGC0106C	IGC0106C	CR	JB	SVC	4		SVC 63
IGC0107B	Communications Task 1052 Console Processor 1 routine (non-MCS)	IEECVPMX	IEECVPM	CCSL	FQ	SVC	4		SVC 72
IGC0107B	Communications Task 1052 Console Processor 1 routine (MCS)	IEECMPMX	IEECVPM	CCSL	FR	SVC	4		SVC 72
IGC0108E	Operator-Initiated DDR	IGC0108E	IGC0108E	*		SVC	4		SVC 85
IGC0203E	Write-to-Programmer (initialization routine)	IEFWTP00	IGC0203E	CCSL		SVC	4	WTO WTOR	SVC 35
IGC0205A	ABDUMP routine (ABDUMP3)	IEAQAD02	IGC0205A	TP	LI	SVC	4		SVC 51
IGC0205B	Restart Housekeeping 2	IGC0205B	IGC0205B	CR	JJ	SVC	4		SVC 52
IGC0206C	Checkpoint Housekeeping 3	IGC0206C	IGC0206C	CR	JC	SVC	4		SVC 63
IGC0207B	Communications Task 1052/ Printer Processor 2 routine (non-MCS)	IEECVPM1	IEECVPM1	CCSL	(Fig. 7-1)	SVC	4		SVC 72
IGC0207B	Communications Task 1052 Processor 2 routine (MCS)	IEECMPM1	IEECMPM1	CCSL	FS	SVC	4		SVC 72
IGC0208E	System-Initiated DDR (Load 1)	IGC0208E	IGC0208E	*		SVC	4		SVC 85
IGC0301C	ABEND routine (ABEND3)	IEAQTM03	IGC0301C	TP	LN	SVC	4		SVC 13
IGC0303E	Write-to-Programmer (processing routine)	IEFWTP01	IGC0303E	CCSL		SVC	4	WTO WTOR	SVC 35
IGC0305A	ABDUMP routine (ABDUMP4)	IEAQAD03	IGC0305A	TP	LI	SVC	4		SVC 51
IGC0308E	System-Initiated DDR (Load 2)	IGC0308E	IGC0308E	*		SVC	4		SVC 85
IGC0401C	ABEND routine (ABEND4)	IEAQTM04	IGC0401C	TP	LO	SVC	4		SVC 13
IGC0403E	Write-to-Programmer (error routine)	IEFWTP02	IGC0403E	CCSL		SVC	4	WTO WTOR	SVC 35
IGC0405A	ABDUMP routine (ABDUMP5)	IEAQAD04	IGC0405A	TP	LI	SVC	4		SVC 51
IGC0408E	DDR Tape Reposition (Load 1)	IGC0408E	IGC0408E	*		SVC	4		SVC 85

Figure 14-1. Module Directory (Part 10 of 17) (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.
IGC0501C	ABEND routine (ABEND5)	IEAQTM05	IGC0501C	TP	LP	SVC	4		SVC 13
IGC0505A	ABDUMP routine (ABDUMP6)	IEAQAD05	IGC0505A	TP	LI	SVC	4		SVC 51
IGC0505B	Repmain 1	IGC0505B	IGC0505B	CR	JK	SVC	4		SVC 52
IGC0506C	Check I/O routine	IGC0506C	IGC0506C	CR	JD	SVC	4		SVC 63
IGC0508E	DDRMSG MOD#1	IGC0508E	IGC0508E	*		SVC	4		SVC 85
IGC0605A	ABDUMP routine (ABDUMP7)	IEAQAD06	IGC0605A	TP	LI	SVC	4		SVC 51
IGC0605B	Repmain 2	IGC0605B	IGC0605B	CR	JL	SVC	4		SVC 52
IGC0608E	DDR Tape Reposition (Load 2)	IGC0608E	IGC0608E	*		SVC	4		SVC 85
IGC0701C	ABEND routine (ABEND7)	IEAQTM07	IGC0701C	TP	LQ	SVC	4		SVC 13
IGC0705A	ABDUMP routine (ABDUMP8)	IEAQAD07	IGC0705A	TP	LI	SVC	4		SVC 51
IGC0705B	Repmain 3	IGC0705B	IGC0705B	CR	JM	SVC	4		SVC 52
IGC0708E	DDR Recording	IGC0708E	IGC0708E	*		SVC	4		SVC 85
IGC0801C	ABEND routine (ABEND8)	IEAQTM08	IGC0801C	TP	LR	SVC	4		SVC 13
IGC0805A	ABDUMP routine (ABDUMP9)	IEAQAD08	IGC0805A	TP	LI	SVC	4		
IGC0805B	Repmain 4	IGC0805B	IGC0805B	CR	JM	SVC	4		SVC 52
IGC0808E	DDRMSG MOD#2	IGC0808E	IGC0808E	*		SVC	4		SVC 85
IGC0901C	ABEND routine (ABEND9)	IEAQTM09	IGC0901C	TP	LS	SVC	4		SVC 13
IGC0905B	Repmain 5	IGC0905B	IGC0905B	CR	JN	SVC	4		SVC 52
IGC0907B	Communications Task NIP Message Buffer Writer	IEECMWTL	IGC0907B	CCSL	FP	SVC	4		SVC 72
IGC1I07B	Communications Task Card Reader Open/Close routine	IEECVOCC	IEECVOC	CCSL	(Fig. 7-2)	SVC	4		SVC 72
IGC1107B	Communications Task Card Reader Processor routine	IEECVPMC	IEECVPM	CCSL	FU	SVC	4		SVC 72
IGC1107B	Communications Task Card Reader Processor (MCS)	IEECMPMC	IEECVPM	CCSL	FV	SVC	4		SVC 72
IGC2I07B	Communications Task Printer Open/Close routine	IEECVOCP	IEECVOC	CCSL	(Fig. 7-1, 7-2)	SVC	4		SVC 72
IGC2107B	Communications Task Printer Processor routine	IEECVPMP	IEECVPM	CCSL	(Fig. 7-1, 7-2)	SVC	4		
IGC2107B	Communications Task Printer Processor (MCS)	IEECMPMP	IEECVPM	CCSL		SVC	4		SVC 72
IGC001	Wait routine	IEAQSY50	IGC001	TS	BG	NUC	1	WAIT	SVC 1
IGC002	Post routine	IEAQSY50	IGC001	TS	BH	NUC	1	POST	SVC 2
IGC002+6	Post routine (branch e.p. from supervisor routines)	IEAQSY50	IGC001	TS	BH	NUC	1		
IGC003	Exit routine	IEAQET00	IGC003	EP	KB	NUC	1		SVC 3

Figure 14-1. Module Directory (Part 11 of 17) (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.
IGC004	GETMAIN routine	IEAQGM00	IEAQGM00	MSS	DA	NUC	1	GETMAIN	SVC 4
IGC005	FREEMAIN routine	IEAQGM00	IEAQGM00	MSS	DA	NUC	1	FREEMAIN	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	CA	NUC	2	XCTL	SVC 7
IGC008	Contents Supervision, common subroutines (e.p. for the LOAD macro instruction)	IEAQLK00	IEAQLK00	CS	CA	NUC	2	LOAD	SVC 8
IGC009	Delete routine	IEAQLK00	IEAQLK00	CS	CC	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,EE	NUC	1	TIME	SVC 11
IGC012	Contents Supervision, common subroutines (e.p. for the SYNCH macro instruction)	IEAQLK00	IEAQLK00	CS	CA	NUC	2	SYNCH	SVC 12
IGC014	SPIE routine	IEAQTB00	IGC014	TS	BF	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	CE	NUC	2	SEGLD SEGWT	SVC 37
IGC040	Extract routine	IEAQTB00	IGC014	TS	BD	NUC	1	EXTRACT	SVC 40
IGC040+8	Extract routine (branch e.p.)								
IGC041	Identify routine	IEAQID00	IGC041	CS	CB	NUC	2	IDENTIFY	SVC 41
IGC042	Attach routine	IEAQAT00	IGC042	TS	BA	NUC	2	ATTACH	SVC 42
IGC043	Stage 1 Exit Effector	IEAQEF00	IGC043	TS	BK	NUC	2	CIRB	SVC 43
IGC044	CHAP routine	IEAQCH00	IGC044	TS	BB,BC	NUC	1	CHAP	SVC 44
IGC044 +12	CHAP routine (branch e.p.)								
IGC045	Overlay Supervisor, resident module (e.p. for branch instruction or CALL macro instruction)	IEWSVOVR	IGC037	CS	CE	NUC	2	CALL	SVC 45
IGC046	TTIMER routine	IEAQST00	IGC046	TMS	EC,EG	NUC	1	TTIMER	SVC 46
IGC047	STIMER routine	IEAQST00	IGC046	TMS	EB,EF	NUC	2	STIMER	SVC 47
IGC048	Dequeue routine	IEAQENQ2	IGC048	TS	BJ	NUC	2	DEQ	SVC 48
IGC056	Enqueue routine	IEAQENQ2	IGC048	TS	BI	NUC	2	ENQ	SVC 56
IGC061	TESTRAN interpreter	IGC0006A	IEGHRSAV	CS		SVC	3	TTSAV	SVC 61
IGC062	Detach routine	IEAQED02	IGC062	TS	BE	NUC	2	DETACH	SVC 62

Figure 14-1. Module Directory (Part 12 of 17) (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.
IGC079	Set Status routine	IEAQSETS	IEAQSETS	TS	BW	NUC	1	STATUS	SVC 79
IGC109	Extended SVC Router Types 3 and 4	IGC116	IGC116	IH	AC	SVC	2		SVC109
IGC116	Extended SVC Router Type 1	IGC116	IGC116	IH	AC	SVC	1		SVC116
IGC117	Extended SVC Router Type 2	IGC116	IGC116	IH	AC	SVC	2		SVC117
IGE0660A	DDR Central	IGE0660A	IGE0660A	*		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			
IGFASR00	System Analysis module for the Model 65	IGFASR00	IGFASR00			SVC			
IGFASR01	MCH Error Recorder module	IGFASR01	IGFASR01			SVC			
IGFASR1A	Refresh Clear Channel module	IGFASR1A	IGFASR1A			SVC			
IGFASR1C	Instruction Retry Analysis	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			
IGFCAT	Channel-Check Handler Central (CCH)	IGFCATAP	IGFCAT	IH		NUC			
IGFCCH48	CCH 155 Analysis Module	IGFCCH48	IGFCCH48			NUC			
IGFCCH60	CCH 2860 Analysis Module	IGFCCH60	IGFCCH60			NUC			
IGFCCH68	CCH 145 Analysis Module	IGFCCH68	IGFCCH68			NUC			
IGFCCH70	CCH 2870 Analysis Module	IGFCCH70	IGFCCH70			NUC			
IGFCCH80	CCH 2880 Analysis Module	IGFCCH80	IGFCCH80			NUC			
IGFDDRSR	DDR SYSRES Module	IGFDDR10	IGFDDRSR	*		NUC			
IGFDDR01	DDR Resident Module	IGFDDRMV	IGFDDR01	*		NUC			
IGFDDR02	DDR Channel End Appendage	IGFDDR02	IGFDDR02	*		NUC			

*Routine discussed in I/O Supervisor PLM, GY28-6676.

Figure 14-1. Module Directory (Part 13 of 17) (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.
IGFDDR03	DDR Abnormal End Appendage	IGFDDR03	IGFDDR03	*		NUC			
IGFDDR05	DDR SYSRES Module	IGFDDR10 IGFDDR00 (without MCS)	IGFDDRSR IGFDDRSR	*		NUC			
IGFMCHE0	MCH Nucleus	IGFMCHE0	IGFMCHE0						
IGFMCHE1	MCH Console Write Module for System/370	IGFMCHE1	IGFMCHE1			SVC			
IGFMCHE2	MCH Error Recorder Module for System/370	IGFMCHE2	IGFMCHE2			SVC			
IGFMCHE3	MCH Emergency Error Recorder Module for System/370	IGFMCHE3	IGFMCHE3			SVC			
IGFMCHF0	MCH Initialization Module for System/360 Model 85 and System/370	IGFMCHF0	IGFMCHF0			SVC			
IGFMCHF4	Refresh Loader Module for System/360 Model 85	IGFMCHF4	IGFMCHF4			SVC			
IGFMCHF5	PDAR Terminator Analysis for System/360 Model 85 and System/370	IGFMCHF5	IGFMCHF5			SVC			
IGFMCHF6	Subsystem Interface Module for System/370 Model 165	IGFMCHF6	IGFMCHF6			SVC			
IGFMCH13	Preliminary Error Analysis Module for System/360 Model 85	IGFMCH13	IGFMCH13			SVC			
IGFMCH14	Repair Refresh Verification Module for System/360 Model 85	IGFMCH14	IGFMCH14			SVC			
IGFMCH15	Storage Protect Feature Analysis Module for System/360 Model 85	IGFMCH15	IGFMCH15			SVC			
IGFMCH16	Buffer Control Module	IGFMCH16	IGFMCH16			SVC			
IGFMCH17	Error Recorder Module for System/360 Model 85	IGFMCH17	IGFMCH17			SVC			
IGFMCH18	Alternate Multiply Control Module for System/360 Model 85	IGFMCH18	IGFMCH18			SVC			
IGFMCH19	Main Storage Analysis Module for System/360 Model 85	IGFMCH19	IGFMCH19			SVC			
IGFMCH20	Soft Machine-Check Handler for System/370 Model 155	IGFMCH20	IGFMCH20			SVC			
IGFMCH21	Main Storage Analysis Module for System/370 Model 155	IGFMCH21	IGFMCH21			SVC			
IGFMCH22	Storage Protect Feature Analysis Module for System/370 Model 155	IGFMCH22	IGFMCH22			SVC			

*Routine discussed in I/O Supervisor PLM, GY28-6676.

Figure 14-1. Module Directory (Part 14 of 17) (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.
IGFMCH23	Repair/Refresh Verification Module for System/370 Model 155	IGFMCH23	IGFMCH23			SVC			
IGFMCH30	Soft Machine-Check Handler for System/370 Model 165	IGFMCH30	IGFMCH30			SVC			
IGFMCH31	Preliminary Error Analysis Module for System/370 Model 165	IGFMCH31	IGFMCH31			SVC			
IGFMCH33	Main Storage Analysis Module for System/370 Model 165	IGFMCH33	IGFMCH33			SVC			
IGFMCH34	Storage Protect Feature Analysis Module for System/370 Model 165	IGFMCH34	IGFMCH34			SVC			
IGFMCH35	Repair/Refresh Verification Module for System/370 Model 165, Part 1	IGFMCH35	IGFMCH35			SVC			
IGFMCH36	Repair/Refresh Verification Module for System/370 Model 165, Part 2	IGFMCH36	IGFMCH36			SVC			
IGFMCH40	Soft Machine-Check Handler for System/370 Model 145	IGFMCH40	IGFMCH40			SVC			
IGFMCH41	Preliminary Error Analysis for System/370 Model 145	IGFMCH41	IGFMCH41			SVC			
IGFMVTF1	System Analysis Module for System/360 Model 85 and System/370	IGFMVTF1	IGFMVTF1			SVC			
IGFMVTF2	System Analysis Module for System/360 Model 85 and System/370	IGFMVTF2	IGFMVTF2			SVC			
IGFMVTF3	System Analysis Module for System/360 Model 85 and System/370	IGFMVTF3	IGFMVTF3			SVC			
IGFN0000	MCH Resident Nucleus for								
	System/360 Model 65	IGFNUC00	IGFNUC00			NUC			
	System/360 Model 85	IGFMCH10	IGFMCH10			NUC			
	System/370	IGFMCHE0	IGFMCHE0			LINK			
IGFN0001	Console/SYSRES Clear Channel Module for								
	System/360 Model 65	IGFASR0A	IGFASR0A			SVC			
	System/360 Model 85	IGFASR5A	IGFASR5A			SVC			
IGFN0002	MCH Termination Module for								
	System/360 Model 65	IGFASR0A	IGFASR0A			SVC			
	System/360 Model 85	IGFMCH12	IGFMCH12			SVC			
IGF08501	Machine Status Control Module Part 1 for System/360 Model 85	IGF08501	IGF08501			SVC			
IGF08502	Machine Status Control Module Part 2 for System/360 Model 85	IGF08502	IGF08502			SVC			
IGF29601	Machine Status Control Module for System/370 Model 155	IGF29601	IGF29601			SVC			

Figure 14-1. Module Directory (Part 15 of 17) (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.
IGF29701	Machine Status Control for System/370 Model 145	IGF29701	IGF29701			SVC			
IGF2403D	APR VARY PATH Command Processor	IGC2403D	IGF2403D	*		SVC			
IGF24MPD	APR VARY PATH Command Processor (Load 1 MP)	IGC2403D	IGF24MPD	*		SVC			
IGF2503D	DDR SWAP Command Processor	IGC2503D	IGF2503D	*		SVC			
IGF34MPD	APR VARY PATH Command Processor (Load 2 MP)	IGC3403D	IGF34MPD	*		SVC			
IGF55301	Machine Status Control Module for System/370 Model 165	IGF29701	IGF29701			SVC			
INTEXTRN	Second CPU Interruption Analysis routine	IEAQFX00	IEAQFX00	IH		NUC			
INTMLFAL	Second CPU Recovery Management System Interface routine	IEAQFX00	IEAQFX00	IH		NUC			
INT025A	Routine in the I/O Supervisor that returns a request element to the free list	IEAQFX00 (IEAQFX and IECIOS macros)	IEAQFX00	EP		NUC			
IORGSW	I/O Switch (in I/O First-Level Interruption Handler)	IEAQNU02	IEAQNU02	IH		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	CE	LINK			
RMBRANCH	GETMAIN/FREEMAIN routines (branch e.p.)	IEAQGM00	IEAQGM00	MSS	DA	NUC	1		
SECMCI	SER0 routine								
	System/360, Model 40	IFBSR040	IFBSR040	IH	AK	LINK			
	System/360, Model 50	IFBSR050	IFBSR050	IH	AK	LINK			
	System/360, Model 65	IFBSR065	IFBSR065	IH	AK	LINK			
	System/360, Model 75	IFBSR075	IFBSR075	IH	AK	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			

*Routine discussed in I/O Supervisor PLM, GY28-6676.

Figure 14-1. Module Directory (Part 16 of 17) (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.
TASEARCH	Transient Area XCTL routine	IEAQTR33	IEAQTR00	CS	CA	NUC			
TATABCK	Transient Area Availability Check routine	IEAQTR33	IEAQTR00	IH	AD	NUC			
TAXEXIT	Transient Area Exit routine	IEAQTR33	IEAQTR00	EP	KC	NUC			
TAXRETRY	Transient Area XCTL routine	IEAQTR33	IEAQTR00	CS	CA	NUC			
TESTDSP	Task Removal routine	IEAQFX00	IEAQFX00	IH		NUC			
TRDISP	Trace routine (e.p. for the Dispatcher)	IEAQTRCE	IEAQTRCE			NUC			
TREX	Trace routine (e.p. for Ext. FLIH)	IEAQTRCE	IEAQTRCE			NUC			
TRIO	Trace routine (e.p. for I/O FLIH)	IEAQTRCE	IEAQTRCE			NUC			
TRPI	Trace routine (e.p. for PC FLIH)	IEAQTRCE	IEAQTRCE			NUC			
TRSVC	Trace routine (e.p. for SVC FLIH)	IEAQTRCE	IEAQTRCE			NUC			
USERORG	SVC table (start of user-assigned SVC numbers)	IEAQBK00	IEAQBK00	IH		NUC			

Figure 14-1. Module Directory (Part 17 of 17) (See legend before Part 1)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
ABDUMP routines		
"resident" module	IGC0A05A	LI
ABDUMP1	IGC0L05A	LI
ABDUMP1.5	IGC0C05A	LI
ABDUMP2	IGC0105A	LI
ABDUMP3	IGC0205A	LI
ABDUMP4	IGC0305A	LI
ABDUMP5	IGC0405A	LI
ABDUMP6	IGC0505A	LI
ABDUMP7	IGC0605A	LI
ABDUMP8	IGC0705A	LI
ABDUMP9	IGC0805A	LI
ABDUMP10 (resident routine)	IGC0A05A	LI
ABDUMP11	IGC0B05A	LI
ABDUMP12	IGC0J05A	LI
ABDUMP13	IGC0K05A	LI
ABDUMP14	IGC0M05A	LI
ABDUMP15	IGC0N05A	LI
ABDUMP16	IGC0P05A	LI
ABDUMPQ	IGC0Q05A	LI
TCAM ABDUMP 1	IGC0D05A	LJ
TCAM ABDUMP 2	IGC0E05A	LJ
TCAM ABDUMP 3	IGC0F05A	LJ
TCAM ABDUMP 4	IGC0G05A	LJ
ABDUMPH	IGC0H05A	None
ABDUMPI	IGC0I05A	None
SVC DUMP 1	IGC0005A	ME
SVC DUMP 2	IGC0Z05A	MF
ABEND routine		
Abnormal Termination Modular Overview		LK
ABEND0	IGC0001C	LL
ABEND1	IGC0101C	LM
ABEND3	IGC0301C	LN

Figure 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 1 of 15)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
ABEND4	IGC0401C	LO
ABEND5	IGC0501C	LP
ABEND7	IGC0701C	LQ
ABEND8	IGC0801C	LR
ABEND9	IGC0901C	LS
ABEND11	IGC0B01C	LT
ABEND12	IGC0C01C	LU
ABEND13	IGC0D01C	LV
ABEND15	IGC0F01C	LW
ABEND16	IGC0G01C	LX
ABEND20	IGC0K01C	LY
ABTERM routine	IEA0AB00 IEA0AB01	LF LF
ABTERM Prologue routine	IEA0PL00	LH
ABTERM Setsubs subroutine	SETSUBS	LG
Alternate Multiply Control module for System/360 Model 85	IGFASR6F IGFMCH18	None None
Alternate Path Retry APR VARY PATH Command Processor	IGF2403D IGF24MPD MP IGF34MPD	None
ATTACH routine	IGC042	BA
Attention routine	IEEBA1	FA
BLDL routine	IECPBLDL	None
Buffer Control module	IGFMCH16	None
CDEXIT routine	CDDESTRY CDEXIT CDHKEEP	KE KE KE
Channel-Check Handler Central routine	IGFCAT	None
Channel-Check Handler 155 Analysis routine	IGFCCH48	None
Channel-Check Handler 2860 Analysis routine	IGFCCH60	None
Channel-Check Handler 145 Analysis routine	IGFCCH68	None
Channel-Check Handler 2870 Analysis routine	IGFCCH70	None
Channel-Check Handler 2880 Analysis Routine	IGFCCH80	None
CHAP routine	IGC044 IGC044+12	BB,BC

Figure 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 2 of 15)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
Checkpoint routines		
Checkpoint Housekeeping 1 routine	IGC0006C	JA
Checkpoint Housekeeping 2 routine	IGC0106C	JB
Checkpoint Housekeeping 3 routine	IGC0206C	JC
Check I/O routine	IGC0506C	JD
Preserve 1 routine	IGC0A06C	JE
Preserve 2 routine	IGC0D06C	JE
Checkmain 1 routine	IGC0F06C	JF
Checkmain 2 routine	IGC0G06C	JF
Checkmain 3 routine	IGC0H06C	JG
Resume I/O routine	IGC0N06C	JH
Checkpoint Exit routine	IGC0Q06C	JH
Checkpoint Message Module	IGC0S06C	JI
Communications Task Attention Interruption Handler	IEEBA1	FA
Communications Task Console Switch routine (MCS)		
Load 1	IGCXL07B	FJ
Load 2	IGCXM07B	FK
Load 3	IGCXN07B	FK
Load 4	IGCXO07B	FL
Communications Task Device Interface routine (MCS)	IEECMDSV	FM
Communications Task DOM Service routine (MCS)	IEECMDOM	FO
Communications Task External Interruption Handler routine	IEEBC1PE	FA
Communications Task External Processor routine (non-MCS)	IGCXL07B	FH
Communications Task Initialization routine	IEECVCTI	FD
Communications Task Mini-Router	IGC0007B	(Fig. 7-3)
Communications Task Misc. Lookup Services routine	IEEVRFRX	None
Communications Task NIP Message Buffer Writer	IGC0907B	FP
Communications Task Reply Processor routine	IEE1203D	None
Communications Task MCS Reply Processor routine	IEE1A03D	None
Communications Task Error Message routine	IEE1B03D	None
Communications Task Request Block (RB)	IEECVPRB	None
Communications Task Router routine (non-MCS)	IGC0007B	FG

Figure 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 3 of 15)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
Communications Task Router routine (MCS)	IEECVCTW	FI
Communications Task TCB	IEECVTCB	None
Communications Task Unit Control Tables	IEECVUCM	None
Communications Task Wait routine (non-MCS)	IEECIR45	FF
Communications Task WTO Handler	IGC0003E	FB
Communications Task WTOR Handler	IGC0103E	FC
Communications Task WTO(R) Service routine (MCS)	IEECMWSV	FN
Communications Task 1052 Console Open/Close routine	IGC0I07B	FT
Communications Task 1052 Console Processor 1 routine (non-MCS)	IGC0107B	FQ
Communications Task 1052 Console Processor 1 routine (MCS)	IGC0107B	FR
Communications Task 1052 Console Processor 2 routine (MCS)	IGC0207B	FS
Communications Task 1052/Printer Processor 2 routine (non-MCS)	IGC0207B	(Fig. 7-1)
Communications Task Printer Open/Close routine	IGC2I07B	(Fig. 7-1, 7-2)
Communications Task Printer Processor routine	IGC2107B	(Fig. 7-1, 7-2)
Communications Task Card Reader Open/Close routine	IGC1I07B	(Fig. 7-2)
Communications Task Card Reader Processor routine	IGC1107B	FU, FV
Communications Task 2740 Console Processor	IEEC274O	FW
Communications Vector Table (CVT)	IEACVT	None
Console/SYSRES Clear Channel routine	IGFN0001	None
Contents Supervision Common Subroutines		
entry points for search phase	CDADVANS CDCONTRL also called IEAQCS02	CA CA
entry point for scheduling phase	CDEPILOG also called IEAQCS03	CA
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	CA
entry point for the LOAD macro instruction	IGC008	CA
entry point for the SYNCH macro instruction	IGC012	CA
CPU Analysis module	IGFASR0B	None

Figure 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 4 of 15)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
Damage Assessment Routine (DAR) 1	IGC0L01C	MA
Damage Assessment Routine (DAR) 2	IGC0M01C	MB
Damage Assessment Routine (DAR) 3	IGC0N01C	MC
Damage Assessment Routine (DAR) 4	IGC0P01C	MD
Data control block (DCB) for the SYS1.LINKLIB data set	LINKDCB	None
Data control block (DCB) for the SYS1.SVCLIB data set	SVCDCB	None
Data extent block (DEB) for the SYS1.LINKLIB data set	LINKDEB	None
Data extent block (DEB) for the SYS1.SVCLIB data set	SVCDEB	None
Decimal Simulator routines for Models 91 and 195		
Add/Subtract/Zero-and-Add Decimal routine	DECASP	MI
Analyzer/End routine	DECD0 DECNEND	ML
Compare Decimal routine	DECCP	MH
Divide Decimal routine	DECDP	MK
Multiply Decimal routine	DECMP	MJ
Simulator Control routine	DECENT	MG
Delete routine	IGC009	CC
Dequeue routine	IGC048	BJ
Dequeue TCB routine	IEADQTCB	LC
Detach routine	IGC062	BE
Dispatcher	IEA0DS	KF-KL
Dynamic Device Reconfiguration		
DDR Central	IGE0660A	None
DDR SVC Router, Initiator, Terminator	IGC0008E	None
DDR Operator-Initiated	IGC0108E	None
DDR System-Initiated (Load 1)	IGC0208E	None
DDR System-Initiated (Load 2)	IGC0308E	None
DDR Tape Reposition (Load 1)	IGC0408E	None
DDRMSG MOD#1	IGC0508E	None
DDR Tape Reposition (Load 2)	IGC0608E	None
DDR Recording	IGC0708E	None
DDRMSG MOD#2	IGC0808E	None
DDR Resident Module	IGFDDR01	None

Figure 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 5 of 15)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
DDR Channel End Appendage	IGFDDR02	None
DDR Abnormal End Appendage	IGFDDR03	None
DDR SYSRES Routine	IGFDDR05	None
DDR SYSRES Routine	IGFDDRSR	None
DDR SWAP Command Processor	IGF2503D	None
Device Independent Display Operator		
Console Support (DIDOCS) routines		
Asynchronous Error routine	IEECVETC	HK
Cleanup routine	IEECVFTG	II
Command routine	IEECVET4	HS
Delete 1 routine	IEECVET6	IA
Delete 2 routine	IEECVET7	IB
Delete 3 routine	IEECVET8	IC
Delete 4 routine	IEECVET9	ID
Display 1 routine	IEECVET2	HO
Display 2 routine	IEECVET3	HP
Display 3 routine	IEECVFT2	HQ
Light Pen/Cursor Service routine	IEECVETF	IE
Message 1 routine	IEECVETD	HL
Message 2 routine	IEECVETE	HM
Message 3 routine	IEECVFTD	HN
Multiple-Line WTO (Non-MCS) Load 1	IEECVML1	GD
Multiple-Line WTO (Non-MCS) Load 2	IEECVML2	GE
Multiple-Line WTO (MCS) Load 1	IEECVML3	GD
Multiple-Line WTO (Non-MCS) Load 3	IEECVML4	GF
Multiple-Line WTO (MCS) Load 2	IEECVML5	GE
Multiple-Line WTO (MCS) Load 3	IEECVML6	GF
Multiple-Line WTO (MCS) Load 4	IEECVML7	GG
Model 85 I/O routine	IEECVETH	HJ
Open/Close routine	IEECVETG	HE

Figure 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 6 of 15)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
Options routine	IEECVETA	HT
PFK routine 1	IEECVFTA	IG
PFK routine 2	IEECVFTB	IH
Processor 0, Load 1	IEECVFT1	HA
Processor 0, Load 2	IEECVFTZ	HB
Processor 1, Load 1	IEECVET1	HC
Processor 1, Load 2	IEECVETZ	HD
Roll Mode routine	IEECVETJ	HR
Status Display Interface 1	IEECVFTL	IJ
Status Display Interface 2	IEECVFTM	IK
Status Display Interface 3	IEECVFTN	IL
Status Display Interface 4	IEECVFTO	IM
Status Display Interface 5	IEECVFTP	IN
Status Display Interface 6	IEECVFTQ	IO
Status Display Interface 7	IEECVFTT	IP
Timer Interpreter routine	IEECVETK	IF
2250 I/O-1 routine	IEECVETP	HF
2250 I/O-2 routine	IEECVETQ	HG
2260 I/O-1 routine	IEECVETR	HH
2260 I/O-2 routine	IEECVFTR	HI
End-of-Task (EOT) routine	EOT	LA
Enqueue routine	IGC056	BI
ENQ/DEQ Purge routine	IEAQEQ01	None
Erase Phase routine	IEAQERA	None
Error Check Circuitry Verification module	IGFASR1D	None
Error Recorder module for System/370 for System/360 Model 85	IGFMCHE2 IGFMCH17	None None
ESR Extended SVC Router		
Type 1 Router	IGC116	AC
Type 2 Router	IGC117	AC
Type 3 and 4 Routers	IGC109	AC

Figure 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 7 of 15)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
EXCP Counter routine	IEASMFEX	MM
EXCP Supervisor	IECXCP	None
Exit routine	IGC003	KB
External First-Level Interruption Handler	IEAQEX00	AH,AI
Extract routine	IGC040	BD
branch entry point	IGC040+8	BD
First CPU Signal routine	FLASH	None
FREEMAIN routine		
branch entry point	FMBRANCH	DA
branch entry point to free a region	FREEPART	DA
entry point for S-form FREEMAIN macro instruction	IGC005	DA
GETMAIN routine		
branch entry point to allocate a region	GETPART	DA
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
GETPART/FREEPART routine	IEAQPR	DB
Graphic Console Initialization routine	IEECVGCI	FE
HALT and WRITELOG CLOSE routine	IEE1403D	(Fig.7-10)
Identify routine	IGC041	CB
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	AJ
I/O Interruption Supervisor	IECINT	None
I/O Supervisor Transient Area	IEAERWA	None
I/O Switch (in I/O FLIH)	IORGSW	None

Figure 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 8 of 15)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
Job File Control Blocks (JFCBs) for the log data sets	IEEVLOGJ	(Fig.7-10)
Link Pack Area Queue	IEAQLPAQ	None
Log and WRITELOG Post routine	IEE1603D	(Fig.7-10)
Log Writer	IEELWAIT	GA
Log (Write-to-Log SVC 36)		
Load 1	IEE0303F	GB
Load 2 - Log Data Set Open/Close	IEE0403F	GC
Machine Status Control Module for		
System/360 Model 85 (Part 1)	IGF08501	None
System/360 Model 85 (Part 2)	IGF08502	None
System/370 Model 145	IGF29701	None
System/370 Model 155	IGF29601	None
System/370 Model 165	IGF55301	None
Main Storage Analysis Module for		
System/360 Model 85	IGFMCH19	None
System/370 Model 155	IGFMCH21	None
System/370 Model 165	IGFMCH33	None
Main Storage Scan module	IGFASR2D	None
Master Scheduler Initialization routine	IEEVIPL	(Fig.7-10)
Master Scheduler Resident Table	IEEMSER	None
Master Scheduler TCB	IEAMSTCB	None
MCH Console Write routine	IGFMCHE1	None
MCH Emergency Error Recorder module	IGFMCHE3	None
MCH Error Recorder module for		
System/360 Model 65	IGFASR01	None
System/360 Model 85	IGFMCH17	None
System/370	IGFMCHE2	None
MCH Initialization module	IGFMCHF0	None
MCH Nucleus	IGFMCHE0	None
MCH Resident Nucleus module	IGFN0000	None
MCH Termination routine	IGFN0002	None
Nucleus Initialization Program (NIP)	IEANIP4	None

Figure 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 9 of 15)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
Overlay Supervisor		
nonresident module	IEWSZOVR	CE
resident module		
entry point for SEGLD or SEGWT macro instruction	IGC037	CE
entry point for a branch instruction or CALL macro instruction	IGC045	CE
PDAR Termination Analysis module for		
System/360 Model 65	IGFASR20	None
System/360 Model 85	IGFMCHF5	None
System/370	IGFMCHF5	None
Post routine		
branch entry point for I/O Supervisor routines	IEAOPT01	BH
branch entry point for I/O Supervisor routines and for supervisor routines	IEAOPT02	BH
entry point for the POST macro instruction	IGC002	BH
branch entry point for supervisor routines	IGC002+6	BH
Preliminary Error Analysis module for		
System/360 Model 85	IGFMCH13	None
System/370 Model 145	IGFMCH41	None
System/370 Model 165	IGFMCH31	None
Program Check First-Level Interruption Handler	IEAQPK00	AF,AG
Program Fetch routine		
entry point for the Overlay Supervisor (IEWSZOVR)	IEWFBOSV	CD
entry point for the Transient Area Fetch routine	IEWFTRAN	CD
entry point for Contents Supervision common subroutines	IEWMSEPT	CD
Program Fetch Channel-End Appendage routine	FTCE01	CD
Program Fetch PCI Appendage routine	FTPCI01	CD
Purge Timer routine	IEAQPGTM	LD
QCB queues, origin of	IEAQQCB0	None
Refresh Clear Channel module		
for System/360, Model 65 and Model 65 Multiprocessor	IGFASR1A	None

Figure 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 10 of 15)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
Refresh Loader module		
for System/360, Model 65 and Model 65 Multiprocessor	IGFASR10	None
for System/360, Model 85	IGFMCHF4	None
Refresh/Repair Verification module for		
System/360 Model 85	IGFMCH14	None
System/370 Model 155	IGFMCH23	None
System/370 Model 165 (Part 1)	IGFMCH35	None
System/370 Model 165 (Part 2)	IGFMCH36	None
Reply Purge routine (also called the WTOR Purge routine)	IEECVPRG	None
Release Loaded Programs routine	IEAQABL	LE
Release Main Storage routine	IEAQSPET	LE
Restart Routines		
DOS Tape Data Set Processor	IGC0U05B	JY
ISAM and BDAM Data Set Processor	IGC0W05B	None
Restart Job Management-SMB Reader	IGC0005B	None
Restart Housekeeping 1	IGC0105B	JJ
Restart Housekeeping 2	IGC0205B	JJ
Repmain 1 routine	IGC0505B	JK
Repmain 2 routine	IGC0605B	JL
Repmain 3 routine	IGC0705B	JM
Repmain 4 routine	IGC0805B	JM
Repmain 5 routine	IGC0905B	JN
REP I/O-JFCB Processor 1	IGC0G05B	JO
REP I/O-JFCB Processor 1A	IGC0G95B	JO
REP I/O-JFCB Processor 2	IGC0I05B	JO
REP I/O-Dummy Data Set Processor	IGC0H05B	JP
REP TCAM Data Set Processor	IGC0J05B	JX
REP I/O-Mount/Verify 1 (Non Direct Access) routine	IGC0K05B	JQ
REP I/O-Mount/Verify 2 (Direct Access) routine	IGC0M05B	JR
REP I/O-SYSIN/SYSOUT Data Set Processor 1	IGC0N05B	JS
REP I/O-SYSIN/SYSOUT Data Set Processor 2	IGC0Q05B	JS

Figure 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 11 of 15)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
REP I/O-Data Set Processor 1	IGC0P05B	JT
REP I/O-Data Set Processor 1A	IGC0S05B	JU
REP I/O-Data Set Processor 2	IGC0R05B	JV
REP I/O-Access Method-Disposition routine	IGC0T05B	JW
Restart Exit routine	IGC0V05B	JW
SYSIN/SYSOUT Non-DASD Data Set Processor	IGC0L05B	None
Rollout/Rollin module	IEAQRORI	DC-DI
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	CE
SER0 routine		
resident module	IEAMCH00	AK
nonresident modules (for System/360 Models 40, 50, 65, 75)	IFBSER00 SECMCI	AK AK
SER1 routine (for System/360 Models 40, 50, 65, 75)	IEAMCH00	AL
(for System/360 Models 91, 95, 195)	IEAMCH00	AM
STATUS Service routine	IGC079	BW
SMF EXCP Counter routine	IEASMFEX	MM
SMF Storage routines	FMSMFCRE GMSMFCRE	DA DA
SMF Wait Time Collection routine	IEAQWAIT	MO
SMF Time/Output Limit Expiration routine	IEATLEXT	MN
Soft Machine-Check Handler for		
System/370 Model 145	IGFMCH40	None
System/370 Model 155	IGFMCH20	None
System/370 Model 165	IGFMCH30	None
SPIE routine	IGC014	BF
STAE Service routine	IGC00060	BP
ABEND/STAE Interface 0 routine	IGC0R01C	BQ
ABEND/STAE Interface 1 routine	IGC0S01C	BR
ABEND/STAE Interface 2 routine	IGC0T01C	BS
ABEND/STAE Interface 3 routine	IGC0U01C	BT

Figure 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 12 of 15)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
ABEND/STAE Interface 4 routine	IGC0V01C	BU
ABEND/STAE Interface 5 routine	IGC0W01C	BV
Stage 1 Exit Effector	IGC043	BK
Stage 2 Exit Effector	IEA0EF00	BL
Stage 3 Exit Effector		
entry points for the Dispatcher	ERFETCH	BM
	IEA0EF03	BM
entry point for an I/O error-handling routine	IECXTLER	BM
STIMER routine	IGC047	EB,EF
Storage Protection Feature Analysis module for System/360 Model 85	IGFMCH15	None
System/370 Model 155	IGFMCH22	None
System/370 Model 165	IGFMCH34	None
Storage Protect Feature Test Module	IGFASR0D	None
Subsystem Interface Module for System/370 Model 165	IGFMCHF6	None
Subsystem Purge routine	IEAASPRG	None
SVC First-Level Interruption Handler	IEAQSC00	AA
SVC Purge routine	IGC016	None
SVC Second-Level Interruption Handler	IEAQTR00	AB
SVC Table		
start of IBM-assigned SVC numbers	IBMORG	None
start of user-assigned SVC numbers	USERORG	None
SWAP Command Processor	IGF2503D	None
System Analysis module for System/360, Model 65 and Model 65 Multiprocessor	IGFASR00	None
System/360 Model 85 and System/370		
Part 1	IGFMVTF1	None
Part 2	IGFMVTF2	None
Part 3	IGFMVTF3	None
System error TCB (associated with I/O Supervisor transient areas)	IEAERTCB	None
Task Removal routine	TESTDSP	None
Task Switching routine	IEA0DS02	BN,BO
Terminal Attention Exit Element Purge routine	IEAKJXP	LB

Figure 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 13 of 15)

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,EE
Timer Second-Level Interruption Handler		
branch entry point	IEAQTD00	ED,EH
entry point for the Dispatcher	IEAQTD01	ED,EH
branch entry point	IEAQTE00	ED,EH
entry point for the External First-Level Interruption Handler	IEA0TI00	ED,EH
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	KC
entry point for the common subroutines of Contents Supvsn.	TAXEXIT	KC
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	None None
Transient Area Refresh routine	IEAQTR02	KD

Figure 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 14 of 15)

Name of Routine, Control Block, Table, Transient Area	Entry Point Name(s)	Chart ID
Transient Area XCTL routine	IEAQTR03 TASEARCH TAXRETRY	CA CA CA
TTIMER routine	IGC046	EC, EG
Type-1 SVC Exit routine	IEA0XE00	KA
Type-1 SVC Switch (in SVC First-Level Interruption Handler)	IEATYPE1	None
Validity Check routine	IEA0VL00	None
VARY PATH Command Processor	IGF2403D IGF24MPD MP IGF34MPD	None
Wait routine	IGC001	BG
Write-to-Log routine	IEE0303F	GB
Write-to-Operator routine (MCS)	IGC0003E	FB
Write-to-Operator routine (Non-MCS)	IGC0003E	FB
Write-to-Programmer routine		
Initialization	IGC0203E	None
Processing	IGC0303E	None
Error	IGC0403E	None
WRITELOG Available Log Data Set routine	IEEVLOUT	(Fig.7-10)
WRITELOG Dispatch routine	IEEVLDSP	(Fig.7-10)
WRITELOG Get Region routine	IEEPLDSP	None
WRITELOG Log Initialization routine	IEEVLIN	(Fig.7-10)
WRITELOG Log Writer routine	IEEVLWTR	(Fig.7-10)
WRITELOG Master Wait routine	IEEVWAIT	(Fig.7-10)
WRITELOG Open Device routine	IEEVLOPN	(Fig.7-10)
WTOR Purge routine (also called the Reply Purge routine)	IEECVPRG	None

Figure 14-2. Directory of Entry Point Names and Flowchart Identifications (Part 15 of 15)

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 LOAD 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 (ABEND0)
SVC 14	SPIE routine
SVC 15	ERREXCP (reinstate system error task)
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)

Figure 14-3. Table of Routines Invoked by SVC Instructions (Part 1 of 2)

SVC Instruction	Name of Routine
SVC 40	Extract routine
SVC 41	Identify routine
SVC 42	Attach routine
SVC 43	Stage 1 Exit Effector
SVC 44	CHAP 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
SVC 85	Dynamic Device Reconfiguration routine
SVC 87	Delete Operator Message routine
SVC 109	Type 3 and Type 4 Extended SVC Router routine
SVC 116	Type 1 Extended SVC Router routine
SVC 117	Type 2 Extended SVC Router routine

Note: Only those routines that are used by the supervisor are included in this list.

Figure 14-3. Table of Routines Invoked by SVC Instructions (Part 2 of 2)

ROUTINE SYNOPSES

Each major routine used by the supervisor is briefly described.

ABDUMP routine: (Chart LI) 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 during an abnormal termination, or by a user routine at any time.

ABEND routine: (Charts LK-LY) Invokes the ABEND/STAE interface routine if STAE processing is indicated. Transfers control to the Damage Assessment Routine (DAR) 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. Depending on the type of ABEND request, terminates either a specific task and its incomplete subtasks, or all tasks of a job step. Issues WTP messages for type-1 SVC routines when possible. 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. Frees the control blocks, main storage, and other resources used by a terminating task and its incomplete subtasks.

ABTERM routine: (Charts LF, LG) 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 Proloque routine: (Chart LH) 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.

Alternate Path Retry (APR): Not model-dependent. Allows an I/O operation that has developed an error on one channel to be retried on another channel (if another channel is assigned to the device performing the I/O operation) by causing channel-detected errors to be retried in a selective manner on the available paths to a device. As paths are found to be inoperative, they are marked offline, thus preventing unnecessary retry from being initiated to failing paths.

Also provides the capability to VARY a path to a device online or offline, using the VARY PATH command. VARY PATH command processor is part of the Master Scheduler (SVC 34). Last path to a device will not be varied offline. Four paths to each device supported; teleprocessing paths not supported.

(For a full description of Alternate Path Retry, refer to the Input/Output Supervisor PLM.

Attach routine: (Chart BA) 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: (Chart FA) 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 KE) Determines if there is an outstanding request for use of a recently completed module. If so, schedules reentry to the module for a waiting requester. 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, 2880, 145 or 155 channels (Model 65 and higher). Receives control from the I/O Supervisor via a branch when a channel error occurs. It performs two major functions. It provides an analysis of the channel logout information in the error recovery procedure interface block (ERPIB) to aid the appropriate device-dependent error recovery procedure in setting up for a retry of the failing operation by the I/O Supervisor. CCH also records environmental data about the channel error in a channel error inboard record entry. This record entry is later written onto the SYS1.LOGREC by the outboard recorder routine (OBR) of the I/O Supervisor. An operator message is issued each time a channel error is recorded. (For a full description of the Channel-Check Handler, refer to the Input/Output Supervisor PLM.

CHAP routine: (Charts BB, BC) 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: (Charts JA-JI) Intercept a task's I/O requests, copy the task's main storage region into a user-supplied data set, and record 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: (Chart FA) 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: (Chart FH) Switches control from the principal console device to the alternate console device, and vice versa.

Communications Task Initialization routine: (Chart FD) 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.

Communications Task MCS Reply Processor routine: Processes replies given by the operator in an MCS environment in response to program messages written via the WTOR macro instruction.

Communications Task Error Message routine: Assembles, edits, and broadcasts accepted replies to a WTOR macro instruction for the Communications Task MCS Reply Processor routine, and writes error messages to the operator when replies are in error.

Communications Task Router routine: (Chart FG) 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: (Chart FF) 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: (Chart FT) 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: (Charts FQ, FR, FS, FU, FV, FW) 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.

Contents Supervision common subroutines: (Chart CA) 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.

Console Support routines: Added to SVC 72 for READ, WRITE, OPEN, and CLOSE functions for the IBM 1052 Printer-Keyboard, as well as OPEN and CLOSE functions for the 2740 Communications Terminal. These routines perform buffer management for the 1052 Printer-Keyboard only if Multiple Console Support (MCS) is included.

Damage Assessment routine (DAR): (Charts MA-MD) Receives control from various ABEND modules. Attempts to write a core image dump, reinstates or initiates reinstatement of a failing task or region, and informs the operator if this is impossible so that the operator may halt system processing.

Decimal Simulator routines: (Charts MG-ML) Perform decimal arithmetic instructions on Models 91 and 195. 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 CC) 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 BJ) 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 requester is ready. Determines if a readied requester 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 requester's task is of higher priority than the caller's, returns control to the caller. Otherwise, returns control to the readied requester, 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: (Chart LC) Is invoked by either the EOT routine or ABEND16 during a normal or abnormal termination. Removes a specified TCB from the TCB queue.

Detach routine: (Chart BE) 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 KF-KL) 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, starts or resumes timing of the task to be given control, and branches to the GTF routines, if active, or the Trace routine to record information about the task switch.

Display Operator Console Support routines: (Charts HA-IP) Added to SVC 72 only if Multiple Console Support (MCS) is included. These routines provide uniform console support for the 2250 Display Unit (Models 1 and 3), the 2260 Display Station (Model 1 with 2848 Display Control, Model 3), and the Model 85 CRT Display (Feature 5450).

Dynamic Device Reconfiguration (DDR): Not model-dependent; standard for M65MP. Allows a demountable volume to be moved from one device to another, and repositioned if necessary, without abnormally terminating the affected job or reperforming IPL. A request to move a volume may be initiated by the operator with the SWAP command; the SWAP command processor is part of the Master Scheduler (SVC 34). System may request a swap of volumes following a permanent I/O error for non-SYSRES devices (interface through OBR/SDR), or following an error in a system fetch operation for SYSRES devices (interface through TA Fetch or error fetch sequence). (For a full description of Dynamic Device Reconfiguration, refer to the Input/Output Supervisor PLM.

End-of-Task (EOT) routine: (Chart LA) 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 BI) 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 requester, returns control to the current requester, with or without a return code (depending on the RET parameter). If the requested resource is already enqueued for another requester, either of two functions are performed, depending on the RET parameter: the current requester is placed in a wait condition, pending the availability of the resource; or control is returned to the current requester, 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 ABEND 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): Is invoked by the EOT routine or ABEND 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: (Chart KB) 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.

Extended SVC Router: (Chart AC) Provide linkage to Supervisor Service routines by logically extending the routing capability of the SVC Interruption Handlers. ESR accomplishes this via a secondary routing

---

[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.

algorithm based on a parameter established prior to issuance of one of the ESR SVCs (116, 117, or 109).

External First-Level Interruption Handler: (Charts AH,AI) Saves the caller's register contents in the TCB and the external old PSW in the current RB. Branches to the GTF routines, if active, or the Trace routine to record information about the external interruption. 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.

In a Model 65 Multiprocessing System, 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 BD) Moves the contents of selected TCB fields to a specified area of main storage.

FREEMAIN routine: (Chart DA) 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.

In a Model 65 Multiprocessing System, branches to the Vary Storage Offline (IFSVRYOF) subroutine to process VQEs which apply to the freed area of main storage.

Generalized Trace Facility (GTF): Assists in tracing program flow by monitoring and recording system events. Initiated by the operator issuing the START command; executes as a system task. When active, the optional Trace Table facility must be disabled.

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 CB) 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: (Chart AJ) 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.

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): Contains 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 7-10) Posts the ECB representing the appropriate command. For log commands also issues a WTL macro instruction.

Machine-Check Handler (MCH): Is available for System/360 Model 65, Model 65 Multiprocessor (MCH/65) and Model 85 (MCH/85) and

System/370. Receives control via hardware loading of the machine check new PSW. This program consists of a resident module, and transient modules which reside on the SYS1. SVCLIB data set. It attempts to recover from a machine check interruption.

MCH/65 first determines if the instruction that was being executed when the machine-check interruption occurred can be retried, and, if retry is possible, re-executes the instruction. This function is handled by machine recovery facilities on the System/360 Model 85 and System/370. If, however, instruction retry is not possible, MCH attempts to assess the damage to the program that was interrupted, and, in some models, repair that damage.

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 complete description of the Machine-Check Handler program consult the manual appropriate for the model being considered.

- Machine-Check Handler for the Model 65 PLM
- Machine-Check Handler for the Model 85 PLM
- Machine-Check Handler for the System/370 Models 135 and 145
- Machine-Check Handler for the System/370 Models 155 and 165 PLM

---

[1]The operator may then load the SEREP program in order to format and print diagnostic information from the CPU logout area.

Master Scheduler Initialization routine: (Figure 7-10) 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. In the Model 65 Multiprocessing System, NIP also initializes those parts of the nucleus that are unique to multiprocessing.

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 CE) 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.

Overlay Supervisor (resident module): (Chart CE) 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: (Chart BH) 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,AG) Saves register contents in program check register save area in lower main storage. Then branches to the Monitor Call Interrupt Handler to filter out valid requests for monitoring. These requests are recorded by GTF (if active), and control is returned directly to the user's program or the dispatcher if a task switch is necessary. If it is not a monitoring request, it is a valid program check and control is returned to the Program Check FLIH after recording the program check interruption in GTF, if GTF is active. If GTF is not active, and 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.

In a Model 65 Multiprocessing System, determines if the interruption was caused by an SSM instruction. If it was, sets the supervisor lock byte if complete enablement is not indicated, records the interruption from the SSM instruction in the GTF trace area, if GTF is active, and returns control to the interrupted routine. Before processing other types of program interruptions, sets the supervisor lock byte.

Program Fetch routine: (Chart CD) 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 CD) 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 CD) 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 LD) Is invoked by the EOT routine or ABEND1 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 LE) Is invoked by the EOT routine or ABEND16 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 LE) Is invoked by the EOT routine or ABEND16 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 JJ-JY) 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-DI) 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 CE) 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 AK) 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 AK) 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 AL,AM) 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: (Chart BW) Sets all tasks of the specified job step nondispatchable by setting the TCBPR0 flags in their TCBs.

SHOLDTAP routine: In a Model 65 Multiprocessing System, 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.

SMF EXCP Counting routine: (Chart MM) Counts and records the number of references to user data sets. Compares EXCP count to output limit specified for SYSOUT data sets.

SMF Storage routines: (Chart DA) Records the number of 2K blocks required within the region for problem program execution.

SMF Time/Output Limit Expiration routine: (Chart MN) Provides an interface with a user time limit expiration routine and with a user output limit routine.

SMF Wait Time Collection routine: (Chart MO) Collects and records system wait time information.

SPIE routine: (Chart BF) 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 BP-BV) 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 BK) Creates and initializes an interruption request block (IRB) to schedule and control execution of a user exit routine.

Stage 2 Exit Effector: (Chart BL) 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: (Chart BM) 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: (Charts EB-EF) Builds and places on the timer queue the elements that represent specified time intervals.

SVC First-Level Interruption Handler: (Chart AA) Saves the caller's register contents. Branches to the GTF routines, if active or the Trace routine to record information about the SVC interruption. Determines from the SVC table the type of SVC routine to be given control. If a type-1 routine, gives control to the rou-

tine. 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 AB) 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 theDispatcher. 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 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: In a Model 65 Multiprocessing System, 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 BN,BO) 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 Model 65 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: (Charts EA,EE) 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,EH) 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 KC) 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 KD) Determines if an SVC routine that occupied a transient area block but was overlaid should be reinstated. If so, schedules reloading of and entry to the routine.

Transient Area XCTL routine: (Chart CA) Prepares for entry to another module of a multi-module (type-4) SVC routine. Reinitializes 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: (Charts EC, EG) 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 KA) 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 fullword boundary alignment, determines if the 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: (Chart BG) 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: (Charts GB,GC) 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: (Charts FB,FC) Prepares buffers and posts the communications task ECB.

Write-to-Programmer routine: For a WTO or WTOR macro instruction with a ROUTCDE=11 parameter, puts the message into the job's system message class output data set.

WRITELOG Available Log Data Set routine: (Figure 7-10) 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 7-10) 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 7-10) 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: (Chart GA) 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 7-10) Passes control to the Log Writer routine when the system log ECB is posted.

WRITELOG Open Device routine: (Figure 7-10) Opens the specified system output-writer data set.

WTOR Purge routine (also called Reply Purge routine): Is invoked by the EOT routine, ASIR5, 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.

APPENDIX A: DIAGNOSTIC AIDS IN ABEND PROCESSING

If an error occurs during ABEND processing, the result is usually an invalid recursion that leads to an entry into DAR. The dump taken by DAR at this time is invaluable in determining the cause of the recursion; however, a stand-alone or ABEND dump can also be helpful. Generally, the programmer only needs to examine the following areas to get some idea of the problem:

1. PSW and registers at ABEND. (These may be in one or more places depending on the type of dump.)
2. The RB chain of the terminating TCB.
3. A trace table with a sufficient number of entries.

Most often, the important registers are found in DAR1's (IEAQTM0L) SVRB register save area. The interruption handlers' register save areas, of which the program interruption save area is the most important, sometimes can give an idea of previous happenings. (The easiest way to find the save area is to examine the machine code in the dump until the STM instruction is located for the particular interruption handler.)

The registers in ABTERM's save area are helpful in understanding an ABEND initiated by a type-1 SVC because there is often useful information left in that SVC's registers when it branches to ABTERM.

Because of the nature of ABEND's purges, invalid recursions often involve the GETMAIN or FREEMAIN functions, usually invoked via SVC 10 or its branch entry. In such a case, ABTERM's save area often contains the following useful information:

Register 2 - The invalid main storage control block detected.

Register 5 - The subpool number.

Register 8 - GETMAIN/FREEMAIN base.

Register 10 - The number of bytes to be freed or requested.

Register 11 - The area to be freed or requested.

Register 12 - The valid SPQE or DQE address that the main storage control block in register 2 was chained to. If this is zero, the main storage queues may have been exhausted in attempting to find an allocation for the referenced storage.

INTERNAL ABEND DEBUGGING FEATURES

The ABEND (SVC 13) modules contain internal features which are present for the express purpose of debugging ABEND. These features are consistent across ABEND and ASIR. Although consistency is not present across all DAR modules, ABEND/DAR interfaces have been altered to ensure that some debugging information is still available when ABEND regains control from DAR. The general debugging aids follow:

1. When an ABEND (SVC 13) module passes control, via an XCTL, to another SVC 13 module, the last four bytes of the CSECT name of the module issuing the XCTL are placed in register 0. Register 1 contains the last four bytes of the CSECT name of the module that gains control after the XCTL. A glance at the trace table tells a system programmer exactly which SVC 13 modules have received control and the order. Note that all SVC 13 modules end in C'*01C' where * is any digit or number of the universal alphabet.
2. At entry to ABEND0 for a true ABEND, the extended save area of the ABEND's SVRB is zeroed out. Any nonzero value in the ESA must have been put there by ABEND.
3. The last two words of the ESA contain:

   C'ABEND',C'*',X'SSS0'

   where

   * is the fifth character in the name of the ABEND CSECT that last got control, and

   SSS is the system completion code for the ABEND that is being processed by the ABEND SVRB. If SSS is 000, either the ABEND was a user ABEND, or the system completion code was not available at the time the first ABEND module gained control.

EXAMPLE: If the ABEND SVRB pointed to by the TCB in an ABEND dump contained the hexadecimal characters 'C1C2C5D5C4F90C10' in its last two words, the ABEND module in control at the time of the dump was IGC0901C, and it was processing a 0C1 program check.

If there were another ABEND SVRB on the chain with a value of 'C1C2C5D5C4F48060' in its ESA, there was an ABEND previous to the 0C1 program check. In analyzing this dump, the programmer may conclude that there must have been an ABEND recursion; the previous ABEND was an 806 ABEND, and the last module to have control in the previous ABEND was IGC0401C.

The system programmer might use this debugging information to determine the following chain of logic in limiting the likely problem area:

1. The last ABEND module in control was IGC0401C. It either terminated abnormally on a program check, or some routine called on its behalf terminated on a program check.

2. The ABEND dump was taken on the recursion; thus ABEND was prepared to handle the recursion validly.

3. IGC0401C has only one valid recursion - across the Write-to-Programmer routine. If WTP should terminate abnormally, a valid recursion configuration flag is set.

4. The programmer may conclude that the Write-to-Programmer routine must itself have abnormally terminated via a program check, or some system routine called on its behalf abnormally terminated. The system programmer would then check other information (such as the RB chain) to verify this conclusion, and continue by checking for the error in WTP.

IBM System/360 Operating System
MVT Supervisor

Order No. GY28-6659-6

READER'S COMMENT FORM

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance or additional publications will delay response, however. *For more direct handling of such requests, please contact your IBM representative or the IBM Branch Office serving your locality.*

How did you use this publication?

- ☐ As an introduction
- ☐ As a reference manual
- ☐ As a text (student)
- ☐ As a text (instructor)
- ☐ For another purpose (explain)_____

_____

Please comment on the general usefulness of the book; suggest additions, deletions, and clarifications; list specific errors and omissions (give page numbers):

What is your occupation?_____

Number of latest Technical Newsletter (if any) concerning this publication: _____

Please include your name and address in the space below if you wish a reply.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

Cut or Fold Along Line

Your comments, please . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Fold Fold

First Class
Permit 40
Armonk
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department 636
Neighborhood Road
Kingston, New York 12401

Fold Fold

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

Cut or Fold Along Line

MVT Supervisor Printed in U.S.A. GY28-6659-6

MVT Supervisor Printed in U.S.A. GY28-6659-6

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]