File No. S360-36
GY28-6612-4

Program Logic

IBM System/360 Operating System

Fixed-Task Supervisor

Program Number 360S-CI-505

This publication describes the internal logic of the Primary Control Program (PCP) Supervisor. The PCP Supervisor is one part of the IBM System/360 Operating System control program. It performs task management as follows:

- Interruption Supervision
- Task Supervision
- Main Storage Supervision
- Contents Supervision
- Program Fetch
- Overlay Supervision
- Time Supervision
- System Environment Recording
- Checkpoint/Restart

It is intended for persons involved in program maintenance, or system programmers who are altering the program design; it is not needed for normal use or operation of the program described.

Fifth Edition (November 1968)

This edition corresponds to Release 17 of the Operating System. It is a major revision of, and obsoletes, Y28-6612-3 and Technical Newsletter Y27-7172.

The following chapters should be reviewed completely:

Chapter 3 - Main Storage Hierarchy Support for IBM 2361 Models 1 and 2 has been added,

Chapter 8 - The chapter has been rewritten and includes SER improvements,

Chapter 9 and Appendix G - New pages describing Checkpoint/Restart

Other changes to the text are indicated by a vertical line to the left of the change; changed or added illustrations are denoted by the symbol (•) to the left of the caption. These changes include:

- Multivolume, Multiextent Link Library
- Blocked PROCLIB/PARMLIB
- STAE
- ENQ, DEQ, and Validity Check Routine Charts

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

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

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

A form for reader's comments appears at the back of this publication. Address any additional comments concerning the contents of this publication to IBM Corporation, Programming Publications, Department 637, Neighborhood Road, Kingston, New York 12401

This manual describes the internal logic of the Primary Control Program (PCP) Supervisor which is part of the IBM System/360 Operating System control program. The PCP Supervisor performs task management in operating systems using the Primary Control Program. The external characteristics of this supervisor are described in the IBM Systems Reference Library.

Information in this document is directed to the customer engineer who maintains and services the IBM System/360 Computing System and who is responsible for field maintenance and updating of the IBM System/360 Operating System. This information may also be used by the programming systems maintenance programmer and the development programmer who will expand the system.

This publication may be used to locate those areas of the system to be analyzed or modified. The information is presented to enable the reader to quickly relate the task management functions to the program listings for those functions. The comments in the listings provide information for thorough analysis and understanding of the functions.

PREREQUISITE PUBLICATIONS

Knowledge of the information in the following publications is required for a full understanding of this manual.

- IBM System/360 Principles of Operation, Form A22-6821
- IBM System/360 Operating System: Concepts and Facilities, Form C28-6535
- IBM System/360 Operating System: Introduction to Control Program Logic, Program Logic Manual, Form Y28-6605
- IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646
- IBM System/360 Operating System: Supervisor and Data Management Macro Instructions, Form C28-6647

The following publications are not required but may be useful for reference.

- IBM System/360 Operating System: TESTRAN, Form C28-6648
- IBM System/360 Operating System: Linkage Editor, Form C28-6538
- IBM System/360 Operating System: System Programmer's Guide, Form C28-6550
- IBM System/360 Operating System: System Generation, Form C28-6554
- IBM System/360 Operating System: Initial Program Loader and Nucleus Initialization Program, Form Y28-6661.

FIGURES

CHARTS

The fixed-task supervisor is a group of service routines that control the use of the central processing unit (CPU) and main storage of IBM System/360. This supervision, called task management in the IBM System Reference Library, includes supervising the interfaces between processing programs and the primary control program. The primary control program is made up of the service routines for task management, data management, and job management. The fixed-task supervisor provides the following task management functions:

- Overlap of central processing unit operations with input/output channel activity.
- Servicing of all hardware interruptions.
- Handling of all supervisor calls (SVCs).
- Allocation of main storage for programs and data.
- Dynamic loading of programs not in main storage.
- Synchronous overlay supervision.
- Use of the hardware timer (optional).
- Recording of machine malfunctions.
- Servicing requests for writing CHECKPOINT records during the execution of a program, and restarting programs at these CHECKPOINTS.

The fixed-task supervisor is part of the primary control program, which is used to process batch jobs sequentially. The primary control program requires a main storage capacity of at least 32,768 bytes, and a minimum machine configuration that includes direct-access auxiliary storage.

MAIN STORAGE AREAS

In the primary control program (PCP) environment, main storage is divided into two areas: the fixed or system area and the dynamic or processing program area. The fixed area is used for system routines that perform control functions during the execution of a processing program. The dynamic area is used for a processing program and its data, control blocks, and tables.

The fixed area is divided into the nucleus and two transient areas. The nucleus contains the more frequently used SVC routines, the interruption handlers, and other routines and control information. The transient areas are two buffers into which less frequently used system routines are brought from the system residence volume. The first, called the SVC transient area, is 1024 bytes long and is used for SVC routines. The second, called the I/O supervisor transient area, is 1024 bytes long and is used for the input/output supervisor's error handling routines.

DYNAMIC AREA USAGE

A processing program is loaded into the lower section of the dynamic area. Routines that the processing program has brought into main storage with a LOAD macro instruction are placed in the upper section of the dynamic area, the section with the numerically-greater main storage addresses. These routines, which may be system or user routines, remain in main storage for the duration of the job-step that loaded them, unless they are removed by using the DELETE macro instruction.

When the processing program issues a LINK macro instruction, the fixed-task supervisor loads the requested routine into main storage following the processing program. If this routine LINKs to another routine, the second routine follows the first in main storage. When one of these routines issues a RETURN macro instruction, control returns to the program or routine that issued the LINK. For example, if routine A LINKs to routine B, routine B finishes and returns to A, and routine A then LINKs to routine C, the fixed-task supervisor overlays routine B with routine C.

When a routine issues an XCTL macro instruction, the main storage area occupied by the routine is freed (if the routine was not originally brought into main storage with a LOAD macro instruction). If the requested routine had not been loaded into main storage previously, it is brought into the lower section of the dynamic area.

Main Storage may be expanded by including IBM 2361 Core Storage in the system. Main Storage Hierarchy Support for IBM 2361 Models 1 and 2 permits selective access to either the processor storage portion (hierarchy 0) or IBM 2361 Core Storage portion (hierarchy 1) of main storage. A hierarchy parameter (HIARCHY=) in the LINK, LOAD, XCTL, ATTACH, GETMAIN, GETPOOL, and DCB macro instructions permits specification of either hierarchy as desired. If IBM 2361 Core Storage is not included in the system, requests for storage within hierarchy 1 are obtained from processor storage.

TASK CONTROL BLOCK (TCB)

Processing programs that operate in a fixed-task environment do so as part of a task, a unit of work for the central processing unit (CPU). In PCP there is one task control block (TCB). It is used to record information about the user's program. The TCB is initialized by the Nucleus Initialization Program (NIP) and is used sequentially by each task performed within the dynamic area. (NIP is described in Appendix B.)

The TCB is 172 bytes long, with an additional 32 bytes preceding the first byte (when required) as a floating point register save area. The format and contents of the TCB are given in the publication IBM System/360 Operating System: System Control Blocks, Form C28-6628.

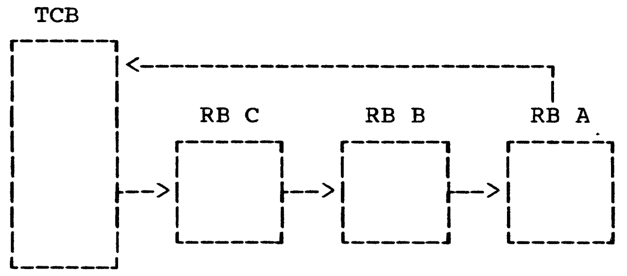
REQUEST BLOCK (RB)

There may be any number of programs (logically distinct sections of code) ready to be executed. Control passes from one program to another through a branch, LINK, XCTL, ATTACH, or as the result of an interruption for which an asynchronous exit has been specified. Every transfer of control other than a direct branch is handled by the supervisor.

Handling such transfers requires the maintenance of information allowing the supervisor to return control through the same sequence of programs but in reverse order. For example, if A links to B and B links to C, the supervisor must have the necessary information to return control to B when C completes operation and then to A when B completes operation. The request block contains this information.

Request blocks are chained together to indicate how control should be transferred. Each request block (RB) addresses the RB of the program which will receive control when the program governed by the first RB completes operation. The last element in the chain is the RB for the first program executed under the task control block (TCB). This RB addresses the TCB instead of another RB.

In the preceding example, the RB for program C addresses the RB for program B. This RB addresses the RB for program A, which points to the TCB. The TCB itself addresses the RB most recently added to the queue, in this case the RB for program C. See Figure 1.



• Figure 1. Transferring Control Using Request Blocks

Normally, one request block precedes the processing program and each requested routine. Request blocks are queued from the task control block (TCB). Request blocks for active routines are queued on the active request block queue; those for loaded routines are queued on the loaded program list.

The first request block (RB) is placed on the active request block queue by NIP. An RB for job management is substituted for this first RB when NIP transfers control, via XCTL, to job management.

In addition to addressing another RB or the TCB, each RB contains the identification of the requested program, the entry point, the interrupted program status word (resume PSW), the size of the request block, the size of the program, and the request block type.

There are six types of request blocks:

- Program Request Block (PRB) -- used to control programs not previously loaded.
- Interruption Request Block (IRB)--used to control system or user asynchronous exit routines.
- System Interruption Request Block (SIRB) -- used to control I/O supervisor error routines. Only one SIRB can exist at a given time.
- Supervisor Request Block (SVRB) -- used to control type 2 (resident), type 3 (non-resident, unimodular), and type 4 (non-resident, multimodular) SVC routines. Types 2, 3, and 4 SVCs may be enabled.

- Loaded Program Request Block (LPRB) -- used to control programs that are LOADed and are ATTACHed, LINKed, or XCTLed; also used to control sections of programs that are specified by the IDENTIFY macro instruction and are ATTACHed.

- Loaded Request Block (LRB) -- shortened form of LPRB, used to control load modules that have the "only-loadable" attribute. (It is invalid to ATTACH, LINK, or XCTL to these load modules.)

The standard formats for all request blocks and a description of their contents are given in the publication IBM System/360 Operating System: System Control Blocks, Form C28-6628.

REQUEST BLOCK QUEUEING

The TCB addresses two request block (RB) queues: the active request block queue and the loaded program list (See Figure 2).

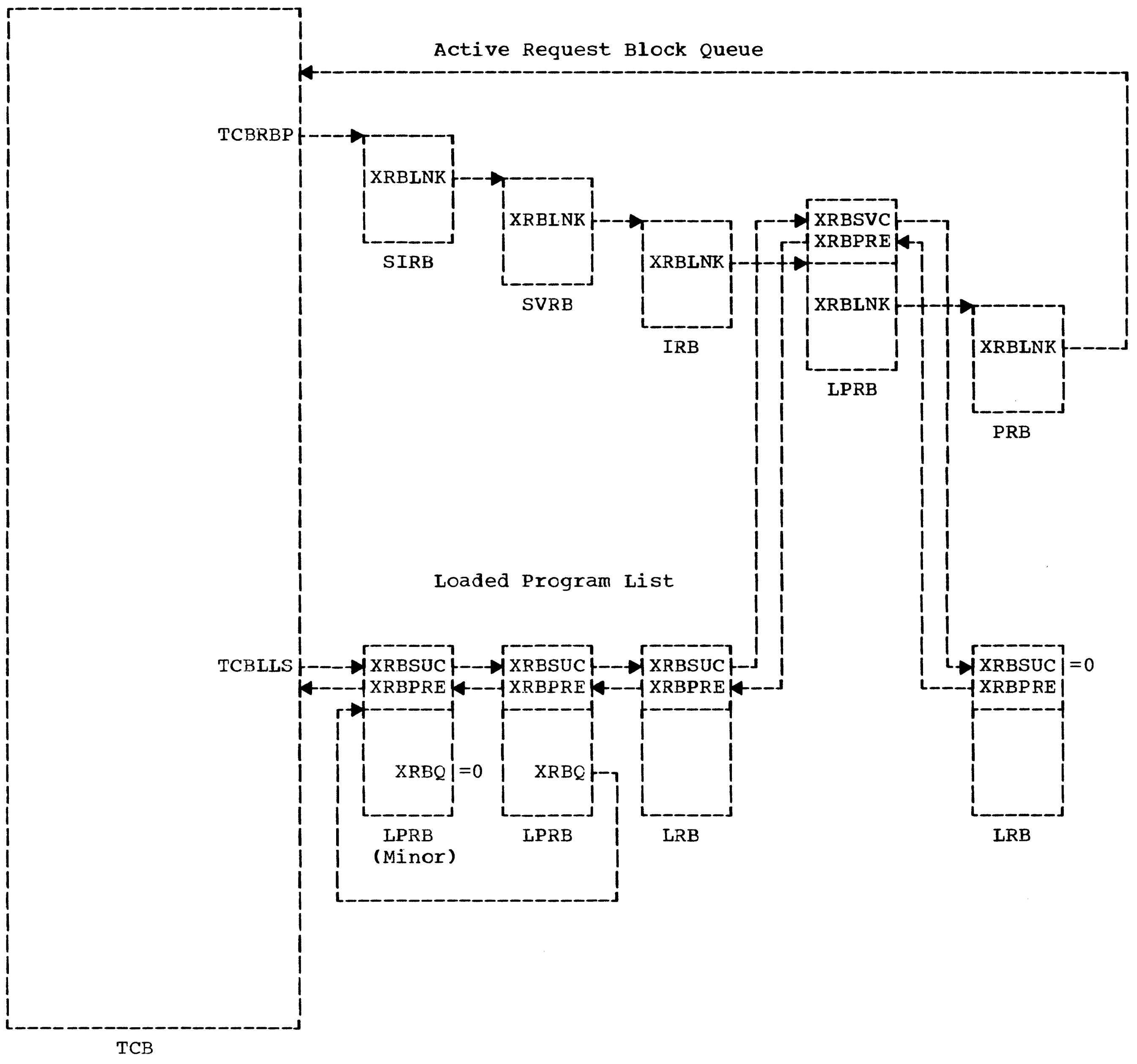


Figure 2. Request Block Queues

Active Request Block Queue

The active request block queue is made up of PRBs, IRBs, SVRBs, LPRBs, and the SIRB. There is one request block (RB) for each program to be executed. TCB field TCBRBP addresses the first (current) RB on the queue. Field XRBLNK of each RB on the queue addresses the next RB on the queue. XRBLNK of the last RB on the queue addresses the TCB.

Loaded Program List

The loaded program list contains LRBs and LPRBs in a two-way chain. Each loaded program is represented in this list. The TCB, through the pointer named TCBLLS, points to the first RB on the loaded program list. The RBs on the list are chained through the XRBSUC and XRBPRE fields. XRBPRE for the first RB in the queue points to the TCB. XRBSUC for the last RB on the list contains zero.

An LPRB may also appear on the active request block queue. If it does, it is maintained on both queues simultaneously by two different sets of pointers.

FIXED-TASK SUPERVISOR COMPONENTS

The fixed-task supervisor is composed of the following major components, each of which is a functional grouping of supervisor service routines or subroutines: interruption supervision, task supervision, main storage supervision, contents supervision, program fetch, overlay supervision, time supervision, system environment recording, and checkpoint/restart.

INTERRUPTION SUPERVISION

The interruption supervision service routines handle all interruptions on a first or introductory level. To do this they:

- Save information about the environment (machine status) at the time of the interruption so that the environment may be recreated later.
- Determine what action needs to be taken and set up the routines needed.
- Route control to the needed routines.
- Return to the interrupted environment.

TASK SUPERVISION

The task supervision service routines maintain control information. They maintain the current status of program and interruption request blocks, task control blocks, and event control blocks. The task supervision service routines are responsible for modifying and terminating task operations.

MAIN STORAGE SUPERVISION

The main storage supervision service routines establish the availability of main storage and dynamically allocate that storage to a task on request, within the dynamic area.

CONTENTS SUPERVISION

The contents supervision service routines maintain a record of the identity of all programs and routines together with their status and characteristics, within the dynamic area. The contents supervision service routines initiate program fetch for the dynamic loading of programs, and maintain the active RB queue to represent requests for the use of programs.

PROGRAM FETCH

The program fetch service routine is a relocating loader which brings a program module processed by the linkage editor from secondary storage into main storage.

OVERLAY SUPERVISION

The overlay supervision service routines monitor the flow of control between segments of a program operating in an overlay structure preestablished by the user through linkage editor. These routines ensure that all dependent program segments are brought into main storage by program fetch before the actual branch is executed.

TIME SUPERVISION

The time supervision service routines set and maintain a clock, and honor requests for time intervals and time-of-day.

SYSTEM ENVIRONMENT RECORDING

The system environment recording service routines are optional control program routines that record and in some cases attempt to minimize the effects of machine malfunctions in System/360 Models 40, 50, 65, and 75.

CHECKPOINT/RESTART

The CHECKPOINT service routine writes a copy of the requesting task's main storage area and environment. The RESTART service routine uses this copy to re-create, at a later time, the conditions which existed when the CHECKPOINT copy was written. The RESTART service routine then gives the task control. Checkpoint/Restart information is applicable only to PCP.

FIXED-TASK SUPERVISOR CONTROL FLOW

As shown in Chart 01, flow in the fixed-task supervisor is flow of interruption supervision, with alternate supplementary flow paths through other fixed-task supervisor components and other control program service routines -- those of data management, input/output supervision, job management, linkage editing, and test translation.

All interruptions in the central processing unit, in the channels, or in the devices attached to the channels, that affect control program processing, are placed before the interruption supervision service routines along with information identifying the cause of the interruption. These interruption handlers pass control to those parts of the control program that service interruptions.

When the interruption has been serviced, the interruption supervision service routines again receive control and return the central processing unit (CPU) to the state in which it was operating before the interruption occurred.

CHAPTER 1: INTERRUPTION SUPERVISION

Interruption supervision provides first level interruption handling: that is, control passes from the processing program to the control program and back again. Interruption supervision service routines:

- Save the interrupted environment.
- Insulate interruption routines from each other.
- Pass control to routines required to service the interruption.
- Return control to the interrupted program when servicing is completed.

In addition, interruption supervision provides, through the SVC handlers, all interface operations associated with the four types of supervisor call routines:

- Type 1 SVC routines. These are always resident and are executed disabled. They usually return control to the interrupted program without entering the dispatcher. A type 1 SVC routine can use the services of another type 1 SVC routine through a direct branch. It cannot use the services of any other type routine because it cannot issue SVC instructions (i.e. it cannot cause interruptions). Examples are GETMAIN, FREEMAIN, EXCP, WAIT, and EXIT.
- Type 2 SVC routines. These are also resident; but they are partially enabled, or they call on other than type 1 SVC routines. These routines are completely reenterable. Examples are LINK, LOAD, and XCTL.
- Type 3 SVC routines. These are like type 2 routines except that they are not resident. They are each brought into the 1024-byte SVC transient area. Examples are IDENTIFY, WTO, and LOCATE.
- Type 4 SVC routines. These are "multiphase" type 3 routines. That is, they are too large to be brought into the transient area at one time and must be brought in in phases, each later phase overlaying an earlier one. Transfer of control from one phase to another is through XCTL. Examples are OPEN, CLOSE, and EOV.

  Note: Type 3 and 4 SVC routines can be made resident. See "Resident Type 3 and 4 SVC Routine Option."

To achieve a high response time for input/output interruptions, interruption supervision has a software-implemented disabling subroutine called the pseudo disable routine. This routine allows input/output interruptions to be processed without the requesting routine losing control -- the routine which was interrupted regains control as soon as the input/output supervisor has processed the interruption. Requesting routines include those system routines, such as the job management write-to-operator routine, that must operate enabled yet not lose control to another routine.

INTERRUPTION SUPERVISION ROUTINES

Interruption supervision includes the following service routines:

- SVC FLIH - The supervisor call first level interruption handler does the introductory work following an SVC interruption, and prepares for the execution of type 1 SVCs.
- SVC SLIH - The supervisor call second level interruption handler monitors the SVC transient area and prepares for the execution of types 2, 3, and 4 SVCs.
- Type 1 Exit - This routine is the exiting procedure for type 1 SVCs.
- EXIT - This SVC routine is the exiting procedure for types 2, 3, and 4 SVCs.
- Dispatcher - This routine passes control from routine to routine, whether system routine or processing program routine. Through two subroutines, the dispatcher sets up the mechanism to handle asynchronous exits and monitors the I/O supervisor transient area.
- I/O FLIH - The input/output first level interruption handler does the introductory work following an input/output interruption and the clean-up work after the input/output supervisor finishes second level handling.
- T/E FLIH - The timer/external first level interruption handler does the introductory work following any timer/external interruption and the clean-up work after the second level handling is completed.

- P FLIH - The program first level interruption handler monitors all program interruptions.

- PROLOG - This routine is used by P FLIH to set up input parameters to the ABTERM service routine of task supervision.

- MC FLIH - A machine check interruption causes control to be given to a system environment recording routine if one of these is included in the system. Otherwise, the system is placed in the wait state.

- Validity Check - This routine is used as a common subroutine by other system routines, such as program fetch. The validity check routine prevents program interruptions caused by invalid addresses (those pointing beyond the boundaries of main storage) passed to the control program by a processing program. In installations that have selected the hardware protection option, this routine also checks for a mismatch between the storage key of the addressed block and the protection key of the TCB.

SVC CONTROL INFORMATION

The supervisor maintains SVC control information in the SVC table and the relocation table. These tables are in a module called IEASVC00, which is assembled during system generation.

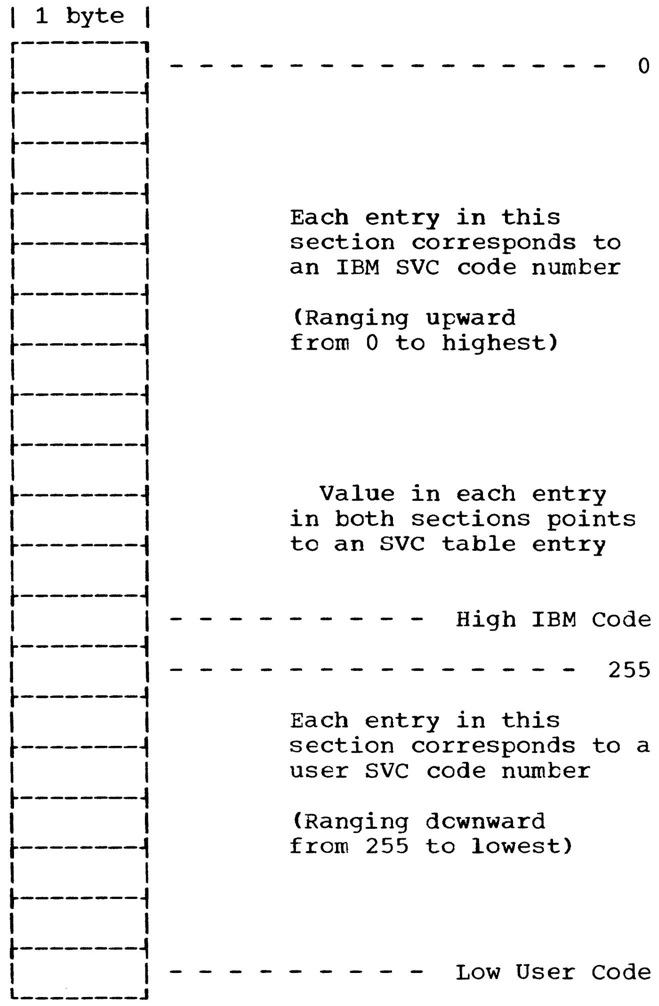
RELOCATION TABLE

The relocation table is used to relate the SVC code number with its corresponding entry in the SVC table. The relocation table consists of a number of 1 byte entries (each of which is addressed through indexing based on the SVC code numbers). Each entry contains a number. If it is zero, then the associated SVC code is invalid. If it is non-zero, then the number is an index to an entry in the SVC table.

The relocation table is divided into two sections. The first section contains entries for IBM codes (that is, codes assigned to IBM-provided SVC routines). There is one entry for each code number from 0 to (but not including) "High IBM Code", whether or not the SVC code is in use in the system.

The second section contains entries for user codes, with one entry for each code number from 255 to (but not including) "Low User Code", whether or not the SVC code is in use in the system.

The size of the relocation table is variable; its maximum size is 256 bytes. Both the size and the contents of the table are determined during system generation (based on the SVC routines included in the system). The relocation table format is shown in Figure 3.



1 byte
0
Each entry in this section corresponds to an IBM SVC code number
(Ranging upward from 0 to highest)
Value in each entry in both sections points to an SVC table entry
High IBM Code
255
Each entry in this section corresponds to a user SVC code number
(Ranging downward from 255 to lowest)
Low User Code

Figure 3. Relocation Table

SVC TABLE

The SVC table is divided into two sections. The first section contains a 3-byte entry for each type 1 or type 2 SVC routine. The second section contains a 1-byte entry for each type 3 or type 4 SVC routine.

Each 3-byte entry contains a 24-bit main storage address with the three low-order bits defined as zero. This address is the address of an SVC routine. The three low-order bits of this address are used to indicate the number of double-words required for the extended save area (ESA) in the request block (RB). Each 1-byte entry contains the extended save area information in the last three bits. If the three bits are zeros, a type 1 SVC is indicated. The SVC table is shown in Figure 4.

```
Bits:
    |-----------21------------|-3-|
    ┌─────────────────────────┬───┐
    │         Address         │ESA│
    ├─────────────────────────┼───┤
    │                         │   │
    ├─────────────────────────┼───┤
    │                         │   │
    └─────────────────────────┴───┘
```

3-byte entries for type 1 and 2 SVC routines

```
Bits:
    |--5--|-3-|
    ┌─────┬───┐
    │  0  │ESA│
    ├─────┼───┤
    │     │   │
    ├─────┼───┤
    │     │   │
    └─────┴───┘
```

1-byte entries for type 3 and 4 SVC routines

Figure 4. SVC Table

Extended SVC Table (Optional)

The SVC table may be extended during system generation so that each entry is four bytes long. The entry for a type 1 or 2 SVC routine contains a high order byte of zeros and a 24-bit address which includes the ESA information. Each entry for a type 3 or 4 SVC routine contains the track address (TT) of the transient SVC routine in the first field, the record number (R) on the track in the second field, the length of the first text record in the third field, and the size of the extended save area in the last field. The extended SVC table is shown in Figure 5.

Note: This option must be selected if the resident type 3 and 4 SVC routine option is chosen.

```
Bits:
    |----8----|-----------21-----------|-3-|
    ┌─────────┬────────────────────────┬───┐
    │   0 0   │        Address         │ESA│
    ├─────────┼────────────────────────┼───┤
    │         │                        │   │
    ├─────────┼────────────────────────┼───┤
    │         │                        │   │
    └─────────┴────────────────────────┴───┘
```

4-byte entries for type 1 and 2 SVC routines

```
Bits:
    |----10-----|-----8----|----11-----|-3-|
    ┌───────────┬──────────┬───────────┬───┐
    │    T T    │    R     │  Length   │ESA│
    ├───────────┼──────────┼───────────┼───┤
    │           │          │           │   │
    ├───────────┼──────────┼───────────┼───┤
    │           │          │           │   │
    └───────────┴──────────┴───────────┴───┘
```

4-byte entries for type 3 and 4 SVC routines

Figure 5. Extended SVC Table (Optional)

INTERRUPTION SUPERVISION CONTROL FLOW

Interruption supervision control flow, shown in Chart 02, starts with an interruption. The five types of interruptions are SVC, input/output, timer/external, program, and machine check.

SVC INTERRUPTIONS

When an SVC interruption occurs, there are two paths to the requested SVC routine. These paths are described under SVC entry procedures. When the SVC routine completes, there are two paths of return. These paths are described under SVC exiting procedures. The dispatcher is discussed after the entry and exiting procedures (to show the flow back to the processing program).

SVC Entry Procedures

Entry to SVC routines is handled by the SVC FLIH and the SVC SLIH. The execution of any SVC instruction causes the hardware to give control to the SVC FLIH by loading a new program status word (PSW) that is disabled for all maskable interruptions except machine check. The SVC instruction contains an 8-bit code which the SVC FLIH checks to determine which service routine is required.

All registers are stored in the SVC save area. The SVC code is compared to the largest valid IBM-provided value plus one. If the code is equal to or larger than the maximum, the code is analyzed to determine whether the request is for a user-provided SVC routine. The task is abnormally terminated if the SVC code is not valid. If the code is a valid IBM code, but is not supported in this system, the SVC instruction is treated as a no-operation (NOP).

Next, the SVC FLIH determines whether the requested SVC routine is listed in a resident SVC table. If listed, the address of the SVC routine is used to enter the routine.

When the request is for other than a type 1 SVC routine, the FLIH branches to the SVC SLIH after moving the original register contents to the TCB. The SLIH creates SVRBs for types 2, 3, and 4 SVC routines. If the routine is a type 2 SVC, the SLIH passes control to the routine directly. If the routine is a type 3 or type 4, then the SLIH passes control only after the routine has been placed in the transient area via the FINCH routine (described in Chapter 4).

The SVC SLIH first separates type 2 requests from types 3 and 4 so that the SLIH's SVRB creation and initialization subroutine can be executed immediately. For type 3 and 4 requests, the SVC SLIH initializes and, if necessary, fetches the required routines.

The SVRB creation and initialization subroutine stores the requestor's PSW in the current request block and then creates an SVRB for the called routine. The size of the SVRB is determined from the three low-order bits of the SVC Table entry for the called routine. (This entry has been placed in register 6 by the SVC FLIH.) The three low-order bits of the entry contain a value between 1 and 7. This value minus one is equal to the number of double words required for the request block extended save area.

After determining the size of the SVRB, the SVRB creation and initialization subroutine clears the three low-order bits of register 6 and issues a GETMAIN for the SVRB. The subroutine then initializes the SVRB and queues it on the active RB queue.

If the SVC routine is a type 2, registers 0, 1, and 15 are restored from the save area of the SVRB, environmental registers are loaded, and the type 2 SVC routine is entered.

If the SVC is a type 3 or 4, the SLIH examines the SVC table, extracts information telling the size of the extended save area needed in the SVRB, and creates and initializes the SVRB.

If the current transient area occupant is not the requested routine, the requested routine must be loaded by FINCH, which is entered by a BALR. When the loading is completed, FINCH returns control to the SVC SLIH.

The separate phases of type 4 SVC routines bring each successive phase into the transient area by using XCTL until the phases are completed. The final phase issues an SVC EXIT instruction.

SVC Exiting Procedures

There are two exiting procedures for SVC routines -- Type 1 Exit and EXIT. Type 1 SVC routines (with the exception of EXIT) return to the Type 1 Exit Routine for handling. Type 1 Exit passes control to the dispatcher or to the interrupted program -- either a processing program or a service routine. Types 2, 3, and 4 SVC routines return to the EXIT Routine. EXIT dequeues the SVRB from the TCB's active RB queue and passes control to the dispatcher.

TYPE 1 EXIT: Type 1 SVC routines branch to the type 1 exit routine when they complete processing. The type 1 SVC indicator is reset to zero, and registers are reloaded from the type 1 register save area of the SVC FLIH. The first word of the TCB pointer (IEATCBP) is compared to zero. If IEATCBP does not equal zero, it means a task switch has not been indicated, and the requestor of the exiting type 1 SVC is reentered by loading the SVC old PSW. If IEATCBP equals zero, a task switch is indicated and the SVC old PSW is checked to determine if the requestor was disabled for any interruptions. If it was disabled, the requestor retains control and is reentered by loading the SVC old PSW. If the requestor was fully enabled, registers are saved in the task control block, the SVC old PSW is saved in the current RB on the active request block queue, and the type 1 exit routine branches to the dispatcher.

EXIT: Types 2, 3, and 4 SVC routines, as well as asynchronous exit routines and routines entered by supervisor-assisted linkages, complete by using the EXIT routine directly or indirectly. Using EXIT directly means issuing an SVC EXIT instruction. Using EXIT indirectly means issuing a branch instruction with register 14 as an operand (or issuing a RETURN macro instruction which expands to include a branch on register 14), where register 14 is preset by the supervisor to point to an SVC EXIT instruction in the nucleus.

EXIT determines the type of routine that is exiting, performs the necessary terminal procedures for the routine, and prepares for return to the routine in control prior to the exiting routine. In addition, EXIT determines if the routine to receive control is an SVC routine executed in the transient area. It is possible that the sequence of events has caused the transient area to be overlayed since the SVC routine last had control. In this case, the transient area refresh subroutine of EXIT is entered to restore the SVC routine to the transient area.

EXIT passes control to either the dispatcher, a processing program, an asynchronous exit routine, or the task termination routine. The first and most common place is the dispatcher. The second, a processing program, is given control when the exit is from a program interruption routine. The third, an asynchronous exit routine, is given control when the exiting routine is an asynchronous exit routine and there are additional requests for the routine (RQEs) queued on the IRB under which it is operating. The fourth, the task termination routine, is given control when the returning program is the highest control level for a task.

When entered, EXIT resets the type 1 switch because, although EXIT is entered as a type 1 SVC routine, it does not return through the normal type 1 exit. This is because it is a transitional routine which passes control from one program to another.

After setting the type 1 switch, EXIT determines if the exiting routine created any STAE control blocks (SCBs) that were not cancelled. If the XCTL-option was not specified for these uncancelled SCBs, EXIT updates the SCB pointer and frees the main storage occupied by these SCBs.

EXIT next determines if the exiting routine is a program interruption routine. If it is, the address of a program interruption element (PIE) is loaded from TCB field TCBPIE. The PIE contains the PSW and the contents of registers 14 through 2 that were in effect when the program interruption occurred, unless they were modified by the user's program interruption routine. The right half of the PSW saved in the PIE is moved to the SVC old PSW, registers 14 through 2 are loaded from the PIE register save area, and the SVC old PSW is loaded to return control to the processing program. Unless the user's program interruption routine modified the values in the PIE or in registers 3 through 13, the processing program regains control at the instruction following that which caused the program interruption.

If the exiting routine is not a program interruption routine, EXIT:

1. Saves registers 10 through 1 in the register save area of the TCB,
2. Obtains the address of the RB for the exiting routine from TCB field TCBRBP,
3. Obtains the address of the RB for the routine next to receive control from field XRBLNK of the exiting program's RB.

EXIT determines if the exiting RB is an IRB or the single SIRB in the system. (Both IRBs and the SIRB are discussed under Dispatcher and Exit Effector.) If it is either, EXIT determines if the RB has:

- Interruption queue elements (IQEs) with 4-byte link fields.
- IQEs with 2-byte link fields.
- No IQEs.

If the RB has interruption queue elements, the IQE at the top of the RB's XRBQ queue is removed. If the IQE has a 2-byte link field, the IQE is returned to the I/O supervisor to be placed on its list of available queue elements. (In the I/O supervisor program logic manual, IQEs with 2-byte link fields are called request elements.) Interruption queue elements with 4-byte link fields are not queued on any other queue and are effectively discarded when they are removed from the XRBQ unless the NEXAVL field of the IRB exists, in which case they are returned to this queue.

The RB is checked for more queue elements. If there are more, and if the new top IQE has a 2-byte link field, the address of the top IQE is loaded into registers 1 and 0. If the top queue element has a 4-byte link field, register 0 contains the address of the IQE, as before, but register 1 contains the data from the second 4-byte field of the queue element. In either case, the return address to be used by the asynchronous exit routine is loaded in register 14, and the entry point address of the asynchronous exit routine from the XRBEP field of the RB is loaded into register 15 before the routine is entered. The first word of the RB, potentially the register save area address, is loaded into register 13.

If there are no other IQEs queued on the RB, the saved registers are moved from the RB's register save area to the TCB's register save area. The exiting RB is dequeued from the task's active request block queue, and the routine to receive control is checked to see if it is in a wait state. If it is, the first word of the TCB pointer is set to zero, indicating that a task switch is necessary. If the RB is not waiting, the status bits in the RB for the routine to regain control are checked to see if the routine is a type 3 or 4 SVC. If it is, the name field in the request block (XRBNM) is compared to the name of the routine in the transient area. If the routine is not in the transient area, the transient area refresh subroutine is entered to bring it in. EXIT branches to the dispatcher.

<u>Dispatcher</u>

Loading a PSW to pass control to a routine associated with a request block is called dispatching. The dispatcher receives control through a branch from EXIT, type 1 exit, I/O FLIH, or T/E FLIH. The dispatcher gives control either to the routine last in control or to a different routine, or places the machine in a wait state.

After receiving control, the dispatcher first determines if there are any asynchronous exit routines to be scheduled. If there are, the dispatcher enters Part 3 of the exit effector to schedule these routines. Then it examines the first word of the TCB pointer, IFATCBP, and dispatches the task whose TCB is addressed. In systems with the timer option (see Chapter 7), the dispatcher dequeues the timer element for a task time request before entering the wait state, and enqueues it again when leaving the wait state.

When dispatching a task, the dispatcher places the address of the task in both words of the TCB pointer, restores the registers, and loads the resume PSW. If the task is not ready, the dispatcher places the computer in a wait state by turning on a bit in the resume PSW before loading it.

The dispatcher has a very important subroutine called the <u>exit effector</u>. The exit effector schedules the input/output supervisor's error routines using the I/O supervisor transient area and schedules requests to enter asynchronous exit routines by:

- Initializing an IRB or the SIRB.
- Placing the IRB or the SIRB on the active RB queue.
- Manipulating the saved registers to allow the dispatching of the asynchronous exit routine.

<u>EXIT EFFECTOR</u>: The exit effector consists of three parts. The first two parts are used by routines that require asynchronous exits. The third part is a dequeueing routine used by the dispatcher.

<u>Part One</u>: The first part of the exit effector is the CIRB service routine. This routine creates and initializes an IRB and, if specified, acquires additional storage within the dynamic area for a register save area and a work area used for building interruption queue elements (IQEs). The address of the register save area is located in the three low-order bytes of the first word of the IRB. The format of the IRB is shown in Figure 6.

96 bytes (required)
NEXAVL=*+4 (optional)
Work area for building IQEs (optional)

Figure 6. IRB Format Options

<u>Part Two</u>: The second part of the exit effector is used by a calling routine to schedule an asynchronous exit routine. Part two queues the IQE provided in register 1 as input, in FIFO order on either the 2-byte AEQ (asynchronous exit queue) or the 4-byte AEQ.

<u>Part Three</u>: The third, dequeueing part of the exit effector is entered by the dispatcher when the dispatcher finds that the AEQ points to an IQE. (Each time it is entered, the dispatcher checks for entries on the AEQ.) Part three dequeues the IQE from the AEQ, finds the IRB and TCB associated with the IQE, queues the IQE on the IRB and the IRB on the TCB's active RB queue. When two or more IQEs refer to the same IRB, they are queued in first-in/first-out (FIFO) order.

Part three ensures that no IRB is scheduled for a task which has the SIRB on its active RB queue. The interruption queue element remains on the asynchronous exit queue to defer scheduling of the current IRB until the SIRB is inactive.

<u>ENTRY TO ASYNCHRONOUS EXIT ROUTINES</u>: The name of the error routine to receive control is generated using information in the UCB pointed to from the second half-word of the IQE. If the requested routine is in the I/O supervisor transient area, the routine is dispatched. Otherwise, FINCH (described in Chapter 4) brings the routine into the I/O supervisor transient area and ensures that the return address, entry point, and IQE address are in the registers and that the current error routine entry point is addressed by the SIRB.

EXITING FROM ASYNCHRONOUS EXIT ROUTINES: When the asynchronous exit routine for the first IQE is completed, EXIT is entered. The IQE is then dequeued from the IRB and is either returned to the I/O supervisor or queued on the NEXAVL field that immediately follows the IRB, or discarded.

If there are no additional IQEs queued on the IRB when an asynchronous exit routine returns, EXIT dequeues the IRB from the active RB queue. If there are additional IQEs queued on the IRB, the necessary initialization steps are executed and the IRB routine is reentered directly.

If the IRB and a work area were obtained by using part one of the exit effector, the work area is freed when the IRB is freed. If the IRB is to be reused, it is dequeued but is not freed.

Resident Type 3 and 4 SVC Routine Option

During system generation, the user can select the resident type 3 and 4 SVC routine option. Frequently used routines can be made resident so they need not be brought into the transient area each time they are required. A resident type 3 or 4 routine assumes the characteristics of a type 2 routine except when it issues an XCTL macro instruction (see Chart 08).

The following differences in operation result when the user chooses the resident option (and the optional extension of the SVC table).

1. When the nucleus initialization program (Appendix B) makes each type 3 or 4 routine resident, the routine's entry in the SVC table is changed. The track address, record number and length fields are overlayed by X'FF' and the entry point address of the routine. Each time a type 3 or 4 SVC routine is requested, the SVC table is checked. X'FF' (a number larger than any track address) indicates that the entry corresponds to a resident type 3 or 4 SVC routine. The format of each entry for a resident type 3 SVC routine or for the first module of a resident type 4 routine is:

```
Bits:
   |----8----|------------21----------|-3-|
   ┌─────────┬────────────────────────┬───┐
   | X'FF'   |  Entry Point Address   |ESA|
   └─────────┴────────────────────────┴───┘
```

2. The SVC entry procedure for a resident type 3 or 4 routine is similar to that for a type 2 routine. A resident type 3 or 4 SVC routine does not require the services of FINCH because, like a type 2, the routine need not be loaded into the transient area.

3. The SVC exiting procedure does not require the services of the transient area refresh subroutine if a resident type 3 or 4 routine receives control since a resident routine does not operate in the transient area and could not have been overlayed since it last had control. The transient area refresh subroutine examines the SVRB of the SVC routine receiving control. The SVRB indicates that the routine is a type 3 or 4. If the entry point in the SVRB does not correspond to the SVC transient area entry point, a resident type 3 or 4 SVC routine is receiving control. If the entry point is that of the transient area, a non-resident routine is being requested and the transient area must be checked to ensure that the routine has not been overlayed since it was last used.

4. The XCTL service routine checks the RSVC load list created by the nucleus initialization program (Appendix B) to determine if the SVC routine is resident or if it requires loading.

INPUT/OUTPUT INTERRUPTIONS

Certain events, such as errors or completed actions in an input/output device or in the channel to which it is attached, cause the number of the device and a word of detailed information (about the status of the channel and the nature of the event) to be placed in storage. The I/O FLIH is not concerned with the channel scheduler or with the details of input/output handling. It performs machine interruption supervision and insulates the input/output interruption from other types of interruptions. The I/O FLIH is given control by the input/output new PSW. The I/O FLIH is entered:

- Disabled for all maskable interruptions other than machine check.
- In supervisor state.

The first instruction of the I/O FLIH is a NOP/branch switch, set to a branch by the first input/output interruption, allowing input/output interruptions to be processed in groups. The first interruption of a group causes the I/O FLIH to execute some initialization instructions which block any further execution of this "first-time logic" for successive interruptions in a group. Registers two through nine are saved.

If the system is not pseudo disabled, the input/output old PSW is saved in the current RB. The wait bit in the input/output old PSW is set to zero (non-wait state), and registers ten through one are saved in the TCB's general register save area.

If the system is pseudo disabled, registers 10 through 1 are saved in the interruption supervision pseudo disable save area, and the input/output old PSW is saved.

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

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

TIMER/EXTERNAL INTERRUPTIONS

Timer/external interruptions may come from the optional hardware timer at location 80, from the interrupt key on the console, and from six external units. The T/E FLIH in the fixed-task supervisor handles two kinds of timer/external interruptions:

1. those caused by the optional hardware timer,

2. those caused by the interrupt key on the console.

The T/E FLIH passes control to time supervision for second level handling of timer interruptions and to job management's external interruption routine for second level handling of interrupt key interruptions.

When an interruption occurs, the hardware stores the current PSW in the timer/external old PSW location, indicates the cause of the interruption in the interruption code field in the T/E old PSW, and loads the new PSW from the timer/external new PSW location. This gives control to the T/E FLIH.

The T/E FLIH saves registers 10 through 1 in the TCB, stores the timer/external old PSW in a standard original old PSW location (see program listing), and examines the interruption code in the timer/external old PSW to determine the interruption type.

When a supported interruption type is identified, the T/E FLIH branches to the appropriate second level handler. When the interruption has been serviced, control returns to the FLIH. Two supported interruptions may have occurred simultaneously. In this case, the FLIH handles the second interruption in the same way as the first. After handling supported timer/external interruptions, the FLIH branches to the dispatcher.

If non-supported timer/external interruptions occur, the T/E FLIH returns control immediately to the interrupted routine rather than to the dispatcher.

PROGRAM INTERRUPTIONS

If the program being executed attempts an improper action, a program interruption occurs and a code describing the attempt is stored in the program old PSW. Improper events causing program interruptions include:

1. addressing non-existent operation codes, and

2. attempting to execute privileged instructions.

Users may specify fixed point overflow, decimal overflow, exponent underflow and significance as additional improper events requiring special handling.

If the user wishes to handle some or all program interruptions, he first issues a SPIE macro instruction which generates a program interruption element (PIE) and inserts its address in the TCB. The program first level interruption handler (P FLIH) is given control by the hardware after any program interruption. The P FLIH checks the TCB for an address of a PIE. If no PIE address is present in the TCB, the interruption is unanticipated, and the P FLIH passes control to the PROLOG routine to initiate abnormal termination of the task.

If a PIE address is present in the TCB, the PIE is examined and the address of a program interruption control area (PICA) is extracted. The P FLIH tests the user's program interruption mask in the PICA to see if the user is handling the type of program interruption that has occurred. The type that has occurred is shown in the interruption code in the program interruption old PSW. If the user is handling the interruption, the P FLIH saves the old PSW and registers 14 through 2 in the PIE.

Register 14 is loaded with a return address, register 1 with the address of the PIE, and register 15 with the address of the user's routine. The P FLIH places the address of the user's interruption routine, obtained from the PICA, into the old PSW, restores the work registers from the save area, and loads the modified old PSW to return to the user's program at the entry point of his program interruption handler.

The user may return to the main body of his program from his program interruption handling routine either by a direct branch or by issuing a RETURN macro instruction. If the user returns to the main body of his program by a direct branch, he must reset the first-time-entry switch in the PIE.

If the program interruption type is not handled by the user, PROLOG is entered by a branch. This routine sets up the abnormal termination linkage and branches to ABTERM.

MACHINE CHECK INTERRUPTIONS

If the error detection equipment finds a machine error, information representing the internal state of the machine is placed in the diagnostic scan-out area of main storage. A machine check can cause control to be passed to a system environment recording routine if one of these is included in the system environment. Otherwise, the system is placed in the wait state.

The task supervision service routines maintain control information, cause tasks to be executed, and perform other task-related services. Task supervision service routines:

- Maintain task control blocks.
- Enter tasks into the wait state.
- Post completed events in the event control block.
- Maintain control levels indicated by request blocks.

TASK SUPERVISION ROUTINES

The task supervision service routines are functionally divided into two areas in the fixed-task supervisor: task modification and task termination.

TASK MODIFICATION

ATTACH: When an ATTACH macro instruction is issued, the supervisor gives control to the ATTACH service routine. The ATTACH service routine passes control to the routine requested in the ATTACH macro instruction and regains control when the requested routine completes. ATTACH optionally posts an event control block to mark the completion, and, also optionally, passes control to a user-specified exit routine. If no special exit is specified, ATTACH returns control to the attaching routine.

EXTRACT, SPIE, STAE: Through the EXTRACT, SPIE, and STAE service routines, task supervision allows the user to make better use of the system's controls. EXTRACT provides a processing program with information contained in specified fields of the task control block. SPIE allows the user to specify the address of an exit routine to be entered when specified program interruptions occur. The SPIE routine sets the program mask in the PSW as specified when a SPIE macro instruction is given. STAE allows the user to specify the address of an exit routine to be entered when an ABEND is scheduled for the task.

WAIT, POST: Through the WAIT and POST service routines, task supervision monitors the movement of the task between the ready and wait states. WAIT prevents the task from continuing until an event specified in the WAIT macro instruction parameters has taken place and has been indicated by the execution of a POST macro instruction. As an option, a WAIT routine to service multiple event completions may be chosen by the user. POST signals that the event represented by a specified event control block has occurred. This may result in the task being moved from a wait state to a ready state.

ENQ, DEQ: For the shared direct access device (shared DASD) feature only, task supervision serializes the use of shared data through the ENQ and DEQ service routines, by indicating to the I/O supervisor when it should reserve and release shared direct access devices. If two CPUs share a direct access device, then each CPU must issue a RESERVE macro instruction (processed by the ENQ service routine) before using the device, and a DEQ macro instruction when finished using the device. This prevents both CPUs from attempting to access the same device simultaneously.

TASK TERMINATION

ABTERM, ABEND, ABDUMP: A task may be terminated by itself or by the system. Task supervision completes a task's execution through ABTERM and ABEND service routines. The ABTERM service routine schedules the ABEND routine, which terminates the task. The ABDUMP service routine is used when a full storage dump is required.

TASK SUPERVISION CONTROL FLOW

As shown in Chart 03, flow of task supervision is the flow of the individual modular service routines. Each receives control from interruption supervision and returns control to its particular exiting procedure. The one exception is the ABTERM routine, which is branched to by any service routine, and returns to that routine by a branch.

ATTACH

The ATTACH service routine searches for the RB of the requested routine in the loaded program list. If the requested routine is not in the dynamic area, ATTACH uses FINCH to bring it in. ATTACH places a request block on the active RB queue for the attached routine. Control is given to the attached routine by loading a PSW with an LPSW. The active request block queue is ordered as follows:

- RB for the attached routine.
- SVRB for the ATTACH routine.
- RB for the attaching routine.

When the attached routine completes, the ATTACH routine is dispatched and optionally posts the event control block. If the attaching routine specified an exit routine in the ETXR parameter of the ATTACH macro instruction, ATTACH places a request block on the active RB queue for the exit routine. When the ATTACH routine completes, the exit routine is dispatched, if this option was specified. When the exit routine completes, the attaching routine is dispatched.

EXTRACT

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

SPIE

The SPIE service routine is used to set up indications that the user has requested program interruption control. SPIE is entered by the SVC SLIH when a SPIE macro instruction is given. Thirty-two bytes of main storage space for a program interruption element (PIE) is obtained, and the address of the PIE is saved in the TCB. In creating the PIE (Figure 7), the SPIE routine places in the first four bytes the address of the program interruption control area (PICA) specified by the processing program in the SPIE macro instruction. The SPIE routine sets aside the second eight bytes as a program interruption old PSW save area, and the next 20 bytes as a 5-register save area.

A program mask whose contents is determined by the interruptions selected is stored into the caller's resume PSW. SPIE executes an SVC EXIT instruction on completion.

```
 0 ┌──────────────────────────────────────┐
   │                                      │
   │        User's PICA Address           │
 4 ├──────────────────────────────────────┤
   │                                      │
   │                                      │
   │        Old PSW Save Area             │
   │                                      │
   │                                      │
12 ├──────────────────────────────────────┤
   │                                      │
   ┴                                      ┴
   ┬        Register Save Area            ┬
   │                                      │
32 └──────────────────────────────────────┘
```

Figure 7. Program Interruption Element (PIE) Format

STAE

The STAE service routine provides an exit routine address at which control will be returned to the user if an ABEND is scheduled for his task. STAE is entered by the SVC Second Level Interruption Handler (SLIH) when a STAE macro instruction is issued. The STAE service routine creates a 16-byte STAE control block (SCB) as shown in Figure 8.

STAE also places the address of the SCB in TCB field TCBNSTAE. If the task enters abnormal termination processing, the TCBNSTAE field is tested by the ABEND routine to determine whether STAE processing (the ABEND/STAE interface routine) should be invoked for the task.

```
 0 ┌──────────────────────────────────────┐
   │               Zero or                │
   │       Address of Previous SCB        │
 4 ├──────────────────────────────────────┤
   │           Address of STAE            │
   │             Exit Routine             │
 8 ├──────────────────────────────────────┤
   │    Address of STAE Exit Routine      │
   │            Parameter List            │
12 ├──────────────────────────────────────┤
   │           Address of User's          │
   │             Request Block            │
16 └──────────────────────────────────────┘
```

• Figure 8. STAE Control Block (SCB) Format

WAIT -- SINGLE EVENT

When WAIT is entered by the SVC interruption handler, the wait count passed as a parameter of the WAIT macro instruction is tested for zero. If it is zero, the routine returns immediately by branching to the type 1 SVC exit. If it is non-zero, then the resume PSW of the caller is enabled for input/output and external interruptions. The wait and complete bits are tested in the ECB whose address was passed by the macro instruction. When the complete bit is on, indicating that this event is already completed, WAIT branches to the type 1 exit. If the wait bit is on, indicating this event is already being waited for, WAIT terminates the task by branching to ABTERM. (Checking the wait bit is performed only if the validity check option is selected during system generation.) If neither bit is on, the wait bit is turned on and the address of the current RB is placed in the ECB. A wait count of one is placed in the current RB, and the first word of the TCB pointer, IEATCBP, is zeroed as a signal to the dispatcher that the task is waiting. WAIT returns by branching to the type 1 exit in interruption supervision.

WAIT -- MULTIPLE EVENT

The WAIT service routine is entered by the SVC FLIH as a result of a WAIT macro instruction. Upon entry to the WAIT routine, the wait count passed as a parameter is tested for zero. If it is zero, the routine returns immediately by branching to the type 1 SVC exit. If the wait count is non-zero, the resume PSW of the caller is enabled for input/output and external interruptions. The wait count is saved and a loop initialized to address the ECBs addressed by the macro instruction parameter list. An ECB counter is incremented as each ECB is addressed.

As in single-event WAIT, on an optional basis, the wait bit in the first ECB is tested. If it is on, indicating that this ECB is already being waited on, the next ECB is addressed. If the wait bit is off, the completion bit is tested. If the completion bit is off, indicating that a POST has not yet occurred, the wait bit is turned on and the address of the current RB is placed in the ECB. If this event has already completed -- if the completion bit is on -- the wait count is decremented and tested for zero. If the count is not zero, a test is made to see if this address is the last element in the parameter ECB list. If it is not the last element, the cycle is repeated. If it is the last element, the loop is exited. If the wait count becomes zero, all the wait bits in the ECBs are turned off and the WAIT routine exits to the type 1 exit, without putting the current RB into a wait state since its count has already been satisfied.

When all ECBs have been addressed and the wait count has not become zero, the total number of ECBs specified is compared to the original wait count. If the number of ECBs specified is less than the count, the count cannot be satisfied; the task is abnormally terminated by scheduling ABEND through a branch to ABTERM.

If the wait count is less than the number of ECBs, a bit is turned on in the RB to indicate to POST that a multiple-event WAIT has been issued where the number of ECBs is greater than the wait count. If the wait count is less than or equal to the number of ECBs, WAIT inserts the wait count into the current RB and sets the first word of the TCB pointer to zero as a signal that the task is waiting. The WAIT service routine returns by branching to the type 1 exit routine of interruption supervision.

POST

The POST service routine is entered by the SVC FLIH after a POST macro instruction is issued, but an alternate entry is provided so that system routines can branch directly to POST. Upon entry, POST tests the completion bit of the ECB whose address was passed as an input parameter. If it is on, indicating that the ECB has already been posted, the POST routine returns by branching to the type 1 exit or to the system routine which entered POST.

If the completion bit is off, the wait bit is tested to see if this event is being waited on. If the bit is off, the completion code is placed in the ECB and the completion bit is turned on. If the wait bit is on, the RB wait count is decremented, the completion code is placed in the ECB, the completion bit is turned on, and the wait bit is turned off. POST returns by branching either to type 1 exit or to the system routine which branched to POST.

In systems with a multiple event WAIT, POST performs further operations. When the wait count in the RB is decremented to zero, POST tests a bit in the waiting RB to see if the number of ECBs specified in the associated WAIT was greater than the wait count specified.

If the number of ECBs was greater, then POST turns off the wait bits in all ECBs in the ECB list specified which have not yet been posted, to indicate that no one is waiting for these events to be completed and to prevent an erroneous POST. The address of the ECB list is located in a register save area belonging to an RB or to the TCB. POST finds the addresses by determining which RB is waiting. If RB 3 in the following diagram is waiting, the address of the ECB list is in the register 1 field of the TCB register save area. If RB 2 is waiting, the list address is in the same field of the register save area of RB

3. If RB 1 is waiting, the address is in the register save area of RB2.

```
 TCB
┌──────┐
│      │<──────────────────────────┐
│      │                           │
│      │     RB 3       RB 2       RB 1
│      │   ┌──────┐   ┌──────┐   ┌──────┐
│      │   │      │   │      │   │      │
│      ├──>│      ├──>│      ├──>│      │
│      │   │      │   │      │   │      │
│      │   └──────┘   └──────┘   └──────┘
└──────┘
```

If the number of events waited on equals the number of events specified, the wait bits are turned off by POST as the events complete. After turning off the wait bits, POST places the completion code in the ECB, and returns.

ENQ

The ENQ service routine is entered from the SVC SLIH after a RESERVE or ENQ macro instruction (SVC 56) is issued. ENQ routine control flow is shown in Chart 04.

On entry, register 1 contains the address of a parameter list which the ENQ routine scans to determine if the routine was entered because of a RESERVE or a normal ENQ. In PCP, a normal ENQ is treated as no-operation (NOP) because enqueueing operations are not required; control returns to the calling routine.

RESERVE is used in PCP to permit direct access storage devices to be shared. When the ENQ service routine processes a RESERVE macro instruction, it gives the requesting task exclusive control of a specified device via a hardware reserve. The task frees the device with a DEQ macro instruction, resulting in a hardware release. If the parameter list indicates that RESERVE is requested, the ENQ routine tests the parameter list for validity (see Chart 06). The list must be in the format shown in Figure 9.

If the parameter list is not in the correct format, the ENQ routine issues an SVC 13 to abnormally terminate the task with a system error code of 438. If all elements in the parameter list are valid, the ENQ routine examines the major queue control block (QCB) queue via the major QCB origin in the CVT. (See Appendix E for major and minor QCB formats.)

If there are no major QCBs (QCB pointer in the CVT is zero) or if the major resource name in the parameter list does not match the major resource name of any major QCB on the queue the routine examines the RET parameter of the ENQ macro instruction. If RET=TEST is specified (indicating an inquiry), the return code in the parameter list is set to zero to indicate that the requested resource is available (has not been previously reserved); control returns to the calling routine. If RET=TEST is not specified, ENQ creates a major

```
 0 ┌─────────┬─────────┬─────────┬─────────┐
   │         │ Rname   │         │ Return  │
   │   FF    │ length  │  Codes  │  code   │
 4 ├─────────┼─────────┴─────────┴─────────┤
   │         │     Address of              │
   │    0    │     Qname                   │
 8 ├─────────┼─────────────────────────────┤
   │         │     Address of              │
   │    0    │     Rname                   │
12 ├─────────┼─────────────────────────────┤
   │         │     Address of              │
   │    0    │     UCB Pointer             │
16 └─────────┴─────────────────────────────┘
```

ENQ Parameter List

Byte 0	indicates the end of the ENQ parameter list (always a single entry list for RESERVE).
Byte 1	length of the minor resource name.
Byte 2	parameter codes: Bit 0 - not used in PCP. Bit 1 - bits 1 and 4 must be 0 and 1 respectively, to indicate the reserve function of ENQ. If bits 1 and 4 are not 0 and 1, ENQ is treated as a NOP. Bit 2 - not used in PCP. Bit 3 - not used in PCP. Bit 4 - see bit 1. Bits 5, 6, and 7 001 - indicates RET=HAVE 011 - indicates RET=USE 111 - indicates RET=TEST.
Byte 3	return code provided by the control program if RET=TEST, USE, or HAVE is specified.
Byte 4	zero.
Bytes 5-7	address of the major resource name.
Byte 8	zero.
Bytes 9-11	address of the minor resource name.
Byte 12	zero.
Bytes 13-15	address of the UCB pointer.

Figure 9. ENQ Parameter List

and minor QCB to indicate that the requestor has control of the specified resource. The routine inserts the UCB address of the specified shared direct access device into the minor QCB, and increments the ENQ count in the TCB by one to indicate that the task has reserved a resource.

In PCP, the ENQ count is only incremented for RESERVE macro instructions issued, since ENQs are NOPs. The ABEND routine examines the TCB ENQ count to determine if a purge of outstanding device reservations is necessary during task termination. The ENQ routine increments the UCB reserve count by one to indicate to the I/O supervisor that reservation of the device is required. This would prevent another CPU from gaining access to the device until the I/O supervisor determines the UCB reserve count is 0 and releases the device. Again, the routine tests the RET parameter. If RET=TEST, RET=USE, or RET= HAVE is specified, the return code in the parameter list is set to 0 to indicate to the requestor that the resource is available; control is returned to the calling routine. If RET=TEST, USE, or HAVE is not specified, the return code is not set. Control returns to the calling routine.

If upon examining the major QCB pointer in the CVT, the ENQ routine finds that a major QCB exists for the major resource name specified in the parameter list, but a minor QCB for the specified minor resource name does not exist, the routine interrogates the RET parameter. If RET=TEST is specified, the routine sets a zero return code in the parameter list to indicate that the requested resource is available and returns control to the calling routine. If RET=TEST is not specified, the routine creates a minor QCB and queues it on the major QCB to indicate that control of the requested resource has been assigned to the caller. The UCB address is placed in the minor QCB, the TCB ENQ count and UCB reserve count are adjusted, and processing continues as above.

If both a major and minor QCB exist for the requested resource, the RET= parameter is tested. If RET=TEST, USE, or HAVE is specified, the routine sets the return code to 8 to indicate that the requested resource has already been acquired by the requestor and control returns to the calling routine. If RET=TEST, USE, or HAVE is not specified, the ENQ routine issues an SVC ABEND instruction to abnormally terminate the requesting task.

DEQ

The DEQ service routine is entered from the SVC SLIH when a DEQ macro instruction (SVC 48) is issued. DEQ routine control flow is shown in Chart 05. DEQ determines whether the I/O supervisor should free a shared direct access device which was reserved by a previous ENQ.

On entry, register 1 contains the address of a parameter list. The DEQ routine performs a validity check of the parameter list (see Chart 06). If the Qname and Rname indicated by the second and third words of the list (see Figure 9) are not valid, DEQ issues an SVC ABEND instruction with a system error code of 430. If the Qname and Rname are valid, the routine examines the major QCB queue. If no major QCBs exist, or the Qname specified in the parameter list does not match the Qname of an existing major QCB, control returns to the caller. Control also returns to the caller if a major QCB exists for the specified Qname, but a minor QCB does not exist for the specified Rname.

If both major and minor QCBs exist for the specified Qname and Rname, the DEQ routine obtains from the minor QCB the address of the UCB representing the direct access device on which the resource resides, and decrements the UCB ENQ count by one. If the UCB reserve count is then zero, an EXCP macro instruction is issued to release the device. If the UCB reserve count is not zero, a release is not performed.

The DEQ routine decrements the TCB ENQ count by one to reflect the task's release of the specified resource. The routine removes the minor QCB from the minor QCB queue and issues a FREEMAIN macro instruction to release the space occupied by the minor QCB. If no more minor QCBs remain on the queue, the routine removes the major QCB from the major QCB queue and frees the space occupied by it. The DEQ routine returns control to the calling routine.

ABTERM

Certain system routines branch to the resident abnormal termination (ABTERM) service routine to schedule the abnormal termination of a task. ABTERM returns to the system routine by branching to the address passed to ABTERM in register 14.

When entered by a type 1 SVC routine, ABTERM saves the right half of the SVC old PSW and replaces the right half with the address of an SVC ABEND instruction. The task completion code is stored in the TCBCMP field provided in the TCB. After turning off the type 1 switch in the SVC FLIH, ABTERM loads registers 0 and 1 from the type 1 SVC save area, restores registers and branches on register 14 as set by the SVC routine which branched to it.

When entered by any other system routine, ABTERM locates the current RB on the RB queue of the TCB, saves the wait count from the RB, replacing it with a zero wait count, and saves the right half of the resume PSW from this RB. The task completion code is stored in the TCBCMP field in the TCB. ABTERM replaces the right half of the resume PSW in the RB with the address of an SVC ABEND instruction, restores the registers and branches on register 14 as set by the system routine which branched to it.

ABEND

The ABEND service routine is a type 4 SVC routine that is used for both normal and abnormal termination of tasks. The basic function of ABEND is to terminate all internal activities of the current task and give control via XCTL to the GO module of job management to continue processing.

Normal End

When ABEND is entered for a normal termination, it checks if all data sets have been closed. If any data sets are open, ABEND calls the data management CLOSE routines. The task completion code is stored in the TCBCMP field of the TCB, and all main storage in the dynamic area is designated as a free area. ABEND then transfers control (through an XCTL) to job management to initiate either the next step of this job or the first step of a new job.

Abnormal End

When ABEND is entered for an abnormal termination, it first determines, from TCB field TCBNSTAE and from the reason for entry, whether STAE processing should be performed for the abnormally terminating task. If STAE processing is indicated, ABEND invokes the ABEND/STAE interface routine, which will eventually return control to the user at the exit routine address specified in the STAE macro instruction.

ABEND next determines if ABTERM was entered. If it was, ABEND restores the PSW and wait count to the RB that called ABEND. If ABTERM was not entered, ABEND stores the completion code in the TCBCMP field of the TCB. ABEND purges all input/output operations initiated for this task using the HALT I/O option. It performs validity checking of the various system queues -- such as main storage supervision queues, contents supervision queues, and data management queues -- to prevent ABEND from being requested while ABEND is in progress. ABEND removes the SIRB from the active RB queue.

ABEND determines the amount of main storage it will need and acquires the storage either by using GETMAIN or by overlaying reentrant code at the beginning of the dynamic area.

ABEND determines if the abnormally terminating routine has requested a dump. If it has, ABEND searches the TIOT for a SYSABEND ddname. If this entry is not located, ABEND assembles pertinent information and packs it in main storage for eventual printing by the job management routines. This information is called an <u>indicative dump</u>. If the SYSABEND entry was located, ABEND opens a DCB and calls the ABDUMP type 4 SVC routine. ABDUMP assembles a full hex-formatted dump of the TCB, PSW, RBs, save areas, and all of main storage. If Main Storage Hierarchy Support is included in the system, ABDUMP recognizes and dumps main storage in hierarchies 0 and/or 1 associated with the terminating job step. Storage limits are obtained from the boundary box.

Upon completion of either the indicative dump or ABDUMP, or if no dump was taken, ABEND attempts to CLOSE all data sets by calling the data management CLOSE routines. As in normal termination, all main storage within the dynamic area is designated as a free area. ABEND transfers control (through an XCTL) to job management to print the indicative dump if provided and to initiate the next task.

Shared Direct Access Device ABEND

If shared DASD is included in the system, ABEND performs additional processing. This processing is the same for normal end and abnormal end. ABEND determines if the task has released all the devices it reserved. If not, ABEND releases them.

The sixth load of ABEND (module IEAATM05, SVC name IGC0501C) examines the TCB ENQ count. If the count is not zero, the reserve counts in all UCBs representing shared direct access devices are inspected. If any UCB reserve count is not zero, it is reset to zero and an EXCP macro instruction is issued to release the device. When all reserved devices have been released, ABEND resets to zero the TCB ENQ count and the major QCB queue origin in the CVT in preparation for the next job step.

Normal termination is not converted to abnormal termination even if the task did not release its reserved devices during termination.

The main storage supervision service routines establish the availability of main storage space and dynamically assign space for program loading and work areas. The main storage supervision service routines:

- Allocate main storage space dynamically.
- Release main storage space dynamically on request.
- Maintain a record of all free areas of main storage.

MAIN STORAGE SUPERVISION ROUTINES

Main storage supervision includes the GETMAIN and FREEMAIN service routines. It is resident within the nucleus, is not reenterable, and is disabled for all maskable interruptions except machine check.

The GETMAIN service routine allocates storage to a task according to its needs, when a GETMAIN macro instruction is issued.

The FREEMAIN service routine releases storage space on request, when a FREEMAIN macro instruction is issued.

MAIN STORAGE SUPERVISION CONTROL FLOW

Main storage supervision control flow is shown in Chart 07. The GETMAIN and FREEMAIN routines receive control from the SVC FLIH and pass control through type 1 exit. Register-type GETMAIN and FREEMAIN requests have a separate entry point. An exception occurs when an error condition is encountered. In this case, control passes to ABTERM through a branch.

In a PCP system, there is only one subpool, and it is unnumbered. All main storage requests are satisfied from this subpool. If subpool numbers are specified in GETMAIN and FREEMAIN macro instructions, they are ignored.

Main storage is divided into two areas, the fixed area and the dynamic area. The main storage supervisor controls only the dynamic area.

If IBM 2361 Core Storage and Main Storage Hierarchy Support for IBM 2361 Models 1 and 2 are included in the system, the dynamic area is divided into two storage areas, processor storage (hierarchy 0) and IBM 2361 Core Storage (hierarchy 1). The main storage supervisor allocates space according to hierarchy in the upper (high address) portion of a storage area for routines requested by LOAD macro instructions and for data areas requested by the user. The main storage supervisor allocates space according to hierarchy in lower (low address) portion of a storage area to the processing program and to routines called through LINK, XCTL, and ATTACH macro instructions.

If IBM 2361 Storage is not included in the system, the entire dynamic area is in processor storage. Space is allocated in the same manner as described in the preceding paragraph except that all space is allocated from processor storage.

Boundary Box

Allocation of space in the dynamic area is controlled through use of a boundary box which is addressed by TCB field TCBMSS. The boundary box is initialized by the nucleus initialization program (see Appendix B) and consists of three words.

- The first word of the boundary box contains the address of the first element of a free area queue for processor storage.
- The second word contains the address of the beginning of the dynamic area.
- The third word contains the address, plus one byte, of the end of processor storage.

If Main Storage Hierarchy Support is included in the system, the boundary box is expanded to six words (see Figure 10). The first byte of the expanded boundary box contains a "1" in bit 7 to indicate that hierarchy support is included. The first three words of the expanded boundary box are used to control the allocation of processor storage space (hierarchy 0). These three words are the same as the three-word boundary box described above.

If IBM 2361 Core Storage is not included in the system but Main Storage Hierarchy Support is included, the last three words of the expanded boundary box are set to zero. If IBM 2361 Core Storage is included in the system, the last three words (Words 4, 5, and 6) of the expanded boundary box are used to control the allocation of this storage space (hierarchy 1).

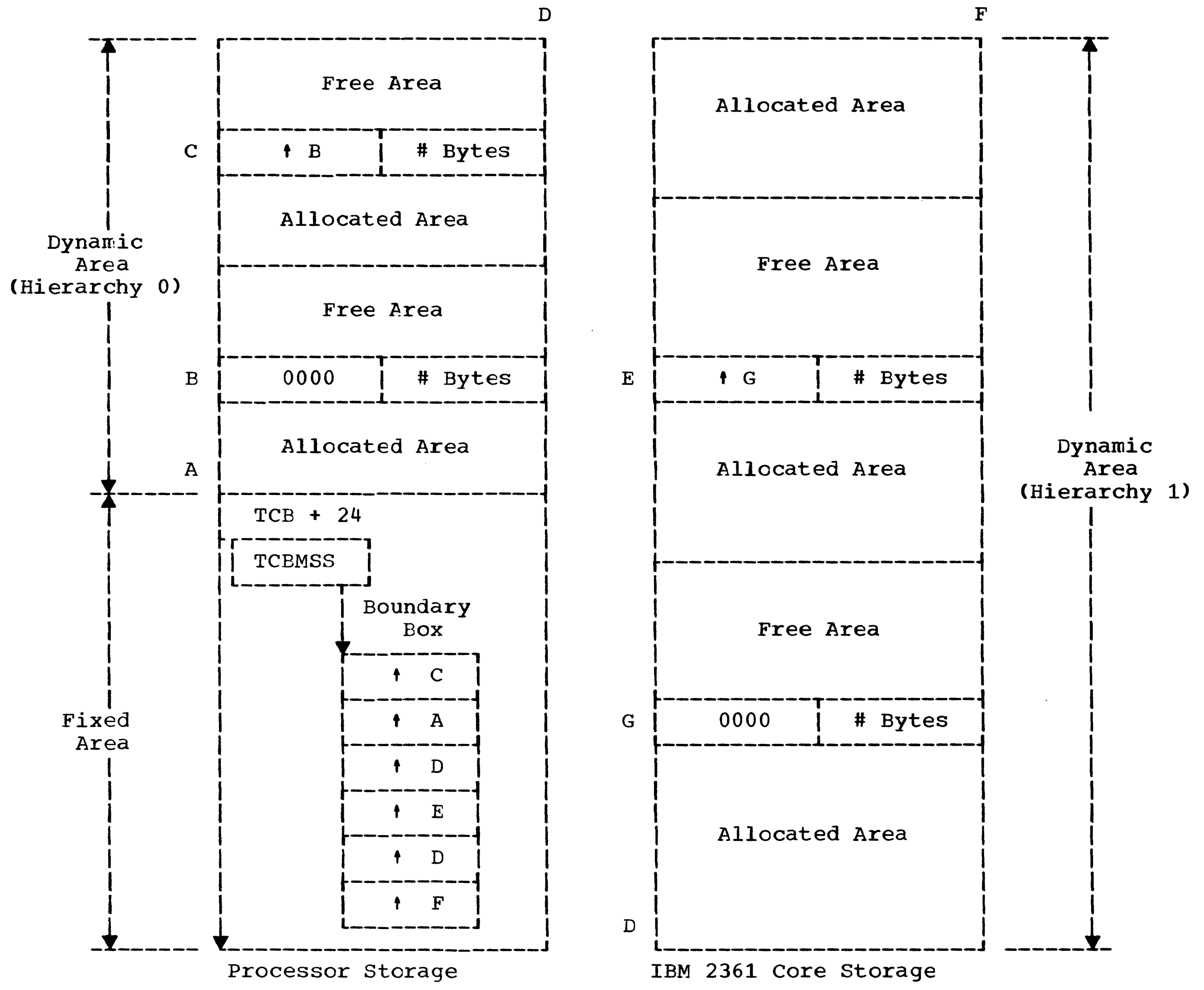


Figure 10. Main Storage Organization

- The fourth word of the boundary box contains the address of the first element of a free area queue for IBM 2361 Core Storage.

- The fifth word contains the address of the beginning of IBM 2361 Core Storage space.

- The sixth word contains the address, plus one byte, of the end of IBM 2361 Core Storage space.

Free Area Queue

A free area queue is a series of free area queue elements which are chained together. The free area queue for processor storage indicates the total amount of hierarchy 0 space not being used at a given time. The free area queue for IBM 2361 Core Storage indicates the total amount of hierarchy 1 space not being used.

Free Area Queue Element

Each free area queue element (FQE) is a double-word which represents a distinct free area. The first word contains the address of the next lower FQE on the queue. The first word of the FQE at the lowest address in a storage area (processor storage or IBM 2361 Core Storage) contains zeros. The second word contains the length of the free area (in bytes). The FQE is always in the lowest eight bytes of each free area (see Figure 10). Each FQE begins and ends on a double word boundary; requests for main storage space are always rounded up to a double word boundary.

GETMAIN

When a GETMAIN is executed, the free area queue is searched for space as large or larger than that required. If found, the space is allocated, and the amount used is subtracted from the free area from which it was removed. If space is not found and the request was conditional, GETMAIN ends by branching to type 1 exit. If the area is not found and the request was unconditional, GETMAIN branches to ABTERM to schedule the termination of the task.

FREEMAIN

When a FREEMAIN is executed, the area to be freed is checked for any overlap with existing free areas. If overlap exists, an error has occurred and FREEMAIN branches to ABTERM for the scheduling of an abnormal termination of the task. Otherwise, FREEMAIN combines the area to be freed with any adjacent free area, by updating that area's FQE. If there are nc adjacent free areas, FREEMAIN creates an FQE for the newly freed area and queues the FQE on the free area queue. On completion, FREEMAIN branches to type 1 exit.

CHAPTER 4: CONTENTS SUPERVISION

Contents supervision service routines record the identity, main storage location, size, properties and users of routines which, with the data they operate on, make up tasks. Completed routines remain in storage if they were originally brought in by a LOAD macro instruction. Contents supervision service routines maintain two lists (see the discussion of request block queueing in the introduction to this manual) of routines in the dynamic area:

- Active request block queue -- a list of active routines given control by type II, III, or IV linkage, excluding type 1 SVCs.
- Loaded program list -- a list of frequently-used routines brought into storage by a LOAD.

Each routine in these lists is represented by an RB that immediately precedes the routine in main storage. Exceptions to this are: the SIRB, which is permanently in the nucleus; SVRBs, which are always in the upper end of main storage, away from their associated routines; and "minors," which are RBs placed on the loaded program list by the optional IDENTIFY macro instruction and which represent routines embedded in the processing program.

Contents supervision maintains the two lists by chaining the RBs for the routines. Each list is addressed by the TCB.

CONTENTS SUPERVISION ROUTINES

Contents supervision is made up of the following service routines: LINK, LOAD, XCTL, IDENTIFY (optional), DELETE, SYNCH, and a common subroutine called FINCH.

LINK: This service routine passes control from the routine that issued the LINK macro instruction to another routine so that the issuer regains control when the second routine completes.

LOAD: This service routine brings a routine specified in the parameters of a LOAD macro instruction into main storage and inserts its RB on the loaded program list with a use count of one. If the routine is already on the list, the service routine merely adds one to the use count, which thus reflects the number of times a LOAD has been issued for this routine minus the number of times a DELETE has been issued for it.

XCTL: This service routine passes control from the routine issuing the XCTL macro instruction to a requested routine. When the requested routine completes, control is not returned to the issuer, which has been removed from the active RB queue, but to the routine which preceded the issuer of the XCTL. The issuer of the XCTL is removed from main storage if it was not brought into main storage by a LOAD macro instruction.

IDENTIFY: This service routine causes a routine named by the issuer of the IDENTIFY macro instruction to have a minor RB created for it, and causes this RB to be chained on the loaded program list. The RB which is the result of the IDENTIFY is on the LOAD list only for control purposes. The RBs of these identified routines are removed from the loaded program list and the RB space is released whenever these routines are deleted.

DELETE: This service routine decrements the use count in the RB of a LOADed routine named by the issuer of a DELETE macro instruction. When the use count becomes zero, DELETE removes the RB from the loaded program list and frees the storage space occupied by the routine. (Note: In systems which include the IDENTIFY macro instruction, any minors associated with the named routine are also removed by DELETE.)

SYNCH: This service routine creates, initializes, and queues program request blocks. System routines or processing programs use this routine to create PRBs for segments of code which they designate by placing an entry point address in register 15 and executing an SVC SYNCH instruction. After the PRB is queued on the active request block queue, SYNCH returns by executing an SVC EXIT instruction.

FINCH: This service routine interfaces with the data management BIDL routine, and with program fetch (described in Chapter 5) to retrieve routines from auxiliary storage. Routines may be retrieved when a LINK, LOAD, XCTL, or ATTACH macro instruction is issued, or when a non-resident SVC routine or non-resident input/output supervisor error routine is requested. After the routines are loaded into main storage, FINCH records information concerning their attributes and main storage locations into the appropriate contents supervision lists.

CONTENTS SUPERVISION CONTROL FLOW

As shown in Chart 08, the flow of contents supervision is essentially the flow of the individual service routines, which receive control from interruption supervision and pass control to their particular exit routine on completion. FINCH is an exception in that it receives control from LINK, LOAD, and XCTL, as well as from a number of other system routines including ATTACH and the SVC FLIH, and returns to whatever routine requested its services.

The routines which service LINK, LOAD, XCTL, and ATTACH macro instructions direct program loading into hierarchies of main storage if Main Storage Hierarchy Support is included in the system. These routines, upon entry from the SVC SLIH, extract the hierarchy number from the parameter list and, if a copy of the requested program must be loaded, pass the hierarchy number (0 or 1) to the FINCH service routine. The GETMAIN request issued by FINCH then allocates storage in the specified hierarchy.

LINK

The LINK service routine is entered by the SVC SLIH in response to a LINK macro instruction.

LINK searches the loaded program list for the RB of the requested routine. If it is found, and it is not already on the active RB queue, LINK prepares the RB for dispatching. If the routine is not found or if it is active, LINK enters FINCH. FINCH constructs an RB for the requested routine and places both the RB and its routine in the lower end of the dynamic area.

On return from FINCH, LINK prepares the RB for dispatching by:

- Initializing LINK's SVRB so that register loading causes the requested routine to execute EXIT when it issues the RETURN macro instruction.
- Flagging the requested routine's RB to indicate that it is active.
- Placing the requested routine's RB on the active RB queue between the RB for LINK and the RB for the issuer of the request, to ensure that the requested routine is entered when LINK issues EXIT.
- Issuing the SVC EXIT instruction.

LOAD

The LOAD service routine is entered by the SVC SLIH when a LOAD macro instruction is issued. LOAD searches the loaded program list for the RB of the requested routine, and if it finds it, increments the use count and passes the requested routine's entry point to the issuer in register 0. LOAD branches to the terminal portion of LINK that issues the SVC EXIT instruction.

If the requested routine is not found on the loaded program list, LOAD branches to FINCH to load the routine into storage. On return from FINCH, LOAD initializes the requested routine's RB and places it on the loaded program list, sets the RB's use count to one and branches to LINK to issue the SVC EXIT instruction.

If the resident access method (RAM) option was selected during system generation and the name of the requested routine is prefixed by IGG019, LOAD searches the RAM system load list first. If the RB of the routine is found there, the use count is not incremented and the entry point of the routine is passed to the user in register 0. If the RB of the routine is not found in the system load list, LOAD searches the loaded program list and proceeds as previously described.

XCTL

The XCTL service routine is entered by the SVC SLIH when an XCTL macro instruction is issued.

If the XCTL macro instruction was issued by an SVC routine operating in the SVC transient area, the XCTL service routine branches to FINCH to locate the routine on the SVC library and bring it into the transient area. XCTL branches to that part of LINK that completes the initialization of the RB and executes an SVC EXIT instruction.

If the XCTL macro instruction was not issued by a transient SVC routine, XCTL dequeues the issuer's RB and its minors from the active RB queue. The routine which issued XCTL and its RB are removed from main storage (unless the routine was LOADed).

If the requested routine is on the loaded program list and is not active, XCTL branches to LINK to:

- Set the active bit in the RB for the requested routine.

- Queue the RB on the active RB queue.

- Issue an SVC EXIT instruction.

If the RB of the requested routine was not found inactive on the loaded program list, XCTL branches to FINCH to bring the routine into main storage. On return from FINCH, XCTL initializes the routine in the same manner as if its RB had been found inactive on the loaded program list.

If the resident type 3 and 4 SVC routine option was selected during system generation and an XCTL macro instruction was issued by a type 3 or 4 routine, the XCTL routine checks the RSVC system load list to determine if the requested routine is resident or requires loading.

IDENTIFY

The IDENTIFY service routine is entered by the SVC SLIH in response to the issuance of an IDENTIFY macro instruction which is an option in the fixed-task environment.

IDENTIFY builds and initializes a minor request block to describe a routine specified in the parameters of the IDENTIFY macro instruction, and chains this minor to the loaded program list and to the RB of the routine which contains the identified routine. IDENTIFY returns to the issuer by issuing an SVC EXIT instruction.

DELETE

The DELETE service routine is entered by the SVC FLIH when a DELETE macro instruction is issued. The DELETE routine decrements the use count in the RB of the routine specified in the parameters of the DELETE macro instruction. If the use count reaches zero, DELETE dequeues the routine from the loaded program list and issues a FREEMAIN macro instruction to release the storage occupied by the specified routine and its RB. On return from FREEMAIN, DELETE repeats the deleting process for any minors belonging to the specified routine. DELETE returns by branching to the type 1 SVC exit.

If the RB of a routine is found in the resident access method (RAM) system load list, the use count is not decremented by DELETE and the FREEMAIN macro instruction is not issued.

SYNCH

The SYNCH service routine is entered by the SVC SLIH when a SYNCH macro instruction is executed. SYNCH uses GETMAIN to obtain 32 bytes of main storage from the lower end of the dynamic area for the creation of a PRB. The PSW in the PRB is initialized by SYNCH to address the location specified in register 15 by the issuer of the macro instruction. SYNCH sets the PSW completely enabled in problem program mode, with the protection key recorded in the task control block. After the PRB is created and initialized, SYNCH queues it on the active request block queue below the SVRB for SYNCH, and returns by issuing an SVC EXIT instruction.

FINCH

The FINCH service routine is a common subroutine. It is entered by a branch from seven other system routines and it returns to them by a branch. The seven service routines which branch to FINCH are:

- ATTACH
- SVC SLIH
- LINK
- EXIT EFFECTOR
- LOAD
- EXIT
- XCTL

FINCH uses the data management BLDL routine to locate a named routine on an external storage device. Using the information provided by BLDL, FINCH initializes the program fetch parameters and uses the program fetch routine to bring the specified routine into main storage. FINCH allows for necessary RBs when issuing GETMAIN, and initializes them with the RB type and the size of the storage space they and their routines occupy.

Program fetch, a part of the resident nucleus, places into main storage load modules obtained from the system library or any other library organized as a partitioned data set. Program fetch is reenterable; that is, it can be used concurrently by more than one task. The module name of program fetch is IEWFTMIN.

Program Controlled Interrupt (PCI) fetch is an optional program fetch module that can be used in place of IEWFTMIN. The module name of PCI fetch is IEWFTPCI. Either IEWFTMIN or IEWFTPCI is selected during system generation. PCI fetch improves performance on some System/360 models by requiring only one revolution of the disk to place the contents of one track into main storage. The differences between standard program fetch (Chart 09) and PCI fetch (Charts 09 and 10) are pointed out in notes throughout the chapter.

Program fetch has two entry points. Contents supervision passes control to program fetch by branching to IEWMSEPT, overlay supervision passes control to program fetch by branching to IEWFBOSV.

A load module is placed into main storage using block loading, which places an entire load module into a contiguous main storage area. IEWFTMIN and IEWFTPCI operate in block loading mode only. Standard program fetch requires one revolution of the disk for each RLD record read. Standard fetch waits for channel end so that it can begin any necessary relocation. When it has completed relocation, standard program fetch issues another EXCP to read the next RLD and/or text record.

Note: PCI fetch reads in the RLD and/or text record and then, rather than waiting for channel end to occur, it uses a PCI appendage to allow the channel program to read the next RLD and/or text record into another buffer. The PCI appendage gives control to the relocation subroutine which performs any relocation that is required on the contents of the previous buffer while the next buffer is being filled. This improved performance assumes:

- That a buffer is always available for RLD records to be read into.
- That no errors occur during I/O execution.
- That no cylinders are crossed while the program is being fetched.
- That the speed of the CPU allows the PCI appendage to change a CCW from a NOP to a TIC to the next channel program before the channel picks up that CCW.

PROGRAM FETCH FUNCTIONS

Program fetch performs the following specific functions:

- Initialization. Performs initialization procedures to prepare for the loading of a module.
- Loading. Reads text records and RLD records of a load module into main storage.
- Relocation. Adjusts values of address constants to reflect the relocation of a module that has been loaded into main storage.
- Termination. Performs termination procedures after a module has been loaded into main storage.

PROGRAM FETCH CONTROL FLOW

Program fetch receives control from contents supervision when either a LINK, ATTACH, LOAD, or XCTL macro instruction is issued and a usable copy of the module specified is not in main storage. When contents supervision requests a block module, program fetch loads the entire module. A load module with the scatter attribute is block loaded. When an overlay module is requested, only the root segment is loaded.

Program fetch receives control from overlay supervision when a segment of an overlay program specifies another segment that is not in main storage either by a branch or by issuing a SEGWT or CALL macro instruction. After receiving control from overlay supervision, program fetch loads the requested segment.

If Main Storage Hierarchy Support is included in the system, the loading of relocatable units of a program can be directed into hierarchy 0 or hierarchy 1 or into both hierarchies by the use of the linkage editor hierarchy loading attribute (HIARCHY=). The loading of overlay structure programs can be directed into either hierarchy, but load segments of the same overlay program cannot be loaded into different hierarchies. When no hierarchy is specified, the overlay structure exists in hierarchy 0.

The initialization procedures shown in Chart 09 are performed each time program fetch begins execution. Control then passes to the loading routine, which reads in the module. Relocatable address constants embedded in text records are adjusted by the relocation routine. Control passes between the loading routine and the relocation routine until the entire segment or module is loaded and relocated. Termination procedures are then performed and control is returned to the caller.

Note: PCI fetch performs relocation asynchronously with its input/output execution.

```
Byte
    ┌─────────────────────────────────────┐
   0│     CHPG1 -Channel Program          │
    │        (7 double words)             │
    │                   ┌─────────┬───────┤
  32│                   │  ECB    │  IOB  │
    │                   │(1 word) │       │
    ├───────────────────┴─────────┼───────┤
  64│ IOB -Input/Output Block     │IOBSKBUF│
    │        (8 words)            │IOB Seek│
    ├─────────┬───────────────────┼───────┤
  96│ Buffer  │SEEKBUF -Fetch Seek│       │
    │(2 words)│Buffer (3 words)   │       │
    ├─────────┴───────────────────┘ ┌─────┤
 128│ REGSAVE -Register Save Area   │     │
    │        (10 words)             │     │
    ├───────────────────────────────┘     │
 160│                                     │
    │                                     │
    │                                     │
 192│                                     │
    │                                     │
    │                                     │
 224│              RLDBUF                 │
    │                                     │
    │    Relocation Dictionary Buffer     │
 256│                                     │
    │            (64 words)               │
    │                                     │
 288│                                     │
    │                                     │
    │                                     │
 320│                                     │
    │                                     │
    │                                     │
 352│                                     │
    │                                     │
    │                                     │
 384│                                     │
    │                                     │
    └─────────────────────────────────────┘
```

Figure 11. Program Fetch Work Area

```
┌───────────┬──────────────────────────────┐
│           │Relocation factor for module  │
│           │                              │
├───────────┴──────────────┬───────────────┤
│                          │ Concatenation │
│                          │ Number        │
├──────────────────────────┴───────────────┤
│TTR0 - relative (to beginning of data     │
│set) disk address of segment 1            │
├──────────────────────────────────────────┤
│TTR0 - relative (to beginning of data     │
│set) disk address of segment 2            │
└──────────────────────────────────────────┘

│                                          │

│                                          │

┌──────────────────────────────────────────┐
│TTR0 - relative (to beginning of data     │
│set) disk address of segment N            │
└──────────────────────────────────────────┘
```

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

Figure 12. Note List (in Main Storage)

INITIALIZATION

Contents supervision supplies program fetch with the following parameters (see program listing for contents of general registers and fetch parameter list):

- Main storage address of applicable partitioned organization directory record.
- Main storage address of an opened data control block (DCB) to be used while loading the module.
- Main storage address of the work area to be used (see Figure 11).
- Main storage address of area into which NOTE list is to be read for overlay programs (see Figure 12).
- Main storage address at which to begin loading the module.
- Return address in general register 14.

Overlay supervision supplies program fetch with the following parameters:

- Main storage address of the data control block (DCB) previously used to read in the root segment.
- Main storage address of the note list (loaded before the root segment).
- Main storage address of a work area for use by program fetch.

Note: The work area for PCI fetch is within the PCI program.

- Segment number of the requested segment multiplied by 4.
- Return address in general register 14.

After receiving control, program fetch uses the parameters supplied to build an input/output block (IOB), an event control block (ECB), and a channel program (CCW list) in the specified work area. The channel program is used to read in the program, and if necessary, the note list containing the relative disk addresses of the overlay module's segments. Figure 13 shows the relationship of the blocks and tables used by program fetch to load block and overlay modules.

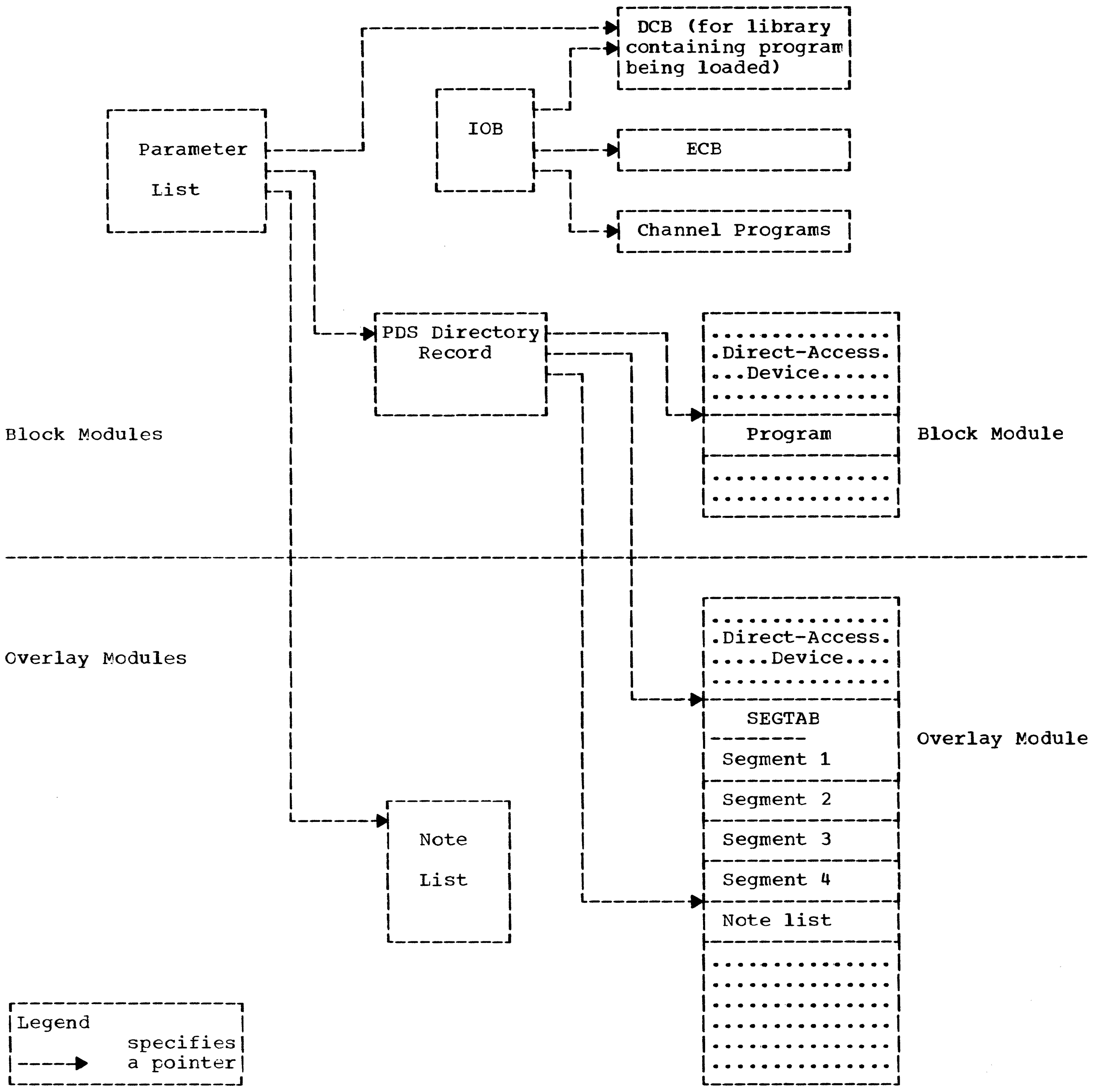


Figure 13. Blocks and Tables Used by Program Fetch

Record 1	Record 2	Record 3	Record 4	Record 5	Record 6	Record 7
Control	Text	Control	Text	RLD	Control-RLD-End-of-seg.	Text
20 bytes	500 bytes	20 bytes	1024 bytes	260 bytes	200 bytes	15 bytes

• Figure 14. Typical Load Module (Logical Format on Direct-Access Device)

Note: PCI fetch builds three channel programs in the PCI fetch work area. The work area also contains three relocation dictionary buffers.

LOADING

A load module (Figure 14) consists of:

1. Control records,
2. Text records,
3. RLD records,
4. Control and RLD records.

These records are of variable length. Their formats are shown in Appendix D.

After control is received from contents supervision, program fetch obtains the length and the relative disk address of a module's first text record from the partitioned organization directory record (see Appendix D). Subsequent text records are read using the length given in the control record preceding each text record. One or more records containing RLD information will follow a text record that has embedded relocatable address constants. Program fetch uses the RLD information to find and adjust the values of the address constants.

When loading a block or overlay module, program fetch alters the mode of its chan-

Condition	Number of Records Read With Each EXCP Issued		Source (if any) of Record Length and Relative Disk Address (TTR), if not reading sequentially
	Standard Fetch	PCI Fetch	
Normal first EXCP for all modules including root segment of overlay modules	2	reads all records connected with the load module	Partitioned Organization Directory Record
Normal Mode	2		Control record provides record length of following text record
First EXCP for a segement	1		NOTE list provides relative disk address (TTR)
EXCP for a NOTE list	1	1	Partitioned Organization Directory Record
EXCP to read a control and/or RLD record that previously caused an incorrect length input/output error	1	not applicable for PCI	None
Previous record was RLD only (did not contain control information)	1	not applicable for PCI	None
EXCP for a module that consists of one text record and no RLD records	1	1	Partitioned Organization Directory Record
Last record of the module is a text record	1	not applicable for PCI	Control record provides record length of following text record.

Figure 15. Conditions Affecting Channel Program Mode

nel program according to the type and sequence of records contained in the module (see Figure 15). The normal sequence of records in a module is: control information - text record - control information - text record. Two records are read at a time as long as the normal sequence -- a text record followed by control information -- is encountered. When the second of the two records read in the normal mode does not contain control information, program fetch alters the mode of the channel program so that a subsequent EXCP macro instruction causes a single record to be read. Each record read singly is checked for control information. If present, program fetch restores its channel program to the normal mode. Text records are read into their assigned main storage location; RLD records are read into the RLD buffer.

As program fetch loads a module, it reads the count record preceding each data record into the fetch seek buffer. The channel program's search command specifies the last count record read. This is the count record that precedes the last data record that was read. When the count record specified by the search command is found, a subsequent read count, key and data command will result in skipping the data record that followed the count record and will begin reading at the next count record, as shown in Figure 16.

Note: For PCI fetch, the search command specifies a count record and the subsequent read begins with the data that follows that count record. See Figure 16.

Program fetch causes a single record to be read by turning off the command chaining bit in the first read CCW of the channel program when either of the following conditions occur:

- The last text record of a module is to be read (indicated by the setting of the end-of-segment bit in the preceding control record).
- A module to be loaded consists of a single text record without any RLD information following it (indicated by the module's attributes in the PDS directory).

Overlay Modules

When an overlay module is loaded, its NOTE list is first read into main storage. The root segment is then read into main storage using normal block loading procedures.

While an overlay program is being executed, the NOTE list which contains the main storage address of the SEGTAB and the relative disk addresses of the module's segments, remains in main storage.

After the root segment has been loaded the SEGTAB is initialized. Program fetch inserts, into SEGTAB, the main storage address of data control block (DCB) and the NOTE list, and if required, sets the TESTRAN indicator.

To read in a segment other than the root segment, program fetch uses a relative disk address found in the NOTE list to read the first control record of the segment. The information in the control record is used to begin reading in the segment in the normal mode.

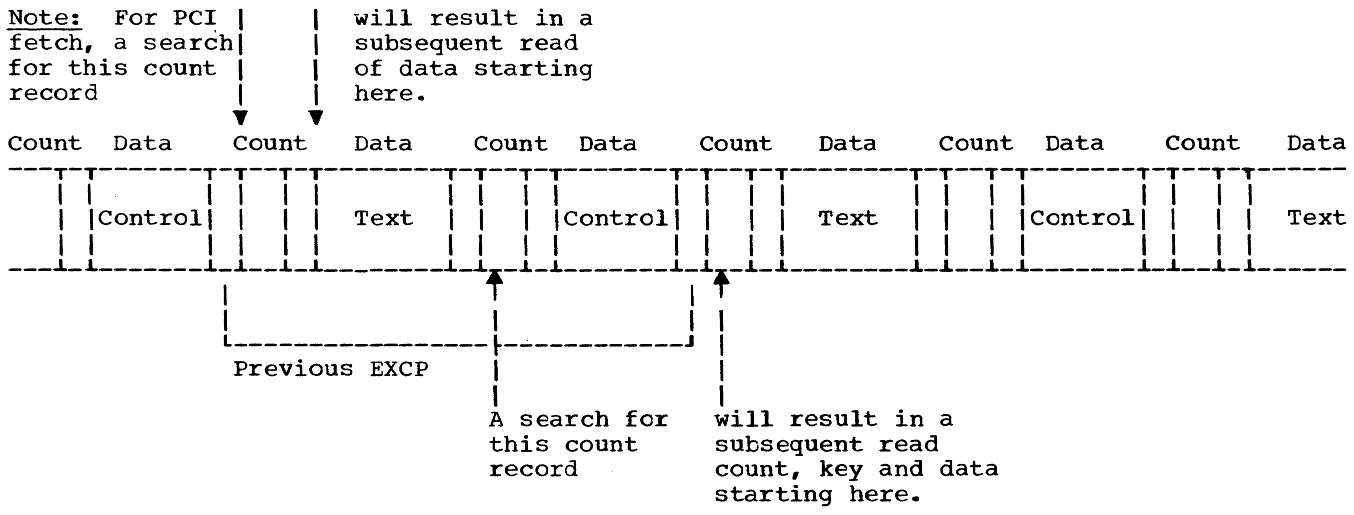

Note: For PCI fetch, a search for this count record
will result in a subsequent read of data starting here.
Count
Data
Control
Text
Previous EXCP
A search for this count record
will result in a subsequent read count, key and data starting here.

Figure 16. Typical Load Module (Physical Format on Direct-Access Device)

End-of-Extent Appendage

A load module may reside in one or more extents on a direct-access device. The boundaries of these extents are specified in the data extent block (DEB) for the library containing the module being loaded. When an EXCP macro instruction is issued that results in crossing one of the extent boundaries within which a portion of the module being loaded resides, the input/output supervisor passes control to program fetch's end-of-extent appendage. The appendage acquires the beginning extent boundary for the next portion of the load module from the DEB, places it into the unit control block (UCB), and returns control to the input/output supervisor.

Input/Output Errors

All input/output errors are handled by the I/O supervisor, except incorrect length errors occurring while reading control and/or RLD records.

Note: For PCI fetch, all input/output errors are handled by the I/O supervisor.

Normally, an incorrect length indication is expected when reading control and/or RLD records, since they are variable length and their specific length is not known in advance. After reading such a record with a maximum possible count (256 bytes), program fetch examines the content of the record to check that what was read was of correct length. If this check fails, program fetch makes one more attempt to read the record, this time with the exact expected count. If the attempt to reread fails, control is given to the caller and an error code is passed.

RELOCATION (ADJUSTING ADDRESS CONSTANTS)

Program fetch adjusts address constants by adding (or subtracting) a relocation factor to (or from) the address constant's value that is embedded in the load module.

When a module is block loaded, its relocation factor is the difference between its linkage editor assigned address, which is always zero, and the first byte of main storage into which the module is to be loaded. For example, assume a module is to be loaded into main storage beginning at address 4000. If the RLD flag bit is positive a relocation factor of +4000 is added to the relocatable address constant. If, however, the RLD flag bit is negative, the relocation factor is subtracted from the address constant (see Appendix D for RLD entry format). The linkage editor assigned address of every relocatable address constant is given by the relocation dictionary (RLD).

Address constants in the root segment of an overlay module are adjusted in the same manner as those in a block module. The root segment's relocation is used to adjust the address constants of all segments of the module since an overlay module is essentially block loaded. The relocation factor is stored in the NOTE list by program fetch and is available throughout the execution of the overlay module.

TERMINATION

When a block module or the root segment of an overlay module has been loaded, program fetch computes the relocated entry point of the module and places it in the fetch (parameter) list. When a root segment of an overlay module is loaded, program fetch also inserts the main storage address of the data control block (DCB) and the NOTE list into the segment table (SEGTAB).

To specify a successful or unsuccessful loading, program fetch passes the appropriate termination code to its caller. Control is then returned to the caller via a branch to the address in the link/return register.

The overlay supervision service routines control the loading of overlay program segments and assist the flow of control between the segments of an overlay program. While performing these functions, these routines place data into and use data from the segment table (SEGTAB) and the entry tables (ENTABs).

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

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

Program fetch and the TESTRAN interpreter each return control to the overlay supervisor after their functions have been performed.

SEGLD is not supported in this configuration; a SEGLD request is treated as a NOP instruction.

TABLES USED BY OVERLAY SUPERVISION

The segment table (SEGTAB) and the entry tables (ENTABs) that contain the data used by the overlay supervisor are created by the linkage editor from information in the relocation dictionary (RLD) and the user's control statements.

Figure 17 shows the SEGTAB and ENTABs in a typical single region overlay structure; the ENTAB and SEGTAB formats are given in Appendix F.

USE OF SEGMENT TABLE

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

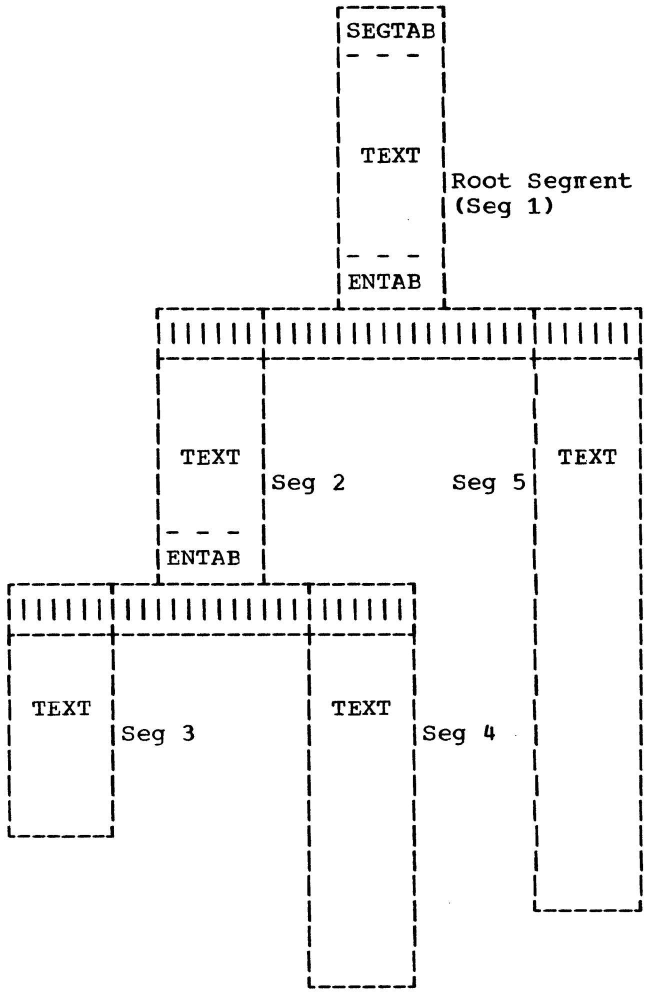


Figure 17. Single-Region Overlay Structure

- TEST indicator. This indicator is set by program fetch if the partitioned organization directory record indicates that the program is being tested under TESTRAN.

- Last segment number of each region. This value defines the segment that ends a region and is used to determine the region that contains a particular segment.

- Previous segment number of each segment in the module. The overlay supervisor uses this field to determine the additional segments that must be loaded with the requested segment. (These additional segments are those in the path of the requested segment.)

The variable information includes:

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

For more information about the SEGTAB, see Appendix F.

USE OF ENTRY TABLES

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

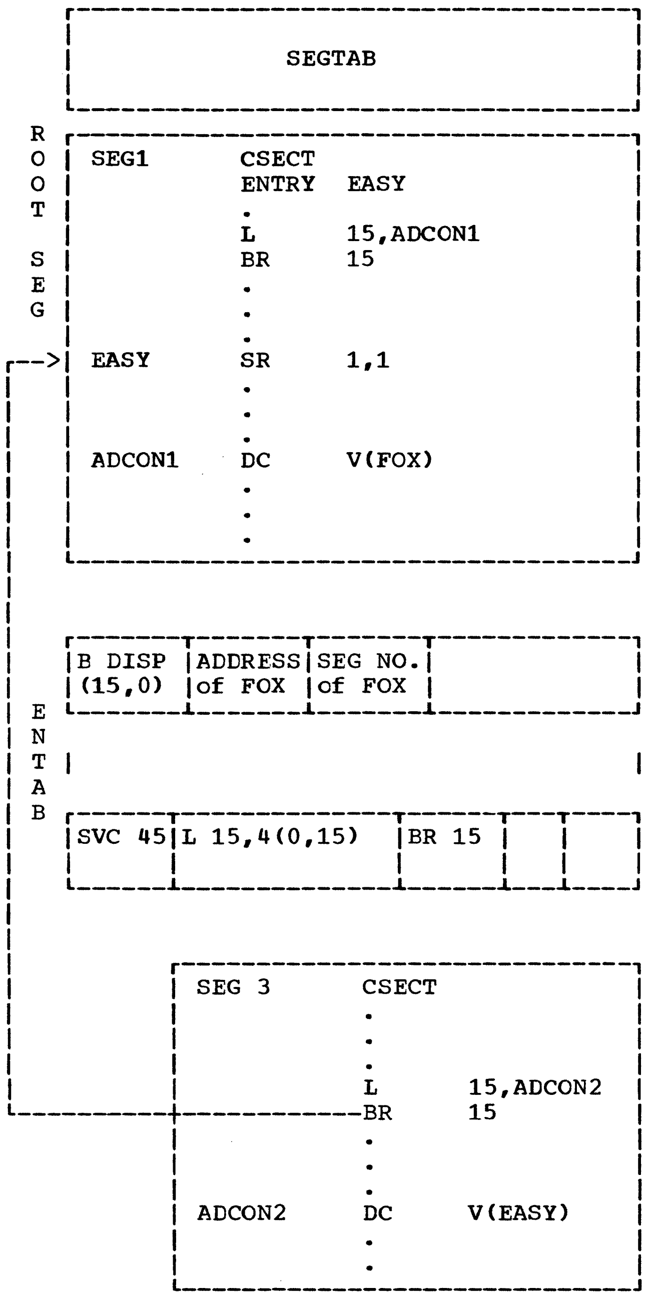
When the overlay program executes an upward branch, the overlay supervisor is not entered, and the ENTABs and SEGTAB are not used. An upward branch is direct because segments in the path are always in main storage (Figure 18).

Branching to a Segment Not in Main Storage

When an overlay program branches to a segment not in main storage, control is passed to the applicable ENTAB (step A of Figure 19). The branch instruction in the ENTAB passes control to an SVC instruction contained in the first field of the last ENTAB entry (step B). The SVC instruction causes an SVC interruption, and passes control to the SVC handler and then to the overlay supervisor (step C). The overlay supervisor uses a pointer in general register 15 to obtain the information required to:

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

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



```
                SEGTAB

ROOT SEG
   SEG1        CSECT
               ENTRY   EASY
               .
               L       15,ADCON1
               BR      15
               .
               .
               .
-->EASY        SR      1,1
               .
               .
               .
   ADCON1      DC      V(FOX)
               .
               .
               .

ENTAB
   |B DISP |ADDRESS|SEG NO.|
   |(15,0) |of FOX |of FOX |

   |SVC 45|L 15,4(0,15)  |BR 15 |   |   |

   SEG 3       CSECT
               .
               .
               .
               L       15,ADCON2
               BR      15
               .
               .
               .
   ADCON2      DC      V(EASY)
               .
               .
```

Figure 18. Overlay Program Upward Branch

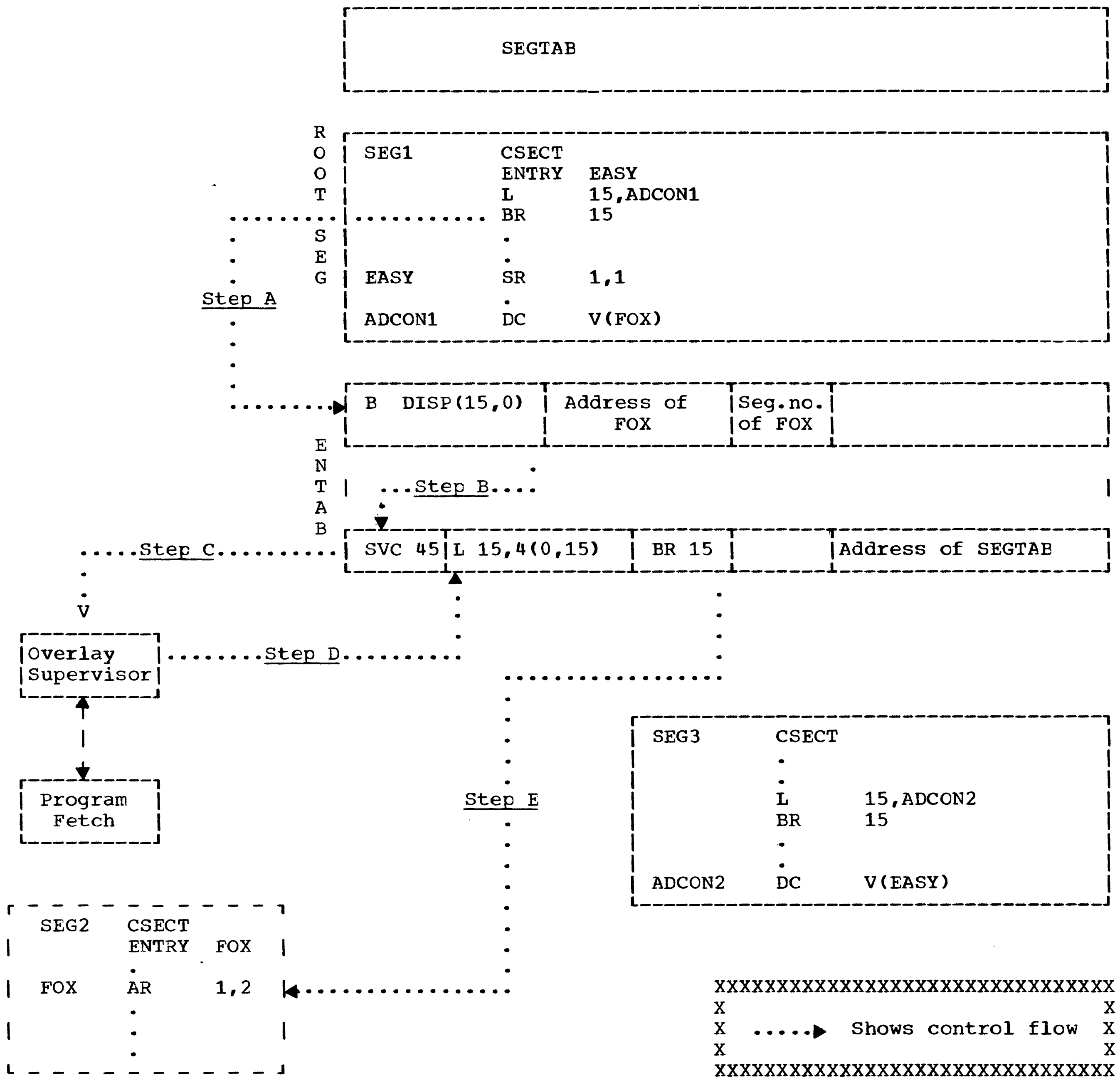


Figure 19. Branch to Segment not in Main Storage

<u>Branching to a Segment in Main Storage</u>

When a segment is loaded into main storage, because of an implicit call (a branch through an ENTAB), the displacement (DISP) field in the ENTAB entry through which the branch was routed is increased by 2 (Figure 20). When the overlay program executes another branch to this ENTAB entry, the SVC instruction is bypassed, and control is given to the second field of the last ENTAB entry. Execution of the instruction in this field causes general register 15 to be loaded with the main storage address assigned to the indicated symbol. A branch to that location is then executed.

A <u>caller</u> is an ENTAB entry that assisted in routing a branch from a segment to an entry point in a segment lower in the path. ENTAB entries that have been modified to bypass the SVC instruction are chained together in a <u>caller chain</u> (Figure 21).

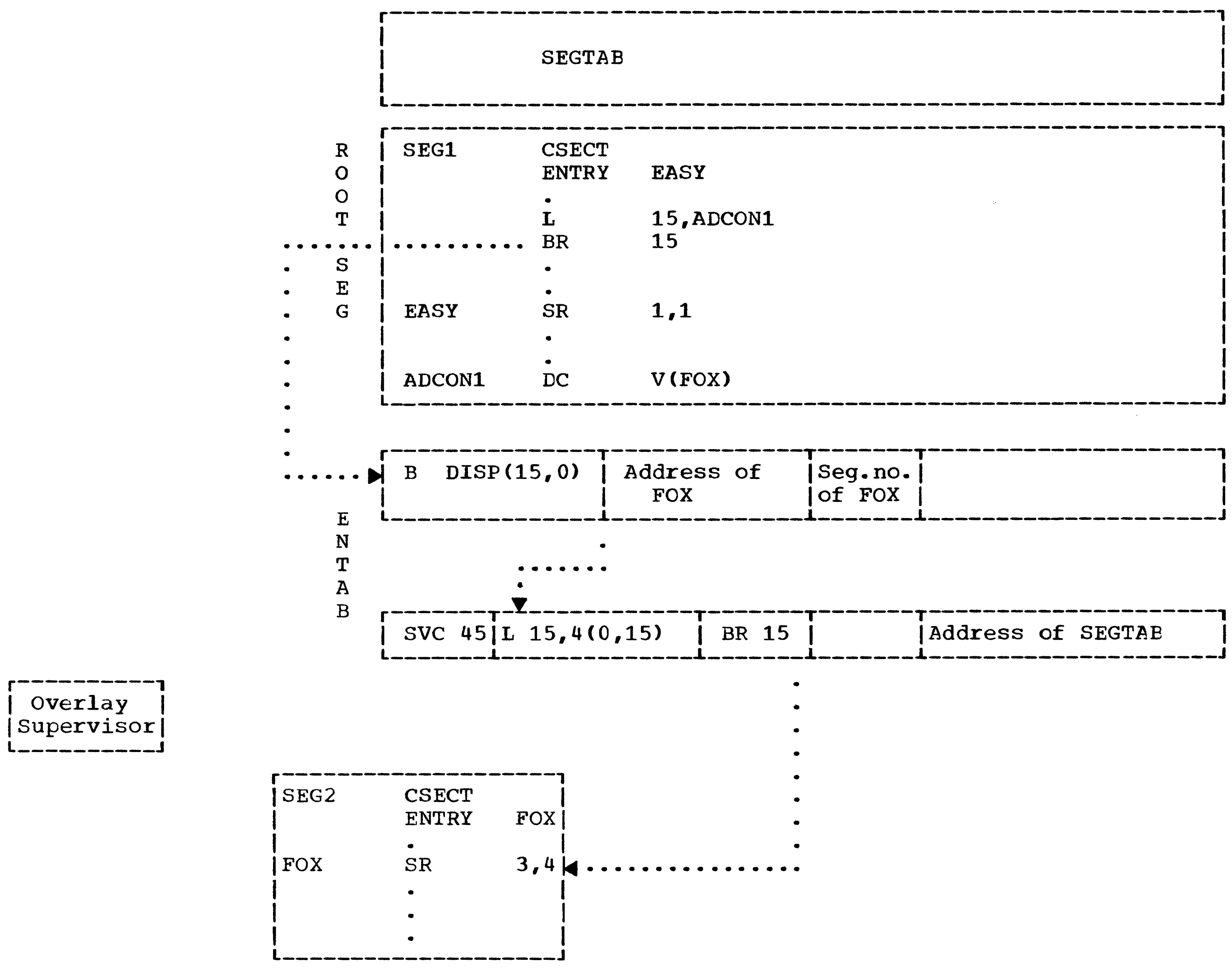


Figure 20. Branch to Segment in Main Storage

These entries are chained only if the called and calling segments are located in the same region. Chaining is accomplished by placing a pointer to (address of) the modified ENTAB entry into the caller field of the SEGTAB when the segment is brought into main storage. If this segment is requested again, the contents of the SEGTAB caller field (a pointer to a previous caller) is placed into the <u>previous caller</u> field of the referred to ENTAB entry, and a pointer to this ENTAB entry is placed in the caller field of the SEGTAB. In this way, a chain is created that begins at the SEGTAB entry and points to all the ENTAB entries (in the same region) that were modified (+2) to bypass the SVC 45 instruction. When the segment is to be overlayed, the caller chain is used to reset all of the modified ENTAB entries in the chain.

OVERLAY SUPERVISION ROUTINES

Overlay supervision is composed of a resident module called overlay supervisor 1 and either of two non-resident modules selected during system generation called overlay supervisor 2.

The module name of overlay supervisor 1 is IEWSVOVR; the module name of overlay supervisor 2 is IEWSYOVR for the basic synchronous module or IEWSXOVR for the basic synchronous module with optional SEGWT checking. To pass control to either version of overlay supervisor 2, overlay supervisor 1 issues a LINK macro instruction that specifies IEWSZOVR, which is the member name of the selected module in the LINKLIB.

OVERLAY SUPERVISION CONTROL FLOW

The resident module has two entry points: IGC037 and IGC045. The SVC handler passes control to IGC037 as a result of an SVC 37 instruction (SEGWT macro instruction), or to IGC045 as a result of an SVC 45 instruction (an intersegment branch that is routed through an ENTAB). An SVC 37 instruction with zero in general register 0 specifies a SEGLD macro instruction, whereas a one in general register 0 specifies a SEGWT macro instruction. (SEGLD is treated as a NOP in a single-task environment.) Chart 11 shows overlay supervisor control flow.

Overlay supervisor 1 is permanently resident in the nucleus of the operating system. It performs the first portion of initialization and then links to overlay supervisor 2. When control is returned to overlay supervisor 1, it performs the remaining termination procedures and issues an SVC EXIT instruction.

When a requested program is an overlay program, contents supervision issues a LOAD macro instruction to bring overlay supervisor 2 into main storage. Overlay supervisor 2 remains in main storage for the duration of the task that required it. When given control by overlay supervisor 1, overlay supervisor 2 performs the remaining initialization procedures, loads the requested segments, updates the segment table (SEGTAB) and entry tables (ENTABs), performs some termination procedures, and then returns control to overlay supervisor 1.

INITIALIZATION

During linkage editor processing, if the address constants of a segment are resolved to an ENTAB, the number of the segment is placed in the high-order byte of the address constants. The V-type address constants that are not resolved to an ENTAB contain a zero in their high-order bytes. The address constants can be the result of an expansion of a SEGLD, SEGWT, or CALL macro instruction, or the result of the user creating an address constant for use with a branch instruction. If a SEGLD or SEGWT request is received and the high-order byte of the V-type address constant is zero, the request is treated as a NOP.

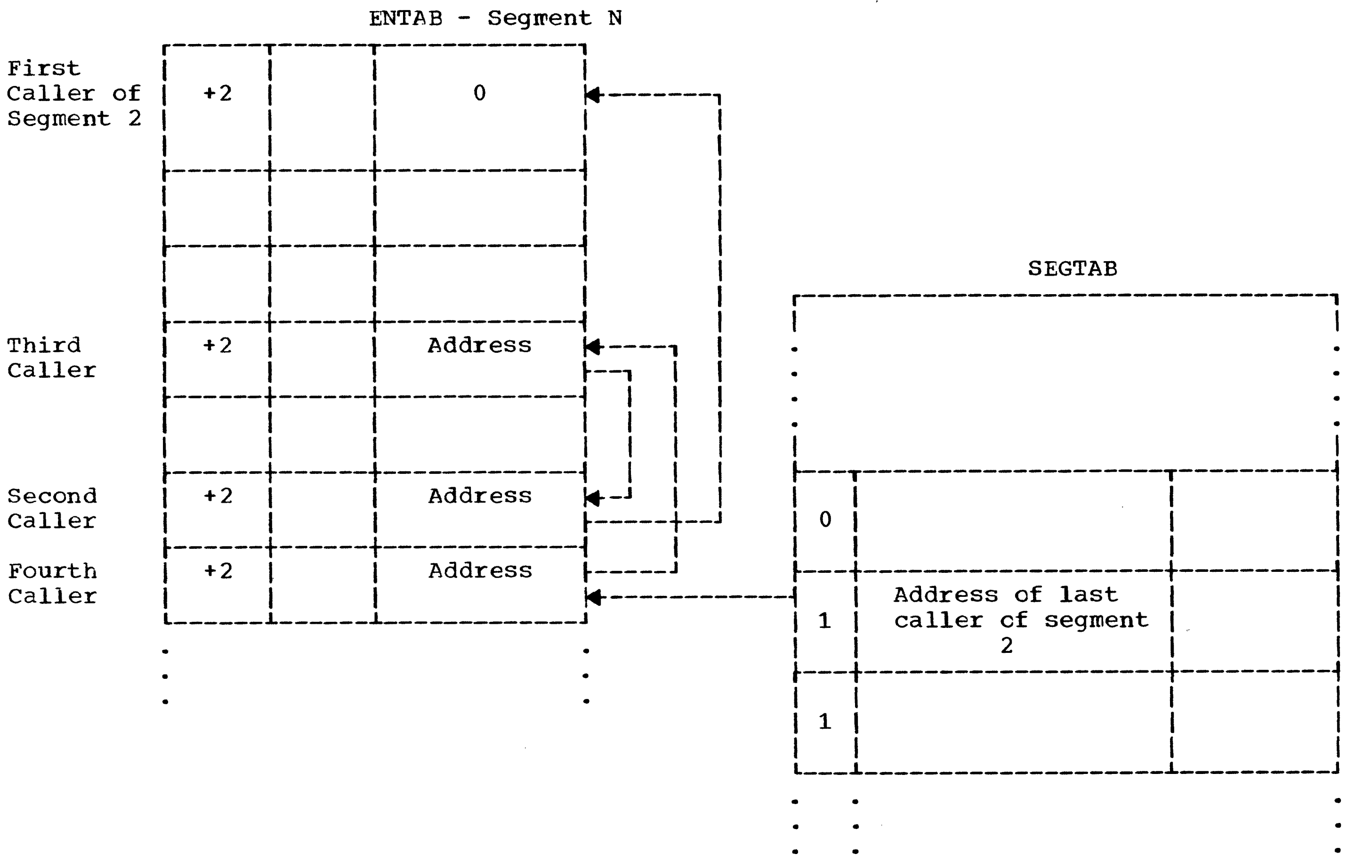


Figure 21. Chaining of ENTAB Entries Used to Branch to a Segment

The overlay supervisor obtains the segment number of the requested segment from the "to segment number" field in the ENTAB. The overlay supervisor obtains the address of the SEGTAB from the last entry in the ENTAB, and checks the SEGTAB to determine the segment's status and relationship to the overlay structure.

The basic synchronous module with optional checking (IEWSXOVR) detects overlay requests that would cause the requesting segment to be overlayed. This module checks only those requests that result from the execution of a SEGWT macro instruction.

UPDATING TABLES

Before segments are loaded, the overlay supervisor updates the SEGTAB and ENTABs of the overlay program to reflect the changes to be made in the overlay structure present in main storage. For each segment that is logically overlayed, a status indicator is reset in the SEGTAB. The SEGTAB is scanned to find the caller chains (Figure 19), which are used to reset the ENTAB entries to their original state (the state before the segment containing the corresponding entry point was loaded into main storage). The ENTAB entries are reset by subtracting +2 from the displacement field of the branch. When the SEGTAB and ENTAB entries of the last segment have been updated, the segments are loaded.

SEGMENT LOADING

During segment loading, the overlay supervisor scans the SEGTAB to determine which segments are needed and directs program fetch to load the requested segment and all segments in its path that are not in main storage.

TERMINATION

The overlay supervisor checks the TEST indicator in the SEGTAB to determine if the overlay program is "under test". If under test, a LINK macro instruction is issued specifying the TESTRAN interpreter. After TESTRAN interpreter execution, control is returned to overlay supervisor.

If the overlay supervisor was entered via an SVC 45 instruction (through an ENTAB), and the ENTAB through which the request was routed is in the root segment or is in the same region as the requested segment, the caller chain is updated (Figure 19) and the address field of the branch is altered in the calling ENTAB. If the requesting and requested segment are not in the same region, the caller chain and the branch instruction in the ENTAB are not altered. Subsequent branches to an altered ENTAB entry are routed directly to the segment. Control is returned to overlay supervisor 1.

The time supervision service routines are an optional feature of the fixed-task supervisor for installations that have selected the hardware timer as a part of their Computing System/360.

Time Supervision processes:

1. TIME macro instructions-- requests for the date and time of day.
2. STIMER macro instructions-- requests to establish an interval to be timed.
3. TTIMER macro instructions-- requests for the time remaining in a previously established interval, or requests to cancel a previously established interval.

Time supervision also maintains a queue of pending time requests and maintains the relationship between the actual time of day and the hardware.

TIME SUPERVISION ROUTINES

Time supervision includes the following service routines: timer second level interruption handler (SLIH), STIMER, TIME, and TTIMER.

The timer SLIH handles all types of interval expirations, including those of the control program, and maintains the queue of time interval requests.

The STIMER service routine sets an interval into a software interval timer, specifies when that interval timer is to be decremented and what action is to be taken when an interruption signals completion of the interval. It does these things in response to an STIMER macro instruction.

The TIME service routine places the time of day in register 0 and the current date in register 1, when requested through a TIME macro instruction. The time returned is the time of day based on a 24-hour clock that is set with local time by the operator through the SET command.

The TTIMER service routine tests the interval timer in response to a TTIMER macro instruction, and places in register 0 the time remaining in the TASK or REAL interval previously set by an STIMER macro instruction. The TTIMER service routine can also cancel previously specified intervals.

THE TIMING ALGORITHM

Within the timer SLIH is a 4-byte field called the 6-hour pseudo clock (SHPC). By manipulating the values contained in the SHPC and the hardware timer, time supervision maintains real time while timing a prespecified interval.

For example, assume that the 6-hour time of day (TOD), defined as equal to the contents of the SHPC minus the contents of the hardware timer, is zero hours. A request is received for a one hour interval. This is accomplished by placing one hour in the SHPC and in the timer.

SHPC - timer = 6-hour TOD
1 hour - 1 hour = 0 hour

After an hour, the contents of the timer have automatically decremented to zero and an interruption occurs.

SHPC - timer = 6-hour TOD
1 hour - 0 hour = 1 hour

If a 2-hour interval is requested, two hours is added to the SHPC and two hours is placed in the timer.

SHPC - timer = 6-hour TOD
(1 hour + 2 hours) - 2 hours = 1 hour

Two hours later, when the interruption occurs, the correct 6-hour TOD of three hours is indicated by the SHPC.

To correlate the internal, software pseudo clock time with real time, two other pseudo clocks are maintained by time supervision. One is a 24-hour pseudo clock (T4PC). The other is a local time pseudo clock (LTPC).

Each time the SHPC reaches six hours the SHPC is reset to zero and six hours is added to T4PC. The T4PC is reset to zero each time 24 hours pass. The T4PC is initially set to zero at initial program load. The contents of the T4PC plus the 6-hour TOD is defined as the T4PC TOD.

The contents of the LTPC initially is equal to the time keyed in at the console by the operator through the SET command. The local time of day which is returned, when requested, is computed by adding the contents of the LTPC to the T4PC TOD.

The three basic time relationships of the timing algorithm are:

- The 6-hour TOD is equal to the contents of the 6-hour pseudo clock minus the contents of the hardware timer.
- The 24-hour TOD is equal to the contents of the 24-hour pseudo clock plus the 6-hour TOD.
- The local TOD is equal to the contents of the local time pseudo clock plus the 24-hour TOD.

Time supervision maintains a queue (Figure 22) of timer queue elements (Figure 23) representing interval requests. The timer queue is a two-way chain ordered so that the request for the next interruption is at the top of the queue, while the request for the last interruption is at the bottom of the queue. To ensure that the timer queue element is inserted at the right place in the queue when a new request is received, the interval requested is translated into a value that is relative to the software clocks. This is done by adding the value of the interval requested to the 6-hour TOD. This new value is placed in the TQVAL field of the timer queue element and is used by the queueing subroutine of the timer SLIH to position the element on the queue.

```
   ┌──────────────────────────────────┐
   │ SHPC = 6-Hour Pseudo Clock       │
   └──────────────────────────────────┘

   ┌──────────────────────────────────┐
   │ T4PC = 24-Hour Pseudo Clock      │
   └──────────────────────────────────┘

   ┌──────────────────────────────────┐
   │ LTPC = Local-Time Pseudo Clock   │
   └──────────────────────────────────┘

   ┌──────────────────────────────────┐
   │ TQPTR = Pointer to Timer Queue   ├──┐
┌─>└──────────────────────────────────┘  │
│                                        │
│  ┌──────────────────────────────────┐<─┘
└──┤          6-Hour Element          ├──┐
┌─>└──────────────────────────────────┘  │
│                                        │
│  ┌──────────────────────────────────┐<─┘
└──┤        Midnight Element          ├──┐
┌─>└──────────────────────────────────┘  │
│                                        │
│  ┌──────────────────────────────────┐<─┘
└──┤          Pseudo Element          │
   └──────────────────────────────────┘
```

Figure 22. Timer Queue

```
┌─────────┬─────────┬───────────────────┐
│ Flags   │ TCB     │ Pointer           │
│         │ Pointer │ to Successor      │
├─────────┴─────────┼───────────────────┤
│ Pointer           │ TQVAL = Time of   │
│ to Predecessor    │ Expiration (TOX)  │
├───────────────────┼───────────────────┤
│ PRB               │ Exit              │
│ Pointer           │ Pointer           │
├───────────────────┴───────────────────┤
│                                       │
│                                       │
│      Save Area for 16 Registers       │
│                                       │
│                                       │
└───────────────────────────────────────┘
```

Figure 23. Timer Queue Element (96 Bytes)

When the element reaches the top of the queue, the interval placed in the timer is calculated by subtracting the value of the contents of the SHPC from the value of the contents of the TQVAL field of the element. The result of this subtraction is added to the timer, while the unsubtracted value of the contents of the TQVAL field of the element is placed in the SHPC.

At initial program load, two permanent entries are placed on the timer queue representing time supervision interval requests. One is a 6-hour interval request and the other is a request for an interval that is calculated to cause an interruption at midnight, local time. When the midnight interruption occurs, time supervisor increments by one the day-of-the-year count obtained from the operator's SET command. When the six-hour interruption occurs, time supervision updates the T4PC and decrements by six hours the contents of the TQVAL field in each of the elements in the timer queue. In addition, a pseudo element is placed at the end of the queue to mark the queue's terminal point.

TIME SUPERVISION CONTROL FLOW

As shown in Chart 12, the flow of time supervision is generally through two paths. In the first path, control is received from the SVC FLIH by one of the three SVC routines -- STIMER, TIME, and TTIMER. STIMER and TTIMER interface with the timer SLIH's queueing and dequeueing subroutines. TIME and TTIMER return by branching to the type 1 SVC exit, while STIMER executes an SVC EXIT instruction. In the second path, control is received from and returned to the T/E FLIH by the timer SLIH by branching.

STIMER

The STIMER service routine sets up time intervals, represented by timer queue elements, at the completion of which a timer/external interruption will occur. When entered, STIMER initializes the timer queue element's fields. STIMER uses the queueing subroutine of the timer SLIH to insert the newly created timer queue element into the timer queue. If a WAIT interval is requested, STIMER executes an SVC WAIT instruction.

TIME

The flow through the TIME service routine consists of testing the input parameters of the TIME macro instruction for the existence of the various options.

The time -- whether formatted in 26-microsecond timer units, ten-millisecond binary units, or packed decimal form -- is always given in terms of local time of day (LTOD). This is calculated according to the formula:

LTOD = LTPC + T4PC + SHPC-timer

where LTPC is the contents of the local time of day pseudo clock, T4PC is the contents of the 24-hour pseudo clock, SHPC is the contents of the 6-hour pseudo clock, and timer is the contents of the hardware timer at location 80.

The local time of day is placed in register 0, and the day of the year in register 1.

TTIMER

The TTIMER service routine determines how much time remains in an interval requested by a previous STIMER macro instruction, and cancels the interval if the CANCEL parameter is present.

When entered, the TTIMER routine determines whether the interval has expired. If it has, no action is taken. If it has not, the time remaining in the tested interval is returned to the user in register 0. TTIMER tests for the cancel option and, if it is present, TTIMER uses the dequeueing subroutine of the timer SLIH to take the timer queue element off the timer queue.

TIMER SLIH

The timer SLIH receives control from the T/E FLIH when a timer interruption occurs. The SLIH identifies the type of interval that has expired and then satisfies the specific requirement.

The SLIH removes the expired timer queue element from the timer queue through one of its two major subroutines (the dequeueing subroutine) resets the hardware timer to time the next interval on the queue, and resets the SHPC. The action taken by the SLIH after an expiration depends on the interval type:

- If it is a WAIT type, the SLIH executes the SVC POST instruction.
- If it is a REAL or TASK type, and an exit address was specified, the exit is scheduled through the Exit Effector routine.
- If it is a 6-hour time supervision type, six hours is subtracted from the TQVAL field of each timer queue element, and the 6-hour interval request is queued again.
- If it is a midnight time supervision type, the day-of-the-year count is incremented by one and the midnight interval request is queued again.

Queueing Subroutine

The queueing subroutine of the timer SLIH is used by the dispatcher, the SLIH, STIMER, and by the SET command handler of job management, to place a timer element on the timer queue. The dispatcher uses the routine when placing a task with a time interval request in control of the CPU.

The queueing subroutine converts the absolute time interval in the element to a relative time based on the 6-hour TOD. If the interval is found to be smaller than the current interval on the queue, the new smaller interval is added to the timer and placed in the SHPC. If the interval is not smaller, the correct insert point on the queue is located for the element, which is queued.

Dequeueing Subroutine

The dequeueing subroutine is used by the dispatcher, STIMER, and TTIMER to remove elements from the timer queue by pointer manipulation. If the element was at the top of the queue, control is passed to the SLIH, which resets the timer and SHPC. Control is passed back to the caller by a branch, at the completion of the dequeueing subroutine, unless a branch was made to the SLIH, which returns control directly to the caller.

CHAPTER 8: SYSTEM ENVIRONMENT RECORDING

System Environment Recording is a set of control program routines which record and in some cases attempt to reduce the effect of machine malfunctions in System/360 Models 40, 50, 65, and 75. System Environment Recording handles two types of machine malfunctions:

- Malfunctions of the central processing unit (CPU), which cause machine-check interruptions, and
- Malfunctions in a channel, which cause input/output interruptions.

There are two versions of System Environment Recording:

- System Environment Recording 0 (SER0), and
- System Environment Recording 1 (SER1).

Either of these versions may be selected when a system is generated for Models 40, 50, 65, or 75. If neither is selected, either SER0 or SER1 is used by default. The version used by default depends on the model (or models) specified, and on the size of the system (see IBM System/360 Operating System: System Generation, Form C28-6554).

SYSTEMS WITHOUT SYSTEM ENVIRONMENT RECORDING

When a machine malfunction (caused by a CPU or channel malfunction) occurs on an IBM System/360 model which does not have System Environment Recording, the computer is placed in a wait state (See Figure 24). If the system is a Model 30, the operator may then load the System Environment Recording, Editing, and Printing (SEREP) program. This program is described in IBM System/360: General Programming Considerations, Form Y20-0005.

ENTRY TO SYSTEM ENVIRONMENT RECORDING

When a machine-check interruption occurs, the machine-check new PSW is loaded. This causes control to pass directly to the System Environment Recording Routine which was selected during system generation (see Figure 24).

When an input/output interruption occurs because of a channel error, the I/O new PSW is loaded. This causes control to pass to the I/O FLIH and then to the I/O Supervisor. The I/O Supervisor enters the SER Interface Subroutine which then loads the machine-check new PSW (see Figure 24).

SER ROUTINES

SER0 is the less complex version of System Environment Recording. It determines the type of malfunction and, if possible, writes a record on the SYS1. LOGREC data set describing the error. SYS1.LOGREC is located on the primary system residence volume. If SER0 cannot write the record, the computer is placed in a wait state and a message is printed requesting that the operator use SEREP. If SER0 can write a partial or complete record, the computer is placed in a wait state and a message is printed requesting that the operator reload the operating system.

SER1 is the more complex version of System Environment Recording. It also collects and writes machine environment data, but in addition, it attempts to associate the malfunction with the task being executed. If the malfunction can be associated with the task and if the control program has not been damaged, the task is abnormally terminated. If not, the computer is placed in a wait state.

When the SYS1.LOGREC data set has been filled, the operator runs the Environment Recording Edit and Print (EREP) Routine. This routine formats the SYS1.LOGREC records and then writes these records onto printer, tape, or disk (according to user specifications). EREP is described in IBM System/360 Operating System: Utilities, Program Logic Manual, Form Y28-6614.

SER0

SER0 collects, formats, and writes error information after a machine-check or a channel error has occurred. See Charts 13 and 14. It is divided into two modules:

1. Module IFBSR000, resident in the nucleus, and
2. Module IFBSR0xx (where xx is the model number: 40, 50, 65, or 75), located on the link library. This module is model dependent. The required modules are selected during system generation.

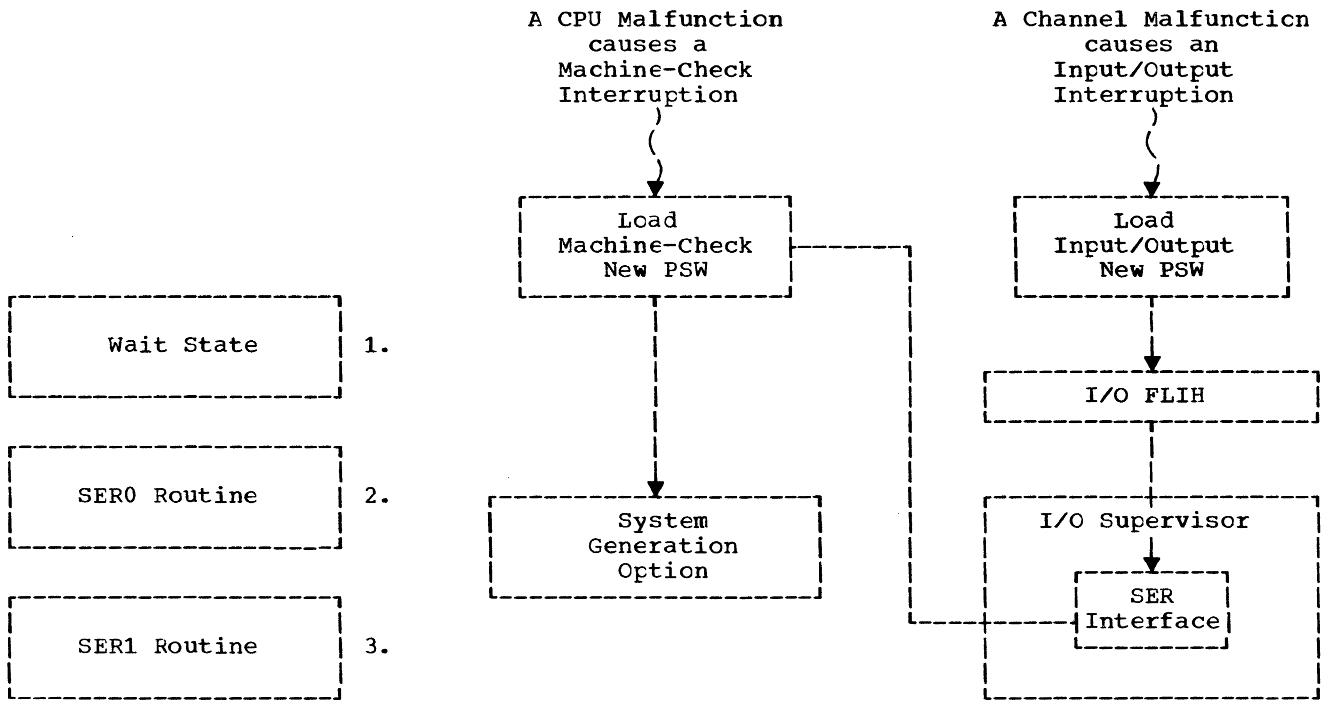

A CPU Malfunction causes a Machine-Check Interruption
A Channel Malfunction causes an Input/Output Interruption
Load Machine-Check New PSW
Load Input/Output New PSW
I/O FLIH
System Generation Option
I/O Supervisor
SER Interface
Wait State
1.
SER0 Routine
2.
SER1 Routine
3.

Figure 24. System Environment Recording

Resident Module -- IFBSR000

Module IFBSR000 is non-reusable and does not require operating system facilities. It halts all I/O activity and then reads the first text record of module IFBSR0xx into main storage (beginning 32 bytes past the end of the nucleus).

Module IFBSR000 saves information (in a 22 byte field in lower storage) to be used later by IFBSR0xx. After it has halted I/O activity on all devices, IFBSR000 attempts to read the first 1024 bytes of module IFBSR0xx into main storage. If after ten retries, these 1024 bytes have not been read into main storage, IFBSR000 builds IOS wait state code 000F0A, and then branches to the Bell Ring/Wait State module which sounds the console alarm and places the computer in a wait state. Wait state code 000F0A is displayed in the instruction counter.

Link Library Module -- IFBSR0xx

Like IFBSR000, module IFBSR0xx does not require operating system facilities. IFBSR0xx first loads the remainder of itself into main storage. It then checks location 50 to determine which type of malfunction has occurred, a machine-check error or a channel error. Location 50 is preassembled to X'FF'. If the error is a machine-check error, location 50 will have been overlayed by the machine-check old PSW. If the error is a channel error, location 50 remains unchanged.

If the error is a machine-check error, IFBSR0xx builds a machine-check record entry in which to place information about the error. If the error is a channel error, IFBSR0xx builds a channel error record entry. The formats of these records is shown in Appendix G.

Module IFBSR0xx then enables machine-check interruptions. General registers are checked for valid parity on all models except Model 40. Parity indicators are available for all registers except 13, 14, and 15 on Models 50 and 75. Floating point registers are also checked for valid parity if the model is equipped with floating point.

Module IFBSR0xx checks the "busy bit" in each unit control block (UCB) to determine which I/O units were busy when the error occurred. The addresses of as many as ten busy I/O devices are collected. IFBSR0xx then builds a record containing the program identification, jobname, stepname, day, and time. After examining the seek address obtained from the header record of the SYS1.LOGREC data set, IFBSR0xx writes (on that data set) the record it has just created and an end-of-file record.

Module IFBSR0xx then prints the following message to the operator:

IFBF05W MACHINE ERROR. RELOAD OS/360

This message indicates that a complete error record has been written on SYS1. LOGREC. If a message cannot be printed, IFBSR0xx builds IOS display code 000F05 and branches to the Bell Ring/Wait State module.

If another machine-check interruption occurs while IFBSR0xx is collecting data for an error record, IFBSR0xx stops collecting data and attempts to write a partial error record on SYS1.LOGREC containing the data it has already collected. If it is able to do this, it prints the following message:

IFBF06W MACHINE ERROR. RELOAD OS/360

If a message cannot be printed, IFBSR0xx builds IOS display code 000F06 and branches to the Bell Ring/Wait State module.

If another machine-check interruption occurs while IFBSR0xx is attempting to write a partial error record, IFBSR0xx cannot continue processing. It prints the following message:

IFBF07S MACHINE ERROR. EXECUTE SEREP

Other errors besides a machine-check interruption may prevent IFBSR0xx from writing an error record on SYS1.LOGREC. These errors (with the messages IFBSR0xx prints to the operator) are as follows:

1. An I/O error,

   IFBF08S MACHINE ERROR. EXECUTE SEREP

2. SYS1.LOGREC data set is full,

   IFBF09S MACHINE ERROR. EXECUTE SEREP

3. Module IFBSR0xx could not be loaded into main storage,

   IFBF0AS MACHINE ERROR. EXECUTE SEREP

SER1

Like SER0, SER1 collects, formats, and writes error information after a machine-check or a channel error has occurred. See Charts 15 and 16. SER1, unlike SER0, is a single, serially reusable module that resides in the nucleus.

In addition to writing error records, SER1 attempts to associate the error with the task which was executing. If it can do this, and if the control program is not damaged by the error, SER1 abnormally terminates the task. The system continues to operate.

If SER1 cannot write a complete error record or cannot associate the error with the task, or if the error damages the control program, the computer is placed in a wait state. The system must then be reloaded.

SER1 checks location 50 to determine which type of malfunction has occurred, a machine-check error or a channel error. Location 50 is preassembled to X'FF'. If the error is a machine-check error, location 50 will have been overlayed by the machine-check old PSW. If the error is a channel error, location 50 remains unchanged.

SER1 gathers error data into either a machine-check record entry or a channel error record entry and writes the record on SYS1.LOGREC. SER1 uses I/O routines provided by the operating system (it uses the EXCP macro instruction to communicate with the SYS1.LOGREC data set) unless the control program was damaged by the error. If the control program was damaged, SER1 uses its own I/O routines. The DEB and DCB required when EXCP is used reside in the nucleus and are opened by the nucleus initialization program (NIP).

If SER1 can associate the error with the task and if the control program is not damaged, SER1 terminates the task by branching to the ABTERM routine. When SER1 regains control from ABTERM, it reinitializes itself and branches to the dispatcher so that the system can continue to operate.

In order for the system to continue operating:

1. Another error cannot occur while SER1 is collecting data for a previous error. If one does, SER1 stops collecting data and attempts to write a partial record of the original error on SYS1.LOGREC. The partial record contains the data collected before the second error occurred.

2. SER1 must be able to associate the error with the task which was executing.

3. The control program cannot be damaged by the error.

If the system cannot continue operating, SER1 prints a message on the primary output device instructing the operator to reload the operating system. SER1 then places the system in a wait state.

ENVIRONMENT RECORDING AREA

SYS1.LOGREC is a data set on the system residence device used exclusively by SER0, SER1, and all preservation recording systems. It is formatted during system generation by utility program IFCDIP00. The data placed in SYS1.LOGREC is edited and printed by utility program IFCEREP0 (EREP). These programs are described in IBM System/360 Operating System: Utilities, Program Logic Manual, Form Y28-6614.

SYS1.LOGREC contains three types of records:

1. Header Record - This is the first record in the data set. It defines the extent of the data set, and addresses the last record written. It also contains a safety byte used to detect overrun. The record is 38 bytes in length.
2. Statistical Data Record Area - This area contains a record for each unit control block (UCB) in the system.
3. Record Entry Area - This area begins on the track following the area occupied by statistical data records. SER0 and SER1 write machine-check records and channel error records in this area. The format of these records is described in Appendix G.

CHAPTER 9: CHECKPOINT/RESTART

The CHECKPOINT and RESTART service routines minimize the amount of time wasted when a program abnormally terminates. CHKPT macro instructions are used to divide the program into sections. When the program abnormally terminates, it can be restarted immediately (this is called _Automatic Restart_) or it can be restarted later by the programmer (this is called _Deferred Restart_).

If abnormal termination occurs in the first section of a program, restart begins at the beginning of the step. This is called _Step Restart_. If abnormal termination occurs in any other section, restart may begin at the beginning of that section. This is called _Checkpoint Restart_. Checkpoint restart eliminates the need to rerun sections of a program which have already run successfully.

If a program is coded using three CHKPT macro instructions, it is divided into four sections (see Figure 25). If abnormal termination occurs in section 3, an automatic checkpoint restart begins at CHKPT B (if the programmer has requested automatic restarts).

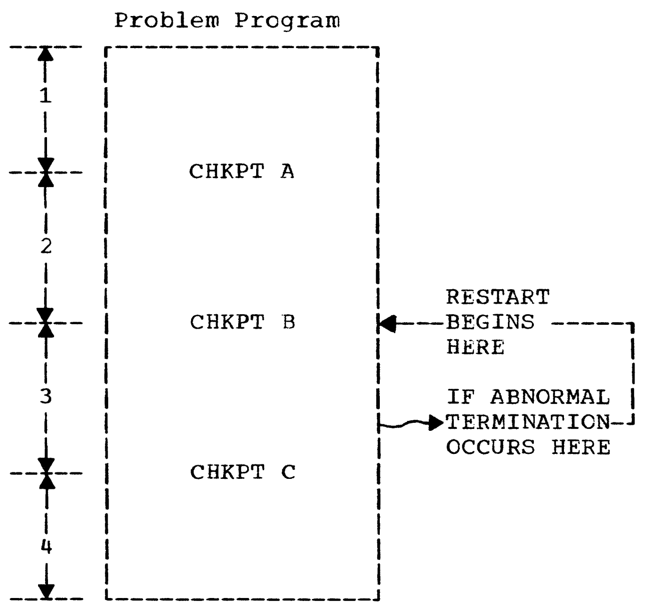


Figure 25. Problem Program Checkpoints

The CHECKPOINT routine is called directly when a problem program issues a CHKPT macro instruction. The RESTART routine is called by a job management program when a restart is scheduled. _Charts 17 and 18_ show the logic flow of these routines.

Appendix H contains the format of records used by CHECKPOINT/RESTART as well as a list of CHECKPOINT/RESTART SVC Modules and Register Usage Table.

CHECKPOINT (SVC 63)

The CHECKPOINT service routine:

1. Suspends user I/O requests,
2. Builds a CHECKPOINT entry and writes it in the CHECKPOINT data set,
3. Restores the user I/O requests,
4. Returns to the caller.

If the caller has suppressed checkpoints, through use of the RD parameter in job control statements, no CHECKPOINT entry is written.

The routine consists of 10 load modules which are executed in the SVC transient area after an SVC 63 instruction (CHKPT macro instruction) is issued. When the SVC 63 instruction is executed, an SVC interruption occurs and control passes to the SVC FLIH, the SVC SLIH, and to the first load module of the CHECKPOINT service routine (see Figure 26). The remaining load modules receive control via XCTL macro instructions.

A description of the 10 CHECKPOINT load modules follows. When reading this description, refer to _Chart 17_.

INITIALIZATION MODULES (IGC0006C, IGC0106C, IGC0206C)

The first load module of CHECKPOINT (IGC0006C) determines if checkpoints have been suppressed. _If they have_, an SVC 3 instruction is issued to pass control to the SVC EXIT routine and return to the caller. _If they have not_, module IGC0006C determines if the CANCEL operand was specified in the CHKPT macro instruction being serviced. _If CANCEL was specified_, processing continues as described in _CANCEL Processing_. _If CANCEL was not specified_, module IGC0006C issues an OPEN macro instruction for the CHECKPOINT data set (if the caller has not already opened the data set) and then issues a GETMAIN macro instruction for a work area in the dynamic area of main storage. The second load module (IGC0106C) tests the validity of the request. If an error is detected, control passes to checkpoint exit module IGC0Q06C. If no errors are found, the third load module (IGC0206C) reads the Job Control Table (JCT) into the work area, builds the CHECKPOINT Header Recorder (CHR) (see Appendix H), and passes control to the Check I/O Module.

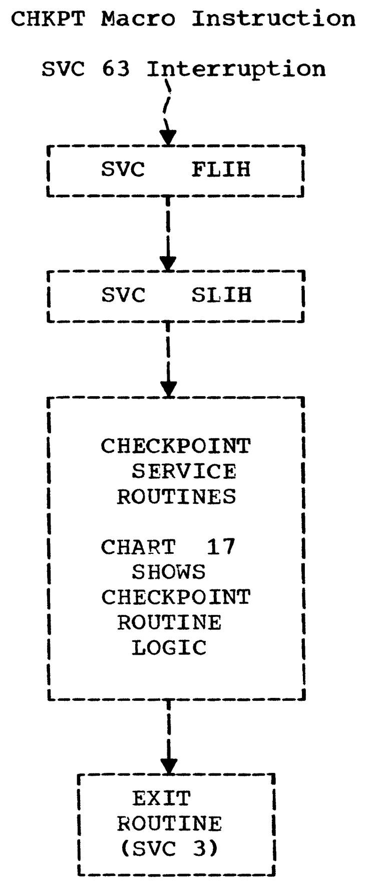


Figure 26. CHECKPOINT Routine Control Flow

CANCEL PROCESSING

The CANCEL operand of the CHKPT macro instruction indicates that the caller does not want to create a new CHECKPOINT entry, but wants to suppress automatic restarts from any previously created checkpoints. When CANCEL is specified, module IGC0006C issues a GETMAIN macro instruction to obtain a small work area and then passes control to module IGC0206C. Module IGC0206C reads the Job Control Table (JCT) into the work area and passes control to exit module IGC0Q06C.

Module IGC0Q06C sets a CHECKPOINT indicator to show that no CHECKPOINT entry has been written, and alters the Job Control Table (JCT) so it doesn't show the previous CHECKPOINT entries which have been written. Module IGC0Q06C then returns the Job Control Table to the input queue and returns control to the caller via an SVC 3 instruction. (No messages are written to the operator.)

If an abnormal termination occurs after CHKPT CANCEL processing has been completed, no automatic checkpoint restart is performed. However, the CHECKPOINT entries which have been written are retained, and the programmer can restart the step from one of these entries at a later time (by submitting the proper restart Job Control Language).

CHECK I/O MODULE (IGC0506C)

The Check I/O Module (IGC0506C) issues the PURGE macro instruction specifying the QUIESCE option for each Data Extent Block (DEB) associated with the caller's Task Control Block. This causes all of the caller's pending I/O requests to be removed from the Logical Channel Queues, or if already started, to be completed. If a permanent error occurs in a completing QSAM or QISAM I/O request, an error code is returned to the caller, and no checkpoint is written (unless the QSAM ACC option was specified for the data set). When all of the caller's I/O activity has completed, control passes to the next module (IGC0A06C).

PRESERVE MODULES (IGC0A06C, IGC0D06C)

The Preserve Modules (IGC0A06C and IGC0D06C) write the CHECKPOINT Header Record (CHR) created by the third module, then build and write a Data Set Descriptor Record (DSDR) for each Job File Control Block, Job File Control Block Extension, and Generation Data Group Bias Count Table. (Formats of these records are shown in Appendix H.) If end-of-volume occurs for the CHECKPOINT data set on tape, IGC0206C is called to attempt to rewrite with a new tape. If end-of-volume occurs for the second time on tape or the CHECKPOINT data set is on a direct access device and end-of-volume is detected or an I/O error occurs in either module, control is transferred via XCTL to the Resume I/O Module. If none of the above errors occur control then passes to the Checkmain Module.

CHECKMAIN MODULE (IGC0F06C)

The Checkmain Module (IGC0F06C) writes the contents of problem program main storage onto Core Image Records (CIRs).

Then a Supervisor Record (SUR) is constructed with task control information and written as the last record in the CHECKPOINT entry. (Formats of these records are shown in Appendix H.) Control then passes to the Resume I/O Module via an XCTL. If an I/O error occurs or end-of-volume is detected on either tape for the second time or on a direct access device, control passes to the Resume I/O Module with an error code. If end-of-volume occurs for the first time on tape, control is passed to IG0206C to reprocess the tape.

RESUME I/O MODULE (IGC0N06C)

The Resume I/O Module (IGC0N06C) issues the RESTORE macro instruction for each Data Extent Block (DEB) associated with a previously suspended I/O request. The requests are restored to the logical channel queues, and if possible, started. Control then passes to exit module IGC0Q06C.

EXIT MODULE (IGC0Q06C)

For a normal exit, module IGC0Q06C issues a STOW macro instruction if the CHECKPOINT data set has partitioned organization. It then issues a CLOSE macro instruction for the CHECKPOINT data set (unless the caller issued the OPEN), updates the CHECKPOINT flags and count fields in the Job Control Table (JCT), restores the JCT to the job queue, and frees the work area. For an exit after an error has occurred, the preceding functions are performed if necessary. Control then passes to the Message Module.

MESSAGE MODULE (IGC0S06C)

The Message Module (IGC0S06C) writes a message indicating successful or unsuccessful completion. One of the following return codes is placed in Register 15 before control is returned to the caller via an SVC 3 instruction:

X'00'	Valid CHECKPOINT entry written.
X'08'	No CHECKPOINT written; calling error.
X'0C'	Permanent I/O error.
X'10'	A valid CHECKPOINT entry was written, but there were outstanding ENQs. It is the responsibility of the user to restore these ENQs during RESTART.

<u>RESTART (SVC 52)</u>

The RESTART service routine uses information in a CHECKPOINT entry to recreate the conditions that existed when the CHECKPOINT entry was written.

The RESTART routine:

1. Restores the problem program to its original location in main storage,

2. Opens and positions any problem program data sets which were open when the CHECKPOINT entry was written,

3. Restores task control information,

4. Passes control to the problem program instruction immediately following the CHKPT macro instruction from which RESTART is occurring.

The routine consists of 14 load modules which are executed in the SVC transient area after an SVC 52 instruction is issued. Before the SVC 52 instruction is issued, a job management routine (IEFDSDRP) adjusts the job queue, and assures that device allocations are compatible with those which were in effect when the CHKPT macro instruction was issued. Just before exiting, IEFDSDRP changes the name of the restarting step to IEFRSTRT. This program consists of only an SVC 52 instruction.

When the SVC 52 instruction is executed, an SVC interruption occurs and control passes to the SVC FLIH, the SVC SLIH, and to the first load module of the RESTART service routine (see Figure 27).

A description of the 14 RESTART load modules follows. When reading this description, refer to <u>Chart 18</u>.

INITIALIZATION MODULES (IGC0005B, IGC0105B)

The first load module of RESTART (IGC0005B) receives the address of a parameter list built by job management routines from information in the CHECKPOINT Header Record (CHR). From this parameter list, RESTART determines what the problem program (dynamic) area boundaries were when the CHECKPOINT entry was written. It then issues a GETMAIN for the same area (this includes a RESTART work area). A Data Control Block (DCB) for the CHECKPOINT data set is contructed in the work area, and the RESTART SVRB and the current Task Input/Output Table (TIOT) are moved into the area. An OPEN macro instruction is issued for the CHECKPOINT data set, and the next module is called.

The second load module (IGC0105B) moves additional CHECKPOINT data set control blocks into the work area. It positions the CHECKPOINT data set at the first Core Image Record (CIR) and calls the Repmain Module.

REPMAIN MODULE (IGC0505B)

The Repmain Module (IGC0505B) reads the Core Image Records (CIRs) into problem program storage, and reads the Supervisor Record (SUR) into the work area. (Formats of these records are shown in Appendix H.) The address of the "old" Free Area Queue Element (FQE) is moved to the Boundary Box, and the TCB fields saved by the CHECKPOINT routine are moved into the current TCB. The Repmain Module restores the floating-point registers, if any, and places the current protection key in all Program Request Blocks (PRBs). Control then passes to the first Job File Control Block (JFCB) processing module.

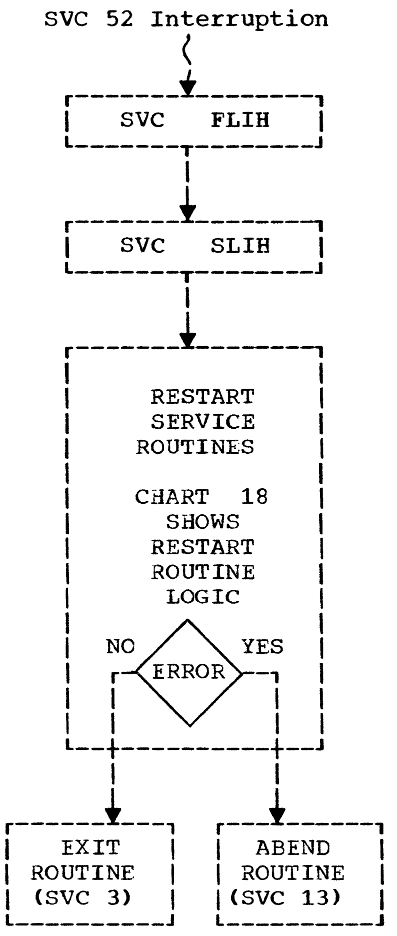


Figure 27. RESTART Routine Control Flow

JOB FILE CONTROL BLOCK PROCESSING MODULES (IGC0G05B, IGC0I05B)

The first Job File Control Block (JFCB) processing module (IGC0G05B) creates a table in the work area for each JFCB associated with a data set that was OPEN when the CHKPT macro instruction was issued. A DEB, DCB, IOB, and ECB are constructed within the table for later repositioning I/O operations. Control then passes to the second JFCB processing module.

The second Job File Control Block (JFCB) processing module (IGC0I05B) reads in a JFCB Extension for Sequential Access Method (SAM) data sets which reside on more than five volumes. Several extensions may be read until the one containing the volume serial number which was in use when the CHKPT macro instruction was issued is found. Unless all problem program data sets reside on direct access storage devices, control passes to the first Mount/Verify module. If all data sets are on direct access storage, control passes to the Direct Access Mount/Verify Module.

MOUNT/VERIFY MODULES (IGC0K05B, IGC0M05B)

The first Mount/Verify module (IGC0K05B) processes all data sets except those residing on direct access devices. For SYSIN, SYSOUT, unit record, and graphics data sets, processing consists only of adjusting the Data Extent Block (DEB).

For magnetic tape data sets, the volume serial number in the primary Unit Control Block (UCB) in the data set's Task Input/Output Table (TIOT) entry is compared to the volume serial number in the work area table entry built from the Data Set Descriptor Record (DSDR). If they are the same, the necessary adjustments are made to the UCB and the DEB. If the volume serial numbers do not match, the secondary UCBs (if any) are searched. If the correct volume is specified in one of them, it becomes the primary UCB. If the volume is not mounted, a suitable UCB is selected from the TIOT entry, and a MOUNT message is written to the operator. For any tapes with nonstandard labels, a user-supplied verification subroutine is called. After all tape data sets are processed, control passes to the Direct Access Mount/Verify Module or to the Non-Direct Access Processor Module.

The Direct Access Mount/Verify Module (IGC0M05B) performs exactly the same functions as the first Mount/Verify module, except that no label checking is done, and all volumes of a concatenated data set with partitioned or direct access organization are mounted. An error, such as no suitable UCB for a volume, causes RESTART to terminate with an error message. If no error occurs, control passes to the first direct access Position I/O module (IGC0N05B) or to the Non-Direct Access Processor Module.

NON-DIRECT ACCESS PROCESSOR MODULE (IGC0L05B)

This module is used only in PCP. The Non-Direct Access Processor Module (IGC0L05B):

1. Writes SYSOUT tape header labels for deferred restarts,
2. Primes buffers for the card reader.

To write header labels, a tape must already be mounted and must have been positioned (by the scheduler) beyond the tape mark which closes the previous file. The JFCB associated with the data set is read into main storage. Information from this JFCB is used to write the header labels. All label fields are in EBCDIC and the labels are followed by a tape mark.

To prime buffers, the user must keep a count of the number of GETs the problem program issued before the CHKPT macro instruction was issued. If the access method is in move mode, all buffers are primed. If it is in locate mode, all but one buffer is primed.

If an I/O error occurred, control passes to the RESTART Exit Module. If not, control passes to module IGC0N05B to position direct access data sets, or to module IGC0P05B if there are no direct access data sets.

POSITION I/O MODULES (IGC0N05B, IGC0Q05B, IGC0P05B, IGC0R05B)

The first direct access Position I/O module (IGC0N05B) determines if any direct access data sets have been deleted. When processing is completed, control passes to the second direct access Position I/O module (IGC0Q05B). This module performs no function in PCP. It passes control to non-direct access Position I/O module IGC0P05B or to direct access Position I/O module IGC0R05B.

The non-direct access Position I/O module (IGC0P05B) moves magnetic tape data sets to where they were located when the CHKPT macro instruction was issued. IGC0P05B assumes the following:

- System input data sets have been positioned by the scheduler to the first data record of the user's input stream.
- Non-standard label data sets have been positioned by the user label routine to the first data record on the current volume.
- If this is a deferred restart, system output data sets have been positioned by module IGC0L05B. (If this is an automatic restart, system output data sets will be rewound and positioned now.)
- All other tape data sets are positioned at load point.

IGC0P05B positions tapes with standard labels or no labels to the first data record. Then, using the BLKCT field of the DCB, IGC0P05B advances each tape data set to where it was located when the CHKPT macro instruction was issued. If the BLKCT field is negative or zero, the data set is positioned to the beginning or end, depending on whether forward or backward processing was in progress when the CHKPT macro instruction was issued. Control then passes to the Final Processing Module or to direct access Position I/O module IGC0R05B.

Direct access Position I/O module IGC0R05B checks each data set residing on a direct access storage device to determine if the space allocation limits of the data set (described in the Data Set Control Block (DSCB) on the volume) have changed since the CHKPT macro instruction was issued. The limits which existed at that time are described in the Data Extent Block (DEB) saved by the CHECKPOINT routine. If the space allocation limits of an _input data set_ have changed (indicating that the data set has been modified), RESTART is terminated.

If the space allocation limits of an _output data set_ have changed, the smaller of the two space allocations is placed in both the DSCB and the DEB. If the DSCB allocation is reduced, the Partial Release module of the CLOSE routine is called to return the released space to the free area. When all direct access data sets have been checked, control passes to the Final Processing Module.

FINAL PROCESSING MODULE (IGC0T05B)

The Final Processing Module (IGC0T05B) reads the directories of any user output data sets with partitioned organization to detect members added after the CHKPT macro instruction was issued. If any are found, they are deleted with the STOW macro instruction.

Finally, the RESTORE macro instruction is used to reschedule any user I/O requests suspended by PURGE during CHECKPOINT processing. Control then passes to the RESTART Exit Module.

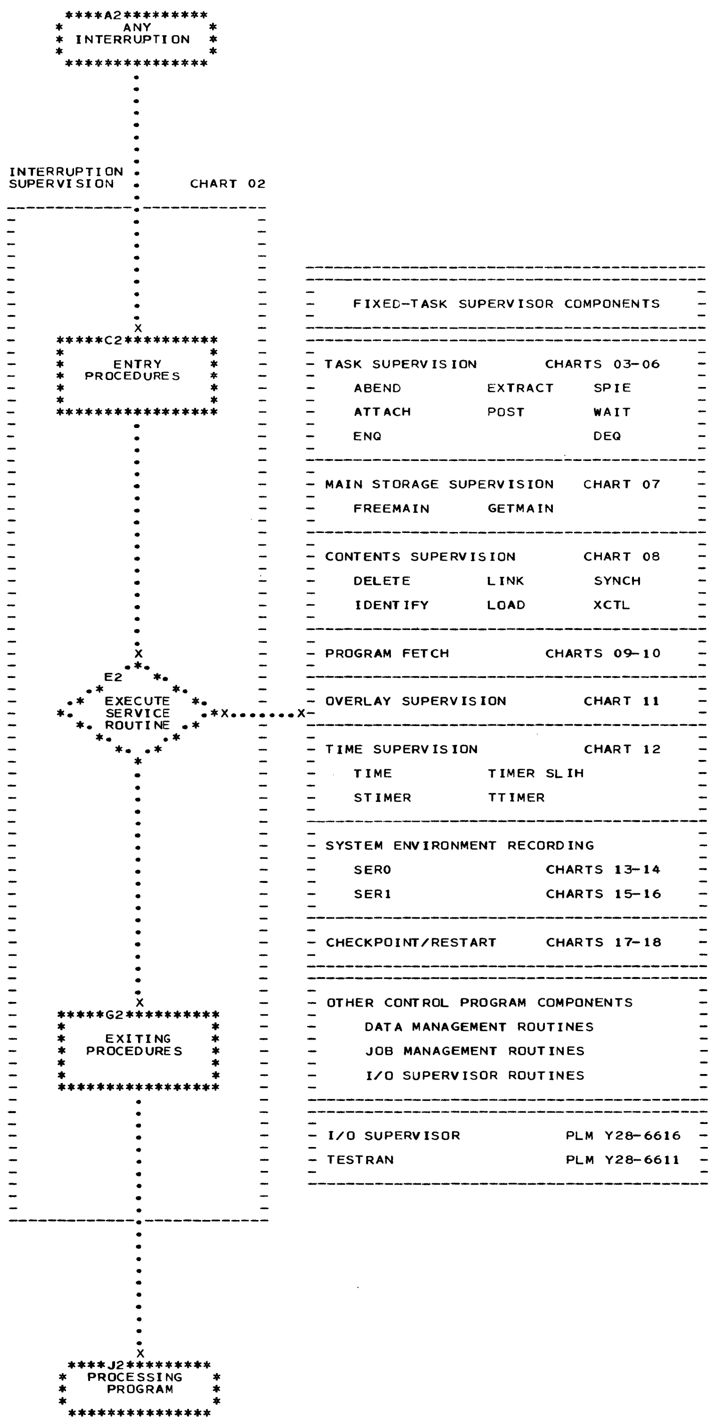
EXIT MODULE (IGC0V05B)

The RESTART Exit Module (IGC0V05B) first tests an error code field in the work area to determine if entry is for an error termination. If an error code is found, message IHJ007I is issued. The exit module then issues an ABEND macro instruction to abnormally terminate the task.

If no error has occurred, the exit module compares the sizes of the old Task Input/Output Table (TIOT) and current TIOT (which was saved in the RESTART work area). If the current TIOT is smaller or equal to the old TIOT, it overlays the old TIOT, and the RESTART work area is freed. If the current TIOT is larger, it is moved to the end of the work area, and both the remainder of the work area and the old TIOT are freed. The exit module writes message IHJ008I to inform the operator that the job is being restarted. It then loads a completion code of X'04' into Register 15 to inform the problem program that it is being restarted, and issues an SVC 3 instruction to pass control to the problem program.

If the program again abnormally terminates (and RESTART has not been deferred), RESTART will again be attempted.

• Chart 01. Fixed-Task Supervisor Control Flow
(Described in the introduction to this manual)



FIXED-TASK SUPERVISOR COMPONENTS		
TASK SUPERVISION		CHARTS 03-06
ABEND	EXTRACT	SPIE
ATTACH	POST	WAIT
ENQ		DEQ
MAIN STORAGE SUPERVISION		CHART 07
FREEMAIN	GETMAIN	
CONTENTS SUPERVISION		CHART 08
DELETE	LINK	SYNCH
IDENTIFY	LOAD	XCTL
PROGRAM FETCH		CHARTS 09-10
OVERLAY SUPERVISION		CHART 11
TIME SUPERVISION		CHART 12
TIME	TIMER SLIH	
STIMER	TTIMER	
SYSTEM ENVIRONMENT RECORDING		
SERO		CHARTS 13-14
SER1		CHARTS 15-16
CHECKPOINT/RESTART		CHARTS 17-18
OTHER CONTROL PROGRAM COMPONENTS		
DATA MANAGEMENT ROUTINES		
JOB MANAGEMENT ROUTINES		
I/O SUPERVISOR ROUTINES		
I/O SUPERVISOR		PLM Y28-6616
TESTRAN		PLM Y28-6611

INITIAL PROGRAM LOADER----------CHART 19
NUCLEUS INITIALIZATION PROGRAM--CHART 20

Chart 02. Interruption Supervision Control Flow
(Described in Chapter 1)

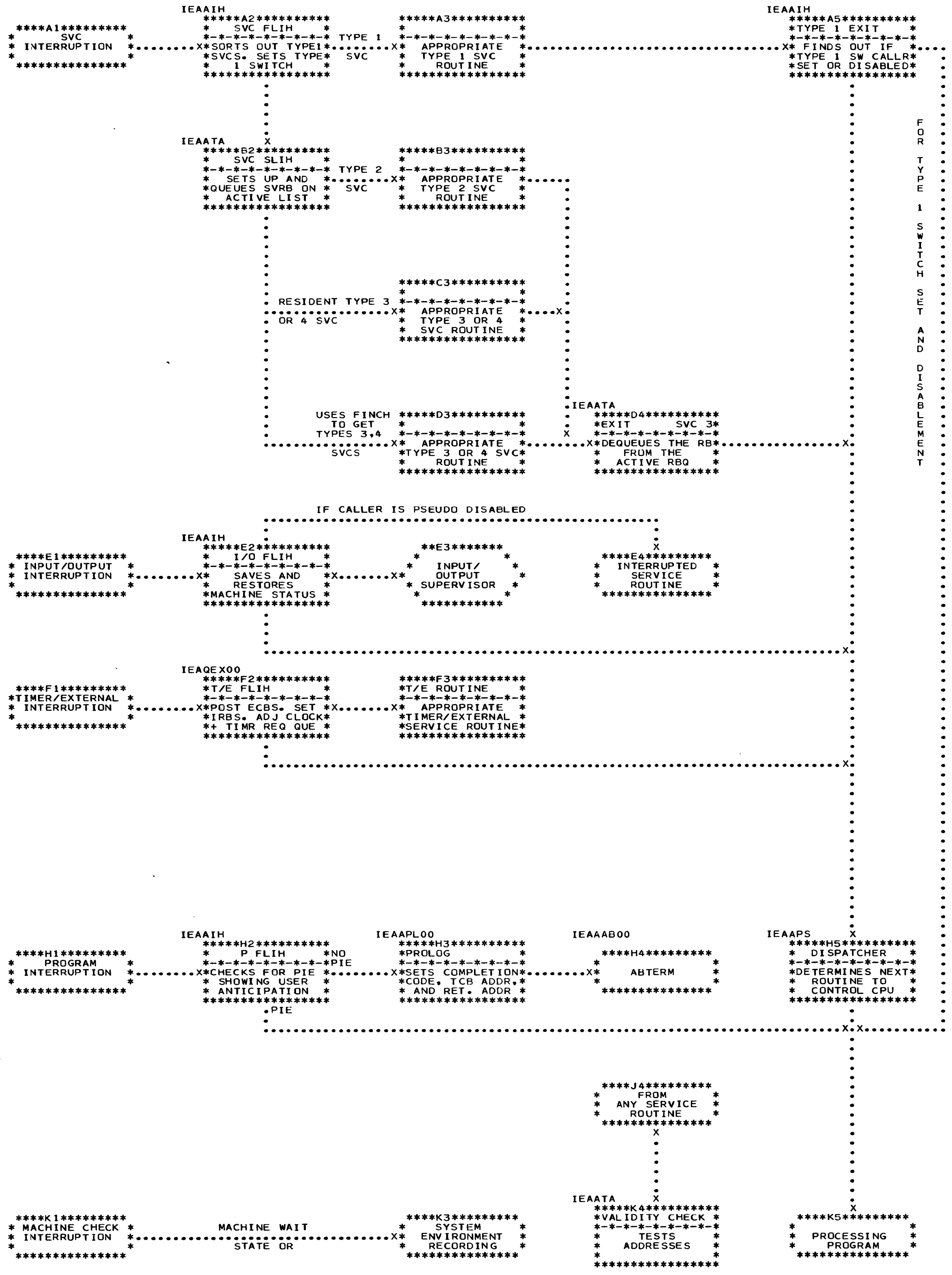
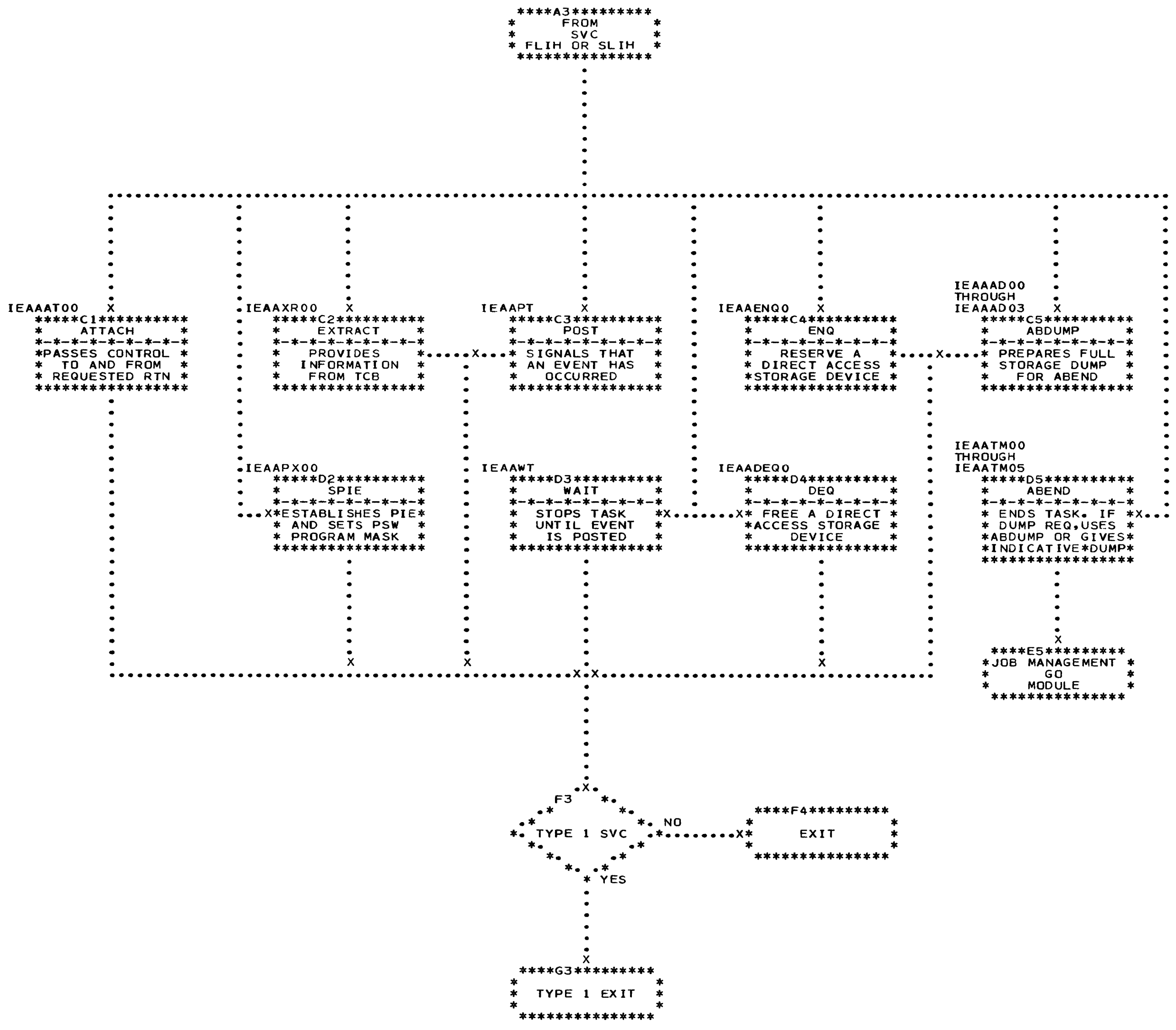
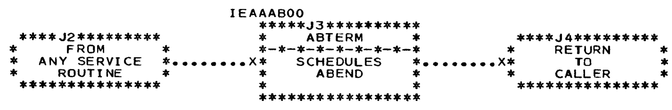

A1 SVC INTERRUPTION
IEAAIH A2 SVC FLIH SORTS OUT TYPE1 SVCS. SETS TYPE 1 SWITCH
TYPE 1 SVC
A3 APPROPRIATE TYPE 1 SVC ROUTINE
IEAAIH A5 TYPE 1 EXIT FINDS OUT IF TYPE 1 SW CALLR SET OR DISABLED
FOR TYPE 1 SWITCH SET AND DISABLEMENT
IEAATA B2 SVC SLIH SETS UP AND QUEUES SVRB ON ACTIVE LIST
TYPE 2 SVC
B3 APPROPRIATE TYPE 2 SVC ROUTINE
RESIDENT TYPE 3 OR 4 SVC
C3 APPROPRIATE TYPE 3 OR 4 SVC ROUTINE
USES FINCH TO GET TYPES 3,4 SVCS
D3 APPROPRIATE TYPE 3 OR 4 SVC ROUTINE
IEAATA D4 EXIT SVC 3 DEQUEUES THE RB FROM THE ACTIVE RBQ
IF CALLER IS PSEUDO DISABLED
E1 INPUT/OUTPUT INTERRUPTION
IEAAIH E2 I/O FLIH SAVES AND RESTORES MACHINE STATUS
E3 INPUT/ OUTPUT SUPERVISOR
E4 INTERRUPTED SERVICE ROUTINE
F1 TIMER/EXTERNAL INTERRUPTION
IEAQEX00 F2 T/E FLIH POST ECBS. SET IRBS. ADJ CLOCK + TIMR REQ QUE
F3 T/E ROUTINE APPROPRIATE TIMER/EXTERNAL SERVICE ROUTINE
H1 PROGRAM INTERRUPTION
IEAAIH H2 P FLIH CHECKS FOR PIE SHOWING USER ANTICIPATION
NO PIE
PIE
IEAAPL00 H3 PROLOG SETS COMPLETION CODE, TCB ADDR, AND RET. ADDR
IEAAAB00 H4 ABTERM
IEAAPS H5 DISPATCHER DETERMINES NEXT ROUTINE TO CONTROL CPU
J4 FROM ANY SERVICE ROUTINE
IEAATA K4 VALIDITY CHECK TESTS ADDRESSES
K1 MACHINE CHECK INTERRUPTION
MACHINE WAIT STATE OR
K3 SYSTEM ENVIRONMENT RECORDING
K5 PROCESSING PROGRAM

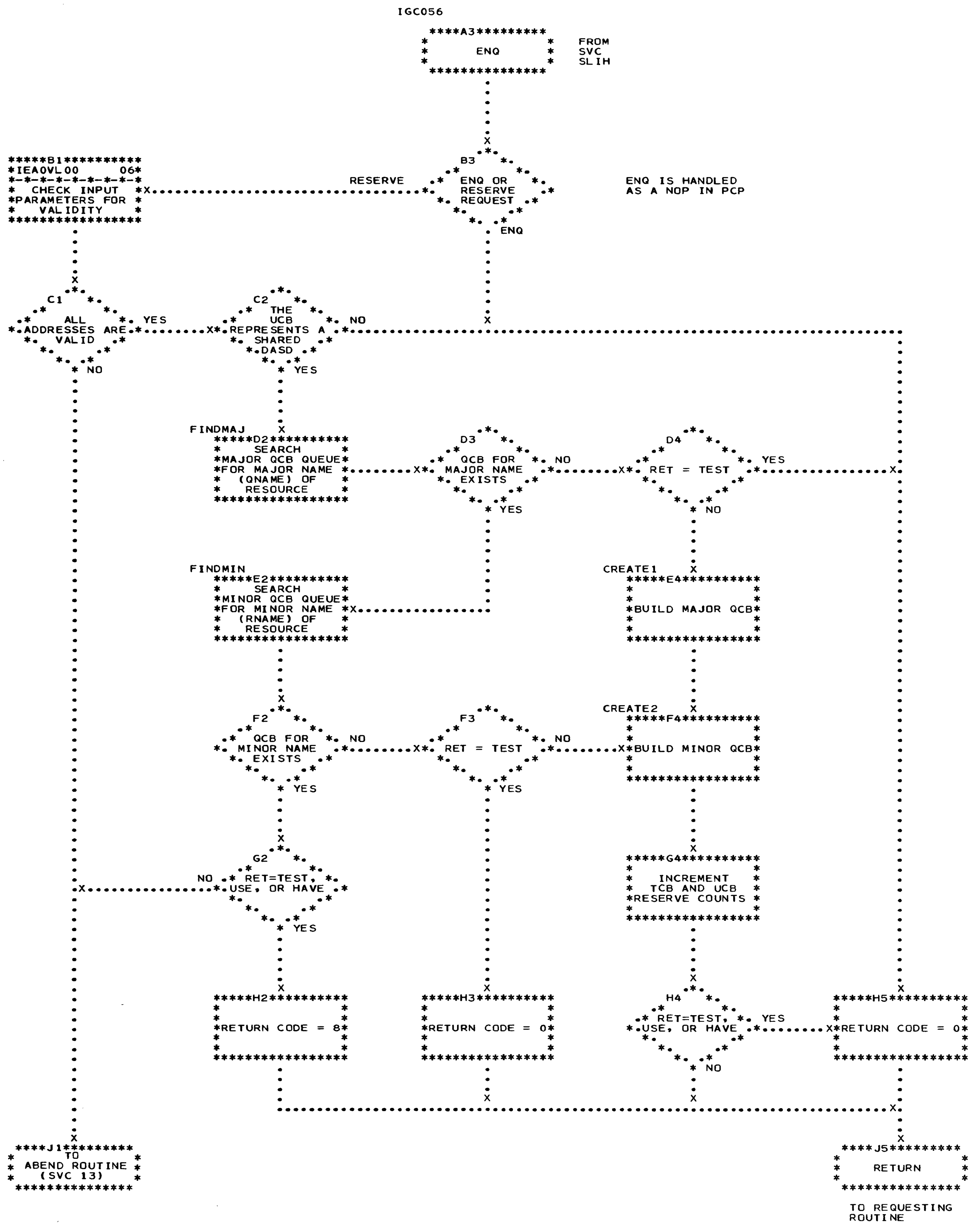
Chart 03. Task Supervision Control Flow
(Described in Chapter 2)


A3 FROM SVC FLIH OR SLIH
IEAAAT00 C1 ATTACH PASSES CONTROL TO AND FROM REQUESTED RTN
IEAAXR00 C2 EXTRACT PROVIDES INFORMATION FROM TCB
IEAAPT C3 POST SIGNALS THAT AN EVENT HAS OCCURRED
IEAAENQ0 C4 ENQ RESERVE A DIRECT ACCESS STORAGE DEVICE
IEAAAD00 THROUGH IEAAAD03 C5 ABDUMP PREPARES FULL STORAGE DUMP FOR ABEND
IEAAPX00 D2 SPIE ESTABLISHES PIE AND SETS PSW PROGRAM MASK
IEAAWT D3 WAIT STOPS TASK UNTIL EVENT IS POSTED
IEAADEQ0 D4 DEQ FREE A DIRECT ACCESS STORAGE DEVICE
IEAATM00 THROUGH IEAATM05 D5 ABEND ENDS TASK. IF DUMP REQ.USES ABDUMP OR GIVES INDICATIVE DUMP
E5 JOB MANAGEMENT GO MODULE
F3 TYPE 1 SVC
NO
YES
F4 EXIT
G3 TYPE 1 EXIT

SVC ENTRY AND EXIT PROCEDURES ARE SHOWN ON CHART 02

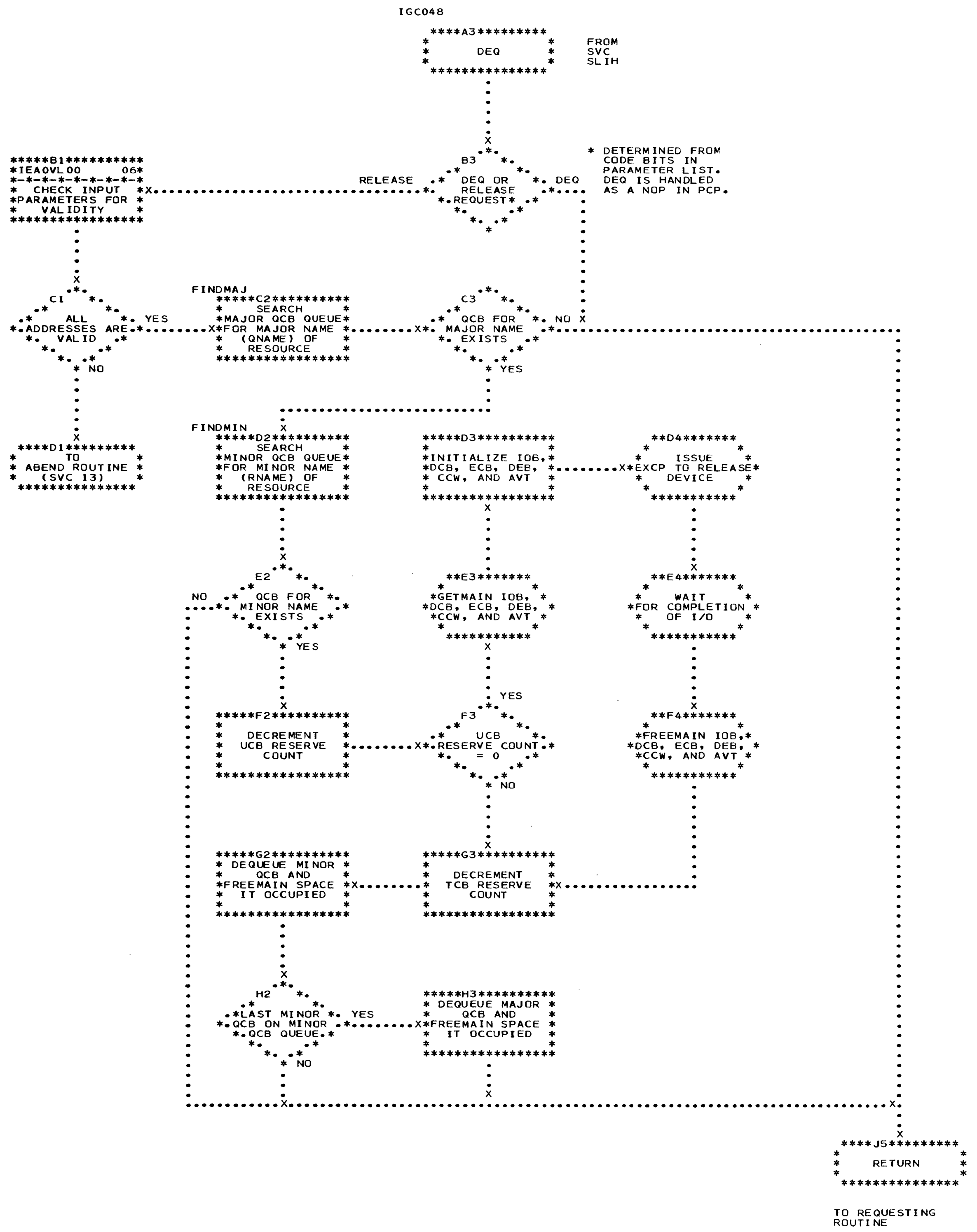

J2 FROM ANY SERVICE ROUTINE
IEAAAB00 J3 ABTERM SCHEDULES ABEND
J4 RETURN TO CALLER

• Chart 04. ENQ/RESERVE Service Routine (IEAAENQ0) (Described in Chapter 2)



IGC056
A3 ENQ
FROM SVC SLIH
B3 ENQ OR RESERVE REQUEST
RESERVE
ENQ
ENQ IS HANDLED AS A NOP IN PCP
B1 IEAOVL00 06 CHECK INPUT PARAMETERS FOR VALIDITY
C1 ALL ADDRESSES ARE VALID
YES
NO
C2 THE UCB REPRESENTS A SHARED DASD
NO
YES
FINDMAJ
D2 SEARCH MAJOR QCB QUEUE FOR MAJOR NAME (QNAME) OF RESOURCE
D3 QCB FOR MAJOR NAME EXISTS
NO
YES
D4 RET = TEST
YES
NO
FINDMIN
E2 SEARCH MINOR QCB QUEUE FOR MINOR NAME (RNAME) OF RESOURCE
CREATE1
E4 BUILD MAJOR QCB
F2 QCB FOR MINOR NAME EXISTS
NO
YES
F3 RET = TEST
NO
YES
CREATE2
F4 BUILD MINOR QCB
G2 RET=TEST, USE, OR HAVE
NO
YES
G4 INCREMENT TCB AND UCB RESERVE COUNTS
H2 RETURN CODE = 8
H3 RETURN CODE = 0
H4 RET=TEST, USE, OR HAVE
YES
NO
H5 RETURN CODE = 0
J1 TO ABEND ROUTINE (SVC 13)
J5 RETURN
TO REQUESTING ROUTINE

• Chart 05. DEQ Service Routine (IEAADEQ0)
(Described in Chapter 2)


IGC048
A3 DEQ
FROM SVC SLIH
B3 DEQ OR RELEASE REQUEST
RELEASE
DEQ
* DETERMINED FROM CODE BITS IN PARAMETER LIST. DEQ IS HANDLED AS A NOP IN PCP.
B1 IEAOVL00 06
CHECK INPUT PARAMETERS FOR VALIDITY
C1 ALL ADDRESSES ARE VALID
YES
NO
FINDMAJ
C2 SEARCH MAJOR QCB QUEUE FOR MAJOR NAME (QNAME) OF RESOURCE
C3 QCB FOR MAJOR NAME EXISTS
NO
YES
D1 TO ABEND ROUTINE (SVC 13)
FINDMIN
D2 SEARCH MINOR QCB QUEUE FOR MINOR NAME (RNAME) OF RESOURCE
D3 INITIALIZE IOB, DCB, ECB, DEB, CCW, AND AVT
D4 ISSUE EXCP TO RELEASE DEVICE
E2 QCB FOR MINOR NAME EXISTS
NO
YES
E3 GETMAIN IOB, DCB, ECB, DEB, CCW, AND AVT
E4 WAIT FOR COMPLETION OF I/O
F2 DECREMENT UCB RESERVE COUNT
F3 UCB RESERVE COUNT = 0
YES
NO
F4 FREEMAIN IOB, DCB, ECB, DEB, CCW, AND AVT
G2 DEQUEUE MINOR QCB AND FREEMAIN SPACE IT OCCUPIED
G3 DECREMENT TCB RESERVE COUNT
H2 LAST MINOR QCB ON MINOR QCB QUEUE
YES
NO
H3 DEQUEUE MAJOR QCB AND FREEMAIN SPACE IT OCCUPIED
J5 RETURN
TO REQUESTING ROUTINE

• Chart 06. Validity Check Subroutine (IEAOVL00) (Described in Chapter 2)

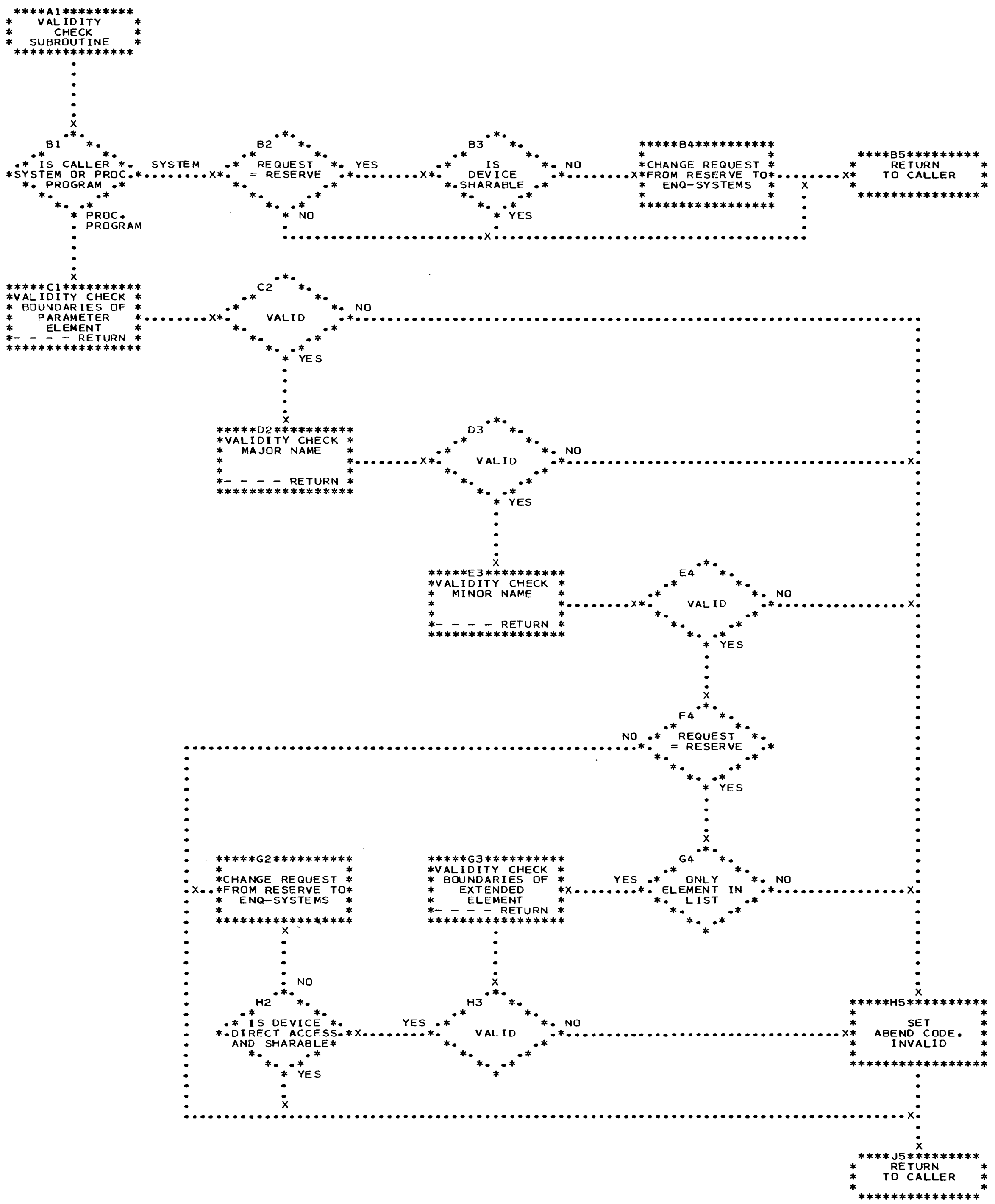
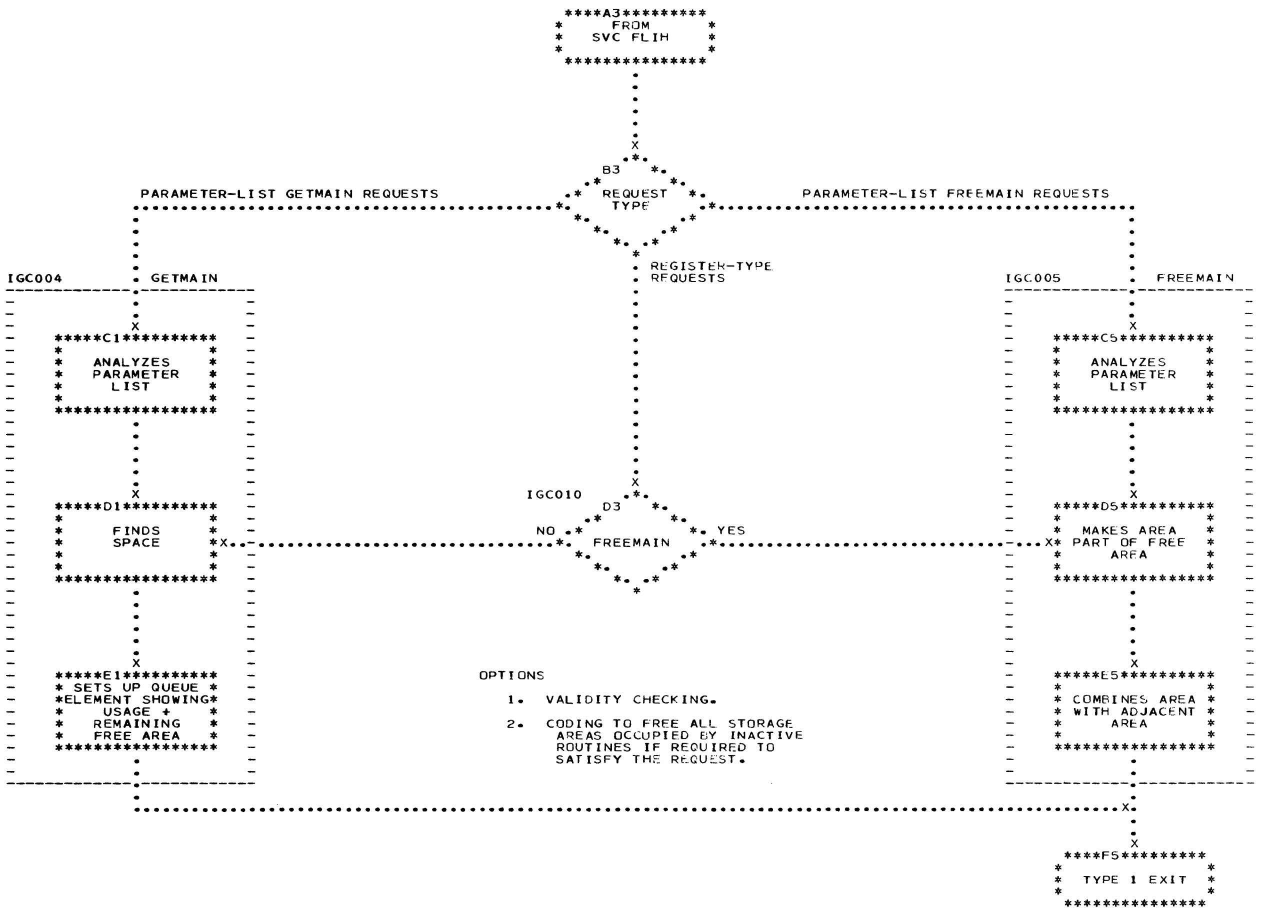

A1 VALIDITY CHECK SUBROUTINE
B1 IS CALLER SYSTEM OR PROC. PROGRAM
SYSTEM
PROC. PROGRAM
B2 REQUEST = RESERVE
YES
NO
B3 IS DEVICE SHARABLE
NO
YES
B4 CHANGE REQUEST FROM RESERVE TO ENQ-SYSTEMS
B5 RETURN TO CALLER
C1 VALIDITY CHECK BOUNDARIES OF PARAMETER ELEMENT RETURN
C2 VALID
NO
YES
D2 VALIDITY CHECK MAJOR NAME RETURN
D3 VALID
NO
YES
E3 VALIDITY CHECK MINOR NAME RETURN
E4 VALID
NO
YES
F4 REQUEST = RESERVE
NO
YES
G2 CHANGE REQUEST FROM RESERVE TO ENQ-SYSTEMS
G3 VALIDITY CHECK BOUNDARIES OF EXTENDED ELEMENT RETURN
G4 ONLY ELEMENT IN LIST
YES
NO
H2 IS DEVICE DIRECT ACCESS AND SHARABLE
NO
YES
H3 VALID
YES
NO
H5 SET ABEND CODE, INVALID
J5 RETURN TO CALLER

Chart 07. Main Storage Supervision Control Flow (Described in Chapter 3)

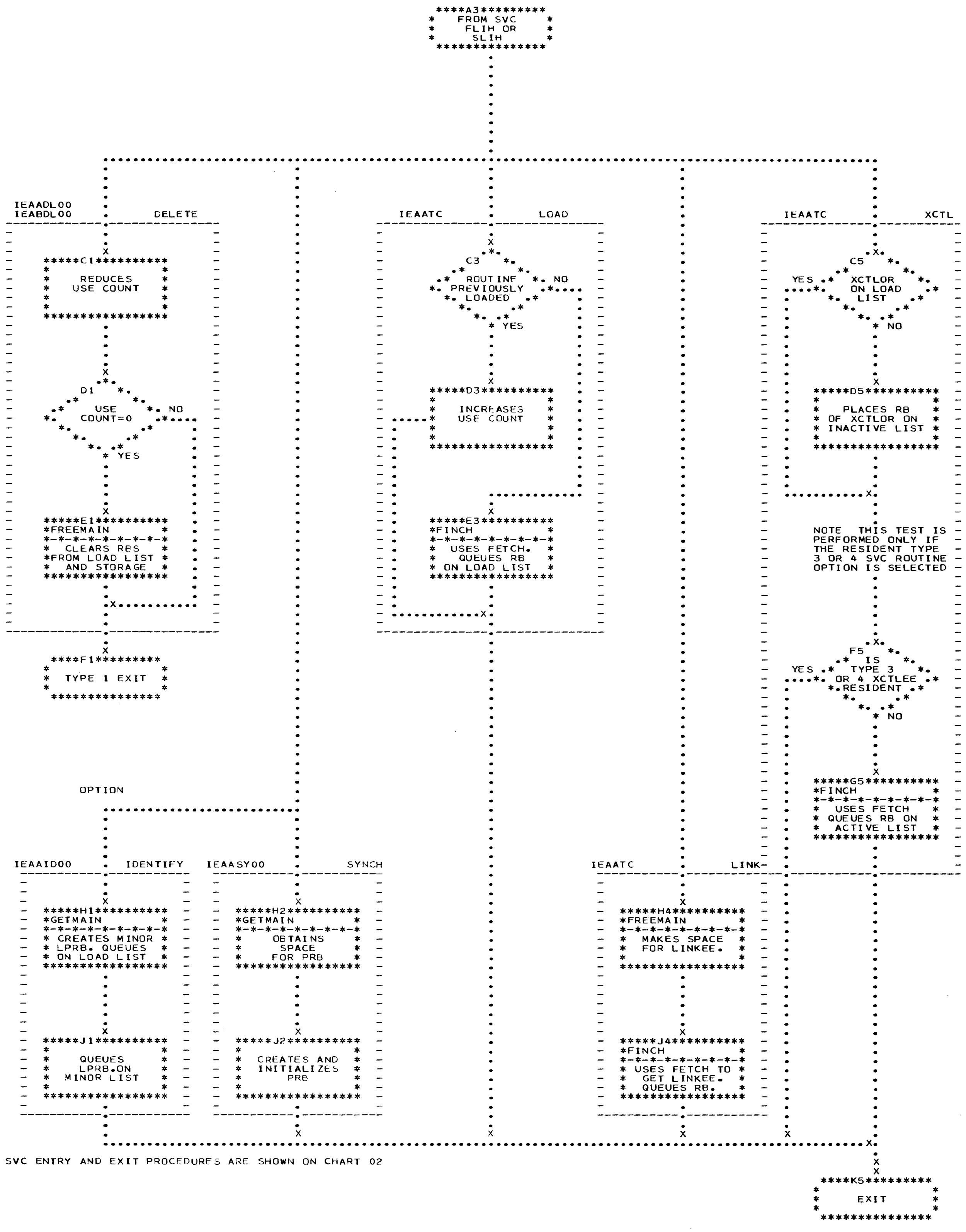
FOR MODULES IEAAMS00,IEABMS00,IEACMS00,IEADMS00



FROM SVC FLIH
REQUEST TYPE
PARAMETER-LIST GETMAIN REQUESTS
PARAMETER-LIST FREEMAIN REQUESTS
REGISTER-TYPE REQUESTS
IGC004 GETMAIN
ANALYZES PARAMETER LIST
FINDS SPACE
SETS UP QUEUE ELEMENT SHOWING USAGE + REMAINING FREE AREA
IGC010
FREEMAIN
NO
YES
IGC005 FREEMAIN
ANALYZES PARAMETER LIST
MAKES AREA PART OF FREE AREA
COMBINES AREA WITH ADJACENT AREA
OPTIONS
1. VALIDITY CHECKING.
2. CODING TO FREE ALL STORAGE AREAS OCCUPIED BY INACTIVE ROUTINES IF REQUIRED TO SATISFY THE REQUEST.
TYPE 1 EXIT

SVC ENTRY AND EXIT PROCEDURES ARE SHOWN ON CHART 02

Chart 08. Contents Supervision Control Flow (Described in Chapter 4)



A3 FROM SVC FLIH OR SLIH
IEAADL00 IEABDL00 DELETE
C1 REDUCES USE COUNT
D1 USE COUNT=0
NO
YES
E1 FREEMAIN CLEARS RBS FROM LOAD LIST AND STORAGE
F1 TYPE 1 EXIT
IEAATC LOAD
C3 ROUTINE PREVIOUSLY LOADED
NO
YES
D3 INCREASES USE COUNT
E3 FINCH USES FETCH. QUEUES RB ON LOAD LIST
IEAATC XCTL
C5 XCTLOR ON LOAD LIST
YES
NO
D5 PLACES RB OF XCTLOR ON INACTIVE LIST
NOTE THIS TEST IS PERFORMED ONLY IF THE RESIDENT TYPE 3 OR 4 SVC ROUTINE OPTION IS SELECTED
F5 IS TYPE 3 OR 4 XCTLEE RESIDENT
YES
NO
G5 FINCH USES FETCH QUEUES RB ON ACTIVE LIST
OPTION
IEAAID00 IDENTIFY
H1 GETMAIN CREATES MINOR LPRB. QUEUES ON LOAD LIST
J1 QUEUES LPRB.ON MINOR LIST
IEAASY00 SYNCH
H2 GETMAIN OBTAINS SPACE FOR PRB
J2 CREATES AND INITIALIZES PRB
IEAATC LINK
H4 FREEMAIN MAKES SPACE FOR LINKEE.
J4 FINCH USES FETCH TO GET LINKEE. QUEUES RB.
K5 EXIT

SVC ENTRY AND EXIT PROCEDURES ARE SHOWN ON CHART 02

• Chart 09. Program Fetch Control Flow
(Described in Chapter 5)

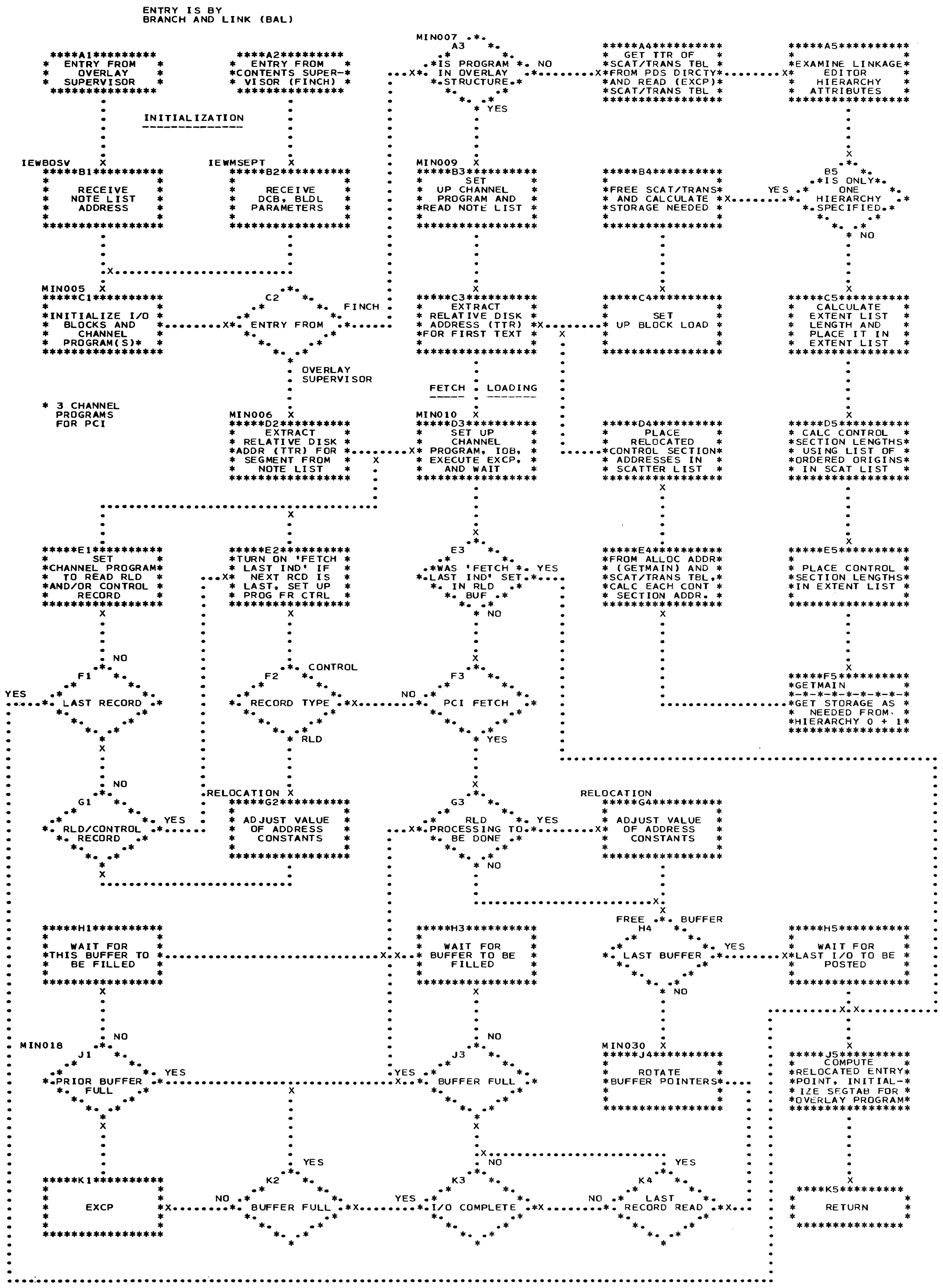

ENTRY IS BY BRANCH AND LINK (BAL)
A1 ENTRY FROM OVERLAY SUPERVISOR
A2 ENTRY FROM CONTENTS SUPER- VISOR (FINCH)
INITIALIZATION
MIN007 A3 IS PROGRAM IN OVERLAY STRUCTURE
NO
YES
A4 GET TTR OF SCAT/TRANS TBL FROM PDS DIRCTY AND READ (EXCP) SCAT/TRANS TBL
A5 EXAMINE LINKAGE EDITOR HIERARCHY ATTRIBUTES
IEWBOSV
B1 RECEIVE NOTE LIST ADDRESS
IEWMSEPT
B2 RECEIVE DCB, BLDL PARAMETERS
MIN009 B3 SET UP CHANNEL PROGRAM AND READ NOTE LIST
B4 FREE SCAT/TRANS AND CALCULATE STORAGE NEEDED
B5 IS ONLY ONE HIERARCHY SPECIFIED
YES
NO
MIN005 C1 INITIALIZE I/O BLOCKS AND CHANNEL PROGRAM(S)*
C2 ENTRY FROM
FINCH
OVERLAY SUPERVISOR
C3 EXTRACT RELATIVE DISK ADDRESS (TTR) FOR FIRST TEXT
C4 SET UP BLOCK LOAD
C5 CALCULATE EXTENT LIST LENGTH AND PLACE IT IN EXTENT LIST
FETCH LOADING
* 3 CHANNEL PROGRAMS FOR PCI
MIN006 D2 EXTRACT RELATIVE DISK ADDR (TTR) FOR SEGMENT FROM NOTE LIST
MIN010 D3 SET UP CHANNEL PROGRAM, IOB, EXECUTE EXCP, AND WAIT
D4 PLACE RELOCATED CONTROL SECTION ADDRESSES IN SCATTER LIST
D5 CALC CONTROL SECTION LENGTHS USING LIST OF ORDERED ORIGINS IN SCAT LIST
E1 SET CHANNEL PROGRAM TO READ RLD AND/OR CONTROL RECORD
E2 TURN ON 'FETCH LAST IND' IF NEXT RCD IS LAST, SET UP PROG FR CTRL
E3 WAS 'FETCH LAST IND' SET IN RLD BUF
YES
NO
E4 FROM ALLOC ADDR (GETMAIN) AND SCAT/TRANS TBL, CALC EACH CONT SECTION ADDR.
E5 PLACE CONTROL SECTION LENGTHS IN EXTENT LIST
F1 LAST RECORD
YES
NO
F2 RECORD TYPE
CONTROL
RLD
F3 PCI FETCH
NO
YES
F5 GETMAIN
GET STORAGE AS NEEDED FROM HIERARCHY 0 + 1
G1 RLD/CONTROL RECORD
NO
YES
RELOCATION G2 ADJUST VALUE OF ADDRESS CONSTANTS
G3 RLD PROCESSING TO BE DONE
YES
NO
RELOCATION G4 ADJUST VALUE OF ADDRESS CONSTANTS
H1 WAIT FOR THIS BUFFER TO BE FILLED
H3 WAIT FOR BUFFER TO BE FILLED
FREE H4 BUFFER LAST BUFFER
YES
NO
H5 WAIT FOR LAST I/O TO BE POSTED
MIN018 J1 PRIOR BUFFER FULL
NO
YES
J3 BUFFER FULL
NO
YES
MIN030 J4 ROTATE BUFFER POINTERS
J5 COMPUTE RELOCATED ENTRY POINT, INITIAL- IZE SEGTAB FOR OVERLAY PROGRAM
K1 EXCP
K2 BUFFER FULL
NO
YES
K3 I/O COMPLETE
YES
NO
K4 LAST RECORD READ
NO
YES
K5 RETURN

Chart 10. PCI and Channel End Appendages
(Described in Chapter 5)

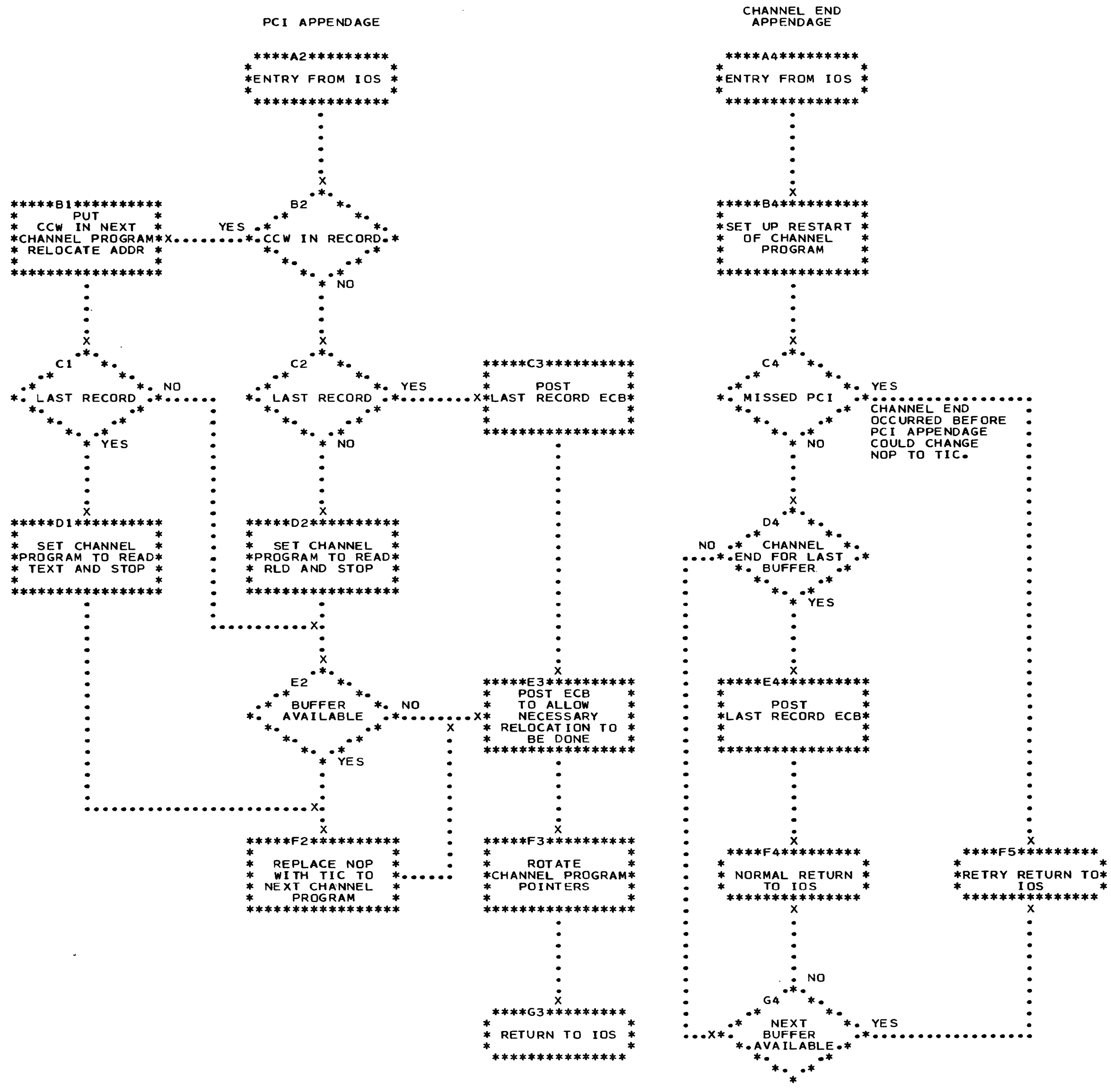
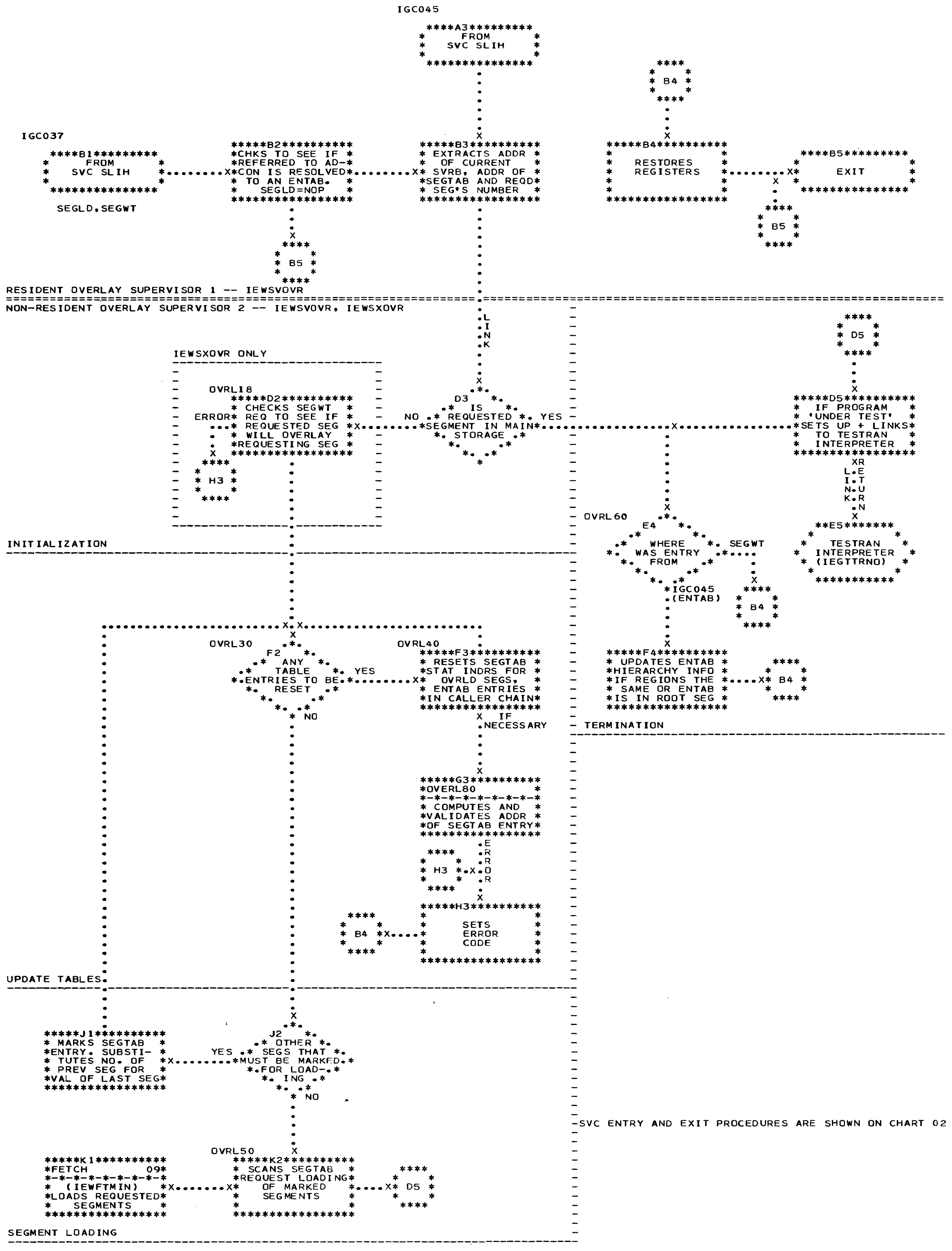
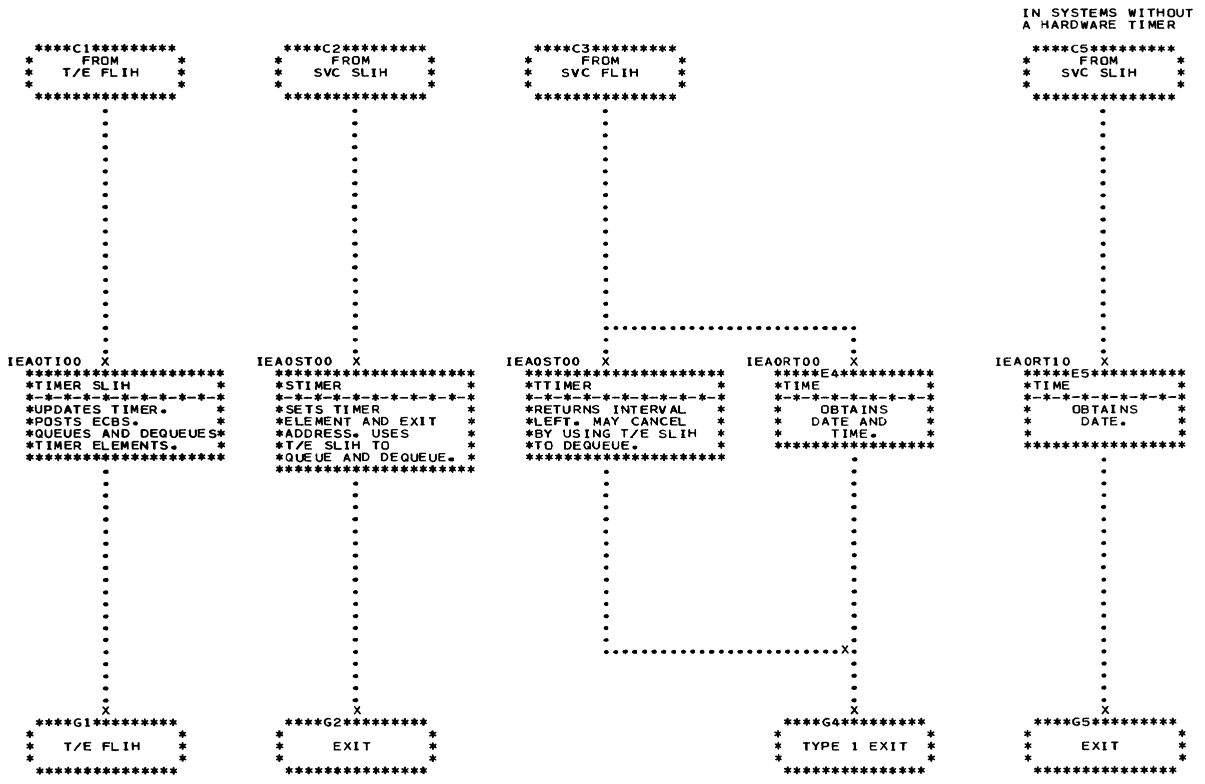

PCI APPENDAGE
CHANNEL END APPENDAGE
A2 ENTRY FROM IOS
B2 CCW IN RECORD
YES
B1 PUT CCW IN NEXT CHANNEL PROGRAM RELOCATE ADDR
NO
C1 LAST RECORD
NO
YES
C2 LAST RECORD
YES
NO
C3 POST LAST RECORD ECB
D1 SET CHANNEL PROGRAM TO READ TEXT AND STOP
D2 SET CHANNEL PROGRAM TO READ RLD AND STOP
E2 BUFFER AVAILABLE
NO
YES
E3 POST ECB TO ALLOW NECESSARY RELOCATION TO BE DONE
F2 REPLACE NOP WITH TIC TO NEXT CHANNEL PROGRAM
F3 ROTATE CHANNEL PROGRAM POINTERS
G3 RETURN TO IOS
A4 ENTRY FROM IOS
B4 SET UP RESTART OF CHANNEL PROGRAM
C4 MISSED PCI
YES
CHANNEL END OCCURRED BEFORE PCI APPENDAGE COULD CHANGE NOP TO TIC.
NO
D4 CHANNEL END FOR LAST BUFFER
NO
YES
E4 POST LAST RECORD ECB
F4 NORMAL RETURN TO IOS
F5 RETRY RETURN TO IOS
NO
G4 NEXT BUFFER AVAILABLE
YES

Chart 11. Overlay Supervision Control Flow
(Described in Chapter 6)



IGC045
A3 FROM SVC SLIH
IGC037
B1 FROM SVC SLIH
SEGLD, SEGWT
B2 CHKS TO SEE IF REFERRED TO ADCON IS RESOLVED TO AN ENTAB. SEGLD=NOP
B3 EXTRACTS ADDR OF CURRENT SVRB, ADDR OF SEGTAB AND REQD SEG'S NUMBER
B4 RESTORES REGISTERS
B5 EXIT
RESIDENT OVERLAY SUPERVISOR 1 -- IEWSVOVR
NON-RESIDENT OVERLAY SUPERVISOR 2 -- IEWSVOVR, IEWSXOVR
LINK
IEWSXOVR ONLY
OVRL18
D2 CHECKS SEGWT REQ TO SEE IF REQUESTED SEG WILL OVERLAY REQUESTING SEG
ERROR
H3
D3 IS SEGMENT IN MAIN STORAGE
NO
YES
D5 IF PROGRAM 'UNDER TEST' SETS UP + LINKS TO TESTRAN INTERPRETER
XR L.E I.T N.U K.R .N
E5 TESTRAN INTERPRETER (IEGTTRNO)
OVRL60
E4 WHERE WAS ENTRY FROM
SEGWT
IGC045 (ENTAB)
B4
INITIALIZATION
OVRL30
F2 ANY TABLE ENTRIES TO BE RESET
YES
NO
OVRL40
F3 RESETS SEGTAB STAT INDRS FOR OVRLD SEGS, ENTAB ENTRIES IN CALLER CHAIN
IF NECESSARY
F4 UPDATES ENTAB HIERARCHY INFO IF REGIONS THE SAME OR ENTAB IS IN ROOT SEG
TERMINATION
G3 OVERL80 COMPUTES AND VALIDATES ADDR OF SEGTAB ENTRY
ERROR
H3 SETS ERROR CODE
UPDATE TABLES
J1 MARKS SEGTAB ENTRY. SUBSTITUTES NO. OF PREV SEG FOR VAL OF LAST SEG
J2 OTHER SEGS THAT MUST BE MARKED FOR LOADING
YES
NO
SVC ENTRY AND EXIT PROCEDURES ARE SHOWN ON CHART 02
OVRL50
K1 FETCH 09 (IEWFTMIN) LOADS REQUESTED SEGMENTS
K2 SCANS SEGTAB REQUEST LOADING OF MARKED SEGMENTS
D5
SEGMENT LOADING

Chart 12. Time Supervision Control Flow
(Described in Chapter 7)



IN SYSTEMS WITHOUT A HARDWARE TIMER
C1 FROM T/E FLIH
C2 FROM SVC SLIH
C3 FROM SVC FLIH
C5 FROM SVC SLIH
IEA0TI00
TIMER SLIH
UPDATES TIMER. POSTS ECBS. QUEUES AND DEQUEUES TIMER ELEMENTS.
IEA0ST00
STIMER
SETS TIMER ELEMENT AND EXIT ADDRESS. USES T/E SLIH TO QUEUE AND DEQUEUE.
IEA0ST00
TTIMER
RETURNS INTERVAL LEFT. MAY CANCEL BY USING T/E SLIH TO DEQUEUE.
IEA0RT00
E4
TIME
OBTAINS DATE AND TIME.
IEA0RT10
E5
TIME
OBTAINS DATE.
G1 T/E FLIH
G2 EXIT
G4 TYPE 1 EXIT
G5 EXIT

SVC ENTRY AND EXIT PROCEDURES ARE SHOWN ON CHART 02

Chart 13. SERO Link Library Module Control Flow
(Described in Chapter 8)

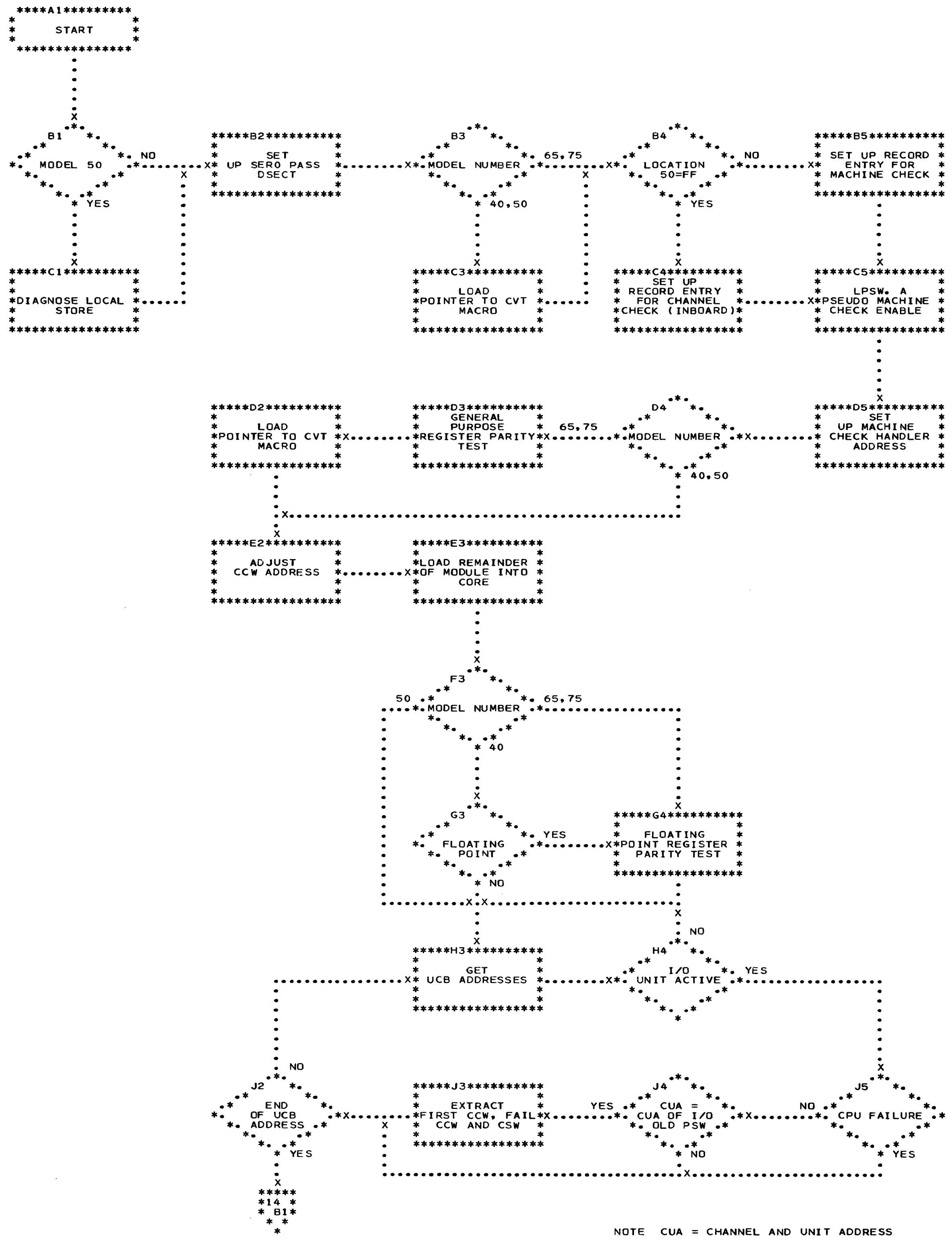

A1 START
B1 MODEL 50
NO
YES
C1 DIAGNOSE LOCAL STORE
B2 SET UP SERO PASS DSECT
B3 MODEL NUMBER
65,75
40,50
C3 LOAD POINTER TO CVT MACRO
B4 LOCATION 50=FF
NO
YES
C4 SET UP RECORD ENTRY FOR CHANNEL CHECK (INBOARD)
B5 SET UP RECORD ENTRY FOR MACHINE CHECK
C5 LPSW. A PSEUDO MACHINE CHECK ENABLE
D5 SET UP MACHINE CHECK HANDLER ADDRESS
D4 MODEL NUMBER
65,75
40,50
D3 GENERAL PURPOSE REGISTER PARITY TEST
D2 LOAD POINTER TO CVT MACRO
E2 ADJUST CCW ADDRESS
E3 LOAD REMAINDER OF MODULE INTO CORE
F3 MODEL NUMBER
50
65,75
40
G3 FLOATING POINT
YES
NO
G4 FLOATING POINT REGISTER PARITY TEST
H3 GET UCB ADDRESSES
H4 I/O UNIT ACTIVE
NO
YES
J2 END OF UCB ADDRESS
NO
YES
14 B1
J3 EXTRACT FIRST CCW, FAIL CCW AND CSW
J4 CUA = CUA OF I/O OLD PSW
YES
NO
J5 CPU FAILURE
NO
YES
NOTE CUA = CHANNEL AND UNIT ADDRESS

Chart 14. SER0 Link Library Module Control Flow (continued)

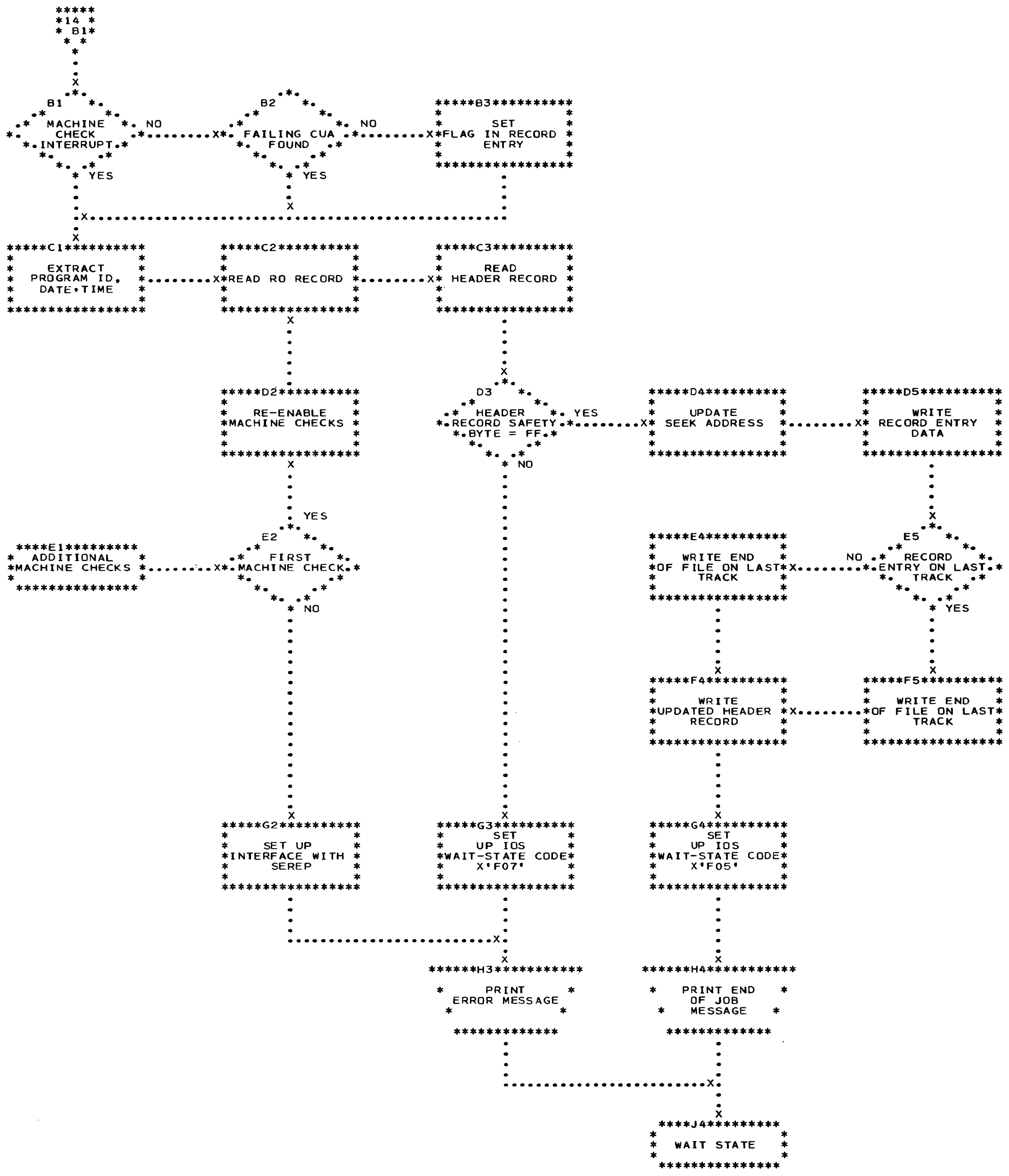
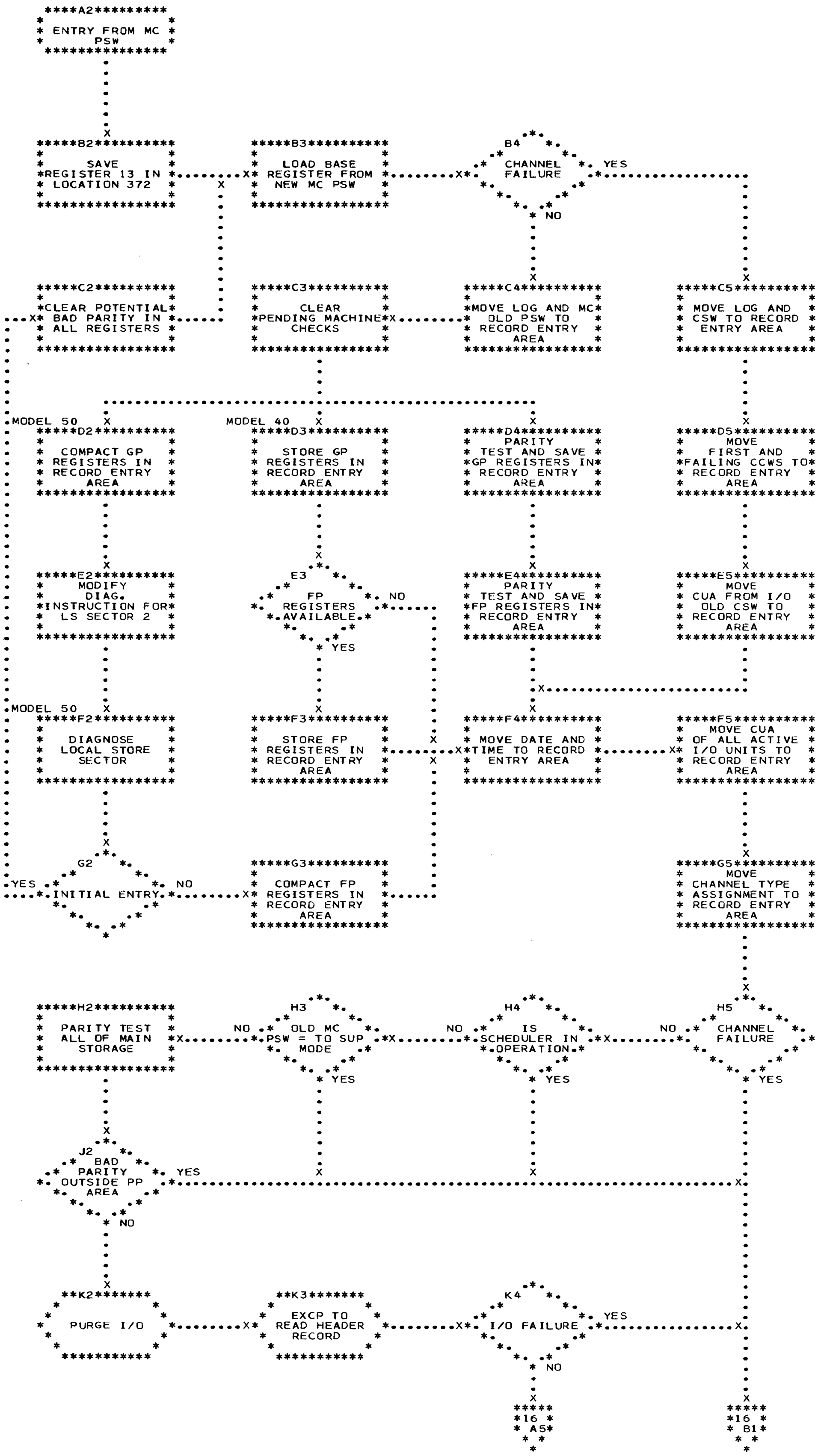
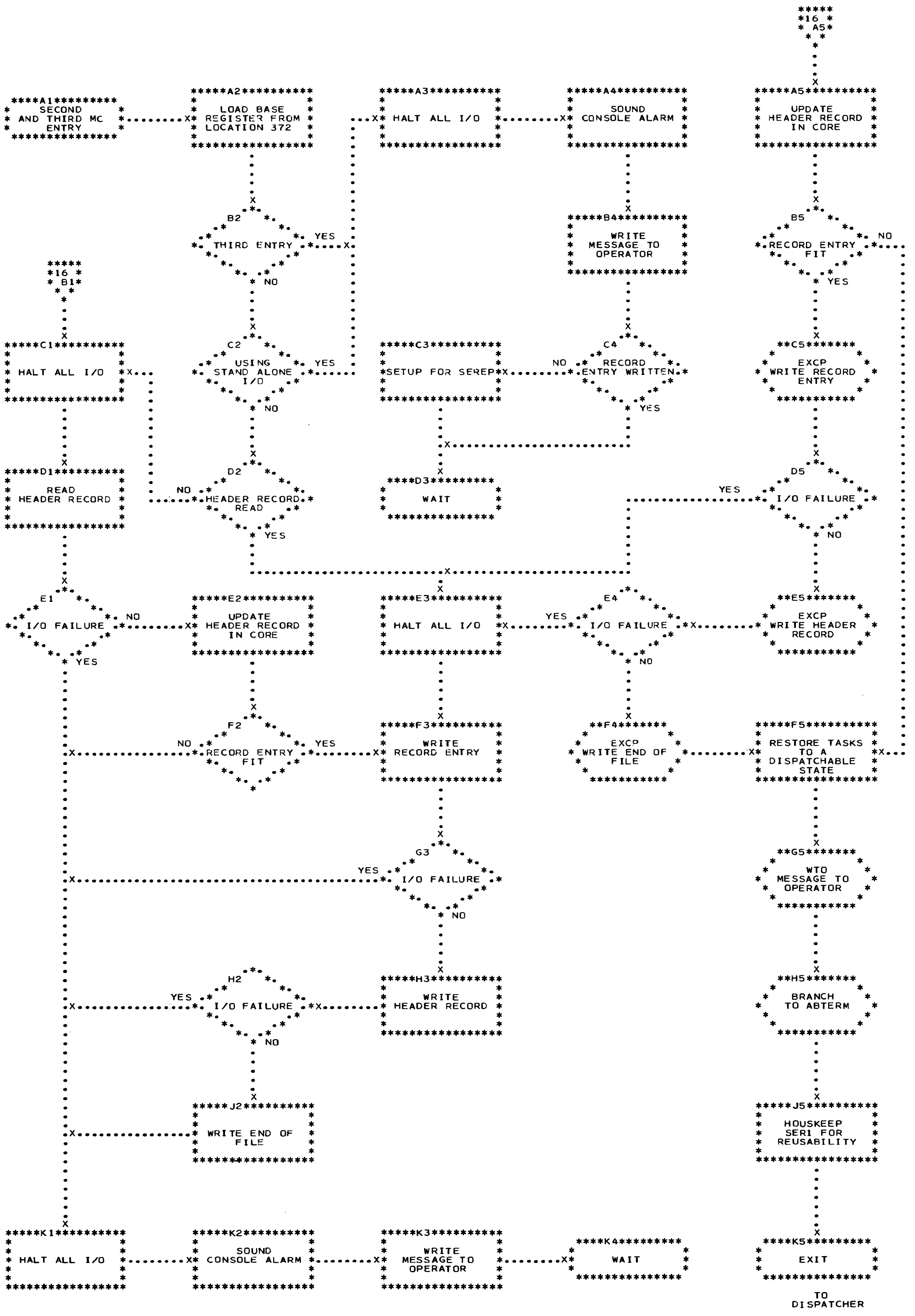

14 B1
B1 MACHINE CHECK INTERRUPT
NO
YES
B2 FAILING CUA FOUND
NO
YES
B3 SET FLAG IN RECORD ENTRY
C1 EXTRACT PROGRAM ID, DATE,TIME
C2 READ R0 RECORD
C3 READ HEADER RECORD
D2 RE-ENABLE MACHINE CHECKS
D3 HEADER RECORD SAFETY BYTE = FF
YES
NO
D4 UPDATE SEEK ADDRESS
D5 WRITE RECORD ENTRY DATA
E1 ADDITIONAL MACHINE CHECKS
E2 FIRST MACHINE CHECK
YES
NO
E4 WRITE END OF FILE ON LAST TRACK
E5 RECORD ENTRY ON LAST TRACK
NO
YES
F4 WRITE UPDATED HEADER RECORD
F5 WRITE END OF FILE ON LAST TRACK
G2 SET UP INTERFACE WITH SEREP
G3 SET UP IOS WAIT-STATE CODE X'F07'
G4 SET UP IOS WAIT-STATE CODE X'F05'
H3 PRINT ERROR MESSAGE
H4 PRINT END OF JOB MESSAGE
J4 WAIT STATE

Chart 15. SER1 Control Flow
(Described in Chapter 8)

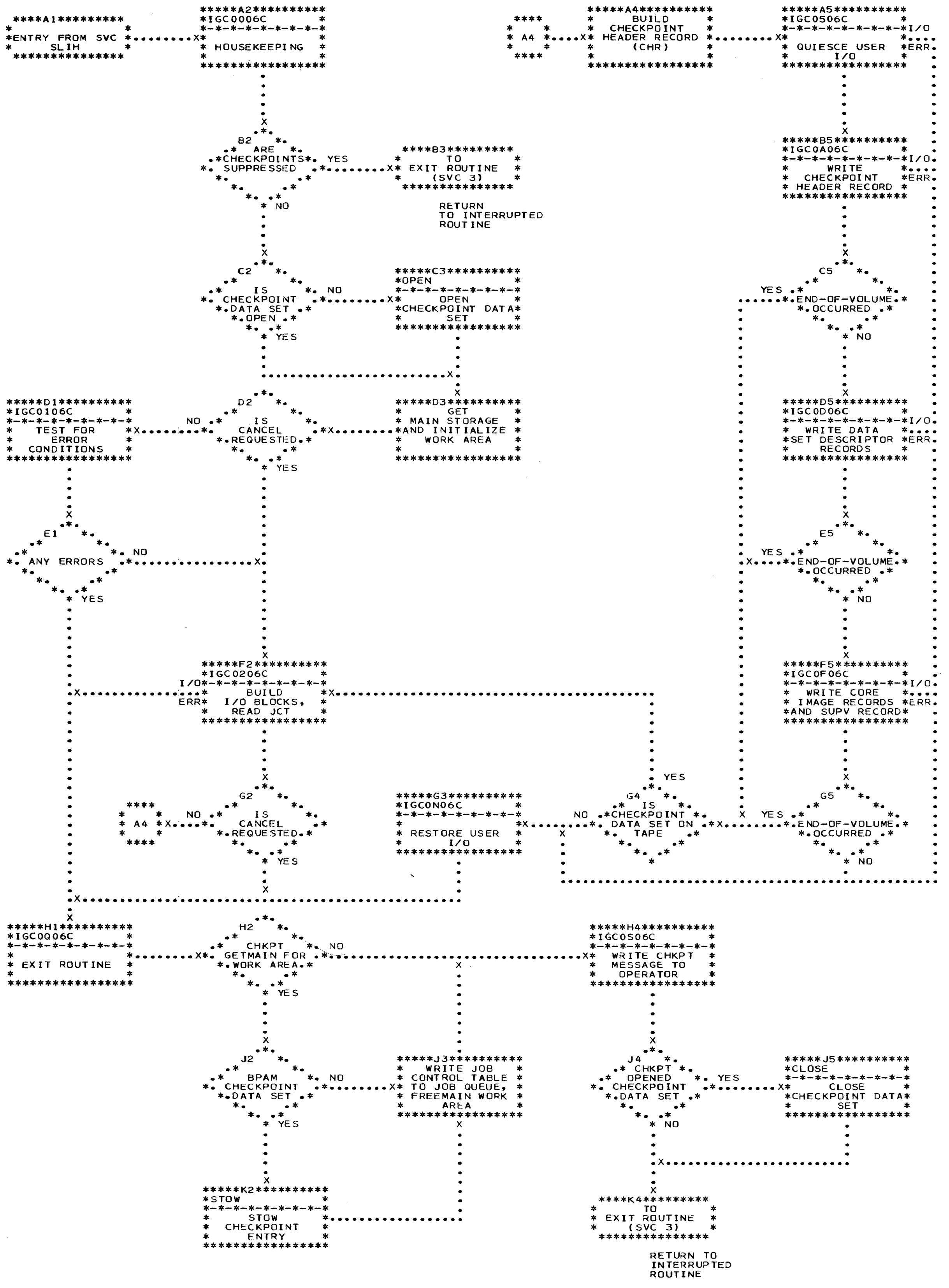


A2 ENTRY FROM MC PSW
B2 SAVE REGISTER 13 IN LOCATION 372
B3 LOAD BASE REGISTER FROM NEW MC PSW
B4 CHANNEL FAILURE
YES
NO
C2 CLEAR POTENTIAL BAD PARITY IN ALL REGISTERS
C3 CLEAR PENDING MACHINE CHECKS
C4 MOVE LOG AND MC OLD PSW TO RECORD ENTRY AREA
C5 MOVE LOG AND CSW TO RECORD ENTRY AREA
MODEL 50
MODEL 40
D2 COMPACT GP REGISTERS IN RECORD ENTRY AREA
D3 STORE GP REGISTERS IN RECORD ENTRY AREA
D4 PARITY TEST AND SAVE GP REGISTERS IN RECORD ENTRY AREA
D5 MOVE FIRST AND FAILING CCWS TO RECORD ENTRY AREA
E2 MODIFY DIAG. INSTRUCTION FOR LS SECTOR 2
E3 FP REGISTERS AVAILABLE
NO
YES
E4 PARITY TEST AND SAVE FP REGISTERS IN RECORD ENTRY AREA
E5 MOVE CUA FROM I/O OLD CSW TO RECORD ENTRY AREA
MODEL 50
F2 DIAGNOSE LOCAL STORE SECTOR
F3 STORE FP REGISTERS IN RECORD ENTRY AREA
F4 MOVE DATE AND TIME TO RECORD ENTRY AREA
F5 MOVE CUA OF ALL ACTIVE I/O UNITS TO RECORD ENTRY AREA
G2 INITIAL ENTRY
YES
NO
G3 COMPACT FP REGISTERS IN RECORD ENTRY AREA
G5 MOVE CHANNEL TYPE ASSIGNMENT TO RECORD ENTRY AREA
H2 PARITY TEST ALL OF MAIN STORAGE
H3 OLD MC PSW = TO SUP MODE
NO
YES
H4 IS SCHEDULER IN OPERATION
NO
YES
H5 CHANNEL FAILURE
NO
YES
J2 BAD PARITY OUTSIDE PP AREA
YES
NO
K2 PURGE I/O
K3 EXCP TO READ HEADER RECORD
K4 I/O FAILURE
YES
NO
16 A5
16 B1

Chart 16. SER1 Control Flow (continued)



• Chart 17. CHECKPOINT (SVC 63) Control Flow (Described in Chapter 9)


A1 ENTRY FROM SVC SLIH
A2 IGC0006C HOUSEKEEPING
A4
A4 BUILD CHECKPOINT HEADER RECORD (CHR)
A5 IGC0S06C QUIESCE USER I/O
I/O ERR.
B2 ARE CHECKPOINTS SUPPRESSED
YES
NO
B3 TO EXIT ROUTINE (SVC 3)
RETURN TO INTERRUPTED ROUTINE
B5 IGC0A06C WRITE CHECKPOINT HEADER RECORD
C2 IS CHECKPOINT DATA SET OPEN
C3 OPEN OPEN CHECKPOINT DATA SET
C5 END-OF-VOLUME OCCURRED
D1 IGC0106C TEST FOR ERROR CONDITIONS
D2 IS CANCEL REQUESTED
D3 GET MAIN STORAGE AND INITIALIZE WORK AREA
D5 IGC0D06C WRITE DATA SET DESCRIPTOR RECORDS
E1 ANY ERRORS
E5 END-OF-VOLUME OCCURRED
F2 IGC0206C BUILD I/O BLOCKS, READ JCT
F5 IGC0F06C WRITE CORE IMAGE RECORDS AND SUPV RECORD
G2 IS CANCEL REQUESTED
G3 IGC0N06C RESTORE USER I/O
G4 IS CHECKPOINT DATA SET ON TAPE
G5 END-OF-VOLUME OCCURRED
H1 IGC0Q06C EXIT ROUTINE
H2 CHKPT GETMAIN FOR WORK AREA
H4 IGC0S06C WRITE CHKPT MESSAGE TO OPERATOR
J2 BPAM CHECKPOINT DATA SET
J3 WRITE JOB CONTROL TABLE TO JOB QUEUE, FREEMAIN WORK AREA
J4 CHKPT OPENED CHECKPOINT DATA SET
J5 CLOSE CLOSE CHECKPOINT DATA SET
K2 STOW STOW CHECKPOINT ENTRY
K4 TO EXIT ROUTINE (SVC 3)
RETURN TO INTERRUPTED ROUTINE

• Chart 18. RESTART (SVC 52) Control Flow
(Described in Chapter 9)

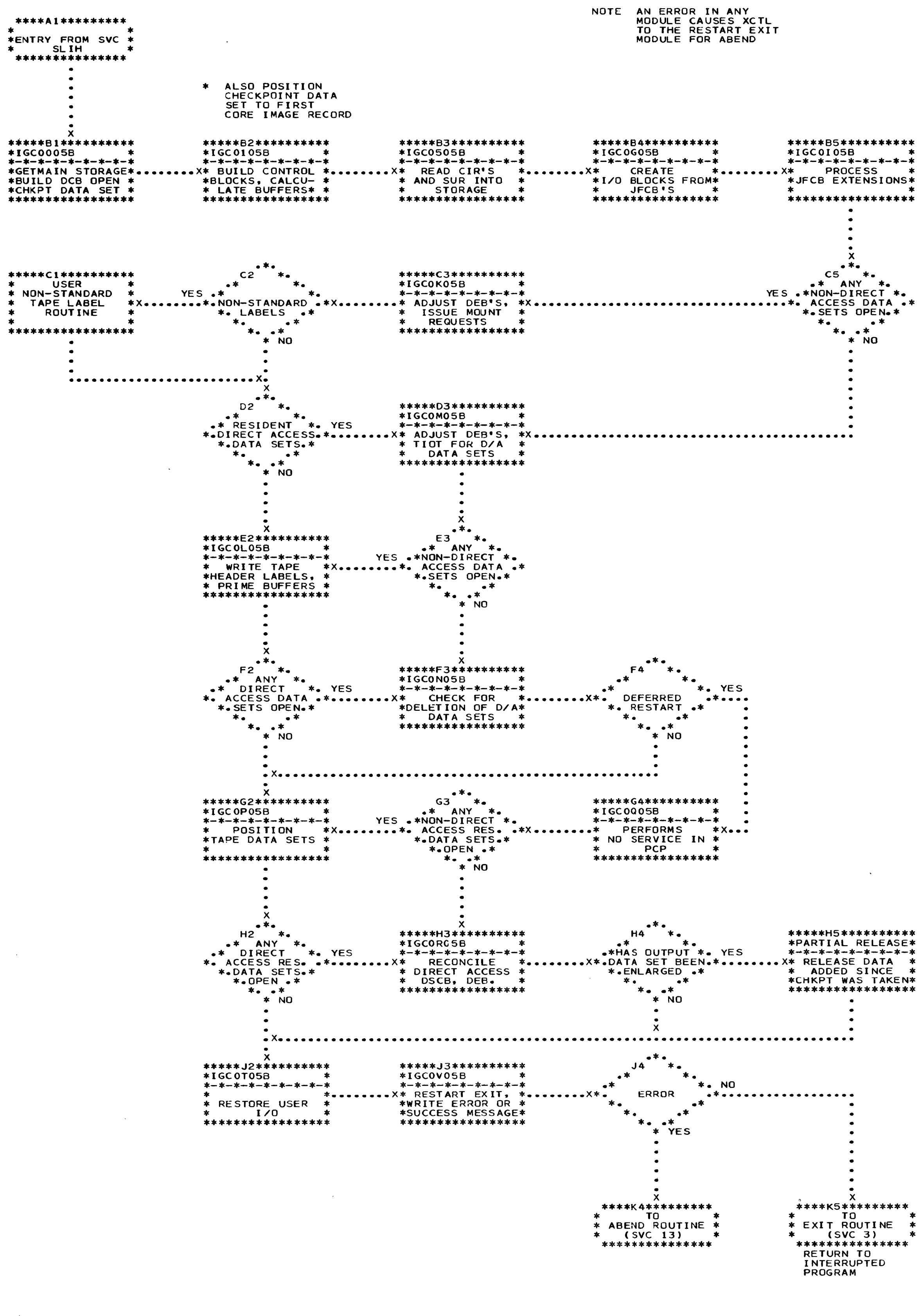
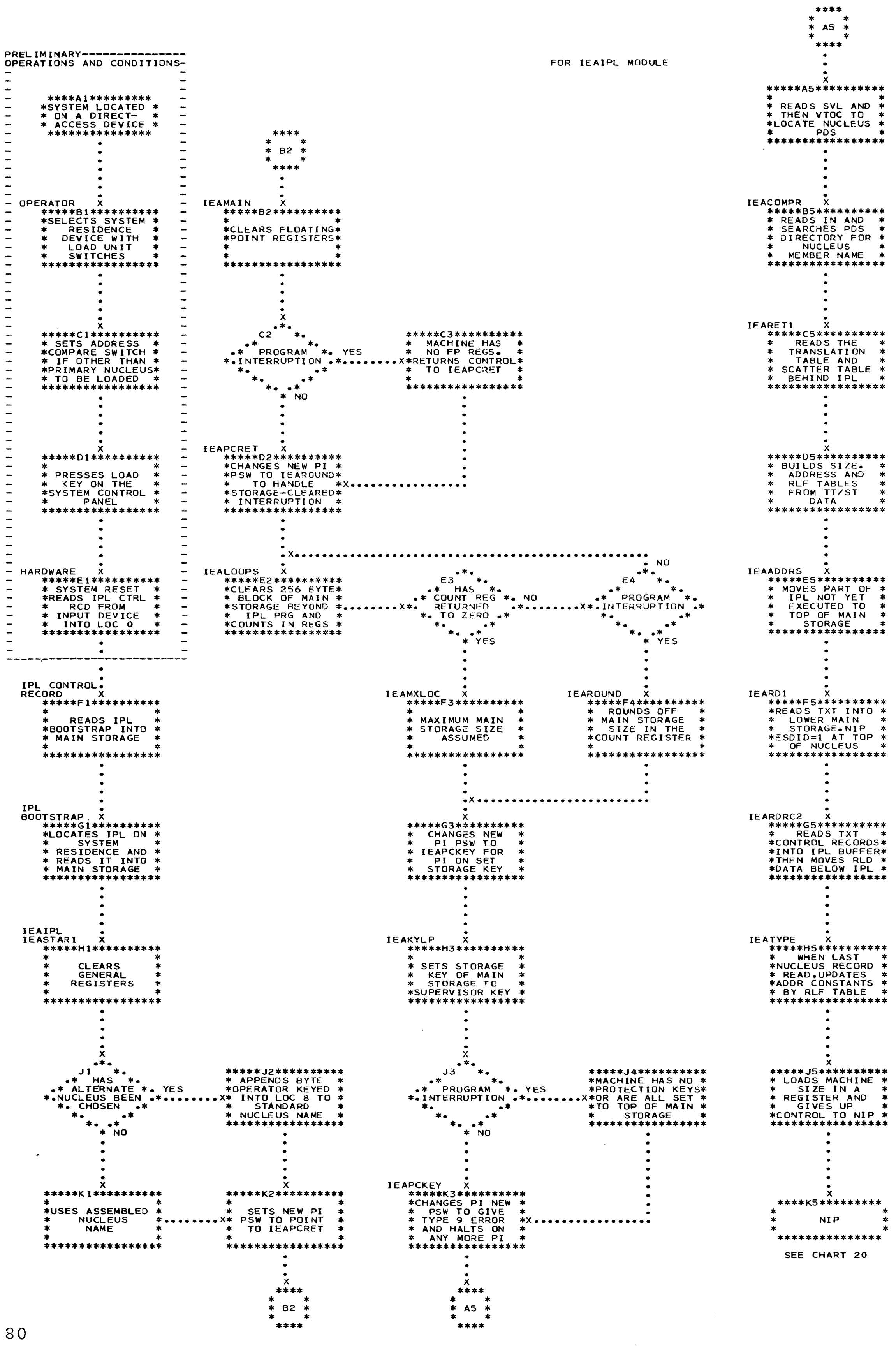
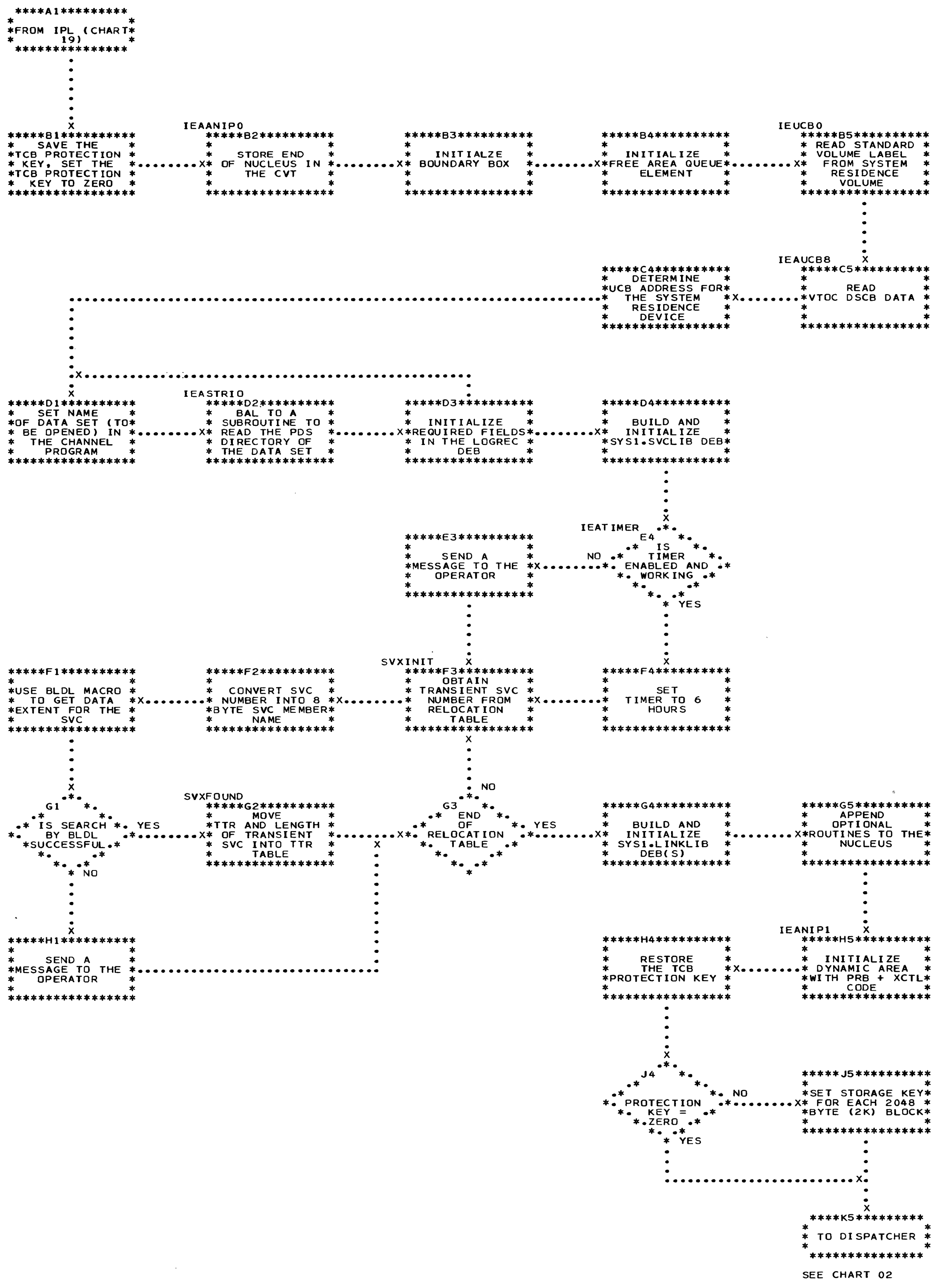

NOTE AN ERROR IN ANY MODULE CAUSES XCTL TO THE RESTART EXIT MODULE FOR ABEND
A1 ENTRY FROM SVC SLIH
* ALSO POSITION CHECKPOINT DATA SET TO FIRST CORE IMAGE RECORD
B1 IGC0005B GETMAIN STORAGE BUILD DCB OPEN CHKPT DATA SET
B2 IGC0105B BUILD CONTROL BLOCKS, CALCU- LATE BUFFERS*
B3 IGC0505B READ CIR'S AND SUR INTO STORAGE
B4 IGC0G05B CREATE I/O BLOCKS FROM JFCB'S
B5 IGC0I05B PROCESS JFCB EXTENSIONS
C1 USER NON-STANDARD TAPE LABEL ROUTINE
C2 NON-STANDARD LABELS
YES
NO
C3 IGC0K05B ADJUST DEB'S, ISSUE MOUNT REQUESTS
C5 ANY NON-DIRECT ACCESS DATA SETS OPEN
YES
NO
D2 RESIDENT DIRECT ACCESS DATA SETS
YES
NO
D3 IGC0M05B ADJUST DEB'S, TIOT FOR D/A DATA SETS
E2 IGC0L05B WRITE TAPE HEADER LABELS, PRIME BUFFERS
E3 ANY NON-DIRECT ACCESS DATA SETS OPEN
YES
NO
F2 ANY DIRECT ACCESS DATA SETS OPEN
YES
NO
F3 IGC0N05B CHECK FOR DELETION OF D/A DATA SETS
F4 DEFERRED RESTART
YES
NO
G2 IGC0P05B POSITION TAPE DATA SETS
G3 ANY NON-DIRECT ACCESS RES. DATA SETS OPEN
YES
NO
G4 IGC0Q05B PERFORMS NO SERVICE IN PCP
H2 ANY DIRECT ACCESS RES. DATA SETS OPEN
YES
NO
H3 IGC0R05B RECONCILE DIRECT ACCESS DSCB, DEB.
H4 HAS OUTPUT DATA SET BEEN ENLARGED
YES
NO
H5 PARTIAL RELEASE RELEASE DATA ADDED SINCE CHKPT WAS TAKEN
J2 IGC0T05B RESTORE USER I/O
J3 IGC0V05B RESTART EXIT, WRITE ERROR OR SUCCESS MESSAGE
J4 ERROR
NO
YES
K4 TO ABEND ROUTINE (SVC 13)
K5 TO EXIT ROUTINE (SVC 3)
RETURN TO INTERRUPTED PROGRAM

Chart 19. Initial Program Loader Control Flow (Described in Appendix A)


PRELIMINARY OPERATIONS AND CONDITIONS
FOR IEAIPL MODULE
A1 SYSTEM LOCATED ON A DIRECT-ACCESS DEVICE
OPERATOR
B1 SELECTS SYSTEM RESIDENCE DEVICE WITH LOAD UNIT SWITCHES
C1 SETS ADDRESS COMPARE SWITCH IF OTHER THAN PRIMARY NUCLEUS TO BE LOADED
D1 PRESSES LOAD KEY ON THE SYSTEM CONTROL PANEL
HARDWARE
E1 SYSTEM RESET READS IPL CTRL RCD FROM INPUT DEVICE INTO LOC 0
IPL CONTROL RECORD
F1 READS IPL BOOTSTRAP INTO MAIN STORAGE
IPL BOOTSTRAP
G1 LOCATES IPL ON SYSTEM RESIDENCE AND READS IT INTO MAIN STORAGE
IEAIPL IEASTAR1
H1 CLEARS GENERAL REGISTERS
J1 HAS ALTERNATE NUCLEUS BEEN CHOSEN
YES
NO
J2 APPENDS BYTE OPERATOR KEYED INTO LOC 8 TO STANDARD NUCLEUS NAME
K1 USES ASSEMBLED NUCLEUS NAME
K2 SETS NEW PI PSW TO POINT TO IEAPCRET
B2
IEAMAIN
B2 CLEARS FLOATING POINT REGISTERS
C2 PROGRAM INTERRUPTION
YES
NO
C3 MACHINE HAS NO FP REGS. RETURNS CONTROL TO IEAPCRET
IEAPCRET
D2 CHANGES NEW PI PSW TO IEAROUND TO HANDLE STORAGE-CLEARED INTERRUPTION
IEALOOPS
E2 CLEARS 256 BYTE BLOCK OF MAIN STORAGE BEYOND IPL PRG AND COUNTS IN REGS
E3 HAS COUNT REG RETURNED TO ZERO
NO
YES
E4 PROGRAM INTERRUPTION
NO
YES
IEAMXLOC
F3 MAXIMUM MAIN STORAGE SIZE ASSUMED
IEAROUND
F4 ROUNDS OFF MAIN STORAGE SIZE IN THE COUNT REGISTER
G3 CHANGES NEW PI PSW TO IEAPCKEY FOR PI ON SET STORAGE KEY
IEAKYLP
H3 SETS STORAGE KEY OF MAIN STORAGE TO SUPERVISOR KEY
J3 PROGRAM INTERRUPTION
YES
NO
J4 MACHINE HAS NO PROTECTION KEYS OR ARE ALL SET TO TOP OF MAIN STORAGE
IEAPCKEY
K3 CHANGES PI NEW PSW TO GIVE TYPE 9 ERROR AND HALTS ON ANY MORE PI
B2
A5
A5
A5 READS SVL AND THEN VTOC TO LOCATE NUCLEUS PDS
IEACOMPR
B5 READS IN AND SEARCHES PDS DIRECTORY FOR NUCLEUS MEMBER NAME
IEARET1
C5 READS THE TRANSLATION TABLE AND SCATTER TABLE BEHIND IPL
D5 BUILDS SIZE, ADDRESS AND RLF TABLES FROM TT/ST DATA
IEAADDRS
E5 MOVES PART OF IPL NOT YET EXECUTED TO TOP OF MAIN STORAGE
IEARD1
F5 READS TXT INTO LOWER MAIN STORAGE.NIP ESDID=1 AT TOP OF NUCLEUS
IEARDRC2
G5 READS TXT CONTROL RECORDS INTO IPL BUFFER THEN MOVES RLD DATA BELOW IPL
IEATYPE
H5 WHEN LAST NUCLEUS RECORD READ,UPDATES ADDR CONSTANTS BY RLF TABLE
J5 LOADS MACHINE SIZE IN A REGISTER AND GIVES UP CONTROL TO NIP
K5 NIP
SEE CHART 20

• Chart 20. Nucleus Initialization Program Control Flow (Described in Appendix B)


A1 FROM IPL (CHART 19)
B1 SAVE THE TCB PROTECTION KEY, SET THE TCB PROTECTION KEY TO ZERO
IEAANIP0
B2 STORE END OF NUCLEUS IN THE CVT
B3 INITIALZE BOUNDARY BOX
B4 INITIALIZE FREE AREA QUEUE ELEMENT
IEUCB0
B5 READ STANDARD VOLUME LABEL FROM SYSTEM RESIDENCE VOLUME
IEAUCB8
C5 READ VTOC DSCB DATA
C4 DETERMINE UCB ADDRESS FOR THE SYSTEM RESIDENCE DEVICE
D1 SET NAME OF DATA SET (TO BE OPENED) IN THE CHANNEL PROGRAM
IEASTRIO
D2 BAL TO A SUBROUTINE TO READ THE PDS DIRECTORY OF THE DATA SET
D3 INITIALIZE REQUIRED FIELDS IN THE LOGREC DEB
D4 BUILD AND INITIALIZE SYS1.SVCLIB DEB
IEATIMER
E4 IS TIMER ENABLED AND WORKING
NO
YES
E3 SEND A MESSAGE TO THE OPERATOR
F4 SET TIMER TO 6 HOURS
SVXINIT
F3 OBTAIN TRANSIENT SVC NUMBER FROM RELOCATION TABLE
F2 CONVERT SVC NUMBER INTO 8 BYTE SVC MEMBER NAME
F1 USE BLDL MACRO TO GET DATA EXTENT FOR THE SVC
G1 IS SEARCH BY BLDL SUCCESSFUL
YES
NO
SVXFOUND
G2 MOVE TTR AND LENGTH OF TRANSIENT SVC INTO TTR TABLE
G3 END OF RELOCATION TABLE
NO
YES
G4 BUILD AND INITIALIZE SYS1.LINKLIB DEB(S)
G5 APPEND OPTIONAL ROUTINES TO THE NUCLEUS
H1 SEND A MESSAGE TO THE OPERATOR
IEANIP1
H5 INITIALIZE DYNAMIC AREA WITH PRB + XCTL CODE
H4 RESTORE THE TCB PROTECTION KEY
J4 PROTECTION KEY = ZERO
NO
YES
J5 SET STORAGE KEY FOR EACH 2048 BYTE (2K) BLOCK
K5 TO DISPATCHER
SEE CHART 02

APPENDIX A: INITIAL PROGRAM LOADER (IPL)

The Initial Program Loader loads the nucleus and the Nucleus Initialization Program into main storage. The operator mounts the system residence volume on a direct access device and presses the LOAD key causing the hardware to read the IPL control record from cylinder 0, track 0 of the system residence volume into location 0 of main storage. The IPL control record loads the bootstrap record (a chain of CCWs) at an address greater than the size of IPL so that the bootstrap record will not be overlayed by IPL instructions. The bootstrap record then loads the IPL control section text (starting at location 0) and passes control through an LPSW instruction. For a more complete description see the publication IBM System/360 Operating System: Initial Program Loader and Nucleus Initialization Program, Form Y28-6661.

The IPL program prepares for loading the nucleus by:

- Clearing main storage and machine registers, and determining main storage size.
- Setting the storage key of main storage to the supervisor protection key, in systems with the protection feature.
- Determining the nucleus to be used.
- Finding the selected nucleus on the system residence volume.
- Assigning main storage addresses to the nucleus.
- Relocating the unexecuted portion of IPL.

When all preparations for nucleus loading are made, IPL:

- Loads the nucleus and the Nucleus Initialization Program.
- Establishes addressability among the nucleus control sections by resolving address constants.

When the Initial Program Loader processing is completed, IPL passes control to the Nucleus Initialization Program.

IPL ORGANIZATION

IPL is made up of two records and eight subroutines:

- IPL Control Record -- This 24-byte record, consisting of an IPL-PSW and two IPL-CCWs, is loaded into main storage at location zero by the hardware circuitry when the operator presses the LOAD key. This record and the IPL bootstrap record are located at track zero, cylinder zero of the system residence device; the IPL subroutines are contained in one record elsewhere on the system residence device.
- IPL Bootstrap Record -- This record, consisting of a chain of CCWs, is loaded into main storage at a location specified by the IPL control record. The IPL bootstrap record loads the IPL subroutines into main storage at location zero.
- Nucleus Selection (IEACOMPR) -- This subroutine selects the nucleus to be loaded.
- Hardware Initialization (IEAMAIN) -- This subroutine clears main storage, machine registers and, where applicable, initializes the storage keys.
- Nucleus Location (IEACOMLP) -- This subroutine locates the nucleus on the system residence device.
- Control Section Data Organization (IEAHOOP) -- This subroutine computes and sequentially arranges nucleus control section data so the nucleus can be loaded into main storage.
- IPL Relocation (IEAADDR) -- This subroutine moves the unexecuted part of IPL to the upper end of main storage to make room for the nucleus.
- Nucleus Load (IEALOAD) -- This subroutine loads the nucleus and NIP into main storage.
- RLD Relocation (IEARELOC) -- This subroutine relocates RLD items within the nucleus text read into main storage.
- Common I/O (IEASTRIO) -- This subroutine, used by IEACOMLP and IEALOAD, issues and tests for the successful completion of START I/O operations.

IPL CONTROL INFORMATION

NIP and the nucleus are combined into one load module and written on the system residence device by the linkage editor during system generation. IPL is supplied with the fixed name of this "nucleus" load module, but not with its location or the location of its DSCB within the VTOC.

The structure of the nucleus load module on the system residence device is the standard structure described in the publication IBM System/360 Operating System: Linkage Editor, Program Logic Manual. That is, its records and text are ordered as follows:

- Composite ESD Record (CESD).
- Scatter/Translation Record.
- Control Record.
- Text Record (TXT).
- Control/RLD Record (here and elsewhere, RLD data on this type of record depends on the presence of RLD items in the previous text).
- TXT.
- Control/RLD Record.
- TXT.
- and so on, until the end of the load module.

The scatter/translation record is made up of the translation table and the scatter table. The translation table corresponds, entry for entry, to the CESD, where each entry represents one control section (CSECT) made up of a control (or control/RLD) record and TXT. Entry 0 of both the translation table and the scatter table is a dummy entry containing zeros. Entry 1, corresponding to an ESDID of 1, represents NIP, which is the first CSECT of the nucleus load module. The translation table contains 2-byte pointers to the 4-byte entries in the scatter table.

IPL TABLES

Since the order of nucleus CSECTs on the system residence device is not fixed until the system is generated, IPL organizes the information available for the CSECTs before loading the text within CSECTs into main storage. IPL organizes the data by creating three tables:

- SIZTABLE -- a table of CSECT sizes.
- ADRTABLE -- a table of addresses where the CSECTs are to be loaded.
- RLFTABLE -- a table of relocation factors.

These tables are arranged in the same sequence as the CSECT entries in the scatter table and have 4-byte entries, making each table the same length as the scatter table.

To make up the SIZTABLE, IPL performs the following:

- Indexes the scatter table by the contents of the translation table entry to determine the address of the scatter table entry corresponding to a CSECT.
- Loads in a register the assembled origin "0" of the CSECT from the scatter table entry.
- Loads in another register the assembled origin "01" of the next CSECT from the consecutive entry in the scatter table.
- Computes the size of the CSECT by subtracting origin "0" from origin "01."
- Stores the size in SIZTABLE in a position relative to the CSECT position in the scatter table.

The size of the CSECT whose linkage-editor assigned origin is available in the last 4-byte entry of the scatter table is computed by subtracting origin "0" from the size of the nucleus which is available in the PDS directory and stored by IPL in the first word of the SIZTABLE which IPL builds behind the scatter table.

To make up the ADRTABLE, IPL performs the following:

- Stores the address where the second CSECT is to be loaded (assumed to be location 0) in the same position in the ADRTABLE as the CSECT occupies in the scatter table.
- Computes the address for the third CSECT by adding the size of the second CSECT to the address of the second CSECT.
- Stores the address for the third CSECT in the same position in the ADRTABLE as the CSECT occupies in the scatter table.
- Repeats the second and third steps above for each ordered CSECT. (Ordered CSECTs are those which must be loaded first and in the order in which they appear in the translation table.)

- Stores the addresses for non-ordered CSECTs, after computing them as they are encountered sequentially following the last of the ordered CSECTs.

The RLFTABLE is similar in structure to the SIZTABLE and ADRTABLE. Its entries are computed by subtracting the linkage-editor assigned origin from the address at which the CSECT is to be loaded.

IPL passes to NIP in registers:

- A pointer to SIZTABLE
- A pointer to ADRTABLE
- The number of entries in each table.

IPL CONTROL FLOW

As shown in Chart 19, IPL begins with several operator actions and prior conditions (see the publication IBM System/360 Operating System: Operator's Guide, Form C28-6540). The operator selects the system residence device with the LOAD-UNIT switches and presses the LOAD key. The hardware circuitry resets the CPU, locates track 0, cylinder 0, and loads the IPL control record into location 0. The control record loads the IPL bootstrap record, which, in turn, loads IPL and passes control to the first subroutine via an LPSW instruction. IPL is executed disabled for all interruptions except program interruptions.

IPL clears storage and registers, selects the nucleus or allows the operator to select a non-standard nucleus, sets storage keys where applicable, searches the VTOC and locates the data set containing the nucleus load module. IPL loads the translation table and the scatter table into main storage, relocates part of IPL (if necessary), calculates relocation constants, and loads the nucleus load module. IPL passes control to NIP by branching to an LPSW instruction in the nucleus.

NUCLEUS SELECTION

This subroutine (IEACOMPR) selects the nucleus for loading or allows the operator to choose a different nucleus, by using the ADDRESS-COMPARE switch and the DATA switch. The procedure for operator-selection of the nucleus is given in the publication IBM System/360 Operating System: Operator's Guide.

HARDWARE INITIALIZATION

This subroutine (IEAMAIN) sets correct parity in the:

- General registers.
- Floating point registers, if present.
- Main storage beyond IPL.

In addition, IEAMAIN sets storage keys to the supervisor protection key.

Program interruptions will occur while setting storage keys in machines without the protection feature, or while correcting parity in machines without floating point registers or without maximum main storage capacity. These interruptions are automatically handled by IEAMAIN. Further program interruptions are unexpected, and this subroutine places the machine in a wait state if they occur.

NUCLEUS LOCATION

This subroutine (IEACOMLP) searches for the location of the specified nucleus name on the system residence device and positions the read head of the system residence device at the first text record of the nucleus. IEACOMLP takes the following steps to locate the nucleus:

- Picks up the system residence device address stored at location 2 by the hardware circuitry.
- Reads the standard volume label to find the VTOC DSCB address.
- Reads the VTOC DSCB data to determine the number of tracks per cylinder on the system residence device.
- Searches the VTOC to find the DSCB for the partitioned data set (PDS) name.
- Seeks the track where the PDS directory starts.
- Searches the directory for a record containing the name of the nucleus, using the SEARCH EQUAL HIGH KEY command.
- Reads the PDS directory record.
- Determines the address of the scatter translation record on the system residence device from the PDS directory record.
- Finds the scatter translation record and reads it into main storage above IPL.

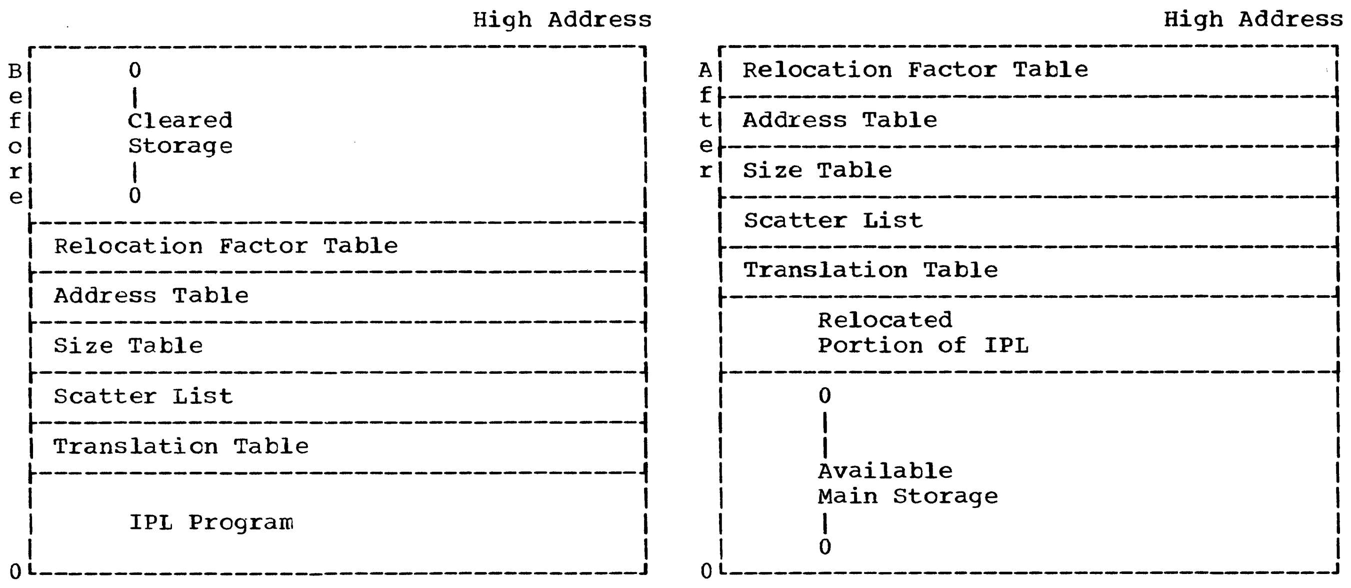


Figure 28. Storage Layout Before and After IPL Relocation

The nucleus location subroutine uses the common I/O subroutine, IEASTRIO, when reading the standard volume label, VTOC, etc., from the system residence device into main storage. Before using the common I/O subroutine, IEACOMLP sets up a channel program with an appropriate chain of CCWs to SEEK, SEARCH, TIC and READ.

CONTROL SECTION DATA ORGANIZATION

This subroutine (IEAHOOP) computes the address for loading the ordered CSECTs and also computes the relocation factor and size of each CSECT. This data is arranged in tables -- SIZTABLE, ADRTABLE, and RLFTABLE -- for use by the nucleus load subroutine. The tables and the procedures IEAHOOP uses to make them are described under the earlier heading, "IPL Tables."

IPL RELOCATION

This subroutine (IEAADDR) relocates the unexecuted portion of IPL and its tables into the numerically high end of main storage so that IPL can load the nucleus text starting at location zero. After the relocation is complete, it moves zeros into the storage it occupied before relocation (see Figure 28).

NUCLEUS LOAD

This subroutine (IEALOAD) loads the nucleus into main storage, placing the relocatable modules into main storage in the order of their position in the translation table. IEAANIP0, the Nucleus Initialization Program, must be the first CSECT; it is loaded into the upper end of main storage just below the relocated portion of IPL. IEAAIH00 must be the second CSECT and is loaded into location zero. IEAAIH00 includes permanent storage assignments, the task control block, first level interruption handlers, the type 1 SVC exit routine, the dispatcher, the exit effector, and the input/output supervisor. Unless INSERT cards are used for each nucleus CSECT prepared by linkage editor, the order of the loading of the remaining relocatable nucleus CSECTS will vary. IPL sets a buffer of 256 bytes in IPL for reading control/RLD records, and performs the following actions:

- Reads a control/RLD record into the buffer and interrogates the record.
- Picks up from the control/RLD record the ESDID of the text record that follows the control/RLD record.
- Determines the address, L, at which the text record of the CSECT is to be read, by adding the relocation factor from the RLFTABLE to the assigned origin of the record.
- Reads the TXT record of the CSECT at address L.
- Adds the number of text bytes read, T, to address, L, to compute the address where the next text record of the same CSECT is to be read. Sets L = L + T.

- Reads into the buffer the control/RLD record following the text record.

- Builds a table of RLDs by moving RLD information bytes from the control/RLD record and keeps a count of the RLD bytes moved into the RLD table above NIP.

- Repeats the above steps until all the records of the nucleus are read into main storage.

The nucleus load subroutine uses the common I/O subroutine when reading the CCW, control/RLD and TXT records of the nucleus load module from the system residence device into main storage. Before using the common I/O subroutine, IEALOAD sets up a channel program with an appropriate CCW to READ the particular record.

RLD RELOCATION

This subroutine (IEARELOC) scans the RLD table created by IEALOAD and relocates the load constants in the nucleus text, using relocation factors stored by IPL in the RLFTABLE. At the completion of IEARELOC, IPL's work is done and control is passed to NIP. Figure 29 shows the layout of main storage when IPL passes control to NIP.

COMMON I/O

This subroutine (IEASTRIO), used by nucleus locate and nucleus load, issues and tests for the successful completion of START I/O operations. Nucleus locate and nucleus load set up the CAW and CCWs and then branch and link to IEASTRIO. After execution of IEASTRIO, control is returned to the IPL subroutine that branched to it.

Error conditions encountered during the execution of IEASTRIO are indicated to the operator by the WAIT light, and the error type is stored in the address field of the WAIT PSW.

The operator can retry IPL when the WAIT light is on. If IPL is unsuccessful after a few trials, the operator displays the address field of the PSW to determine the error type, and informs the customer engineer. The ten error types are shown in Figure 30.

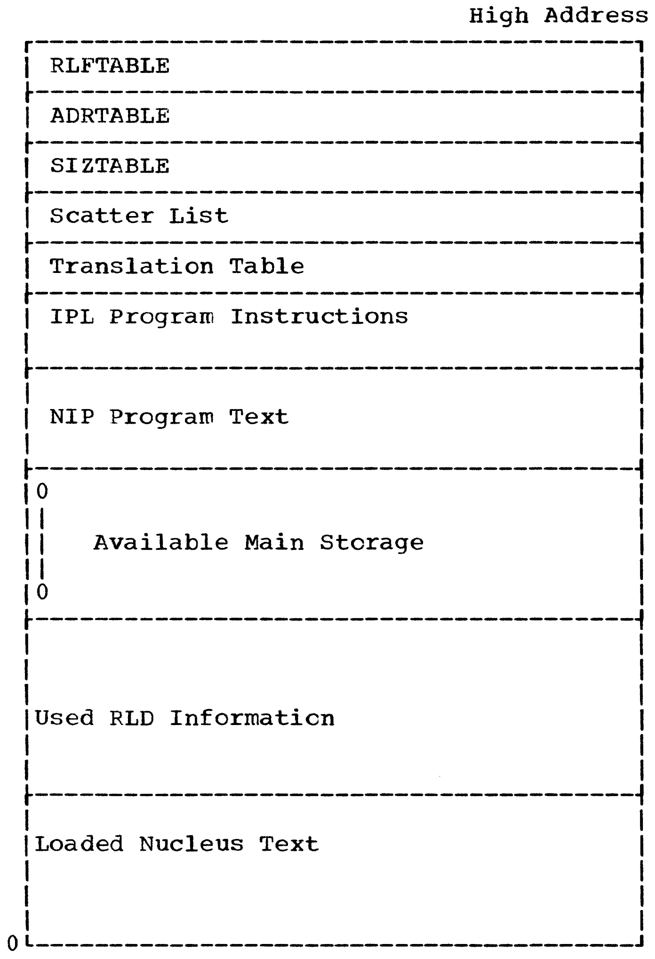


Figure 29. Storage Layout at End of IPL Program Execution

Error Type	Bit Pattern Displayed	Meaning
1	00000001	I/O is not operational.
2	00000010	I/O operation is not initiated. CSW is stored. Unit is not busy.
3	00000011	I/O operation is not initiated. CSW is not stored. Channel is not busy.
4	00000100	During TEST I/O. Channel is not busy. CSW is not stored.
5	00000101	During TEST I/O. Unit check condition is indicated. Location X'4C' contains the address of the ccw causing the original unit check, and X'54' contains the first four sense bytes.
6	00000110	During TEST I/O. Any of these conditions are indicated: Interface control check. Channel control check. Channel Data check. Channel chaining check. Program Check.
17	00010111	During START I/O. Unit check on a sense command is indicated.
18	00011000	Available space for reading RLD records has been exceeded.
19	00011001	Unexpected program interruption. IPL damaged.
FF	11111111	No IPL on this direct-access device.

Figure 30. IPL Error Types

APPENDIX B: NUCLEUS INITIALIZATION PROGRAM (NIP)

The Nucleus Initialization Program (NIP) consists of several subroutines, each of which performs an initialization function required by the resident portion (nucleus) of the primary control program including:

1. Opening the SVC and Link libraries,
2. Setting the protection key of main storage (in systems with the optional storage protection feature),
3. Placing the addresses of the upper and lower boundaries of the dynamic area into the boundary box.

The NIP sub-routines are packaged in one non-resident module, processed by the linkage editor together with the nucleus modules. NIP is loaded into main storage immediately before the relocated portion of the IPL program (see Figure 29). NIP is entered from the IPL program and, on completion, passes control to the dispatcher, after which it may be overlayed by the processing programs.

NIP operates partially under its own stand-alone input/output routine and under system routines including the I/O supervisor. NIP has its own TCB, RB and boundary box pre-assembled within NIP code. For more complete information, see the publication: Initial Program Loader and Nucleus Initialization Program.

The NIP module initializes the following:

- Communication Vector Table (CVT).
- Dynamic Area.
- Boundary Box.
- Free Area Queue Element.
- SYS1.SVCLIB, SYS1.LINKLIB, SYS1.LOGREC DEB.
- SVC Table Extension (optional).
- Protection Key (optional).
- Timer (optional).
- Resident BLDL Table (optional).
- Resident Access Method Routines (optional).
- Resident Type 3 and 4 SVC Routines (optional).
- Resident Job Queue (optional).

NIP FUNCTIONS

NIP control flow is shown in Chart 20. When entered from the IPL program, NIP saves the address of the system residence device, stored in register 10 by the IPL program. It rounds the address of the end of nucleus up to a double-word or 2048 byte boundary and stores this value in the CVT for use by the system environment recorder (SERO).

To initialize the dynamic area, NIP moves a PRB and XCTL code (which have been pre-assembled within NIP) to the beginning of the dynamic area. NIP then relocates the address constants within the PRB and XCTL code. When this XCTL code receives control (at the completion of NIP), it causes control to be passed to the job scheduler.

NIP determines the addresses of the free area queue element and lower and upper boundaries for the dynamic area. It stores these addresses in the boundary box. It also stores the number of free bytes in the dynamic area in the free area queue element.

NIP builds and initializes DEBs for SYS1.LINKLIB and SYS1.SVCLIB, and completes initialization for the SYS1.LOGREC DEB.

NIP optionally extends the SVC table to contain the TTR and the length of each transient SVC routine.

NIP optionally determines if the timer is enabled and working. If the timer is not enabled and working, NIP sends a timer-status message to the operator. If the timer is enabled and working, it sets the timer with a value of six hours.

NIP optionally determines the protection key for the dynamic area from the "protect key" field within the TCB. It sets the storage key of the dynamic area.

NIP optionally makes all or part of the SYS1.LINKLIB directory resident.

NIP optionally makes resident a group of access method modules.

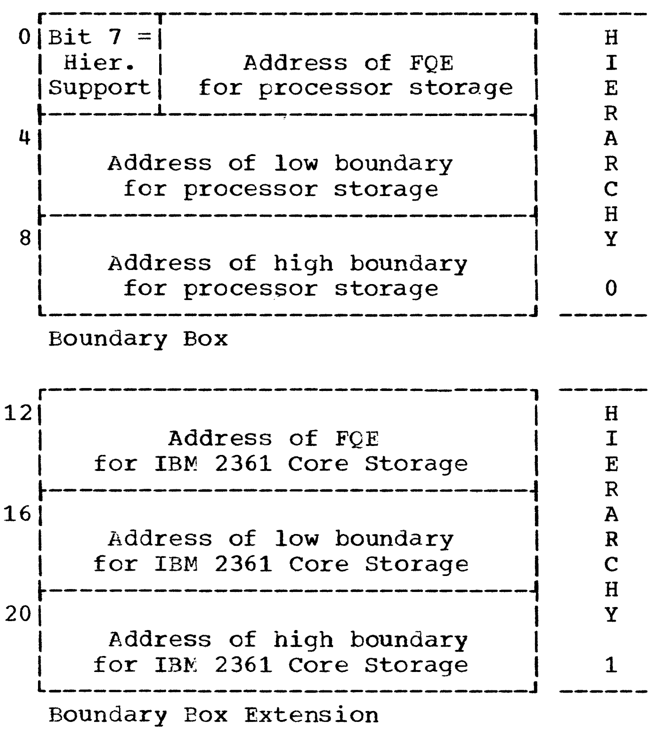
NIP optionally makes resident a group of type 3 and type 4 SVC routines.

NIP optionally obtains space for job queue records.

After completing all initialization procedures, NIP passes control to the dispatcher. The dispatcher then gives control to the XCTL macro instruction which NIP had placed in the dynamic area. This causes the job scheduler to be brought into the dynamic area and given control.

CVT INITIALIZATION

If the optional BLDL table or RAM and RSVC routines (explained later in this section) are to be included in the system, they will be added at the end of the nucleus. NIP then increases the address at the end of the nucleus (or at the end of the optional routines) to a double-word boundary in systems without storage protection. In protected systems, NIP increases this address to a 2048 byte (2K) boundary. NIP then stores this address (the lowest address not in the fixed area) in CVT field CVTNUCB. It is used by the system environment recorder.



0
Bit 7 = Hier. Support
Address of FQE for processor storage
4
Address of low boundary for processor storage
8
Address of high boundary for processor storage
HIERARCHY 0
Boundary Box
12
Address of FQE for IBM 2361 Core Storage
16
Address of low boundary for IBM 2361 Core Storage
20
Address of high boundary for IBM 2361 Core Storage
HIERARCHY 1
Boundary Box Extension

• Figure 31. Boundary Box

DYNAMIC AREA INITIALIZATION

The portion of main storage outside the fixed area is called the dynamic area. NIP initializes the dynamic area as follows:

- Pre-assembled code is moved from NIP to the beginning of the dynamic area. This code includes a PRB and the XCTL code that causes loading of the job scheduler through an XCTL macro instruction.
- The address constants are relocated in the PRB and XCTL code.

BOUNDARY BOX INITIALIZATION

A three word (12 byte) boundary box specifies the boundaries of the dynamic area (see Figure 31). NIP initializes the boundary box as follows:

Word 1 Address of a free area queue element (FQE) describing all free space in processor storage.

Word 2 Address of the beginning of the dynamic area. In an unprotected system, this is the address of the end of the fixed area rounded up to a double word boundary. In a storage protected system, the address of the end of the fixed area is rounded up to a 2048 byte (2K) boundary.

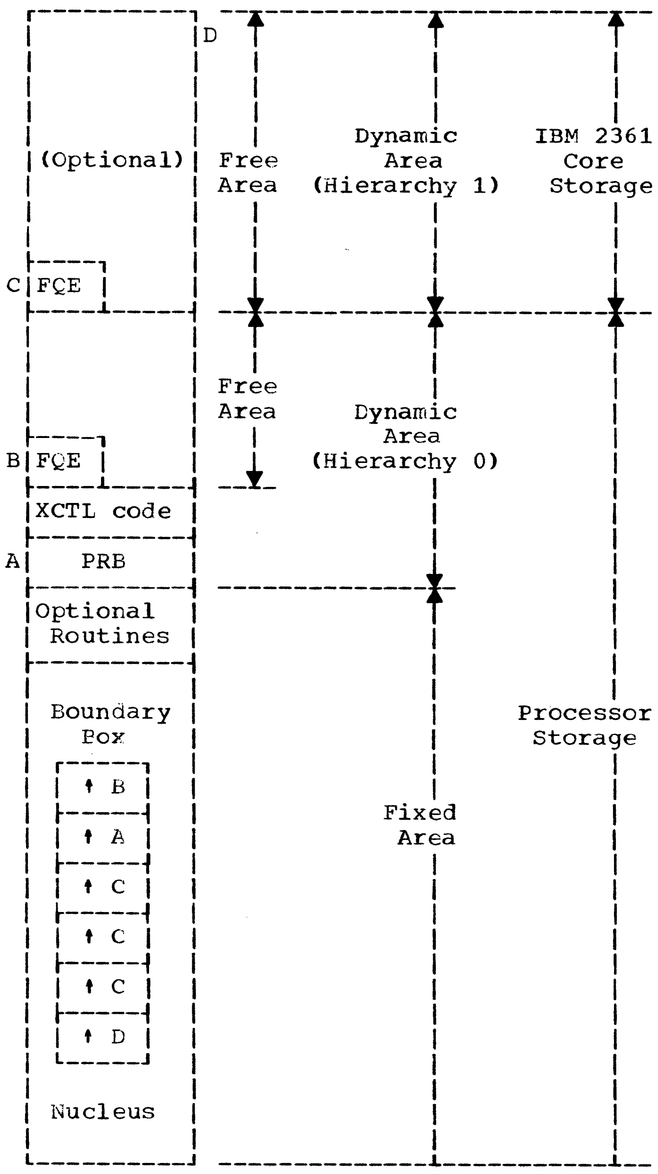
Word 3 Highest address, plus one byte, in processor storage. This address is passed to NIP by IPL.

If Main Storage Hierarchy Support is included in the system, bit 7 of the first byte in the boundary box is set to "1", and the boundary box is expanded to six words (see Figure 31). <u>If IBM 2361 Core Storage is not included in the system</u>, the first three words of the boundary box are initialized as shown in the preceding paragraph, and the additional three words are set to zero. <u>If it is included in the system</u>, then the dynamic area is divided into two parts. The portion of the dynamic area within processor storage is known as hierarchy 0; the IBM 2361 Core Storage portion of the dynamic area is known as hierarchy 1. The first three words of the boundary box describe hierarchy 0 and are initialized as shown in the preceding paragraph. The additional three words (Words 4, 5, and 6) describe hierarchy 1 and are initialized as follows:

Word 4 Address of an FQE describing all IBM 2361 Core Storage space.

Word 5	Address of the beginning of IBM 2361 Core Storage. This address is one higher than the last processor storage address.
Word 6	Highest address, plus one byte, in IBM 2361 Core Storage. This address is passed to NIP by IPL.

Figure 32 shows main storage and the boundary box (for a system including Main Storage Hierarchy Support and IBM 2361 Core Storage) after being initialized by NIP.



• Figure 32. Dynamic Area and Boundary Box Initialization

FREE AREA QUEUE ELEMENT INITIALIZATION

The free area queue element (FQE) for processor storage (hierarchy 0 of the dynamic area) is a double word following the PRB and XCTL code in the dynamic area. NIP initializes this FQE by:

- Calculating the length of the free area within processor storage and storing this value in the second word of the FQE. The free area is defined as the entire area from the address of the FQE to the end of processor storage (see Figure 32).

- Storing zeros in the first word of the FQE.

The FQE for IBM 2361 Core Storage (hierarchy 1 of the dynamic area) is a double word at the beginning of this storage space. NIP initializes this FQE by:

- Calculating the total length of IBM 2361 storage space and storing this value in the second word of the FQE.

- Storing zeros in the first word of the FQE.

Figure 33 shows an FQE built by NIP.

0	4	8
Zeros	Length of free area (in bytes)	

• Figure 33. Free Area Queue Element (FQE) Built by NIP

SYS1.SVCLIB, SYS1.LINKLIB, AND SYS1.LOGREC DEB INITIALIZATION

NIP builds DEBs (data extent blocks) for the SYS1.LINKLIB and SYS1.SVCLIB system data sets. Main storage for the DEBs is acquired at the upper end of the nucleus. The size of the DEBs, and the extent descriptions, depends on their associated data set control blocks (DSCBs). As many as 16 extents may be specified, and SYS1.LINKLIB may consist of as many as 16 concatenated data sets (listed in member LNKLST00 of SYS1.PARMLIB), with a maximum of 16 extents each.

The SYS1.LINKLIB and SYS1.SVCLIB DEBs are built and initialized with information from the system catalog, the VTOCs (volume table of contents), and the DSCBs and DCBs (data control blocks) for these data sets. NIP also completes initialization for the SYS1.LOGREC system data set, with information obtained from its DCB and DSCB.

To initialize the DEB, NIP obtains the following data and stores them in the corresponding DEB fields:

- Start cylinder address and track address (CCHH) of the data set.
- End CCHH of the data set.
- Number of tracks occupied by the data set.
- UCB address for the system residence device.
- I/O Appendage Table address.

Figure 34 shows the DEB fields which are initialized by NIP.

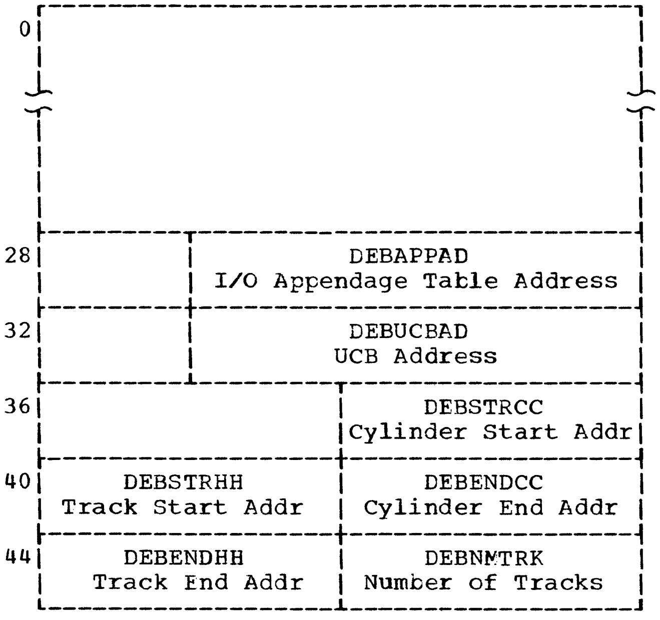


Figure 34. DEB Initialization

NIP executes in a stand-alone environment using its own input/output routine. To initialize the DEB, NIP:

1. Reads the standard volume label to determine the volume table of contents (VTOC) address on the system residence device.
2. Reads the data portion of VTOC DSCB to determine the tracks per cylinder for the system residence device.
3. Determines the UCB address of the system residence device through UCB table look up.
4. Determines the DEB address for SYS1.LOGREC. The DEB Address is available within the DCB. The DCB address for SYS1.LOGREC is available in CVT field CVTDCB.
5. Searches the VTOC and reads, into a buffer, the data portion of the DSCB for the data set.
6. Moves Start CCHH and End CCHH for the data set from the buffer into the DEB.
7. Computes the number of tracks contained within the data set extent and stores this value in the DEB.
8. Stores the UCB address into the DEB.
9. Moves the I/O Appendage table address from CVT field CVTXAPG to the DEB.

Note: NIP also completes initialization of all fields in the SYS1. LINKLIB and SYS1.SVCLIB DEBs. See IBM System/360 Operating System: System Control Blocks, Form C28-6628, for further information.

SVC TABLE EXTENSION (TTR TABLE) INITIALIZATION

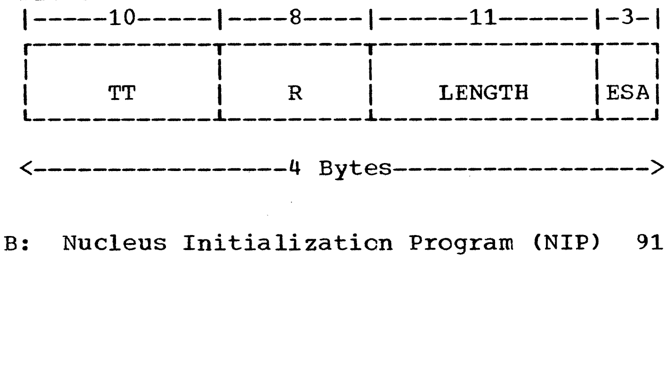
This is an optional NIP function that is selected during system generation.

The TTR address and length (L) of each non-resident SVC routine are available in the partitioned data set (PDS) directory of the SVC library.

NIP initializes the TTR table by:

- Searching the PDS directory of the SVC library to find the TTR and length of each transient SVC routine.
- Storing TTR and L of each transient SVC routine in a table within the nucleus. The assigned area for this table is within the SVC handler routine.

The TTR table contains a 4-byte entry for each transient SVC routine. The format of each 4-byte entry in the table is shown below:



where:

TT =	Track address of the transient SVC routine relative to the start of the SYS1. SVCLIB data set.
R =	Record number on the track.
L =	Length in bytes of the transient SVC routine.
ESA =	Extended save area in double words. This field is pre-assembled in the table.

NIP uses the following information available in the SVC handler routine to initialize the TTR table:

- Relocation table, containing a 1-byte index number for each SVC in the SVC table.
- Highest number assigned to an IBM supplied SVC routine.
- Highest number assigned to a resident SVC routine.

To initialize the TTR table, NIP:

1. Constructs an eight byte name for the transient SVC by using the relocation table and the highest resident SVC number, as explained below:
    - Picks up the entry in the relocation table which corresponds to a transient SVC.
    - Translates the entry number in the relocation table to a SVC number.
    - Converts the SVC number from binary to decimal.
    - Unpacks the decimal number to a 4-byte number.
    - Constructs an 8-byte name for the SVC routine by placing the 4-byte unpacked decimal number beside a pre-assembled four character prefix for the SVC names, as follows:

IGC0	XXXX
pre-assembled prefix	unpacked decimal number

2. Loads the following registers:
    - Address of the input parameter list to the BLDL macro instruction is placed in register 0.
    - Address of the SYS1.SVCLIB DCB is placed in register 1.
3. Issues the BLDL macro instruction to search the SYS1.SVCLIB directory.
4. Tests for the successful execution of the BLDL macro instruction.
5. On successful completion, BLDL returns the data extent for the SVC routine in a return area. NIP moves the TTR and length of the SVC routine from the return area into the TTR table, in a format shown in the diagram above.
6. When unsuccessful, BLDL returns an error code in register 15. NIP tests the error code and sends one of the following error messages to the operator:

   "IEA101I SVC ROUTINE IGC0XXXX NOT AVAILABLE - PERMANENT I/O ERROR ON SVC LIBRARY."

   "IEA102I SVC ROUTINE IGC0XXXX NOT AVAILABLE - NOT FOUND ON SVC LIBRARY."
7. Scans the relocation table and repeats the above procedure for each transient SVC routine.

PROTECTION KEY INITIALIZATION

Main storage protection is an optional hardware feature. If this hardware is included in the Central Processing Unit, storage protection can be selected during System Generation through use of the PROTECT option in the SUPRVSOR macro instruction. When protection is selected, the storage keys are set as follows:

- The storage occupied by the nucleus is set to a key of zero.
- The dynamic area is set to the non-zero key specified in the "protect key" field of the TCB.

TIMER INITIALIZATION

The timer is an optional hardware feature. It can be enabled or disabled by a switch on the system control panel.

To initialize the timer, NIP:

1. Determines if the timer is working by:
    - Setting location 80 to a value of six hours (X'6309109E).
    - Waiting for the timer to decrement.
    - Comparing the contents of location 80 with the original six hour value.

If the contents of location 80 are equal to six hours, NIP sends the following message to the operator:

"IEA100A TIMER IS NOT WORKING. PUT TIMER SWITCH ON."

2. Resets location 80 to a value of six hours.

BUILDING A RESIDENT DIRECTORY FOR SYS1.LINKLIB

This section is applicable only if the resident BLDL table option was selected during system generation.

Each time an ATTACH, LINK, XCTL, or LOAD macro instruction is issued, the system issues a BLDL with a subsequent program fetch of the module. When the resident BLDL table option is selected during system generation, a standard list which includes all or part of the SYS1.LINKLIB directory can be made resident in the nucleus by the nucleus initialization program. Any linkage to a SYS1.LINKLIB module causes a scan of the resident table before a direct access device search is initiated in the BLDL routine.

The message:

IEA101A SPECIFY SYSTEM PARAMETERS

is issued to the operator if the COMM option was specified in the SUPRVSOR system generation macro instruction. The operator may then:

1. Specify an alternate list of SYS1.LINKLIB modules whose directory entries are to be made resident.
2. Request a listing of the names of the modules whose directory entries were made resident.
3. Cancel the option for the current IPL.

If a list is selected, NIP then:

1. Reads the specified list from member IEABLDxx in SYS1.PARMLIB (where xx=00 or is replaced by two alphanumeric characters supplied by the operator).
2. Places the names in a table which is filled in by the BLDL routine.
3. Issues a BLDL.

If a normal return is received from the BLDL routine, the boundary box is adjusted to include the resident directory table as a part of the nucleus.

If an error code is returned from the BLDL routine, NIP issues one of the following messages:

IEA108I PERMANENT I/O ERROR DURING BLDL

The BLDL function is not performed. NIP continues to initialize the nucleus.

IEA109I BLDL FAILED FOR FOLLOWING MODULES

This message is followed by a list of names of the modules whose directory entries were not made resident because they were not found in SYS1.LINKLIB. NIP adjusts the boundary box to include the incomplete BLDL table and continues as though the table had been completed.

NIP places the address of the BLDL table into an area in the BLDL routine, IECPFND1.

RESIDENT ACCESS METHOD (RAM) INITIALIZATION

When the RAM option is selected during system generation, a group of access method modules are preloaded as part of the nucleus by the nucleus initialization program, thus creating a permanent system load list. Each time a LOAD is issued for any access method module, the system load list is checked. A program fetch is not performed if the module is found in the system load list. Otherwise, the system loads the module in the standard manner.

If the COMM option was specified in the SUPRVSOR macro instruction during system generation, NIP issues the following message to the operator:

IEA101A SPECIFY SYSTEM PARAMETERS

The operator may then:

1. Specify an alternate list of access method modules to be loaded.
2. Request a listing of the names of the access method modules that were loaded.
3. Cancel the option for the current IPL.

If a list was selected, NIP then:

1. Reads the specified list of access method modules from member IEAIGGxx in SYS1.PARMLIB.
2. Issues a LOAD macro instruction for each module in the list. This creates a load list attached to the TCB. The list pointer is moved to an area in the nucleus which is reserved for the system load list pointer.

If NIP is unable to load an access method module, it issues the following message:

IEA110I LOAD FAILED FOR (module name)

NIP continues to initialize the nucleus even though the named access method module was not loaded as part of the RAM option.

3. The boundary box is adjusted to include the system load list and access method modules as part of the nucleus.

RESIDENT TYPE 3 AND 4 SVC ROUTINE INITIALIZATION

When the resident type 3 and 4 SVC routine option is selected during system generation, a standard list of type 3 and 4 SVC routines may be loaded as part of the nucleus by NIP. If the COMM option was specified in the SUPRVSOR macro instruction during system generation, NIP issues the following message to the operator:

IEA101A SPECIFY SYSTEM PARAMETERS

The operator may then:

1. Specify an alternate list of type 3 and 4 SVC routines to be loaded.
2. Request a listing of the names of the routines that were loaded.
3. Cancel the option for the current IPL.

If a list was selected, NIP then:

1. Reads the specified list of SVC routines from member IEARSVxx in SYS1.PARMLIB.
2. Issues a LOAD macro instruction for each module in the list. This creates a load list attached to the TCB. If the module is a type 3 routine or the first module of a type 4 routine, its entry point is placed in the SVC table as discussed in the section entitled "Resident Type 3 and 4 SVC Routine Option." After all loading has been completed, the load list contains entries for routines requested by type 4 SVC routines via XCTL macro instructions. Following these entries, regardless of the order in which the routines were actually loaded, are entries for the first loads of type 3 or 4 SVC routines. The list pointer is moved to an area in the nucleus which is reserved for the RSVC system load list pointer. If NIP is unable to load an SVC routine, it issues the following message:

IEA1101I LOAD FAILED FOR (module name)

NIP continues to initialize the nucleus even though the named routine was not loaded as part of the resident type 3 and 4 SVC routine option.

If a requested SVC routine is not supported at the installation, NIP issues the following message:

IEA114I SVC (xxx) NOT SUPPORTED

The named SVC routine is defined but cannot be loaded because it is not supported at the installation.

If a requested SVC routine is undefined, NIP issues the following message:

IEA115I SVC (xxx) NOT DEFINED

Indicating that no such SVC routine exists.

3. The boundary box is adjusted to include the RSVC load list and SVC routines as part of the nucleus.

RESIDENT JOB QUEUE INITIALIZATION

When the resident job queue option is selected during system generation, NIP obtains the area needed to hold a specified number of job queue records. If the COMM option was specified in the SUPRVSOR macro instruction during system generation, the number of resident job queue records specified during system generation may be overridden when the nucleus is initialized. In this case, NIP issues the following message to the operator:

IEA101A SPECIFY SYSTEM PARAMETERS

The operator may then vary the number of job queue records for the current IPL. After the operator responds, NIP obtains an area whose size is based on the number of records to be made resident. The area becomes part of the nucleus. A pointer to the area is saved in a portion if the nucleus that was reserved for this purpose when the resident job queue option was selected.

Csect Name	Sysgen Output Macro to be Checked for Module Name	Microfiche Module Name	Routine Name (or Other Specified Function; e.g., Table)
IEAAIH00	IEAAIH IEAAPS IECIOS	[1] [1] [1]	First Level Interruption Handlers (FLIHs) Dispatcher and Exit Effector I/O Supervisor
IGC009	---	IEAADL00	Delete
IGC012	---	IEAASY00	Synch
IGC010	---	IEAAMS00	Getmain
IEA0PL00	---	IEAAPL00	Prolog
IGC011	---	IEA0RT10	Timer SVC
IEEBA1	---	IEECIR01	Console Interruption (Job Management)
IEA0AB00	---	IEAAAB00	Abterm
IGC001	IEAAWT	[1]	Wait
IHASVC00	SGIEA2SV	[1]	SVC Table
IEAATA00	IEAATA	[1]	SVC Second Level Interruption Handler (SLIH) Exit and Transient Area Handler
IEACVT	CVT	[1]	Communications Vector Table
IGC002	IEAAPT	[1]	Post
IGC006	IEAATC	[1]	Link, Load, XCTL Transient Area
IEATCB00	IEATCB	[1]	Control Blocks
IEWFTMIN	---	IEWFTMIN	Program Fetch
IEWFTPCI	---	IEWFTPCI	Program Controlled Interrupt Fetch
IEFJOB	---	IEFKRESA	Job Scheduler Tables and Work Area (Job Management)

[1]Variable module names, dependent on macro instruction's use.

(continued)

(continued)

Csect Name	Sysgen Output Macro to be Checked for Module Name	Microfiche Module Name	Routine Name (or Other Specified Function; e.g., Table)
IFBDCB00	---	IFBDCB00	System Environment Recorder (SER) Data Control Block
IGC018	---	IECPFIND	Find (Data Management)
IGC037	---	IEWSVOVR	Overlay Supervisor
IEEBC1PE	---	IEEBC1PE	External Interruption (Job Management)
IEC2311A	---	IEC2311A	Disk Error Routine (I/O Supervisor)
IEFDPOST	---	IEFDPOST	Unsolicited Interruption (Job Management)
IEEMSLT	SGIEE001	[1]	Master Scheduler Resident Control Data Area (Job Management)
IECZDTAB	SGIEC0DT	[1]	Direct Access Device Table (I/O Supervisor)
IECINTRP	---	IECINTRP	Sense and Status Interpreter (I/O Supervisor)
IEAANIP0	IEAANIP	[1]	Nucleus Initialization Program

[1]Variable module names, dependent on macro instruction's use.

CONTRCL RECORD - (LOAD MODULE)

```
┌─┬───┬──┬──┬─────────┬──┬──┬─             ─────────────────────────────────────────┐
│0│1-3│4,│6,│8-15     │  │  │                                                        │
│ │   │5 │7 │         │  │  │                       Record length is 20 bytes         │
└─┴───┴──┴──┴─────────┴──┴──┴─             ─────────────────────────────────────────┘
 │ │   │  │  │         │  │
 │ │   │  │  │         │  │
 │ │   │  │  │         │  └──Length of control section
 │ │   │  │  │         │            the control section (in bytes) that the text in
 │ │   │  │  │         │            the following record belongs to (2 bytes)
 │ │   │  │  │         │
 │ │   │  │  │         └─────────────CESD entry number - specifies the composite
 │ │   │  │  │                           external symbol dictionary entry that
 │ │   │  │  │                           contains the control section name of the
 │ │   │  │  │                           control section that this text is part
 │ │   │  │  │                           of (2 bytes)
 │ │   │  │  │
 │ │   │  │  └──Channel Command Word (CCW) - that could be used to read the text
 │ │   │  │              record that follows.  The data address field contains
 │ │   │  │              the linkage editor assigned address of the first byte
 │ │   │  │              of text in the text record that follows.  (8 bytes)
 │ │   │  │
 │ │   │  │
 │ │   │  └──Count - contains two bytes of binary zeros.  The count field contains the
 │ │   │              length of the record.
 │ │   │
 │ │   └──Count - in bytes of the control information (CESD ID, length of
 │ │               control section) following the CCW field  (2 bytes)
 │ │
 │ └──Spare - contains three bytes of binary zeros
 │
 └──Identification - specifies that this is:     (1 byte)
```

- A control record - 0000 0001
- The control record that precedes the last text record of this overlay segment - 0000 0101
- The control record that precedes the last text record of the module - 0000 1101

RELOCATION DICTIONARY RECORD - (LOAD MODULE)

```
┌─┬───┬──┬──┬─────────┬────────        ──────────────────────────────────────────┐
│0│1-3│4,│6,│8-15     │16-255                 Record length can be               │
│ │   │5 │7 │         │                       between 24 and 256 bytes           │
└─┴───┴──┴──┴─────────┴────────        ──────────────────────────────────────────┘
 │ │    │  │  │         │
 │ │    │  │  │         │
 │ │    │  │  │         └--RLD data -- see below
 │ │    │  │  │
 │ │    │  │  └--Spare - contains 8 bytes of binary zeros
 │ │    │  │
 │ │    │  └--Count - in bytes of the relocation dictionary information following
 │ │    │             the spare 8 byte field  (2 bytes)
 │ │    │
 │ │    └--Count - contains two bytes of binary zeros
 │ │
 │ └--Spare - contains three bytes of binary zeros
 │
 └--Identification - specifies that this is:     (1 byte)

                        •  A relocation dictionary record - 0000 0010

                        •  The last record of the segment - 0000 0110

                        •  The last record of the module - 0000 1110
```

RLD Data -- see above

```
┌──┬──┬─┬───┬─┬───┐      ┌─┬───┬──┬──┬─┬───┬──┬──┬─┬───┐
│  │  │ │   │ │   │      │ │   │  │  │ │   │  │  │ │   │
│R │P │F│ A │F│ A │      │F│ A │R │P │F│ A │R │P │F│ A │
└──┴──┴─┴───┴─┴───┘      └─┴───┴──┴──┴─┴───┴──┴──┴─┴───┘
 │  │       │                          │
 │  │       │                          └--Address - linkage editor assigned
 │  │       │                                       address of the address
 │  │       │                                       constant (3 bytes)
 │  │       │
 │  │       └--Flag - specifies miscellaneous information as follows:  (1 byte)
 │  │                 when byte format is xxxxLLST:
 │  │                 xxxx specifies the type of this RLD item (address constant)
 │  │                 0000 -- non-branch type in assembler language,a DC A(name)
 │  │                 0001 -- branch type (in assembler language, a DC V(name)
 │  │                 0010 -- pseudo register displacement value
 │  │                 0011 -- pseudo register cumulative displacement value
 │  │                 1000 and 1001 -- this address constant is not to be relocated,
 │  │                 because it refers to an unresolved symbol.
 │  │                 LL specifies the length of the address constant
 │  │                 01 -- two byte
 │  │                 10 -- three byte
 │  │                 11 -- four byte
 │  │                 S specifies the direction of relocation
 │  │                 0 -- positive
 │  │                 1 -- negative
 │  │                 T specifies the type of RLD item following this one
 │  │                 0 -- the following RLD item has a different relocation
 │  │                      and/or position pointer
 │  │                 1 -- the following RLD item has the same relocation and
 │  │                      position pointers as this one, and therefore is omitted
 │  │
 │  └--Position pointer - contains the entry number of the CESD entry (or trans-
 │                        lation table entry) that indicates which control section
 │                        the address constant is in (2 bytes)
 │
 └--Relocation pointer -  contains the entry number of the CESD entry (or transla-
                          tion table entry) that indicates which symbol's value
                          is to be used in the computation of the
                          address constant's value  (2 bytes)
```

CONTROL AND RELOCATION DICTIONARY RECORD - (LOAD MODULE)

```
┌─┬───┬──┬──┬─────────┬──┬──┬─┬────┬─┬───┐   ┌─┬───┬──┬──┐
│0│1-3│4,│6,│8-15     │  │  │ │    │ │   │   │ │   │  │  │
│ │   │5 │7 │         │  │  │ │    │ │   │   │ │   │  │  │
│ │   │  │  │         │  │  │ │    │ │   │   │ │   │  │  │
└─┴───┴──┴──┴─────────┴──┴──┴─┴────┴─┴───┘   └─┴───┴──┴──┘
 │ │   │  │  │         │  │  │ │    │ │             │  │
 │ │   │  │  │         │  │  │ │    │ │             │  │
 │ │   │  │  │         │  │  │ │    │ └--Address    │  └--Length of control
 │ │   │  │  │         │  │  │ │    │               │     section (2 bytes)
 │ │   │  │  │         │  │  │ │    └--Flag         │
 │ │   │  │  │         │  │  │ │                    └--CESD entry number
 │ │   │  │  │         │  │  │ │                        (2 bytes)
 │ │   │  │  │         │  │  │ │
 │ │   │  │  │         │  │  │ └--Address (3 bytes)
 │ │   │  │  │         │  │  │
 │ │   │  │  │         │  │  └--Flag (1 byte)
 │ │   │  │  │         │  │
 │ │   │  │  │         │  └--Position pointer (2 bytes)
 │ │   │  │  │         │
 │ │   │  │  │         └--Relocation pointer (2 bytes)
 │ │   │  │  │
 │ │   │  │  └--Channel Command Word (8 bytes)
 │ │   │  │
 │ │   │  └--Count of RLD information (2 bytes)
 │ │   │
 │ │   └--Count of control information (2 bytes) - the control information contains the
 │ │      ID and length of control sections in the following text record.
 │ │
 │ └--Spare (3 bytes)
 │
 └--Identification (1 byte) - specifies that this record is:

                               • A control and RLD record - 0000 0011

                               • A control and RLD record that is followed by the
                                 last text record of a segment - 0000 0111

                               • A control and RLD record that is followed by the
                                 last text record of a module - 0000 1111
```

Note: For detailed descriptions of the data fields see:

Relocation Dictionary Record
Control Record

The record length will vary from 20 to 260 bytes.

PARTITIONED ORGANIZATION DIRECTORY RECORD - (AS RECEIVED FROM BLDL)

```
 Byte
   ┌────────────────────────────────────────────────────────────────────┐
  0│                                                                    │
   │                Name of load module (member or alias name)          │
  4│                                                                    │
   ├───────────────────────────────────────────────────────┬────────────┤
  8│ Relative (to beginning of data set) disk address of   │ Concatenation│
   │                 module  (TTR)                         │  number    │
   ├─────────────────┬─────────────────┬───────────────────┴────────────┤
 12│ Byte of binary  │Alias indicator and│ Relative (to beginning of data set) │
   │ zeros.  1       │miscellaneous info.│disk address of first text record.   │
   ├─────────────────┼─────────────────┼────────────────────────────────┤
 16│continuation of  │ Byte of binary  │Relative (to beginning of data set)│
   │disk address     │ zeros           │disk address of NOTE List or Scatter-│
   ├─────────────────┼─────────────────┼────────────────────────────────┤
 20│ translation record│Number of entries│ Module attributes (see next page)│
   │                 │in NOTE List   2 │0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15│
   ├─────────────────┴─────────────────┴─────────────────┬──────────────┤
 24│ Total contiguous quantity of main storage required by the│Length(in bytes) of│
   │ module                                              │first text record.│
   ├─────────────────┬───────────────────────────────────┴──────────────┤
 28│ continuation of │Module's linkage editor assigned entry point address│
   │ Length.         │                                                  │
   ├─────────────────┴───────────────────────────────────┬──────────────┘
 32│Linkage editor assigned origin of first text record. │
   │                                                     │
   └─────────────────────────────────────────────────────┘
                                                         ┌──────────────┐
                                                         │Length of scatter│
        For load modules in scatter format add:          │              │
   ┌─────────────────┬───────────────────────────────────┼──────────────┤
 36│List (in bytes)  │Length of translation table (in bytes)│ESDID (CESD entry│
   │                 │                                   │number of control│
   ├─────────────────┼───────────────────────────────────┼──────────────┘
 40│section name) for│ESDID (CESD entry number of control │
   │first text record.│section name) containing entry point.│
   └─────────────────┴───────────────────────────────────┘
                                                         ┌──────────────┐
        For load modules with RENT or REUS attribute and Alias │Entry point address│
        names add:                                       │              │
   ┌─────────────────────────────────┬───────────────────┴──────────────┤
 36│ of the member name.             │                                  │
   │                                 │                                  │
   ├─────────────────────────────────┘                                  │
 40│                          Member name                               │
   │                                 ┌──────────────────────────────────┘
 44│                                 │
   └─────────────────────────────────┘

   ┌────────────────────────────────────────────────────────────────────┐
   │       SSI Bytes - Aligned on a half-word boundary at the end of the PDS │
   │                   record.                                          │
   └────────────────────────────────────────────────────────────────────┘
```

Alias indicator and miscellaneous Information:
1. Alias indicator -- 0 signifies none,1 signifies alias -- bit 0
2. Number of relative disk addresses (TTR)in user data field -- bits 1,2
3. Length of user data field (in halfwords) -- bits 3-7

PDS Directory Record size (for SSI, add 4 bytes to sizes):

Block format	36 bytes	Scatter format	44 bytes
Block format with alias names	46 bytes	Scatter format with alias names	54 bytes

1 This is normally a zero byte inserted to maintain half-word boundaries. If the DCB operand in the BLDL macro instruction was specified as zero, this byte will contain a 1 if the name was found in the link library, and a 2 if the name was found in the job library.

2 This byte contains zero if load module is not in overlay structure.

MODULE ATTRIBUTES

(see bytes 22 and 23 on the preceding page.)

Bit Number	Attribute	Bit setting	Indication
0	RENT	0	Not reenterable
		1	Reenterable
1	REUS	0	Not reusable
		1	Reusable
2	OVLY	0	Not an overlay module
		1	Overlay module
3	TEST	0	Not under test
		1	Under test
4	LOAD	0	Not only loadable
		1	Only loadable [1]
5	Format	0	Block format
		1	Scatter format
6	Executable	0	Not executable
		1	Executable
7	Format	0	Module contains more than one text record and/or RLD record(s).
		1	Module contains only one text record and no RLD record.
8	Compatibility	0	Module can be processed by all levels of linkage editor.
		1	Module cannot be reprocessed by linkage editor-E.
9	Format	0	Linkage editor assigned origin of first text record is not zero.
		1	Linkage editor assigned origin of first text record is zero.
10	Format	0	Linkage editor assigned entry point is not zero.
		1	Linkage editor assigned entry point is zero.
11	Format	0	Module contains RLD record(s)
		1	Module does not contain an RLD record.
12	Editability	0	Module can be reprocessed by linkage editor.
		1	Module cannot be reprocessed by linkage editor.
13	Format	0	Module does not contain TESTRAN symbol records.
		1	Module contains TESTRAN symbol records.
14	E44 Linkage Editor	0	module not created by E44 Linkage Editor
		1	module created by E44 Linkage Editor
15	Refreshable	0	not refreshable
		1	refreshable

[1] Module can only be loaded 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.

APPENDIX E: ENQ/DEQ QUEUE CONTROL BLOCK (QCB) FORMATS

MAJOR QUEUE CONTROL BLOCK (MAJOR QCB)

The beginning of the major QCB queue is addressed by CVT field IEAAQCB0. The format of a major QCB is:

```
 0 ┌──────────────┬──────────────────────────────────────────┐
   │              │                                          │
   │      0       │          Address of next major QCB       │
 4 ├──────────────┼──────────────────────────────────────────┤
   │              │                                          │
   │      0       │          Address of previous major QCB   │
 8 ├──────────────┼──────────────────────────────────────────┤
   │              │                                          │
   │      0       │          Address of first minor QCB      │
12 ├──────────────┴──────────────────────────────────────────┤
   │                                                         │
   │                                                         │
16 │                         Major name                      │
   │                                                         │
   │                                                         │
20 └─────────────────────────────────────────────────────────┘
```

Address of next major QCB - the address of the next QCB on the major QCB queue. If this is the last QCB on the queue, this field is zero.

Address of the previous major QCB - the address of the previous major QCB on the major QCB queue. If this is the first QCB on the queue, this field is the address of CVT field IEAAQCB0.

Address of the first minor QCB - the address of the first minor QCB for this major QCB.

Major name - the 8-byte major resource name.

MINOR QUEUE CONTROL BLOCK (MINOR QCB)

The format of a minor QCB is:

```
 0 ┌──────────────┬────────────────────────────────────────┐
   │              │                                        │
   │      0       │          Address of UCB                │
 4 ├──────────────┼────────────────────────────────────────┤
   │              │                                        │
   │      0       │          Address of previous minor QCB │
 8 ├──────────────┼────────────────────────────────────────┤
   │              │                                        │
   │      0       │          Address of next minor QCB     │
12 ├──────────────┼──────────────┬─────────────────────────┤
   │  Minor name  │QCB protection│                         │
   │    length    │     key      │                         │
16 ├──────────────┴──────────────┘       Minor name        │
   ┴                                                       ┴
   ┬                                                       ┬
   └───────────────────────────────────────────────────────┘
```

<u>Address of UCB</u> - the address of the UCB representing the direct access device on which the named resource resides.

<u>Address of the previous minor QCB</u> - the address of the previous minor QCB. If this is the first minor QCB on the queue, this field contains the address of the major QCB.

<u>Address of the next minor QCB</u> - the address of the next minor QCB on the queue. If this is the last minor QCB, this field is zero.

<u>Minor name length</u> - the length in bytes of the minor resource name.

<u>QCB protection key</u> - the protection key of the job step (if applicable).

<u>Minor name</u> - a variable length (1 to 255 bytes) minor name.

APPENDIX F: ENTRY AND SEGMENT TABLE FORMATS

ENTRY TABLE (ENTAB)

```
┌─────────────────────────────┬─────────────────────────────┬───────┬───────────────────┐
│Unconditional branch to last │Address of symbol referred to│"to"seg│Previous Caller    │
│ entry BC 15,DISP(15,0)      │                             │number │(zero initially)   │
├─────────────────────────────┼─────────────────────────────┼───────┼───────────────────┤
│Unconditional branch to last │Address of symbol referred to│"to"seg│Previous Caller    │
│ entry BC 15,DISP(15,0)      │                             │number │(zero initially)   │
└─────────────────────────────┴─────────────────────────────┴───────┴───────────────────┘

│                             │                             │       │                   │

│                             │                             │       │                   │

│                             │                             │       │                   │

┌─────────────────────────────┬─────────────────────────────┬───────┬───────────────────┐
│Unconditional branch to last │Address of symbol referred to│"to"seg│Previous Caller    │
│ entry-BC 15,DISP(15,0)      │                             │number │(zero initially)   │
├──────────────┬──────────────┴──────────────┬──────────────┼───────┼───────────────────┤
│   SVC 45     │L 15,4(0,15) Loads GR15 with │ BCR 15,15    │"from" │Address of segment │
│              │the value of the ADCON.      │              │seg.no.│ table (SEGTAB)    │
└──────────────┴─────────────────────────────┴──────────────┴───────┴───────────────────┘
│<---2 bytes-->│<--2 bytes--->│<--2 bytes--->│<---2 bytes-->│<1byte>│<-----3 bytes----->│
```

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.

SEGMENT TABLE (SEGTAB)

```
┌────┬───────────────┬──────────────────────────────────────────────────────────┐
│TEST│               │Address of Data Control Block (DCB) used to load module [1]│
│ind.│               │                                                          │
├────┴───────────────┼──────────────────────────────────────────────────────────┤
│                    │                  Address of note list                 [1]│
│                    │                                                          │
├────────────────────┼──────────────────────┬───────────────────┬───────────────┤
│Last segment        │Highest segment no.   │Last segment       │Highest segment no.│
│number of region 1  │in storage-region 1   │number of region 2 │in storage-region 2│
├────────────────────┼──────────────────────┼───────────────────┼───────────────┤
│Last segment        │Highest segment no.   │Last segment       │Highest segment no.│
│number of region 3  │in storage-region 3   │number of region 4 │in storage-region 4│
├────────────────────┼──────────────────────┴───────────────────┴───────────────┤
│Zero                │(Not used in the Fixed-Task Supervisor)                [1]│
│                    │                                                          │
├────────────────────┴──────────────────────────────────────────────────────────┤
│                     (Not used in the Fixed-Task Supervisor)                [1]│
│                                                                               │
├────────────────────┬─────────────────────────────────────────────────┬───────┤
│Previous segment [1]│                    Zero                         │status │
│number for segment1 │                                                 │indctr │
├────────────────────┼─────────────────────────────────────────────────┼───────┤
│Previous segment    │Address of entry table entry (when caller     [1]│status │
│number for segment2 │chain exists)                                    │indctr │
└────────────────────┴─────────────────────────────────────────────────┴───────┘
 .                    .                                                 .       .
 .                    .                                                 .       .
┌────────────────────┬─────────────────────────────────────────────────┬───────┐
│Previous segment    │Address of entry table entry (when caller     [1]│status │
│number for segmentN │chain exists)                                    │indctr │
└────────────────────┴─────────────────────────────────────────────────┴───────┘
│<-------------------------------4 bytes------------------------------------->│
```

TEST indicator -- specifies that this module is "under test" using TESTRAN. (Bit 1) Initialized by program fetch.

Highest segment no. in storage -- is initially set to 00 except for region 1 which is initially set to 01 by linkage editor.

Status indicator -- indicates the status of this segment with the two last bits of the entry table address field as follows:

00 -- segment is in main storage as a result of a branch to the segment.
10 -- segment is in main storage, no caller chain exists.
01 -- segment is not in main storage, but is scheduled to be loaded.
11 -- segment is not in main storage.

The status indicator for segment 1 is initially set to 10, all the rest are initially set to 11.

[1] Set to zero by linkage editor.

APPENDIX G: SER0 AND SER1 RECORD ENTRY FORMATS

SER0 and SER1 produce two types of record entries corresponding to the two types of errors processed: machine-check and channel errors. Record size varies with the type of record and with the machine model. The formats of the record entries produced by SER0 and SER1 are:

Machine Check Record Entry Format

```
┌──────────────┬─────┬─────┬─────┬─────────┐
│              │ SYS │ MOD │R.E. │         │
│ R.E. LABEL   │ ID  │ NO. │TYPE │  FLAGS  │
├──────────────┴─────┼─────┴─────┴─────────┤
│                    │                     │
│       DATE         │        TIME         │
│                    │                     │
├────────────────────┴─────────────────────┤
│                                          │
│            PROGRAM IDENTITY              │
│                                          │
├──────────────────────────────────────────┤
│                                          │
│          MACHINE CHECK OLD PSW           │
│                                          │
├──────────────────────────────────────────┤
│                                          │
│            ACTIVE I/O UNITS              │
│                                          │
│                    ┌─────────────────────┤
│                    │ CHANNEL TYPE        │
│                    │ ASSIGNMENTS         │
├────────────────────┴─────────────────────┤
│                                          │
│            GENERAL PURPOSE               │
│            REGISTER CONTENTS             │
│                                          │
├──────────────────────────────────────────┤
│                                          │
│             FLOATING POINT               │
│            REGISTER CONTENTS             │
│                                          │
├──────────────────────────────────────────┤
│                                          │
│  GENERAL PURPOSE REGISTER PARITIES       │
│                                          │
├────────────────────┬─────────────────────┤
│                    │                     │
│  FPR PARITIES      │        CPU          │
│                    │ HARDWARE LOGOUT     │
├────────────────────┘                     │
│                                          │
│  MODEL    BYTES                          │
│                                          │
│   40       256                           │
│   50       164     ┌─────────────────────┘
│   65       176     │
│   75       152     │
└────────────────────┘
```

Channel Error Record Entry Format

```
┌──────────────┬─────┬─────┬─────┬─────────┐
│              │ SYS │ MOD │R.E. │         │
│ R.E. LABEL   │ ID  │ NO. │TYPE │  FLAGS  │
├──────────────┴─────┼─────┴─────┴─────────┤
│                    │                     │
│       DATE         │        TIME         │
│                    │                     │
├────────────────────┴─────────────────────┤
│                                          │
│            PROGRAM IDENTITY              │
│                                          │
├──────────────────────────────────────────┤
│                                          │
│       FIRST CCW OF FAILING CHAIN         │
│                                          │
├──────────────────────────────────────────┤
│                                          │
│              FAILING CCW                 │
│                                          │
├──────────────────────────────────────────┤
│                                          │
│                  CSW                     │
│                                          │
├──────────────────────────────────────────┤
│                                          │
│            ACTIVE I/O UNITS              │
│                                          │
│                    ┌─────────────────────┤
│                    │ CHANNEL TYPE        │
│                    │ ASSIGNMENTS         │
├───────────┬────────┼─────────────────────┤
│ CHANNEL   │        │                     │
│ and UNIT  │ FLAGS  │        I/O          │
│ ADDRESS   │        │                     │
├───────────┴────────┘                     │
│                                          │
│            HARDWARE LOGOUT               │
│                                          │
├──────────────────────────────────────────┤
│                                          │
│  MODEL    BYTES                          │
│   40        0                            │
│   50       48                            │
│  65,75     24                            │
│                                          │
│                    ┌─────────────────────┘
│                    │
│                    │
└────────────────────┘
```

The fields in the record entry are interpreted as follows:

Record Entry Label - 3 bytes
Identifies the record as output from SER. It is set to SER in EBCDIC.

System Identifier - 1 byte
Identifies the version of SER which created the record.

0 = SER0, 1 = SER1

Model Number - 1 byte
Identifies the System/360 model on which the record was created.

Record Entry Type - 1 byte
Identifies the type of error that caused the record to be created.

C = machine check
I = channel error

Flags - 2 bytes

Byte 0

Bit 0 =		Spare bit
Bit 1 =	0	Record entry is complete
	= 1	Record entry is not complete
Bit 2 =	0	Channel and unit address matches a system UCB
	= 1	Channel and unit address does not match any system UCB
Bit 3 =	0	The operating system could not continue after the error
	= 1	The operating system could continue after the error
Bit 4 =	0	The scheduler was not in control when the machine check occurred.
	= 1	The scheduler was in control when the machine check occurred.

Byte 1

Bit 0 =	0	Program data was obtained
	= 1	Program data could not be obtained because the area from which it would have been extracted was overlayed. (Applies only to SER0.)

Other bits - unused

Date - 4 bytes
Identifies the year and day in packed decimal as follows:

00	XX	XXX	F
Unused	Year	Day	Zone

Time - 4 bytes
Identifies the time of day when the record entry was created.

XX	XX	XX	X	X
Hour	Minute	Second	Tenths	Hundredths

If the model does not have an interval timer, this field is zero.

Program Identity - 8 bytes
Identifies the program in process or the program requesting service when the error occurred.

Machine Check Old PSW - 8 bytes
The field is taken directly from locations 48-55.

Active I/O Units - 20 bytes
Identifies by channel and unit address a maximum of ten devices that were busy when the error occurred.

Channel Type Assignments - 4 bytes
Identifies the channel configuration of the system as follows:

```
     BYTE 0         BYTE 1
┌──────┬──────┬──────┬──────┬────────>
│CHAN 0│CHAN 1│CHAN 2│CHAN 3│ETC.
└──────┴──────┴──────┴──────┴────────>
```

Bit 0 =	0	Channel not present
	= 1	Channel present
Bit 1 =	0	Multiplexor channel
	= 1	Selector channel
Bit 2 =	0	Low speed
	= 1	High speed
Bit 3 =	0	Not a storage channel
	= 1	Storage channel

General Purpose Register Contents - 64 bytes

Identifies the contents of the GPRs at the time the error occurred. For the Model 50, only bits 0-27 and the parity bits are stored for each register. For Models 65 and 75, GPRs are tested for parity errors and corrected if necessary before being stored in this field.

Floating Point Register Contents - 32 bytes
Identifies the contents of all FPRs at the time the error occurred. The field is zero for Models 30 and 40 not equipped with the floating point feature.

General Purpose Register Parities - 8 bytes
For Model 40, this field is zero because hardware corrects parity during part of the machine check interrupt cycle, making parity indications unavailable. For Model 50, the field contains the last four bits of each register with the exception of registers 13, 14, and 15. (Applies only to SERO.) For Models 65 and 75, the field identifies the GPRs that contained parity errors when the error occurred. Only the first two bytes of the field are used. They are interpreted as follows:

```
Byte 0               Byte 1

0 0 0 0 0 1 0 0      0 0 1 0 0 0 0 0

Register 0               Register 15
```

Registers 5 and 10 had parity errors.

Note: If this information is stored by the SERO program for the model 75, no parity errors will be indicated for registers 13, 14, and 15 because SERO cannot determine the parity in these registers.

Floating Point Register Parities - 4 bytes
Identifies the FPRs that contained parity errors when the error occurred. The contents of the field differs according to model and is interpreted in the same manner as the GPR parity field. The field is zero for a Model 40 record.

CPU Hardware Logout - 152 to 256 bytes
Represents all or part of the contents of locations Hex 80 through Hex 17F.

First CCW of Failing Chain - 8 bytes
Identifies the first CCW of a chain of CCWs being executed when an error occurred.

Failiing CCW - 8 bytes
Identifies the specific CCW being executed when an error occurred.

CSW - 8 bytes
Identifies the CSW that was stored when an I/O error occurred.

Channel and Unit Address - 2 bytes
Identifies the device being serviced at the time of the channel failure.

Flags - 2 bytes
Not used.

I/O Hardware Logout - 0 to 48 bytes
Identifies the status of the failing channel when an I/O error interrupt occurred.

RECORD FORMATS

This section shows the format of the records included in a CHECKPOINT entry. These records are created and written by the CHECKPOINT service routine (SVC 63).

CHECKPOINT HEADER RECORD (CHR)

dec	hex				
0				Number of CHKPTs	CHECKID Length
4	4	CHECKID (left justified) (Checkpoint Entry Identification)			
20	14	DDNAME of CHECKPOINT Data Set			
28	1C	Lower Boundary of Problem Program Storage		Upper Boundary of Problem Program Storage	
36	24	CHKPT Blocksize	TIOT Length	CHECKPOINT Work Area Size	
44	2C	CHECKPOINT Work Area Address		CHECKPOINT SVRB Address	
52	34	Lower Boundary of IBM 2361 Core Storage		Upper Boundary of IBM 2361 Core Storage	

Note: The CHR is 400 bytes long and is padded with ones.

CORE IMAGE RECORD (CIR)

Problem Program Core

Direct copy of problem program storage, from the highest to the lowest address.

Blocksize 1. Is specified by the caller, or

2. If not specified, is the maximum for the device type.

DATA SET DESCRIPTOR RECORDS

Type 1 DSDR

```
 0       2                             178                              186             190
 ┌───────┬──────────────/ /─────────────┬────────────────────────────────┬────────────────┐
 │       │                             │                                │                │
 │X'0000'│             JFCB            │             DDNAME             │     UCBTYP     │
 │       │                             │                                │                │
 └───┬───┴──────────────/ /─────────────┴───────────────┬────────────────┴───────┬────────┘
     │                  │                                │                        │
     │                  │                                │                        │
     │                  │                                │                        │
 Type 1 DSDR         Job File                      DDNAME of the        Unit Control Block
 Identifier       Control Block                CHECKPOINT Data Set          Type Field
 (2 bytes)         (176 bytes)                      (8 bytes)               (4 bytes)
```

Type 2 DSDR

```
 0       2                             178
 ┌───────┬──────────────/ /─────────────┐
 │       │                             │
 │X'0004'│        JFCB Extension       │
 │       │                             │
 └───┬───┴──────────────/ /─────────────┘
     │                  │
     │                  │
     │                  │
 Type 2 DSDR    Job File Control
 Identifier      Block Extension
 (2 bytes)         (176 bytes)
```

Type 3 DSDR

```
 0       2                             178
 ┌───────┬──────────────/ /─────────────┐
 │       │                             │
 │X'0008'│            GDG BCT          │
 │       │                             │
 └───┬───┴──────────────/ /─────────────┘
     │                  │
     │                  │
     │                  │
 Type 3 DSDR   Generation Data Group
 Identifier       Bias Count Table
 (2 bytes)          (176 bytes)
```

Special Identifiers

```
 0       2                                   0       2
 ┌───────┐                                   ┌───────┐
 │       │  Indicates that the               │       │  Indicates that the
 │X'0010'│--previous DSDR is the             │X'0014'│--previous DSDR is the
 │       │  last one.                        │       │  last one in the block.
 └───────┘                                   └───────┘
```

SUPERVISOR RECORD (SUR)

```
0                                                            48 49    52
┌────────────────────────────────//──────────────────────────┬──┬──────┐
│                                                            │  │      │
│          First 48 bytes of the user's Task Control Block   │  │ FSA  │
│                                                            │  │      │
└────────────────────────────────//──────────────────────────┴──┴───┬──┘
                                                               │    │
                                                               │    │
                                                 Reserved ─────┘    │
                                                                    │
                              Address of the first user save area ──┘
```

```
52      56      60 61   64                          96      100     104
┌───────┬───────┬──┬────┬─────────────//────────────┬───────┬───────┐
│       │       │  │    │                           │       │       │
│  DCB  │  FQE  │FP│FQE │ Floating Point Registers  │ SYNAD │Return │
│       │       │  │    │                           │       │ Area  │
└───┬───┴───┬───┴┬─┴─┬──┴─────────────//────────────┴───┬───┴───┬───┘
    │       │    │   │                                  │       │
    │       │    │   └── Zero or Address of the first   │       │
    │       │    │       FQE for IBM 2361 Core Storage  │       │
    │       │    │                                      │       │
    │       │    └── X'01'  Floating point registers exist      │
    │       │        X'02'  Floating point registers do not exist│
    │       │                                           │       │
    │       └─── Address of the first FQE for processor storage │
    │                                                   │       │
    │                              CHKPT DCB SYNAD ─────┘       │
    └──────── Address of the CHKPT DCB                          │
                         Address of the return area for CHECKID ┘
```

```
104             112     116                                        156
┌───────────────┬───────┬─────────────────//────────────────────────┐
│               │       │                                           │
│    CHECKID    │Offset │          Tape SYSOUT Information          │
│               │       │                                           │
└───────┬───────┴───┬───┴─────────────────//────────────────────────┘
        │           │
        │           └── Displacement from the beginning of the SVC
        │               transient area to the next sequential instruction
        │               for the end-of-volume (EOV) routine, IGG0551A
        │
        └────────────── Checkpoint entry identification name
```

CHECKPOINT/RESTART SVC MODULE LIST

The table below shows the entry point name, functional name, object module name, and CSECT name for each of the CHECKPOINT and RESTART transient SVC routines.

Load modules use a work area in problem program storage to communicate with each other. The address of this work area is passed in register 1. The modules are listed in the order executed.

Object Module Name and Microfiche Name	Name of Routine	Control Section Name and Entry Point Name
IHJACP00	CHECKPOINT ROUTINE--HOUSEKEEPING 1	IGC0006C
IHJACP01	CHECKPOINT ROUTINE--HOUSEKEEPING 2	IGC0106C
IHJACP02	CHECKPOINT ROUTINE--HOUSEKEEPING 3	IGC0206C
IHJACP10	CHECKI/O	IGC0506C
IHJACP20	PRESERVE 1	IGC0A06C
IHJACP25	PRESERVE 2	IGC0C06C
IHJACP30	CHECKMAIN	IGC0F06C
IHJACP40	RESUME I/O	IGC0N06C
IHJACP50	CHECKPOINT EXIT ROUTINE	IGC0Q06C
IHJACP70	CHECKPOINT MESSAGE ROUTINE	IGC0S06C
IHJARS00	RESTART HOUSEKEEPING 1	IGC0005B
IHJARS01	RESTART HOUSEKEEPING 2	IGC0105B
IHJARS20	REPMAIN	IGC0505B
IHJARS40	JFCB PROCESSOR 1	IGC0G05B
IHJARS41	JFCB PROCESSOR 2	IGC0I05B
IHJARS43	MOUNT/VERIFY NON-DIRECT ACCESS	IGC0K05B
IHJARS45	MOUNT/VERIFY DIRECT ACCESS	IGC0M05B
IHJARS4C	SYSIN/SYSOUT NON-DIRECT ACCESS PROCESSOR	IGC0L05B
IHJARS4D	SYSIN/SYSOUT DIRECT ACCESS POSITIONING 1	IGC0N05B
IHJARS4E	SYSIN/SYSOUT DIRECT ACCESS POSITIONING 2	IGC0Q05B
IHJARS47	NON-DIRECT ACCESS POSITIONING	IGC0P05B
IHJARS49	DIRECT ACCESS POSITIONING	IGC0R05B
IHJARS4B	FINAL PROCESSING MODULE	IGC0T05B
IHJARS60	RESTART EXIT ROUTINE	IGC0V05B

• CHECKPOINT/RESTART REGISTER USAGE TABLE

	Input to Module					Internal Use					Output from Module			
Modules	1	3	4	5	8	2	3	6	11	12	1	5	8	15
IHJACP00	A[1]	B	C	D					E	F	F[2]	[2]		
IHJACP01	G								E	F	F			
IHJACP02	G								E	F	F			
IHJACP10	G					E		F			F			
IHJACP20	G								E	F	F			
IHJACP25	G								E	F	F			
IHJACP30	G								E	F	F			
IHJACP40	G					E		F			F			
IHJACP50	G[2]			[2]					E	F	H			[3]
IHJACP70	H								E	F				J
IHJARS00	A	B	C	D					E	K	K			
IHJARS01	K								E	K	K			
IHJARS20	K								E	K				
IHJARS40	K						E				K		L	
IHJARS41	K				M		E				K		L	
IHJARS43	K				M		E				K		L	
IHJARS45	K				M		E				K		L	
IHJARS4C	K				M		E				K		L	
IHJARS4D	K				M		E				K		L	
IHJARS4E	K				M		E				K		L	
IHJARS47	K				M		E				K		L	
IHJARS49	K				M		E				K		L	
IHJARS4B	K				M		E				K			
IHJARS60	K								E	K				J

[1]If CHKPT CANCEL, R1 contains all zeros
[2]If an error in Module IHJACP00, R1 contains all zeros and R5 addresses CHECKPOINT'S SVRB
[3]If CHKPT CANCEL, R15 contains return code

A - parameter list address	G - address of CHKPT work area
B - CVT address	H - address of CHKPT's SVRB
C - TCB address	J - return code
D - SVRB address	K - address of RESTART WORK AREA
E - base register	L - address of data set entry in work area
F - address of CHKPT work area	M - address of data set entry in work table

Where more than one page reference is given, the first page number indicates the major reference.

GY28-6612-4

Printed in U.S.A. GY28-6612-4

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]

READER'S COMMENT FORM

IBM System/360 Operating System; Fixed-Task Supervisor
Program Number 360S-CI-505

GY28-6612-4

Please check or fill in the items below, adding explanations and other comments in the space provided.

Which of the following terms best describes your job?

⌶ Programmer	⌶ Systems Analyst	⌶ Customer Engineer
⌶ Manager	⌶ Engineer	⌶ Systems Engineer
⌶ Operator	⌶ Mathematician	⌶ Sales Representative
⌶ Instructor	⌶ Student/Trainee	⌶ Other (explain) ____________

Does your installation subscribe to the SRL Revision Service? ⌶ Yes ⌶ No

How did you use this publication?

⌶ As an introduction
⌶ As a reference manual
⌶ As a text (student)
⌶ As a text (instructor)
⌶ For another purpose (explain) ______________________________

Did you find the material easy to read and understand? ⌶ Yes ⌶ No (explain below)

Did you find the material organized for convenient use? ⌶ Yes ⌶ No (explain below)

Specific criticisms (explain below)

Clarifications on pages ______________________________

Additions on pages ______________________________

Deletions on pages ______________________________

Errors on pages ______________________________

Explanations and other comments:

Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

GY28-6612-4

YOUR COMMENTS PLEASE . . .

This manual is one of a series which serves as reference sources for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

FOLD FOLD

FIRST CLASS
PERMIT NO. 116
KINGSTON, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
NEIGHBORHOOD ROAD
KINGSTON, N. Y. 12401

ATTN: PROGRAMMING PUBLICATIONS
DEPARTMENT 637

FOLD FOLD