POUGHKEEPSIE PROGRAMMING CENTER

INSTALLATION GUIDE

ABDUMP

DEBUGGING PROCEDURES

Third Version
August, 1966

PREFACE

This chapter was written assuming the user of OS/360 has coded a program and faced with the task of debugging the program using the ABEND hex dump (ABDUMP). Areas such as System Control Flow, RB Queues and the trace area will be explained and explored as to their use in debugging. Other sections of this chapter deal with evaluating I/O error messages and status of DASD data sets. This chapter will hopefully provide all the diagnostic aids and supporting literature necessary to evaluate OS/360 debugging facilities in the Primary Control Program (PCP).

This document assumes that the reader is a programmer who has a general knowledge of OS/360 logic flow. A recommended prerequisite to provide this general system knowledge is the IBM Operating System/360 Concepts and Facilities manual C28-6535, or Part 1 of the IBM publication; "Introduction to Control Program Logic," Z28-6605. At specified points within the document it is recommended that Appendix B of the IBM Messages and Completion Codes Manual C28-6608, be read to supplement this manual.

DEBUGGING PROCEDURES

Table of Contents

I. INTRODUCTION

Prior to going in and examining the different facets of the ABEND dump it is necessary to understand the system control flow of the OS/360 control program. This understanding is essential in order to comprehend and evaluate the full ABDUMP.

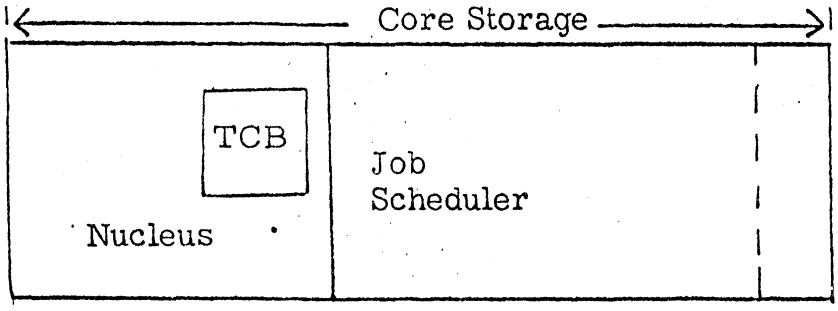
Whenever possible, actual dump examples have been included to supplement the text. However, due to size limitations it is impossible to include examples in all areas. For this reason it is suggested that the reader supplement the text using his or her own ABDUMP.

II SYSTEM CONTROL FLOW

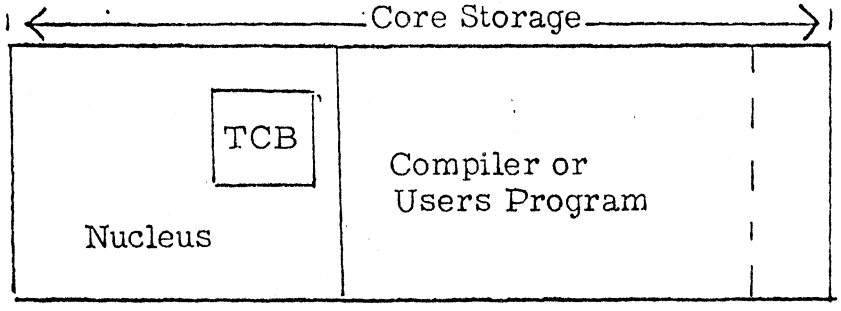
A. Task Control Block (TCB) and its associated request blocks (RB).

1. What is a TCB?

The task control block is the operating system's way of keeping, in one location, pointers to all pertinent information about the job step and consequently the task that the job scheduler has scheduled. In the primary control program (PCP) there is only one TCB in the system. It is located in the nucleus and used by all programs that reside in the problem program as shown in Figures 1a and b.



a. TCB associated with job scheduler



b. Reinitialized TCB associated with problem program.

Figure 1

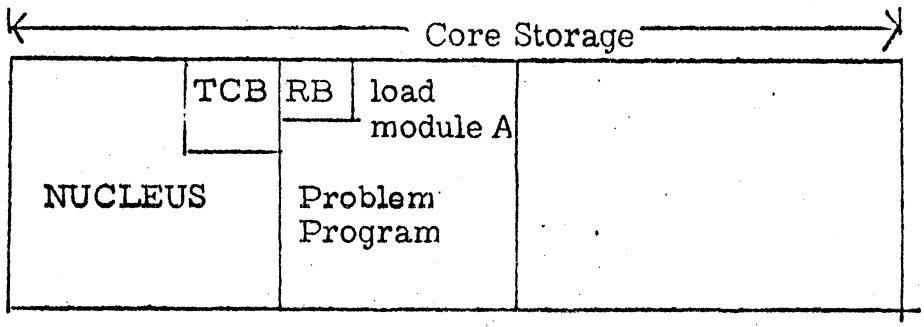
This one TCB is reinitialized every time the job scheduler completes its scheduling of a job step and control is given to the problem program. A similar action occurs when the problem program completes its function and specifies the RETURN macro.

2. What type of control information is kept in the TCB?

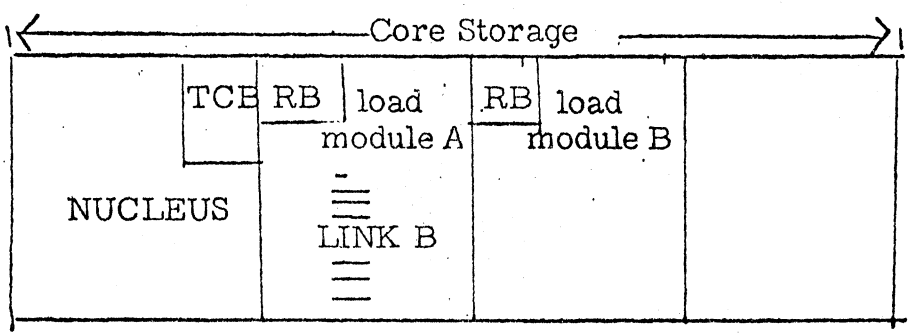
The TCB primarily contains pointers to other system blocks imbedded in the control program nucleus. For instance, one can determine what I/O devices have been allocated to this job step by locating the Task I/O table (TIOT); which data sets are open by checking the DEB list; determine the end of the nucleus by looking in the main storage supervisor's boundary box. All of these system created tables and control blocks are pointed to by the TCB. How each of these fields are used as diagnostic aids will be covered later.

3. What is a Request Block (RB) and why is it needed?

It is very possible that all the routines for any one problem program or job scheduler will not be brought into core with the initial load module. The use of the LINK, XCTL, ATTACH and LOAD mocro used the OPEN routine to dynamically bring in the access routines, is a good example of this. This dynamic loading capability forces the control program to add a control block, in addition to the TCB, called a request block (RB). This request block retains control information at the load module level. One of these RB's is created by the control program whenever a dynamic request to fetch a load module for execution is given. The RB is located in problem core as indicated in Figure 2a.



a. RB located in problem core



b. RB created for load module B

Figure 2

Figure 2b shows the relative position of a second RB which was created when load module A issues a LINK macro. The control program nucleus responded to the LINK macro by fetching load module B into core and creating an RB to retain control information about this module.

4. What kind of control information is kept in an RB?

The content of the RB is an important element of the debugging procedure. Its content and size vary depending on what type of RB it is. If we assume that the RB is for a load module that was LINKed to, as in Figure 2, it would be 32 bytes in size and would contain such things as the member name or alias that this load module was fetched by; the size of this load module and its RB; and the PSW as it existed when control was passed to the nucleus the last time. How each of these fields are used to aid in debugging is covered later.

5. How does the system know which load module to give control to?

The question is very valid. Prior to this, the discussion centered around the TCB and the RB, but not how the two are interconnected. This is a good point to introduce the two request block queues that the system uses to control what load module gets control of CPU time. They are called the Active RB Queue and the Load List. The next section describes their actions.

B. RB Queues

As indicated earlier, there are two RB queues that the system creates and maintains. These queues provide the means by which the primary control program keeps track of and allocates two very important resources - CPU time and load modules (programs) currently in core.

1. Active RB Queue

Prior to developing the active RB queue, it is necessary to explain the TCB - RB relationship. To clarify this subject, it is convenient to initially assume that; the job scheduler has been in core, read the job control language (JCL) defining the job step; allocated the I/O devices and requested the control program, via an XCTL macro, to fetch load module A. The control program nucleus has created an RB for load module A, fetched the module into core storage and given control of CPU to module A. At this point in time, core storage contains the nucleus, load module A, and its RB in lower core. With the exception of a system table in upper core, the rest of core is free for allocation as indicated in Figure 3.

Load module A was defined to the system via an EXEC PGM=A card in the input job stream. It should be noted that the system always places the RB, associated with load module (program A), on the first doubleword boundary outside the nucleus (assuming no storage protect). With storage protect the first RB is placed on the first 2048 byte boundary outside the nucleus. The load module A is contiguous with the request block.

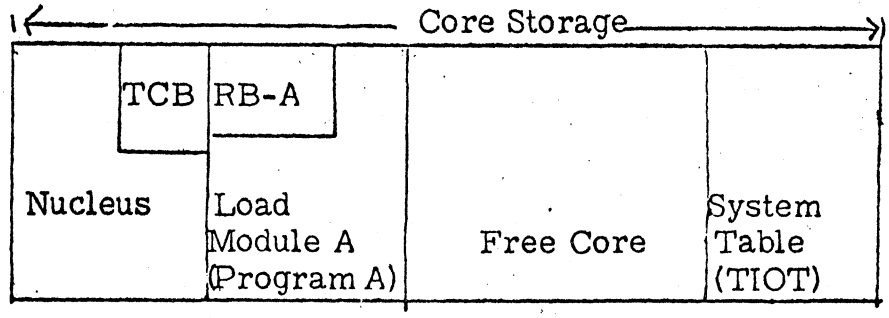


Figure 3

Logically, the TCB and RB are linked up as shown in Figure 4. The TCB always contains a pointer to the RB whose associated load module has control. In our example, only one load module is in core, so the address of its request block is placed in the TCB. This pointer in the first RB contains a pointer to either a previous RB or the TCB. In our example no previous RB's exist, so this field does point back to the TCB. Additional fields, such as the member name and entry point are shown to reflect some of the control information that is retained in the request block.

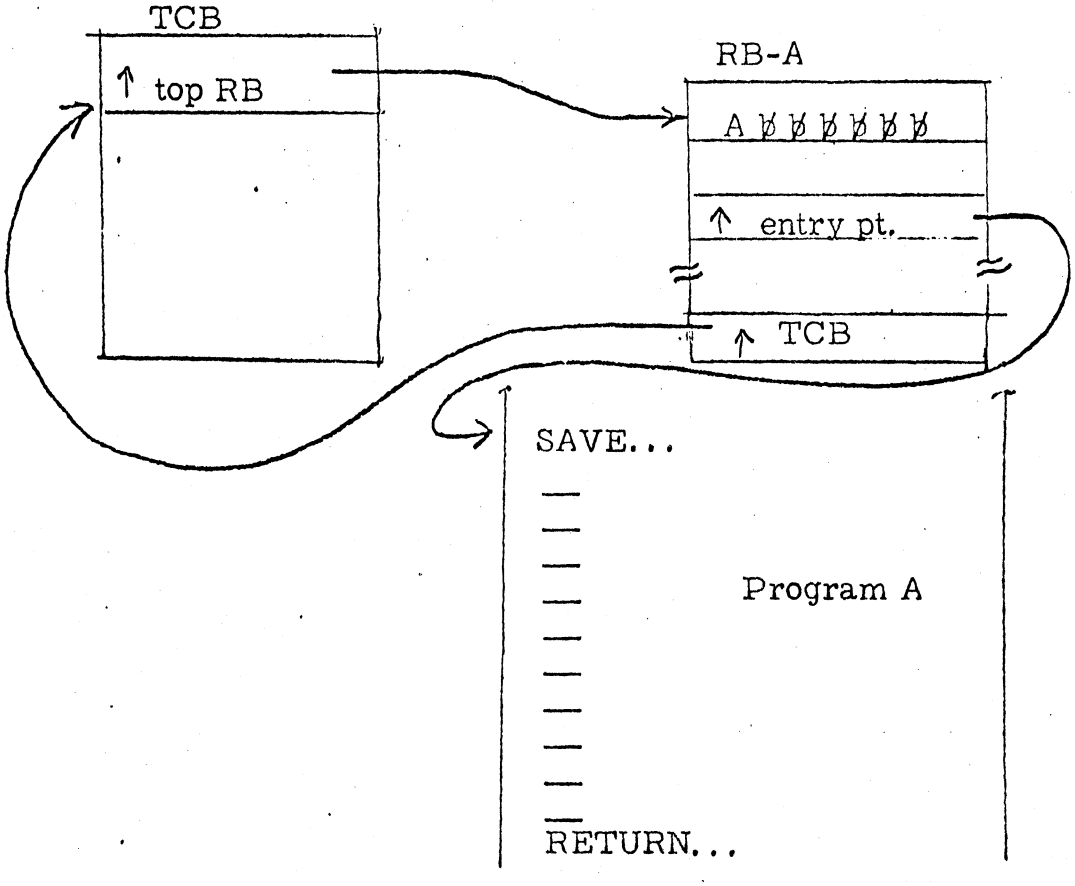


Figure 4

The linking of the TCB and RB together creates the beginning of the active RB queue. By definition this is a queue of request blocks that keep track of active load modules (programs) that have been LINKed, XCTLed, or ATTACHed.

To elaborate on this, one must expand the previous example. Assume that program A now LINKs to program B. The LINK macro, when assembled, degenerates into an SVC 6 which causes an interrupt and gives control to the nucleus. Figure 5 shows the updating of the control block pointers that takes place prior to giving control to program B.

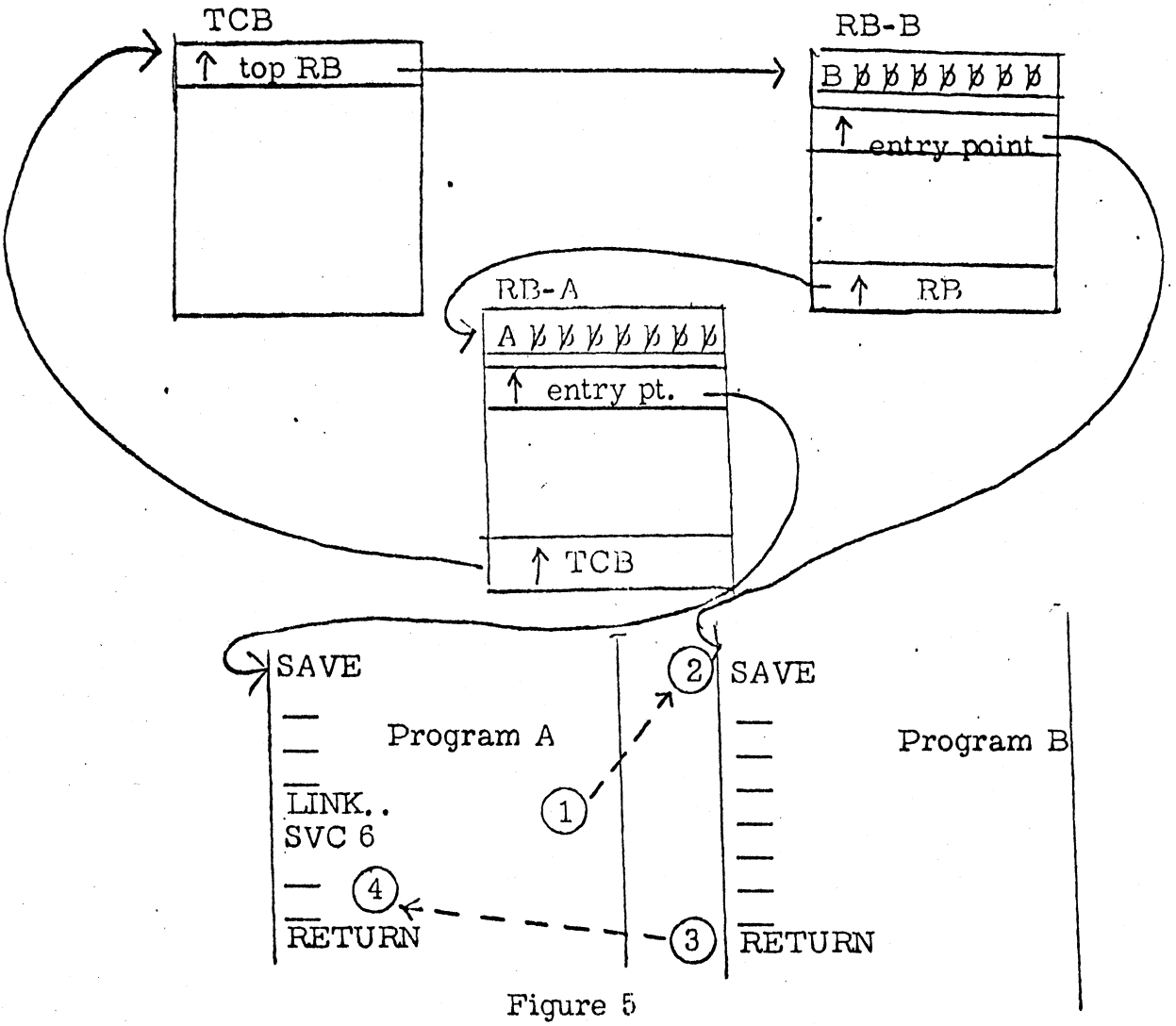


Figure 5

At time (2) when Program B gains control of CPU, the active RB queue is connected as described below. The top RB pointer in the TCB points to the RB - B (whose program is in control). The link field in RB - B points back to RB - A, whose associated program will regain control when Program B completes. The link field in RB - A points to the TCB because it is the first RB on the queue. A snapshot of core at time (2) would look like Figure 6.

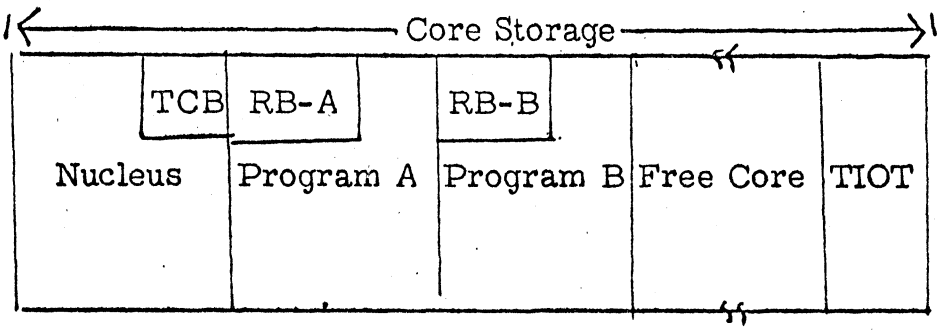


Figure 6

a. How does the nucleus know where to return control to in Program A?

Referring to time (1), Figure 5, when the SVC 6 causes an SVC interrupt, the nucleus retains the SVC old PSW in the request block for program A. This 8 byte field in RB - A, called RESUME PSW, will therefore contain all pertinent information about Program A's resumption point. The nucleus, between times (3) and (4), performs an LPSW instruction specifying the Resume PSW field of RB - A and Program A regains control at the proper point.

2. The Load List

The load list, by definition, is a queue of request blocks that keeps track of load modules that have been fetched into core via the LOAD macro. These programs or load modules will remain in core until either a DELETE macro is issued for them, or job step termination occurs. Modules that are LOADed and their associated RB's, are fetched into the upper end of core storage. As you recall, modules that are XCTLed, ATTACHed, or LINKed to are fetched into the lower end of core. This tends to keep contiguous free core between them as shown in Figure 7.

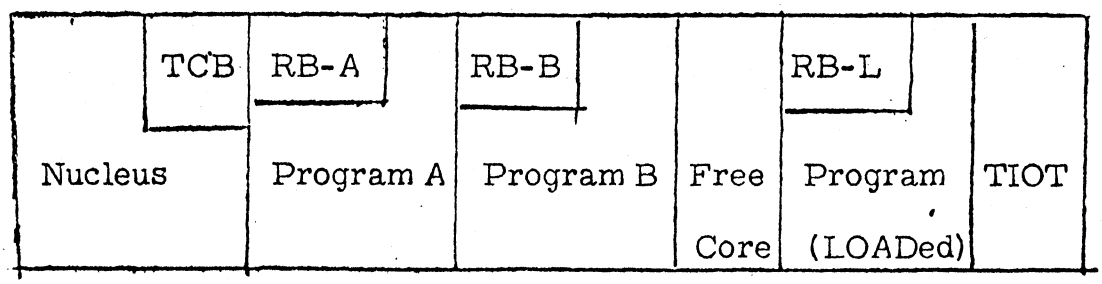


Figure 7

The linking of the TCB and the RB's on the load list differ slightly from the active RB queue. To understand the need for a different type of queue, one must understand the logic behind the LOAD and DELETE macro. Upon issuing the LOAD macro, the nucleus fetches the requested load module, creates an RB, and initializes a load list as indicated in Figure 8.

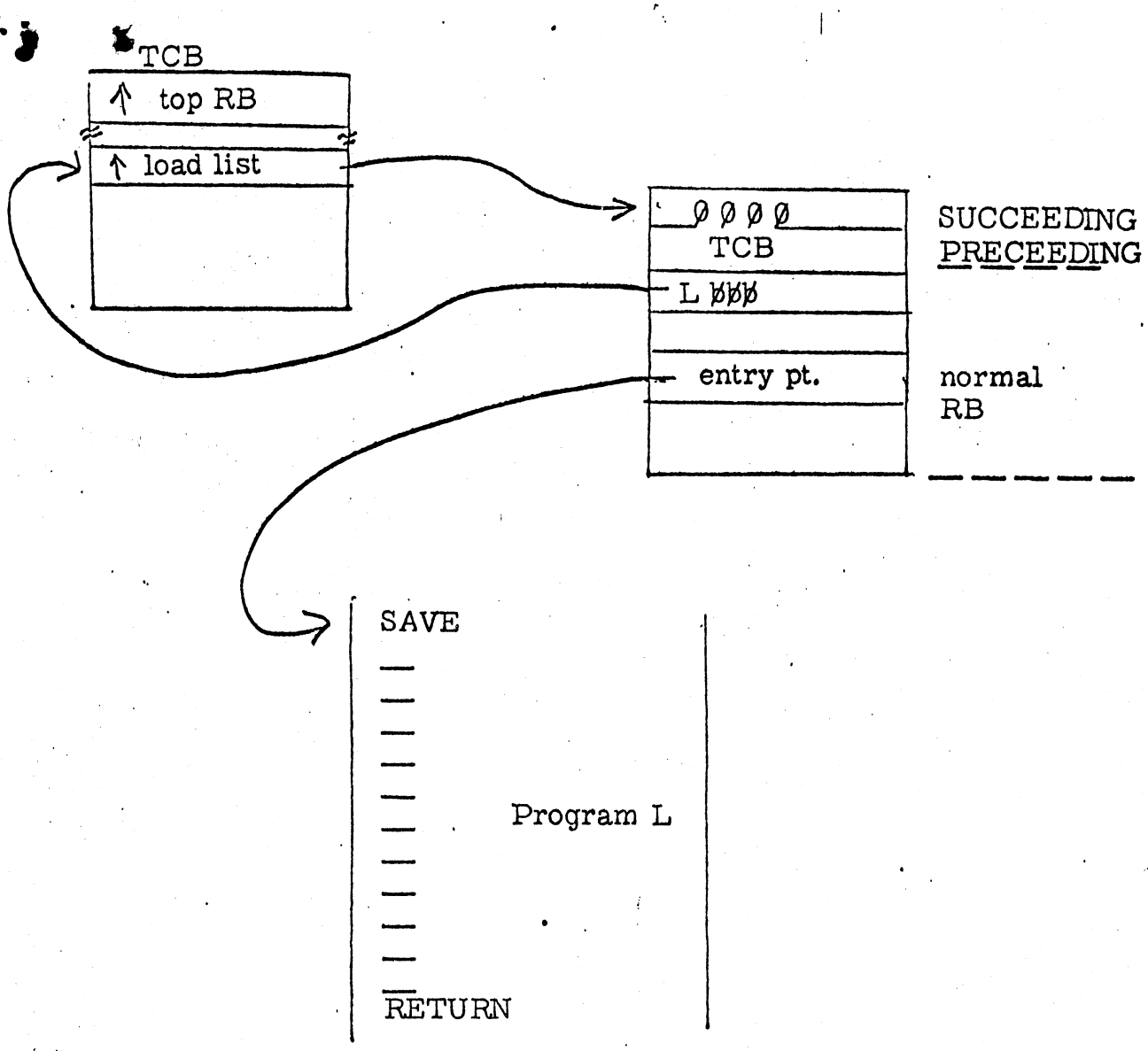


Figure 8

The nucleus then passes back to the issuer, in Register 0, the entry point to the requested routine. At this point the issuer of the LOAD macro regains control and may, at will, branch to this loaded routine, using the entry point passed to him. An important fact is that the nucleus does not know when the user is operating in the loaded routine and therefore cannot delete it.

The system must wait for a DELETE macro to be issued by the user or step termination to take place before freeing up the RB and core associated with a LOADed program.

Upon issuing of the DELETE macro, the nucleus gains control, via an SVC interrupt, and searches the Load List for the specified load module. It uses the two pointers of the extended RB, shown in Figure 8, to perform this search and find the requested load module. The SUCCEEDING pointer contains zeros. The PRECEEDING pointer of the RB points to the previous RB on the Load List. This field in the first RB on the Load List points back to the Load List field in the TCB. The purpose of the two pointers is evident when one considers that the system must be able to DELETE load modules whose RB's are in the middle of the list. Both pointers are necessary to delete the RB and link both ends of the load list back together again.

a. Does the Primary Control Program use the LOAD macro to bring in any of its routines?

   Yes, a good example of the use of the LOAD macro is its use by the OPEN routine. Access method routines are brought into core at OPEN time via the LOAD macro. At CLOSE time the DELETE macro purges these routines if they are not in use.

b. How could these access method routines be in use?

   When initially LOADed at OPEN time these modules are brought into core and a one (1) is placed in the USE COUNT field of the RBs associated with these modules. If a second data set is OPENed, specifying the same access method or requesting the same load modules, the control program simply increments the USE COUNT in the RB, passes back the modules entrypoint in Register 0 and does <u>not</u> fetch another copy. At CLOSE time the DELETE macro decrements the USE COUNT in the RB and <u>only</u> if it goes to zero, purges the module and frees up the core. Otherwise, the module would remain in and be purged at the CLOSE of the second data set.

C. System interaction between the two queues

The Active RB Queue and the Load List make up the contents directory for the sequential system. Depending on which macro is given, XCTL, ATTACH, LINK or LOAD, the system reacts differently as to the queues it checks before it fetches another load module. Let's take a few examples to point out the differences.

1. Figure 9 shows the resulting active RB queue after Program B LINKs to Program C.

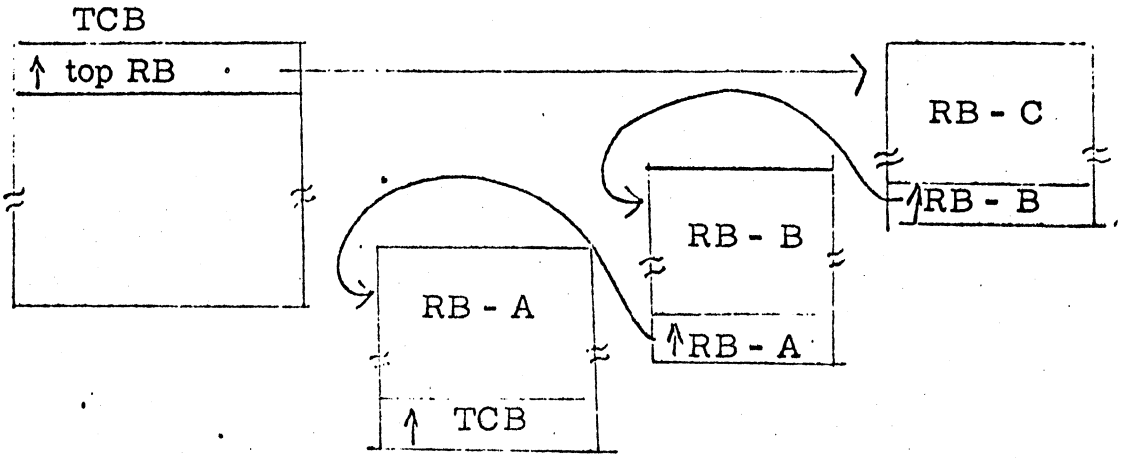


Figure 9

Upon return from Program C, its RB and associated load module is purged and the core freed up. The same action takes place upon the return from Program B to A. At that point the Active RB Queue looks like Figure 10.

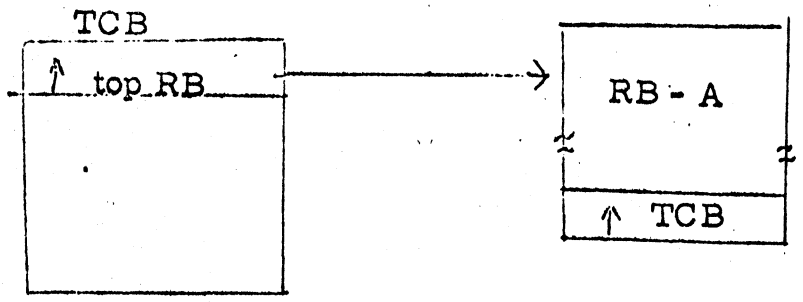


Figure 10

If at this point Program A LINKed to B again, the following sequence takes place. The Load List is searched for B; if not found, a new copy of load module B is brought into core; an RB created; and the Active RB Queue updated accordingly. Control then passes to B. If load module B has an RB queued on the load list, then the primary control program (PCP) determines if the module is usable. If it is, the same RB that is queued on the load list, is queued on the Active RB Queue and control is given to load module B as shown in Figure 11. The control program determines whether a load module is usable or not by evaluating the STATUS field of the associated RB. How this STATUS field is set will be expanded upon later when RB types are covered.

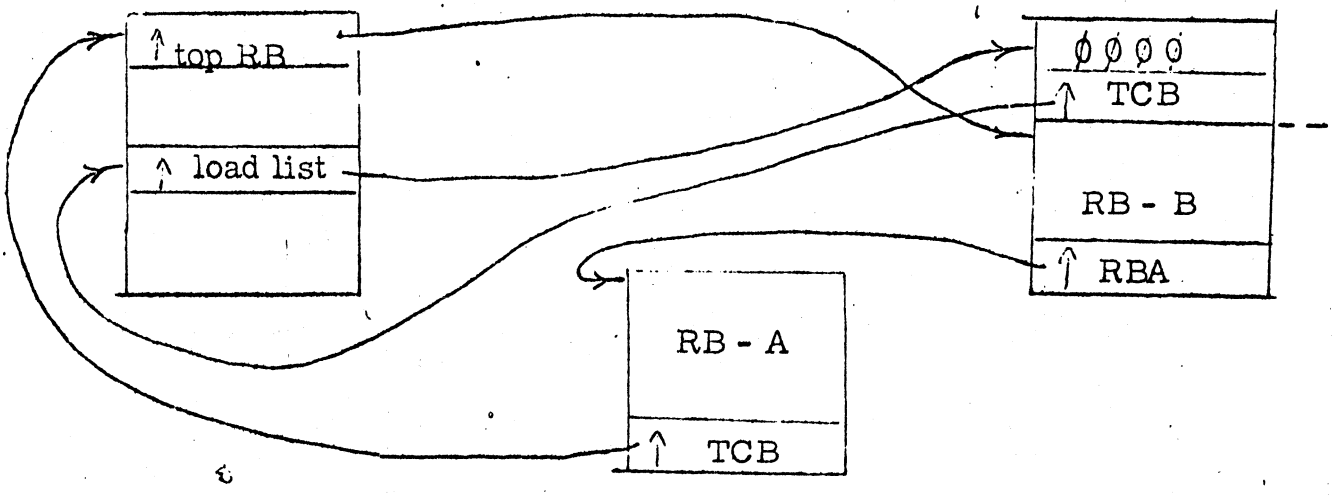


Figure 11

2. How does the issuing of an XCTL macro differ from the LINK macro we just talked about?

XCTL's effect on the system is best explained by referring back to the in core situation indicated in Figure 6. Basically Program A LINKed to Program B. Let's assume that the programmer wished to execute Program C and then RETURN to Program A without going back to Program B. XCTL provides this capability by logically overlaying Program B with Program C. Upon returning from Program C processing is resumed in Program A. At the point when the XCTL macro is issued in Program B the system blocks are queued as shown in Figure 12.

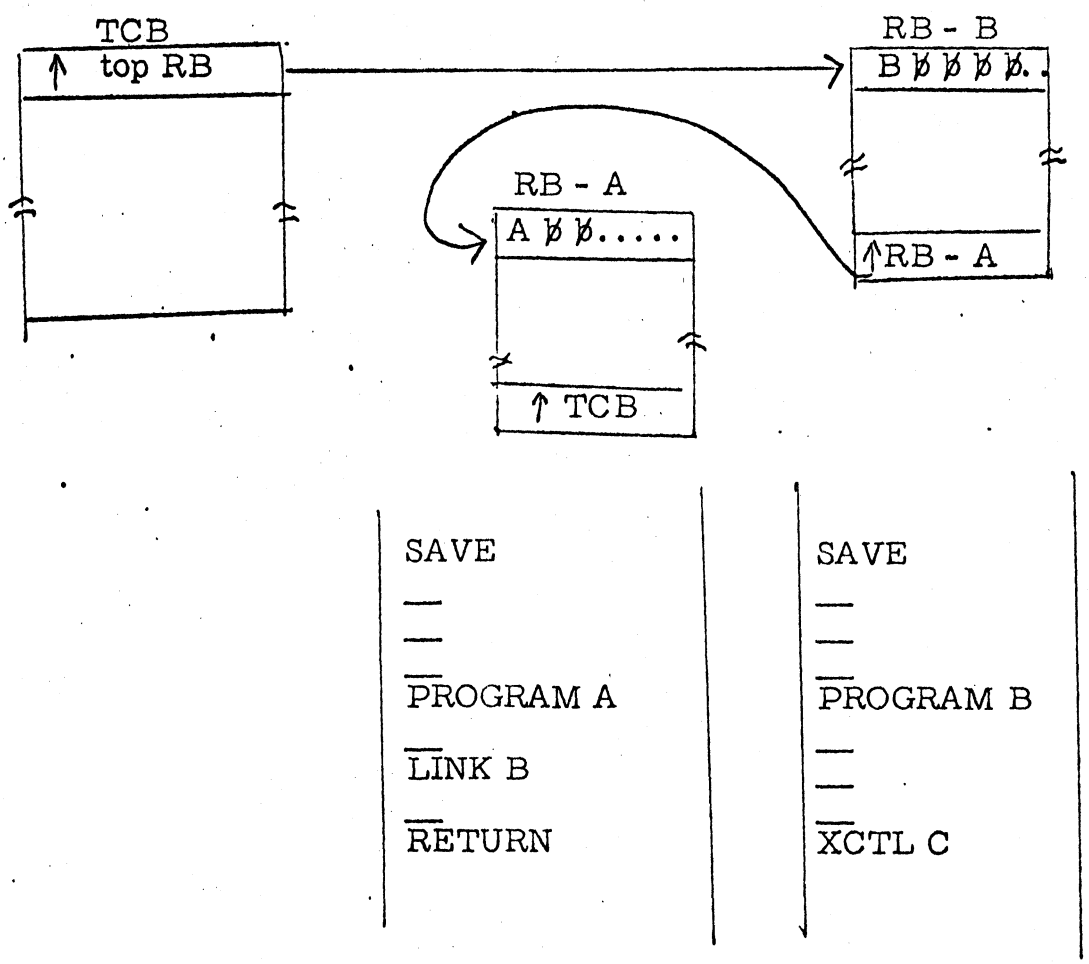


Figure 12

The following events occur in the nucleus upon issuance of the XCTL macro. It, like LINK, degenerates into an SVC interrupt allowing the nucleus or control program to gain control. The first function performed is to search the Load List for load module C. If found and usable, RB - C is queued on the Active RB Queue; RB - B and load module B are purged and its core is freed; and control is given to C.

If load module C is not on the load list, program B and its RB is purged leaving program A and its RB as the only routine in user core. The nucleus then fetches module C, creates an RB, and chains it on to the Active RB Queue as indicated in Figure 13.

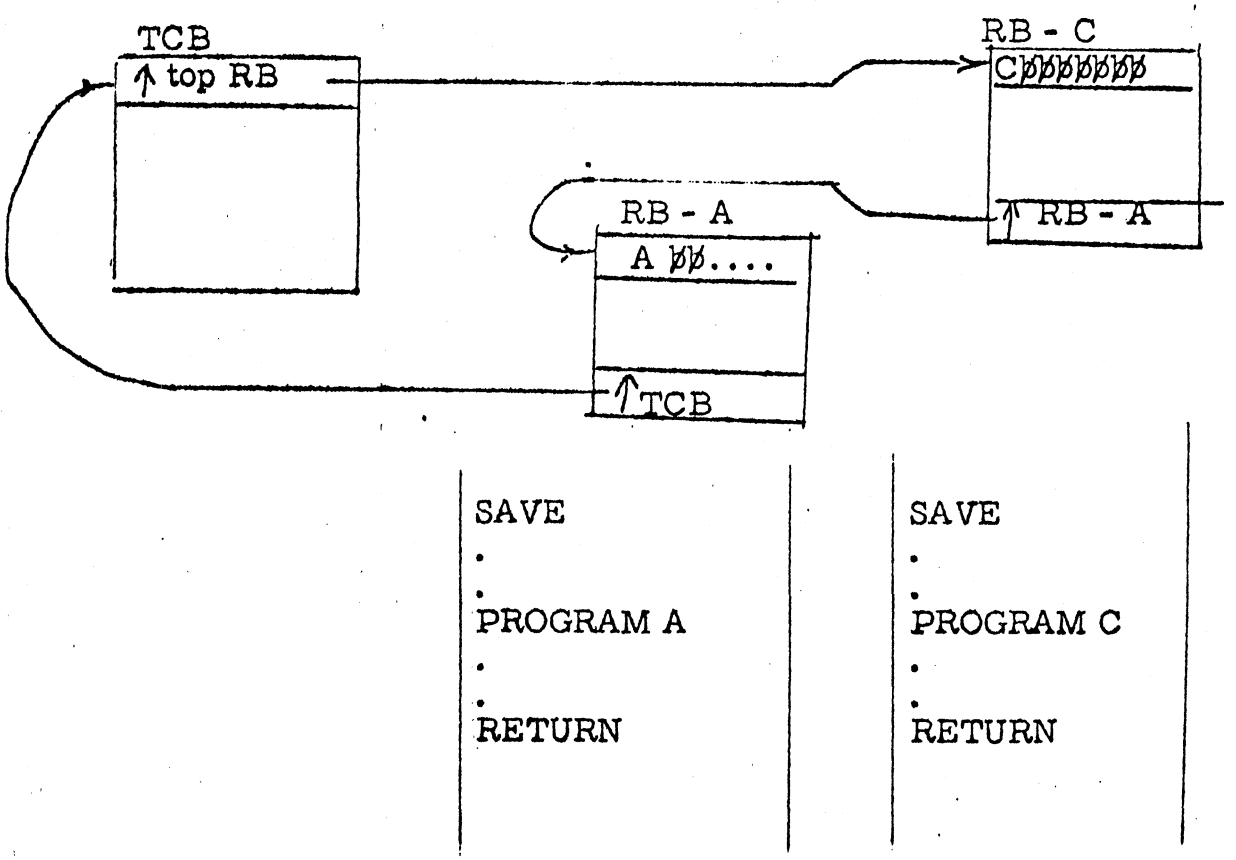


Figure 13

Figure 14 reflects a snap shot of core as it exists while executing program C.

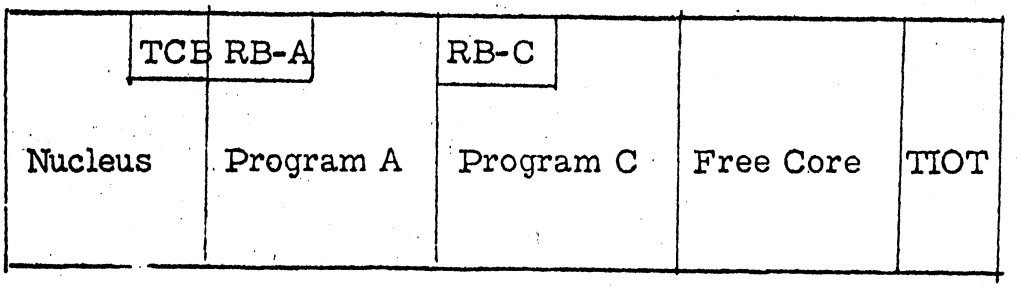


Figure 14

The important thing to note is when an XCTL macro is issued another level of request block is not created as is with a LINK or ATTACH macro.

3. How is an ATTACH handled on the primary control program?

The ATTACH macro normally initiates a section of the overall program, called a subtask, that can be processed in parallel with the main program asychronously. However, in the primary control program only one task control block exists. It is impossible to dynamically create another TCB, and impossible to process a subtask asychronously. The next best thing is to allow the use of the ATTACH macro, when looking ahead to the multitask operating system, and to perform the ATTACHed routine serially. This is what the primary control program does. It performs the ATTACHed routine much like a LINK with a few exceptions. The ATTACH macro allows specification of an exit routine (EXTR parameter) and the posting of an event control block upon completion of the ATTACHed routine. Both the exit routine and ECB are located within the ATTACHing load module. These features are handled by placing additional request blocks on the active RB Queue as shown in Figure 15.

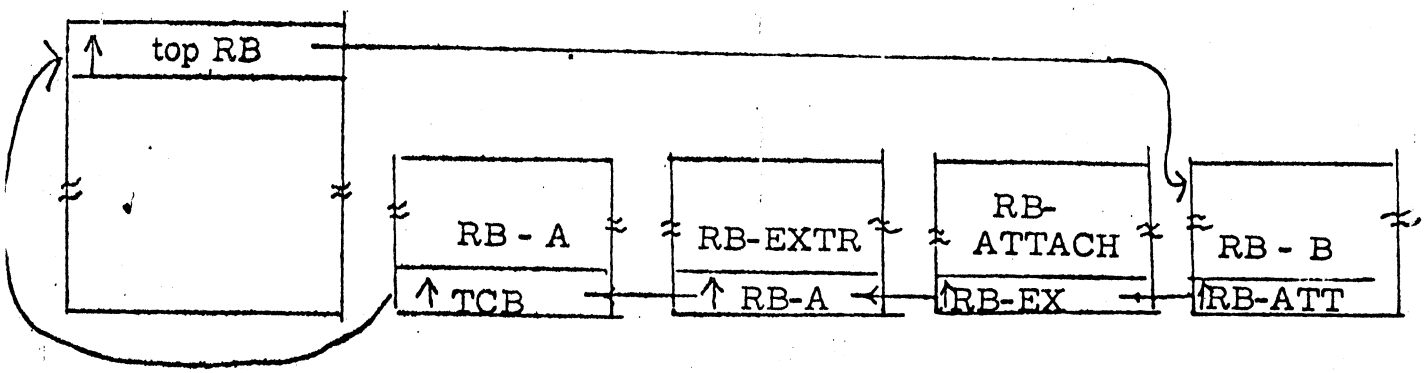


Figure 15

In the example shown (Figure 15), program A ATTACHes program B specifying an exit routine to enter upon completion of program B, and an event control block within program A to be posted upon completion of B. The system makes the normal search of the Load List as the LINK routine did. If the requested program is not found, it is fetched into core and the Active RB Queue is updated as shown in Figure 15. Program B gains control of CPU. Upon completion of program B, the ATTACH routine in the nucleus would receive control and POST the event control block indicated. Control would then be given to the EXTR routine. Upon RETURN from the EXTR routine control again passes to program A to resume processing.

5. Summary of Sequential Program Execution

Program A was fetched into core as the result of the Job Scheduler reading an EXEC PGM=A control card. The logical sequence of control flow that occurs, starting at IPL time, is something like the following:

a. By depressing the IPL key on the console and specifying the SYSRES device in the load address keys, the computing system reads in an IPL Bootstrap record. This Bootstrap record, consisting of a chain of channel commands, reads in the IPL PROGRAM LOADER. At this point the load light on the console goes out and control is passed to this IPL Program Loader. Its function is to read in the nucleus plus the nucleus initialization program (NIP). NIP and the nucleus are combined as one load module and written on the system residence device at system generation time.

b. Once the nucleus and NIP are in core, NIP gains control and proceeds to initialize the nucleus. Just prior to NIP's completion the Active RB Queue would look similar to Figure 16.

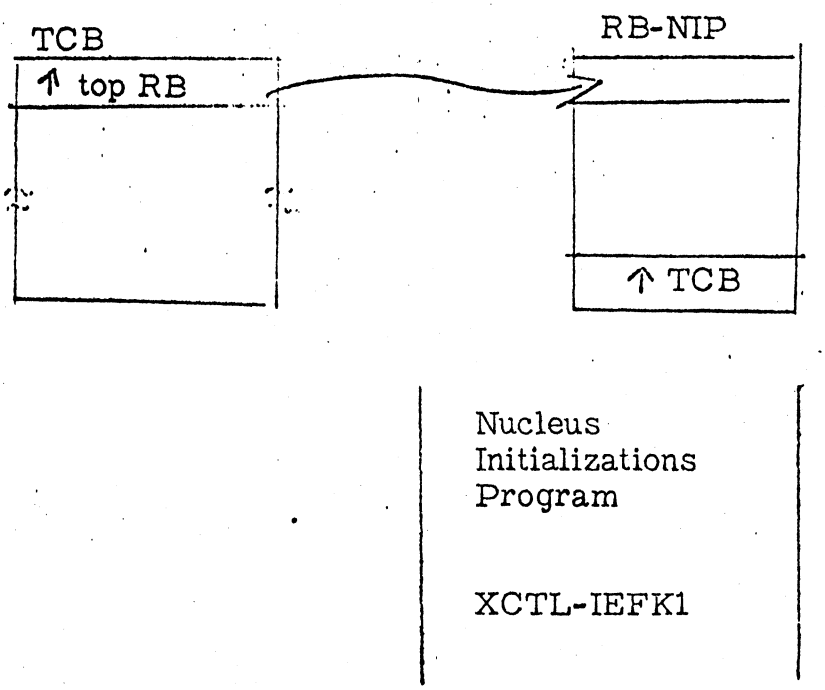


Figure 16

NIP would then logically XCTL to the master scheduler function of an XCTL, NIP will be overlayed by the master scheduler. The master scheduler would then issue the READY message and wait for the SET DATE command to be entered at the console. Upon receiving this command from the operator, the system will then issue the automatic START RDR, START WTR commands and wait on changes to these commands or/and a START command. The master scheduler XCTL's to the reader interpreter. At this point the system blocks logically looks like Figure 17.

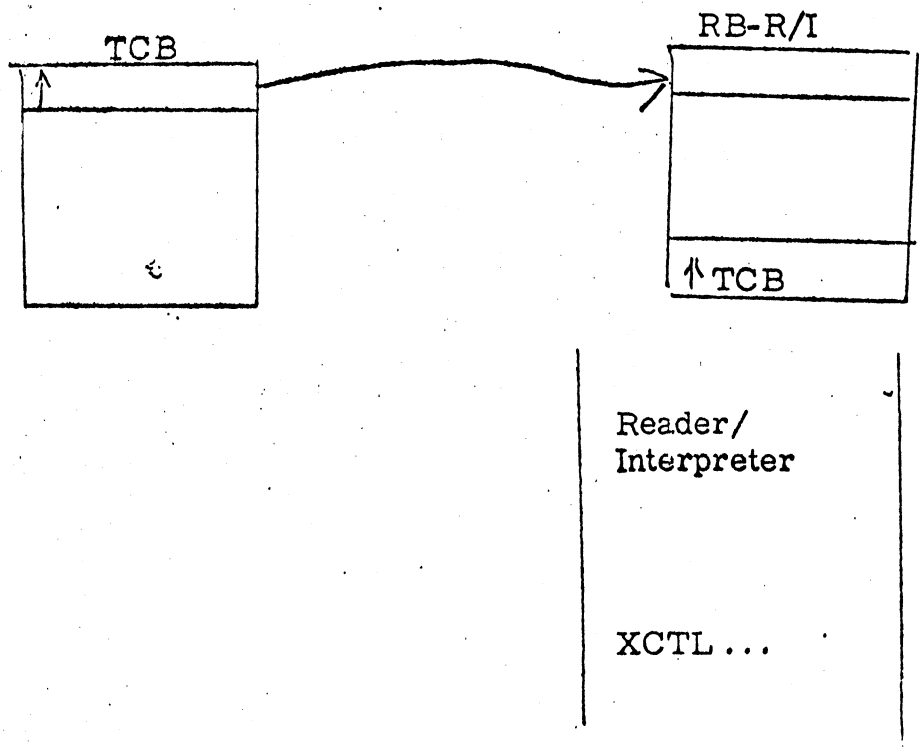


Figure 17

The Reader Interpreter opens the system input and output devices and proceeds to read the input stream. It then creates the system tables necessary for job initiation and I/O device allocation. The Reader/Interpreter will continue to read the input stream and create tables until DD* card, null card or another JOB card is read. Upon recognizing one of these, it XCTL's to the Initiator. System blocks at this point in the job step initilation cycle logically looks like Figure 18.

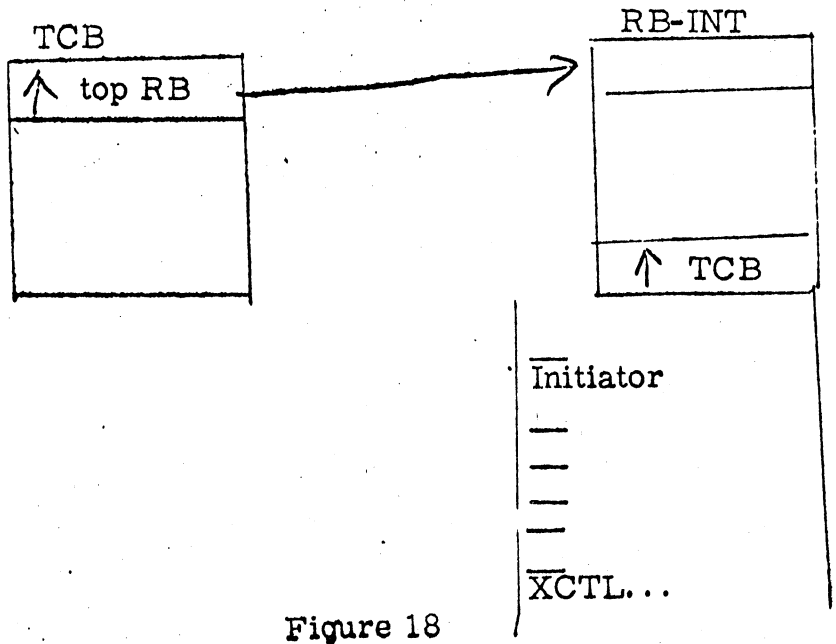


Figure 18

The Initiators primary function is to allocate I/O devices to the job step and obtain DASD space. Once the allocation of devices is complete the Initiator builds a task I/O table (TIOT) which lists the ddname and a pointer to the assigned device or devices for each data set of this job step. This TIOT table is placed in upper core and is the system table that has been referred to on previous core snapshots. By using this table, one can determine what devices the job scheduler has assigned to user data sets. The Initiator then XCTL's to the load module specified in the EXEC card. In our example it was program A.

Upon completion of the problem program (A in our example) the RETURN macro will return control to the nucleus and it, in turn, will XCTL to the termination section of the initiator. This phase of the job scheduler takes care of the disposition of the data sets and frees up the proper I/O devices. It would then initiate the next job step or XCTL to the Reader Interpreter to read in the next JCL. The whole sequence starts again.

Logically, this past sequence of events is how the system blocks are used both by the job scheduler and the problem program. Due to the different means of packaging the job scheduler the names of the modules and the number of XCTL's will vary. The sequence shown is for logic purposes only. Refer to the Job Management PLM for actual module names and number of loads.

D. RB Types

Up to this point the assumption has been that there is one type of request block. In reality there are six different types of request blocks. The RB type, its size and the fields used within it varies depending on the routine or load module it is associated with. To be able to analyze the RB queues, one must be able to recognize the different types and and know their function. Figure 19 shows the fields and core size required by the six different types of request blocks.

REQUEST BLOCK (RB) FORMAT

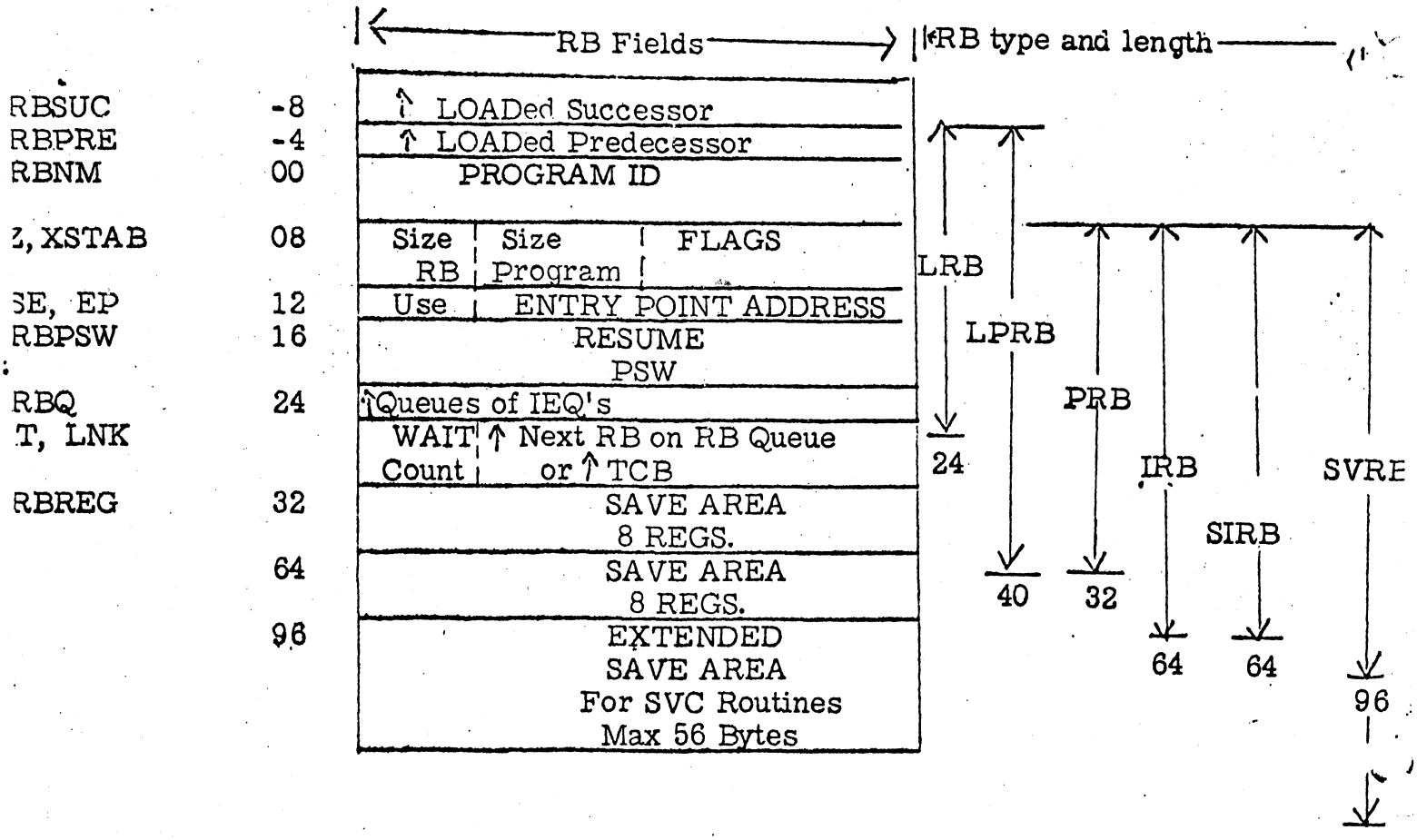


Figure 19

LRB - Loaded Request Block
LPRB - Loaded Program Request Block
PRB - Program Request Block
IRB - Interrupt Request Block
SIRB - Supervisor Interrupt Request Block
SVRB - Supervisor Request Block

1. Program Request Block (PRB)

The system initially starts out with one PRB and it is associated with NIP as indicated in an earlier section. This RB is the most common and primarily the one described in the previous examples. PRB's are created by the nucleus whenever an XCTL, ATTACH, or LINK macro is issued. This request block is 32 bytes in size and always contiguous to the load module that it is associated with.

2. Supervisor Request Block (SVRB)

The supervisor or nucleus when it gains control of CPU executes its code in two basic operational modes. The first operational mode is when an interrupt occurs, the supervisor executes a routine which performs some function, and returns control to the problem program. In this mode the supervisor disables all interrupts except the machine check. Because there is no possibility of interruption, no request block is required to retain interruption status. The second mode of operation occurs when the code that the supervisor is executing allows interrupts or operates enabled. This mode, therefore, requires a request block to retain status and save registers. in case an interruption occurs. This request block is called a supervisor request block (SVRB). It is created for, and associated with, SVC interrupt routines. An SVRB is dynamically created by the nucleus whenever the supervisor determines the requested SVC routine operates enabled. Free core is obtained at the upper end of core to build there SVRBs.

a. Do all SVC routines operate with interrupts enabled?

No, all SVC routines are broken down into types I through IV as described below.
Type I - These routines are always resident and operate disabled.
Type II - These routines are also resident; but are partially enabled and reenterable.
Type III - These routines are non-resident and reentrant. They are brought into the 1024 byte SVC transient area for execution from the SYS1. SVCLIB data set.

Type IV - These are multi-phase type III routines. They are too large to be brought into the transient area at one time and must be brought in phases. Control is passed from phase to phase via an XCTL macro.

SVRB's are, therefore, created for control of the type II, III, and IV SVC routines. Appendix A contains a list of the SVC's and their type classification.

3. Interrupt Request Block (IRB)

The processing environment of the operating system is such that at certain points in the processing cycle functions are defined to be processed if an asynchronous or unpredictable event occurs. The IRB is used to control these user and system asychronous exit routines. A good example of its use is when a timer routine is specified in the STIMER macro. This user routine, located in problem core, is given control when the specified time internal ends. Because one cannot predict when this internal will end, the STIMER SVC routine creates, by a GETMAIN requesting upper core, an IRB to control this user timer routine. When the interval times out and causes an external interrupt, the supervisor initializes the IRB, chains it onto the Active Queue and gives control to the user routine. The Active RB Queue at this time would look like Figure 20.

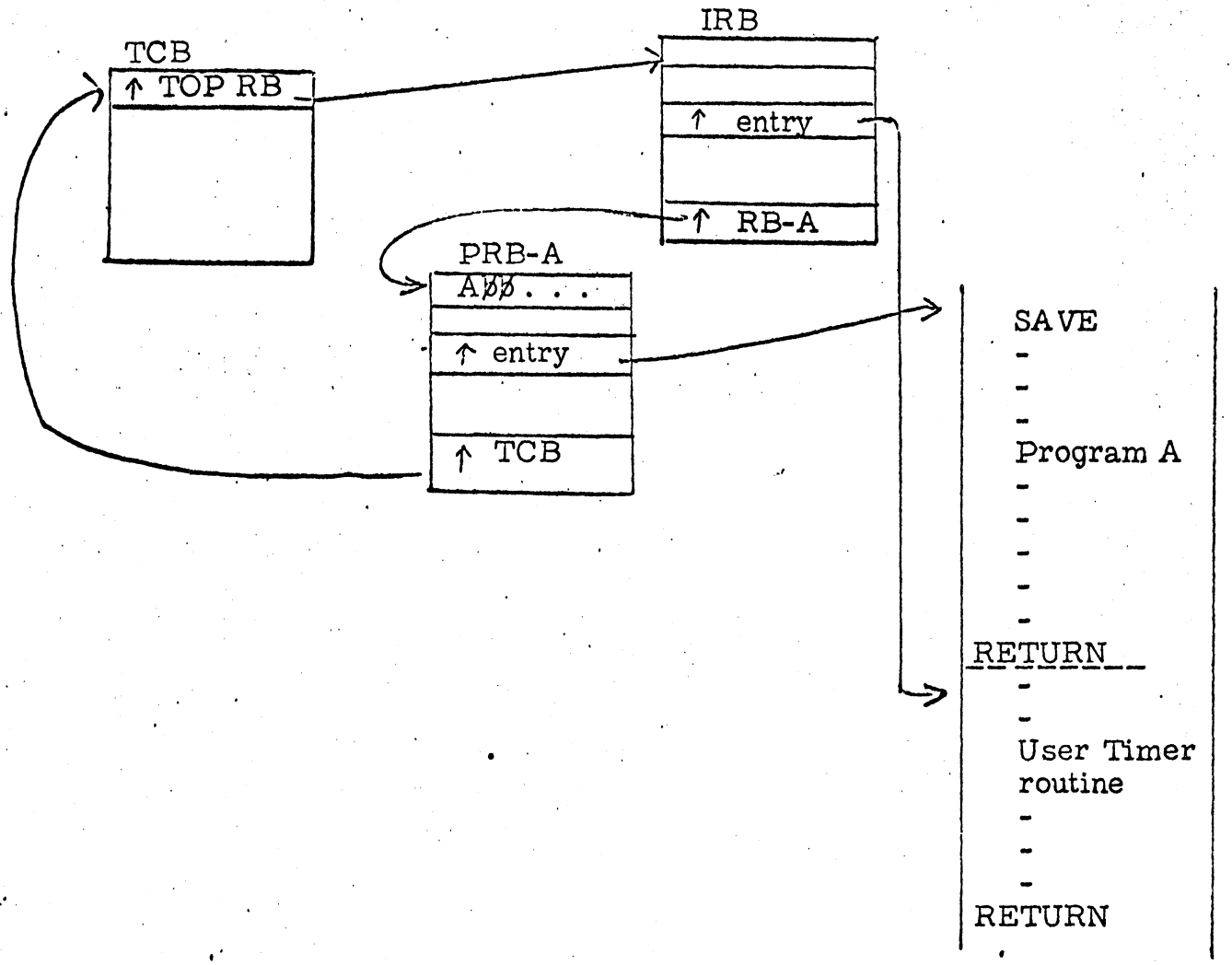

TCB
↑ TOP RB
IRB
↑ entry
↑ RB-A
PRB-A
AØØ . . .
↑ entry
↑ TCB
SAVE
Program A
RETURN
User Timer routine
RETURN

Figure 20.

4. Supervisor Interrupt Request Block (SIRB)

The function of an SIRB is similar to an IRB only an SIRB is associated only with the IBM supplied system I/O error routines. There are two additional features that are unique to an SIRB. There is only one SIRB and its associated routine operates out of a 400 byte transient area in the nucleus. Its associated routines are fetched from the SYS1. SVCLIB data set.

5. Loaded Program Request Block (LPRB)

These request blocks are used to control programs that are fetched into core as the result of the LOAD macro. An LPRB is always chained onto the Load List via use of the SUCCEEDing and PRECEEDing pointers which are unique to RB's associated with LOADed programs. An LPRB may also appear on the Active RB Queue as the result of an ATTACH, XCTL, or LINK being issued for its associated load module. In this case, the RB is maintained on both queues simultaneously through two different sets of pointers as indicated in appendix A, Figure 1. The LPRB is physically located adjacent to its LOADed routine and are allocated in core storage starting at the high end and working toward the middle.

6. Loaded Request Block (LRB)

This request block is a shortened form of an LPRB and used to control LOADed modules that have the only loadable (OL) attribute. This means that once loaded, this routine may be entered only by a branch. It is invalid to ATTACH, LINK, or XCTL to modules with this "OL" attribute. An LRB will be chained onto the Load List and will never be found on the Active RB Queue. It will be contiguous with its load module similar to the LPRB. A load module obtains this "only loadable" attribute at linkage editor time via the programmer specifying the OL subparameter in the PARM field of the EXEC control card. The most common reason for a programmer to specify this attribute is that he has not followed the linkage conventions required by the ATTACH, XCTL and LINK. Both the LRB and LPRB remain on the Load List until their routines are deleted as described in section B-3.

E. TCB and RB Fields

This section explains the TCB and RB fields using the formated output of the ABDUMP.

1. The TCB ABDUMP format is shown in Figure 21

```
TCB  000180     RB     01F83C  PIE   000000
DEB  01F7BC     TIOT   01FF5C  CMP   0C6000
TRN  00000000

MSS    00003CB8 PK/FLGS 00910400 FLGS/LDP 00000000
LLS    01F890   JLB      01FEE0  JSE      00000000
ID/FSA 0401FFB4 TCB      000000  TME      003CCC
```

Figure ~~28~~ 21

The following is an explanation of TCB fields that are helpful in debugging problem programs. All TCB fields are dumped in hexadecimal. The first 6 hex digits on line 1 of Figure 21 reflect the location of the TCB.

a. RB - This 4 byte field contains a pointer to the top request block on the Active RB Queue.

b. PIE - This 4 byte field contains a pointer to the Program Interrupt Element (PIE) if a SPIE macro has been issued by the problem program. Otherwise it contains zeros. A SPIE macro allows the programmer to specify an exit routine to be entered when specified program interruptions occur. The control program creates a 32 byte Program Interrupt Element to accomplish this function.

c. DEB - This field contains a 4 byte pointer to the Data Extent Blocks created for the OPENED data sets of the current job step. By using this field in conjunction with the second word in the chained DEB's one can determine which data sets have been opened. This area will be expanded later on.

d. TIOT - This 4 byte pointer contains the location of the Task I/O Table. From this table, one may determine which I/O device and associated unit control block (UCB) has been allocated to a specific data set.

e. CMP - This is a 4 byte field which contains the task completion code in hexadecimal. Only the right three bytes are used and these are split in two. The left three hex digits represent a completion code supplied by the problem program through a subparameter of an ABEND macro-instruction.

f. TRN - A 4 byte field used by TESTRAN contains the address of the Control Core table for controlling testing in a task.

g. MSS - A 4 byte field containing a pointer to the Main Storage Supervisor's boundary box. Useful in determining core size and resident control program size.

h. PK/FLGS - The first two hex digits (1 byte) are the contents of the protection key field. When protection is implemented this field will contain the assigned protect key for the problem program. The last six hex digits are the first three bytes of the flag field. Interpretation of these flags will help determine how ABEND was entered.

i. FLGS/LDP - This 4 byte field will be used later under Option 2 and 4 environments for additional flags and indicating dispatching priority.

j. LLS - A 6 digit hex address of the most recently added request block on the Load List. This is the Load List pointer that was explained in an earlier section. Total field length is 4 bytes.

k. JLB - A 6 digit hex pointer to the address of the DCB for the job library. If a JOBLIB DD card was not specified for this job, this field will contain zeros. Total field length is 4 bytes.

l. JSE - This 4 byte field contains a pointer to the Inactive Program List explained in an earlier section.

m. ID/FSA - This is an 8 digit hex number. The first two digits (1 byte) are always 04 in the PCP. The last six digits (3 bytes) are the starting address of the system supplied first problem program save area.

n. TCB - This is a 4 byte hex field which will contain zeros. Later, under option 2 and 4, this field will be used to chain the TCB's together to form the ready/wait queue.

o. TME - A 4 byte field which contains a pointer to the timer element. This field is not printed if the computing system does not contain the timer option at system generation.

It is important to note that the unformatted TCB within the dump has additional fields that are not formatted by ABDUMP. Use the Introduction to Control Program Logic Manual Z28-6605 as a guide to decipher an unformatted TCB.

2. The RB ABDUMP format is shown in Figure 22.

```
A001    005AA8 NM      LAST     SZ/STAB  005600C0
USE/EP 00005AC8 PSW   FF05000D 80005BFC
Q       000000 WT/LNK 00000264 UB       005D58
```

Figure 22

A001 on the first line indicates this is the request block that is chained to the TCB. (Lowest RB on the Active RB Queue). The next 6 hex digits indicates the location of the request block being formatted.

a. NM - This is an 8 character name or program identifier field of the request block. The contents of this field will vary depending on the type of RB. A program request block (PRB) will contain the member name by which the program was fetched. A supervisor request block (SVRB) for a type III and IV SVC routine will contain the signed decimal SVC code of the routine associated with this SVRB. For example, ABEND is a type IV SVC routine. Its associated SVRB will contain a 01C in this field. This is the signed decimal number for the ABEND SVC code $_{(013)}^{+}$.

b. SZ/STAB - This is an eight hex digit printout. The first four hex digits give the size of the RB plus its associated load module in hex, in doublewords. In Figure 22 the size field contains hex 0056. This represents an RB plus load module size of 688 bytes in decimal. This size field is used for this purpose only in loaded program request blocks (LPRB), loaded request blocks (LRB), or program request blocks (PRB). In SVRB's, IRB's, and SIRB's the size field indicates the size of the request blocks only. The last four hex digits are a set of status indicator denoting the RB type, active/inactive status, etc. See Appendix A figure 2 and 3 for a further breakdown of this field.

c. USE/EP - This is an eight digit hex printout. The first 2 hex digits are the use count. This field contains a count of the number of LOAD requests for a program. As explained earlier, the use count applies only to LRB's and LPRB's. The next 6 hex digits of this printout contain the entry point (EP) for the program or load module associated with the RB.

d. PSW - A 16 hex digit representation of the resume program status word (PSW). This field contains the last old PSW from either an I/O interrupt, or a type II, III, IV SVC interrupt.

e. Q - A 6 hex digit representation of the secondary queuing field. This field contains one of the following:

1. For interruption request blocks (IRB), the address of a 12- or 16-byte request element.
2. For program request blocks (PRB) and loaded program request blocks (LPRB), the address of a loaded program request block, which describes an entry point identified by an IDENTIFY macro-instruction.
3. For a supervisor request block (SVRB) for a type III or IV SVC routine, system information.

f. WT/LNK - The first two hex digits are the wait count. This is the number of WAITS which must be satisfied before this RB and its associated program may be dispatched. The last six digits are a pointer to the next RB on the Active RB Queue. A pointer to the TCB will reside in this field for the lowest RB (A001).

g. UB - A three byte field calculated by ABDUMP to indicate the upper bounds (hex address) of the load module associated with a PRB, LRB and LPRB.

h. REGS 0-7
REGS 8-15 - These fields represent the contents of the registers and are associated only with supervisor request blocks (SVRB) and Interrupt request blocks (IRB). This field is used like a users save area for the supervisor routines associated with these RB's. (These fields are not shown in Figure 22).

At this point one should have a good knowledge of the system control flow. This knowledge should be supplemented by locating the various system queues and control blocks on an actual ABDUMP output. This would also be a good point to read Appendix B, of the Control Program Messages and Completion Codes Manual C28-6608, which explains the fields within ABDUMP. It is recommended that the reader place Appendix B at the back of this chapter to insure the ABDUMP writeup and explanation is located in one place.

III. ABDUMP

A. User or System Problem?

Determining whether the problem is a system deficiency or a problem program error is the first step in any debugging process. In using the dump, one must first determine the type of error and second what program was in control when the error occurred.

1. Determining the type of error

To determine the type of error, the most positive clue is the completion code. One can quickly distinguish whether the user or PCP system supplied the completion code. The system prints out in decimal the user supplied completion code preceded by USER=notation. System supplied completion codes are preceded by SYSTEM=notation and printed in hexadecimal. A system completion code does not always mean that the user is not at fault because a user program can indirectly cause the ABDUMP. Therefore, the answer to whether it is a user or system problem cannot be arrived at until the Active RB Queue is investigated.

2. RB Queue Evaluation

The highest priority RB (the top of the Active RB Queue), will always be an SVRB for the SVC routine 051, which is ABDUMP. This SVC routine formats and prints the dump.. The next RB on the queue below ABDUMP is an SVRB for the SVC routine 013, which is ABEND. This SVC routine gains control whenever the system or user issues the ABEND macro. All RB's in the chain preceding these will be either the users, and/or those of data management (OPEN, CLOSE, etc.), other Type II, III, IV SVC routines, or the job scheduler. A typical Active RB Queue at ABDUMP time is shown in Figure 23.

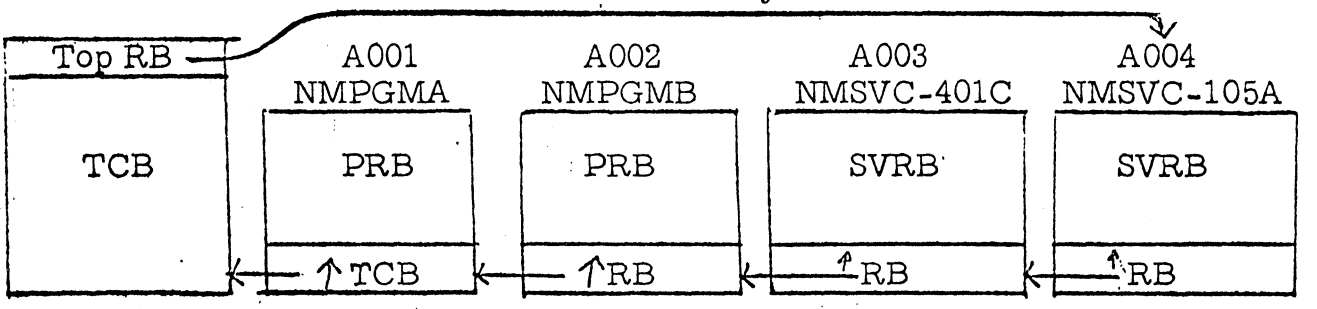


Figure 23

It is easily determined, by looking at the NM field in the RB, whether the users problem program was in control or not. In Figure 23, assume the load module called PGMB was in control at the time ABEND was called. PGMB may have issued the ABEND macro, in which case a users completion code would appear on the dump. A second possibility is that PGMB may have caused the system to issue an ABEND (i.e. - causing a program check). In the latter case, a system completion code would be printed on the dump but the problem was caused by the users program.

3. Naming Conventions

The programmer should know his load module names which appear in the NM field of the RB. However, the NM field for SVRB's and PRB's associated with system components is not quite so easily interpreted. It is, therefore, appropriate at this time to cover the naming conventions for system components and non-resident SVC routines. All system component load module names have the first three characters coded. The prefixes are listed in Appendix A under Operating System/360 Naming Conventions. If for example, a PRB was on the Active RB Queue for a job scheduler module, the NM field of the PRB would contain IEFzzzzz; where zzzzz represents the rest of the module name unique within the job scheduler.

The conventions for naming non-resident or transient SVC routines adds additional conventions to the three unique characters, IGC, which denotes transient SVC routines. The following conventions are used:

Type III - IGC00nnn; nnn is the signed decimal number of the SVC routine. This name must be the name of a member of a partitioned data set (SYS1.SVCLIB).

Type IV - IGCssnnn; nnn is the signed decimal number of the SVC routine, and ss is the number of the load module minus one, e.g., ss is 01 for the second load module of the routine. This name must be the name of a member of a partitioned data set (SYS1.SVCLIB).

In Figure 23 the load module member name, on the data set SYS1.SVCLIB, is IGC401C (in extended BCD). The system places only the last four characters, 401C, in the NM field of the associated SVRB. This is possible because the system knows that all SVC type III and IV routines are preceded by IGC0. SVRB's for Type II SVC routines (enabled and resident) do not have any meaningful names in the NM field because the system does not require a module name, it simply branches to the appropriate resident routine.

At this point, one should be able to determine via the completion code and evaluation of the Active RB Queue, who issued the ABEND, and what load module was in control at ABEND time. In addition one should be able to determine, by evaluating the STATUS and NM fields, of the controlling RB, that the load module was either a system component or user program.

B. User Problem

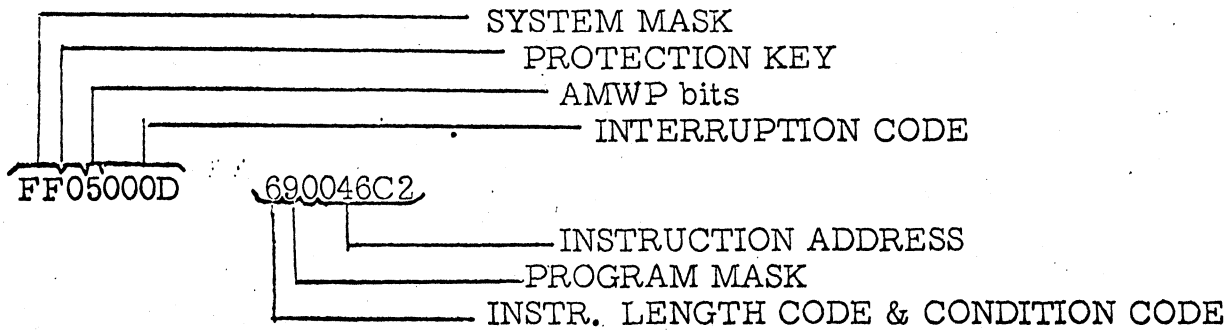
The assumption at this point is that the user has determined that the problem is either within his program or caused by some function performed in his program. To cover the first case, assume the user issued the ABEND. He should be able to localize the trouble using the completion code, which he issued in an ABEND macro instruction, and tie this back to his source listing. Next step would be to locate his program in core to analyze the present status of the program.

1. User Program Location

How do I find my program in core? The location of any program may be calculated using information in the associated request block. If the entry point of a particular program is the first instruction in that program, then the program boundary is given to the user via the EP field and upper bound (UP) field in the RB. If, as in the case of a FORTRAN written module, the entry point is not the lower bounds of the program, there are several ways of calculating the beginning location. The easiest way is to use the address of the RB and add 20 hex to its value. This is possible because the RB and its associated program are contiguous to one another.

2. Analyze Current PSW

What was the last instruction given in my program? Normally the next step in debugging is to determine where, within the failing program, was the last instruction given. The assumption here is that your problem is most likely localized in the area that the ABEND was issued. The last instruction in our example will be an SVC 13 which is ABEND. The answer to "where within the dump is this instruction located", can be found in two places. Evaluation of the 16 digit PSW UPON ENTRY TO ABEND printout is the first place. This is the current PSW as it existed upon entry to the ABEND SVC routine. It should be subdivided as shown in Figure 24.



bits 32 and 33	bits 34 and 35
00 - not available	00 - 0
01 - 2 bytes	01 - 1
10 - 4 bytes	10 - 2
11 - 6 bytes	11 - 3

Figure 24

The two fields that are meaningful in our search for the last instruction executed is the Instruction Address and Instruction Length Code. The Instruction Address field contains the address of the next instruction that would be executed by the CPU. Make sure this address is within the program boundaries calculated earlier in Step 1. If it isn't, you're on the wrong track. Assuming the contents of the Instruction Address field points within the problem program, the next step is to decrease this address by the size of the last instruction executed. This value is kept in the Instruction Length Code and Condition Code field. The left two bits of this four bit field indicates the length of the last instruction performed. In Figure 24, the last instruction length is 2 bytes, subtract this value hexadecimally from the Instruction Address field and the result (0046C0), points to the last instruction performed prior to ABEND.

3. Additional PSW Information

It is convenient at this time to talk about some of the other fields in this PSW. The AMWP bits are helpful in quickly distinguishing what mode the system was operating in when ABEND was issued. If the P bit is on, ABEND was issued as the result of something that occurred while the CPU was operating in the problem state. If the P bit is off, ABEND was issued as the result of a malfunction while the CPU was operating in the supervisor state. The Interruption Code field for an "old PSW" normally varies depending on the type of interrupt that occurs. For instance, this field in the I/O old PSW, after an I/O interruption, contains the hexadecimal address of the device that caused the interrupt; this field in the PROGRAM old PSW, after a program interrupt, contains the program exception code of 1 to 15; this field in the SVC old PSW, after an SVC interruption, contains the hexadecimal equivalent to the SVC code. This field in the "PSW upon entry to ABEND" will always contain 000D. This fact could cause problems and will be expanded on in a later section.

The RESUME PSW field, in the RB associated with the program that issued the ABEND, will also contain the information as was indicated in the PSW printout at the top of the dump.

4. User Debugging Steps

What steps should one take in debugging a user created problem? This question can best be answered by stepping through a typical problem. Assume for this example that the completion code is a system code 0C6. The following steps should be taken.

a. Determine, using the IBM Messages and Completions manual C28-6608 or the list in Appendix A, what type of error occurred. In this example, the error is a program specification.

b. The next step is to evaluate the instruction address field of the "PSW UPON ENTRY TO ABEND". Using this address in conjunction with the Active RB Queue, determine if the program check was within the boundaries of the user program.

c. Assuming the problem is within the user program, the next step is to pinpoint the instruction that failed. This is accomplished by evaluating the Instruction Length Code and Instruction Address fields in the "PSW UPON ENTRY TO ABEND". This procedure was explained earlier.

d. Now, using the instruction address calculated in c, one should find in the hex dump the instruction that failed and determine its function. At this point the procedure will vary depending on what source language the program was written in.

e. If the source program was written in Assembler Language, it is a simple matter to evaluate the instruction and determine, using the Principles of Operation Manual A22-6821, what could cause this type of programming error.

f. If the source program was written in a higher level language one must evaluate the instructions prior to and after the failure to determine what function they are performing and tie this back to the source program.

C. System Problem

As was indicated earlier, a system completion code does not mean that the user is free from fault. Careful evaluation of the completion code explanations and Active RB Queue is necessary to pinpoint the problem area. It is difficult to set up a precise sequence one should follow when debugging a system problem. The following are a few procedures one might consider.

1. Common Errors

Some of the more common errors evolve out of improper use of the job control language. When this occurs the Active RB Queue will contain an RB with a meaningful module name. This module name, indicated in the NM field of the RB, allows the user to refer to the Job Management PLM, Z28-6613, and determine what function this module performs. The primary step one must perform on system problems is to tie the problem back to a module and refer to the PLM's to determine its function.

Another factor to consider, when determining whether the problem is in a user program, is that the access routines are branched to. They, therefore, operate under the users RB. If one finds a high address in the instruction address field of the "PSW UPON ENTRY TO ABEND", evaluate the Load List to determine if the failure occurred within one of the loaded access routines. If the problem is within an access routine, refer to the proper PLM to determine the routine's function.

2. System Blocks

How does one find the system blocks not formatted by the ABDUMP? Listed below are the meaningful system blocks with an explanation of how to find and use them.

a. Communication Vector Table - (CVT)

The Communication Vector Table provides the means for nonresident routines to refer to information in the nucleus. The address of the first location of the CVT is placed in main storage location 16 (decimal) or 10 (hexadecimal). One should refer to the Introduction to Control Program Logic manual for the contents of this table. This table is useful in finding the Unit Control Blocks (UCB) for system data sets.

b. Task I/O Table (TIOT)

The Task I/O Table is constructed by job management and resides in the higher portion of the dynamic area of main storage during step execution. It provides the I/O support routines (OPEN, CLOSE, EOV) with pointers to the Job File Control Blocks (JFCB) and Unit Control Blocks (UCB). The JFCB, which resides on disk, contains information specified in the DD card for a specific data set. The UCB's, one of which is created for each device specified at system generation time, describes the device or devices allocated to a specific data set by the job scheduler. The TIOT would therefore contain the following information about each data set described for a particular job step:

* disposition and label status
* ddname of the data set
* relative pointer to the JFCB on a DASD device
* main storage pointer to the UCB's allocated to this data set.

A pointer to the TIOT may be found in the fourth word of the TCB.

c. MSS Boundary Box

The Main Storage Supervisor's (MSS) Boundary Box is a table of three addresses. It is pointed to by the MSS field in the TCB. The first address in the boundary box points to the first Free Queue Element (FQE) of what could be a chain of FQE's which describe the free core within the system at any given time. Each FQE describes a block of contiguous free core of which it is the first 8 bytes as shown in Figure 25.

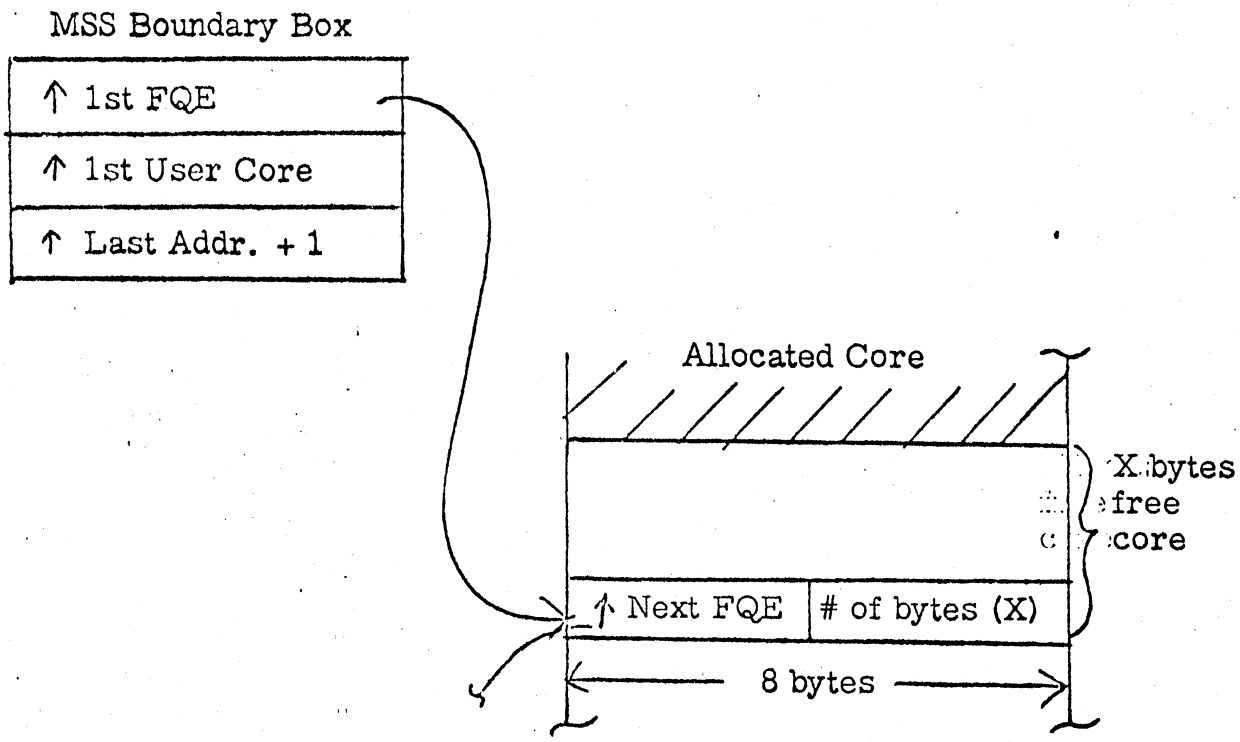

MSS Boundary Box
↑ 1st FQE
↑ 1st User Core
↑ Last Addr. + 1
Allocated Core
X bytes
free
core
↑ Next FQE
# of bytes (X)
8 bytes

Figure 25

The first 4 bytes of an FQE contains either; a pointer to the next FQE or contains zeros if it's the last FQE on the chain. FQE's are chained from the top of core (high addresses) toward the nucleus. The Main Storage Supervisor allocates user core requests starting at the top of core and running the FQE's til the request can be satisfied. The second 4 bytes specifies the number of free bytes available in this contiguous block of core. This count is in hexadecimal and includes the FQE size.

The second word in the boundary box points to the first available address outside the nucleus. The third word points to the last address in core plus one. All three of these pointers are calculated and initialized by the Nucleus Initialization Program (NIP) at IPL time.

It is important to note that ABDUMP does not dump free core. One can calculate the free core at the time of the dump by scanning the core addresses at the left of the dump listing and noting the number of bytes skipped when non-contiguous addresses are listed.

d. Data Extent Block (DEB)

The DEB contains an extension of the information in the DCB. Each DEB is associated with a DCB, and is created at OPEN time. There is a pointer (word 3) in the TCB that points to the first DEB on the chain of DEB's associated with the current job step. If there is a DEB on this chain, then the data set associated with this DEB has been OPENed. The DEB is the key block with which the user can find all the control blocks associated with a particular data set. Figure 26 shows the control block structure and the pointers indicating their relationship used by data management.

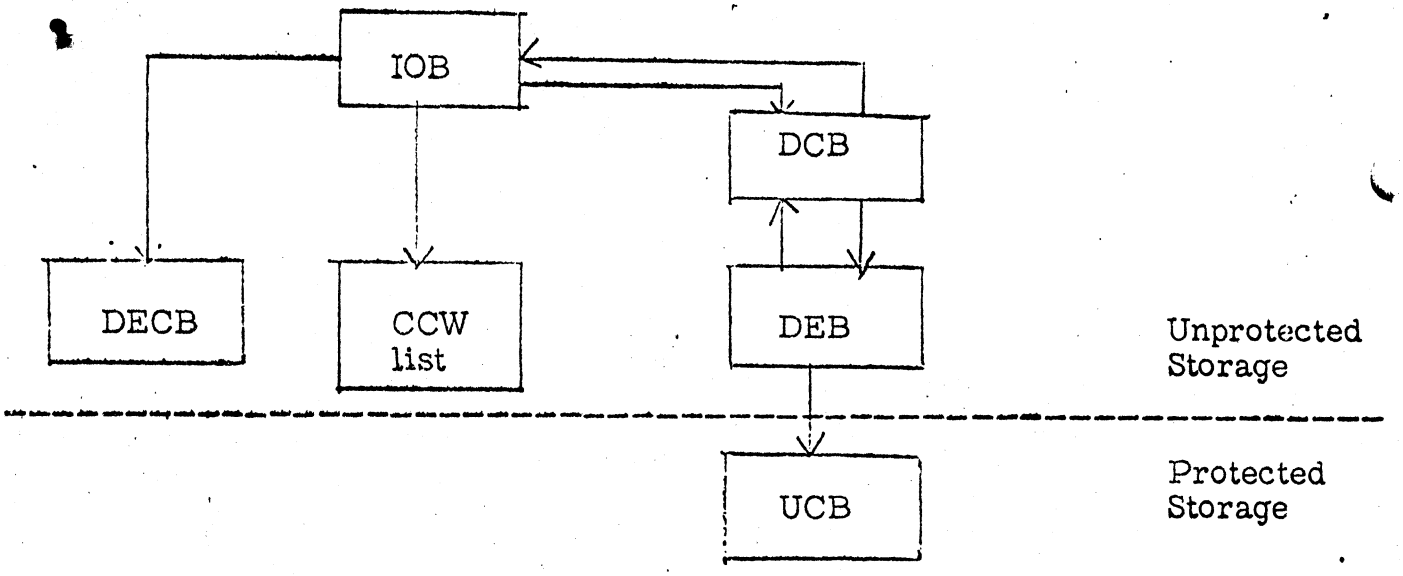


Figure 26

e. Input/Output Block (IOB)

The IOB is the communication between a routine that requests an I/O operation and the I/O supervisor. All the information required by the I/O supervisor to execute an I/O operation is contained in an IOB or is pointed to by the IOB. When an EXCP macro is issued by an access routine Register 1 contains a pointer to the IOB. This is an important fact to remember when evaluating the trace table. From the IOB, one cannot only find the other control blocks but also locate the list of channel commands (CCW's) performed or to be performed on a particular device.

f. Data Extent Control Blocks (DECB)

The DECB is created by the expansion of a READ or WRITE system macro. It is the communication link between the user and the access method. The access method in turn notifies the user of an completed I/O event by POSTing the Event Control Block contained in the DECB. This block provides synchronism between the user and the asynchronous I/O operation.

g. Unit Control Block (UCB)

There is a UCB for each device attached to the system. It describes the characteristics of the device to the I/O supervisor. The UCB is the only location where the user can obtain all the sense bytes passed back from the last sense command to a particular device.

Each of these blocks are further defined in the Introduction to Control Program Logic Manual Z28-6605.

IV. SAVE AREA

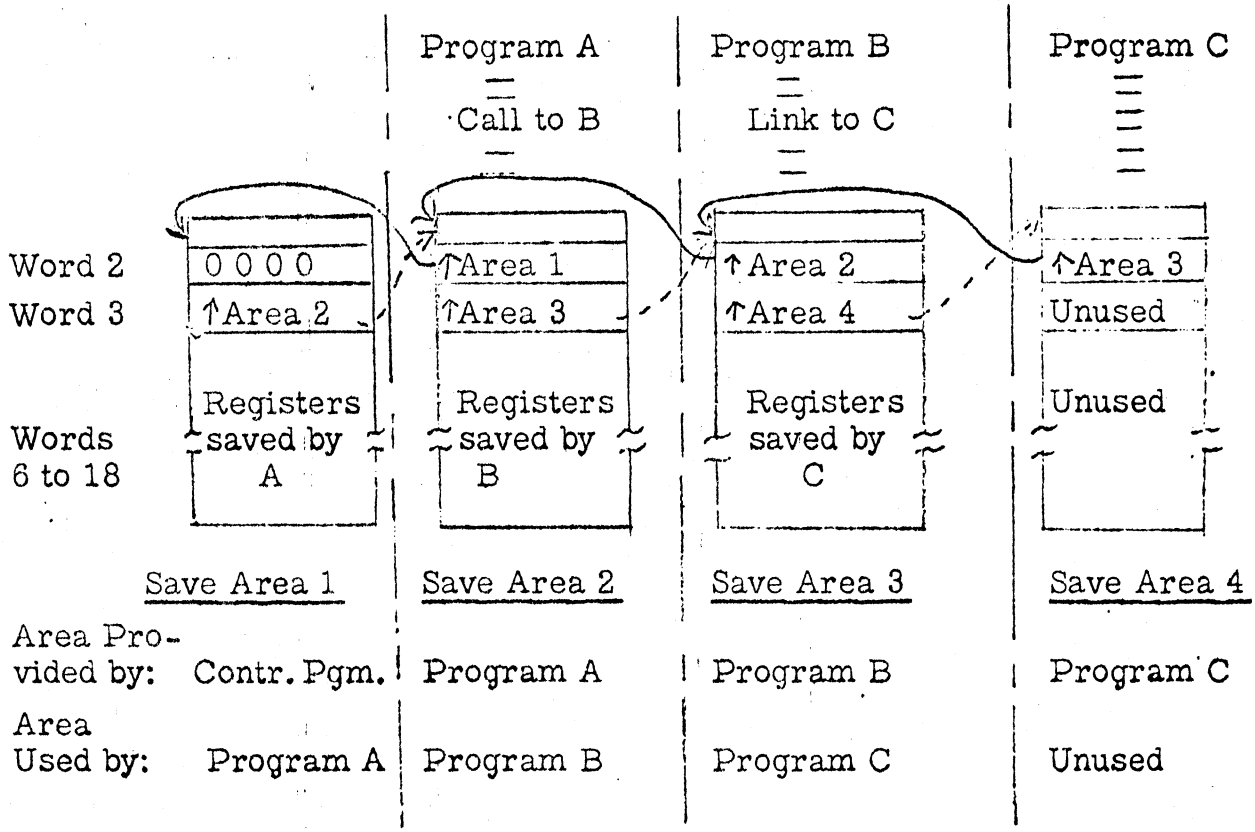
To understand the save area trace and its associated messages, one must understand what responsibilities the user has and what functions the control program performs when different linkages take place within a program.

A. Save Area Chaining

The following examples illustrate the chaining of save areas when different linkages are used, and relate the chaining sequence to the concept of control levels. Each example concentrates on (1) the use of words two and three of a save area, (2) the contents of Register 13 at the point of linkage, and (3) the responsibility of programs to provide save areas. Pointers to save areas in higher control level programs are shown as solid lines and called the back chain; pointers to save areas in lower control level programs are optional; are shown as dotted lines; and are called the forward chain.

EXAMPLE 1: The job stream contains an EXEC statement for module ALPHA. ALPHA consists of Program A and Program B, which was included in the module as a result of a CALL macro-instruction. Program B contains a LINK macro-instruction to Program C.

EXAMPLE 1



In this example, Program A is considered to be at the highest control level and Program C at the lowest. When Program A receives control, word 2 of save area 1, which is provided by the control program, contains zeros, and Register 13 points to this save area.

It is Program A's responsibility to:

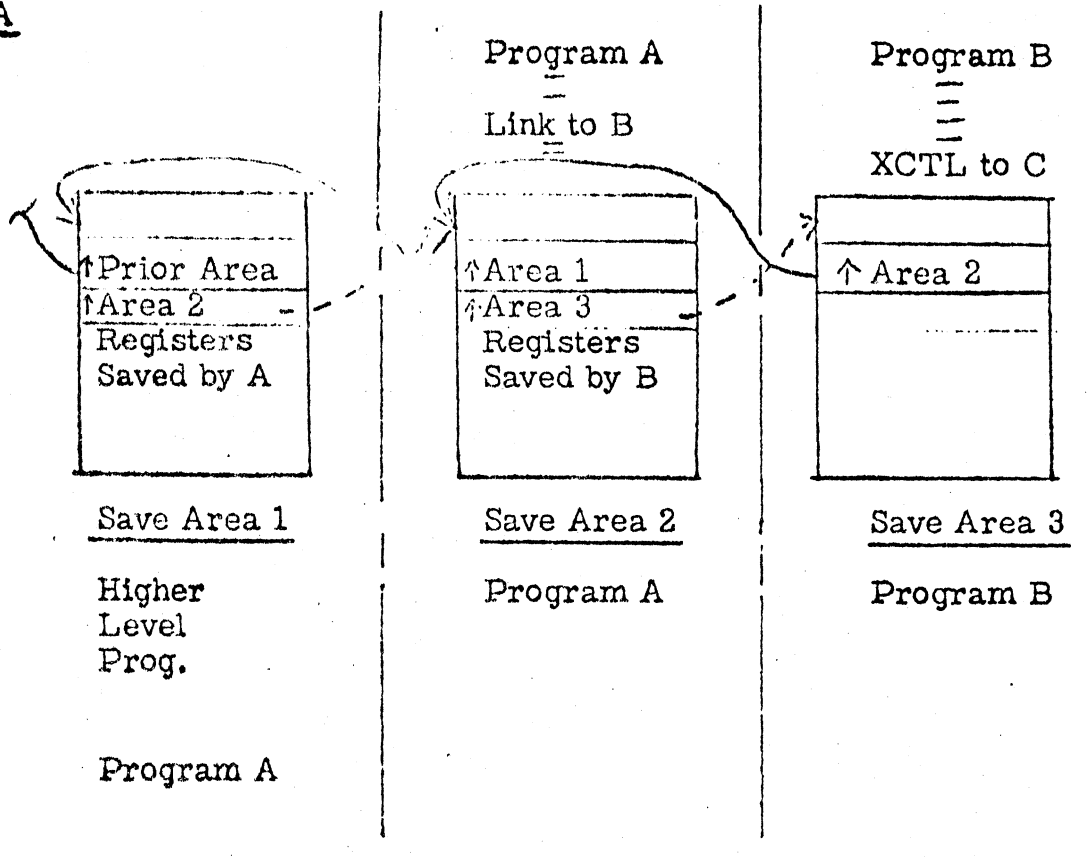
* save registers in save Area 1
* place the current register 13 into its own save Area 2, Word 2
* place the address of Save Area 2 into register 13 and word 3 of save area 1.

This procedure insures that at the time of each linkage from Program A, register 13 points to the save area of the higher control level program. A similar procedure is necessary upon entry to Program B. Since Program C does not either contain a linkage to a lower control level or issue a system macro instruction, save area 4 is not required. (Program C need only save register 13 until the return linkage.) The save area is shown here for generality, since Program C might require the area during another execution.

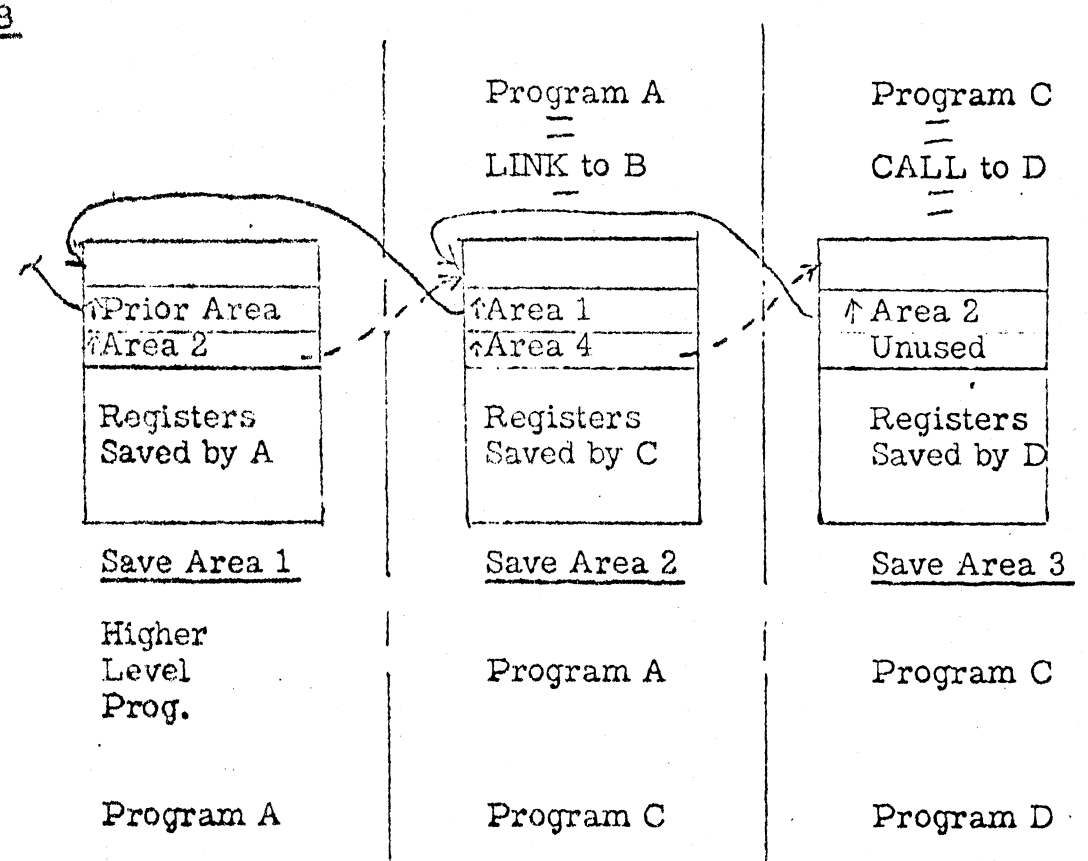
Example 2A and 2B:

Program A receives control from a higher level program and issues a LINK macro-instruction to Program B, which in turn issues an XCTL macro-instruction to Program C. Finally, Program C calls Program D. The major consideration here is the use of save area 2 by both Program B (before the XCTL macro-instruction Example 2A) and Program C (after the XCTL macro-instruction, Example 2B).

EXAMPLE 2A



EXAMPLE 2B



B. Save Area Trace

This heading identifies the next lines as a trace of the save areas for the program being terminated. Each save area is presented in the dump in three or four lines as shown in Figure 27. The first line gives information about the linkage that last used the save area. This line will not appear when the request block for the linkage cannot be found. The second line gives the contents of words 0 through 5 of the save area. The third and fourth lines give the contents of words 6 through 18 of the save area; these words are the contents of Registers 0 through 12.

```
SAVE AREA TRACE

PROGA  WAS ENTERED

SA    0003FFB8   WD1  00002449   HSA  00000000   LSA  00033D04
RET   00003180   EP   400330D0   00   00000030   01   0003FF1C
02    0000006C   03   00002449   04   00005318   05   0003FF4C
06    00003130   07   00000000   08   0000003C   09   4003A41A
10    0003FF0C   11   0003FF4C   12   00002448

SA    00033D04   WD1  48D0A004   HSA  0003FFB8   LSA  5AD02014
RET   5003400E   EP   00000000   00   19544078   01   00033D90
02    0003EDBA   03   00033DCC   04   C807DA96   05   0003EDA9
06    0003FCC0   07   00000008   08   0003EDA8   09   00034CA8
10    00033D90   11   00000000   12   40033CD6

INTERRUPT AT C34CAA

PROCEEDING BACK VIA REG 13

SA    00033D04   WD1  48D0A004   HSA  0003FFB8   LSA  5AD02014
RET   5003400E   EP   00000000   00   19544078   01   00033D90
02    0003EDBA   03   00033DCC   04   C807DA96   05   0003EDA9
06    0003FCC0   07   00000008   08   0003EDA8   09   00034CA8
10    00033D90   11   00000000   12   40033CD6
```

Figure 27

To provide a forward (descending) save area trace, the save areas are presented in the dump in the following order: (Reference Figure 27).

* The save area pointed to in the TCBFSA field of the task control block. This save area is the first one for the problem program; it was set up by the supervisor when the job step was initiated.

* If the third word of the first save area was filled by the problem program, then the second save area in the dump is that of the next lower level program of the task. However, if the third word of the first area points to a location whose second word does not point back to the first area, the message INCORRECT BACK CHAIN appears in the dump. This message is followed by the contents of the possible second save area.

* The third, fourth, etc., save areas are then presented in the dump, if the third word was filled in each higher save area and the second word of each lower save area points to the next higher save area. This process is continued until the end of the chain is reached (the third word in a save area contains zeros) or the message INCORRECT BACK CHAIN appears.

Following the forward trace, the line INTERRUPT AT hhhhhh appears, followed by the line PROCEEDING BACK VIA REG 13. Next, the save area in the lowest level program is presented, followed by the save area in the next higher level. The lowest save area is assumed to be the save area pointed to by the contents of Register 13. These two save areas appear in the dump only if the contents of Register 13 point to a full word boundary and are not zero.

1. SAVE AREA FORMAT

cccccccc WAS ENTERED

The 8-character name of the program that stored register contents in the save area. This name is obtained from the request block.

VIA LINK (CALL) ddddd

Either the LINK or CALL appears. The word LINK or CALL indicates whether a LINK or CALL macro-instruction was used to give control to the next lower level program. The 5-digit number is the ID operand, if it was specified, of the LINK or CALL macro-instruction.

AT EP cccccc . . .

The string of up to 70 characters is the entry point of the identifier. This identifier appears in the dump only if it was specified in the SAVE macro-instruction that used the save area being presented.

SA hhhhhhhh

The 8-digit address of the save area being presented.

WD 1 hhhhhhhh

The 8-digit representation of the contents of the first word of the save area. (Use of this word is optional).

HSA hhhhhhhh

The 8-digit representation of the contents of the second word of the save area; this word contains the address of the save area in the next higher level program. In the first save area, this word contains zeros. In all other save areas, this word is required to be filled.

LSA hhhhhhhh

The 8-digit representation of the contents of the third word of the save area; this word optionally contains the address of the save area in the called (lower level) program (register 13 contents).

RET hhhhhhhh

The 8-digit representation of the contents of the fourth word of the save area; this word optionally contains the return address (register 14 contents).

EP hhhhhhhh

The 8-digit representation of the contents of the fifth word of the save area; this word optionally contains the address of the entry point to the called program (register 15 contents).

00 hhhhhhhh 01 hhhhhhhh . . . 12 hhhhhhhh

These 8-digit numbers are the contents of registers 0 through 12 for the program containing the save area immediately after the linkage.

2. SAVE AREA TRACE MESSAGES

INCORRECT BACK CHAIN

This message indicates that the following three lines in the dump may not be a save area.

INTERRUPT AT hhhhhh

The 6-digit address of the next instruction to be executed in the problem program. It is obtained from the resume program status word of the last program request block (PRB) or loaded program request block (LPRB) in the active request block queue.

PROCEEDING BACK VIA REG 13

This heading indicates that the next two save areas in the abnormal termination dump are (1) the save area in the lowest level program, followed by (2) the save area in the next higher level. If register 13 contains zeros, these two save areas do not appear in the dump.

V. TRACE

The tracing routine is an Operating System/360 optional feature which you can use as a debugging and maintenance aid. The tracing routine stores, in a table, information pertaining to the following conditions:

* SIO instruction execution.

* SVC interruption.

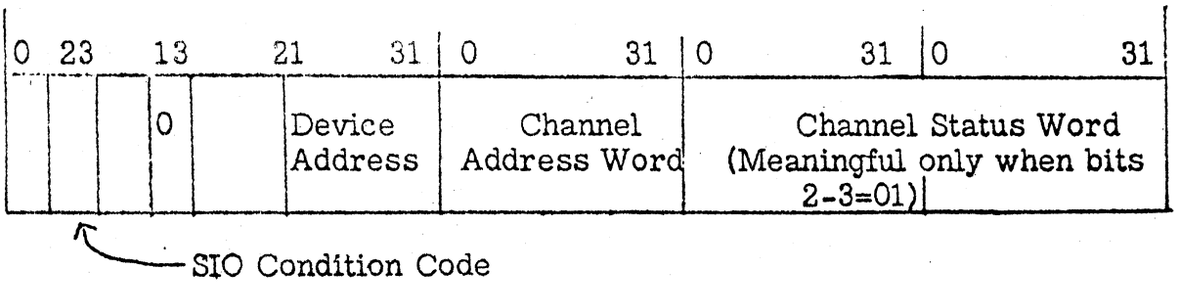
* I/O interruption.

You can include the tracing routine and its table in the control program during the system generation process. This is done using the TRACE option in the SUPRVSOR macro-instruction. The format of this option requires you to supply the number of entries in the table. Each table entry can contain information relating to one of the traced conditions. When the last entry in the table is filled, the next entry will overlay the first.

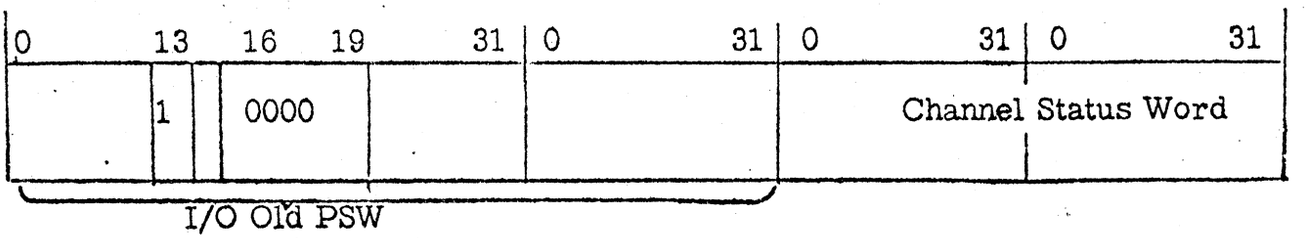
A. Table Entry Formats

Table entry formats are in Figure 28.

SIO Instruction



I/O Interruption



SVC Interruption

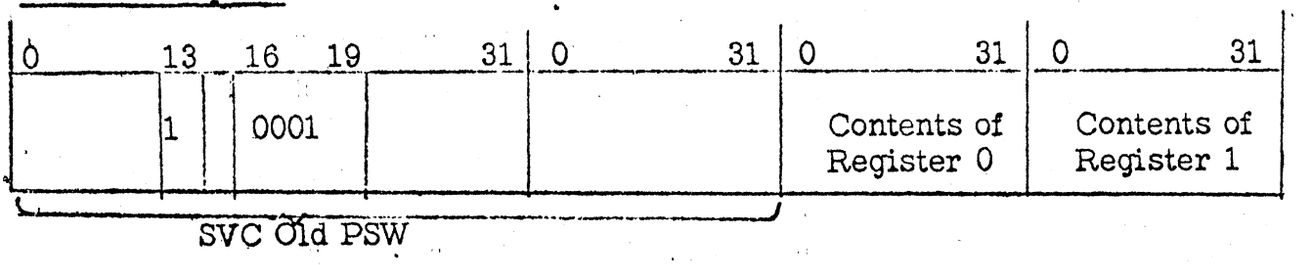


Figure 28

B. Location of the Table

The addresses of the last entry made in the table, the beginning of the table, and the end of the table are contained in a 12-byte field. The address of this field is contained in the full word starting at location decimal 20 (hex 14). The format of the field is as follows:

0 31	0 31	0 31
Address of the Last Entry	Address of the Table Beginning	Address of the Table End

The tracing routine is bypassed during the abnormal termination procedures.

C. Trace Examples and Explanation

1. SIO entry - A typical SIO entry is shown in Figure 29. This entry can be distinguished from an I/O interruption and SVC interruption entry by checking the fourth hex digit. If this digit is a zero then the trace entry is an SIO. One would use the channel address word to locate the channel command words (CCW) which reflect the operation initiated by the Start I/O instruction. The Channel Status word entry is meaningful only if the condition code, set by the SIO instruction, is 1. This condition code is reflected in the first hex byte, bits 2 and 3 of the first word printed. The unit address reflects the device which the I/O operation was initiated.

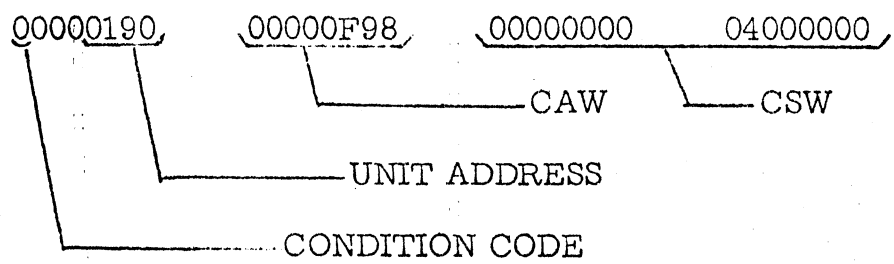


Figure 29

2. I/O Interruption Entry - A typical I/O interrupt entry is shown in Figure 30. By checking of the fourth hex digit one may distinguish between an SIO and I/O Interrupt entry. However, to distinguish between an I/O Interrupt entry and SVC entry, a further check of the fifth hex digit is necessary. If the fifth hex digit is zero, then the trace entry is an I/O interrupt. If the fifth hex digit is one, the entry is an SVC interrupt. By evaluating the I/O old PSW a number of facts can be determined. For example, the interrupt code field (hex digits 5 - 8) contains the address of the device that caused the interrupt, the AMWP bits (hex digit 4) indicates whether the interruption occurred in problem state or supervisor state; and the instructions address indicates where the interruption occurred. The CSW provides the user with unit and channel status about the device and channel that caused the interrupt. The first four hex digits of the second word indicates unit status and channel status respectively.

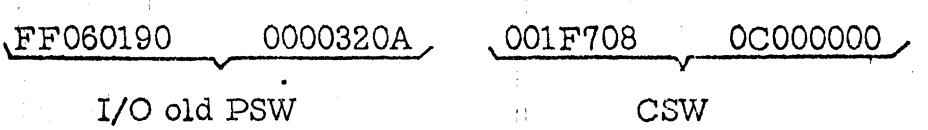


Figure 30

In Figure 30 the unit status indicates a channel end and device end. No channel status bits were on. This status denotes a successful I/O operation. The Principle of Operations manual should be referred to for further explanation of the channel and unit status fields. Upon determining that an entry is a successful I/O operation, the first word of the CSW may be checked to determine what commands have just been completed. The address in this word reflects the last CCW address plus eight. In Figure 30 the last command to be performed is located at 1F700.

3. SVC Interruption Entry - A typical SVC interrupt entry is shown in Figure 31. The first two bytes contain the SVC old PSW. The last two bytes of the entry reflect the contents of Register 0 and Register 1 at the time the SVC interrupt occurred. These registers are parameter passing registers used by many system macros. Most system macros degenerate into unique SVC interrupts. Appendix A lists the system macros and their associated SVC numbers. The interrupt code field (hex digits 6 - 8 of the first word) contains the hexadecimal value of this SVC number.

In Figure 31 an SVC 0 or the EXCP macro was issued. Additional checking of the SVC old PSW will reflect what mode, problem or supervisor, and where the SVC instruction was issued. Also note that in the case of an EXCP macro, Register 1 reflects the address of the IOB for this I/O operation. By knowing the IOB location, one can find all the associated data management control blocks for this operation.

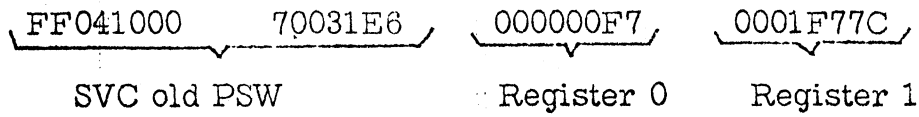


Figure 31

There is one case where the SVC entry may be misleading. This occurs when a program interruption causes the ABDUMP. In this case, an SVC 13 is placed in the trace table that gives all indications that the users program issued the ABEND macro. Actually the system issued the ABEND using the PSW as it existed when the program interruption occurred.

4. What is the recommended method of using the trace table?

The trace should be used to further pinpoint system and user problems. A suggested procedure is to:

a. find the last entry made in the trace table.

b. back up from that entry to the last SVC entry made indicating the problem state.

c. tie this entry back to some function performed in the user's program.

d. work forward in the trace table from this problem state entry and try to determine what functions took place between this entry and the last entry prior to ABEND.

VI. DASD DATA SETS

This section will discuss the availability and location of information about DASD data sets that would be of interest to the installation programmers.

The items to be covered are broken down into the following groups:

* The Volume Table of Contents -- VTOC
* Data Set Control Block -- DSCB
* PartitionedData Set Directory Entry

A. The Volume Table of Contents (VTOC) Evaluation

Prior to evaluating the contents of the VTOC one should understand the functions performed by the direct access device space management (DADSM) routines.

1. DADSM

Direct-access device space management (DADSM) consists of routines that allocate space to data sets on direct-access volumes. DADSM performs this function by maintaining the volume table of contents (VTOC), itself a data set that is included in every direct-access volume by the volume initialization utility program (DADSI). A VTOC contains a data set control block (DSCB) for each data set on the volume and for all unused space on the volume.

DADSM routines update VTOCs by creating DSCBs for new data sets and deleting the DSCBs of data sets purged from storage. When a data set is created or an existing data set is enlarged, DADSM finds unused space by searching the appropriate DSCB in the VTOC, allocating the space to the extent of the data set, and removing it from available space. When DADSM deletes a data set, it al so removes the DSCB of the data set from the VTOC; and the extent of the data set identified in the DSCB is again available for future allocation. DADSM can also return unused space at the end of a data set extent to available space by updating the DSCB of the data set.

Additional information concerning DADSM can be found in IBM System 360 O/S Direct Access Device Space Management PLM - Form #Z28-6607.

2. VTOC - Listing and Description

The first step in checking the status of a DASD data set requires the execution of the IEHLIST program which will print the contents of the VTOC.

The VTOC is a data set that contains a DSCB for every data set and for all available space on the Direct Access volume. The VTOC data set is always a single contiguous area. Its size and location are determined when the volume is initialized by the VTOCD control card in the DASDI input stream. On a Primary Systems pack, the VTOC can be located anywhere following the IPL records and volume label. On other than Primary Systems packs, the VTOC can be anywhere following the volume label. The starting address of the VTOC is recorded in the standard volume label.

The characteristics of the VTOC data set, following volume initialization, are as follows:

a. The initial volume label contains the absolute track address of the VTOC data set.

b. The VTOC contains two DSCB's:

   1. The first is a Format 4 DSCB which describes the VTOC data set. This DSCB is always the first block of the VTOC.

   2. The second DSCB, a Format 5, describes the space on the volume not occupied by data sets. This is located immediately after the Format 4 DSCB of the VTOC.

c. Every DSCB in the VTOC is 140 bytes in length (44 byte key and 96 byte data portion). Unoccupied space in the VTOC contains all zeros (Format 0 DSCB's).

d. A DSCB, Format 1, is created as a result of a DD card specifying a data set name and including a space parameter. The information from the DD card statement is included in the DSCB for investigation and use by the DADSM routines.

3. Formated VTOC

A sample formated VTOC of a volume with serial number 111111 is shown in Figure 32. For reference purposes, the information about the data set SYS1.LINKLIB will be investigated. The following items are found:

a. Created - This is the date that the data set was created. The date being picked up from the set date command given by the operator during the IPL operation.

b. Purge - This is the date that the data set may be purged. This date is entered when the data set is created by a parameter of the DD card.

c. File Type - This item specifys the type of data set organization.

d. Extents - A data set may have up to 16 extents. The extents contain the physical location of the data set on disk. Three extents can be contained in the Format 1 DSCB. If more are needed, a Format 3 DSCB which can contain up to 13 extents, is chained to a Format 1 DSCB to account for the 16 possible extents.

e. File Serial - This is the serial number of the data set contained in the DSCB.

f. Volume Sequence Number - If the data set requires multiple volumes, this is the method of keeping track of the order of volumes.

g. Security - None at this time.

Figure 32 is the formated IEHLIST of the VTOC. An example of a dumped VTOC will be covered later in this chapter.

CONTENTS OF VTOC ON VOL 111111

DATA SET NAME	CREATED	PURGE	FILE TYPE
SYSCTLG	07066	35099	NOT DEFINED
SYS1.SVCLIB	07066	35099	PARTIONED
SYS1.SYSJOBQE	36917	36917	NOT DEFINED
SYS1.LINKLIB	07066	35099	PARTIONED
SYS1.PROCLIB	07066	35099	PARTIONED
SYS1.NUCLEUS	08866	36598	PARTIONED
SYS1.SORTLIB	07066	35099	PARTIONED
SYS1.COBLIB	07066	35099	PARTIONED
SYS1.FORTLIB	07066	35099	PARTIONED
SYS1.BLOCKLIB	36099	35099	PARTIONED
SYS1.OBJ	35599	35599	PARTIONED
F.ASSEMBLR.MASTER.TESTCASE	00166	35099	SEQUENTIAL
F.MACROGEN.MASTER.PATCHES	00166	35099	SEQUENTIAL
SYS1.LOGREC	35599	35599	SEQUENTIAL
F.MACROGEN.MASTER.LIBRARY	00166	35099	PARTIONED
F.ASSEMBLR.MASTER.PATCHES	00166	35099	SEQUENTIAL
F.ASSEMBLR.MASTER.LIBRARY	00166	35099	PARTIONED.

THERE ARE 0067 EMPTY CYLINDERS PLUS 0012 EMPTY TRACKS ON THIS VOLUME
THERE ARE 0141 BLANK DSCBS IN THE VTOC ON THIS VOLUME

	EXTENTS	FILE SERIAL	VOL. SEQ.	SECURITY
SYSCTLG (cont.)	00001	111111	00000	NO
SYS1.SVCLIB (cont.)	00001	111111	00000	NO
SYS1.SYSJOBQE (cont.)	00001	111111	00000	NO
SYS1.LINKLIB (cont.)	00001	111111	00000	NO
SYS1.PROCLIB (cont.)	00001	111111	00000	NO
SYS1.NUCLEUS (cont.)	00001	111111	00000	NO
SYS1.COBLIB (cont.)	00001	111111	00000	NO
SYS1.FORTLIB (cont.)	00001	111111	00000	NO
SYS1.BLOCKLIB (cont.)	00001	111111	00000	NO
SYS.OBJ (cont.)	00001	111111	00000	NO
	00001	111111	00000	NO
	00001	111111	00000	NO
	00001	111111	00000	NO
	00001	111111	00000	NO
	00001	111111	00000	NO
	00001	111111	00000	NO
	00001	111111	00000	NO
	00001	111111	00000	NO

Figure 32

4. The Data Set Control Block (DSCB)

For each data set on a direct-access volume, there must be a corresponding data set control block (DSCB) in the VTOC of that volume. A DSCB, which describes the attributes and extents of a data set, consists of up to three physical blocks chained together to form one logical record (DSCB) in a VTOC. Each block is 140 bytes long (a 44-byte key and a 96-byte data portion), and contains information used by data management to control access to a data set. If a data set resides on more than one volume, there must be a DSCB for the data set in each VTOC.

DSCBs consist of blocks of which there are seven formats:

A format 1 block can identify any data set, except for the VTOC, on direct-access storage. Within its structure, it can identify up to three noncontiguous areas of a data set extent. Additional format 2 or format 3 blocks can be chained to a format 1 block to constitute one DSCB.

A format 2 block describes an indexed sequential data set. A format 2 block, if used, must be chained to a format 1 block.

A format 3 block describes multiple extents of a data set if more than three noncontiguous areas are allocated. A format 3 block, if used must be chained to a format 1 or format 2 block. A maximum of 13 non-contiguous areas may be described by a format 3 block.

A format 4 block describes the extent of the VTOC. It always appears first in any VTOC. One format 4 block constitutes one DSCB and can not be chained to other blocks.

A format 5 block describes up to 26 noncontiguous areas that are available for allocation on a volume. Each area is indicated in a separate "extent entry" in the block. Format 5 blocks can be chained together if the volume contains more than 26 available areas.

A format 6 block describes up to 26 split-cylinder data set extents. This block has the same format as the format 5 block, but describes extents shared by more than one data set. Each area of an extent is identified in a separate "extent entry" in the block.

A format 0 block is available space in the VTOC. The block contains all zeros, and can be imagined as a "hole". When a data set is deleted from a volume, a format 0 block is written over the DSCB of the data set.

Except on basic operating system volumes and on volumes containing split cylinder data sets, the total number of tracks accounted for in all DSCBs of a VTOC, at any time, is the total number of tracks on the volume. The unused tracks are identified in the format 5 DSCB, and used tracks are identified in blocks of format 1, 3, and 4 of data set DSCBs.

When a data set is created, the ALLOCATE routine finds space on the volume by searching the format 5 DSCB. A new DSCB is created for the new data set and is placed in the VTOC in the first available hole (format 0 block). When a data set is deleted, its format 1, format 2, and format 3 blocks are replaced by format 0 blocks (holes), and the extent used by the data set is returned to available space (i.e., added back into the format 5 DSCB).

The DSCB of one data set consists of one, two, or three blocks, depending on the access method used to process the data set, and on the number of noncontiguous areas in the data set's extent. The blocks are chained together in the VTOC in the following sequences:

a. A format 1 block alone for a data set with an extent of not more than three noncontiguous areas.

b. A format 3 block chained to a format 1 block for a data set with more than three noncontiguous areas.

c. A format 2 block chained to a format 1 block for an indexed sequential data set with an extent of no more than three noncontiguous areas.

d. A format 3 block chained to a format 2 block, which is chained to a format 1 block, for an indexed sequential data set with an extent of more than three noncontiguous areas.

The blocks of one DSCB do not necessarily appear in the VTOC as contiguous blocks or in a defined sequence. Except for the first and second DSCB blocks in a VTOC, new blocks are placed in the hole nearest the beginning of the VTOC when they are created. The relationship of two blocks is shown by chain addresses.

5. Dumped VTOC

From the previous section it is seen that the DSCB contains all the information about data sets. When the IEHLIST program is run requesting a Dumped VTOC, the result is the DSCB listed as shown in Figure 33. We will again look at the data set named SYS1.LINKLIB. There are a number of items that did not appear in the formated VTOC. In order to easily decipher the Format 1 DSCB bytes, two templates will be used. These templates are shown in figures 35 through 38.

Assume that the characteristics of the SYS1.LINKLIB data set listed below are needed.

a. type of organization

b. number of extents

c. location of the first extent

d. record format

e. block length

f. secondary allocation

By placing the template, Figures 35 and 36, over lines one and two of the DSCB, the items above can be answered as follows:

a. Item K--2 bytes--0200--Bits 0000 00 10 0--0, bit 6 is on--Partitioned Organization.

b. Item F--1 byte--01 extent

c. --not found on line 1 or 2

d. Item L--1 byte--CO Bits 1100 0000 11 - undefined

e. Item N--2 bytes--0400--1024 bytes

f. Item S--last 3 bytes of a 4 byte field--000000--
no secondary allocation.

Item C does not appear on lines one or two so template, Figures 37 and 38, will be used to look at line 3 of the DSCB. The first extent location description is Item C -- 10 bytes --

First byte 81 -- The extent begins and ends on cylinder boundaries.

Second byte 00 -- Extent sequence number.

Third - Sixth bytes -- 00 14 00 00 lower limit
C C H H of extent

Seventh - Tenth bytes -- 00 3B 00 09 upper limit
C C H H of extent

CONTENTS OF VTOC ON VOL 111111

```
LINE 1 = DSNAME                    . . . . 5 . . . .10 . . . .15 . . . .20
LINE 2 = 45 . . . .50 . . . .55 . . . .60 . . . .65 . . . .70 . . . .75
LINE 3 =100 . . . 105 . . . 110 . . . 115 . . . 120 . . . 125 . . . 130
FORMAT 4 DSCB                  040404040404040404040404040404040404040404
F4000900020100220C6800000001E0001000000C5000A0E29523714010219100
0C0000000000C01000C09C000000090009000000000000000000000000000000000
FORMAT 5 DSCB                  050505050001000000903FC0001000622002A0900
F5000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
SYS1.LOGREC
F1F1F1F1F1F1F1000000000000000000010000000000000000000000000000000000
031404A10000810C000C1000000002000900000000000000000000000000000000000
SYS1.SYSJOBQE
F1F1F1F1F1F1F1000001111111111110100000000000000000000000000000000000
3B0FFFCE0000C81000003C000000080009000000000000000000000000000000000
SYS1.SVCLIB
F1F1F1F1F1F1F1000000000000620160010000000000000000000000000000000000
400209700000810000CA0000001300090000000000000000000000000000000000
SYS1.LINKLIB
F1F1F1F1F1F1F1000000000000620160010000000000000000000000000000000000
F40309C300000810000140000003B0009000000000000000000000000000000000
SYSCTLG
F1F1F1F1F1F1F1000000000000620160010000000000000000000000000000000000
00FF0E2900000810000300000003C0009000000000000000000000000000000000
```

```
                    . . . .25 . . . .30 . . . .35 . . . .40 . . . .
                    . . . .80 . . . .85 . . . .90 . . . .95 . . . .
                    . . . 135 . . . 140          DSCB ADDR (CCHHR)
Format 4 (cont.)    040404040404040404040404040404040404040404040404
                    0A0000000000000000000000000000000000000000000000
                    0000000000000000000000          0009000001
Format 5 (cont.)    000000000000000000000000000000000000000000000000
                    000000000000000000000000000000000000000000000000
                    000000000000000000000          0009000002
SYS1.LOGREC (cont.)
                    000000000000000000800002600000200000000000000000
                    000000000000000000000          0009000003
SYS1.SYSJOBQE (cont.)
                    000000000000000000800002600000200000000000000000
                    000000000000000000000          0009000004
SYS1.SVCLIB (cont.)
                    000000000000000000200000400000000000000000000000
                    000000000000000000000          0009000005
SYS1.LINKLIB (cont.)
                    000000000000000000200000400000000000000000000000
                    000000000000000000000          0009000006
SYSCTLG (cont.)
                    000000000000000000000000000000000000000000000000
                    000000000000000000000          0009000007
```

DATA SET CONTROL BLOCK

FORMAT 1 LINE 1 and 2 L.

(A)	1	Format identifier - Hex F1
(B)	6	Data set serial number
(C)	2	Volume sequence number
(D)	3	Creation date - ydd where y = year (0-99) dd = day (1 - 366)
(E)	3	Expiration date
(F)	1	Number of separate extents
(G)	1	Number of bytes used in the last PDS directory block
(H-J)	1-7	Reserved for future use
(I)	13	System code to identify the programming system
(K)	2	Data set organization

Specified	Bits	Settings	Meaning
	0		Reserved for future use
PS	1	1	Physical sequential organization
DA	2	1	Direct organization
	3-5		Reserved for future use
PO	6	1	Partitioned organization
U	7	1	Unmovable - the data contains location dependent information

2nd Byte - Reserved for future use

(L) 1 Record Format

Specified	Bits	Settings	Meaning
F	0-1	10	Fixed
V	0-1	01	Variable
U	0-1	11	Undefined
T	2	1	Track overflow
B	3	1	Blocked: may not occur with U
S	4	1	Standard: no truncated blocks or unfilled tracks are embedded in the data set
A	5-6	10	ASA control character
M	5-6	01	Machine control character
	5-6	00	No control character
	7	0	Always zero

→ Data Set Name

Figure 35

A B C D E F G H

DATA SET CONTROL BLOCK

FORMAT 1 LINE 1 and 2 R.

(M)	1	Option code - same as DCBOPTCD field in DCB
(N)	2	Block length for fixed length records or maximum block for variable or undefined length records
(O)	2	Logical record length
(P)	1	Key length
(Q)	2	Relative key position in the data block
(R)	1	Data set indicators

Bits	Settings	Meaning
0	1	This is the last volume on which this data set normally resides
1		Reserved for future use
2	1	Block length must always be a multiple of 8 bytes
3-7		Reserved for future use

(S) 4 Secondary Allocation

First Byte - Allocation parameters

This field indicates the type of request that was issued for initial allocation and is to be used for subsequent extensions.

Bits	Settings	Meaning
0-1	00	Original request was in tracks relative to a specific location. No secondary allocation will be allowed
0-1	01	Original request was in blocks (physical records)
0-1	10	Original request was in tracks
0-1	11	Original request was in cylinders
2-3		Reserved for future use
4	1	Original request was for a contiguous extent
5	1	Original request was for the maximum contiguous extent on the volume
6	1	Original request was for the five (or less) largest extents that are greater than or equal to a specified minimum
7	1	Original request in records was to be rounded up to a cylinder boundary

Last Three Bytes - Secondary allocation quantity

The contents of this binary field indicate the number of blocks, tracks, or cylinders to be requested at end of data set when processing a sequential data set.

Figure 36

Ⓐ 5 Last block pointer: The contents of this field identifies the last block written in a sequential or partitioned organization data set. It is in the format TTRLL:

TT is the relative address of the track containing the last block
R is the block number on that track
LL is the number of bytes remaining on that track following the block

If the entire field contains binary zeros, the last block pointer does not apply.

Ⓑ 2 Reserved for future use.

Ⓒ 10 First extent description

C1 First Byte - Data Set extent type indicator

Hex Code	Meaning
00	Following 9 bytes do not indicate any extent.
01	The extent contains the data blocks (user's blocks)
80	The extent described is sharing one or more cylinders with one or more data sets
81	The extent described begins and ends on cylinder boundaries, i.e., the extent is composed of one or more cylinders

C2 Second Byte - Extent sequence number

C3 Third - Sixth Bytes - lower limit of this extent (CCHH)

C4 Seventh - Tenth Bytes - upper limit of the extent (CCHH)

Ⓓ 10 Second extent description - same format as SD1EXT1

Figure 37

DATA SET CONTROL BLOCK

FORMAT 1 LINE 3 R.

E	10	Third extent description - same format as DS1EXT1
F	5	Pointer to Format 3 DSCB if a continuation is needed to describe this data set. This pointer has the format CCHHR.
H	5	DSCB ADD (CCHHR)

Figure 38

6. Extents

There is a possibility of having a data set composed of up to 16 separate extents. The Format 1 DSCB can contain 3 extents. Further expansion of the data set requires a Format 3 DSCB to contain the extent information. The following discussion describes the Format 3 DSCB and also shows how it is located. Refer to Figure 39 and the description of the Format 3 DSCB in the Introduction to Control Program Logic Manual Z28-6605 to follow the example given.

The data set, SYS1.UT3, will be looked at by using the templates of Figures 37 and 38. It is seen that 3 extents are used in the Format 1 DSCB. There is a pointer on line 3 of the Format 1 DSCB to 00 05 00 00 0E (CCH HR). At this location in the VTOC there is a Format 3 DSCB. This DSCB contains extents that are part of the data set, SYS1.UT3. From the introduction to Control Program Logic Manual, it is seen that there are 13 - 10 byte fields that are similar to the extent fields of the Format 1 DSCB.

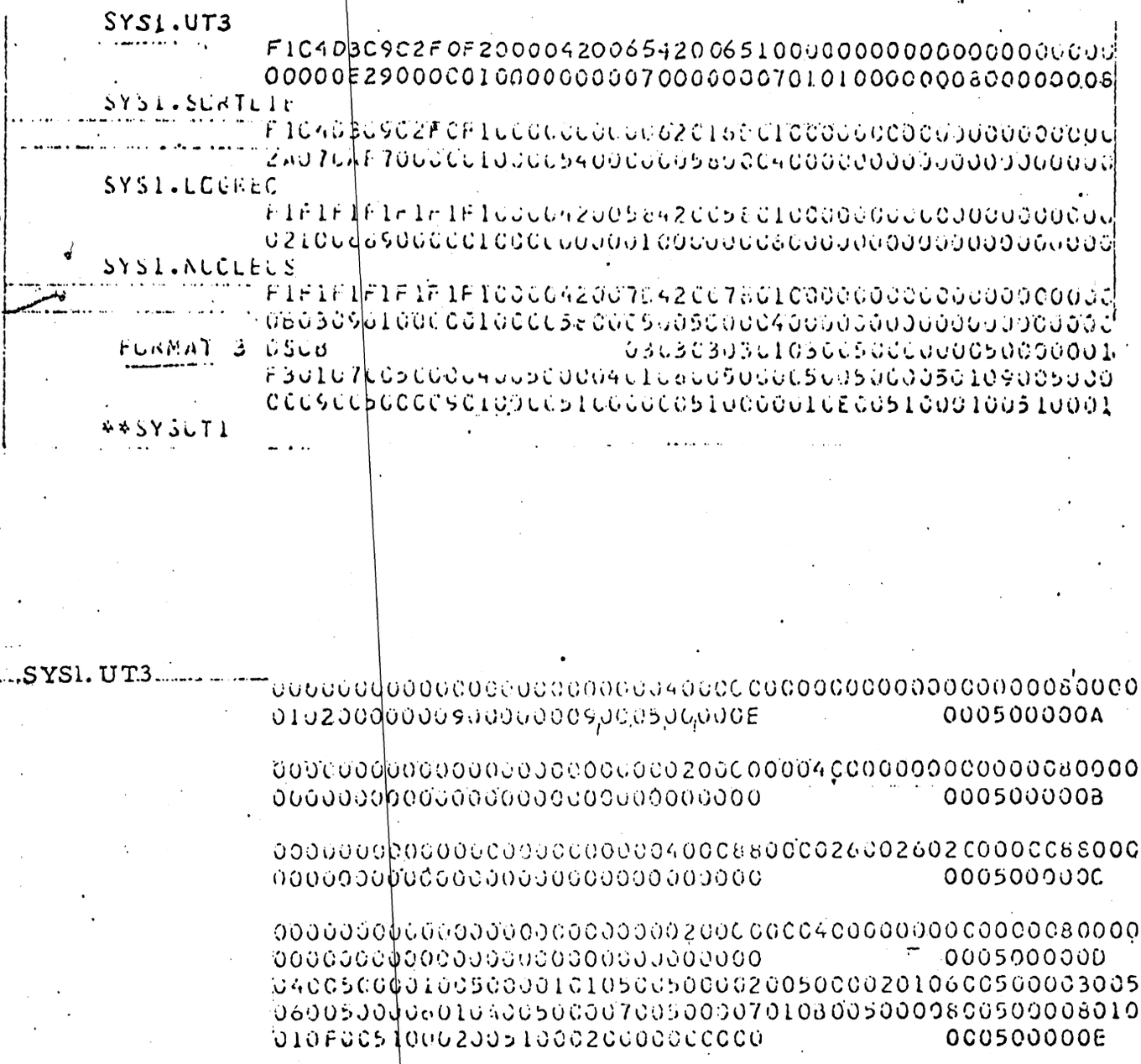

SYS1.UT3
SYS1.SCRTCHF
SYS1.LOGREC
SYS1.ACCLECS
FORMAT 3 DSCB
**SYSCT1
SYS1.UT3

Figure 39

B. Partitioned Data Set (PDS) Directory Information

The output load module produced by the linkage editor contains all the information necessary to load and relocate the module in main storage. When the load module is placed in the output module library, the name of the module and control information describing its characteristics are placed in the library partitioned data set directory. Pertinent information about the load module, such as the module attributes are used by the Control program fetch routine when the program is loaded for execution. They are also placed in the status field of the RB associated with the load module. It's important, therefore, to be able to locate this type of information in the PDS directory.

1. Directory Organizations

The PDS directory occupies the beginning of the extent allocated to the data set on a direct-access device. The directory consists of variable-length logical records arranged in ascending order according to the binary value of the member name or alias.

The directory records are blocked into 256 - byte blocks, each containing as many complete entries as will fit in a maximum of 254 bytes.

Each logical record in a directory block contains a name, TTR, and count field. It may also contain a user data field. The last logical record in the last active directory block has a name field of maximum binary value.

2. Directory Contents

The method of investigating a member of a PDS starts with the execution of the IEHLIST program with a LISTPDS control card included for the particular data set that contains the member in question.

The information listed as a result is shown in figure 40. The contents of the PDS directory entry, Format 1, can be used to answer the following questions:

a. What is the relative location of the member in the data set?

b. How much core storage is required?

c. Is the member name an alias?

d. Is the module in overlay structure?

e. What are the System Status Indicator Values?

Using the template, Figure 41, and the member IEABDLOO shown on Figure 40, the following values are found for the above questions:

a. The relative TTR is 001907 - Item B

b. 00 00 A0 -- Contiguous Core required - Item F

c. Bit 0 of the indicator byte is zero, therefore, name is not alias - Item B

d. Bit 2 of attribute byte one is zero, therefore, not overlay structure - Item E.

e. 00 05 31 36 -- SSI bytes -- Item J

3. Directory Size and Number of Entries

The PDS Directory entry provides ready reference to information about a member of PDS.

It may be necessary to expand or contract the size of a PDS. In order to do this you need to know how many directory blocks are needed and how many entries can be contained in one block.

The following method can be used to look at a PDS as it appears on the DISK.

A. Using the IEBPTPCH program, define the PDS as a sequential data set. This will cause the directory to be printed. Each directory block is 256 bytes long so it is possible by inspection to check the number of directory blocks that were specified. The number of entries per block can also be determined by inspection.

B. Using the IEBPTPCH program, define the PDS as a PDS and select any or all members of the data set for listing.

Refer to SRL C28-6586-1 page 38, for further information.

```
IEAAADC2  00140F2D  0015C10000  0000CC0370  0003E603E8  0000000000  0000010130  15
IEAAADC3  00150320  0015C50000  0000CC0370  0003F003F0  0000000000  0000010130  15
IEAAAIOC  00150720  0015C90000  0000C00370  0002260226  0000000000  0000010530  15
IEAAALOC  00150B20  0015CC1000  0000000260  000006C008  0000000000  0000000550  42
IEAAAPCC  00160420  0016C60000  0000000370  0000920092  0000000000  0000010130  15
IEAAILOC  00160F20  0016CA0000  0000000370  0001430143  0000000000  0000000547  74
IEAAMSOC  00161320  0016CE0000  0000000260  00023A023A  0000000000  0000000550  42
IEAAPLOC  00161720  0016D30000  0000000260  000000006C  0000000000  0000000551  09
IEAAPXOC  00161B20  0016D80000  0000000370  00003E003E  0000000000  0000000540  91
IEAASYOC  00161F20  0016DC0000  0000000370  00003A003A  0000000000  0000000501  91
IEAATHOC  00170320  0017E70000  0000CC0370  000030C030C  0000000000  0000000160  21
IEAATM01  00170720  0017E80000  0000000370  00030D003CD  0000000000  0000000131  80
IEAATM02  00170B20  0016C10000  0000CC0370  00030E003DE  0000000000  0000000131  81
IEAATM03  00180320  0016C50000  0000000370  0001A001A4  0000000000  0000000131  82
IEAATM04  00180720  0016C90000  0000CC0370  0003A103A1  0000000000  0000000160  07
IEAATM05  00180B20  0019C10000  0000000370  0002400240  0000000000  0000010130  15
IEAAXRCC  00190320  0019C50000  0000CC0370  0000540054  0000000000  0000000547  74
IEABDL00  0019072D  0019090000  0000000260  0000A000A0  0000000000  0000000531  36
IEACMSCC  00190B20  0019CE0000  0000CC0260  0002740274  0000000000  0000000550  42
IEACMSCL  00191120  0019D10000  0000000260  0002720272  0000000000  0000000550  42
IEACMSCC  00191520  0019D50000  0000CC0260  0002A002A0  0000000000  0000000521  31
IEACNTCC  00191920  0019D80000  0000000370  0000E000E  0000000000  0000000540  59
IEACNTIC  00191D20  0019DC0000  0000CC0370  0000660006  0000000000  0000000540  59
IEACSTCC  00192120  0019E10000  0000000260  0001D001D8  0000000000  0000000580  40
IEACTICC  00192520  0019E50000  0000000260  0002180218  0000000000  0000000580  40
IEACTIOI  00192920  0019E80000  0000CC0370  0000350035  0000000000  0000000540  59
IEBCARAL  00193D20  0019EF0000  0000000260  0000330353  0000000000  0000000580  47
IEBCCSCZ  00194120  0019F30000  0000000260  0003770400  0000000000  0000000540  26
IEBCHAIR  00210020  0020C10000  0000CC0260  0018020400  0000000000  0000010330  15
IEBCOMPP  00210420  0020C70000  0000000270  0004800400  0000000000  0000000580  47
IEBCUNIZ  00210820  0020D10000  0000000370  0001520152  0000000000  0000000590  54
IEBCUNPZ  00210C20  0020D50000  0000000370  0000760076  0000000000  0000000540  11
IEBCUNZZ  00211020  0020D90000  0000000370  0000740074  0000000000  0000000590  04
IEBCUPYA  00211420  0020DC0000  0000000260  0005000400  0000000000  0000000580  13
IEBCUPYC  00211820  0021E10000  0000CC0270  0004800400  0000000000  0000000500  96
IEBCUPYC  00211C20  0021E70000  0000CC0260  0004800400  0000000000  0000000500  96
IEBCUPYU  00212020  0021EC0000  0000000260  0006CC0400  0000000000  0000000510  58
IEBCRUDI  00212420  0021F10000  0000000260  0004B002A0  0000000000  0000000510  00
IEBEDIT2  00212820  0021F50000  0000000260  0001960196  0000000000  0000000530  61
IEBGENER  00212C20  0021F80000  0000000260  0006040400  0000000000  0000000500  38
IEBGENR3  00213020  0021FC0000  0000CC0260  0006F00400  0000000000  0000000500  35
IEBGRESC  00213420  0022010000  0000000260  0002260228  0000000000  0000000540  11
IEBGSCAN  00213820  0022050000  0000000260  0003810400  0000000000  0000000580  30
IEBLERP2  00213C20  0022090000  0000000260  [illegible]
```

Figure 40
-76-

PARTITIONED DATA SET (PDS) DIRECTORY ENTRY - FORMAT 1

Ⓐ MEMBER NAME

Ⓑ 3 TTR of the first block of the named member

1 Indicators

Bits	Bit Settings	Meaning
0	1	Name is an alias
1-2	(variable)	Number of TTR's in the user data field. A maximum of three is allowed.
3-7	(variable)	Length of the user data field in half words.

Ⓒ 3 TTR of the first block of text

1 Zeros

Ⓓ 3 TTR of the Note List or Scatter/Translation Table. Used for modules in scatter load format or overlay structure only.

1 The number of entries in the note list for modules in overlay structure; otherwise zero.

Ⓔ 3 Total contiguous main storage requirement of module

Ⓖ 2 Length of the first block of text

Ⓗ 3 Entry point address associated with member name or with alias name if the alias indicator is on

Ⓘ 3 Linkage editor assigned origin of the first block of text

Ⓙ 2 Attributes

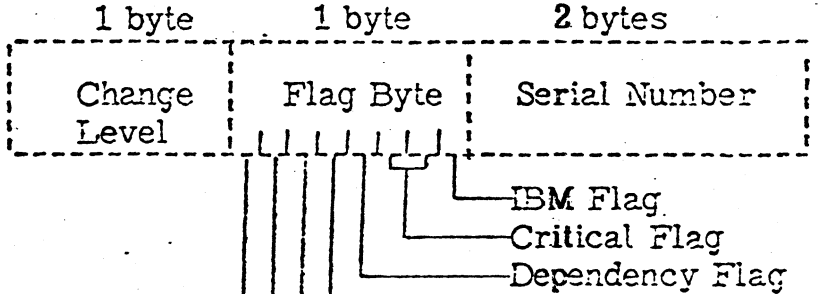
First Byte

Bits	Bit Settings	Meaning
0	1	Reenterable
1	1	Reusable
2	1	In overlay structure
3	1	Module to be tested - TESTRAN
4	1	Only loadable
5	1	Scatter format
6	1	Executable
7	1	Module contains no RLD items and only one block of text
7	0	Module contains multiple records with at least one block of text

Ⓚ Second Byte

Bits	Bit Settings	Meaning
0	1	Module can be processed onl by F level of linkage editor
0	0	Module can be processed by levels of linkage editor
1	1	Linkage editor assigned orig of first block of text is zero
1	0	Linkage editor assigned orig of first block of text is not z
2	1	Entry point assigned by linka editor is zero
3	1	Module contains no RLD iten
4	1	Module cannot be reprocesse by linkage editro
5	1	Module contains TESTRAN symbol cards
6-7		Reserved for future use

Ⓛ



Format of SSI Bytes

Figure 41

APPENDIX A

This appendix contains two lists: the first is a list of those macro-instructions whose expansion includes an SVC instruction and the SVC number (decimal) associated with that instruction; the second is a list of the routines thatperform the services requested via the SVCs and the program logic manuals (PLMs) in which these routines are described:

Macro-Instruction	SVC No.	Hexa-decimal	Macro-Instruction	SVC No.	Hexa-decimal
ABEND	13	0D	FEOV	31	1F
ATTACH	42	2A	FIND	18	12
BLDL	18	12	FREEBUF	57	39
BSP	69	45	FREEMAIN	05	05
CATALOG	26	1A	GETMAIN	04	04
CHAP	44	2C	IDENTIFY	41	29
CHKPT	50	32	INDEX	26	1A
GIRB	43	2B	LINK	06	06
CLOSE	20	14	IOHALT	33	21
CLOSE (TYPE=T)	23	17	LOAD	08	08
DELETE	09	09	LOCATE	26	1A
DEQ	48	30	OBTAIN	27	1B
DEVTYPE	24	18	OPEN	19	13
DETACH	62	3E	OPEN (TYPE=J)	22	16
ENQ	56	38	POST	02	02
EOV	55	37	PURGE	16	10
EXCP	00	00	RELEX	53	35
EXTRACT	40	28	RENAME	30	1E

Macro-Instruction	SVC No.	Hexa-decimal
RESTART	52	34
RESTORE	17	11
SCRATCH	29	1D
SEGLD	37	25
SEGWT	37	25
SPIE	14	0E
STAE	60	3C
STIMER	47	2F
STOW	21	15
SYNCH	12	0C
TIME	11	0B
TTIMER	46	2E
WAIT	01	01
WAITR	01	01
WTO	35	23
WTOR	35	23
WTL	36	24
XCTL	07	07

In the following list, 'ROUTINE NAME' indicates the name by which each SVC routine is referred to in the associated PLM. Two entries in the 'TYPE' field indicate that at system generation time, the user can choose either type for this SVC routine. The first number indicated is the dominant one and is the type assigned unless the second number is explicitly specified.

Use of an SVC number that has '**' in the 'ROUTINE NAME' field causes the SVC interruption handler to abnormally terminate the job step. All unassigned and some nonsupported SVCs fall into this category.

Use of the remaining nonsupported SVC numbers is effectively a no-operation instruction. An interruption will occur, but after the SVC interruption handler analyzes the SVC, it immediately passes CPU control to the SVC exit routine. Nonsupported or unassigned SVC numbers cannot be assigned to user-written SVC routines.

SVC NUMBER	ROUTINE NAME	TYPE	PLM
00	EXCP	1	Input/Output Supervisor
01	Wait	1	Fixed-Task Supervisor
02	Post	1	Fixed-Task Supervisor
03	Exit	1	Fixed-Task Supervisor
04	Getmain	1	Fixed-Task Supervisor
05	Freemain	1	Fixed-Task Supervisor
06	Link	2	Fixed-Task Supervisor
07	XCTL	2	Fixed-Task Supervisor
08	Load	2	Fixed-Task Supervisor
09	Delete	1	Fixed-Task Supervisor
10	Getmain/Freemain	1	Fixed-Task Supervisor
11	Time	1	Fixed-Task Supervisor
12	SYNCH	2	Fixed-Task Supervisor
13	ABEND	4	Fixed-Task Supervisor
14	SPIE	3,2	Fixed-Task Supervisor
15	ERREXCP	1	Input/Output Supervisor
16	Purge	3	Input/Output Supervisor
17	Restore	3	Input/Output Supervisor
18	BLDL	2	Sequential Access Methods
19	Open	4	Input/Output Support (OPEN/CLOSE/EOV)
20	Close	4	Input/Output Support (OPEN/CLOSE/EOV)
21	Stow	3	Sequential Access Methods
22	OpenJ	4	Input/Output Support (OPEN/CLOSE/EOV)
23	Tclose	4	Input/Output Support (OPEN/CLOSE/EOV)
24	DEVTYPE	3	Input/Output Supervisor
25	Track Balance	3	Sequential Access Methods
26	Catalog	4	Catalog Management
27	Obtain	3	Direct Access Device Space Management
28	CVOL	4	Catalog Management
29	Scratch	4	Direct Access Device Space Management
30	Rename	4	Direct Access Device Space Management
31	EOV	4	Input/Output Support (OPEN/CLOSE/EOV)
32	Allocate	4	Direct Access Device Space Management
33	IOHALT	3	Input/Output Supervisor
34	Master Command EXCP	4	Job Management
35	Write to Operator	3	Job Management
36			Not supported in this configuration
37	Overlay Supervisor	2	Fixed-Task Supervisor
38	Resident SVC	2	TESTRAN

SVC NUMBER	ROUTINE NAME	TYPE	PLM
39	**		Unassigned
40	Extract	3,2	Fixed-Task Supervisor
41	Identify	3,2	Fixed-Task Supervisor
42	Attach	3,2	Fixed-Task Supervisor
43	CIRB	3	Fixed-Task Supervisor
44			Not supported in this configuration
45	Overlay Supervisor	2	Fixed-Task Supervisor
46	Ttimer	1	Fixed-Task Supervisor
47	Stimer	2	Fixed-Task Supervisor
48			Not supported in this configuration
49	Ttopen1	3	TESTRAN
50			Not supported in this configuration
51	ABDUMP	4	Fixed-Task Supervisor
52			Not supported in this configuration
53	**		Not supported in this configuration
54	**		Not supported in this configuration
55	EOV	4	Input/Output Support (OPEN/CLOSE/EOV) Sequential Access Methods
56			Not supported in this configuration
57	**		Not supported in this configuration
58	**		Unassigned
59	**		Not supported in this configuration
60			Not supported in this configuration
61	Save	3	TESTRAN
62			Not supported in this configuration
63	**		Unassigned
64	RDJFCB	3	Input/Output Support (OPEN/CLOSE/EOV)
65	**		Not supported in this configuration
66	**		Not supported in this configuration
67	**		Not supported in this configuration
68	**		Not supported in this configuration
69	Backspace	3	Sequential Access Methods
70	GSERV	2	Graphics Access Method
71	**		Not supported in this configuration
72-199	**		Unassigned
200-255	Available for assignment to user-written SVC routines. Until a number is assigned, its use in a processing program causes termination.		

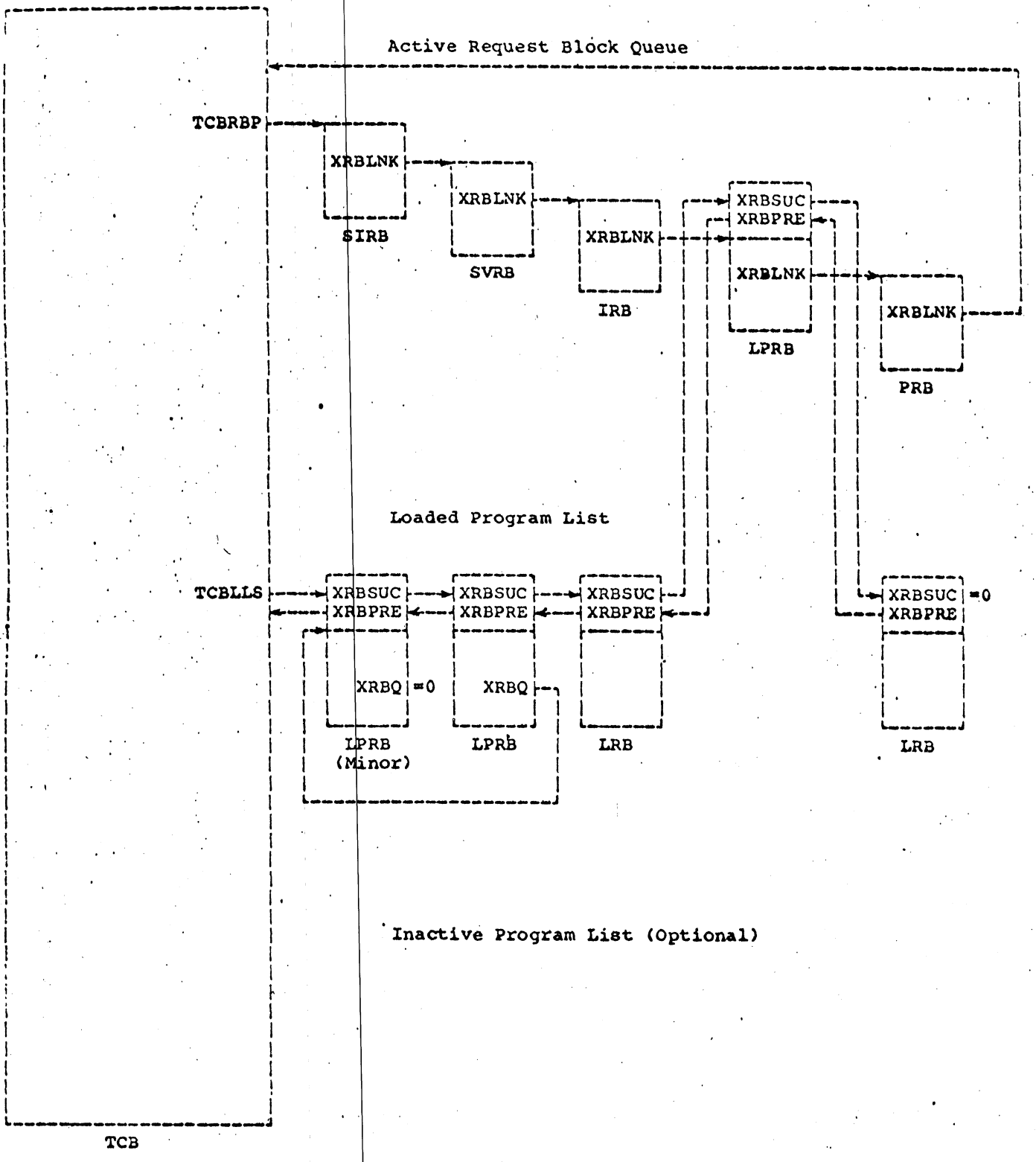

Active Request Block Queue
TCBRBP
XRBLNK
SIRB
XRBLNK
SVRB
XRBLNK
IRB
XRBSUC
XRBPRE
XRBLNK
LPRB
XRBLNK
PRB
Loaded Program List
TCBLLS
XRBSUC
XRBPRE
XRBQ =0
LPRB
(Minor)
XRBSUC
XRBPRE
XRBQ
LPRB
XRBSUC
XRBPRE
LRB
XRBSUC =0
XRBPRE
LRB
Inactive Program List (Optional)
TCB

Figure 1.

This is a breakdown of the status bytes (STAB) in the Request Blocks (RB) for the sequential system.

bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
STAB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

0, 1 — Used to distinguish between PRB, IRB, SIRB, and SVRB (see chart 1)

2 — determines if program is LOADed or not

3 — determines if program is in transient area or not

4 — must be zero

5 — must be zero

6 — must be zero

7 — must be zero

8 — primary queueing field addresses TCB

9 — active program

10 — priviledged program, 16 registers saved in RB

11 — rescheduable program (re-enterable or reusable)

12, 13 — RB for IOS / IQE's appended are 12*'s

12	13	
0	0	- no IQE's
0	1	- 12 byte IQE's
1	1	- 16 byte IQE's

14 — RB exists disjoint from program (RB freed by EXIT routine)

15 — off - WAIT on single event or all events
on - WAIT on fewer than the number of events specified.

FIGURE 2

The combination of the first four bits in the first flag byte have the specified definition as shown in chart 1

BITS			
01	2	3	Description
00	0	0	PRB, not LOADed, does not have minor entries. (IDENTIFYed)
00	0	1	PRB, not LOADed, does have minor entries.
00	1	0	PRB, LOADed, no minor entries.
00	1	1	PRB, LOADed, minor entries
01	0	0	IRB
01	0	1	
01	1	0	
01	1	1	
10	0	0	SIRB
10	0	1	
10	1	0	
10	1	1	
11	0	0	SVRB, Type II
11	0	1	SVRB, Type III
11	1	0	PRB, LOADed, Minor entry
11	1	1	LRB

FIGURE 3

OPERATING SYSTEM/360 NAMING CONVENTIONS

IEA	SUPERVISOR
IEB	DATA SET UTILITIES
IEC	INPUT/OUTPUT
IEE	MASTER SCHEDULER
IEF	BATCH SCHEDULER
IEG	PROGRAM TEST
IEH	SYSTEM AND SUPPORT UTILITIES
IEI	SYSTEM GENERATOR
IEJ	FORTRAN COMPILER (E)
IEK	FORTRAN COMPILER (H)
IEM	PL/1 COMPILER (F)
IEN	PL/1 COMPILER (H)
IEP	COBOL COMPILER (E)
IEQ	COBOL COMPILER (F)
IER	SORT/MERGE
IES	REPORT PROGRAM GENERATOR
IET	ASSEMBLER (E)
IEU	ASSEMBLER (F)
IEW	LINKAGE EDITOR
IFB	SYSTEM ENVIRONMENT RECORDING AND RETRY, SER0, SER1
IFC	ENVIRONMENT RECORDING EDIT AND PRINT
IFF	GRAPHIC PROGRAMMING SUPPORT
IGC	TRANSIENT SVC ROUTINES
IGE	I/O ERROR ROUTINES
IGG	CLOSE, OPEN AND RELATED ROUTINES
IHB	SYSTEM MACRO-INSTRUCTION DEFINITIONS
IHC	LIBRARY SUBROUTINES (FORTRAN)
IHD	LIBRARY SUBROUTINES (COBOL)
IHE	LIBRARY SUBROUTINES (PL/1)

Completion Codes

Excerpts from Form C28-6608

001 BSAM CHECK with no SYNAD routine or QSAM DCBEROPT specified termination.
002 BSAM Write or QSAM Put record exceeds track length.
008 SYNAD routine error following a BSAM CHECK.
020 BDAM OPEN with an invalid DCBMACRF.
025 BDAM processing error DCBSQND address outside task boundaries.
030 BISAM or QISAM OPEN with an invalid DCBMACRF.
031 QISAM processing error and DCB did not contain a SYNAD routine.
032 BISAM or QISAM OPEN with an invalid DCBMACRF field.
033 BISAM highest level index I/O error.
034 BISAM OPEN main storage area too small to contain the highest level index.
035 BISAM OPEN with the DCBMSWA and DCBSMSW specifying too small an area to contain one track of prime data.
036 BISAM or QISAM OPEN with no space allocated as the prime area in the DD Card or DSCB was modified erroneously.
037 BISAM or QISAM OPEN with an invalid buffer specification.
038 QISAM OPEN for load mode with insufficient space allocated or (2) the high level indexes crossed volumes the DD Card SPACE parameter.
039 QISAM scan reached the end with no DCBEODAD routine.
03A BISAM or QISAM CLOSE I/O error in updating the Format 2 DSCB.
0B0 Job Scheduler I/O error in SYS1.SYSJOBQE.
0Cx A program check occurred without a recovery routine. X specifies:

Code	Program Interruption Cause	Code	Program Interruption Cause
1	Operation	8	Fixed-point overflow
2	Privileged operation	9	Fixed-point divide
3	Execute	A	Decimal overflow
4	Protection	B	Decimal divide
5	Addressing	C	Exponent overflow
6	Specification	D	Exponent underflow
7	Data	E	Significance
		F	Floating-point divide

0F1 A program check occurred in the I/O Supervisor.
0F2 A program check occurred in the execution of a Type 1 SVC routine.
100 Device not operational.
101 WAIT error from more events than available ECB's.
102 POST error from an invalid ECB address.
106 LINK, LOAD, ATTACH or XCTL error indicated in register 15:
OD Invalid record type found when loading the program.
OE Invalid address found when loading the program.
OF I/O error occurred when loading the program.

13	OPEN I/O error in reading the JFCB or (2) failure to locate the JFCB pointer in TYPE=JOPEN.
117	BSAM CLOSE I/O error on tape.
122	CANCEL, DUMP requested by the operator.
126	TESTRAN error from a modified TESTRAN CSECT.
12D	Overlay Supervisor found incorrect words 3 and 4 of the segment table.
131	TESTRAN error from a modified TESTRAN CSECT.
137	E of V I/O error in tape label processing.
200	I/O error occurred with all I/O request elements in use.
201	WAIT error from an invalid ECB address.
202	POST error from an invalid RB address in the ECB end.
207	A SYNAD routine attempted to execute an XCTL macro instead of RETURN.
213	OPEN failed to find the DSCB or had I/O error. Verify the DISP parameter of the DD card.
214	CLOSE I/O error in tape positioning or volume disposition.
217	BSAM CLOSE I/O error in reading the JFCB.
222	CANCEL requested by the operator. No dump requested.
22D	Overlay Supervisor found an invalid address in the segment table.
237	E of V verification error in label processing. Correct the volume serial number in the DD card and re-execute.
301	WAIT specified an ECB whose wait bit was on.
308	LOAD error when Option 3 but not 4 is included.
313	OPEN I/O error in reading a Format 3 DSCB.
14	CLOSE I/O error in reading the DSCB.
317	BSAM CLOSE I/O error in reading a DSCB.
326	TESTRAN instructions exceeded the limit specified.
331	TESTRAN needs an address for the TEST OPEN instruction.
337	No DCBEODAD routine available for the end of a data set.
400	An invalid DCB, DEB, IOB or an improper DD Card detected.
406	LINK, ATTACH, or XCTL error from "only loadable" program or an IDENTIFY entry point program specified without Option 4 in the PCP.
413	OPEN failed in reading the volume label, the volume could not be mounted on the allocated device, or no volume serial number was specified in the SER parameter of the DD Card.
414	CLOSE I/O error in updating the DSCB.
417	BSAM CLOSE I/O error in writing the updated DSCB.
425	SEGWT error during execution of an overlay program.
426	TESTRAN output limit exceeded.
431	TESTRAN symbol table and control dictionaries could not be read.
437	End of Volume error when the DEBDEBID protection key did not match the TCBPKF
506	LINK, LOAD, ATTACH or XCTL used in an overlay program under TESTRAN.
513	OPEN for a data set on a magnetic tape used for another data set.
514	CLOSE I/O error in reading the JFCB.
517	BSAM CLOSE error because the PCP used does not support user labels.
526	TESTRAN used without a TEST OPEN instruction.
531	TESTRAN used without a DD statement for the unedited data.
804	GETMAIN error from an erroneous address or length in an inactive program, (2) and erroneous address in the macro, or (3) a free area exceeded the bounds of the task.

705	FREEMAIN error from a free area exceeding the bounds of the task.
706	LINK, LOAD, ATTACH, or XCTL error from lack of available storage.
60A	GETMAIN or FREEMAIN error from an erroneous address in an inactive program.
613	OPEN I/O error in tape positioning or label processing.
614	CLOSE I/O error in writing the end-of-file.
626	TESTRAN encountered a machine-check while tracing the program.
637	End of Volume I/O error in writing the tape mark, positioning the tape, or reading the label.
700	Unit Check Status Indication set on.
705	FREEMAIN issued with L operand and PCP Option 4 not included.
706	LINK, LOAD, ATTACH, or XCTL requested an unexecutable program.
713	OPEN for an unexpired data set created with an EXPDT or RETPD parameter in the DD Card.
714	CLOSE I/O error in tape label processing.
717	BSAM CLOSE I/O error in tape label processing.
800	Program or Protection Interruption during an I/O operation.
804	GETMAIN with EU or VU mode operand not supported by the PCP in use.
914	CLOSE functions not supported by the PCP in use.
926	Machine Check when TESTRAN attempted to return control to the user program.
937	End of Volume functions not supported by the PCP in use.
A04	GETMAIN error from an erroneous address or length in an inactive program.
A05	FREEMAIN error when an address and length specification in the release request defined an area overlapping a free area.
)A	GETMAIN or FREEMAIN found the address and length specification in an inactive program or released program defined an overlapped free area.
A13	OPEN failed to find the DD Card specified file sequence number on tape.
A14	CLOSE I/O error in the release of unused storage on the DASD as specified in the DD Card.
A26	TESTRAN could not return to the problem address specified.
B04	GETMAIN specified a subpool number greater than 127.
B05	FREEMAIN specified a subpool number greater than 127.
B0A	GETMAIN or FREEMAIN specified a subpool number greater than 127.
B14	CLOSE error when STOW was issued. STOW was unable to store, modify, or delete data. The error is indicated in register 15: 04 Name already exists in the directory. 0C No space is left in the directory. 10 A I/O error during the search.
B37	EOV occurred with no space available for required functions and no volume available for dismounting.
C13	OPEN I/O error or (2) a concatenated data set could not be found.
D2D	Overlay Supervisor detected an invalid record type when loading.
D37	Output area filled and no secondary quantity SPACE parameter supplied for a DASD.
E2D	Overlay Supervisor detected an invalid address when loading.
E37	Output area filled or (2) 16 Extents already used in a PDS.
F13	OPEN failed to find a member name specified in a DD Card or (2) there are conflicting or unsupported parameters in the DCB.
F2D	Overlay Supervisor detected an incorrect length or an I/O error when loading.
F3F	QSAM FEOV I/O error in writing the remaining output buffers.
F37	EOV or secondary space allocation I/O error.
Fnn	Invalid SVC operand. nn = the SVC number.